Michelangelo

Jeremy Hermann, Machine Learning Platform @ Uber

Uber

ML at Uber

Uber Data

75 Million Riders

and 3 million drivers

4 Billion Trips

completed worldwide in 2017

65 Countries

and 600+ cities worldwide

15 Million Trips

completed each day

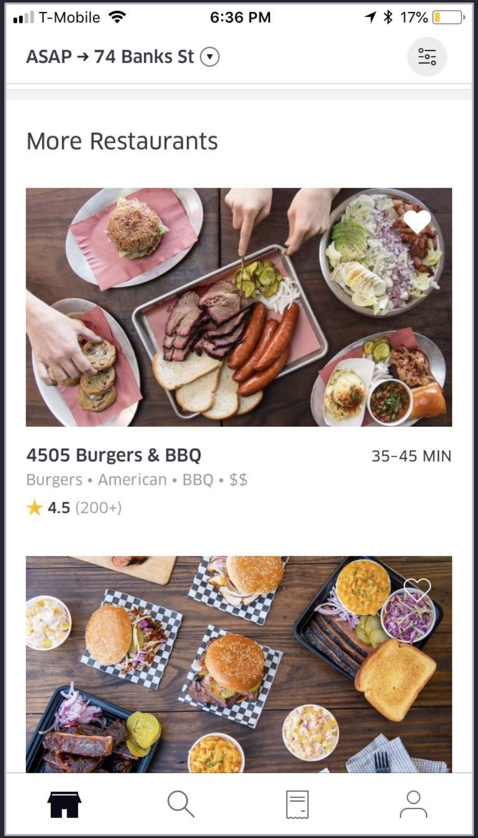
Uber Data

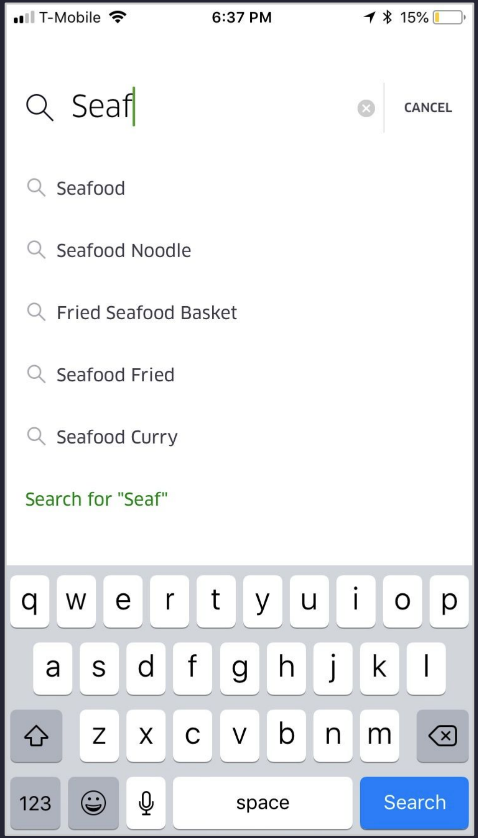
ML at Uber

- Uber Eats
- ETAs
- Self-Driving Cars
- Customer Support
- Dispatch
- Personalization
- Demand Modeling
- Dynamic Pricing
- Forecasting
- Maps
- Fraud
- Safety
- Destination Predictions
- Anomaly Detection
- Capacity Planning
- And many more...

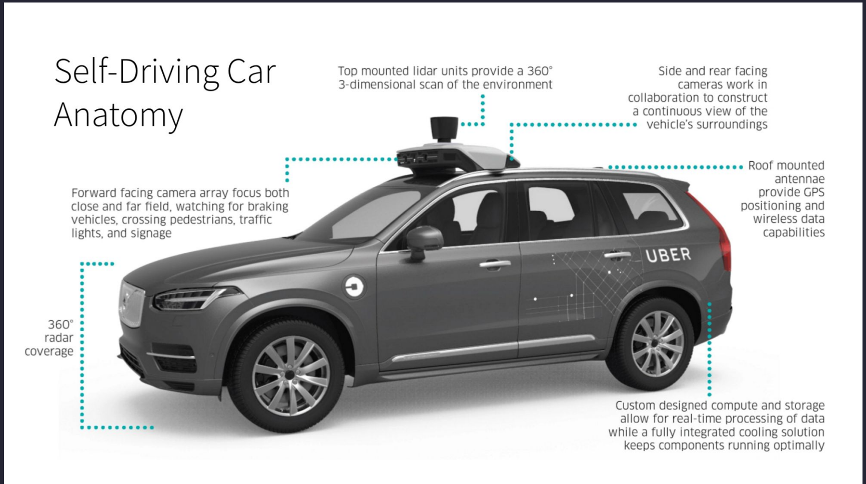
ML at Uber - Eats

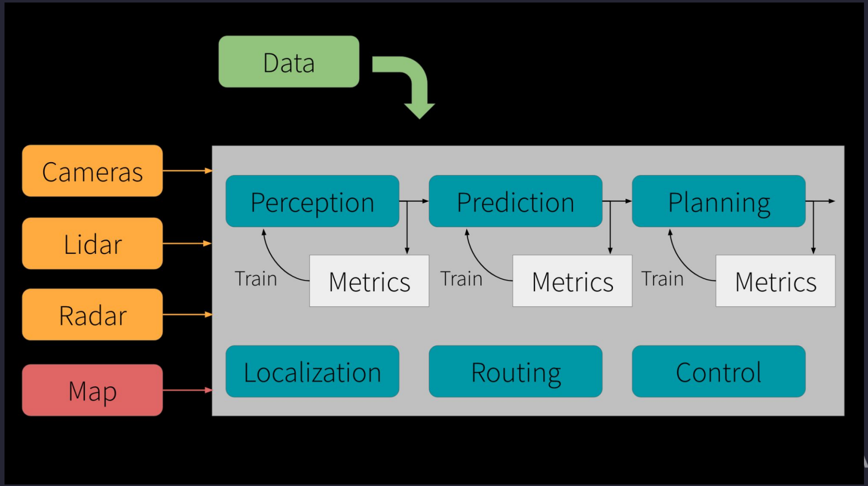
- Models used for
  - Ranking of restaurants and dishes
  - Delivery times
  - Search ranking
- 100s of ML models called to render Eats homepage





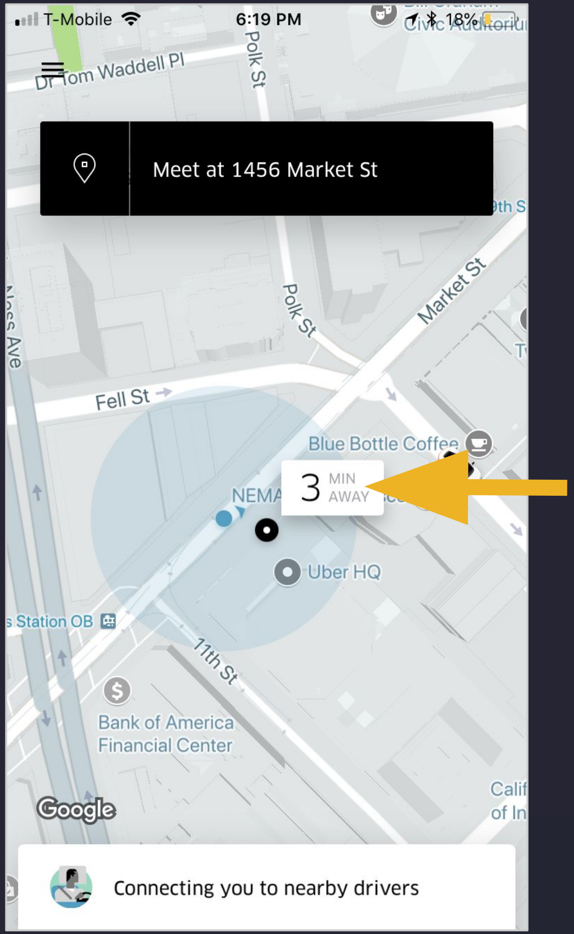
ML at Uber - Self-Driving Cars



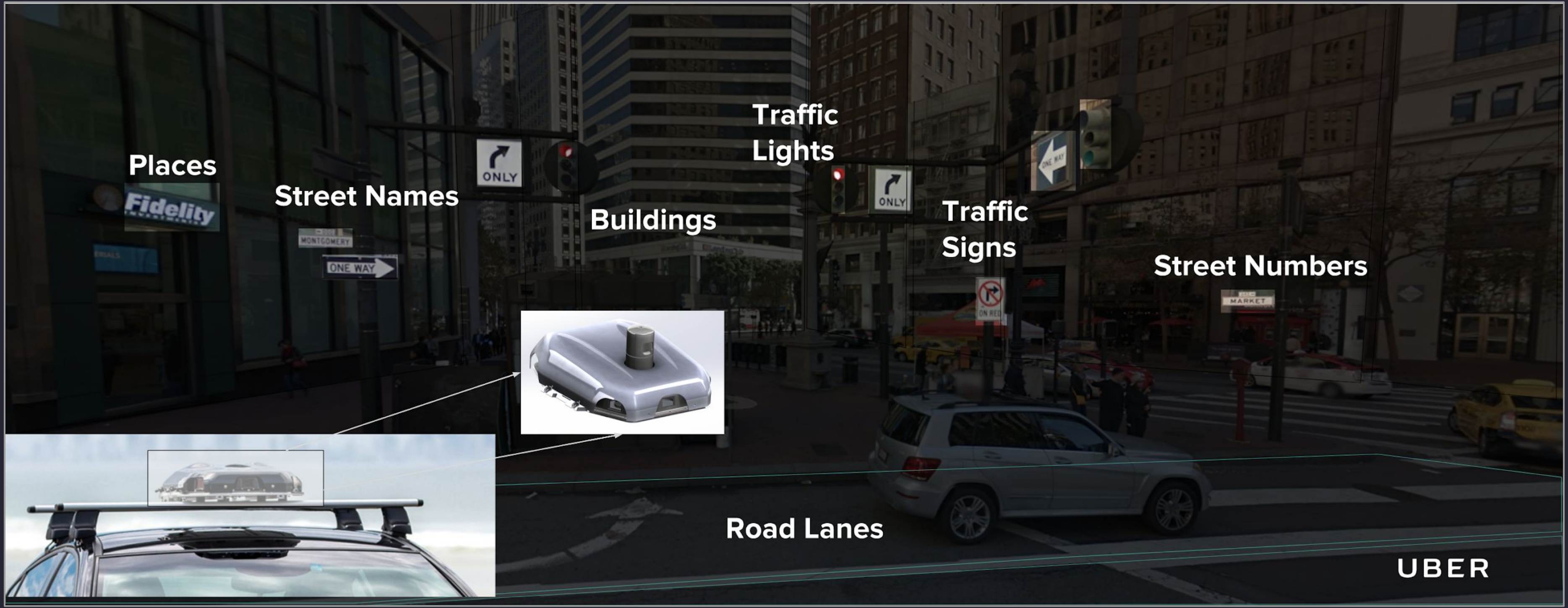


ML at Uber - ETAs

- ETAs are core to customer experience
- ETAs used by myriad internal systems
- ETA are generated by route-based algorithm called Garafu
- ML model predicts the Garafu error
- Use the predicted error to correct the ETA
- ETAs now dramatically more accurate



