ORACLE

MySQL HeatWave ML

A Deep Dive into AutoML Capabilities

MySQL Belgian Days 2024

Matteo Casserini

Consulting Member of Technical Staff, Oracle

Matteo Casserini

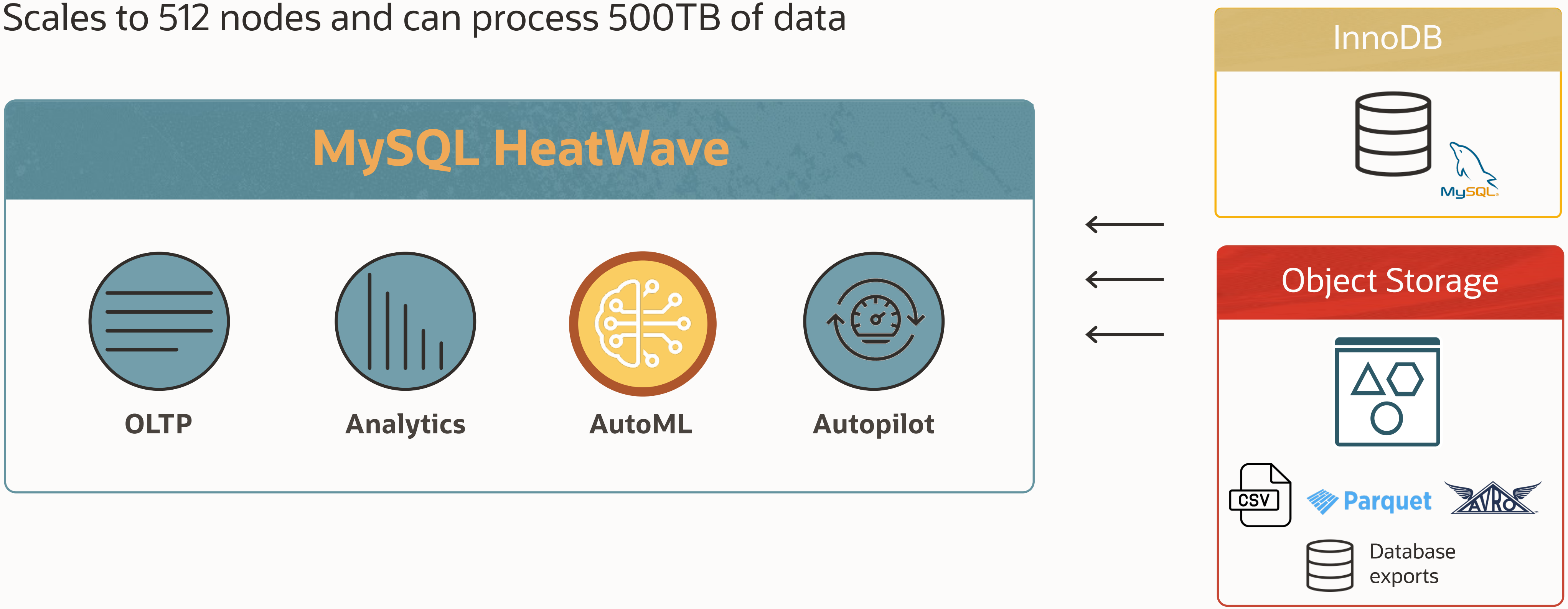
- Working on ML and AI since 2011
- Joined Oracle in August 2018
  - August 2018 – October 2023: Oracle Labs
  - Since November 2023: MySQL HeatWave

