Machine Learning Systems

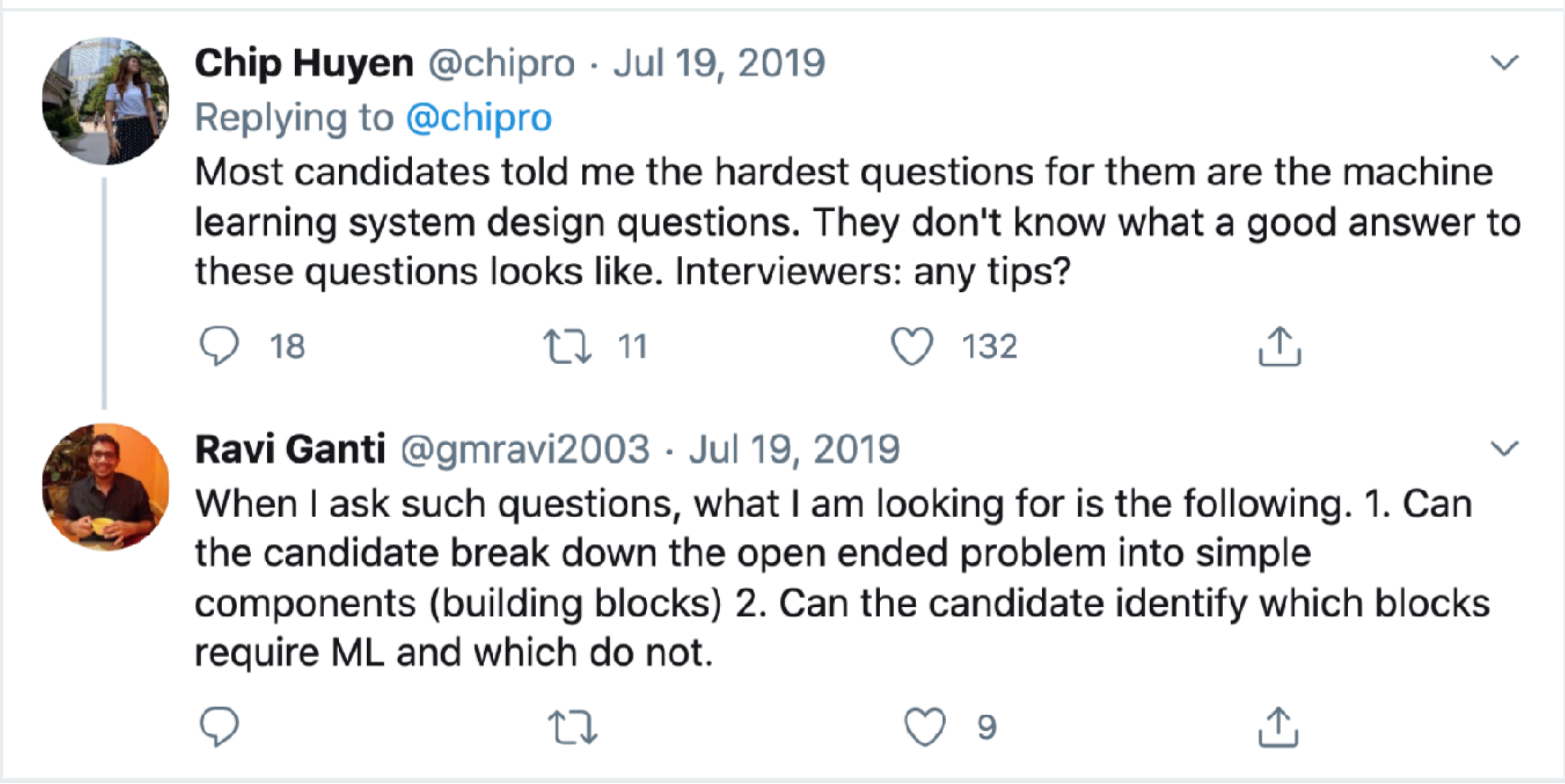
Lecture 4: Designing ML Systems

Pooyan Jamshidi

UofSC™

What is missing?

The gap between ML Research and Production



Chip Huyen @chipro · Jul 19, 2019

Replying to @chipro

Most candidates told me the hardest questions for them are the machine learning system design questions. They don't know what a good answer to these questions looks like. Interviewers: any tips?

18 11 132

Ravi Ganti @gmravi2003 · Jul 19, 2019

When I ask such questions, what I am looking for is the following. 1. Can the candidate break down the open ended problem into simple components (building blocks) 2. Can the candidate identify which blocks require ML and which do not.

9

What is missing?

The gap between ML Research and Production

Dmitry Kislyuk @dkislyuk · Jul 19, 2019

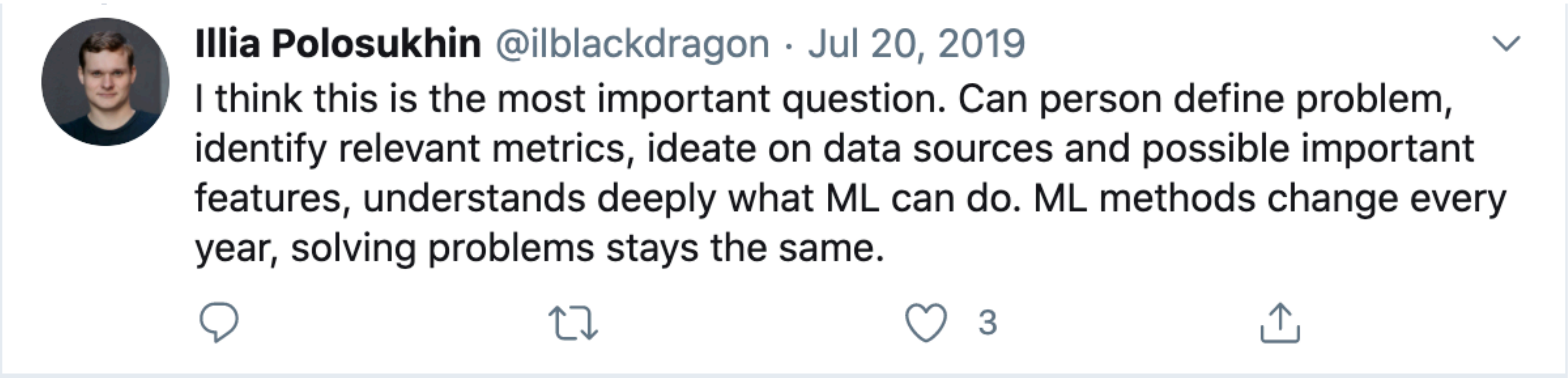
Replying to @lishali88 and @chipro

Most candidates know the model classes (linear, decision trees, lstms, convnets) and memorize the relevant info, so for me the interesting bits in ML systems interviews are data cleaning, data prep, logging, eval metrics, scalable inference, feature stores (recommenders/rankers)

11

What is missing?

The gap between ML Research and Production


Illia Polosukhin @ilblackdragon · Jul 20, 2019

I think this is the most important question. Can person define problem, identify relevant metrics, ideate on data sources and possible important features, understands deeply what ML can do. ML methods change every year, solving problems stays the same.