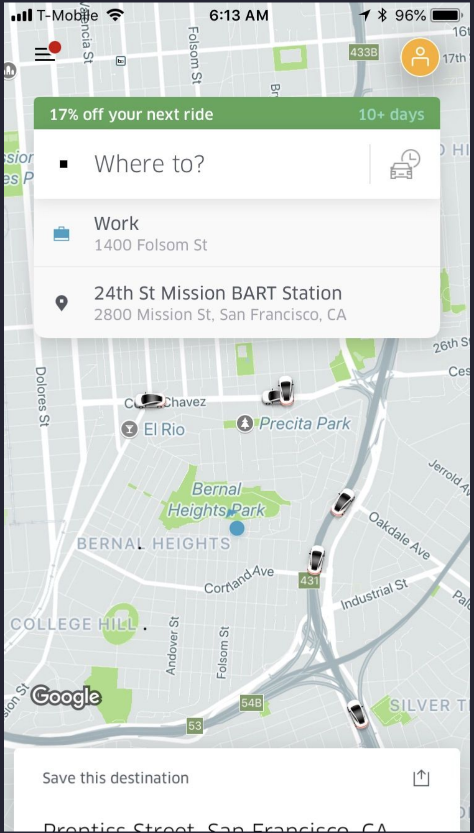
ML at Uber - Map Making

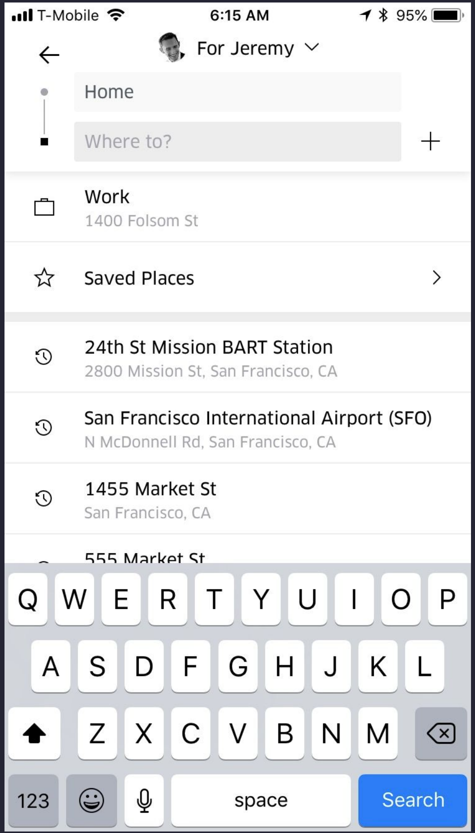


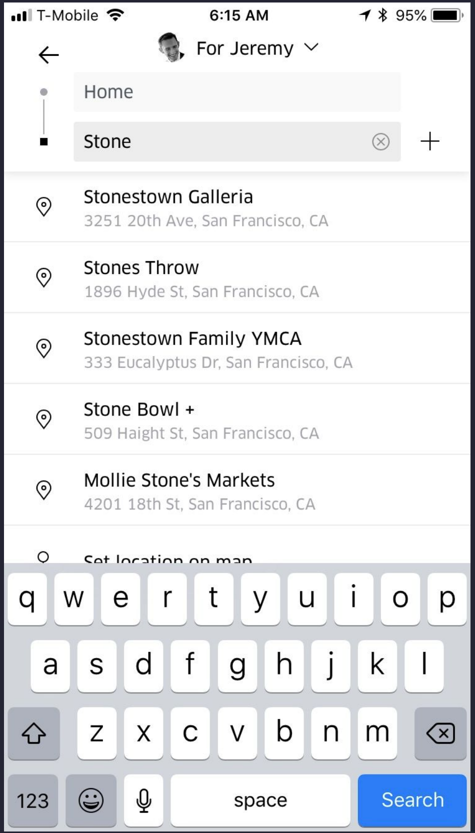
ML at Uber - Map Making

ML at Uber - Map Making

ML at Uber - Destination Prediction







ML at Uber - Spatiotemporal Forecasting

Supply

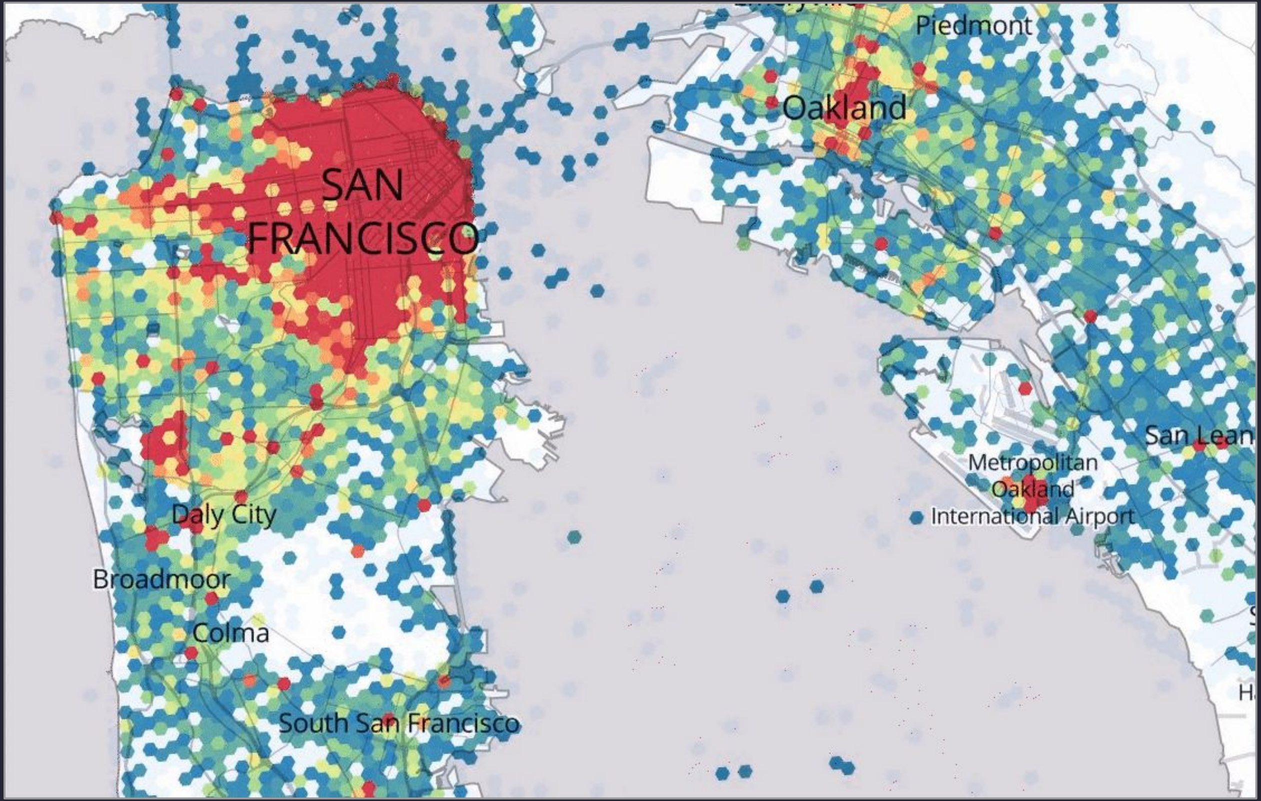
- Available Drivers

Demand

- Open Apps

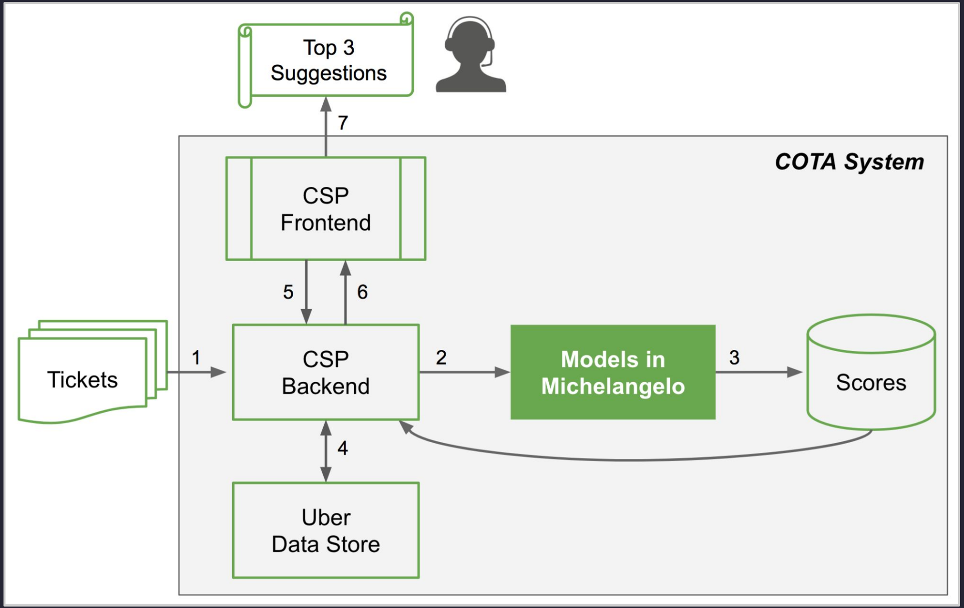
Other

- Request Times
- Arrival Times
- Airport Demand



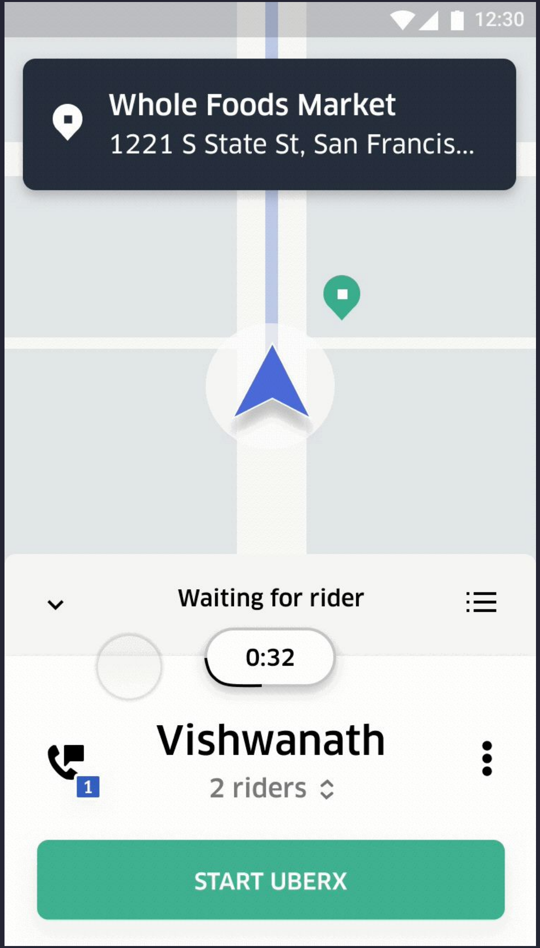
ML at Uber - Customer Support

- 5 customer-agent communication channels
- Hundreds of thousands of tickets surfacing daily on the platform across 400+ cities
- NLP models classify tickets and suggest response templates
- Reduce ticket resolution time by 10%+ with same or higher CSAT



ML at Uber - One Click Chat

- It's important for riders and driver partners to be able to communicate efficiently during pickup
- The one click chat feature streamlines communication between riders and driver-partners
- Uses natural language processing (NLP) models that predict and display the most likely replies to in-app chat messages.

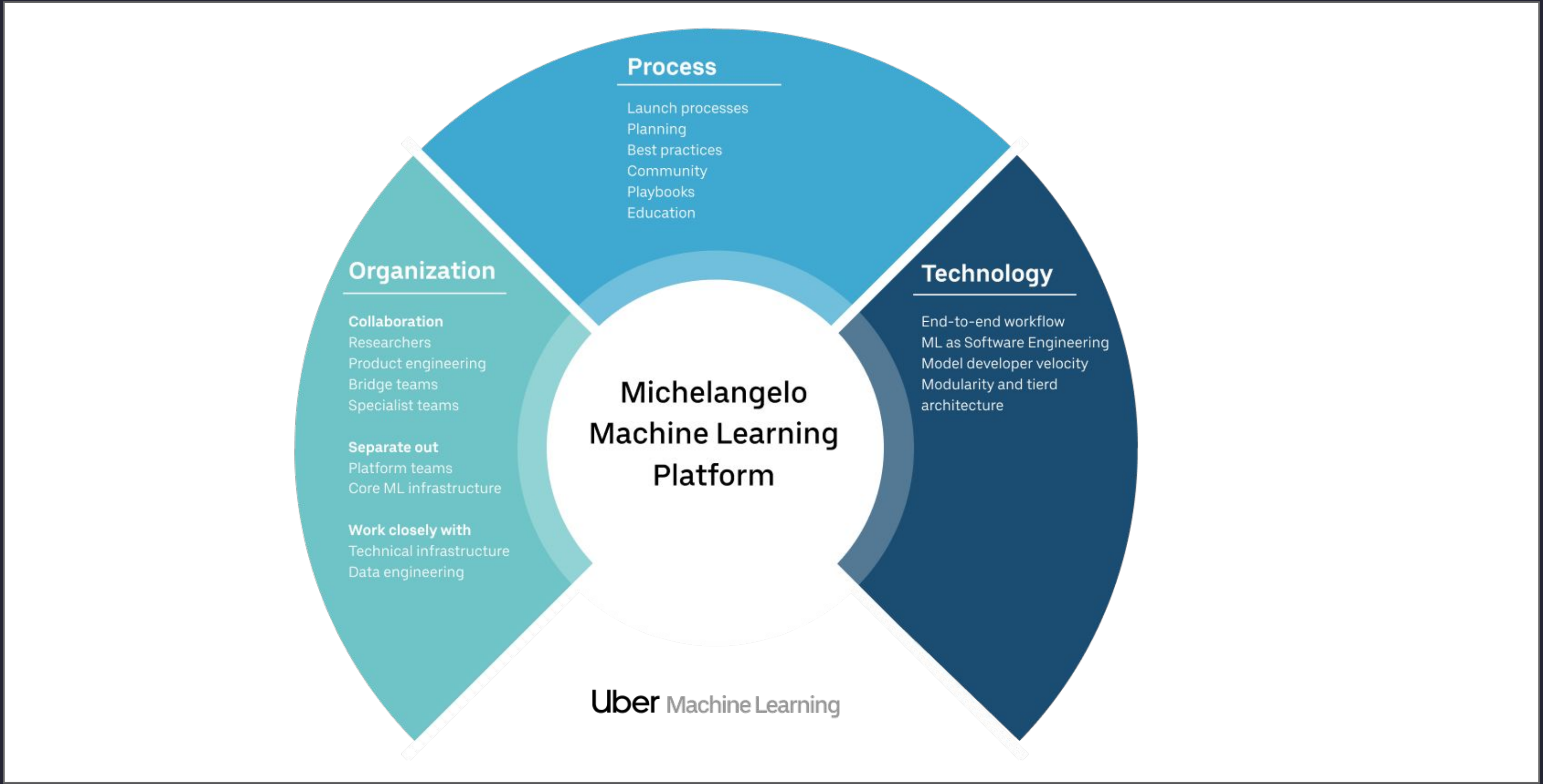


ML Platform

MISSION

Enable engineers and data scientists across the company to easily build and deploy machine learning solutions at scale.

