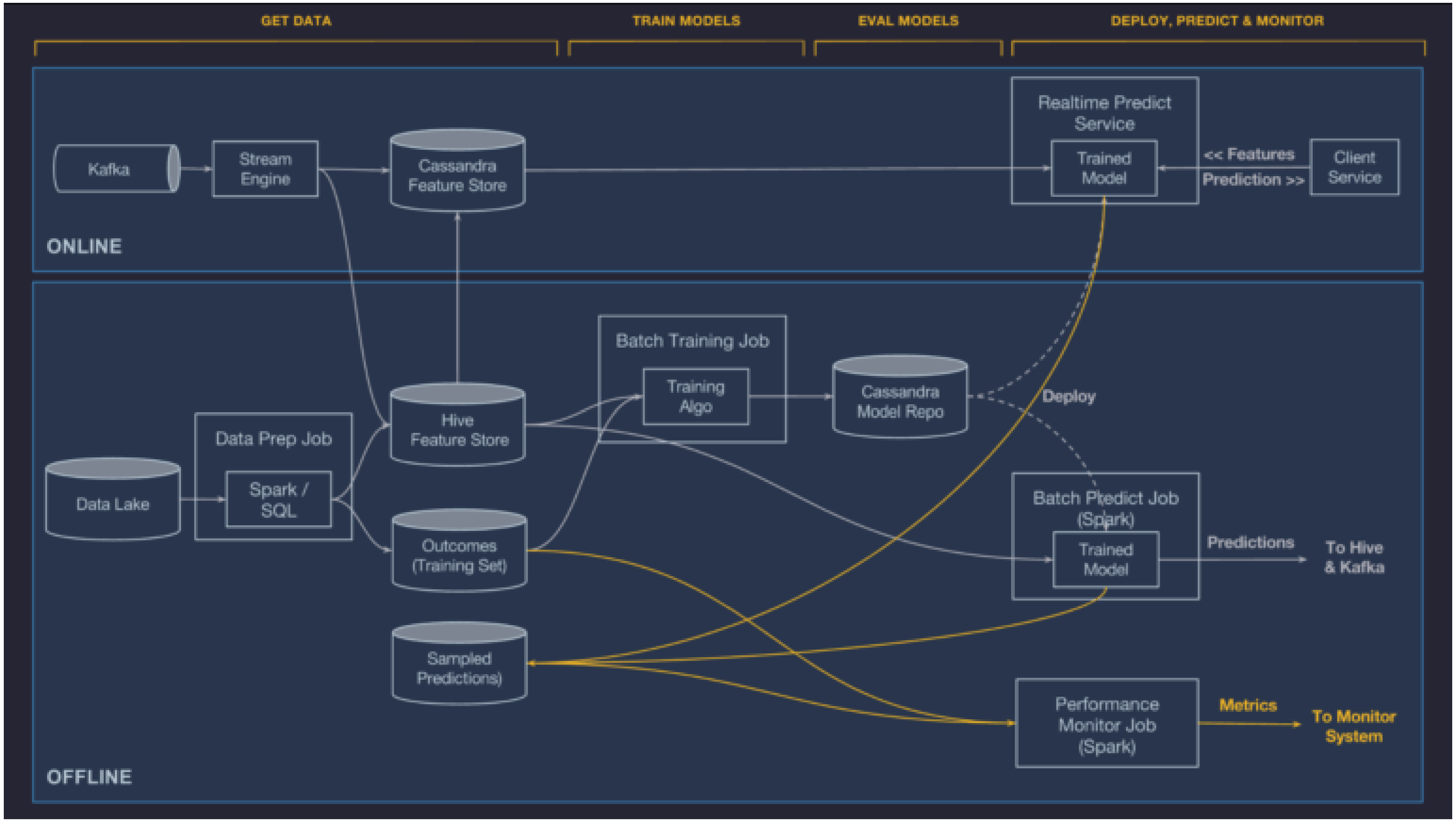
Best Practices for Deep Learning

To support a viable business using Deep Learning, you absolutely need an architecture that supports sustainable improvement in the presence of frequent and unexpected changes in the environment.

“there were no systems in place to build reliable, uniform, and reproducible pipelines for creating and managing training and prediction data at scale.”