3

In ML Systems, only a small fraction is comprised of actual ML code

Hidden Technical Debt in Machine Learning Systems

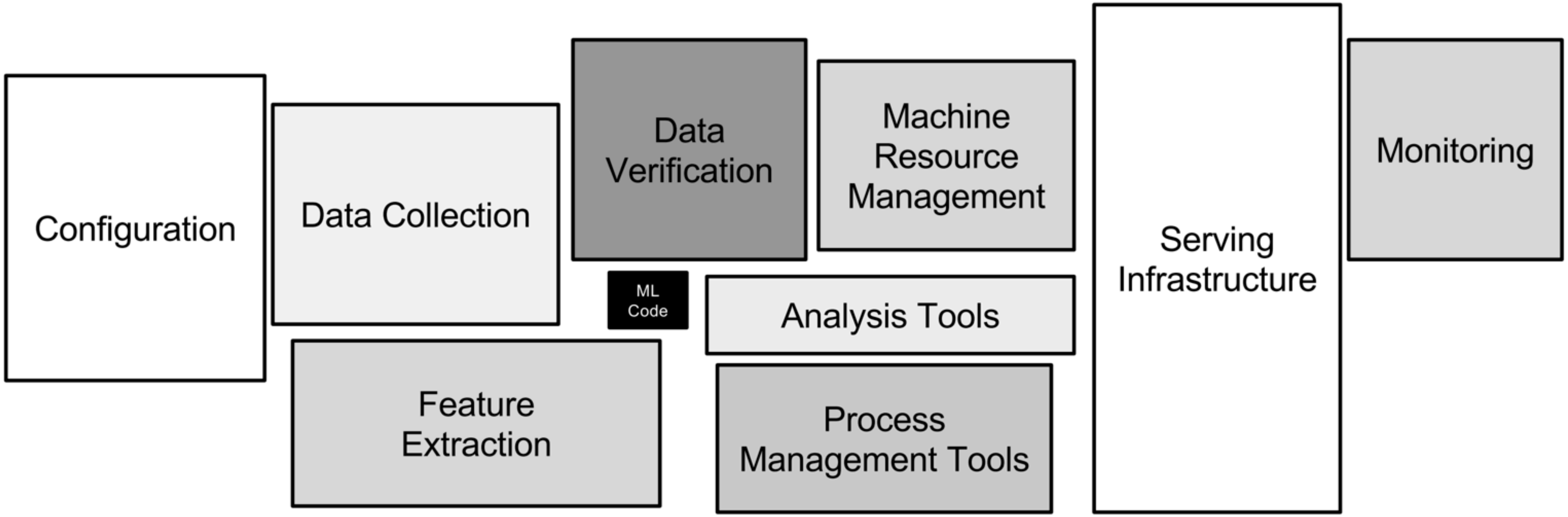
D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

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.

A vast array of surrounding infrastructure and processes is needed to support evolution of ML systems



Technical debt that can accumulate in ML systems

- Data dependencies
- Model complexity
- Reproducibility
- Testing
- Monitoring
- Configuration issues
- External changes

Systems issues in ML Systems

Understanding the Nature of System-Related Issues in Machine Learning Frameworks: An Exploratory Study

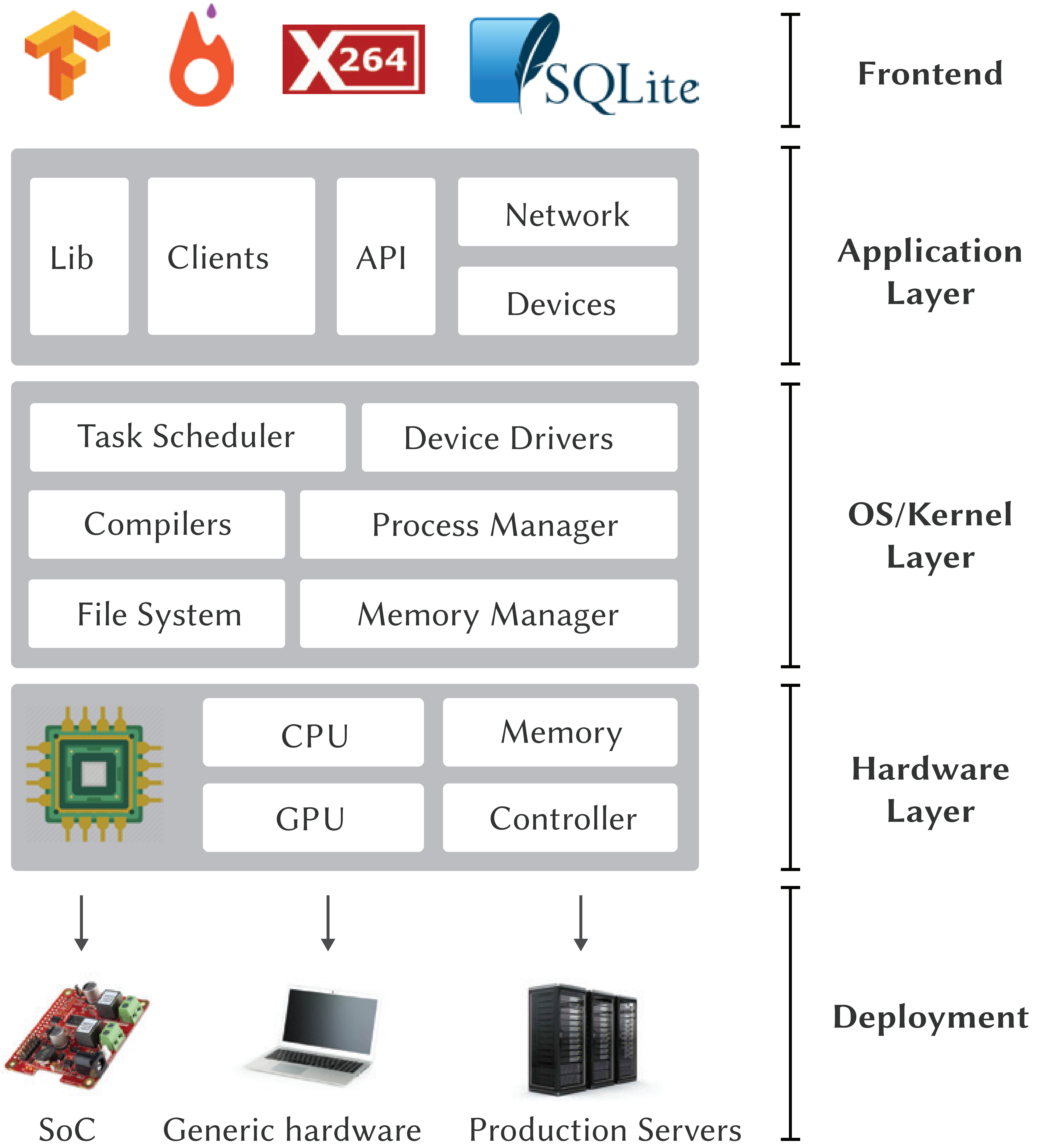
Yang Ren
University of South Carolina
USA

