Optimizing ML Prediction Queries and Beyond on Modern Data Engines

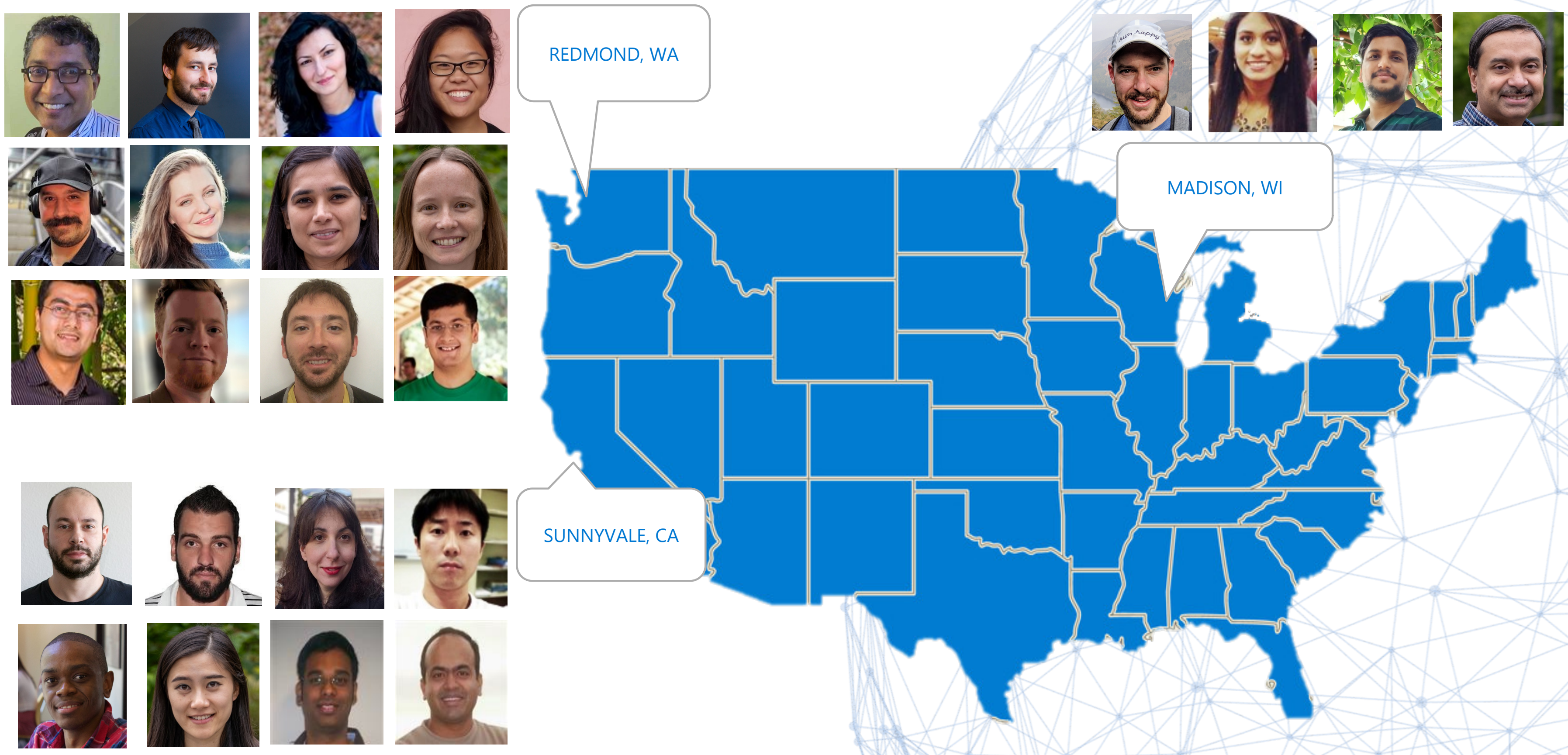
Konstantinos Karanasos

DSDSD
October 1st, 2021

Gray Systems Lab

Azure Data applied research lab

Focus on Systems, DB, ML research to deliver product, OSS, and academic impact



~25 scientists, research engineers, and data scientists

ML-related GSL Areas of Focus

Enterprise-Grade ML

ML for Systems

The Data Science Dream

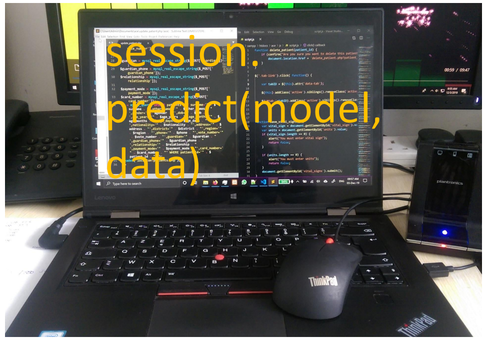
The Data Science Dream

Data Scientist

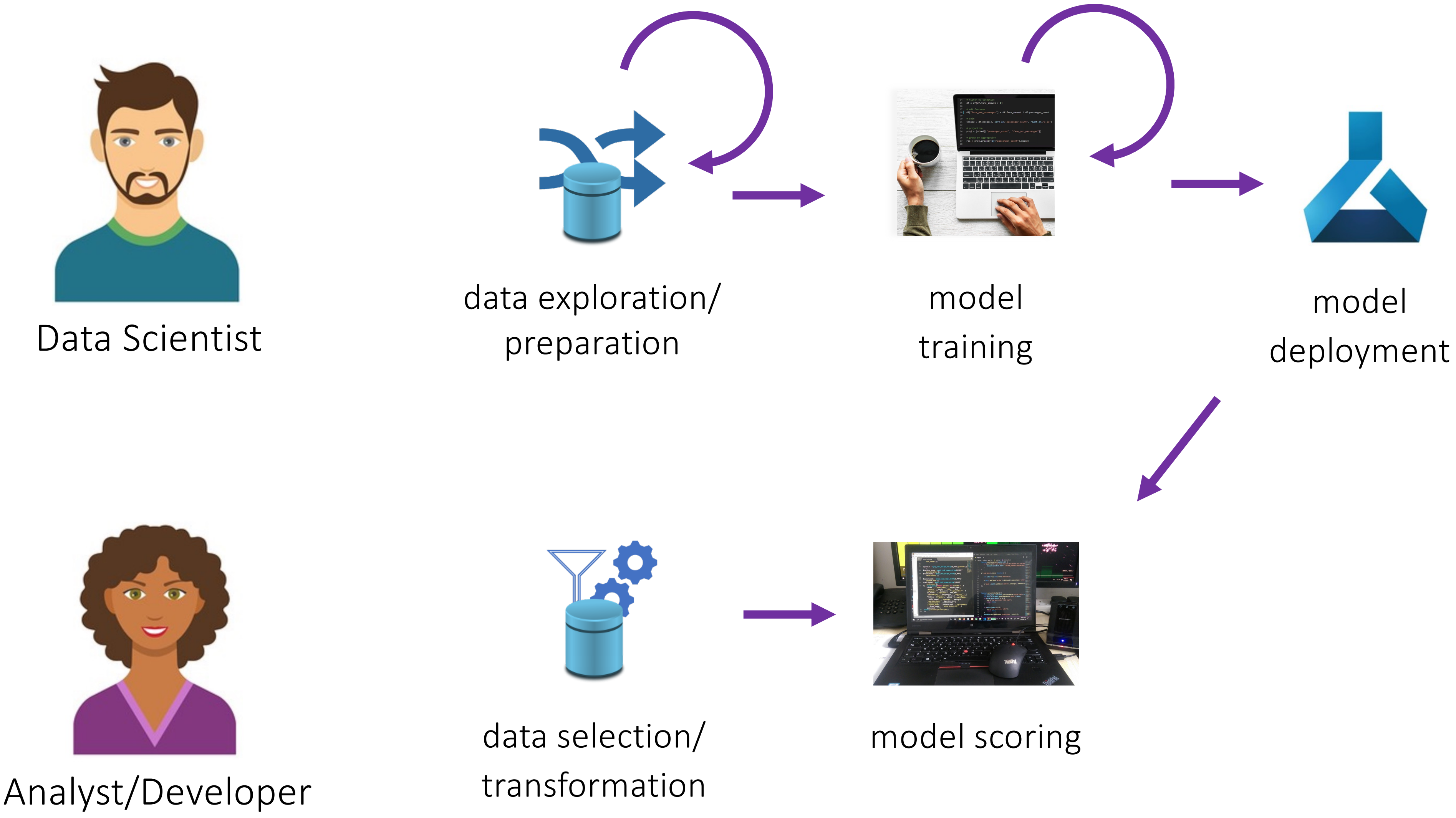
"I want to train models on my laptop and share them"

