Pitfalls of Machine Learning in production

Antoine Sauray - Nantes Machine Learning Meetup - 7/10/19

Who am I ?

Antoine Sauray 👋 @asauray

Software engineer, Scala dev

Founder of Hyperplan ML platform

We are hosted at the Halle 6 in Ile de Nantes

Previously worked at iAdvize as ML Engineer

4 years exp in mobile dev (Android and iOS native)

Software engineering

AND

Machine Learning

Machine Learning is a small part of your system

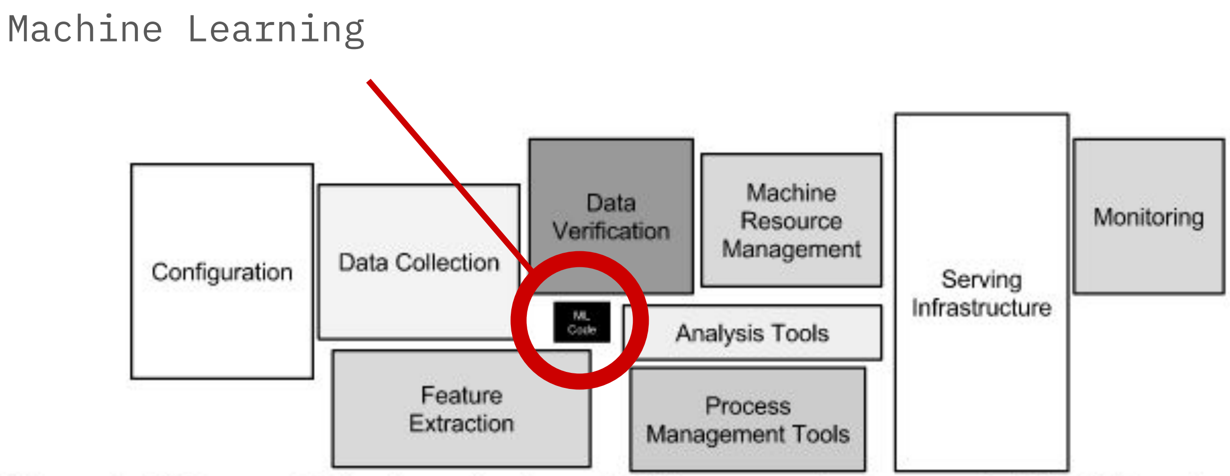


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

“Hidden Technical Debt in Machine Learning Systems” - 2015

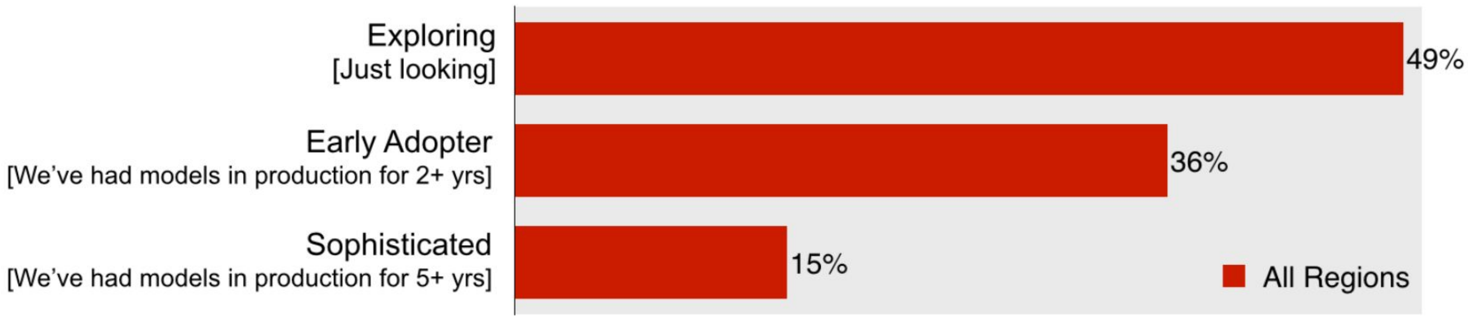
Machine Learning is a small part of your system

Not Machine Learning

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

“Hidden Technical Debt in Machine Learning Systems” - 2015

Machine Learning is hard to put in production



O'Reilly survey - August 2018

Machine Learning creates more tech debt

*“ML systems have a special capacity for incurring technical debt, because they have **all of the maintenance problems** of traditional code plus an additional set of **ML-specific issues.**”*