Gregory Gay
Chalmers and the University of Gutenberg
Sweden

Christian Kästner
Carnegie Mellon University
USA

Pooyan Jamshidi
University of South Carolina
USA

System = Software + Middleware + Hardware

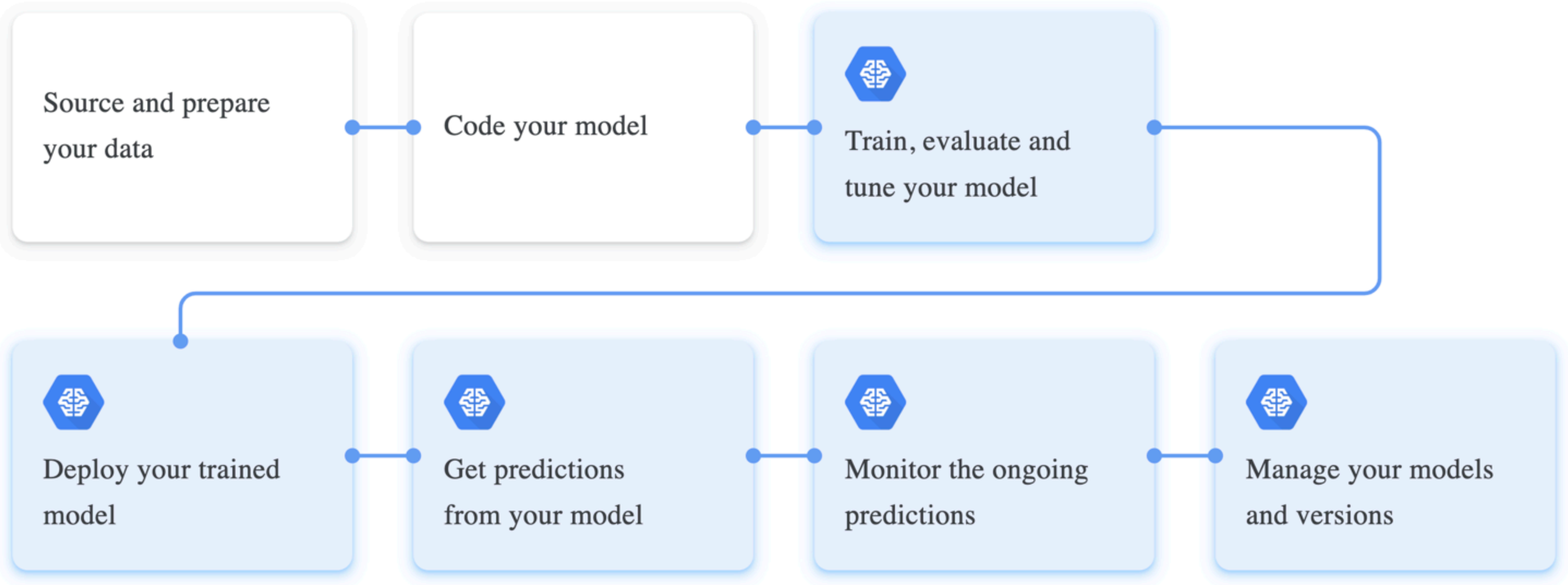


Systems issues in ML Systems

Category (Short Title)	Definition
API Mismatch (API)	Change to API version or mixed usage of APIs leading to performance degradation.
Compilation Error (Compl)	Failure to compile the source code.
Configuration Error (Config)	Configuration settings lead to performance degradation or error.
Connection Error (Conn)	Unexpected or wrongly-formatted connection request leads to error.
Data Race (Race)	Two or more threads access the same memory location concurrently.
Execution Error (Exec)	Unexpected error leads to the execution process crashing.
Hardware-Architecture Mismatch (HA)	Unfit hardware architecture leads to performance degradation or compilation error.
Memory Allocation (MA)	Memory allocation leads to performance degradation.
I/O Slowdown (I/O)	Issues with I/O processes lead to performance degradation.
Memory Leak (ML)	A failure in a program to release memory.
Model Conversion (Conv)	Performance degradation due to type conversion/cast.
Multi-Threading Error (MT)	Performance degradation due to thread interaction.
Performance Regression (PR)	Performance degradation after a change to the system.
Slow Synchronization (SYNC)	Synchronization between components leads to performance degradation.
Unexpected Resource Usage (RU)	Unusual system resource usage or requests leading to error or performance degradation.

The Building Process of ML Systems

Continuous Delivery for ML Systems



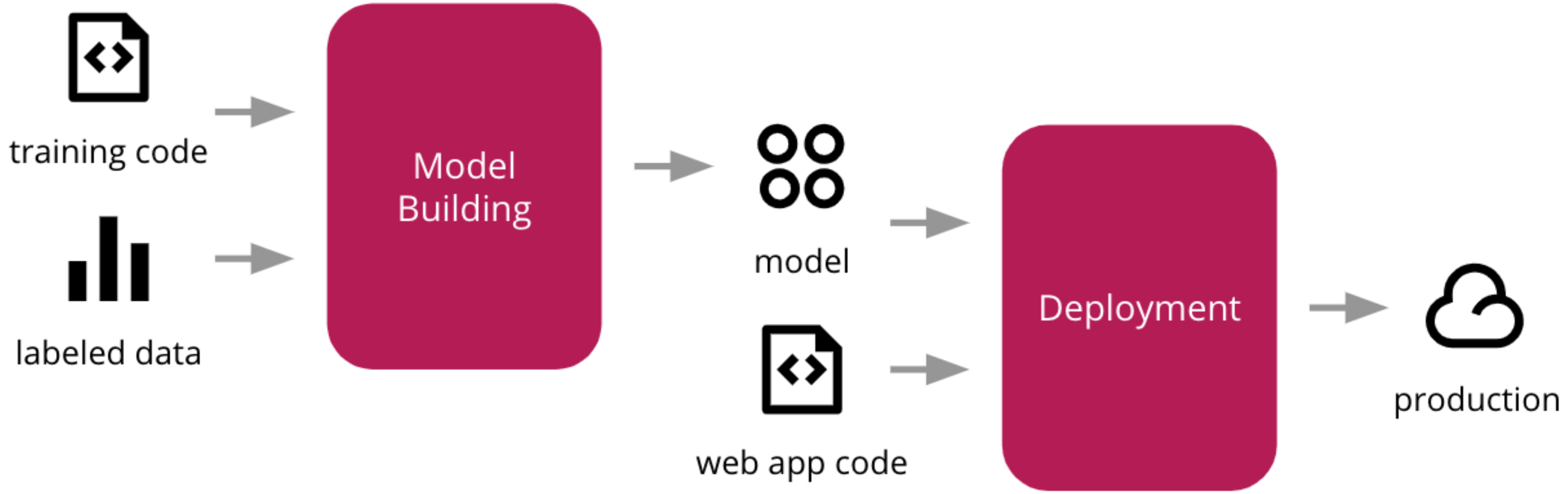
A Machine Learning System is more than just a model

Change in ML Systems

Data	Model	Code
Schema	Algorithms	Business Needs
Sampling over Time	More Training	Bug Fixes
Volume	Experiments	Configuration