in matteocasserini

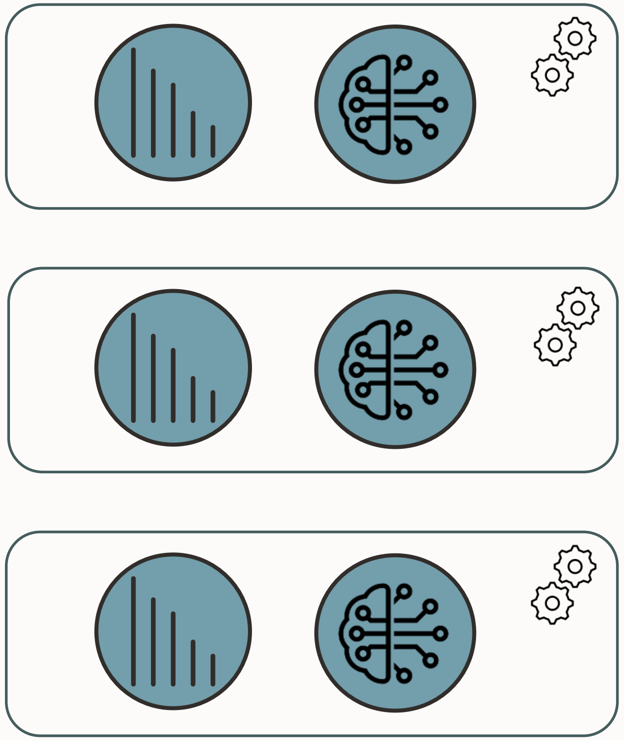
MySQL HeatWave Lakehouse

Process data inside database as well as object storage

- Efficient support for analytics, machine learning, OLTP
- Helps with MySQL and non-MySQL workloads
- Scales to 512 nodes and can process 500TB of data



Leverage HeatWave Cluster for ML Workloads

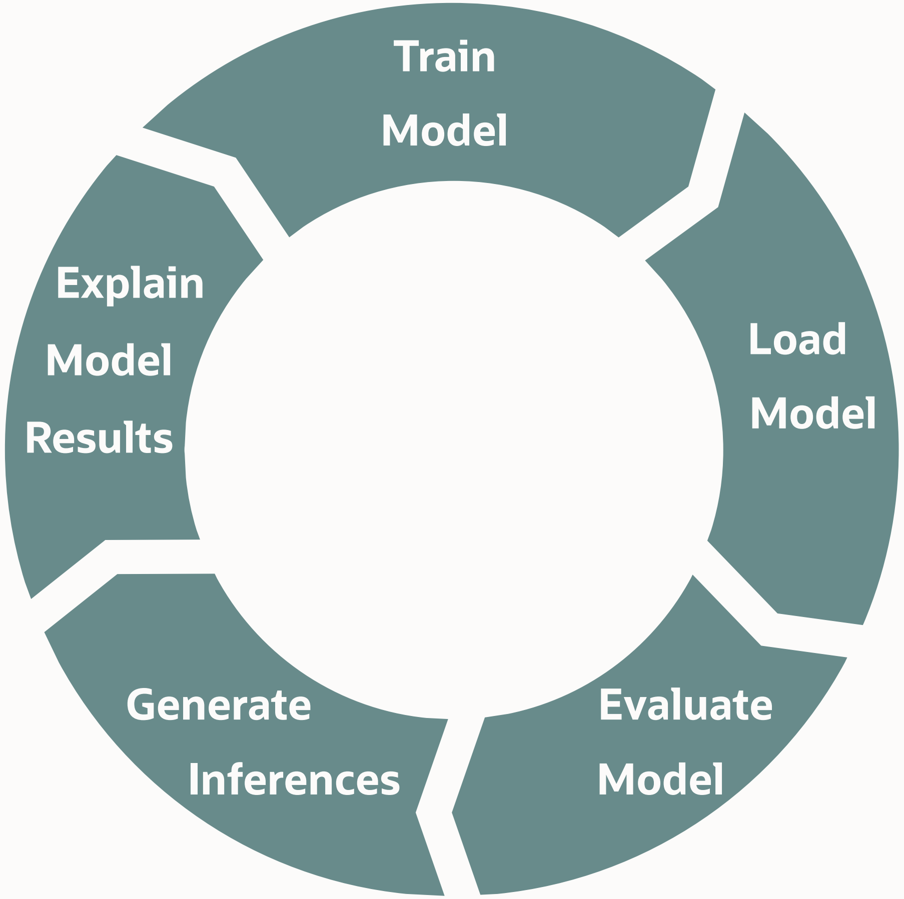


HeatWave cluster

- Part of the resources of the HeatWave cluster available for AutoML workloads
- Tables must be loaded to HeatWave memory before any ML operations
- OLAP takes priority