Hidden Technical Debt in Machine Learning Systems - 2015

Software practices VS **ML practices**

Strong abstractions
OOP, FP
Hexagonal architecture

Tooling
Version Control
Testing frameworks
Monitoring
Logging
CI

Deployments
CD
A/B tests

VS

ML practices

Strong abstractions
OOP, FP
Hexagonal architecture

Tooling
Version Control
Testing frameworks
Monitoring
Logging
CI

Deployments
CD
A/B tests

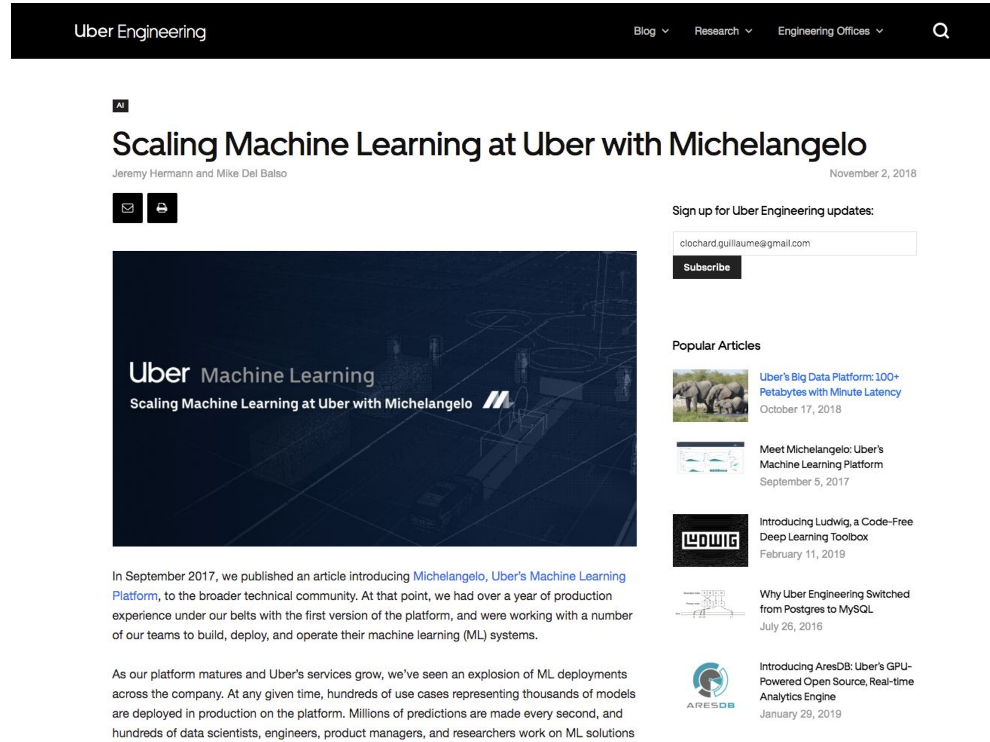
VS

?

+ specific ML issues
Python
drift concept
etc.

Uber

"ML as software engineering"



Uber Engineering   Blog   Research   Engineering Offices

AI

Scaling Machine Learning at Uber with Michelangelo

Jeremy Hermann and Mike Del Balso   November 2, 2018

Uber Machine Learning
Scaling Machine Learning at Uber with Michelangelo

In September 2017, we published an article introducing Michelangelo, Uber's Machine Learning Platform, to the broader technical community. At that point, we had over a year of production experience under our belts with the first version of the platform, and were working with a number of our teams to build, deploy, and operate their machine learning (ML) systems.

As our platform matures and Uber's services grow, we've seen an explosion of ML deployments across the company. At any given time, hundreds of use cases representing thousands of models are deployed in production on the platform. Millions of predictions are made every second, and hundreds of data scientists, engineers, product managers, and researchers work on ML solutions

Sign up for Uber Engineering updates:

clochard.guillaume@gmail.com

Subscribe

Popular Articles

Uber's Big Data Platform: 100+ Petabytes with Minute Latency
October 17, 2018

Meet Michelangelo: Uber's Machine Learning Platform
September 5, 2017

Introducing Ludwig, a Code-Free Deep Learning Toolbox
February 11, 2019

Why Uber Engineering Switched from Postgres to MySQL
July 26, 2016

Introducing AresDB: Uber's GPU-Powered Open Source, Real-time Analytics Engine
January 29, 2019

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.


“Working
code is a
side effect”
Alberto Brandolini

Diving into issues

Diving into issues

Collaboration

Data dependencies

Training and Serving Skews

Bad and good patterns

Collaboration difficulties

Collaboration difficulties

How to track experiments ?

What algorithm has been trained ?

When ?

On what data ?

