What Is BigQuery ML?

BQML eliminates the biggest friction in enterprise ML: moving data out of the warehouse and into a separate training environment. BigQuery ML lets you create, train, evaluate, and predict with ML models using SQL directly in BigQuery -- no data movement, no separate ML infrastructure.

Core Syntax

These three SQL statements -- CREATE MODEL, ML.PREDICT, and ML.EVALUATE -- are the backbone of every BQML workflow. Master them and you can train, evaluate, and deploy models without leaving SQL.

CREATE MODEL

```
CREATE OR REPLACE MODEL `project.dataset.model_name`
OPTIONS(
  model_type = 'LOGISTIC_REG',          -- required
  input_label_cols = ['target_column'],   -- required for supervised
  auto_class_weights = TRUE,              -- optional: handle imbalanced classes
  max_iterations = 20,                    -- optional: training iterations
  learn_rate = 0.1,                       -- optional: learning rate
  l1_reg = 0.001,                         -- optional: L1 regularization
  l2_reg = 0.001,                         -- optional: L2 regularization
  early_stop = TRUE,                      -- optional: stop when no improvement
  data_split_method = 'AUTO_SPLIT'        -- optional: train/eval split
) AS
SELECT
  feature1,
  feature2,
  feature3,
  target_column
FROM `project.dataset.training_table`
WHERE date BETWEEN '2025-01-01' AND '2025-12-31';
```

ML.PREDICT

```
-- Batch prediction
SELECT *
FROM ML.PREDICT(
  MODEL `project.dataset.model_name`,
  (SELECT feature1, feature2, feature3
   FROM `project.dataset.new_data`)
);

-- With threshold for classification
SELECT *,
  CASE WHEN predicted_target_column_probs[OFFSET(1)].prob > 0.7
       THEN 'positive' ELSE 'negative'
  END AS high_confidence_label
FROM ML.PREDICT(
  MODEL `project.dataset.model_name`,
```

```
  (SELECT * FROM `project.dataset.new_data`)
);
```

ML.EVALUATE

```
-- Overall model evaluation
SELECT *
FROM ML.EVALUATE(
  MODEL `project.dataset.model_name`,
  (SELECT * FROM `project.dataset.eval_data`)
);

-- Evaluate with specific threshold
SELECT *
FROM ML.EVALUATE(
  MODEL `project.dataset.model_name`,
  (SELECT * FROM `project.dataset.eval_data`),
  STRUCT(0.5 AS threshold)
);
```

Supported Model Types

BQML supports everything from simple linear regression to imported TensorFlow models and remote LLM calls. Knowing which model_type value to use saves you from digging through documentation mid-query.

Model Type	`model_type` Value	Task	Notes
Linear Regression	`LINEAR_REG`	Regression	Continuous target
Logistic Regression	`LOGISTIC_REG`	Binary/multi-class	Default for classification
K-Means	`KMEANS`	Clustering	Unsupervised
Matrix Factorization	`MATRIX_FACTORIZATION`	Recommendation	User-item interactions
PCA	`PCA`	Dimensionality reduction	Unsupervised
XGBoost	`BOOSTED_TREE_CLASSIFIER`	Classification	Gradient boosting
XGBoost	`BOOSTED_TREE_REGRESSOR`	Regression	Gradient boosting
Random Forest	`RANDOM_FOREST_CLASSIFIER`	Classification	Ensemble trees
Random Forest	`RANDOM_FOREST_REGRESSOR`	Regression	Ensemble trees
DNN	`DNN_CLASSIFIER`	Classification	Deep neural network
DNN	`DNN_REGRESSOR`	Regression	Deep neural network
Wide & Deep	`DNN_LINEAR_COMBINED_CLASSIFIER`	Classification	Wide + deep combo
ARIMA Plus	`ARIMA_PLUS`	Time series forecast	Automatic seasonality

Model Type	`model_type` Value	Task	Notes
AutoML Tables	`AUTOML_CLASSIFIER`	Classification	Automatic architecture
AutoML Tables	`AUTOML_REGRESSOR`	Regression	Automatic architecture
TensorFlow (imported)	`TENSORFLOW`	Any	Import SavedModel
ONNX (imported)	`ONNX`	Any	Import ONNX model
LLM (remote)	`CLOUD_AI_LLM_V1`	Text generation	Gemini, PaLM

Model Selection Guide

When you are not sure which model type to pick, default to boosted trees for tabular data -- they consistently outperform other BQML options on structured datasets.