HeatWave AutoML

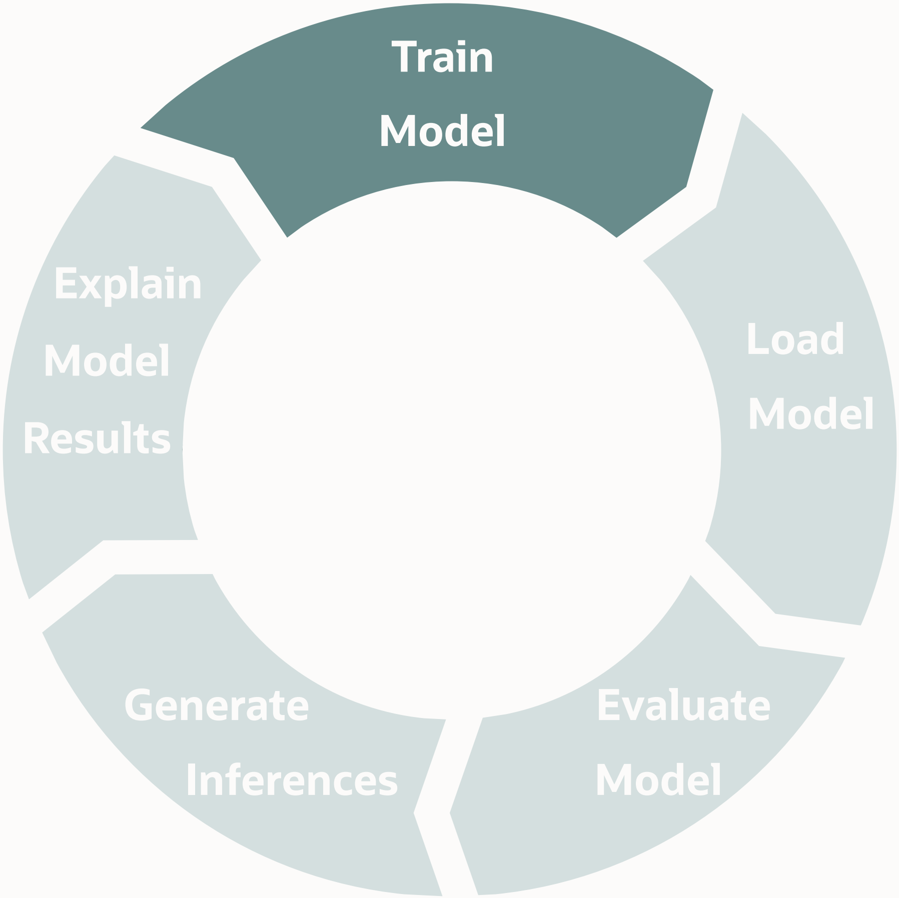
In-database Training, Inference and Explanation



In-database Management of the Entire Model Lifecycle!

- Fully automated ML with minimal number of required parameters → no advanced ML/data science expertise needed
- Data and ML model never leave the database
- Familiar SQL Interface
- Performance and Scalability