Analyst/Developer

"I want to use existing models to make predictions"

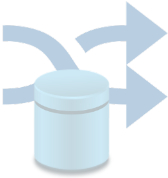


The Data Science Reality



The Data Science Challenges

- Data cleaning
- Combining datasets
- Featurization



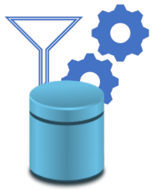
data exploration/ preparation

model training

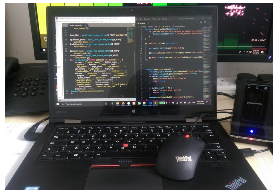
model deployment

- Data provenance
- Model provenance
- Policy enforcement

- SQL vs. Python
- Scale computation
- Support multiple engines



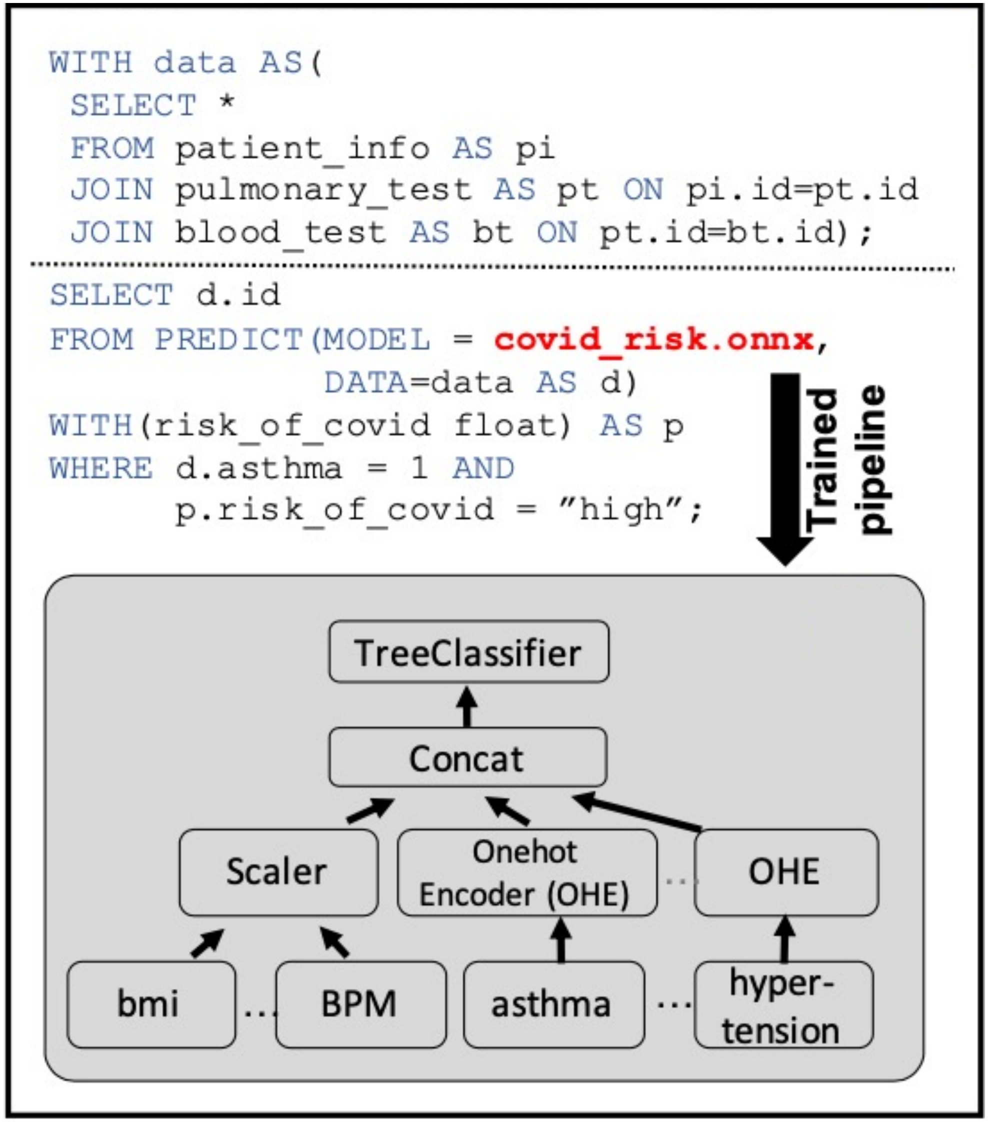
data selection/ transformation



model scoring

- Different model types
- End-to-end pipeline optimization
- Hardware acceleration

Prediction Query: Example



```sql
WITH data AS(
 SELECT *
 FROM patient_info AS pi
 JOIN pulmonary_test AS pt ON pi.id=pt.id
 JOIN blood_test AS bt ON pt.id=bt.id);
SELECT d.id
FROM PREDICT(MODEL = covid_risk.onnx,
             DATA=data AS d)
WITH(risk_of_covid float) AS p
WHERE d.asthma = 1 AND
      p.risk_of_covid = ”high”;
```


“Find patients with asthma who are at high risk of developing a severe COVID-19 case”

Enterprise Prediction Queries: Motivation

Scoring drives the cost of ML in the enterprise

Train once, predict several times
Pre-trained models
Up to 90% of the ML cost per estimates

Batch scoring is preferrable (or at least sufficient)

In 130 customer engagements, 91% were covered with batch
An additional 6% were okay with batch at short intervals

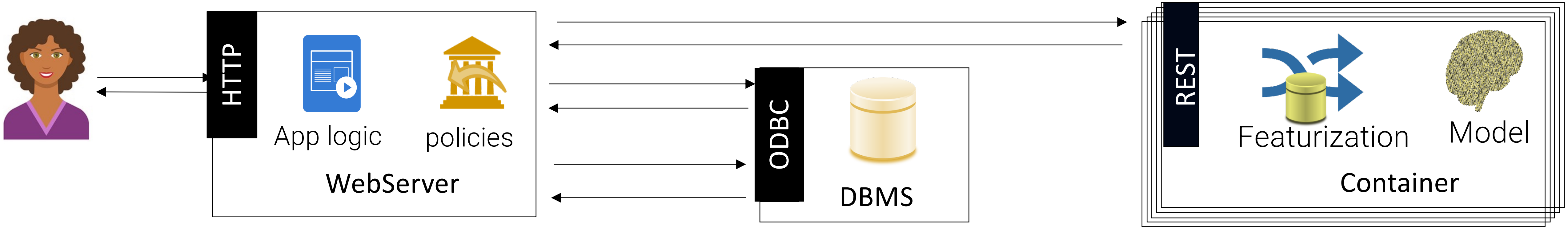
Traditional ML is most widely used over structured data

Linear/logistic regression, tree-based models, featurization
Kaggle survey: 80% of responders use it (43% use DL)
Analysis of 10M GitHub notebooks: <20% use DL `[arXiv2019]`

Execution of prediction queries:

Bring models closer to the data

Prediction Queries: Baseline Approach



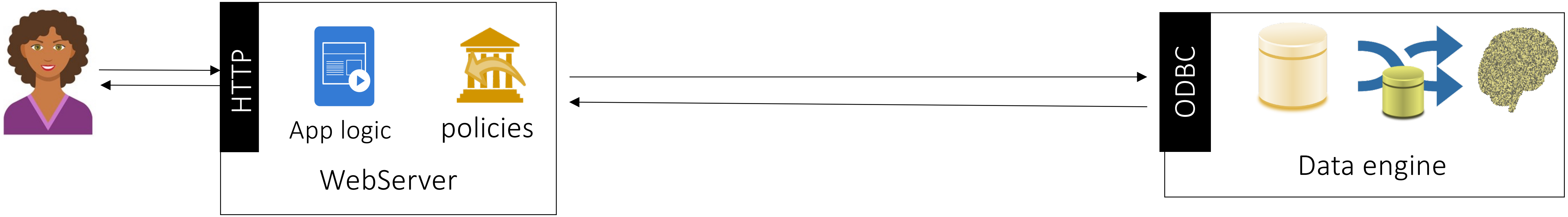
Enterprise Features

Security: data and models outside of the DB
Extra infrastructure
Lack of tooling/best-practices

Performance

Data movement
Latency
Throughput

Prediction Queries: In-Engine Evaluation



Enterprise Features

Security: Data and models within the DBMS

Reuse Existing infrastructure

Language/tools/best practices

Performance ?