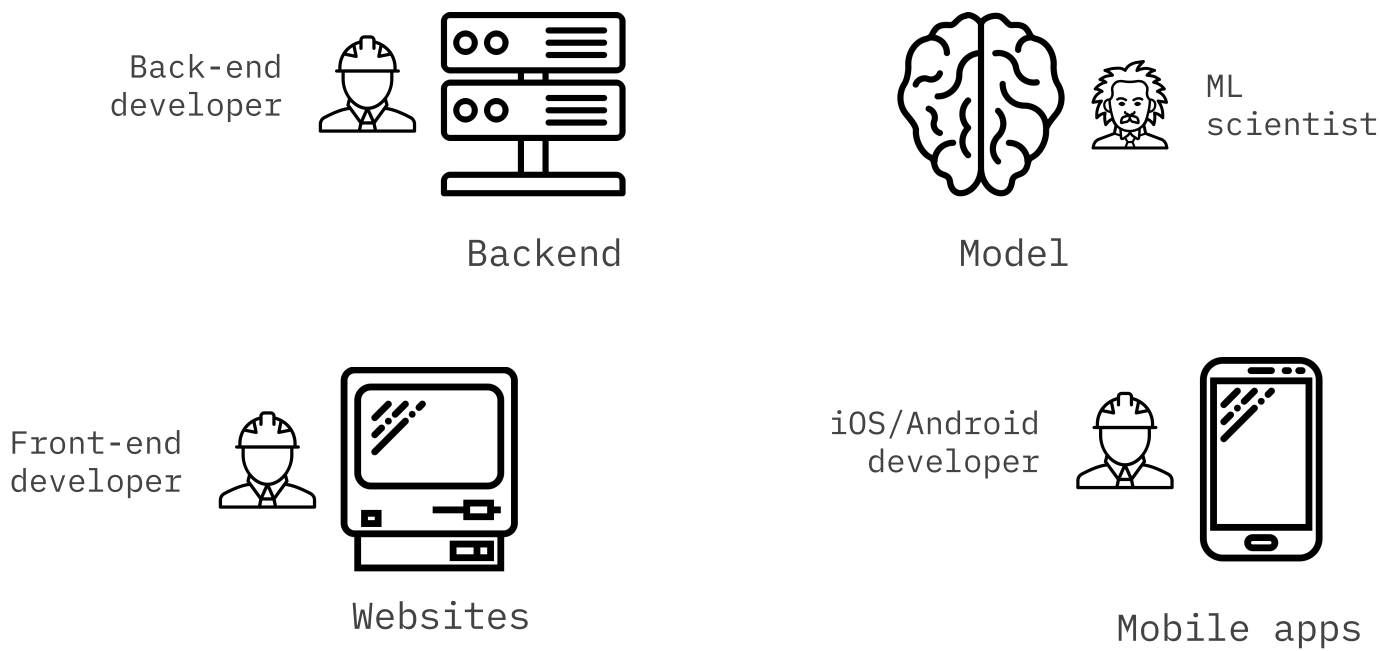
With which set of parameters ?

How can we compare them ?

- Avoid repeating same experiments
- Reuse code

Collaboration difficulties

Working with others



Collaboration difficulties

Working with others

Teams of engineers and researchers have different goals

Data scientists and researchers seek **quality**, at the cost of simplicity
Dev and ops like seek **simplicity**, at the cost of performance

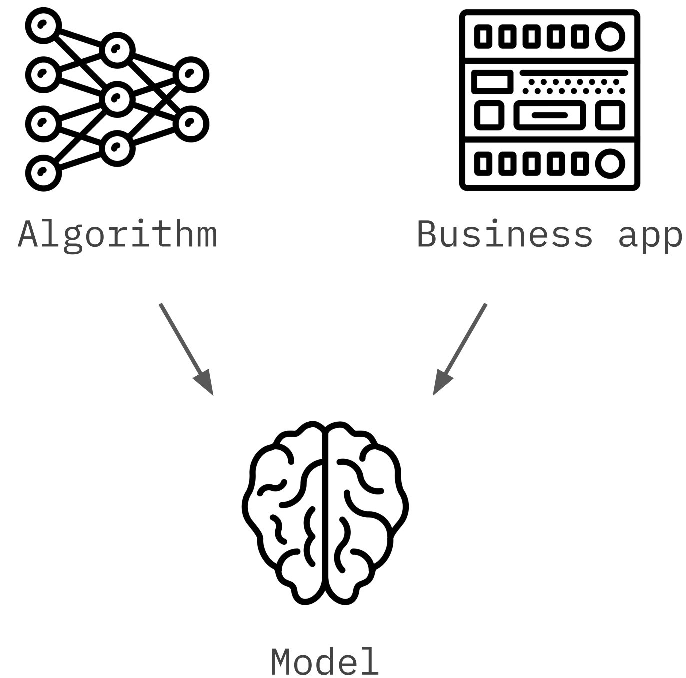
Data Dependencies

Data Dependencies

Unstable Data dependencies

A model can consume signals from:

- An algorithm (possibly ML) that updates over time
- An application that may receive updates



CACE: **C**hange **A**nything **C**hanges **E**verything

Data Dependencies

Underutilized Data dependencies

Keep the features to a strict minimum reduces the multicollinearity risk.

- Legacy Features can be forgotten
- Features can be added as bundles
- ǫ-Features are complex and bring little value

Underutilized dependencies can be detected via exhaustive **leave-one-feature-out** evaluations.

Training and Serving Skews

According to the TensorFlow doc, 4 skews:

- Schema
- Feature
- Distribution
- Scoring and Serving

Training and Serving Skews

Schema skew

Example: after a migration/the deprecation of a field.

This may be undetected and cause you model to silently fail.

```
{
  "surfaceAppartement": 70,
}
```

↓

```
{
  "surfaceAppartement": null,
  "appartmentSurface": 70
}
```

Training and Serving Skews

Feature skew

When different teams implement the machine learning models and the documentation is nonexistent or insufficient.

This is even more important **when serving on multiple platforms** with different teams.

```
features = {
  # is this milliseconds or
  # seconds ?
  'video_duration': 10
}

prediction =
  clf.predict(features)
```

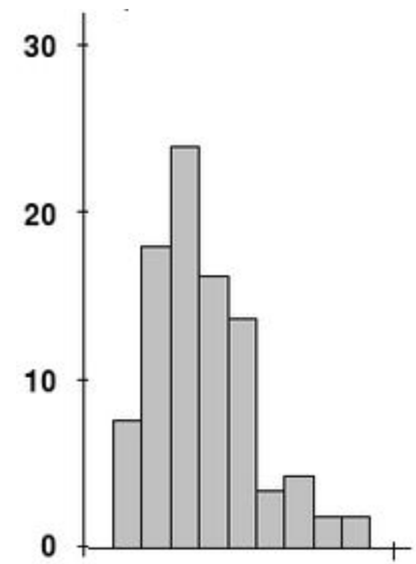
Training and Serving Skews

Distribution skew

When the distribution of feature values is different between training and serving.