HeatWave AutoML



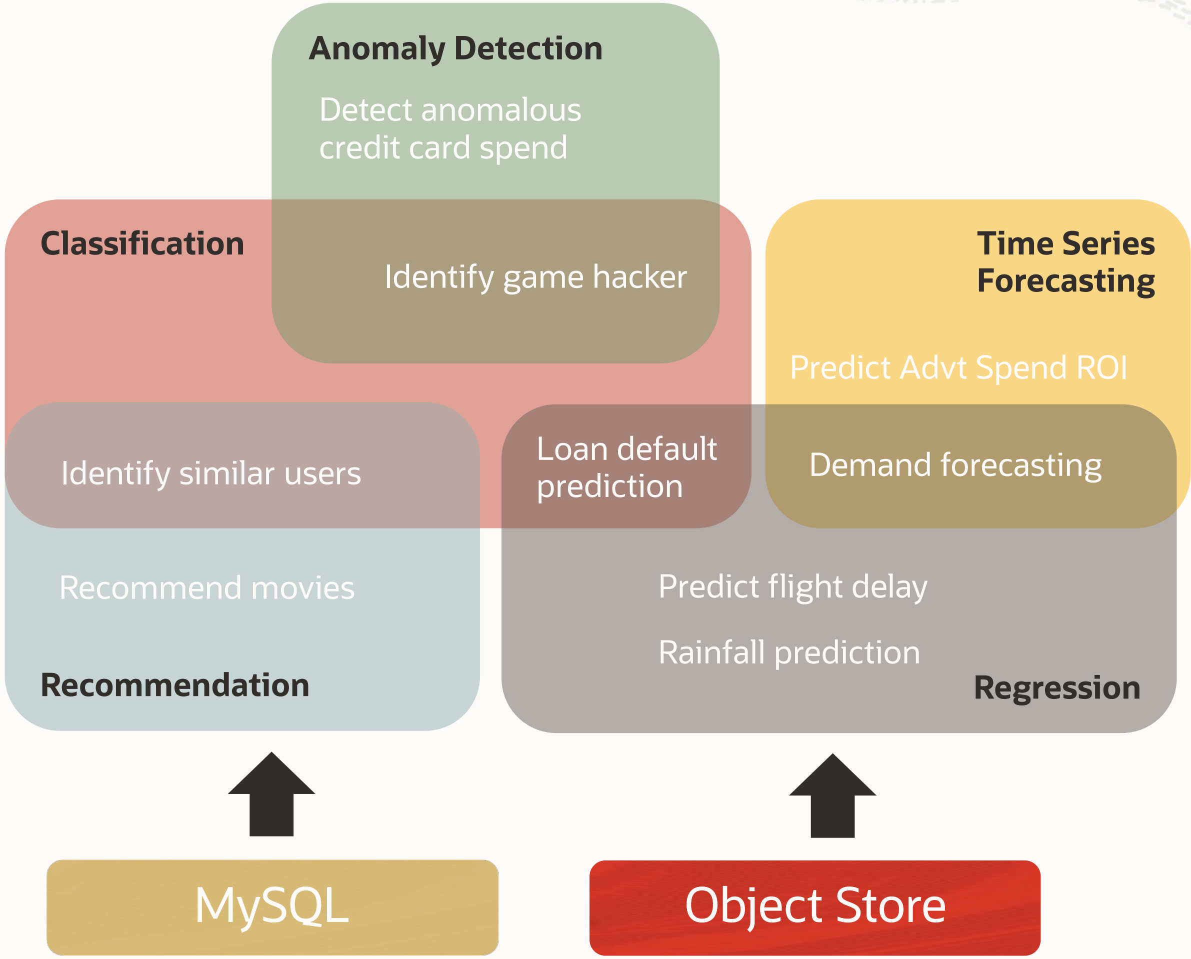
Model Training

Model training in machine learning usually requires

- Extensive computational resources
- Considerable ML expertise

HeatWave ML greatly enhances and simplifies this step

- High automation of model training for different tasks as classification, regression, anomaly detection…
- User only needs to prepare training data containing key data attributes for the relevant task
- Automatically perform all steps necessary for training depending on the task: preprocess data, feature selection, hyperparameter tuning and model selection
- Already creates the corresponding explanation model
- High-performing architecture that scales with the cluster
  - 25x faster than Redshift
  - Faster training → more frequent re-training → better quality



Model Training API

```
mysql> CALL sys.ML_TRAIN ('table_name', 'target_column_name', [options], model_handle);
```

'table_name': fully qualified name of the table containing the training dataset.

'target_column_name': name of the column in *'table_name'* representing the target, i.e. ground truth values (required for some tasks).

[*options*]: *optional* training parameters as key-value pairs in JSON format.

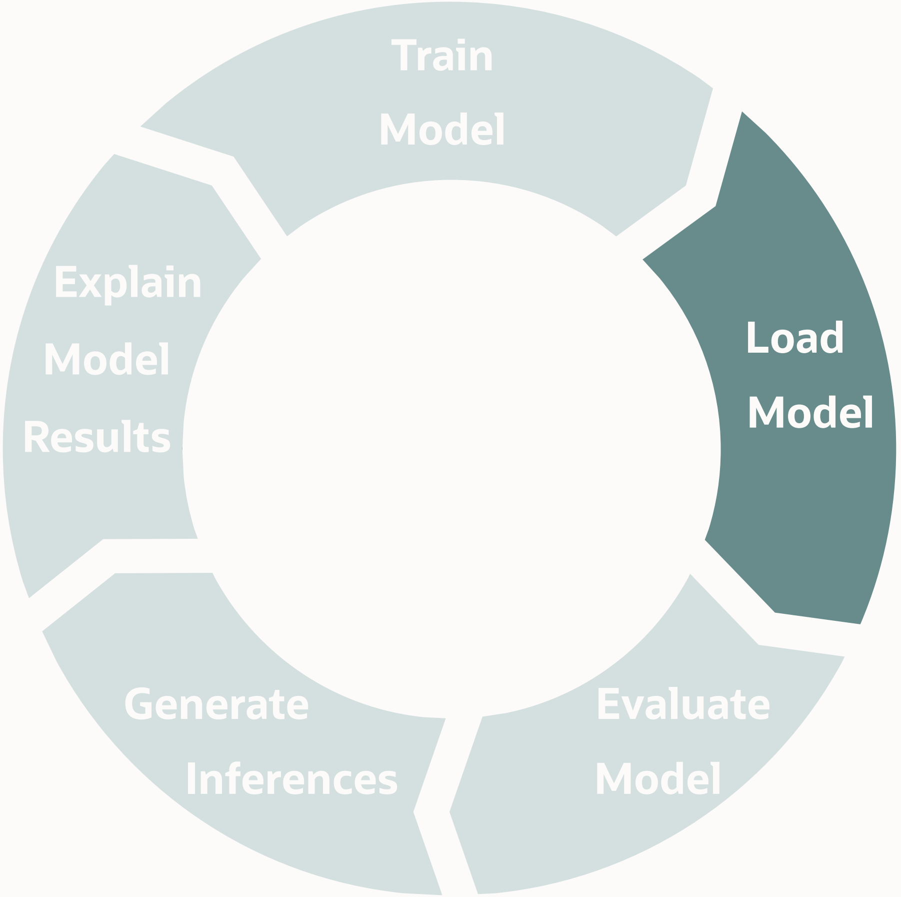
- The *most important* parameter is 'task', which specifies the ML task to be performed (if not specified, 'classification' is assumed);
- Other parameters allow finer-grained control on the training task.