--Uber



The architecture supports the following workflow:

- Manage data
- Train models
- Evaluate models
- Deploy, predict and monitor

The Uber system is not a strictly Deep Learning system, but rather a Machine Learning system that can employ many ML methods depending on suitability. It is built on the following open source components: HDFS, Spark, Samza, Cassandra, MLLib, XGBoost, and TensorFlow. <u>So, it’s a conventional BigData system that incorporates Machine Learning components for its analytics</u>

At the moment, we have approximately 10,000 features in Feature Store that are used to accelerate machine learning projects, and teams across the company are adding new ones all the time. Features in the Feature Store are automatically calculated and updated daily.

The model information contains:

- Who trained the model
- Start and end time of the training job
- Full model configuration (features used, hyper-parameter values, etc.)
- Reference to training and test data sets
- Distribution and relative importance of each feature
- Model accuracy metrics
- Standard charts and graphs for each model type (e.g. ROC curve, PR curve, and confusion matrix for a binary classifier)
- Full learned parameters of the model
- Summary statistics for model visualization

“

TFX: A TensorFlow-based production scale machine learning platform

”

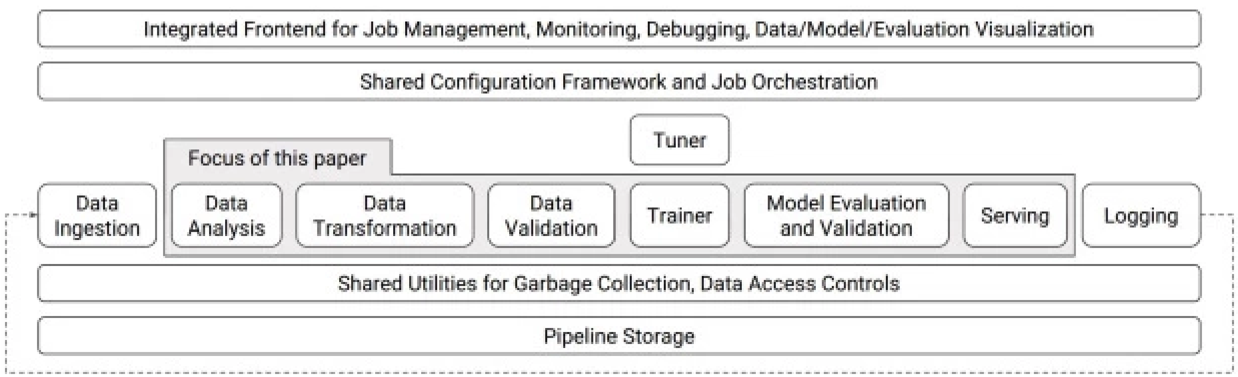


Figure 1: High-level component overview of a machine learning platform.

As compared to available open source DL frameworks, there is a greater emphasis in managing and sharing of meta-information.

Tensorflow Deploy

Distributed TensorFlow

how to create a cluster of TensorFlow servers, and how to distribute a computation graph across that cluster.

Create a cluster

```
tf.train.ClusterSpec({
    "worker": [
        "worker0.example.com:2222",
        "worker1.example.com:2222",
        "worker2.example.com:2222"
    ],
    "ps": [
        "ps0.example.com:2222",
        "ps1.example.com:2222"
    ]})
```

/job:worker/task:0

/job:worker/task:1

/job:worker/task:2

/job:ps/task:0

/job:ps/task:1

Create a server

```
# In task 0:
cluster = tf.train.ClusterSpec({"local": ["localhost:2222", "localhost:2223"]})
server = tf.train.Server(cluster, job_name="local", task_index=0)
```

```
# In task 1:
cluster = tf.train.ClusterSpec({"local": ["localhost:2222", "localhost:2223"]})
server = tf.train.Server(cluster, job_name="local", task_index=1)
```

Specifying distributed devices

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.Variable(...)
  biases_1 = tf.Variable(...)

with tf.device("/job:ps/task:1"):
  weights_2 = tf.Variable(...)
  biases_2 = tf.Variable(...)

with tf.device("/job:worker/task:7"):
  input, labels = ...
  layer_1 = tf.nn.relu(tf.matmul(input, weights_1) + biases_1)
  logits = tf.nn.relu(tf.matmul(layer_1, weights_2) + biases_2)
  # ...
  train_op = ...