When data scientists train their algorithms with a faulty sampling mechanism.

When a different corpus is used for training to overcome lack of initial data.



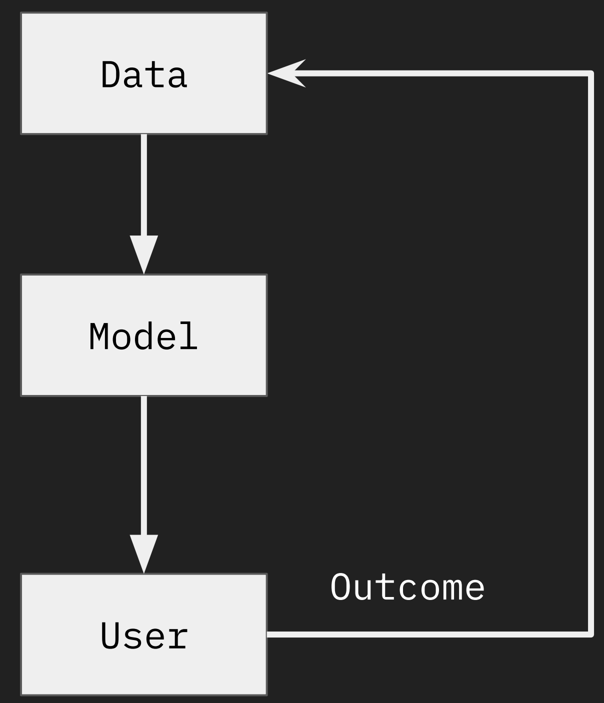
Training and Serving Skews

Scoring and Serving skew

A Direct Feedback loop can help you measure the results of your algorithm and generate training data for later.

But! Models may influence the data they will be trained on later.

Exploration / Exploitation trade off



Training and Serving Skews

Scoring and Serving skew

Hardcoded thresholds can lead to errors because they are model specific.

If model is automatically trained with new batches of data, the previous threshold is invalidated

But thresholds can be learnt.

```
def keep_above_threshold(prediction, t):
    for p in predictions:
        if p.prob > t:
            yield p

clf = load('model1.joblib')
predictions = clf.predict(features)
return keep_above_threshold(predictions, 0.6)
```

```
clf = load('model2.joblib')

predictions = clf.predict(features)
return keep_above_threshold(predictions, 0.6)
```

Bad and good patterns

Bad and good patterns

Bad: **Glue code**

Wikipedia: “... executable code that serves solely to «adapt» different parts of code that would otherwise be incompatible”.

Problems
Maintenance is hard
Complexity is high

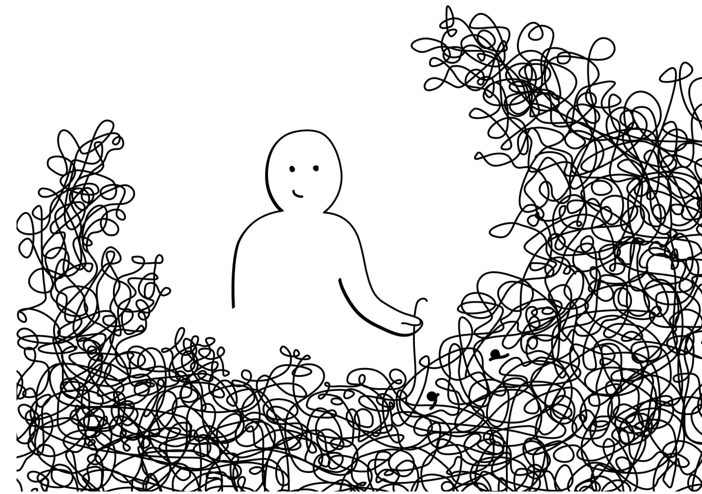
Bad and good patterns

Bad: **Pipeline jungle**

When thinking about data collection after creating the algorithms.

It is often hard to gather data in real time for the execution of the algorithm.

- Many collection steps
- SQL Join everywhere
- Intermediate results



This is **fragile**

→ Rethink the pipeline as a whole

Bad and good patterns

Bad: **Dead experimental codepath**

Shortcuts taken by developers to implement features faster.