model_handle: user-defined session variable storing the ML model handle for the duration of the connection.

Examples:

```
mysql> CALL sys.ML_TRAIN('heatwaveml_bench.census_train', 'revenue', JSON_OBJECT('task', 'classification'), @census_model);
```

HeatWave AutoML



Model Management

The ML model generated by `ML_TRAIN` routine is stored in the **Model Catalog**

- Table `MODEL_CATALOG` within the user ML schema (`ML_SCHEMA_<username>`) created by `ML_TRAIN`
- Each row contains ML model + metadata
- ML models become 1st class citizens
  - Can be integrated in standard DB procedures: backup, restore, encryption…
  - Sharing models between users follows usual access control management

Model handles created by `ML_TRAIN` are also stored in the model catalog, so that they can be conveniently be re-used (or reassigned to session variables) when the connection is terminated, e.g.

```
mysql> SET @my_model = (SELECT model_handle FROM ML_SCHEMA_user1.MODEL_CATALOG ORDER BY model_id DESC LIMIT 1);
```

Model Load/Unload API

Used to load the model in memory (required before model can be used, even immediately after `ML_TRAIN`!)

```
mysql> CALL sys.ML_MODEL_LOAD(model_handle, user);
```

model_handle: explicit model handle string or session variable containing the model handle.
user: MySQL user name of the model owner (if NULL, defaults to current user).

Examples:

```
mysql> CALL sys.ML_MODEL_LOAD('ml_data.iris_train_user1_1636729526', NULL);
mysql> CALL sys.ML_MODEL_LOAD(@iris_model, NULL);
```

Model Load/Unload API

Used to remove models from memory to free it up when they are not needed anymore

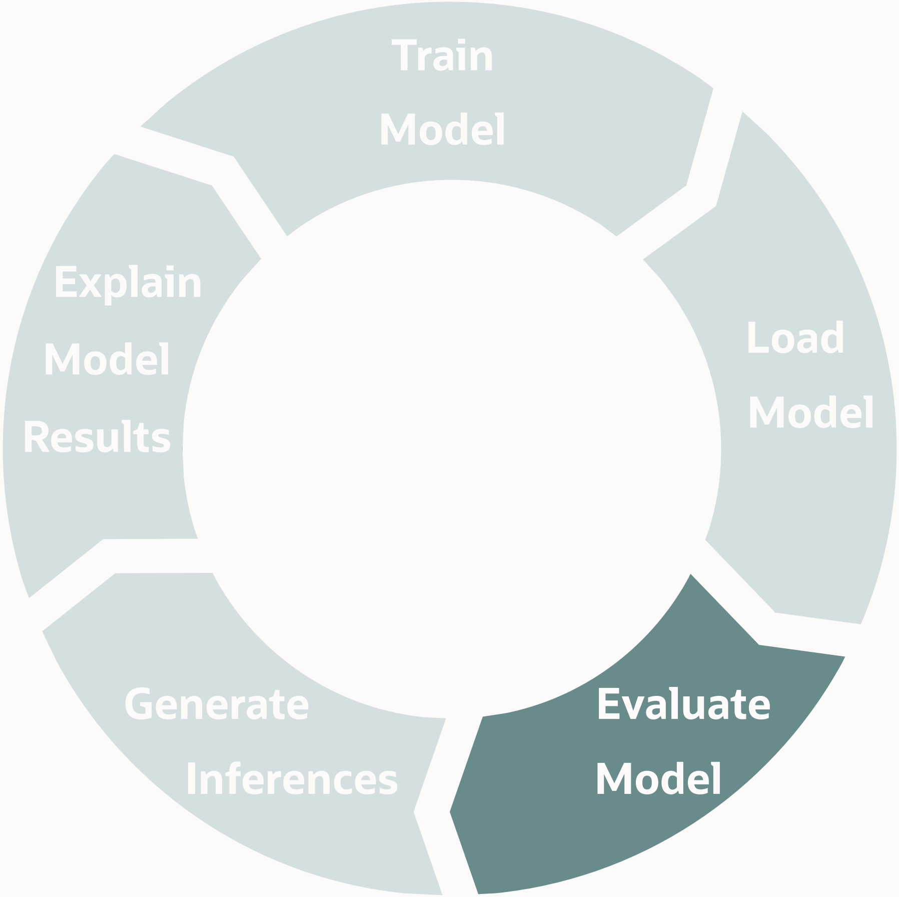
```
mysql> CALL sys.ML_MODEL_UNLOAD(model_handle);
```