Up to 13x faster on Spark

Up to 330x faster on SQL Server

Prediction Queries in Azure Data Engines

SQL Server

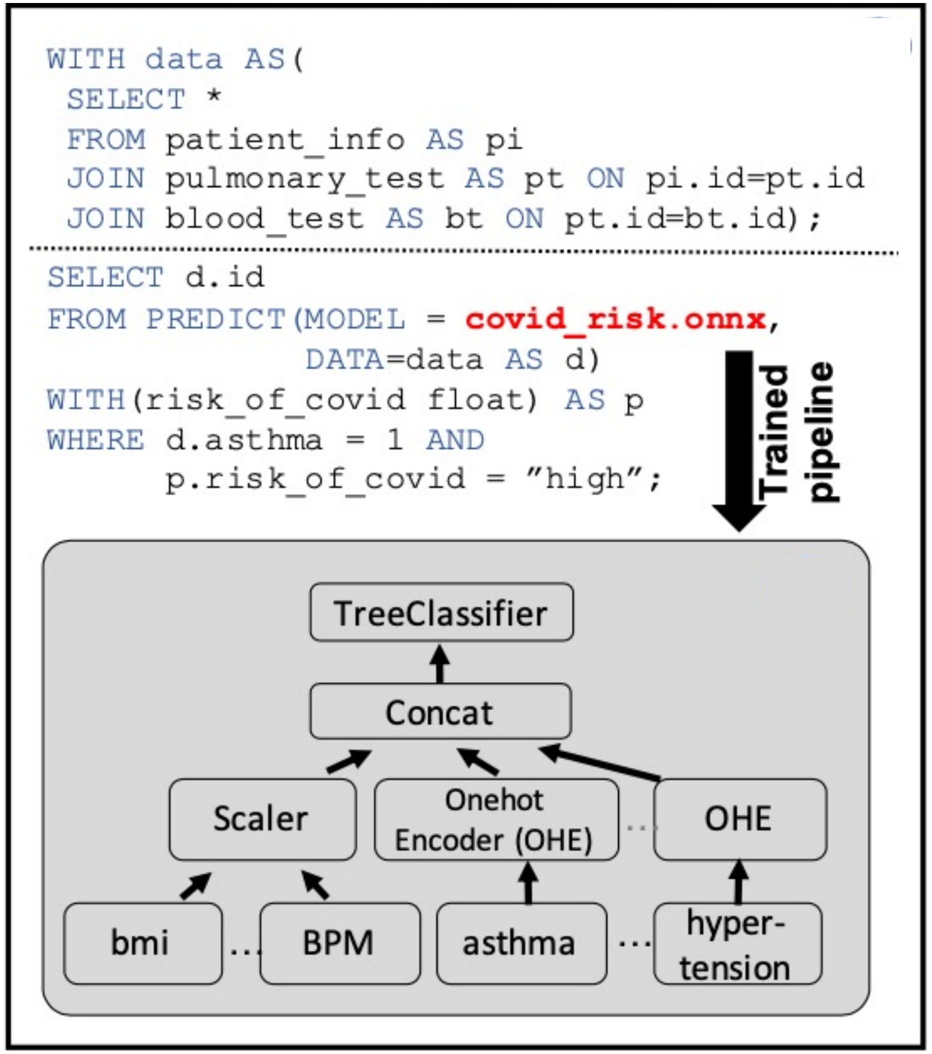
- PREDICT statement in SQL Server
- Embedded ONNX Runtime in the engine
- Available in Azure SQL Edge and SQL DW (part of Azure Synapse Analytics)

Spark

- Introduced a new PREDICT operator
- Similar syntax to SQL Server
- Support for different types of models

North-star goal

- Support any model on any engine



Any Model on Any Engine

Any model type

- Common representation (MLflow)
- Common API for predictions (through Python or REST)

Data movement between data and ML engines

- Text-based, Arrow?
- Local vs. remote models

Model lifecycle management

- Integration with model registry (e.g., AzureML)