Avoid conditional branching in production because of

- Cyclomatic complexity
- Things break unexpectedly

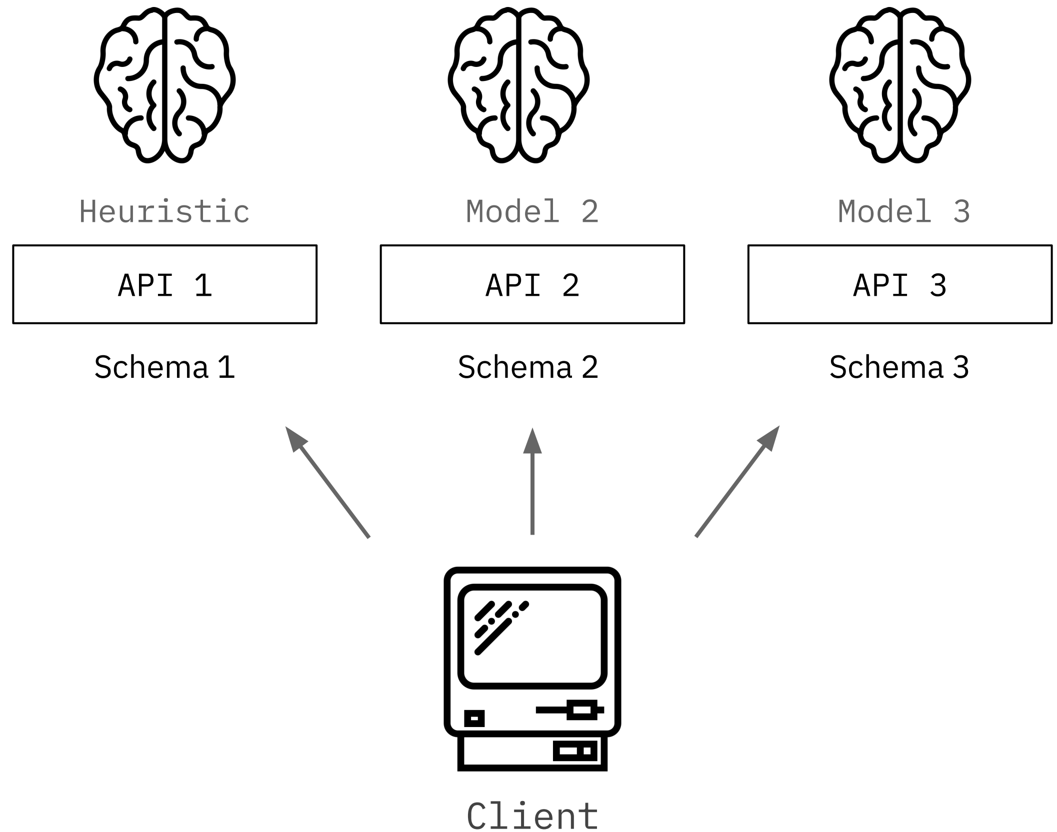
```
if customer_id == "1":
    # but customer 1 does not exist anymore
    return clf1.predict(data)
else:
    return clf2.predict(data)
```

Bad and good patterns

Bad: **No Abstraction**

Client handle different APIs for the same problem.

- More code client side
- Duplication of code in APIs
- Failover and AB tests are managed on the client



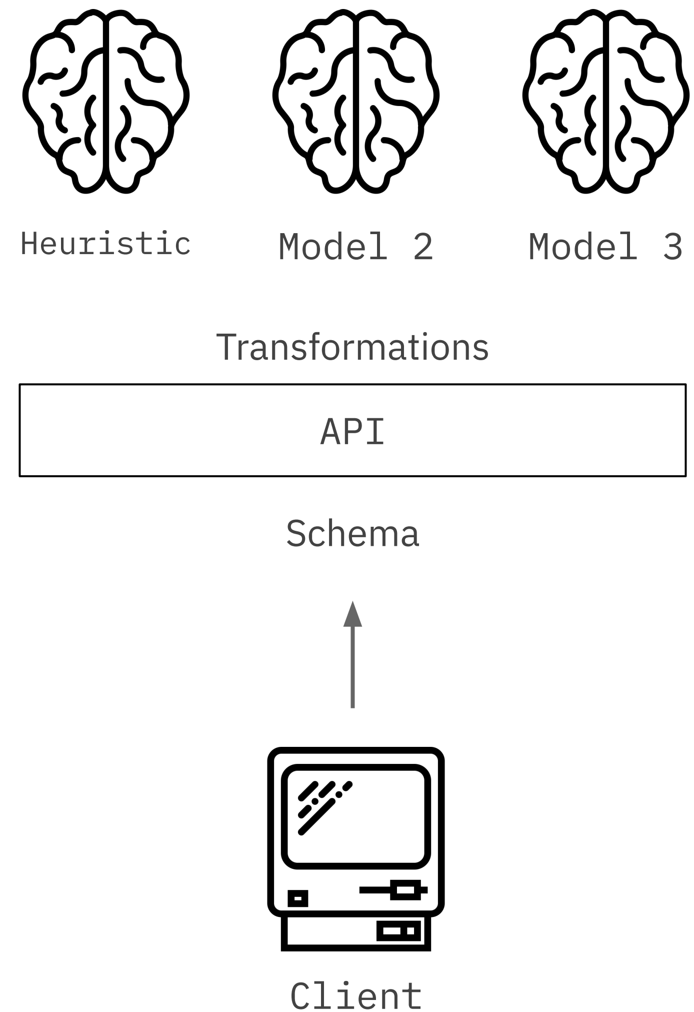
Bad and good patterns

Good: **Abstraction**

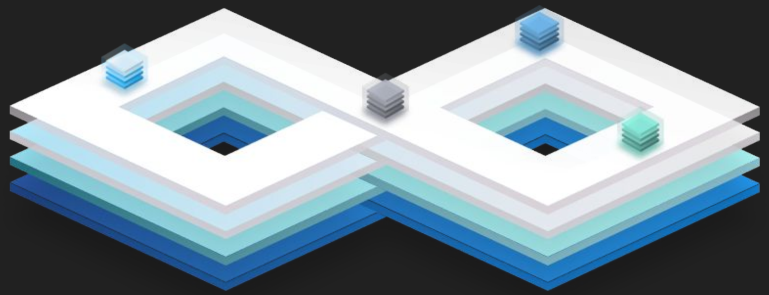
An interface between the client and the models