model_handle: explicit model handle string or session variable containing the model handle.

Examples:

```
mysql> CALL sys.ML_MODEL_UNLOAD('ml_data.iris_train_user1_1636729526');
mysql> CALL sys.ML_MODEL_UNLOAD(@iris_model);
```

HeatWave AutoML



Model Evaluation

Before using models in production, best practice is to **evaluate** and **test the model**!

- Necessary to ensure *model quality*, in particular by evaluating its *generalization performance* (how well does it perform on data unseen during training?)
- Requires a dataset with the *same columns* as the dataset used for training, but *different data points*
  - Usual approach: *randomly split* the data available in a subset that will be used for training, and another that will be used for testing
- Requires to choose an appropriate *score* metric
  - No unique answer on the most appropriate metric, depends on the business need (e.g. prioritize low false positive rate? Or rather detect as many true positive as possible?)
  - Common choices: accuracy, f1-score

Model Score API

```
mysql> CALL sys.ML_SCORE(table_name, target_column_name, model_handle, metric, score, [options]);
```

'*table_name*': fully qualified name of the table containing the dataset used to compute model quality.

'*target_column_name*': name of the target column in '*table_name*' containing ground truth values.

model_handle: explicit model handle string or session variable containing the model handle.

metric: specifies which metric should be used to evaluate model quality. Different values can be used depending on ML task and target variable (e.g. `f1`, `precision`, `recall`, `roc_auc`, `f1_weighted`, `balanced_accuracy`…).

score: user-defined session variable name storing the computed score for the duration of the connection.

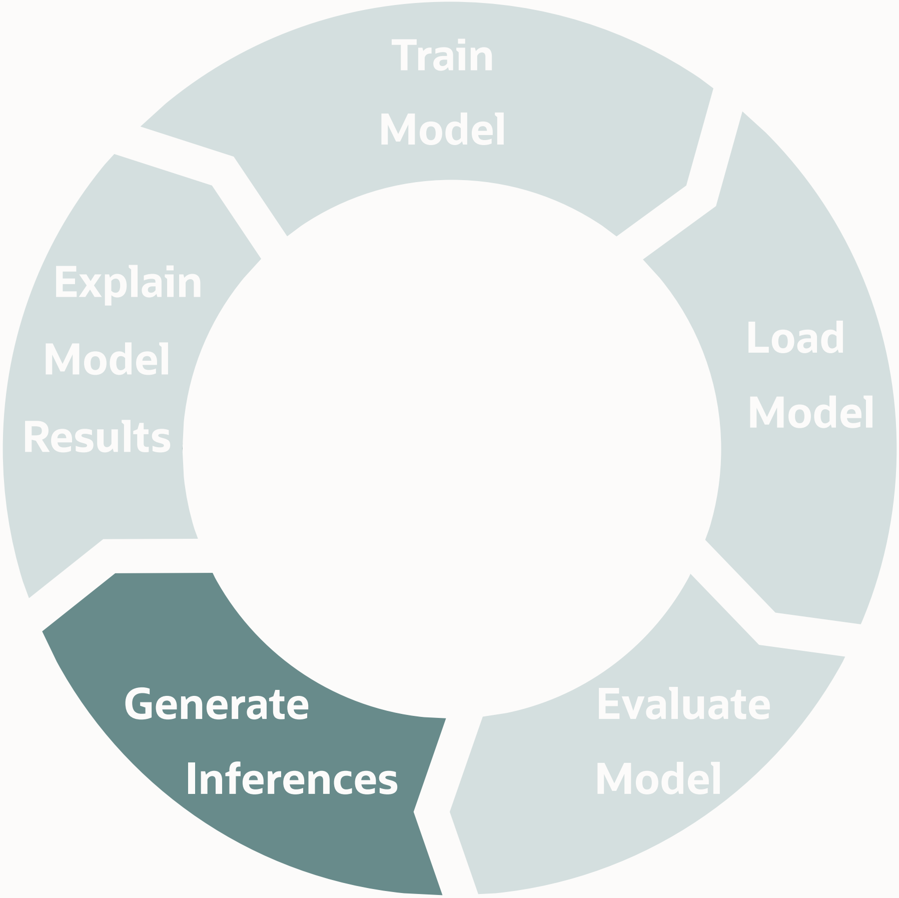
[*options*]: a set of optional key-value pairs, can be specified only starting in MySQL 8.0.32 and only for some tasks.

