Bighead
Airbnb’s End-to-End Machine Learning Infrastructure

Atul Kale and Xiaohan Zeng
ML Infra @ Airbnb
Background
Airbnb’s Mission:

“Create a world where anyone can belong anywhere”
Airbnb’s Product

A global travel community that offers magical end-to-end trips, including where you stay, what you do and the people you meet.
Airbnb is already driven by Machine Learning

Search Ranking

Smart Pricing

Fraud Detection

Risk scoring

Every Airbnb reservation is scored for risk before it’s confirmed. We use predictive analytics and machine learning to instantly evaluate hundreds of signals that help us flag and investigate suspicious activity before it happens.
But there are *many* more opportunities for ML

- Paid Growth - Hosts
- Classifying / Categorizing Listings
- Experience Ranking + Personalization
- Room Type Categorizations
- Customer Service Ticket Routing
- Airbnb Plus
- Listing Photo Quality
- Object Detection - Amenities
- ....
Intrinsic Complexities with Machine Learning

- Understanding the business domain
- Selecting the appropriate Model
- Selecting the appropriate Features
- Fine tuning
Incidental Complexities with Machine Learning

- Integrating with Airbnb’s Data Warehouse
- Scaling model training & serving
- Keeping consistency between: Prototyping vs Production, Training vs Inference
- Keeping track of multiple models, versions, experiments
- Supporting iteration on ML models

ML models take on average 8 to 12 weeks to build

ML workflows tended to be slow, fragmented, and brittle
The ML Infrastructure Team addresses these challenges

Vision
Airbnb routinely ships ML-powered features throughout the product.

Mission
Equip Airbnb with shared technology to build production-ready ML applications with no incidental complexity.
Machine Learning Infrastructure Team

Atul Kale
Engineering Manager

Andrew Hoh
Product Manager

Varant Zanoyan
Software Engineer

Alfredo Luque
Software Engineer

John Park
Software Engineer

Nikhil Simha
Software Engineer

Patrick Yoon
Software Engineer

Xiaohan Zeng
Software Engineer

Aaron Siegel
Engineering Lead, Data Platform

Evgeny Shapiro
Software Engineer

Conglei Shi
Software Engineer

Andrew Cheong
Software Engineer
Bighead: Design Goals
Seamless

Versatile

Consistent

Scalable
Seamless

- Easy to prototype, easy to productionize
- Same workflow across different frameworks
Versatile

- Supports all major ML frameworks
- Meets various requirements
  - Online and Offline
  - Data size
  - SLA
  - GPU training
  - Scheduled and Ad hoc
Consistent

- Consistent environment across the stack
- Consistent data transformation
  - Prototyping and Production
  - Online and Offline
Scalable

- Horizontal
- Elastic
Bighead: Architecture Deep Dive
Prototyping

Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**

Lifecycle Management

Production

Real Time Inference

Batch Training + Inference

Deep Thought

ML Automator

Bighead Service / UI

Redspot

Airflow

Spark

Kubernetes
Redspot

Prototyping with Jupyter Notebooks
Jupyter Notebooks?

What are those?

“Creators need an immediate connection to what they are creating.”
- Bret Victor
The ideal Machine Learning development environment?

- Interactivity and Feedback
- Access to Powerful Hardware
- Access to **Data**
Redspot
a Supercharged Jupyter Notebook Service

- A fork of the JupyterHub project
- Integrated with our Data Warehouse
- Access to specialized hardware (e.g. GPUs)
- File sharing between users via AWS EFS
- Packaged in a familiar Jupyterhub UI
Choose your Jupyter environment

Select a job profile:
Remote Docker

Docker image configuration:
Ubuntu 18.04 image with python 2.7 (for CPU)

Instance configuration
Instance Type: t2.medium
Billing Group: ml_infra

Launch a new instance
- t2.medium (minimal environment)
- c4.xlarge (CPU-optimized (4 CPUs))
- r4.2xlarge (memory-optimized 61GB)
- p2.xlarge (Nvidia Tesla K80 x 1)
Redspot
a Supercharged Jupyter Notebook Service

Consistent
● Promotes prototyping in the exact environment that your model will use in production