Advantages:

- Start with simple heuristics to get data early
- Add data on the API schema progressively
- Ship a new model safely with rollbacks
- Write less code on the client side



Platforms



The basics

We have some basics tools like classic software

Versioning

- Git LFS
- DVC

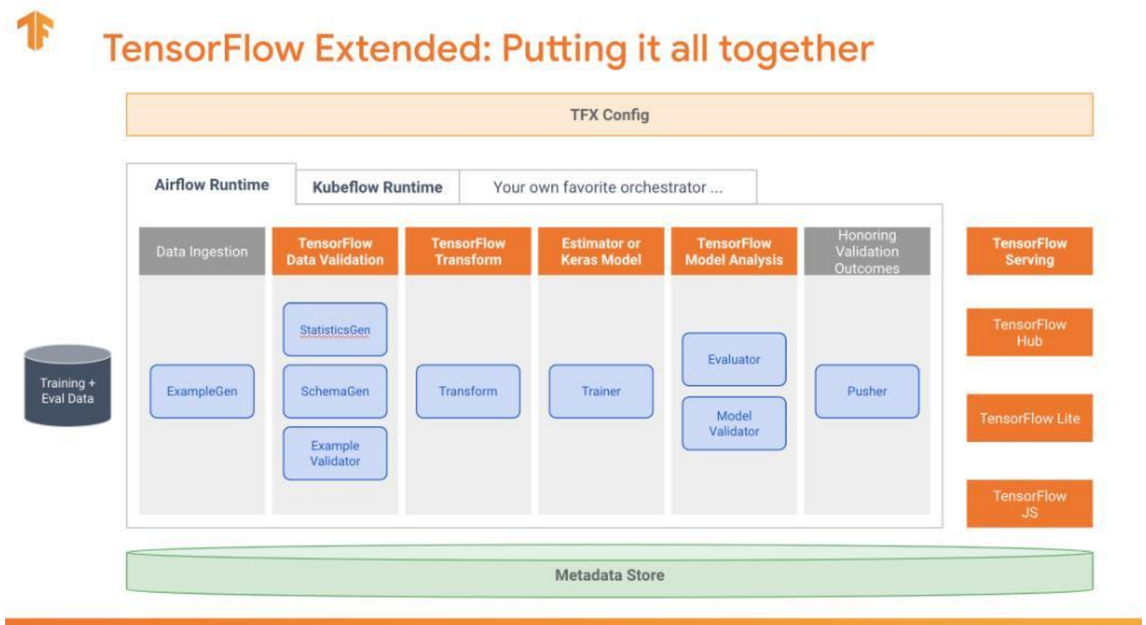
Tracking platforms

- Weights and biases wandb.com
- Datmo github.com/datmo/datmo
- MLFlow Tracking

TensorFlow Extended

End-to-end platform for deploying production ML pipelines

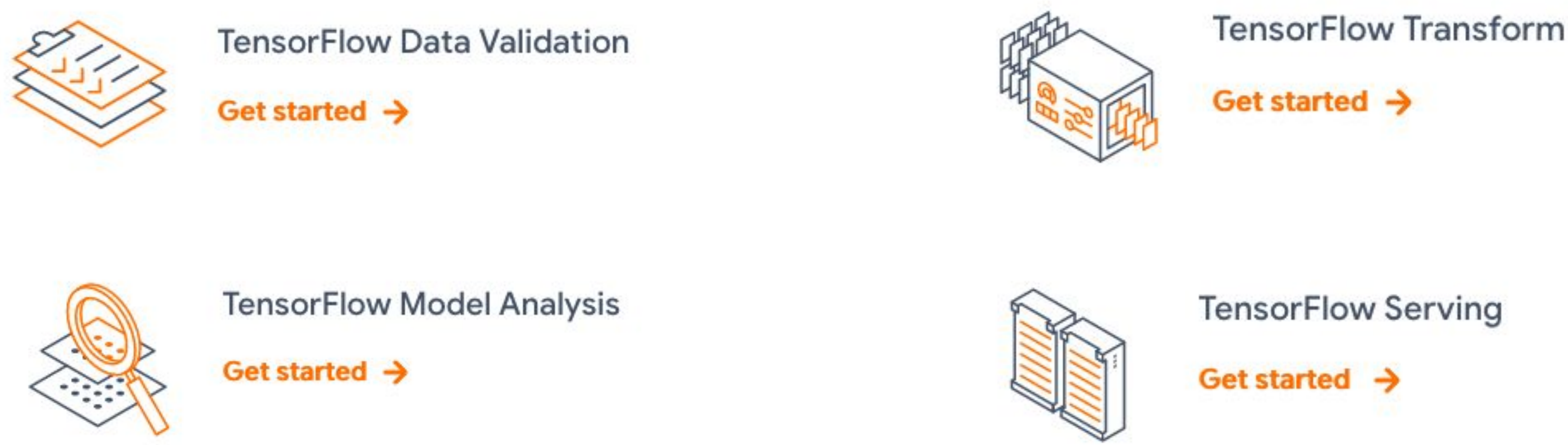
Since 2018



TensorFlow Extended

End-to-end platform for deploying production ML pipelines

Since 2018



TensorFlow Extended

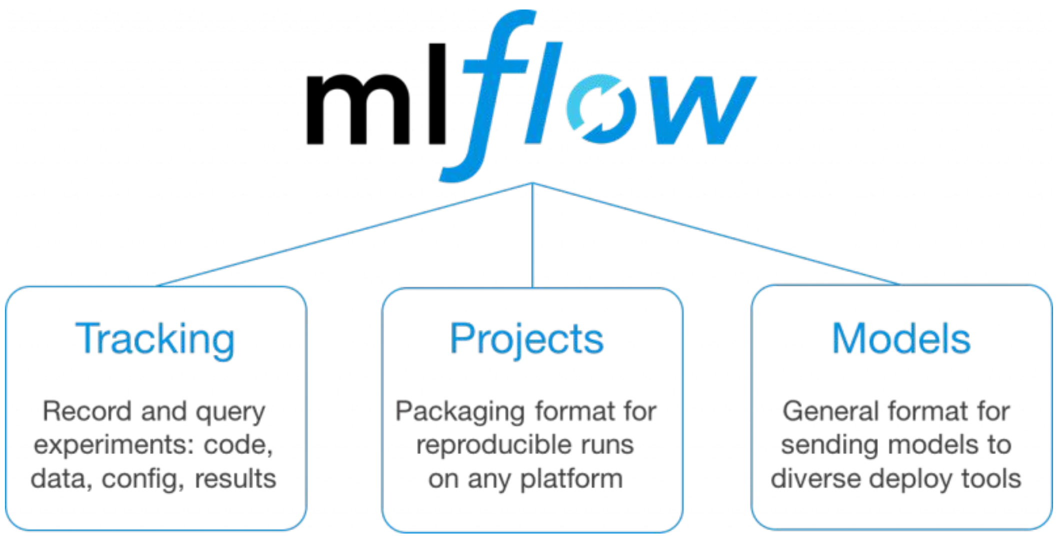
Google

Tracking and Reproducibility	❌