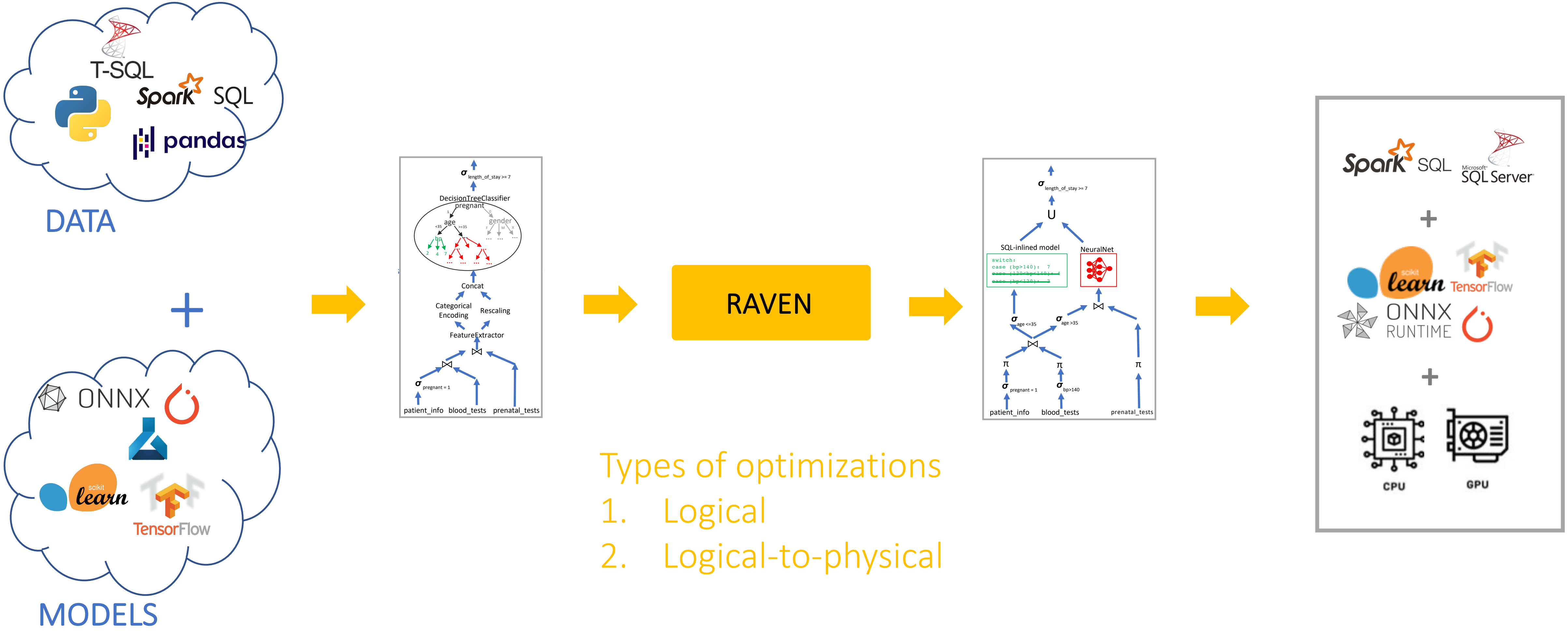
Containerized execution

- Library dependencies
- Any model, any language bindings, across any engine

Raven

Optimization of Prediction Queries

Raven



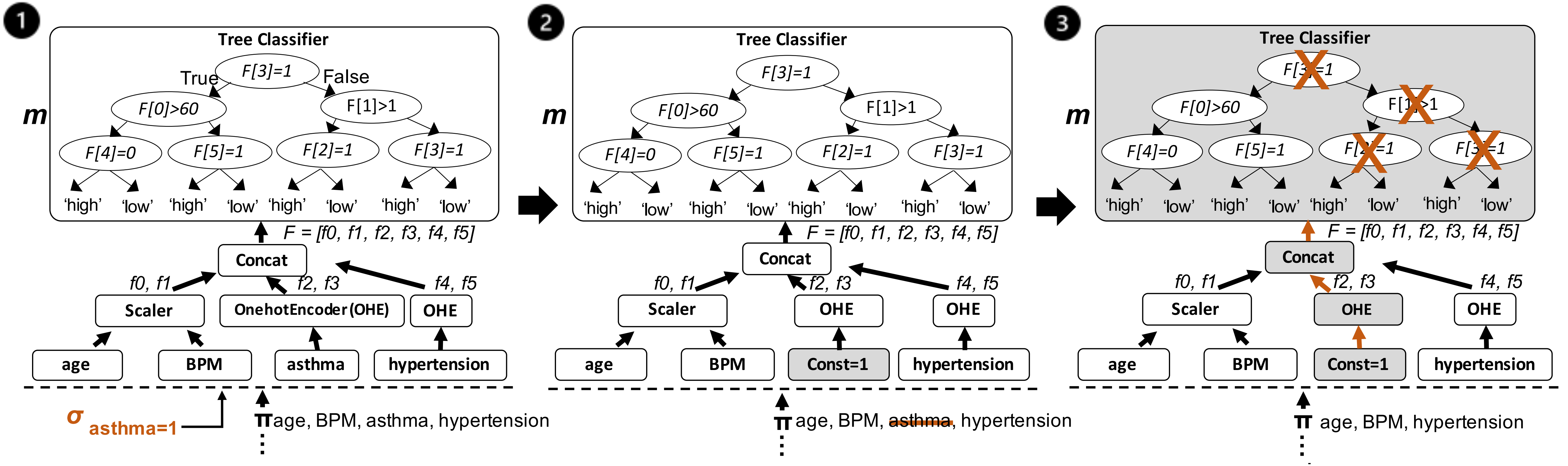
Types of optimizations

1. Logical
2. Logical-to-physical

[Initial vision presented in CIDR2020]

Logical Optimizations: Predicate-based Model Pruning

Information passing from the data part to the ML part



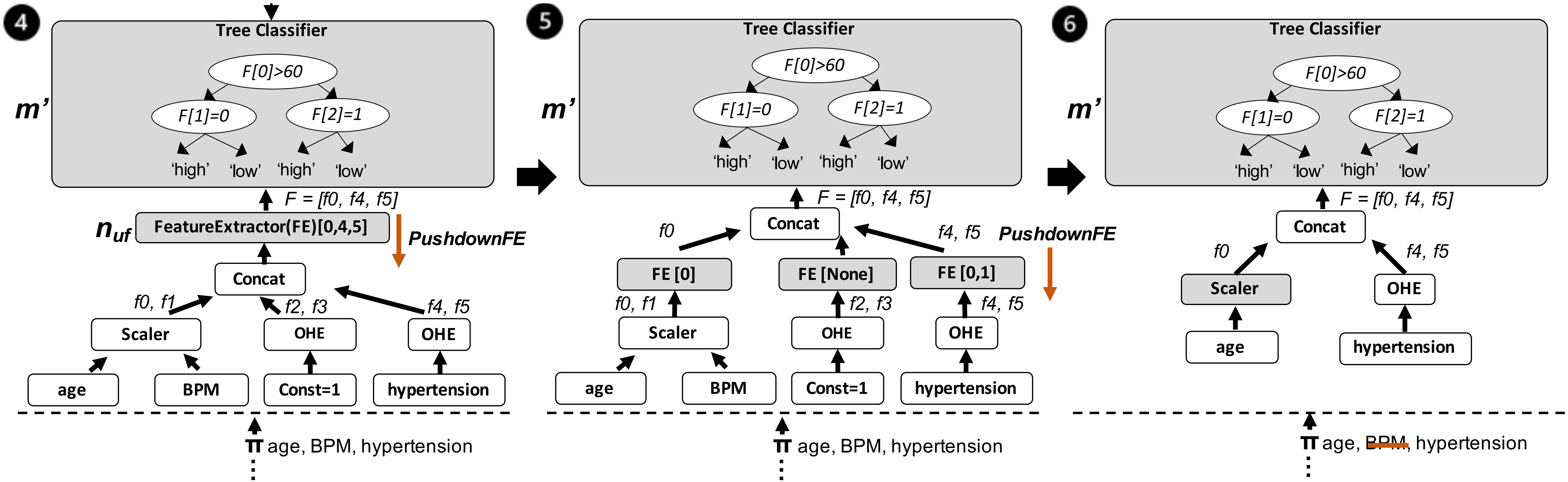
Data-induced optimizations:

Induce predicates from the data (based on statistics)

Compile different models per data partition

Logical Optimizations: Model Projection Pushdown

Information passing from the ML part to the data part



Very applicable in practice:

in 508 OpenML models we analyzed, 46% of features remained unused

Logical-to-Physical Optimizations

MLtoSQL

Turn a model to an equivalent SQL statement

Avoid invoking the ML runtime

MLtoDNN (Hummingbird)