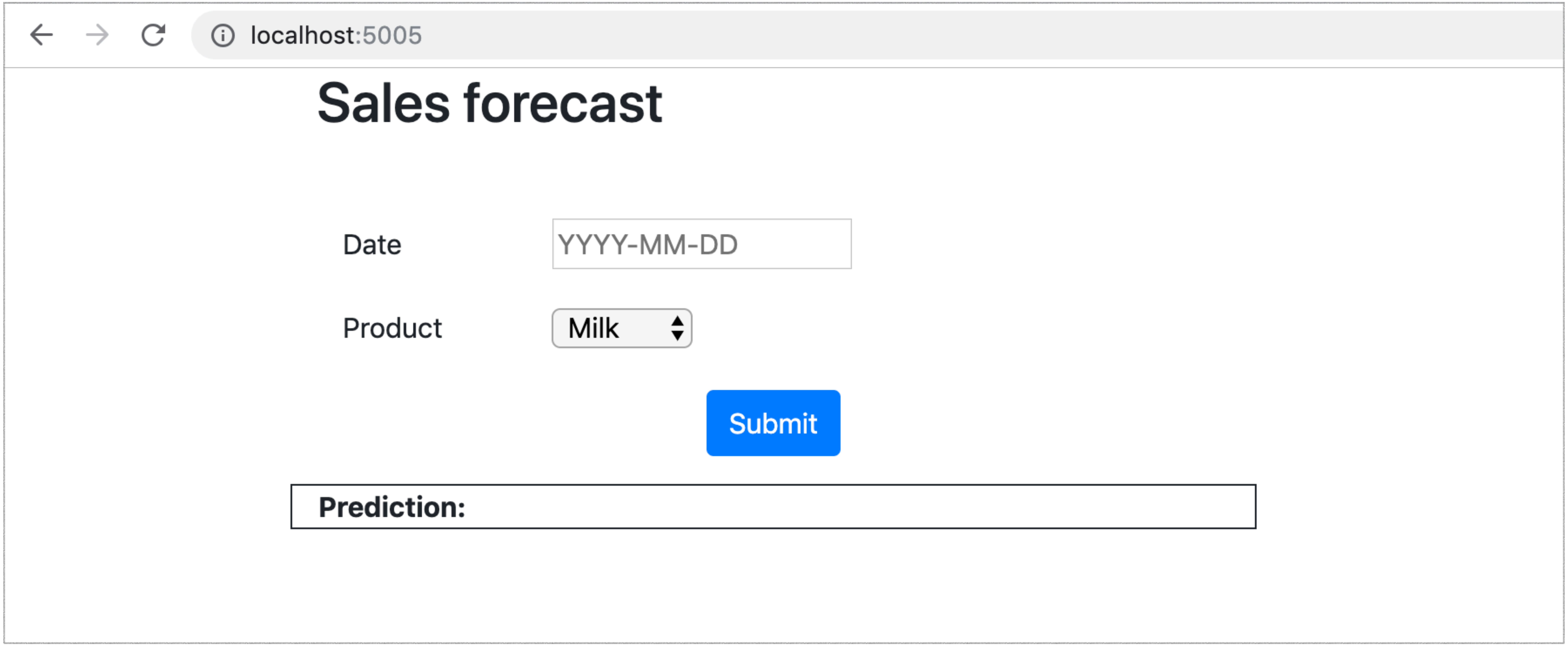
Continuous Delivery for Machine Learning

Train ML model, integrate it with an application, and deploy into production



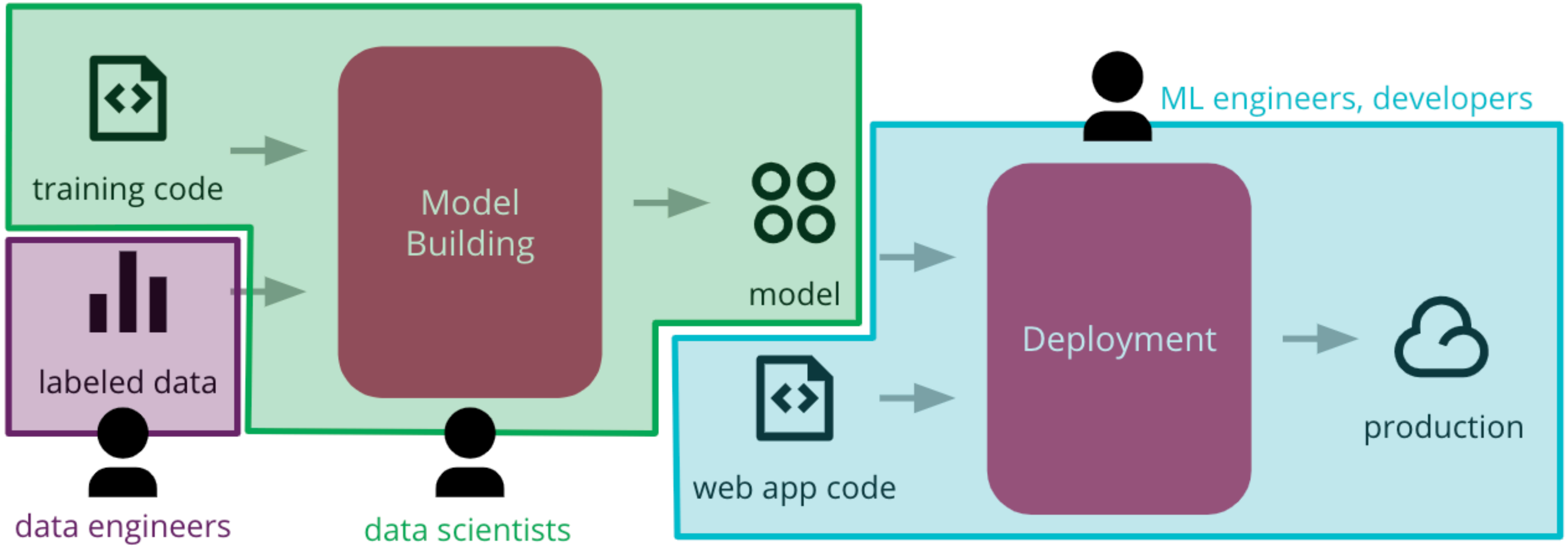
Continuous Delivery for Machine Learning