Skew Detection	✔️ (schema, feature, distribution)
Consistent Pipeline (Training & Serving)	✔️ (only tf.Serving and Apache Beam)
Multiple Deployment Targets	✔️ (API, browser, on device)
Multi Framework	❌ (possible via TensorFlow 2.0)

MLFlow

Machine Learning Platform for ML Lifecycle management

v1: summer 2019



MLFlow

databricks

Tracking and Reproducibility	✓
Skew Detection	✗
Consistent Pipeline (Training & Serving)	✓ (via `save_model`)
Multiple Deployment Targets	✗ (only REST API)
Multi Framework	✓

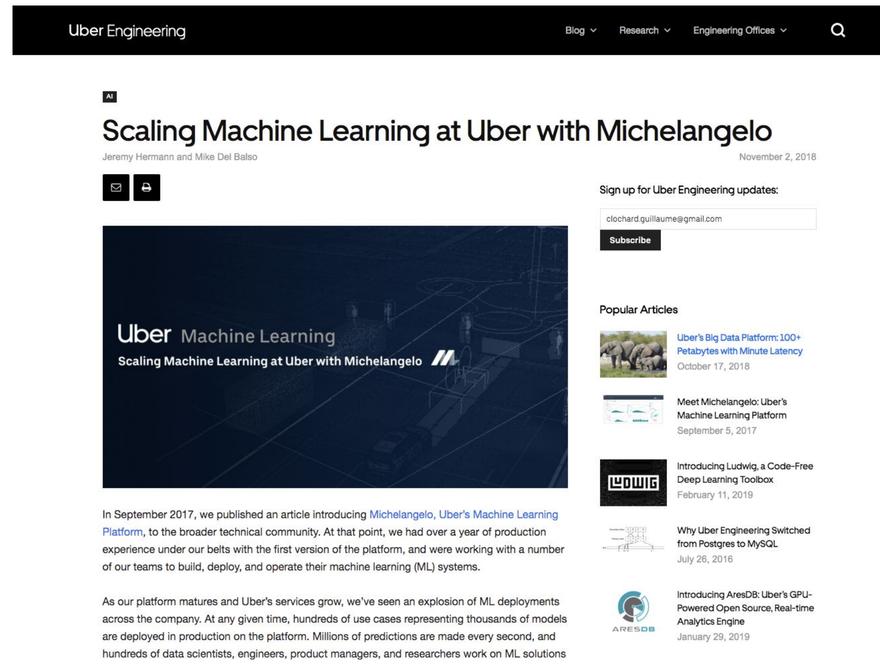
Michelangelo (closed source)

Uber

Since 2016

“Seamlessly build, deploy, and operate machine learning solutions **at Uber’s scale**”

eng.uber.com/michelangelo



Michelangelo (closed source)