Versatile
● Customized Hardware: AWS EC2 Instance Types e.g. P3, X1
● Customized Dependencies: Docker Images e.g. Py2.7, Py3.6+Tensorflow

Seamless
● Integrated with Bighead Service & Docker Image Service via APIs & UI widgets
Prototyping:
- Redspot

Lifecycle Management:
- Bighead Service / UI

Production:
- Deep Thought
- ML Automator

Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**
Docker Image Service
Environment Customization
ML Users have a diverse, heterogeneous set of dependencies

Need an easy way to bootstrap their own runtime environments

Need to be consistent with the rest of Airbnb’s infrastructure
Our configuration management solution

A composition layer on top of Docker

Includes a customization service that faces our users

Promotes **Consistency** and **Versatility**

**Docker Image Service - Dependency Customization**
Bighead Service
Model Lifecycle Management
Model Lifecycle Management - why?

- Tracking ML model changes is just as important as tracking code changes.
- ML model work needs to be **reproducible** to be **sustainable**.
- Comparing experiments before you launch models into production is critical.
Introducing Bighead

Explore, deploy, and monitor machine learning models.

[learn more]

Showing 15 machine learning projects

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Active Artifact Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>census_income_xgb_classification</td>
<td>conglei_shi</td>
<td>auto_trained_model #0</td>
</tr>
<tr>
<td>price_prediction_keras</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-0</td>
</tr>
<tr>
<td>census_income_pytorch_classification</td>
<td>conglei_shi</td>
<td>auto_trained_model #18-1</td>
</tr>
<tr>
<td>image_classification_tensorflow</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-2</td>
</tr>
<tr>
<td>image_classification_pytorch_CNN</td>
<td>conglei_shi</td>
<td>N/A</td>
</tr>
<tr>
<td>speech_recognition_spanish</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-19</td>
</tr>
<tr>
<td>census_income_mxnet_classification</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-20</td>
</tr>
<tr>
<td>speech_recognition_english</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-21</td>
</tr>
<tr>
<td>translation_model_eng_esp</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-22</td>
</tr>
<tr>
<td>object_detection_image_tensorflow</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-23</td>
</tr>
<tr>
<td>vr_listing_spark</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-24</td>
</tr>
<tr>
<td>translation_model_eng_chn</td>
<td>conglei_shi</td>
<td>auto_trained_model #0-25</td>
</tr>
</tbody>
</table>
Bighead Service

Consistent

- Central model management service
- Single source of truth about the state of a model, its dependencies, and what’s deployed

Seamless

- Context-aware visualizations that carry over from the prototyping experience
Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**

**Prototyping**
- **Redspot**
- **jupyter**
- **docker**

**Lifecycle Management**
- **Bighead Service / UI**

**Production**
- **Deep Thought**
- **kubernetes**
- **ML Automator**
- **Airflow**
- **Spark**

**Prototyping**
- **Real Time Inference**
- **Batch Training + Inference**

**Execution Management:** Bighead Library

**Feature Data Management:** Zipline
ML Models are highly heterogeneous in:

---

**Frameworks**
- TensorFlow
- PyTorch
- Keras
- mllib
- xgboost
- mllearn
- Spark
- H2O.ai
- R
- DL4J

**Training data**
- Data quality
- Structured vs Unstructured (image, text)

**Environment**
- GPU vs CPU
- Dependencies
ML Models are hard to keep consistent

- Data in *production* is different from data in *training*
- *Offline* pipeline is different from *online* pipeline
- Everyone does everything in a *different* way
Bighead Library

Versatile
- Pipeline on steroids - compute graph for preprocessing / inference / training / evaluation / visualization
- Composable, Reusable, Shareable
- Support popular frameworks
  - Fast primitives for preprocessing
  - Metadata for trained models

Consistent
- Uniform API
- Serializable - same pipeline used in training, offline inference, online inference
Bighead Library: ML Pipeline

categorical = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'gender',
    'native-country',
]
numeric = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week',
]

p = Pipeline('ClassifyCensusIncome')
p[numeric] >>> [NaNToMean(dtype=np.float32), StandardScaler()]
p[categorical] >>> OneHotLabelEncoder()
p >>> XGBClассификатор(objective='binary:logistic',
                      n_estimators=100,
                      learning_rate=0.1,
                      max_depth=5)
Visualization - Pipeline
Easy to Serialize/Deserialize

```python
In [ ]: p.serialize('test.tar.xz')

In [ ]: p2 = Pipeline.deserialize('test.tar.xz')
```
Visualization - Training Data

```python
In [2]: train, test = load_census_income()
categorical = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'gender',
    'native-country',
]
numeric = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week',
]
lables = np.array([0 if x==' <=50K' else 1 for x in train['income'].values])
```

```python
In [3]: from bighead.core.visualization import show_stats
show_stats(train)
```