ML model behind a web application

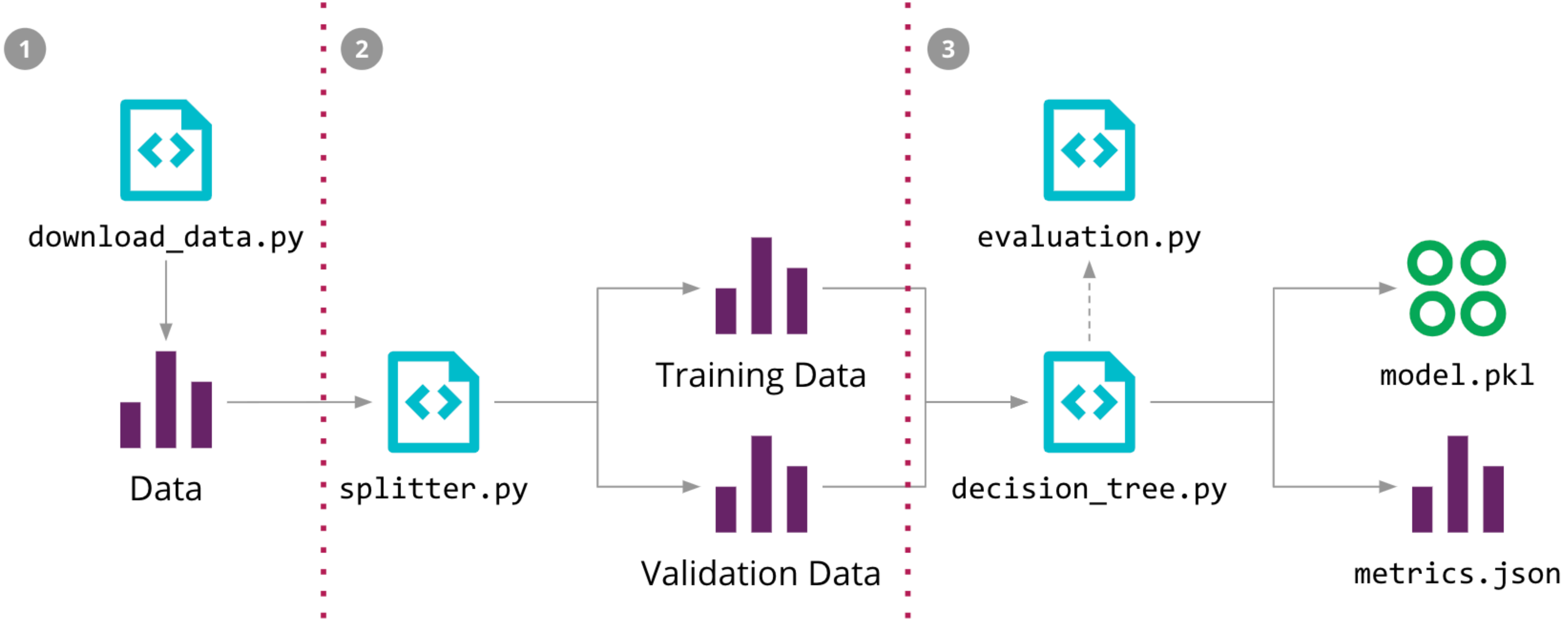


Challenges

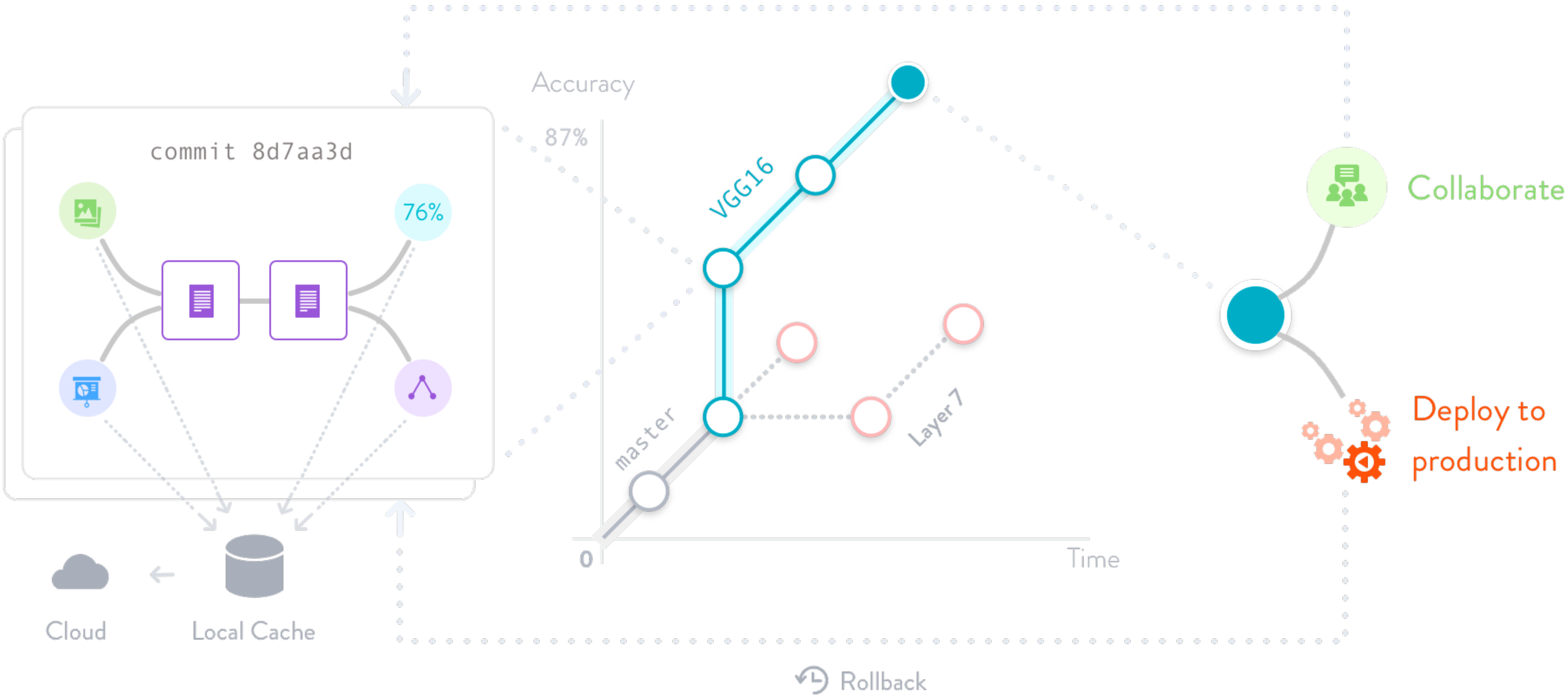
- Throw over the wall
- Models that only work in a lab environment
- Even if make it to production, they become stale and hard to update
- Reproducible and auditable



ML pipeline


1
2
3
download_data.py
Data
splitter.py
Training Data
Validation Data
evaluation.py
decision_tree.py
model.pkl
metrics.json

Configure ML pipeline: DVC tracks ML models and data sets



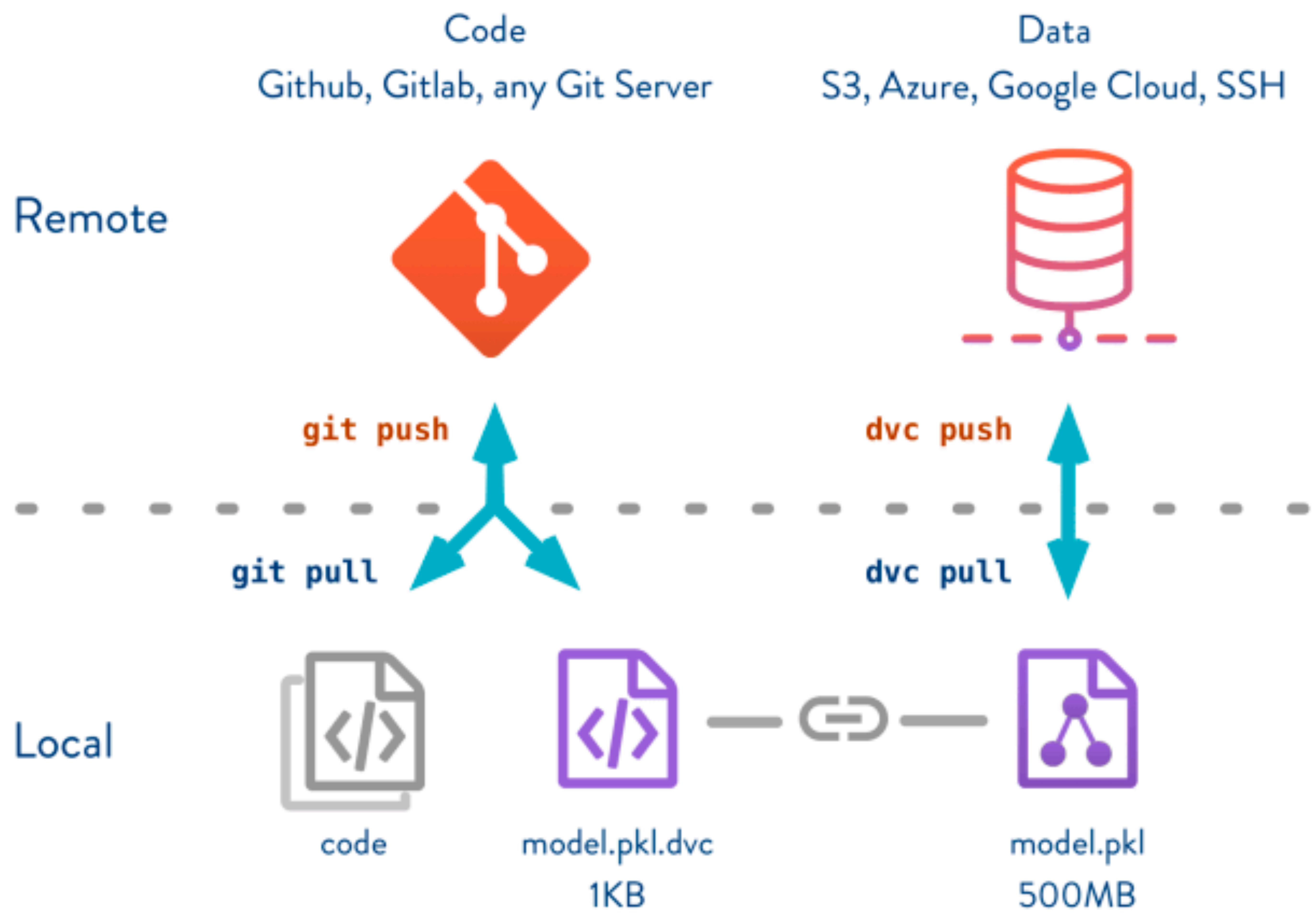
Configure ML pipeline: DVC tracks ML models and data sets

```
dvc run -f input.dvc \ ❶
  -d src/download_data.py -o data/raw/store47-2016.csv python src/download_data.py
dvc run -f split.dvc \ ❷
  -d data/raw/store47-2016.csv -d src/splitter.py \
  -o data/splitter/train.csv -o data/splitter/validation.csv python src/splitter.py
dvc run ❸
  -d data/splitter/train.csv -d data/splitter/validation.csv -d src/decision_tree.py \
  -o data/decision_tree/model.pkl -M results/metrics.json python src/decision_tree.py
```