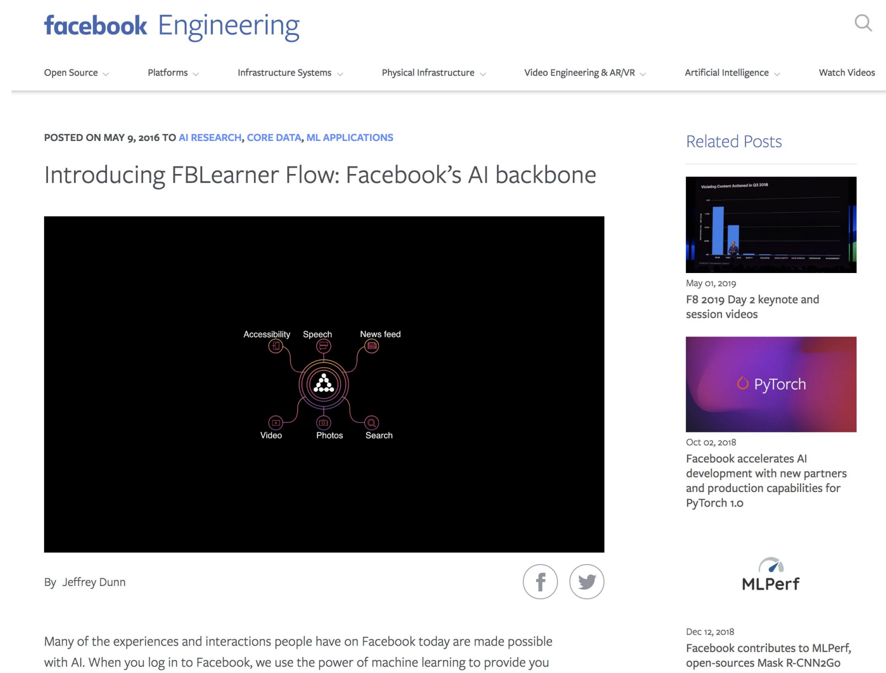
Uber

Tracking and Reproducibility	✔️
Skew Detection	❌
Consistent Pipeline (Training & Serving)	with a compiled DSL ✔️
Multiple Deployment Targets	NA
Multi Framework	✔️

FB Learner Flow (closed source)

A platform capable of easily reusing algorithms in different products, scaling to run thousands of simultaneous custom experiments, and managing experiments with ease.

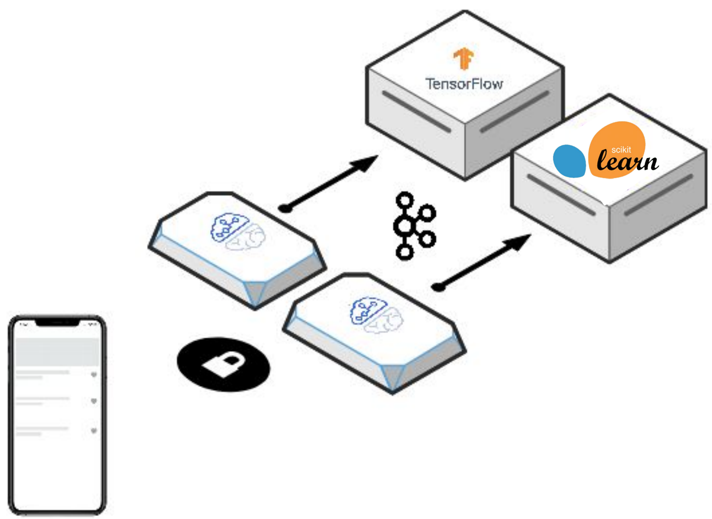
engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone



FB Learner Flow (closed source)

Tracking and Reproducibility	✓
Skew Detection	NA
Consistent Pipeline (Training & Serving)	NA
Multiple Deployment Targets	NA
Multi Framework	✓

Hyperplan



A Proxy for Machine Learning

Serve your algorithms in production

- Skew detection
- Feedback loop
- AB Testing

Still under active development

Hyperplan

Tracking and Reproducibility	❌
Skew Detection	✔️ (schema, feature, distribution)
Consistent Pipeline (Training & Serving)	❌
Multiple Deployment Targets	✔️
Multi Framework	✔️

To sum up

		mlflow	Uber		Hyperplan
Tracking and Reproducibility	✗	✓	✓	✓	✗
Skew Detection	✓	✗	✗	NA	✓
Consistent Pipeline (Training & Serving)	✗	✓	✓	NA	✗
Multiple Deployment Targets	✓	✗	NA	NA	✓
Multi Framework	✗	✓	✓	✓	✓

To sum up

TensorFlow Extended and MLFlow are great for big orgs
TFX has a proven record for solving production issues, but only with TensorFlow
MLFlow is still young and targets many problems, pick what you need

Uber and Facebook ?
Designed specifically for their org, huge platforms

Hyperplan is made for small and medium orgs
Benefits of monitoring at a small cost

References

Machine Learning: The High Interest Credit Card Of Technical Debt, 2014

Hidden Technical Debt in Machine Learning Systems, (Google) 2015

Machine Learning in Production: Developing and Optimizing Data Science Workflows and Applications, First Edition (O'Reilly)

The State of Machine Learning Adoption in the Enterprise (O'Reilly)

TensorFlow Docs https://www.tensorflow.org/tfx

MLFlow Docs https://www.mlflow.org/docs/latest/index.html

Merci !

@asauray - hyperplan.io