ML Foundations - Organization, Process, and Technology



ML Platform Evolution

V1: Enable ML at Scale

- End-to-end workflow
- High scale training
- High scale model serving
- Feature Store

V2: Accelerate ML

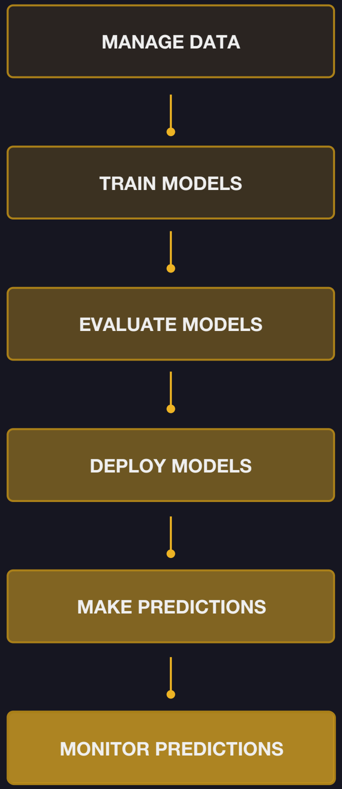
- PyML
- Horovod
- AutoTune
- Manifold Model Viz
- Realtime Model Monitoring

Enable ML at Scale

Machine Learning Workflow

Same basic ML workflow & system requirements for

- Traditional ML & Deep Learning
- Supervised, unsupervised, & semi-supervised learning
- Online learning
- Batch, online, & mobile deployments
- Time-series forecasting



Enable ML at Scale:

Manage Data

Feature Store (aka Palette)

Problem

- Hardest part of ML is finding good features
- Same features are often used by different models built by different teams

Solution

- Centralized feature store for collecting and sharing features
- Platform team curates core set of widely applicable features
- Modellers contribute more features as part of ongoing model building
- Meta-data for each feature to track ownership, how computed, where used, etc
- Modellers select features by name & join key. Offline & online pipelines are automatically deployed

Enable ML at Scale:

Train Models

Distributed Training of Non-DL Models

Large-scale distributed training (billions of samples)

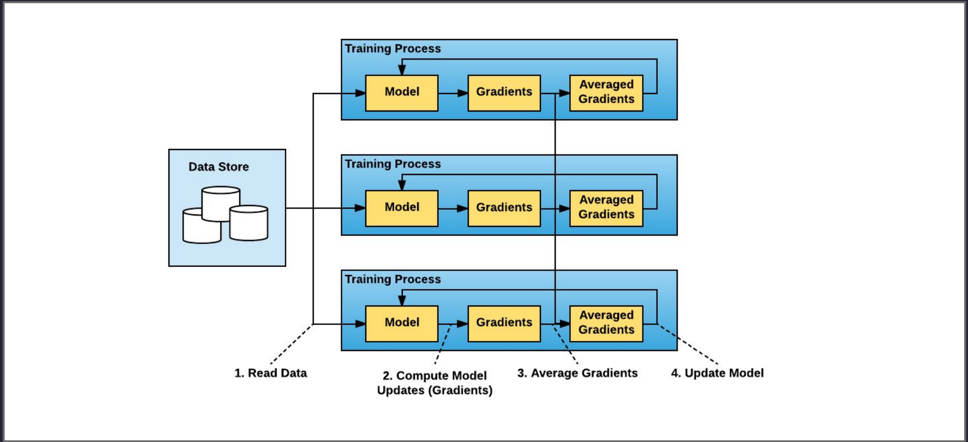
- Decision trees
- Linear and logistic models
- Unsupervised learning
- Time series forecasting
- Hyperparameter search for all model types

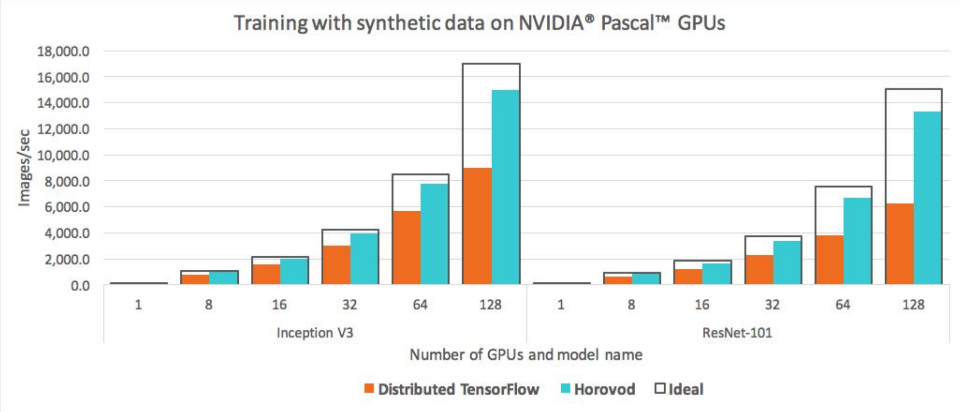
Speed and reliability

- Fuse operators into single job for speed
- Break operators into separate jobs to reliability

Distributed Training of Deep Learning Models with Horovod

- Data-parallelism works best when model is small enough to fit on each GPU
- Ring-allreduce is more efficient than parameter servers for averaging weights
- Faster training and better GPU utilization
- Much simpler training scripts
- More details at http://eng.uber.com/horovod





Enable ML at Scale:

Manage & Eval Models

Evaluate Models

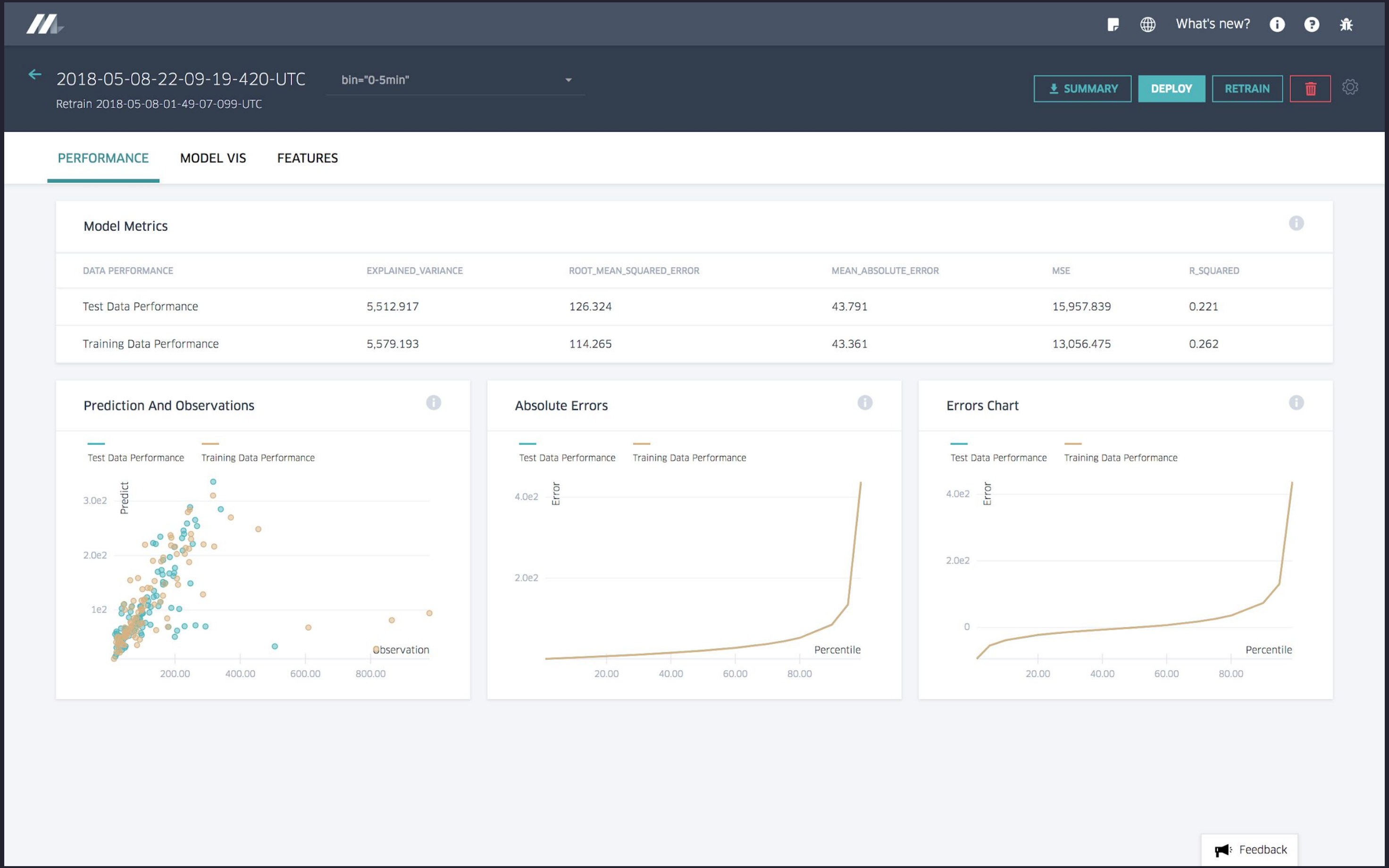
Problem

- It takes many iterations to produce a good model
- Keeping track of how a model was built is important
- Evaluating and comparing models is hard

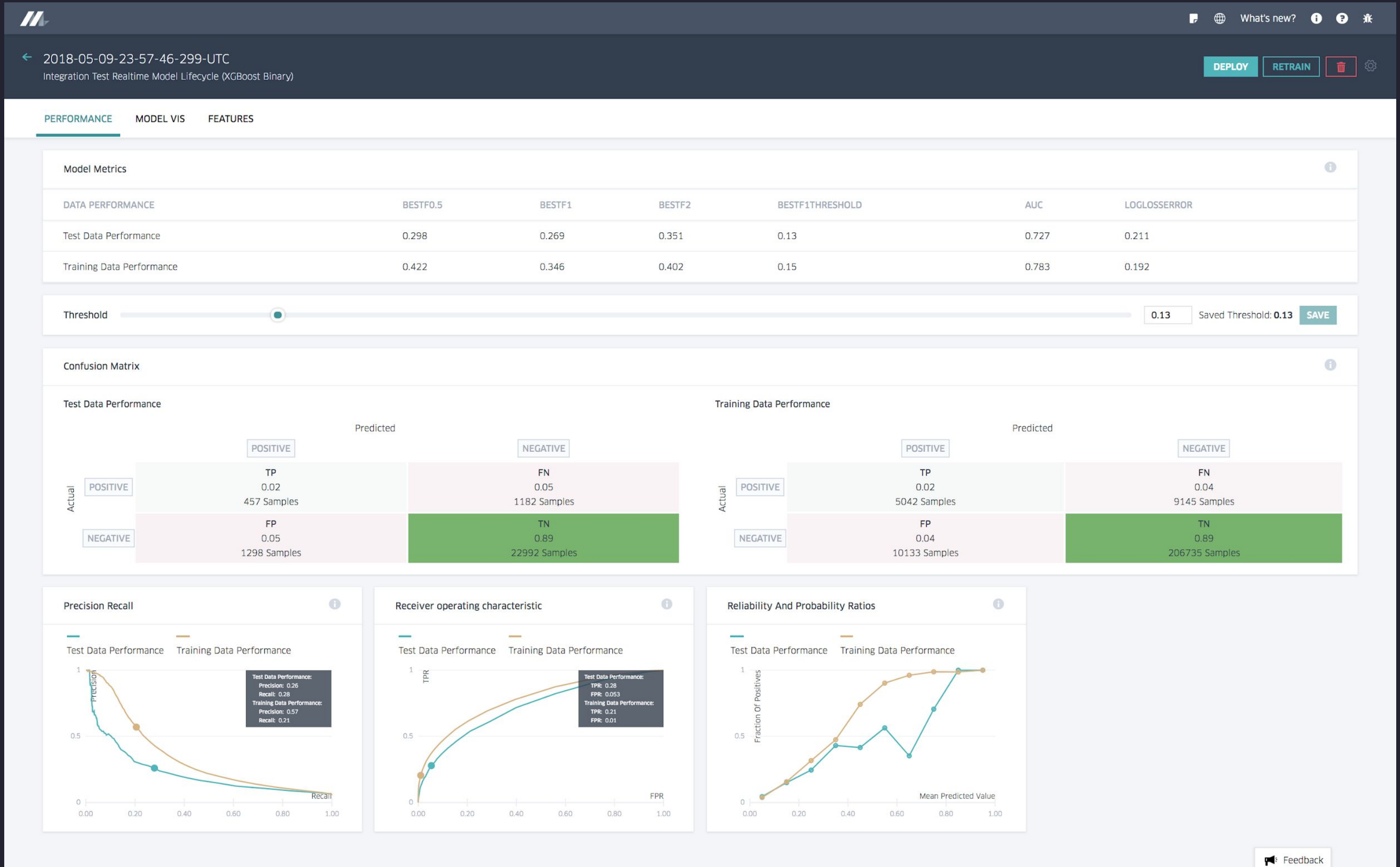
With every trained model, we capture standard metadata and reports

- Full model configuration, including train and test datasets
- Training job metrics
- Model accuracy metrics
- Performance of model after deployment

Model Visualization - Regression Model



Model Visualization - Classification Model



Model Visualization - Feature Report

What's new?