**Feature Overview**

<table>
<thead>
<tr>
<th>Name</th>
<th>Not-null</th>
<th>Distinct</th>
<th>Num</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>workclass</td>
<td>categorical</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relationship</td>
<td>categorical</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gender</td>
<td>categorical</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Correlation Analysis**

- age
- workclass
- fnlwgt
- education
- education-num
- marital-status
- occupation
- relationship

...
Visualization - Transformer

```python
from bighead.core.visualization import visualize
visualize(xgboost.get_feature_importance_metadata()[0])
```
Prototyping

Redspot

Lifecycle Management

Bighead Service / UI

Production

Deep Thought

ML Automator

Environment Management: Docker Image Service

Execution Management: Bighead Library

Feature Data Management: Zipline

Real Time Inference

Batch Training + Inference

Airflow

Apache Spark
Deep Thought
Online Inference
<table>
<thead>
<tr>
<th>Consistent with training</th>
<th>Easy to do</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Different data</td>
<td>- Data scientists can’t launch models without engineer team</td>
<td>- Resource requirements varies across models</td>
</tr>
<tr>
<td>- Different pipeline</td>
<td>- Engineers often need to rebuild models</td>
<td>- Throughput fluctuates across time</td>
</tr>
<tr>
<td>- Different dependencies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Thought</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consistent</strong></td>
<td><strong>Seamless</strong></td>
<td><strong>Scalable</strong></td>
</tr>
<tr>
<td>- Docker + Bighead Library: Same data source, pipeline, environment from training</td>
<td>- Integration with event logging, dashboard</td>
<td>- Kubernetes: Model pods can easily scale</td>
</tr>
<tr>
<td></td>
<td>- Integration with Zipline</td>
<td>- Resource segregation across models</td>
</tr>
</tbody>
</table>
Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**
ML Automator
Offline Training and Batch Inference
ML Automator - Why

Automated training, inference, and evaluation are necessary

- Scheduling
- Resource allocation
- Saving results
- Dashboards and alerts
- Orchestration
ML Automator

Consistent
- Docker + Bighead Library: Same data source, pipeline, environment across the stack

Seamless
- Automate tasks via Airflow: Generate DAGs for training, inference, etc. with appropriate resources
- Integration with Zipline for training and scoring data

Scalable
- Spark: Distributed computing for large datasets
Execution Management: **Bighead Library**

Environment Management: **Docker Image Service**

Production:
- **Deep Thought**
  - Real Time Inference
- **ML Automator**
  - Batch Training + Inference

Feature Data Management: **Zipline**
Zipline
ML Data Management Framework
Feature management is hard

- **Inconsistent** offline and online datasets
- **Tricky** to generate training sets that depend on time correctly
- **Slow** training sets backfill
- **Inadequate** data quality checks or monitoring
- **Unclear** feature ownership and sharing
For more information on Zipline, please come to our other talk

Zipline: Airbnb’s Data Management Platform for Machine Learning

2:55pm - 1A 21/22
End-to-End platform to build and deploy ML models to production that is **seamless**, **versatile**, **consistent**, and **scalable**

- Model lifecycle management
- Feature generation & management
- Online & offline inference
- Pipeline library supporting major frameworks
- Docker image customization service
- Multi-tenant training environment

Built on **open source** technology

- TensorFlow, PyTorch, Keras, MXNet, Scikit-learn, XGBoost
- Spark, Jupyter, Kubernetes, Docker, Airflow
Future Works

Monitoring

● Feature distribution
● Model performance

BigQueue

● On demand compute service with heterogeneous resources
One more thing...
To be Open Sourced in Q1 2019

If you want to collaborate come talk to us

or email andrew.hoh@airbnb.com
Questions?
Appendix