Turn a traditional ML model to an equivalent neural network

Exploit modern DNN engines and HW acceleration

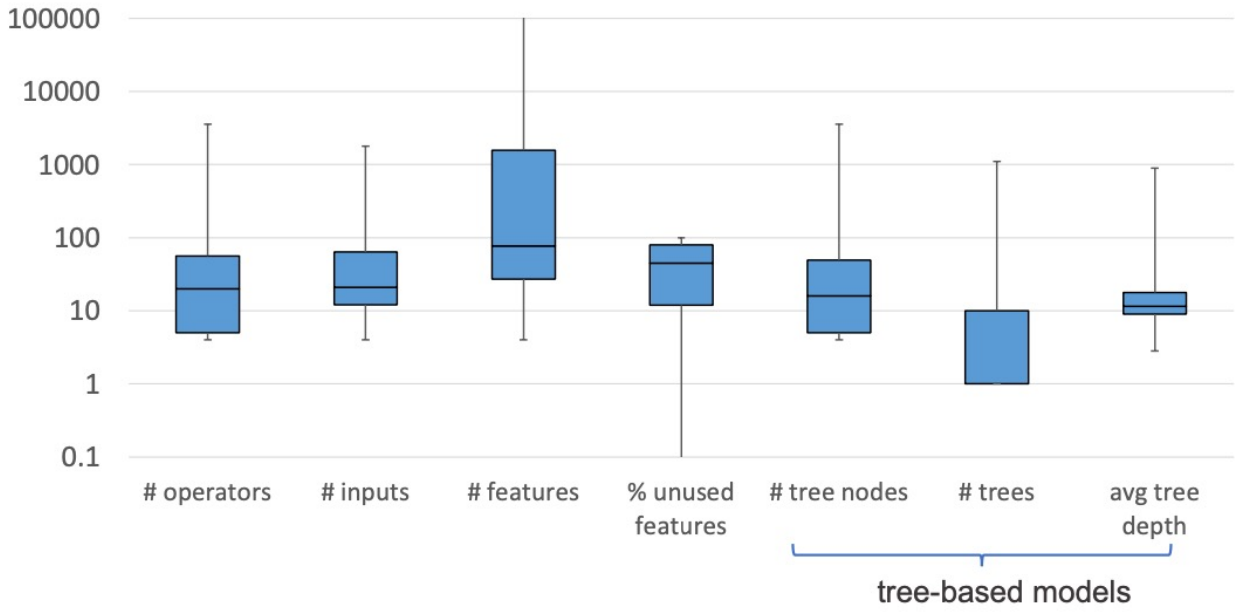
Optimization strategies

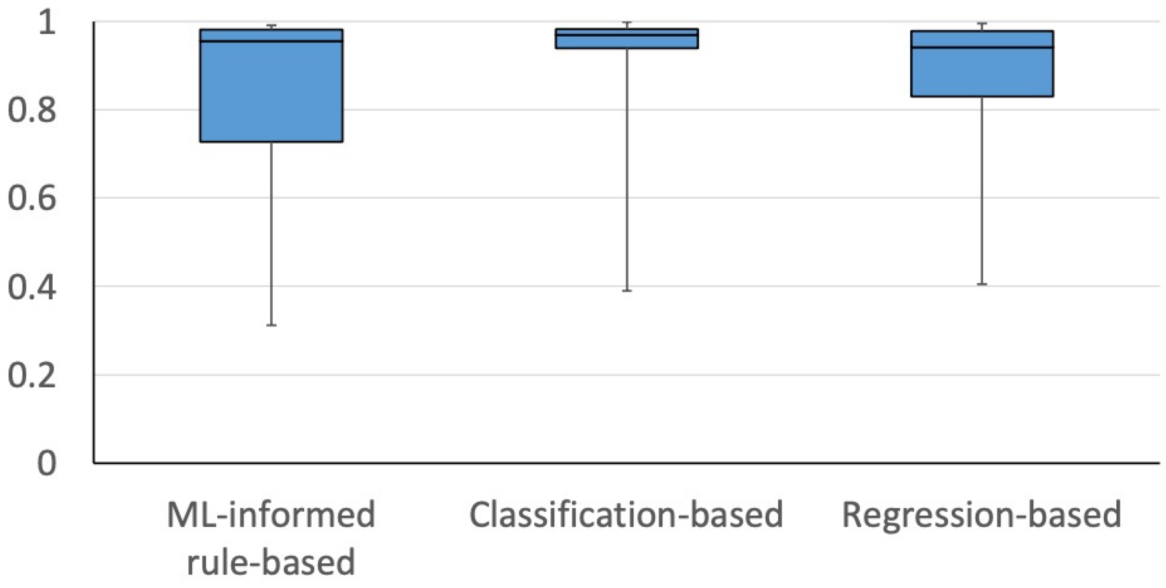
Which runtime to use: data engine or ML engine (traditional or DNN)?

Should we use the GPU (when available)?

Data-driven strategies to avoid hardcoded rules

Data-driven Optimization Strategies



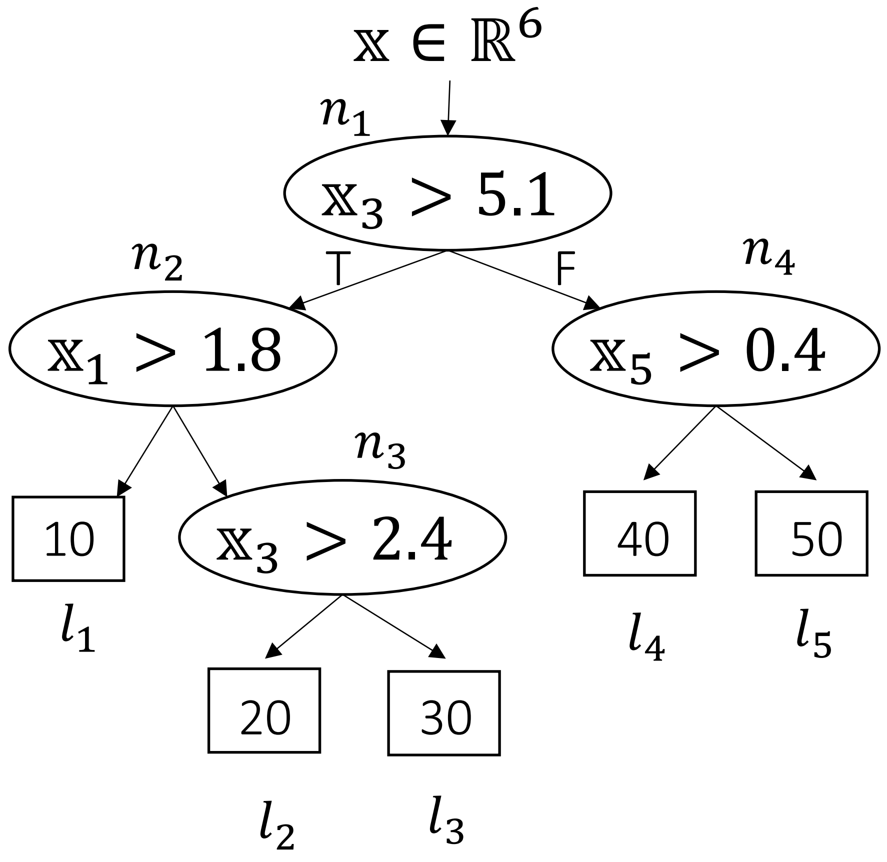


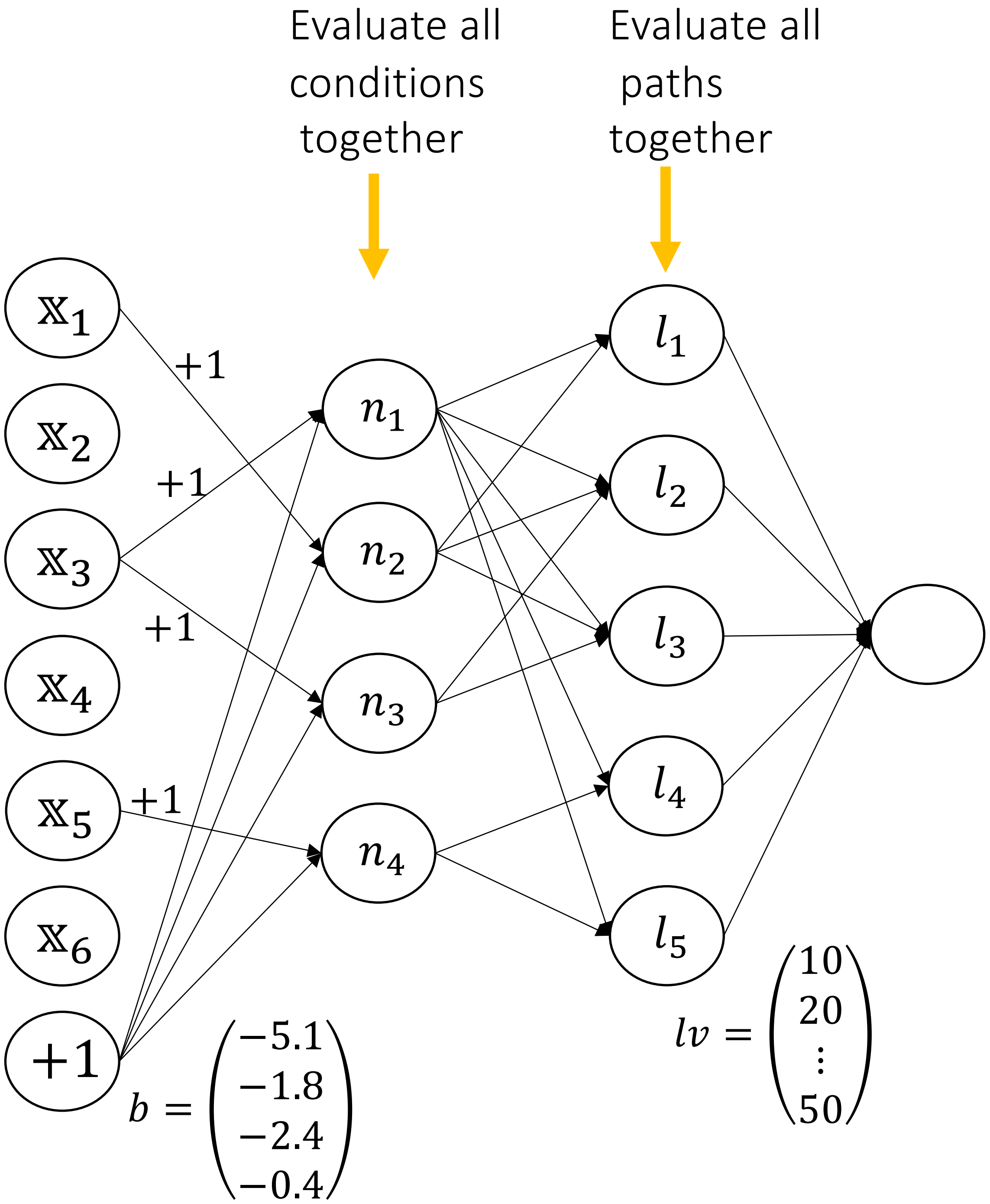
Trained pipelines vary greatly

Trained over OpenML dataset
Classification has best results but
ML-informed requires no model