Configure ML pipeline: DVC tracks ML models and data sets

- Each run will create a file, that can be committed to version control
- DVC allows other people to reproduce the entire ML pipeline, by executing the *dvc repro* command.
- Once we find a suitable model, we will treat it as an artifact that needs to be *versioned* and *deployed* to production.
- With DVC, we can use the *dvc push* and *dvc pull* commands to publish and fetch it from *external storage*.

Configure ML pipeline: DVC tracks ML models and data sets

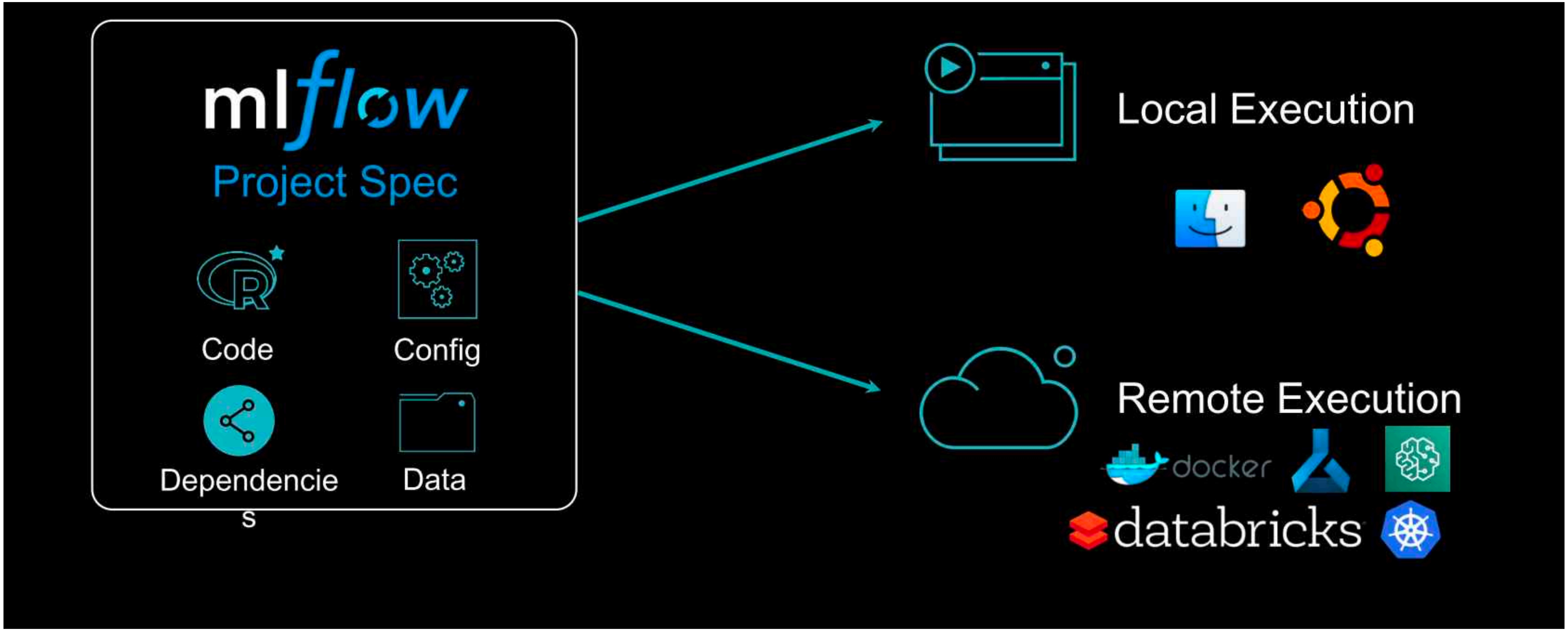


There are other open source tools for versioning

Pachyderm

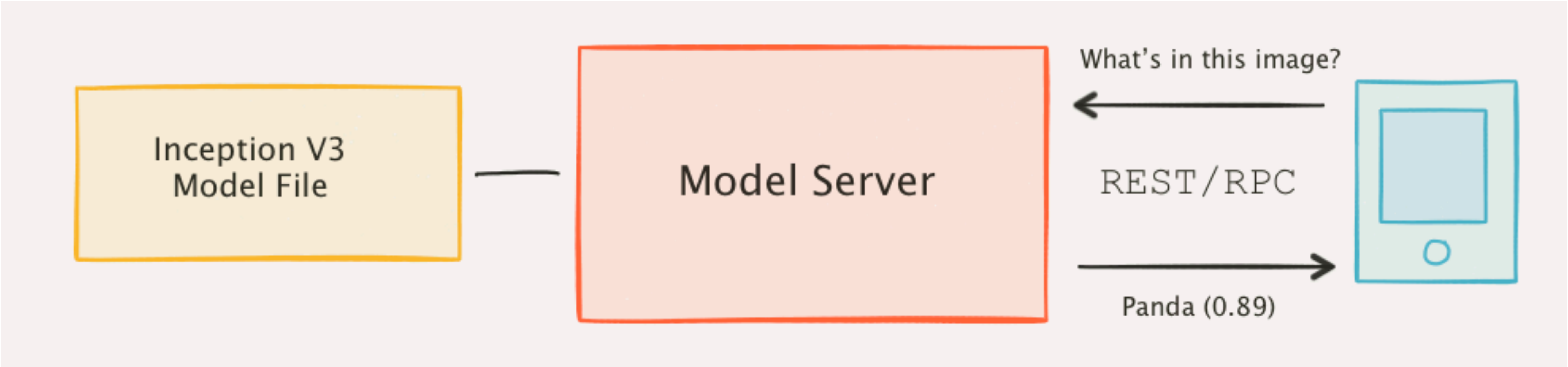
There are other open source tools for versioning