2018-05-08-22-09-19-420-UTC

Retrain 2018-05-08-01-49-07-099-UTC

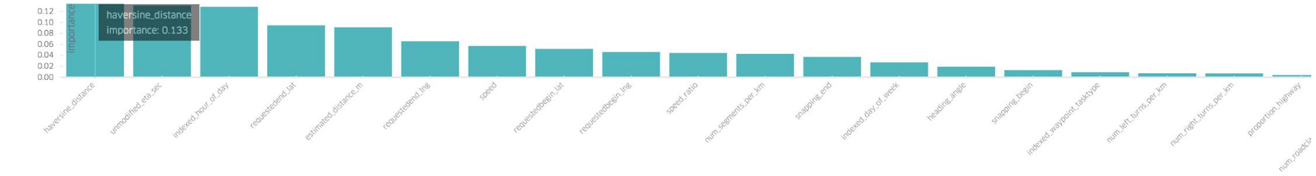
bin="5-10min"

SUMMARY DEPLOY RETRAIN

PERFORMANCE MODEL VIS FEATURES

Overview

All features by importance



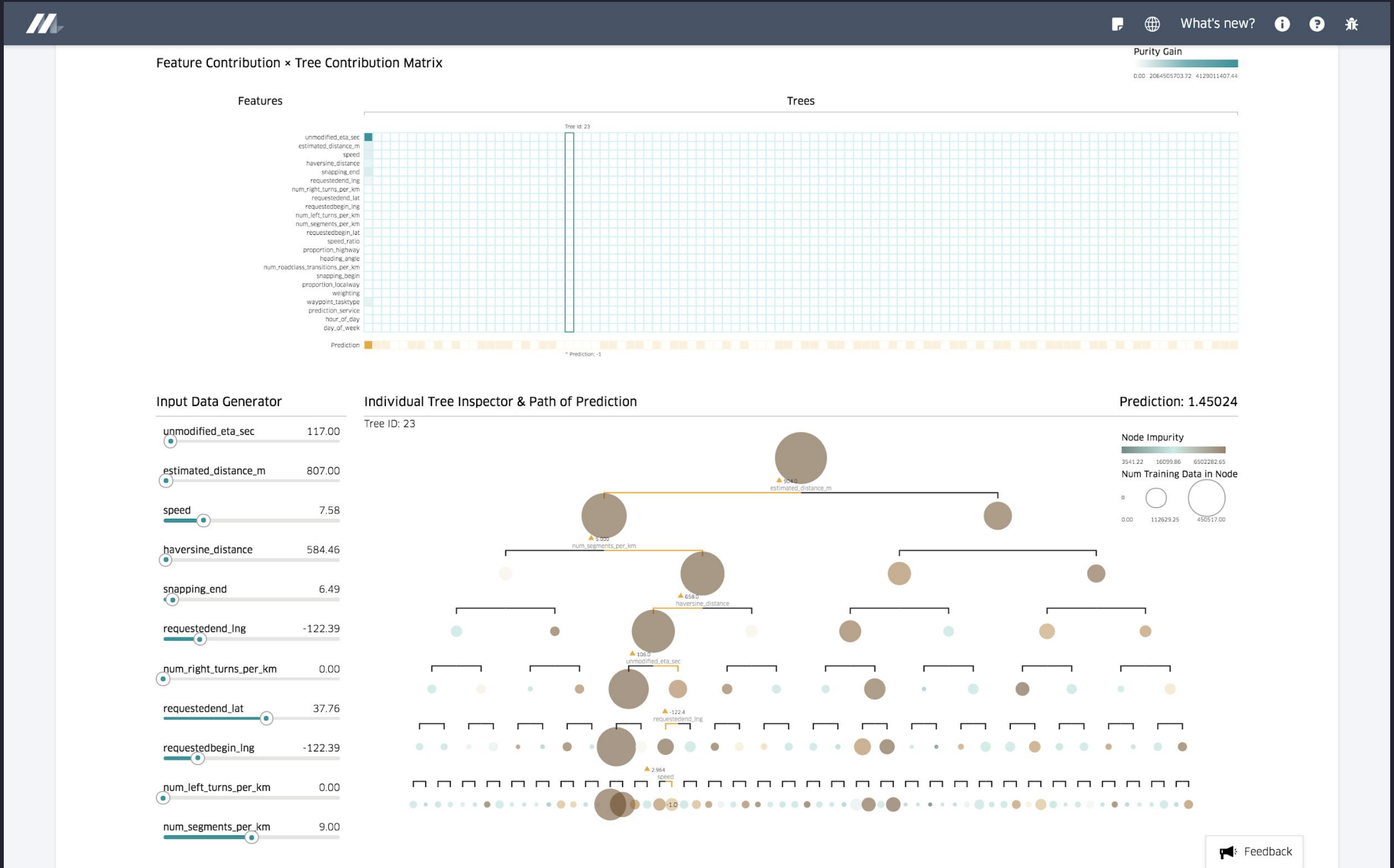
Feature Details

Numeric (18) Categorical (5)

EXPAND ALL | COLLAPSE ALL	VARIATION COEFF	NULLS	ZEROS	MEAN	STD	P01	P99	HISTOGRAM
speed	0.873	0	1577	9.318	8.133	0.001518	33.261	
num_left_turns_per_km	1.817	0	1.6955e+5	1.045	1.899	0	9	
snapping_end	1.526	0	0	14.679	22.394	0.059324	121.56	
num_roadclass_transitions_per_km	7.185	0	2.7247e+5	0.099	0.713	0	4	
num_segments_per_km	0.368	0	0	8.546	3.143	2	16	
speed_ratio	0.819	0	1051	1.112	0.911	0.0010291	4.5148	
heading_angle	0.541	0	0	193.13	104.41	3.9649	357.33	
proportion_localway	0.132	0	1227	0.971	0.128	0.22568	1	
proportion_highway	4.364	0	2.5739e+5	0.029	0.128	0	0.77765	

Feedback

Model Visualization - Decision Tree



Enable ML at Scale:

Deployment & Serving

Online Prediction Service

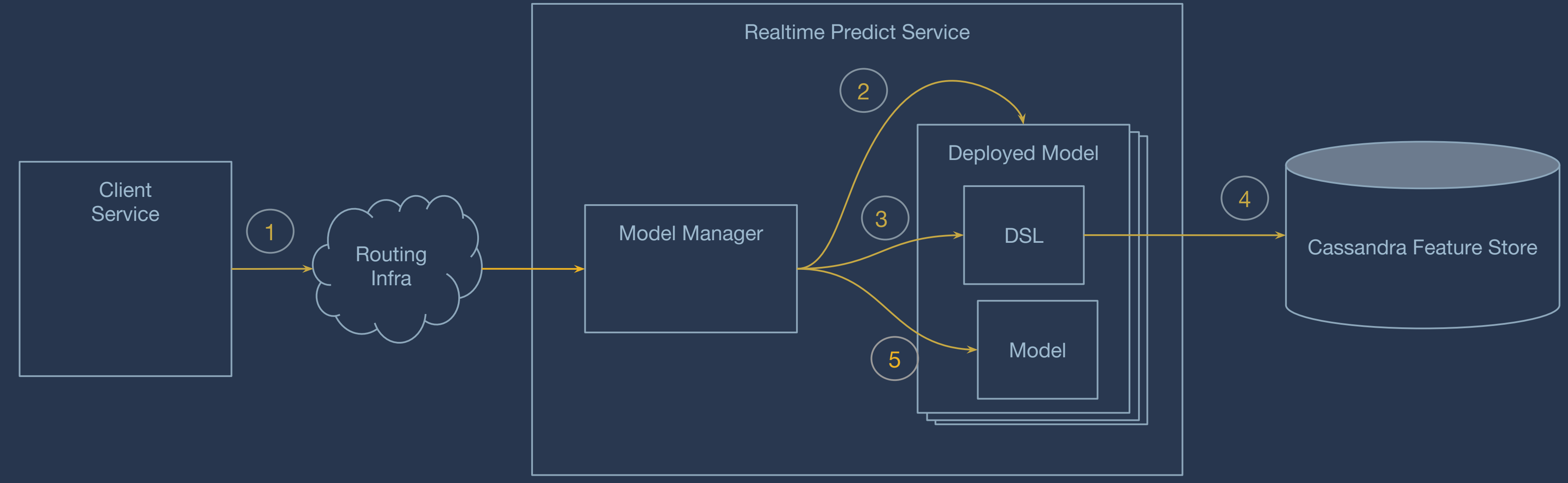
Prediction Service

- Thrift service container for one or more models
- Scale out in Docker on Mesos
- Single or multi-tenant deployments
- Connection management and batched / parallelized queries to Cassandra
- Monitoring & alerting

Deployment

- Model & DSL packaged as JAR file
- One click deploy across DCs via standard Uber deployment infrastructure
- Health checks and rollback

Online Prediction Service



Online Prediction Service

Typical p95 latency from client service

- ~5ms when all features from client service
- ~10ms when joining pre-computed features from Cassandra

Peak prediction volume across current online deployments

- 1M+ QPS

Enable ML at Scale:

Monitor Models in Production

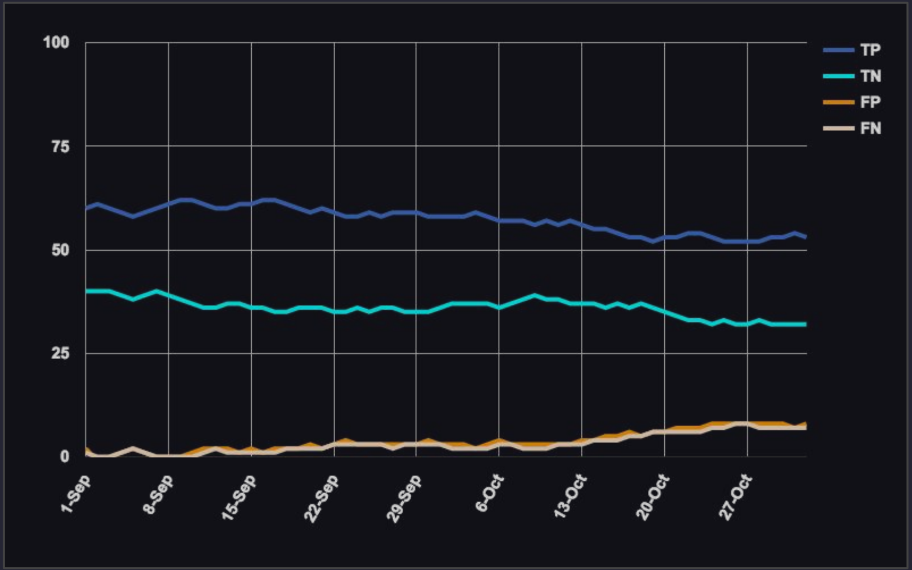
Monitor Predictions

Problem

- Models trained and evaluated against historical data
- Need to ensure deployed model is making good predictions going forward