Hummingbird: From trees to NNs

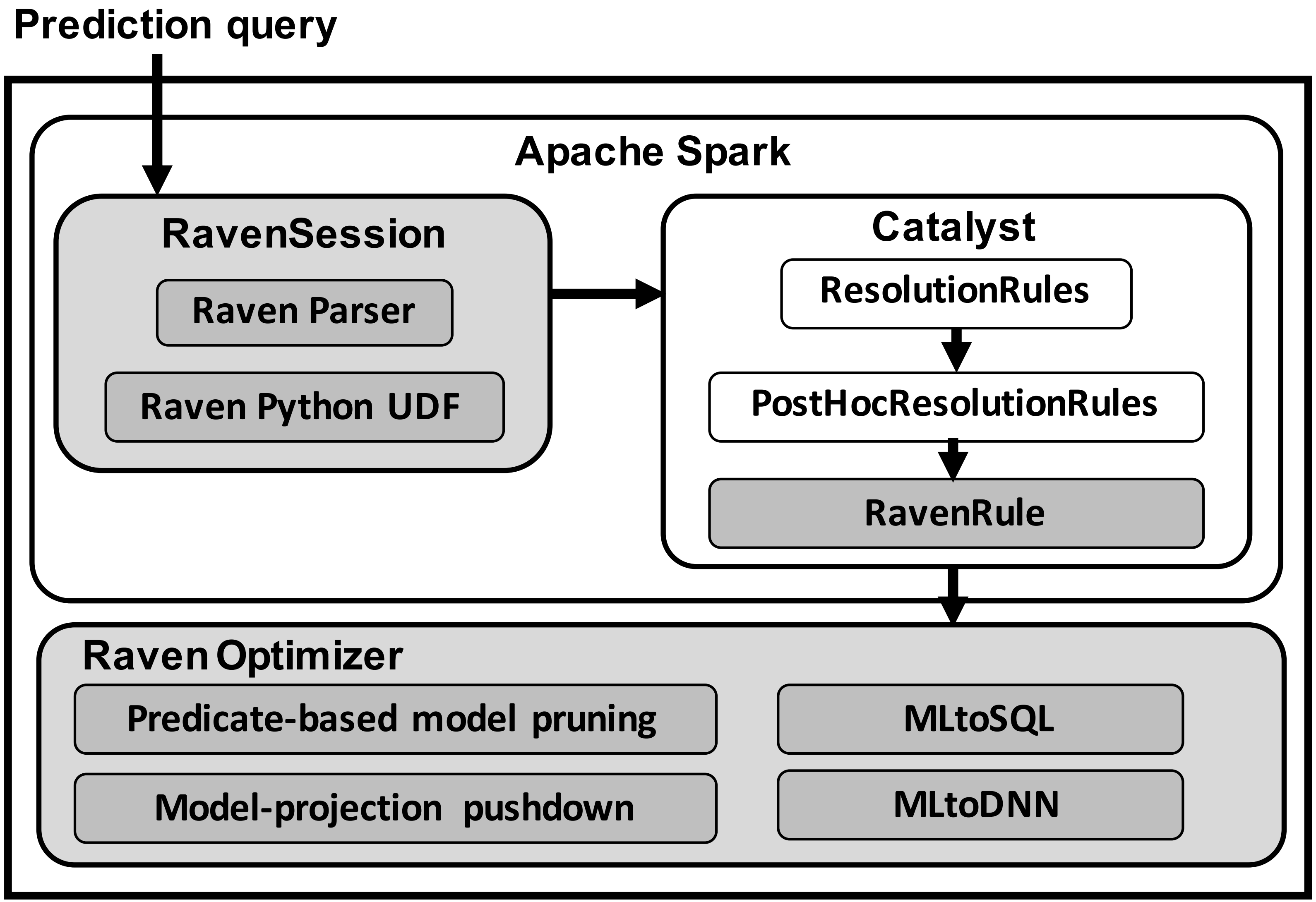
[OSDI2020]





https://github.com/microsoft/hummingbird

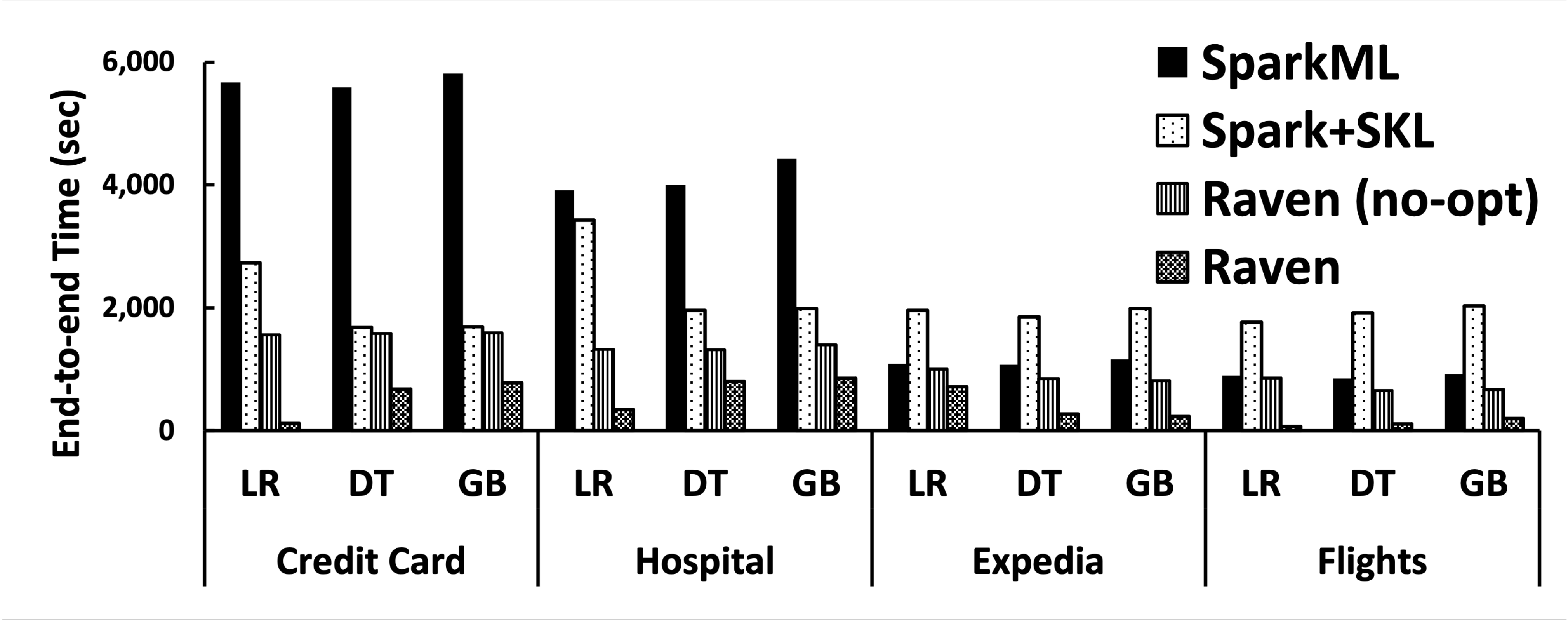
Raven Implementation on Spark



Developed as a Spark extension (can be used by any Spark installation)