MLflow



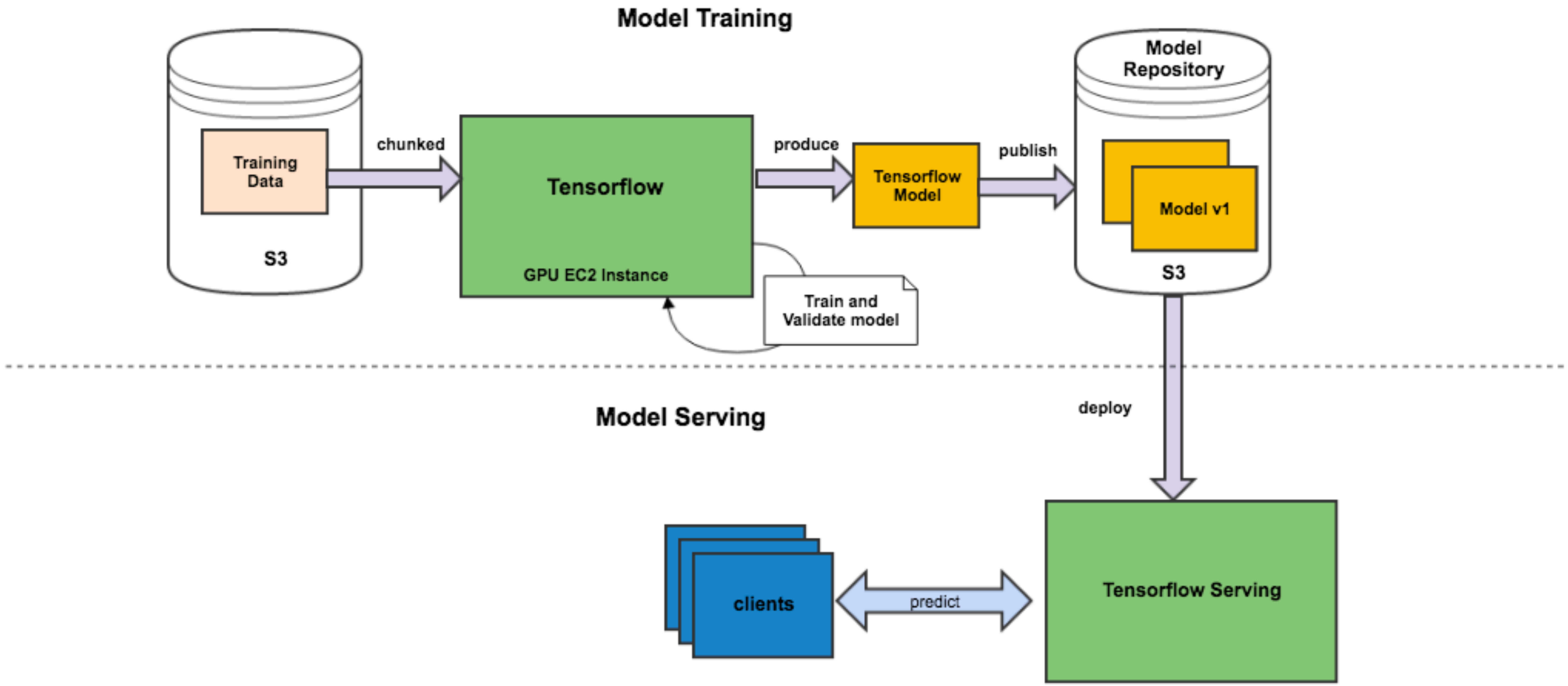
Model Serving

Abstract level



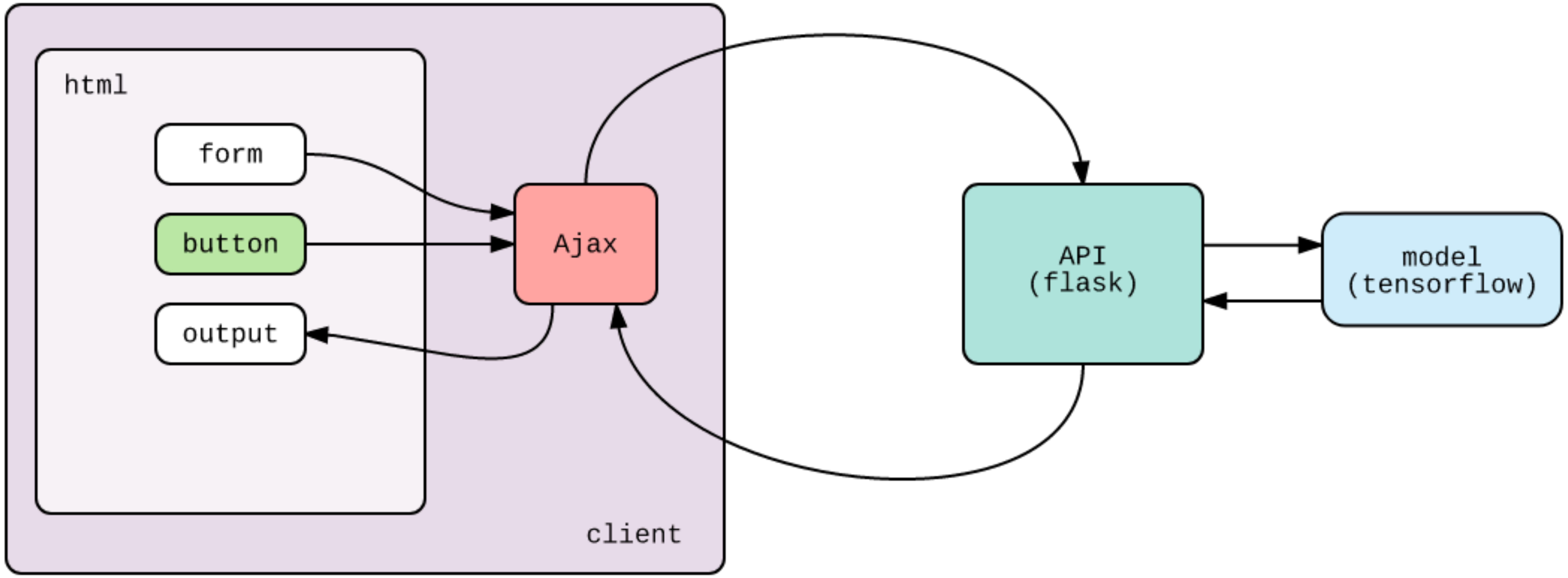
Model Serving