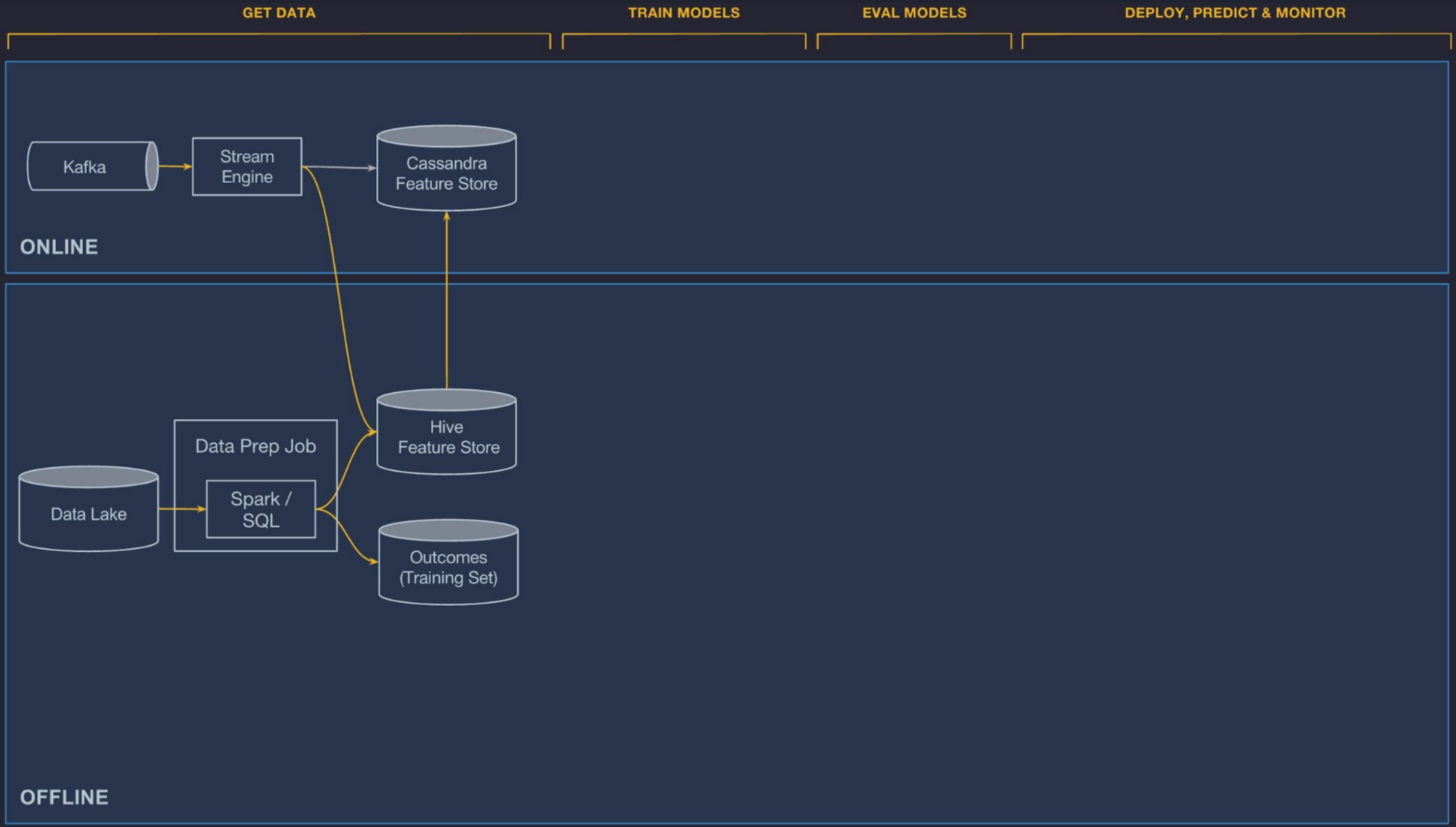
Solution

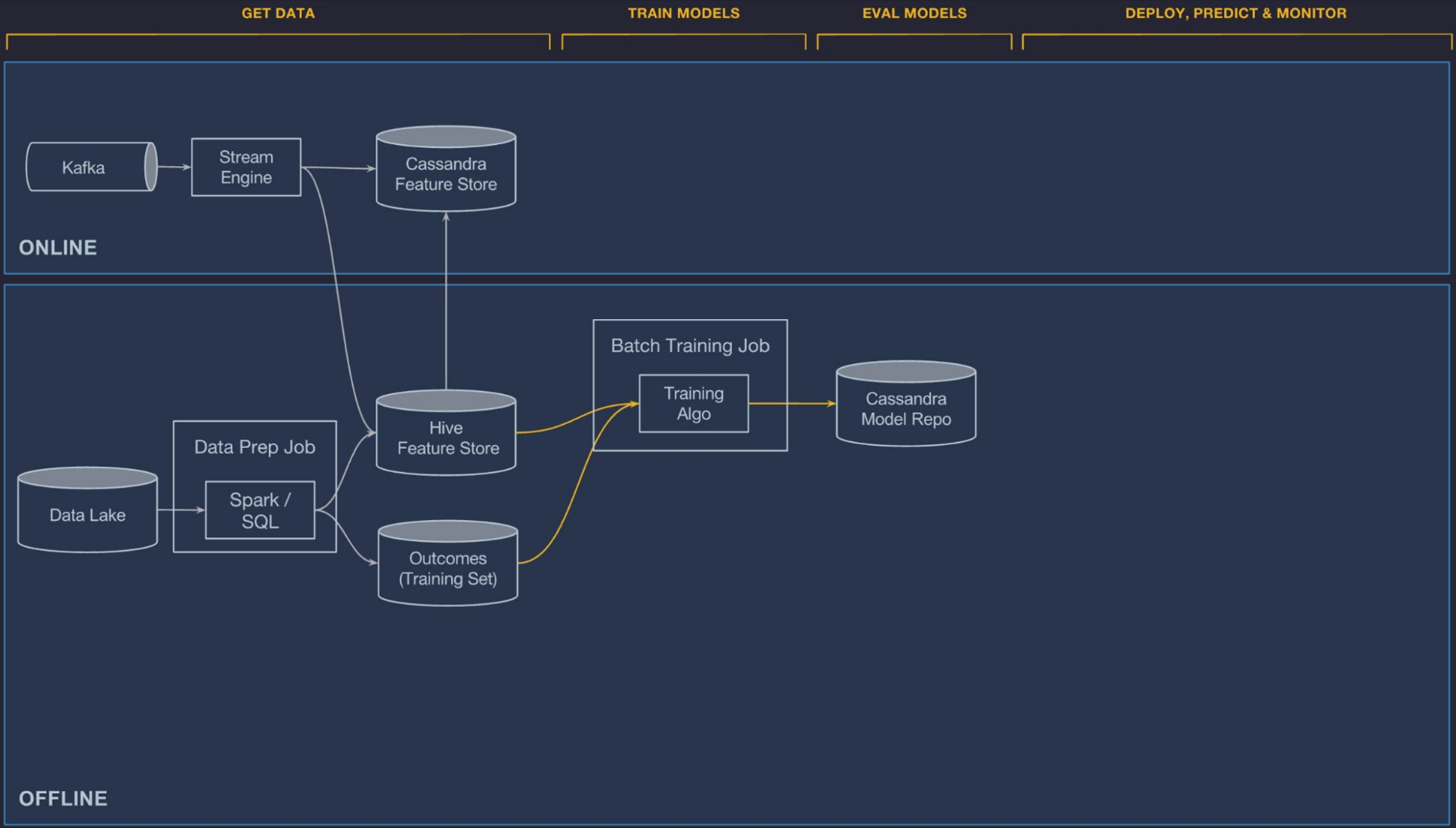
- Log predictions & join to actual outcomes
- Publish metrics feature and prediction distributions over time
- Dashboards and alerts