Single Scala rule triggering the Raven Optimizer written in Python

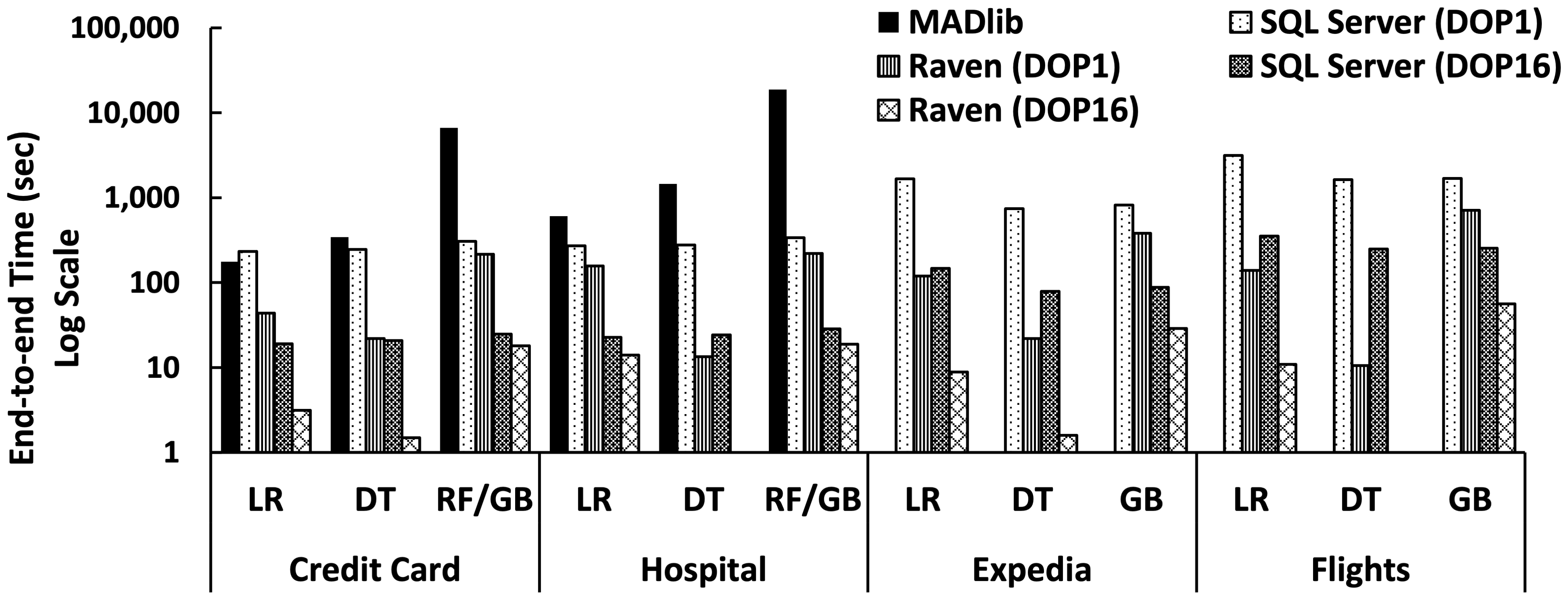
Evaluation: Raven on Spark



5-node Spark cluster
8 cores/56 GB per machine

1.4—13.1x faster than Raven without optimizations
1.5—48x faster than SparkML
2.15–25.3x faster than Spark with scikit-learn

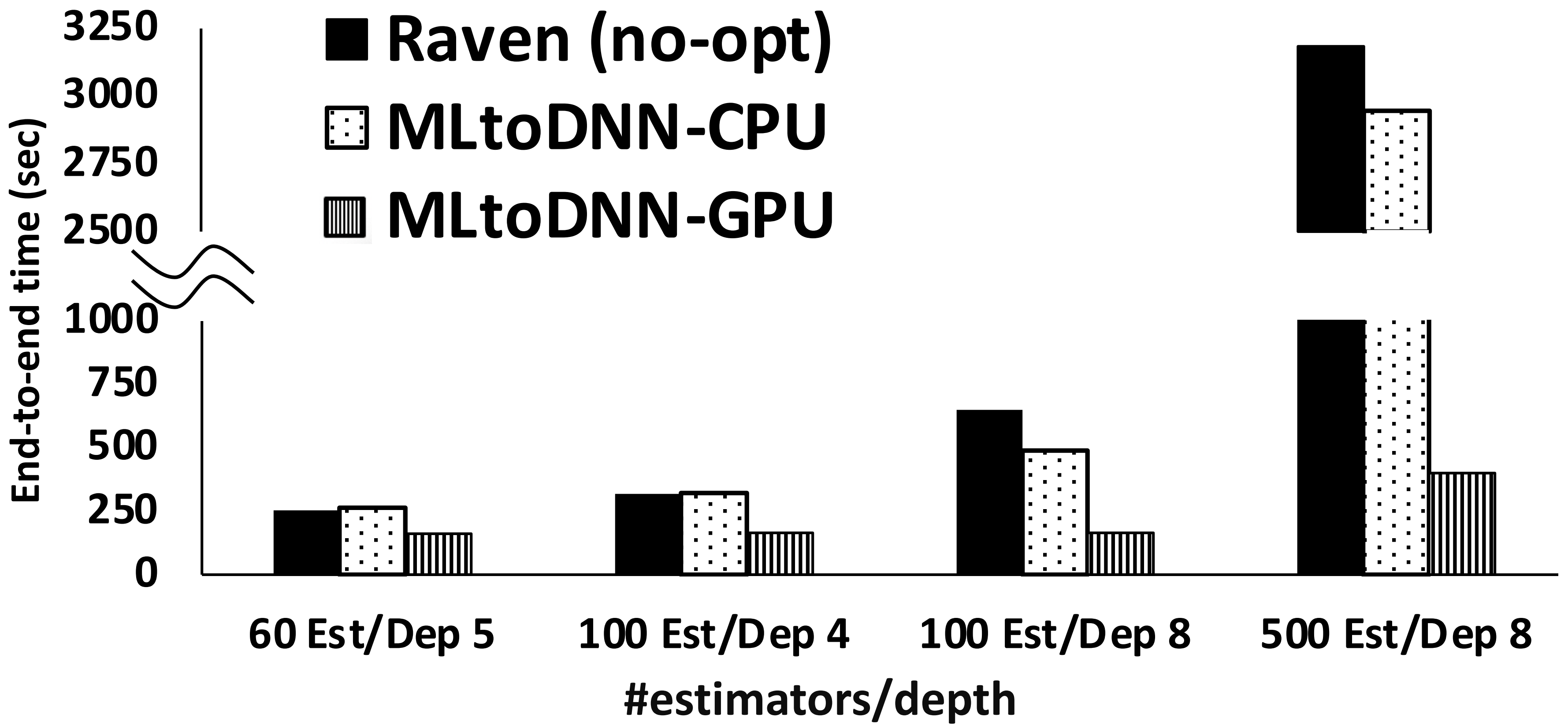
Evaluation: Raven Plans on SQL Server



32 vCores, 128 GB
Postgres for MADlib

1.4—330x faster than Raven without optimizations
3.9—108x faster than MADlib (for the queries it supports)

Evaluation: Impact of MLtoDNN on Complex Models



Hospital dataset
GB model of increasing complexity

Up to 1.33x speedup on CPU
1.6—8x speedup on GPU

Surakav

Using Tensor Runtimes Beyond ML

Offloading Relational Operators to Tensor Runtimes

[Vision in VLDB2021]

SQL

Query Optimizer

Surakav

Tensor Runtime

CPU

GPU

FPGA

Custom ASICS

Main ideas

1. Columnar data mapped into 2d tensors
2. Relational operators implemented using tensors operations
3. Hardware consciousness provided by the TR

Benefits

- Run queries on any HW supported by the tensor runtimes
- Leverage the massive development in tensor runtimes
- Avoid N*M explosion in implementation effort

Main Challenges

Expressivity

Can we cover all relational operators?