Scenario	Recommended Model	Why
Quick binary classification	`LOGISTIC_REG`	Fast, interpretable
Tabular classification (best accuracy)	`BOOSTED_TREE_CLASSIFIER`	Usually best for tabular
Regression with many features	`BOOSTED_TREE_REGRESSOR`	Handles non-linearity
Customer segmentation	`KMEANS`	Unsupervised grouping
Sales forecasting	`ARIMA_PLUS`	Time series native
Recommendation engine	`MATRIX_FACTORIZATION`	Collaborative filtering
Don't know what to pick	`AUTOML_CLASSIFIER/REGRESSOR`	Let BQML decide
Use existing TF model	`TENSORFLOW`	Import and serve

Feature Engineering in BQML

BQML's TRANSFORM clause lets you define feature engineering in SQL, and it automatically applies the same transforms at prediction time. This eliminates training-serving skew -- the most common cause of model performance drops in production.

Built-in Transforms

```
CREATE OR REPLACE MODEL `project.dataset.my_model`
TRANSFORM(
  -- Numeric: standardize
  ML.STANDARD_SCALER(age) OVER() AS age_scaled,
  ML.MIN_MAX_SCALER(income) OVER() AS income_scaled,

  -- Numeric: bucketize
  ML.BUCKETIZE(age, [18, 25, 35, 50, 65]) AS age_bucket,
```

```
  -- Quantile bucketize
  ML.QUANTILE_BUCKETIZE(income, 10) OVER() AS income_decile,

  -- Categorical: one-hot (automatic for STRING/BOOL)
  category,

  -- Text: TF-IDF or bag of words
  ML.NGRAMS(description, [1, 2]) AS description_ngrams,

  -- Polynomial
  ML.POLYNOMIAL_EXPAND(STRUCT(feature1, feature2), 2) AS poly_features,

  -- Feature cross
  ML.FEATURE_CROSS(STRUCT(city, device_type)) AS city_device,

  -- Label
  target_column
)
OPTIONS(model_type='BOOSTED_TREE_CLASSIFIER', input_label_cols=['target_column'])
AS SELECT * FROM `project.dataset.training_data`;
```

Common Feature Patterns

Pattern	SQL Example	Use Case
Date parts	`EXTRACT(DAYOFWEEK FROM date)`	Cyclical patterns
Ratios	`clicks / NULLIF(impressions, 0)`	Rate metrics
Lag features	`LAG(value, 7) OVER(ORDER BY date)`	Time series
Rolling avg	`AVG(value) OVER(ORDER BY date ROWS 7 PRECEDING)`	Smoothing
Log transform	`LN(value + 1)`	Skewed distributions
Null handling	`IFNULL(value, 0)`	Missing data
String length	`LENGTH(text_field)`	Text features
Count encoding	Subquery with COUNT(*) per category	High cardinality

Time Series with ARIMA_PLUS

ARIMA_PLUS is BQML's most powerful built-in model -- it automatically handles seasonality, holidays, spike cleaning, and multiple time series in a single query. For most business forecasting tasks, it is the fastest path to production-quality predictions.

```
-- Create time series model
CREATE OR REPLACE MODEL `project.dataset.sales_forecast`
OPTIONS(
  model_type = 'ARIMA_PLUS',
  time_series_timestamp_col = 'date',
  time_series_data_col = 'daily_sales',
  time_series_id_col = 'store_id',  -- multiple series
  auto_arima = TRUE,
```

```
  data_frequency = 'DAILY',
  holiday_region = 'US',
  clean_spikes_and_dips = TRUE,
  adjust_step_changes = TRUE
) AS
SELECT date, store_id, daily_sales
FROM `project.dataset.historical_sales`;

-- Forecast future values
SELECT *
FROM ML.FORECAST(
  MODEL `project.dataset.sales_forecast`,
  STRUCT(30 AS horizon, 0.9 AS confidence_level)
);

-- Explain forecast components
SELECT *
FROM ML.EXPLAIN_FORECAST(
  MODEL `project.dataset.sales_forecast`,
  STRUCT(30 AS horizon, 0.9 AS confidence_level)
);
```

LLM Integration in BQML

BQML can call Gemini and other Vertex AI models directly from SQL, enabling text generation and embedding at warehouse scale. This lets you enrich millions of rows with LLM outputs without building a separate pipeline.

```
-- Create a remote model connection to Gemini
CREATE OR REPLACE MODEL `project.dataset.gemini_model`
REMOTE WITH CONNECTION `project.us.vertex-ai-connection`
OPTIONS(endpoint = 'gemini-2.0-flash');

-- Use for text generation
SELECT *
FROM ML.GENERATE_TEXT(
  MODEL `project.dataset.gemini_model`,
  (SELECT
    CONCAT('Summarize this review: ', review_text) AS prompt
   FROM `project.dataset.reviews`
   LIMIT 100),
  STRUCT(
    0.2 AS temperature,
    1024 AS max_output_tokens,
    TRUE AS flatten_json_output
  )
);

-- Use for embeddings
CREATE OR REPLACE MODEL `project.dataset.embedding_model`
REMOTE WITH CONNECTION `project.us.vertex-ai-connection`
OPTIONS(endpoint = 'text-embedding-004');

SELECT *
FROM ML.GENERATE_EMBEDDING(
```

```
  MODEL `project.dataset.embedding_model`,
  (SELECT content AS content FROM `project.dataset.documents`),
  STRUCT(TRUE AS flatten_json_output)
);
```