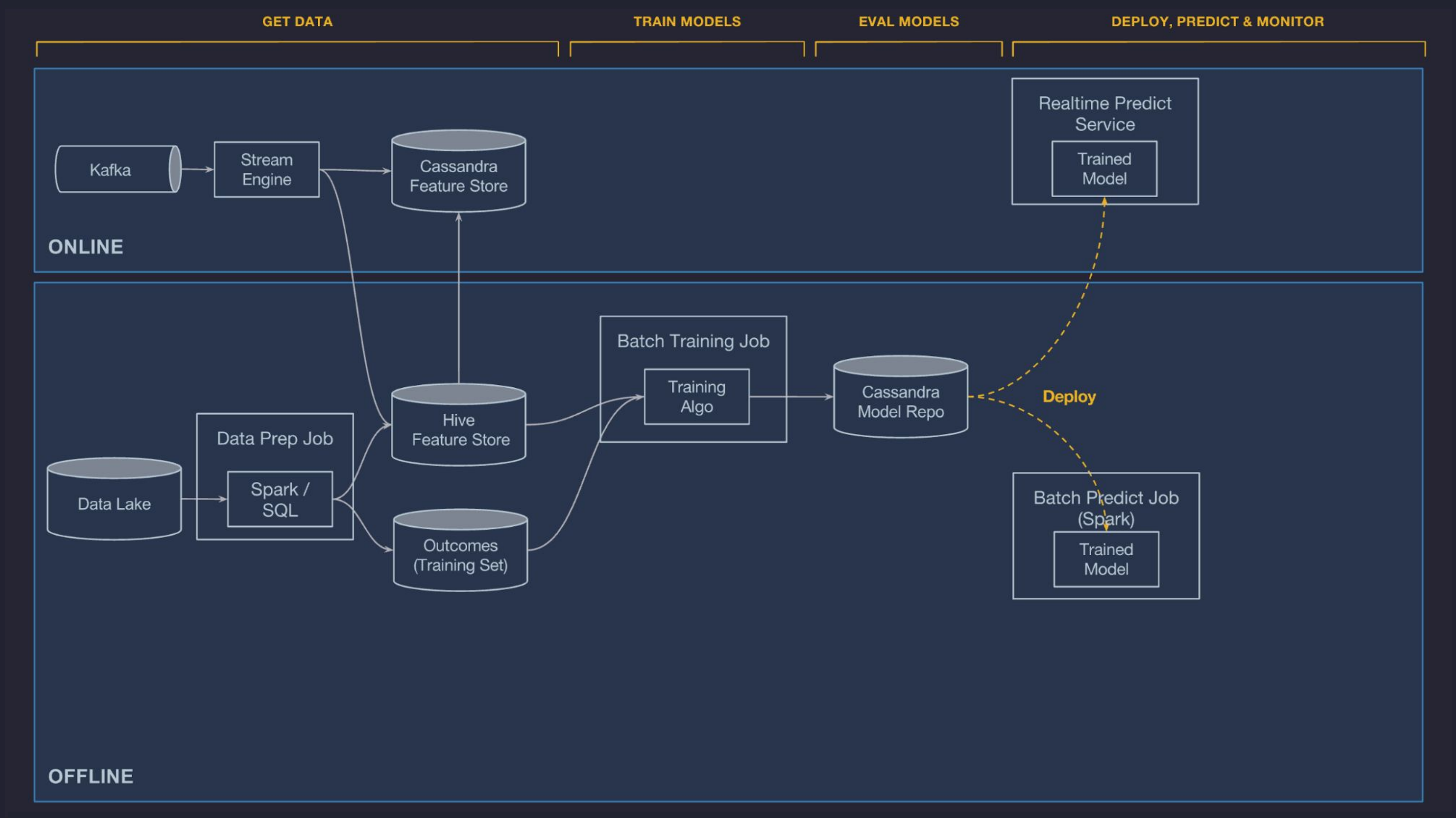


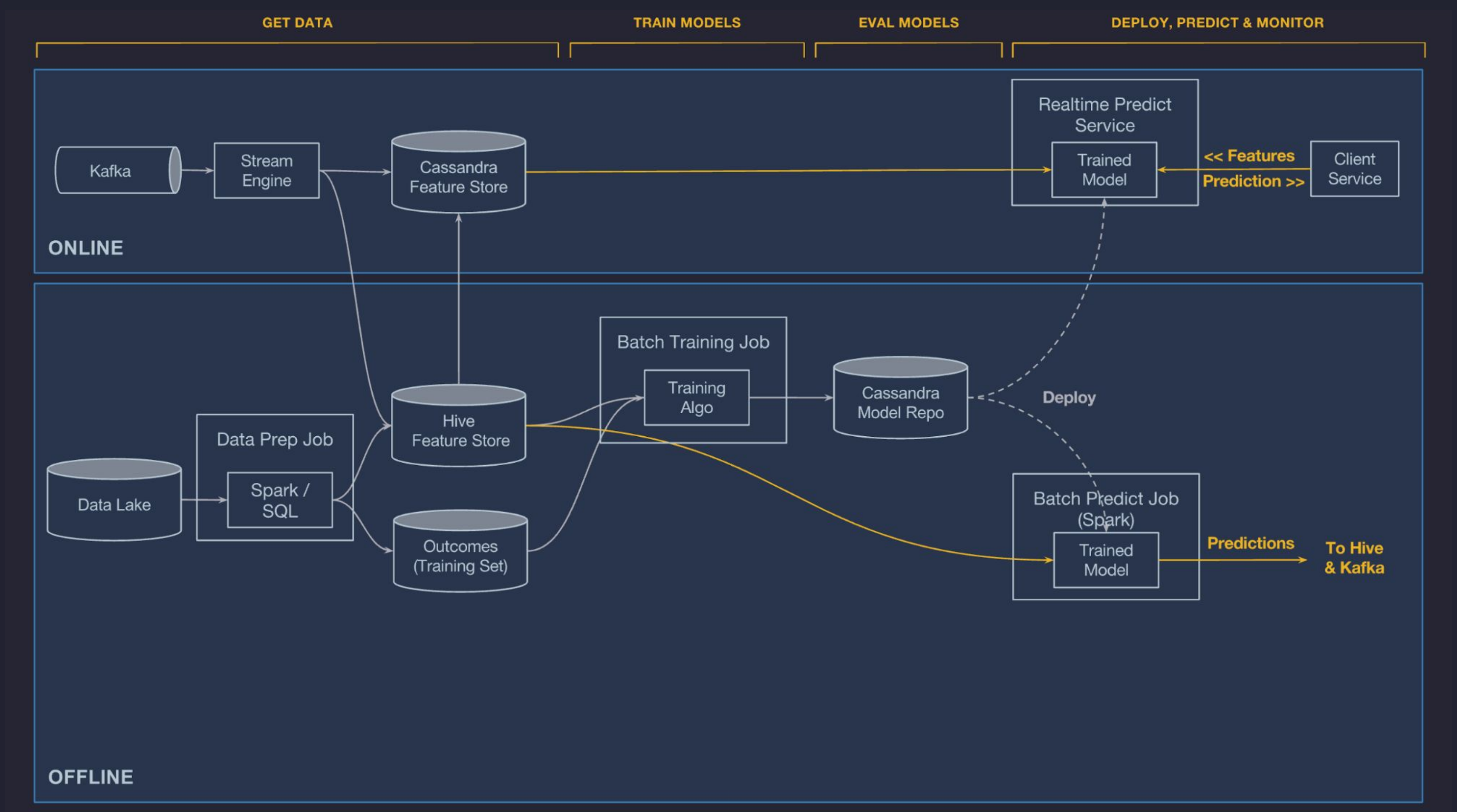
Enable ML at Scale:

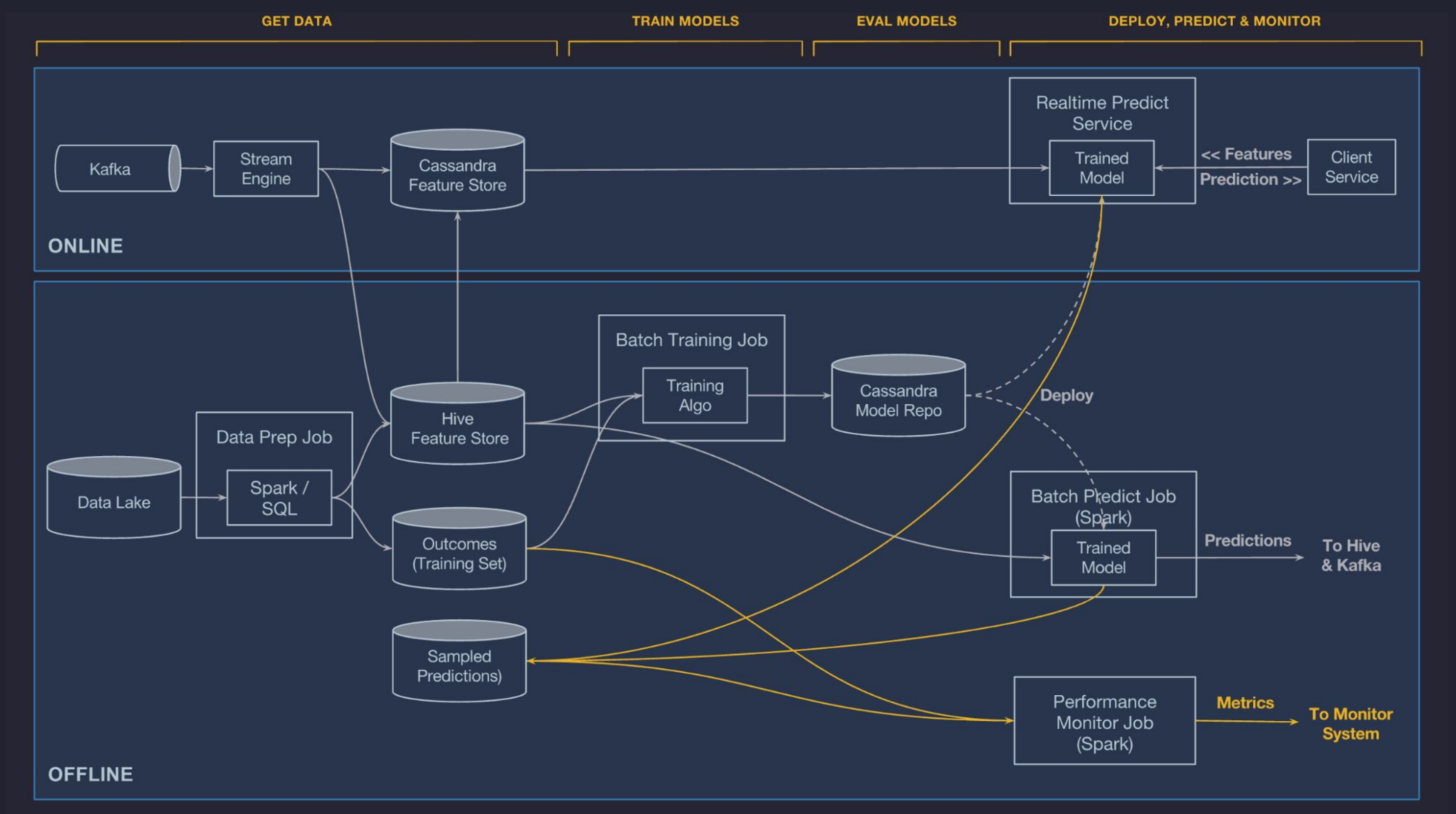
System Architecture


GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Kafka
Stream Engine
Cassandra Feature Store
ONLINE
Data Prep Job
Spark / SQL
Data Lake
Hive Feature Store
Outcomes (Training Set)
OFFLINE

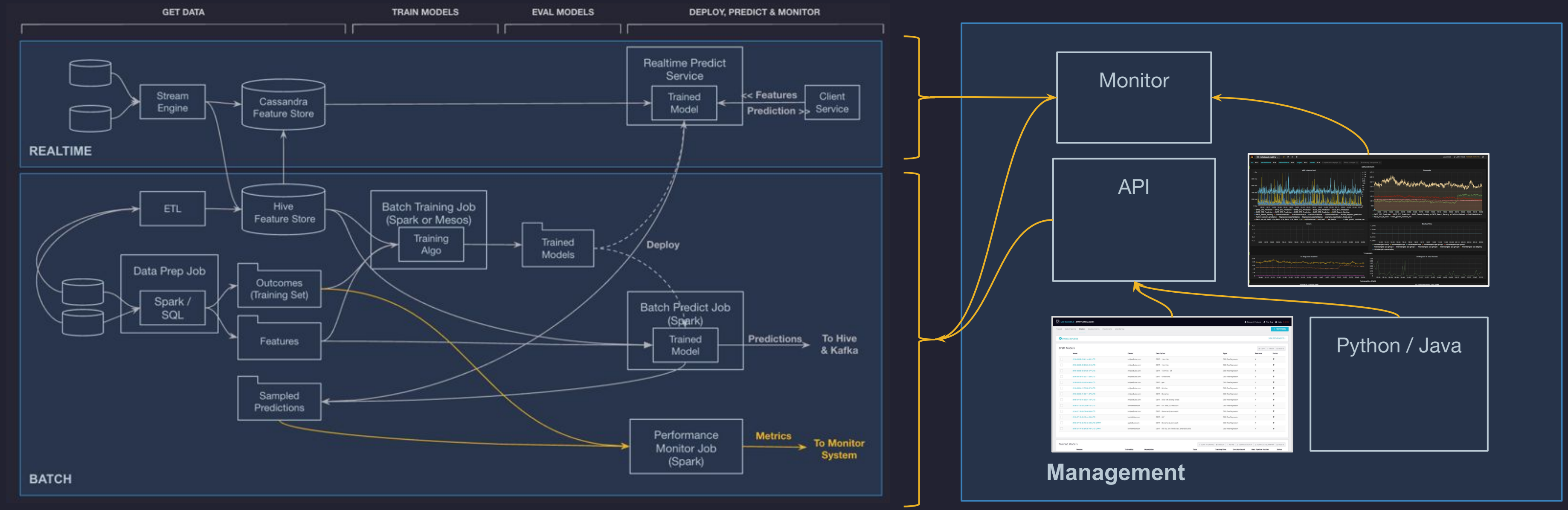

GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Kafka
Stream Engine
Cassandra Feature Store
ONLINE
Batch Training Job
Training Algo
Cassandra Model Repo
Data Prep Job
Hive Feature Store
Data Lake
Spark / SQL
Outcomes (Training Set)
OFFLINE


GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
ONLINE
Kafka
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
OFFLINE
Data Lake
Data Prep Job
Spark / SQL
Hive Feature Store
Outcomes (Training Set)
Batch Training Job
Training Algo
Cassandra Model Repo
Deploy
Batch Predict Job (Spark)
Trained Model


GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Kafka
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
<< Features
Prediction >>
Client Service
ONLINE
Batch Training Job
Training Algo
Cassandra Model Repo
Deploy
Data Prep Job
Spark / SQL
Hive Feature Store
Data Lake
Outcomes (Training Set)
Batch Predict Job (Spark)
Trained Model
Predictions
To Hive & Kafka
OFFLINE



GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
ONLINE
Kafka
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
<< Features
Prediction >>
Client Service
OFFLINE
Data Lake
Data Prep Job
Spark / SQL
Hive Feature Store
Outcomes (Training Set)
Sampled Predictions)
Batch Training Job
Training Algo
Cassandra Model Repo
Deploy
Batch Predict Job (Spark)
Trained Model
Predictions
To Hive & Kafka
Performance Monitor Job (Spark)
Metrics
To Monitor System


GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
<< Features
Prediction >>
Client Service
REALTIME
ETL
Hive Feature Store
Batch Training Job (Spark or Mesos)
Training Algo
Trained Models
Deploy
Data Prep Job
Spark / SQL
Outcomes (Training Set)
Features
Batch Predict Job (Spark)
Trained Model
Predictions
To Hive & Kafka
Sampled Predictions
Performance Monitor Job (Spark)
Metrics
To Monitor System
BATCH
Monitor
API
Python / Java
Management


WHAT'S NEW?
tm20180725-174721-ELRYCYFA-KWCCRK
DEPLOY
RETRAIN
PERFORMANCE
MODEL VIS
FEATURES
Model Vis
Features
Trees

Accelerate ML

ML Platform Evolution

V1: Enable ML at Scale

- End-to-end workflow
- High scale training
- High scale model serving
- Feature Store

V2: Accelerate ML

- PyML
- Horovod
- AutoTune
- Manifold Model Viz
- Realtime Model Monitoring

Keys to High Velocity ML

- Reduce friction at every step of complex, iterative workflow
- End-to-end ownership by modeler - no handoffs
- Bring the tools to the modeler
- Simple, elegant APIs
- Rich visual tools for understanding data and models
- Measure time from idea to model in production

Accelerate ML:

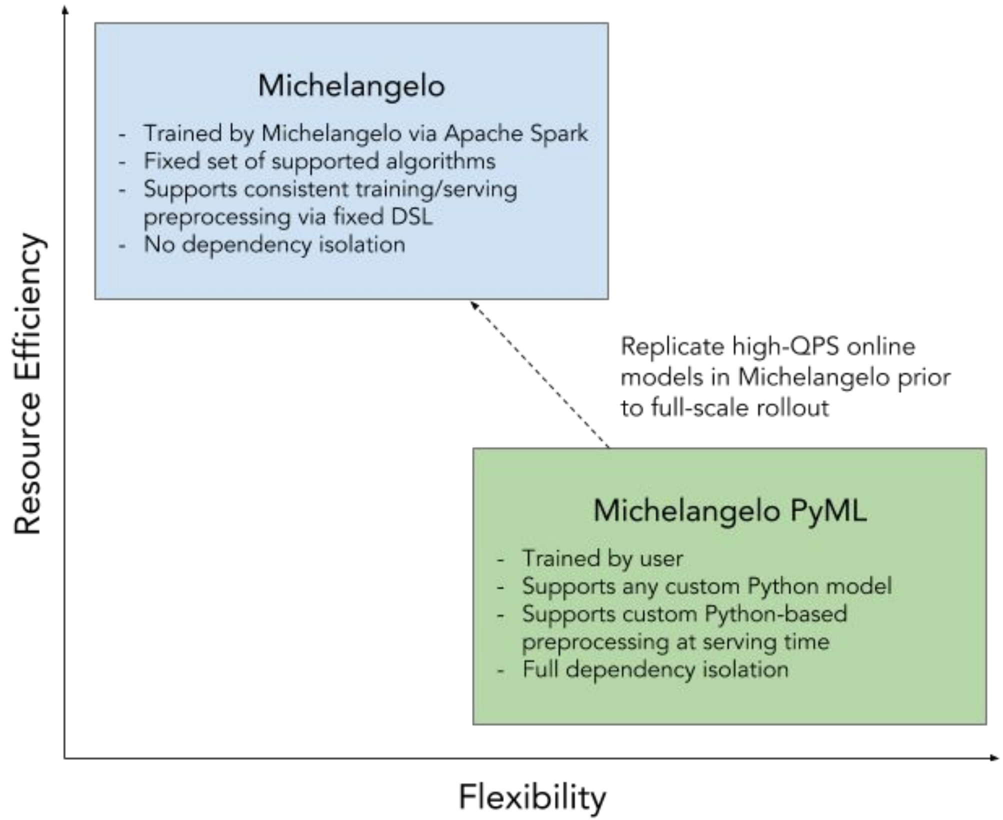
PyML

PyML

- Problem
  - Michelangelo initially targeted high scale use cases
  - Good for first wave of production use cases
  - But, not very easy for early prototyping and limited flexibility