Examples:

```
mysql> CALL sys.ML_SCORE('ml_data.iris_validate', 'class', @iris_model, 'balanced_accuracy', @score, NULL);

mysql> SELECT @score;
+--------------------+
| @score             |
+--------------------+
| 0.958333313  |
+--------------------+
```

HeatWave AutoML



Model Inference

Apply trained model on new data points to generate *inference* results

HeatWave ML offers significant advantages:

- Inference performed in-database (where the data for inference resides)
- Inference scales with cluster size

2 different needs depending on application:

- Generate inference results for a single observation at a time
- Generate inference results for an entire table of observations

Model Inference API

Stored function to generate in-line inference for one or more rows of data specified in JSON format

```
mysql> SELECT sys.ML_PREDICT_ROW(input_data, model_handle);
```

input_data: specifies data for which inference results should be generated. Must contain all columns used during `ML_TRAIN`

- If a single row: specify the row data in JSON format
- If multiple rows: specify the columns where data resides as key-value pairs in JSON format, and select from a table

model_handle: explicit model handle string or session variable containing the model handle.

Examples:

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("sepal length", 7.3, "sepal width", 2.9, "petal length", 6.3, "petal width", 1.8), @iris_model);
```

```
mysql> SELECT sys.ML_PREDICT_ROW(JSON_OBJECT("sepal length", iris_test.`sepal length`, "sepal width", iris_test.`sepal width`, "petal length",
iris_test.`petal length`, "petal width", iris_test.`petal width`), @iris_model, NULL) FROM ml_data.iris_test LIMIT 5;
```

Model Inference API

Stored procedure to generate inference for an entire table, saving the inference results in another table