Current support: selection, filter, join, group-by, aggregates, sort, case, in, subqueries

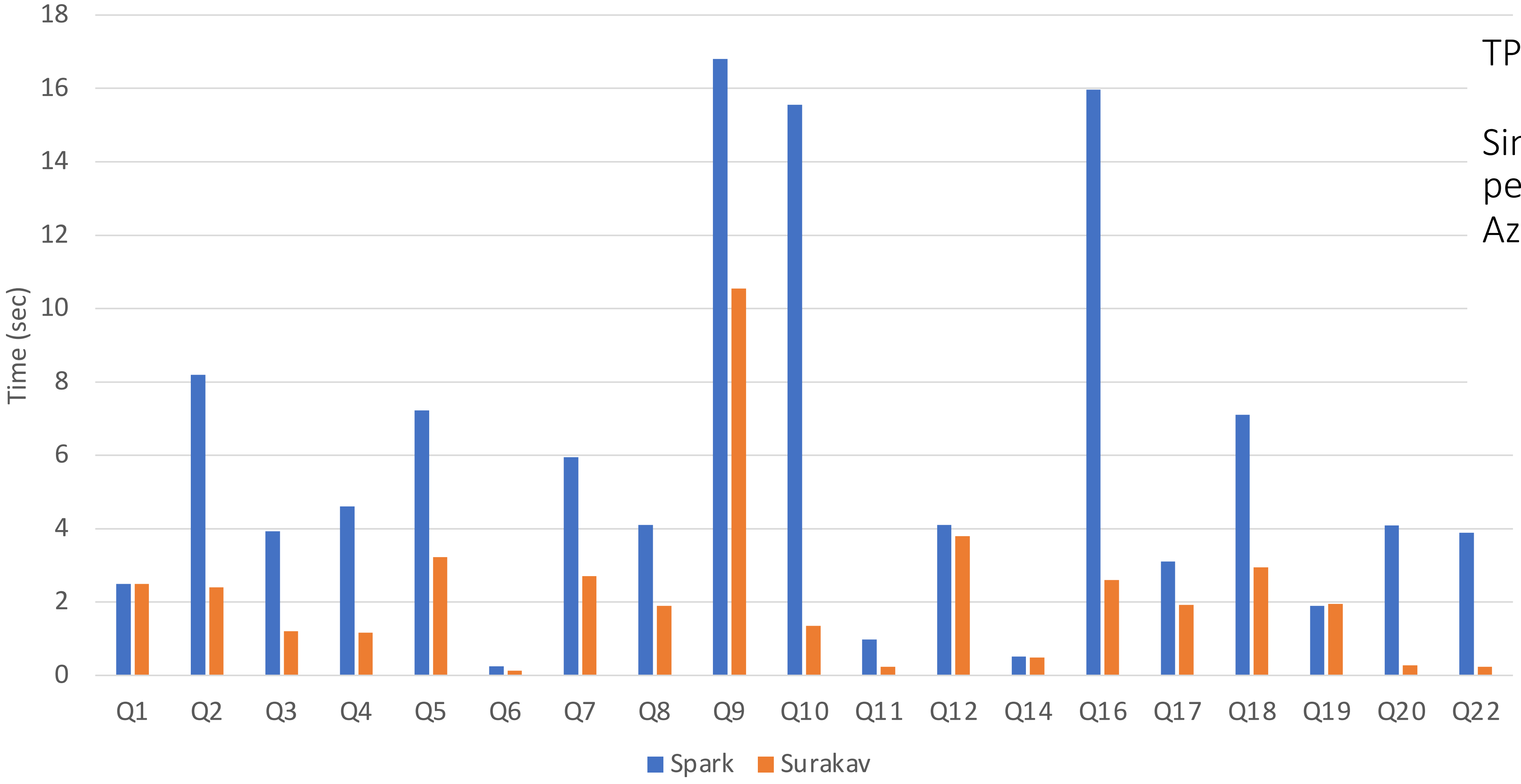
19 out of 22 TPC-H queries

Performance

Operator implementations should be “tensorized” to exploit GPU parallelism

Avoid loops as much as possible (breaks vectorization)

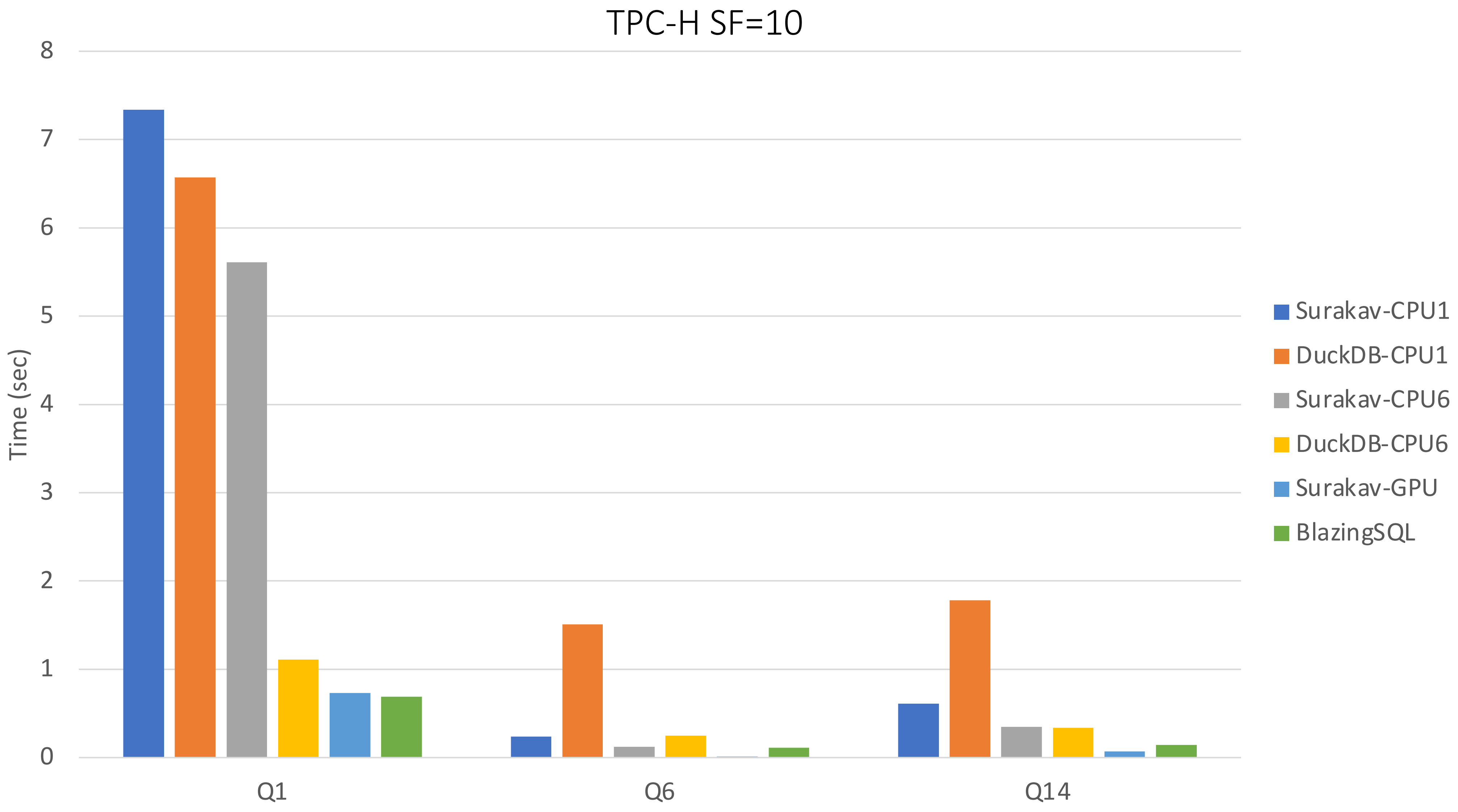
Surakav vs. Spark on CPU



TPC-H SF=1

Single-core performance on an Azure NC6 v2 machine

Surakav on CPU/GPU vs. DuckDB and BlazingSQL



Wrap-Up

Execution of prediction queries

Bring models closer to the data

Embed ML runtimes in data engines

Optimization of prediction queries

Logical and logical-to-physical optimizations

End-to-end implementation as a Spark extension

Up to 13x (330x) performance improvement on Spark (SQL Server)

Tensor runtimes beyond ML

Offload relational operators to ML accelerators

Leverage compiler advancements and modern hardware

Promising initial results against Spark, DuckDB, BlazingSQL

Microsoft

Thank you!

https://azuredata.microsoft.com/labs/gsl