- Solution
  - Support regular Python for lower scale, easy to use and very flexible modeling
  - Import any libraries
  - Train any model
  - Implement serving interface with predict() method
  - Call API to package model artifacts and upload to Michelangelo
  - Deploy to production via API or Michelangelo UI

PyML


Michelangelo
- Trained by Michelangelo via Apache Spark
- Fixed set of supported algorithms
- Supports consistent training/serving preprocessing via fixed DSL
- No dependency isolation
Replicate high-QPS online models in Michelangelo prior to full-scale rollout
Michelangelo PyML
- Trained by user
- Supports any custom Python model
- Supports custom Python-based preprocessing at serving time
- Full dependency isolation
Resource Efficiency
Flexibility

PyML - 1. Train and Test Model

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer

# Prepare the dataset
dataset = load_breast_cancer()
feature_columns = [name.replace(' ','_') for name in dataset.feature_names.tolist()]
pandas_df = pd.DataFrame(data= np.c_[dataset.data, dataset.target],
                     columns=feature_columns + ['target'])

# Train logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dataset.data,
                                                    dataset.target,
                                                    random_state=42)
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Test that model works on first few records
log_reg.predict_proba(pandas_df[feature_columns])[:10,0]
```

```
array([ 1.        ,  0.99999995,  0.99999994,  0.41742828,  0.99998692,
        0.76040329,  0.99999709,  0.97970554,  0.90590637,  0.99841058])
```

PyML - 2. Save Model

```
# Create prediction_model folder
!mkdir -p prediction_model

# Export the model weights
from sklearn.externals import joblib
joblib.dump(log_reg, 'prediction_model/weights.pkl')

# Export feature columns
import pickle
pickle.dump(feature_columns, open('prediction_model/feature_columns.pkl', 'wb'))
```

PyML - 3. Implement Serving Interface

```
# Create model.py inline via Jupyter command "writefile"
%%writefile prediction_model/model.py
import pandas as pd
import numpy as np
import pickle

from pyml.model.dataframe_model import DataFrameModel
from sklearn.externals import joblib

class LogisticRegressionModel(DataFrameModel):
    """A DataFrameModel takes input as a pandas DataFrame and produces predictions as a Pandas
Dataframe.
    """
    def __init__(self):
        super(LogisticRegressionModel, self).__init__()

        # Load the model weights and feature columns
        self.clf = joblib.load('weights.pkl')
        self.feature_columns = pickle.load(open('feature_columns.pkl', 'rb'))

    def predict(self, df):
        df['probability'] = self.clf.predict_proba(
            df[self.feature_columns])[:,0]
        return df
```

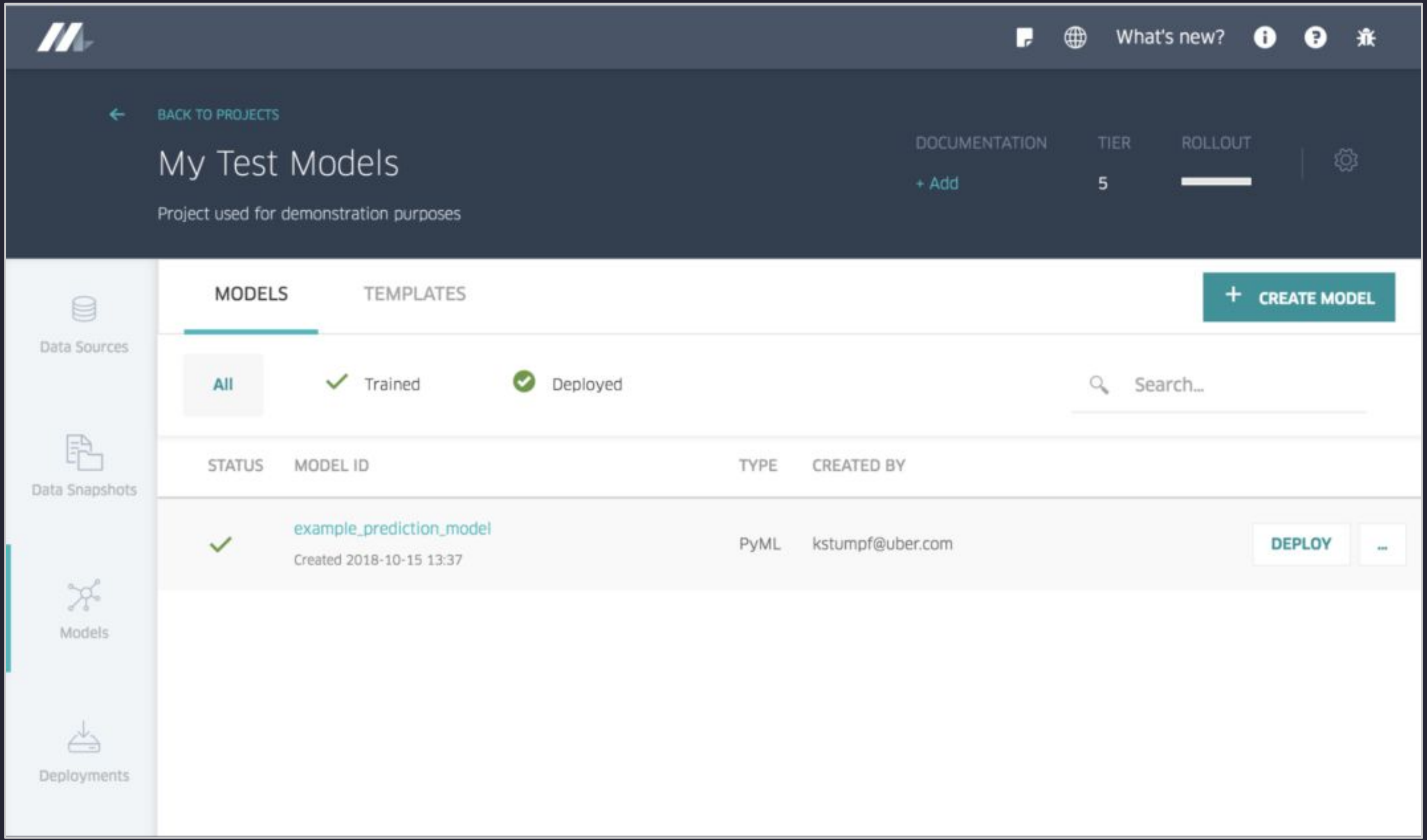
PyML - 4. Package and Upload Model

```
from pyml import PyMLModel

pyml_model = PyMLModel(model_path="prediction_model/", model_name=example_prediction_model)
pyml_model.predict(pandas_df)[:2]['probability']
```

```
0    0.999995
1    0.999996
```

```
from pyml import Client
client = Client(user_email="kstumpf@uber.com", team_name="michelangelo")

# Upload the model and build the model's Docker image
model_id = client.upload_model(pyml_model)
```

PyML - 5. See Model in Michelangelo UI



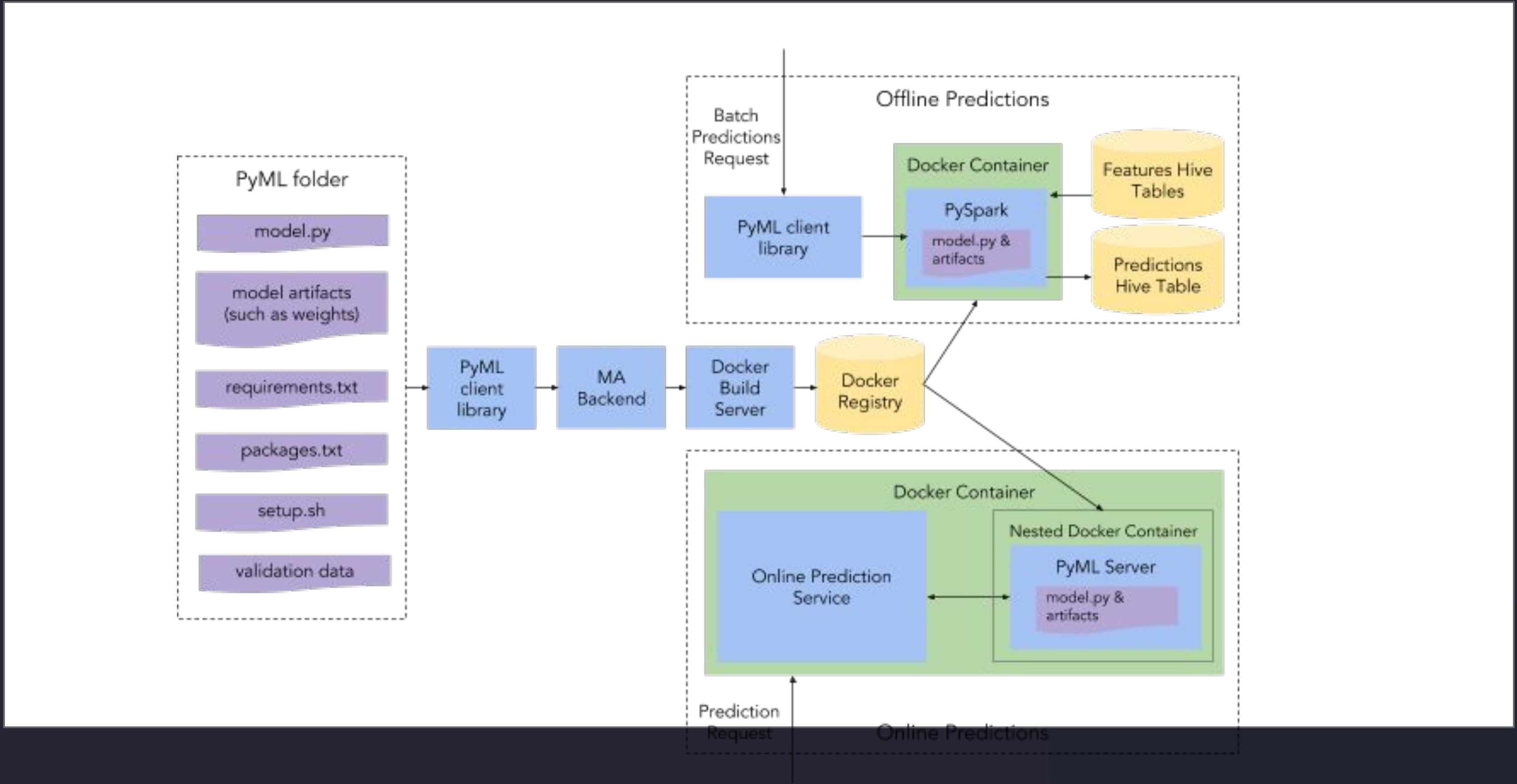
PyML - 6. Deploy and Test Model