```
mysql> CALL sys.ML_PREDICT_TABLE('table_name', model_handle, 'output_table_name'), [options]);
```

'*table_name*': fully qualified name of the table containing the input dataset.

model_handle: explicit model handle string or session variable containing the model handle.

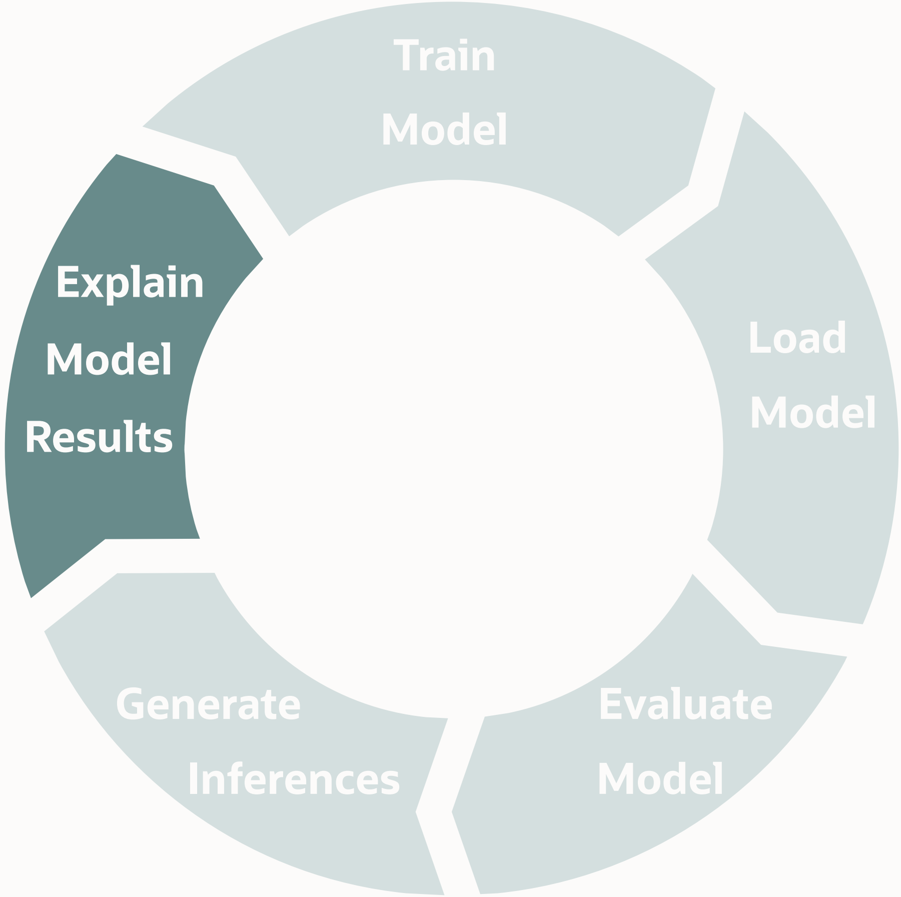
'*output_table_name*': fully qualified name of the table where to store inference results. An error is thrown if the table already exists.

[*options*]: a set of optional key-value pairs, can be specified only starting in MySQL 8.0.32 and only for some tasks.

Examples:

```
mysql> CALL sys.ML_PREDICT_TABLE('ml_data.iris_test', @iris_model, 'ml_data.iris_predictions');
```

HeatWave AutoML



Model Explanations

Explanations help understand which features have the biggest impact on a model's decisions

2 types of model explainers in HeatWave ML

- *Prediction explainers*: generate explanations for specific inference results
    - Allow to understand what features contributed the most to a model's inference result for each specific data point
- *Model explainers*: identify features that had *globally* the most impact on a model (based on the training set)
    - Allow to better understand the model characteristics

Explanations are generated as *feature importances*, ranging from -1 to 1:

- Magnitude indicates the *strength* of the feature impact;
- Sign indicates whether it contributes *towards* the prediction, or *away* from it.

Model Explain API

Stored function to generate in-line explanations for one or more rows of data specified in JSON format

```
mysql> SELECT sys.ML_EXPLAIN_ROW(input_data, model_handle, [options]);
```

input_data: specifies data for which inference results should be generated. Must match exactly columns used during `ML_TRAIN`

- If a single row: specify the row data in JSON format
- If multiple rows: specify the columns where data resides as key-value pairs in JSON format, and select from a table

model_handle: explicit model handle string or session variable containing the model handle.

[*options*]: a set of optional key-value pairs, currently only supports `prediction_explainer` (model used to generate explanations).

Examples:

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT("sepal length", 7.3, "sepal width", 2.9, "petal length", 6.3, "petal width", 1.8), @iris_model,
JSON_OBJECT('prediction_explainer', 'permutation_importance'));
```

```
mysql> SELECT sys.ML_EXPLAIN_ROW(JSON_OBJECT('sepal length',`iris_test`.`sepal length`, 'sepal width',`iris_test`.`sepal width`,'petal
length',`iris_test`.`petal length`, 'petal width',`iris_test`.`petal width` ), @iris_model, JSON_OBJECT('prediction_explainer', 'shap')) FROM
`iris_test` LIMIT 4;
```

Model Explain API

Stored procedure to generate explanations for an entire table, saving the explanation results in another table

```
mysql> CALL sys.ML_EXPLAIN_TABLE('table_name', model_handle, 'output_table_name'), [options]);
```

'*table_name*': fully qualified name of the table containing the input dataset.

model_handle: explicit model handle string or session variable containing the model handle.

'*output_table_name*': fully qualified name of the table where to store explanation results. An error is thrown if the table already exists.

[*options*]: a set of optional key-value pairs, supports `prediction_explainer` and `batch_size`.

Examples:

```
mysql> CALL sys.ML_EXPLAIN_TABLE('ml_data.iris_test', @iris_model, 'ml_data.iris_predictions');
```

Thank you!

Q&A