with tf.Session("grpc://worker7.example.com:2222") as sess:
  for _ in range(10000):
    sess.run(train_op)
```

TensorFlow Serving

a flexible, high-performance serving system for machine learning models, designed for production environments

TensorFlow Serving 是一个用于机器学习模型 serving 的高性能开源库。它可以将训练好的机器学习模型部署到线上，使用 gRPC 作为接口接受外部调用。更加让人眼前一亮的是，它支持模型热更新与自动模型版本管理。这意味着一旦部署 TensorFlow Serving 后，你再也不需要为线上服务操心，只需要关心你的线下模型训练。

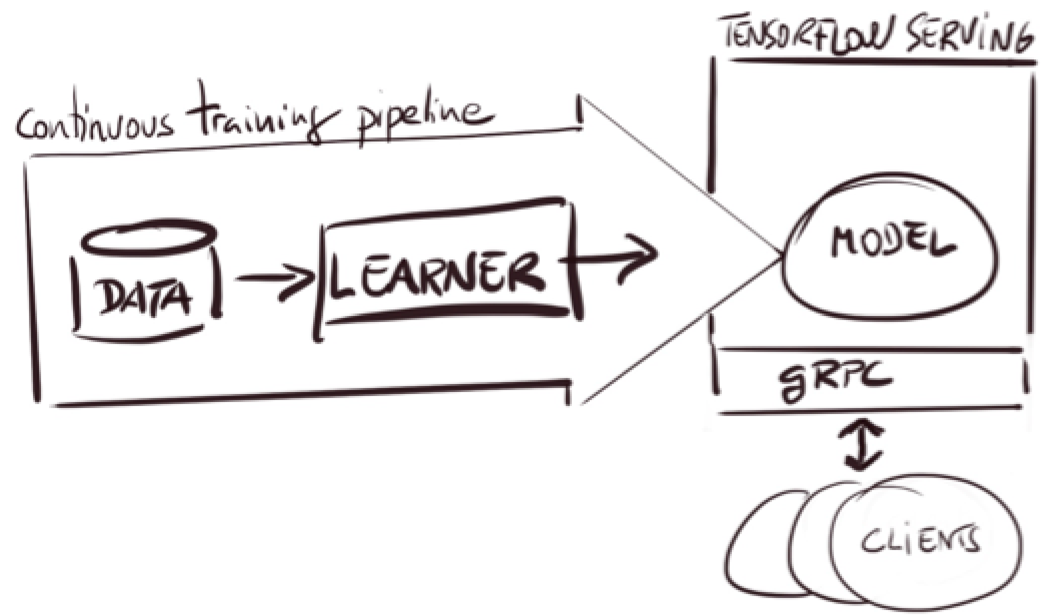
TensorFlow Serving 目前依赖 Google 的开源编译工具 Bazel。 Bazel 是 Google 内部编译工具 Blaze 的开源版本，功能与性能基本一致。具体的安装可以参考官方文档。此外还需要安装 gRPC (Google 又一个内部工具的开源版)。

```
saver.save(sess, “/tmp/test.ckpt”)
```

```
saver.restore(sess, “/tmp/test.ckpt”)
```

tf.contrib.session_bundle.exporter.Exporter

```
model_exporter = exporter.Exporter(saver)
model_exporter.init(
    sess.graph.as_graph_def(),
    named_graph_signatures={
        'inputs': exporter.generic_signature({'x': x}),
        'outputs': exporter.generic_signature({'y': y_pred}
model_exporter.export(FLAGS.work_dir,
                      tf.constant(FLAGS.export_version),ses
```



模型部署

```
$ bazel build //tensorflow_serving/model_servers:tensorflow_model_server
$ bazel-bin/tensorflow_serving/model_servers/tensorflow_model_server --port=9000 --model_name=test
--model_base_path=/tmp/test/
```

客户端

```
from grpc.beta import implementations
import numpy as np
import tensorflow as tf
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2
tf.app.flags.DEFINE_string('server', 'localhost:9000',
                           'PredictionService host:port')
FLAGS = tf.app.flags.FLAGS
n_samples = 100
host, port = FLAGS.server.split(':')
channel = implementations.insecure_channel(host, int(port))
stub = prediction_service_pb2.beta_create_PredictionService_stub(channel)
```

Generate test data

```
x_data = np.arange(n_samples, step=1, dtype=np.float32)
x_data = np.reshape(x_data, (n_samples, 1))
```

Send request

```
request = predict_pb2.PredictRequest()
request.model_spec.name = 'test'
request.inputs['x'].CopyFrom(tf.contrib.util.make_tensor_proto(x
_data, shape=[100, 1]))
result = stub.Predict(request, 10.0) # 10 secs timeout
```

配置一下 bazel 的 BUILD 文件

```
bazel build //tensorflow_serving/test:test_client && ./bazel-bin/tensorflow_serving/test/test_client
```