```
client.deploy_model(model_id)
```

```
from pyml import OnlineClient

online_client = OnlineClient(model_id=model_id)
output_df = online_client.predict(pandas_df[:2])
print output_df[['target','probability']]
```

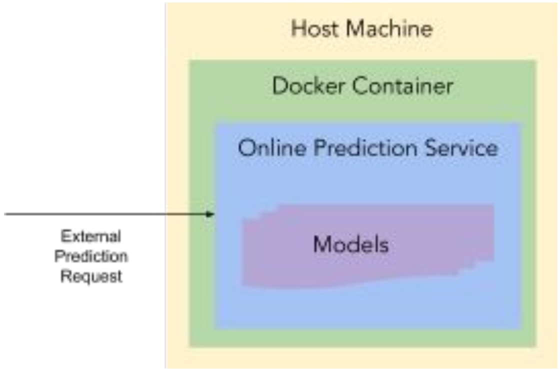
```
   target  probability
0     0.0     0.999996
1     0.0     0.999995
```

PyML - Architecture

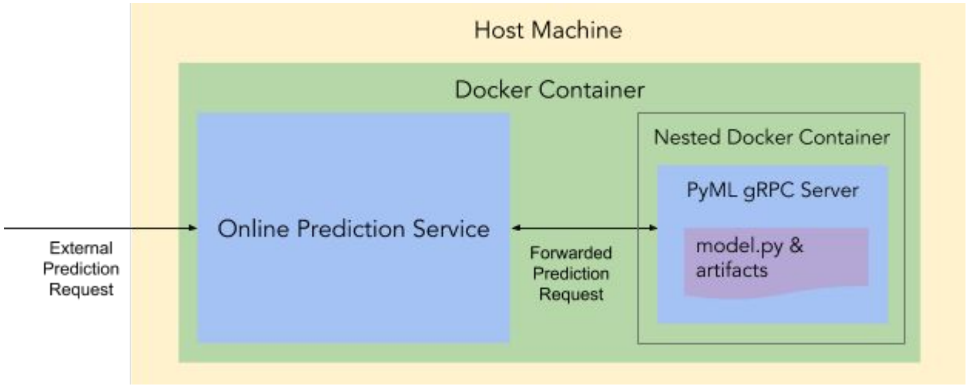


PyML - Serving Architecture

Standard Model



PyML Model



Accelerate ML:

Horovod

Horovod - Intro

- There are many ways to do data-parallel training.
- Some are more confusing than others. UX varies greatly.

- Our goals:
  - Infrastructure people deal with choosing servers, network gear, container environment, default containers, and tuning distributed training performance.
  - ML engineers focus on making great models that improve business using deep learning frameworks that they love.

Horovod - Complex Parameter Server Setup

```python
import argparse
import sys

import tensorflow as tf

FLAGS = None

def main(_):
  ps_hosts = FLAGS.ps_hosts.split(",")
  worker_hosts = FLAGS.worker_hosts.split(",")

  # Create a cluster from the parameter server and worker hosts.
  cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

  # Create and start a server for the local task.
  server = tf.train.Server(cluster,
                           job_name=FLAGS.job_name,
                           task_index=FLAGS.task_index)

  if FLAGS.job_name == "ps":
    server.join()
  elif FLAGS.job_name == "worker":

    # Assigns ops to the local worker by default.
    with tf.device(tf.train.replica_device_setter(
        worker_device="/job:worker/task:%d" % FLAGS.task_index,
        cluster=cluster)):

      # Build model...
      loss = ...
      global_step = tf.contrib.framework.get_or_create_global_step()

      train_op = tf.train.AdagradOptimizer(0.01).minimize(
          loss, global_step=global_step)

    # The StopAtStepHook handles stopping after running given steps.
    hooks=[tf.train.StopAtStepHook(last_step=1000000)]

    # The MonitoredTrainingSession takes care of session initialization,
    # restoring from a checkpoint, saving to a checkpoint, and closing when done
    # or an error occurs.
    with tf.train.MonitoredTrainingSession(master=server.target,
                                           is_chief=(FLAGS.task_index == 0),
                                           checkpoint_dir="/tmp/train_logs",
                                           hooks=hooks) as mon_sess:
      while not mon_sess.should_stop():
        # Run a training step asynchronously.
        # See `tf.train.SyncReplicasOptimizer` for additional details on how to
        # perform *synchronous* training.
        # mon_sess.run handles AbortedError in case of preempted PS.
        mon_sess.run(train_op)

if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.register("type", "bool", lambda v: v.lower() == "true")
  # Flags for defining the tf.train.ClusterSpec
  parser.add_argument(
      "--ps_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--worker_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--job_name",
      type=str,
      default="",
      help="One of 'ps', 'worker'"
  )
  # Flags for defining the tf.train.Server
  parser.add_argument(
      "--task_index",
      type=int,
      default=0,
      help="Index of task within the job"
  )
  FLAGS, unparsed = parser.parse_known_args()
```

Image Source: TensorFlow
-- https://www.tensorflow.org/deploy/distributed

Horovod - Simple Horovod Setup

```
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                       config=config, hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Initialize Horovod

Assign a GPU to each TensorFlow process

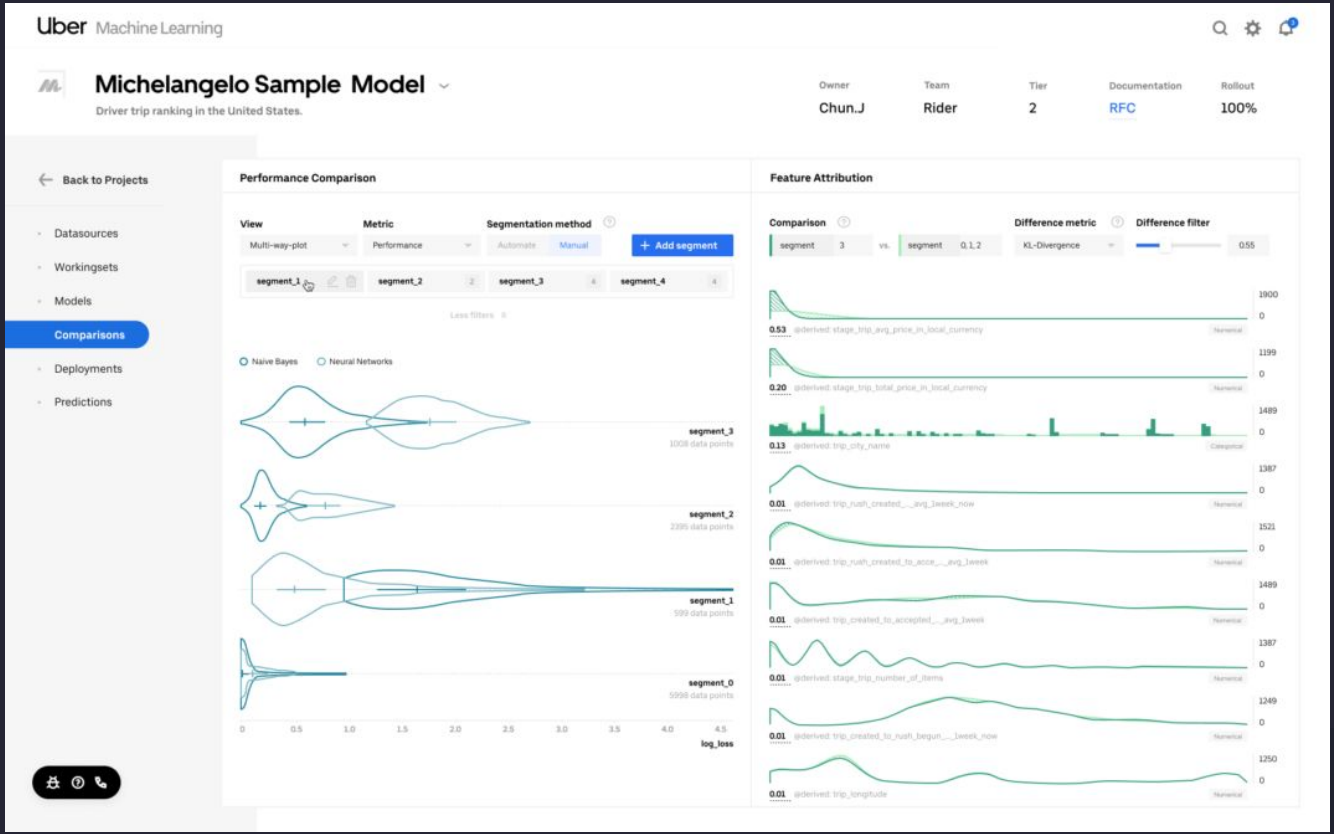
Wrap regular TensorFlow optimizer with Horovod optimizer which takes care of averaging gradients using ring-allreduce

Broadcast variables from the first process to all other processes to ensure consistent initialization

Accelerate ML:

Manifold Viz

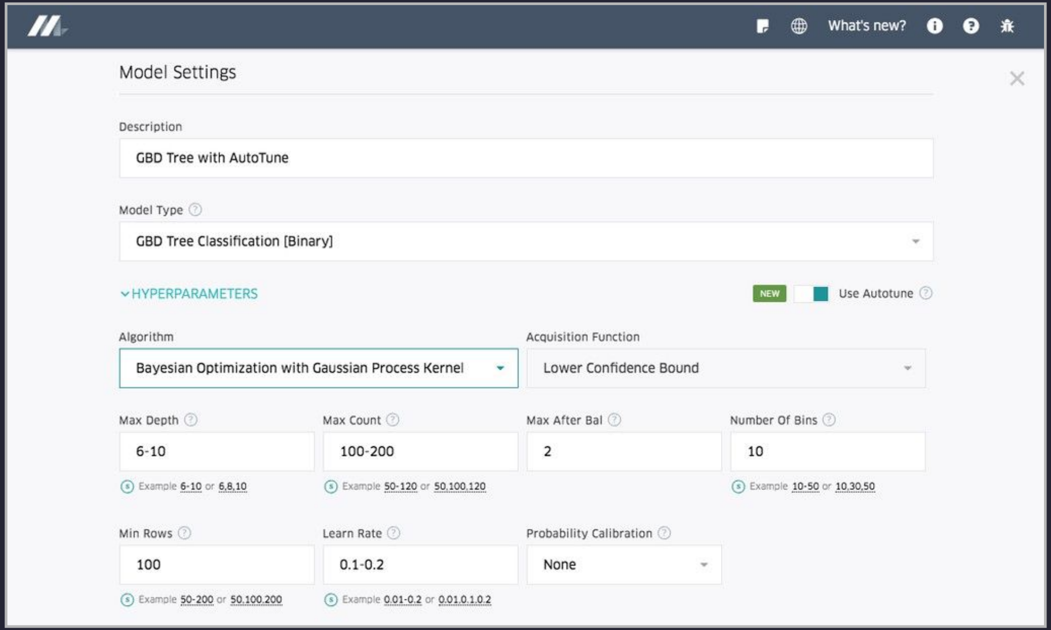
Manifold Viz



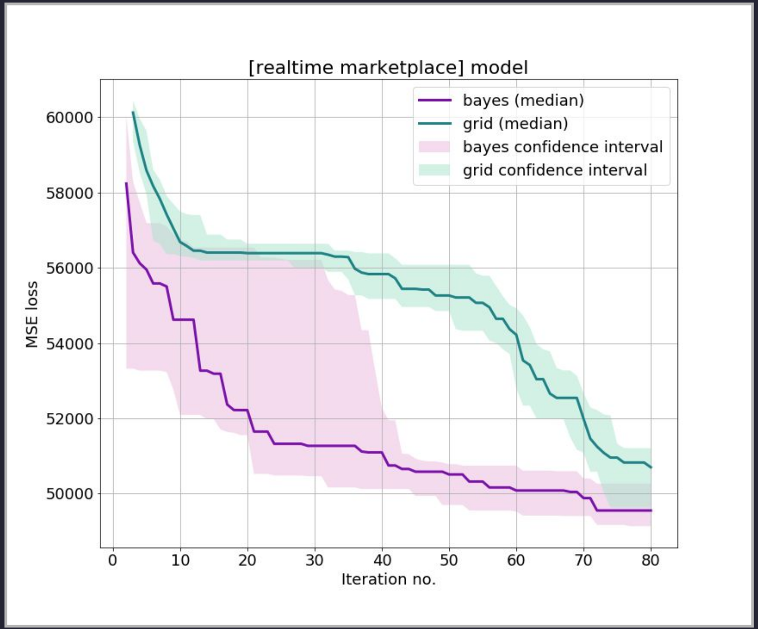
Accelerate ML:

AutoTune

AutoTune



What's new?
Model Settings
Description
GBD Tree with AutoTune
Model Type
GBD Tree Classification [Binary]
HYPERPARAMETERS
NEW
Use Autotune
Algorithm
Bayesian Optimization with Gaussian Process Kernel
Acquisition Function
Lower Confidence Bound
Max Depth
6-10
Example 6-10 or 6,8,10
Max Count
100-200
Example 50-120 or 50,100,120
Max After Bal
2
Number Of Bins
10
Example 10-50 or 10,30,50
Min Rows
100
Example 50-200 or 50,100,200
Learn Rate
0.1-0.2
Example 0.01-0.2 or 0.01,0.1,0.2
Probability Calibration
None



[realtime marketplace] model
bayes (median)
grid (median)
bayes confidence interval
grid confidence interval
MSE loss
60000
58000
56000
54000
52000
50000
0
10
20
30
40
50
60
70
80
Iteration no.

Key Lessons Learned

Key Lessons Learned

- Let developers use the tools that they want
- Data is the hardest part of ML and the most important piece to get right
- It can take significant effort to make open source and commercial components work well at scale.
- Develop iteratively based on user feedback, with the long-term vision in mind
- Real-time ML is challenging to get right

Thank you!

eng.uber.com/michelangelo
eng.uber.com/horovod
uber.com/careers