Evaluation Metrics by Model Type

ML.EVALUATE returns different metrics depending on your model type. Knowing which metrics to expect -- and which evaluation queries to run -- prevents confusion when interpreting model performance.

Model Type	Metrics Returned by ML.EVALUATE
Classification	precision, recall, accuracy, f1_score, log_loss, roc_auc
Regression	mean_absolute_error, mean_squared_error, r2_score, mean_squared_log_error
Clustering	davies_bouldin_index, mean_squared_distance
Time Series	AIC, variance, log_likelihood, seasonal_periods

Useful Evaluation Queries

```
-- Confusion matrix
SELECT *
FROM ML.CONFUSION_MATRIX(
  MODEL `project.dataset.model_name`,
  (SELECT * FROM `project.dataset.eval_data`)
);

-- ROC curve
SELECT *
FROM ML.ROC_CURVE(
  MODEL `project.dataset.model_name`,
  (SELECT * FROM `project.dataset.eval_data`)
);

-- Feature importance (tree models)
SELECT *
FROM ML.FEATURE_IMPORTANCE(MODEL `project.dataset.model_name`);

-- Feature info
SELECT *
FROM ML.FEATURE_INFO(MODEL `project.dataset.model_name`);

-- Model weights (linear/logistic)
SELECT *
FROM ML.WEIGHTS(MODEL `project.dataset.model_name`);
```

Model Export

BQML models trained in BigQuery can be exported and deployed to Vertex AI endpoints for low-latency serving. Not all model types support export -- check this table before planning your deployment strategy.

```
-- Export to Cloud Storage
EXPORT MODEL `project.dataset.model_name`
OPTIONS(uri = 'gs://bucket/exported_model/');
```

Model Type	Export Format	Deploy To
Linear/Logistic	TensorFlow SavedModel	Vertex AI, TF Serving
Boosted Tree	Booster + TF SavedModel	Vertex AI, XGBoost
DNN	TensorFlow SavedModel	Vertex AI, TF Serving
ARIMA_PLUS	Not exportable	Use ML.FORECAST in BQ
AutoML	Not directly	Use Vertex AI

When to Use BQML

BQML shines when your data is already in BigQuery and your team thinks in SQL. It is not the right tool for every ML problem -- know its strengths and limitations before committing.

Use BQML When

- Data already lives in BigQuery
- Team is SQL-proficient but not ML-expert
- Need quick prototyping or baseline models
- Tabular data classification/regression
- Time series forecasting
- Want to avoid data movement

Do NOT Use BQML When

- Need custom neural architectures
- Working with images, audio, or video (train from scratch)
- Need real-time online learning
- Require model interpretability beyond built-in methods
- Training budget is very tight (BQML can be expensive at scale)

Cost Considerations

BQML charges for both data processing and ML compute slots, and costs can escalate quickly with large datasets on on-demand pricing. Understand the pricing model before running CREATE MODEL on your full dataset.

Operation	Pricing
CREATE MODEL	Charged per bytes processed + ML training slot time

Operation	Pricing
ML.PREDICT	Charged per bytes processed
ML.EVALUATE	Charged per bytes processed
Model storage	Standard BQ storage rates
On-demand training	~$250/slot/hour for ML
Flat-rate (editions)	Included in Enterprise/Enterprise Plus

Common Pitfalls

BQML's SQL simplicity makes it easy to train models quickly but also easy to make mistakes that go unnoticed. These pitfalls are especially common for SQL practitioners new to ML.

Pitfall	Problem	Fix
No eval split	Overfitting not detected	Use `data_split_method='AUTO_SPLIT'`
All features as-is	Poor model performance	Use TRANSFORM for scaling, encoding
Ignoring class imbalance	Biased predictions	Set `auto_class_weights=TRUE`
SELECT * for training	Includes ID columns, data leakage	Explicitly select features
No feature importance	Can't explain model	Run ML.FEATURE_IMPORTANCE
Forecasting without cleaning	Bad forecasts from outliers	Set `clean_spikes_and_dips=TRUE`
Large training data with on-demand	High cost	Use flat-rate pricing or sample data
Not exporting for production	Stuck in BQ for serving	Export and deploy to Vertex AI endpoint