TF Serving



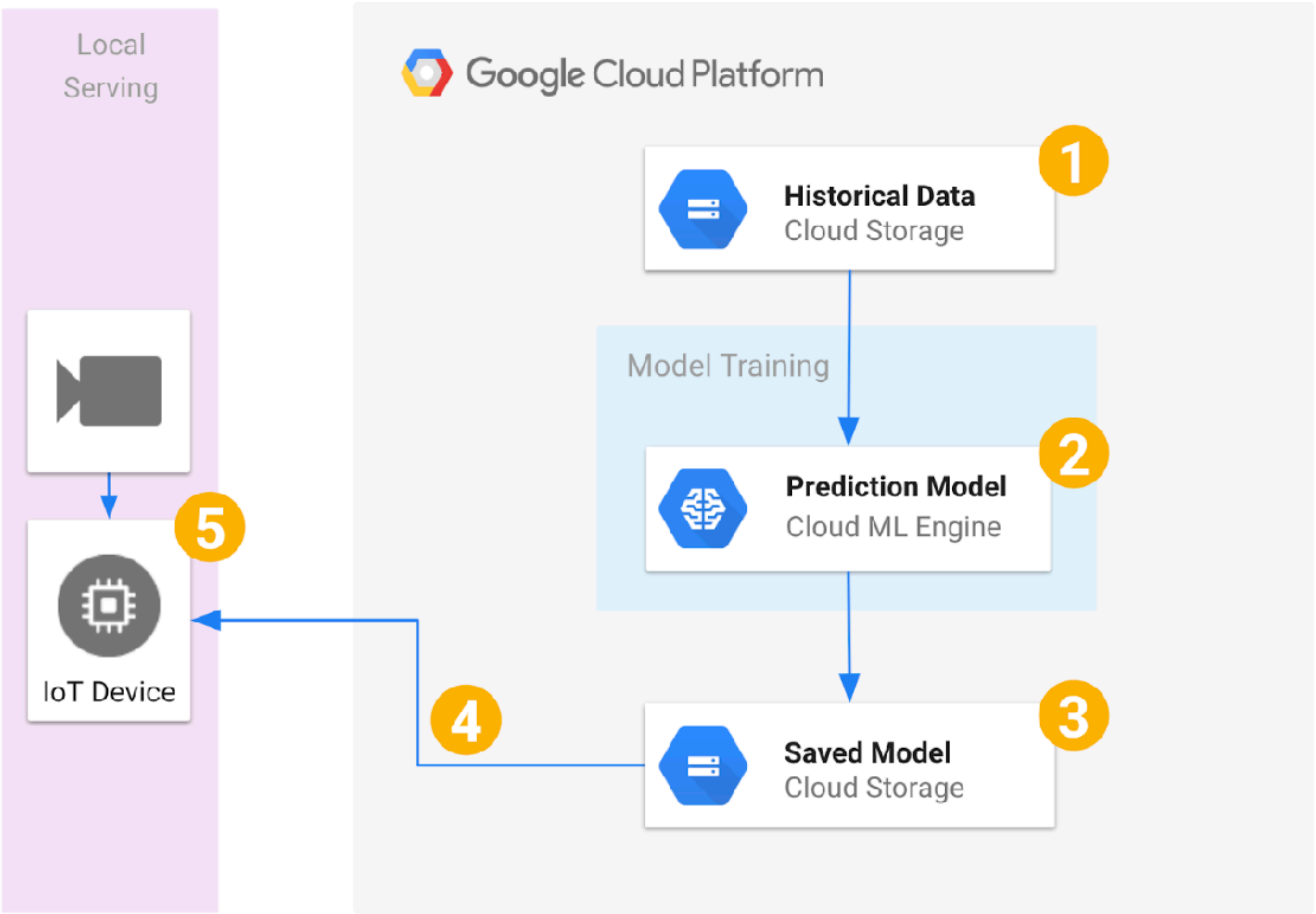
Model Serving

Web app



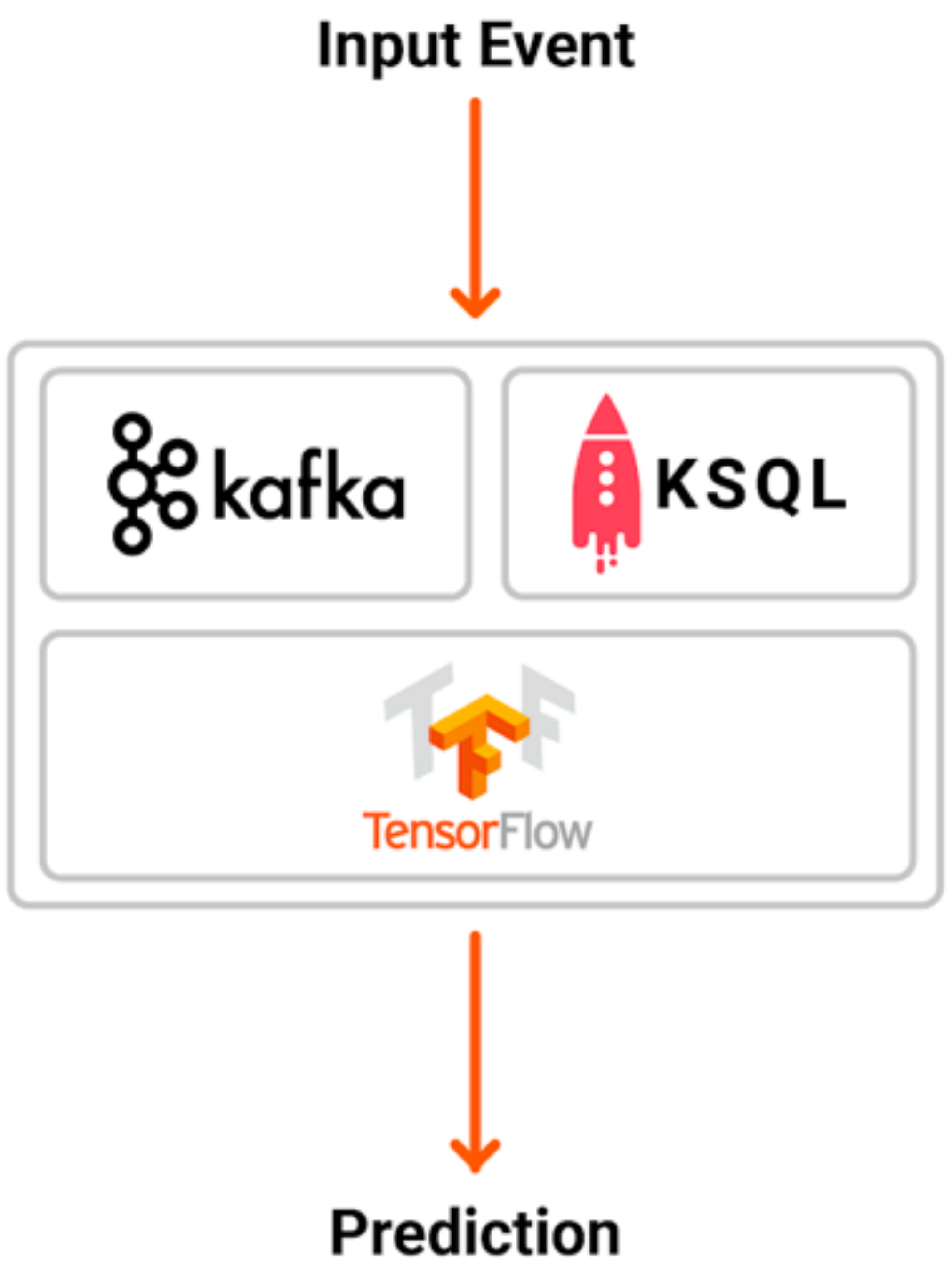
Model Serving

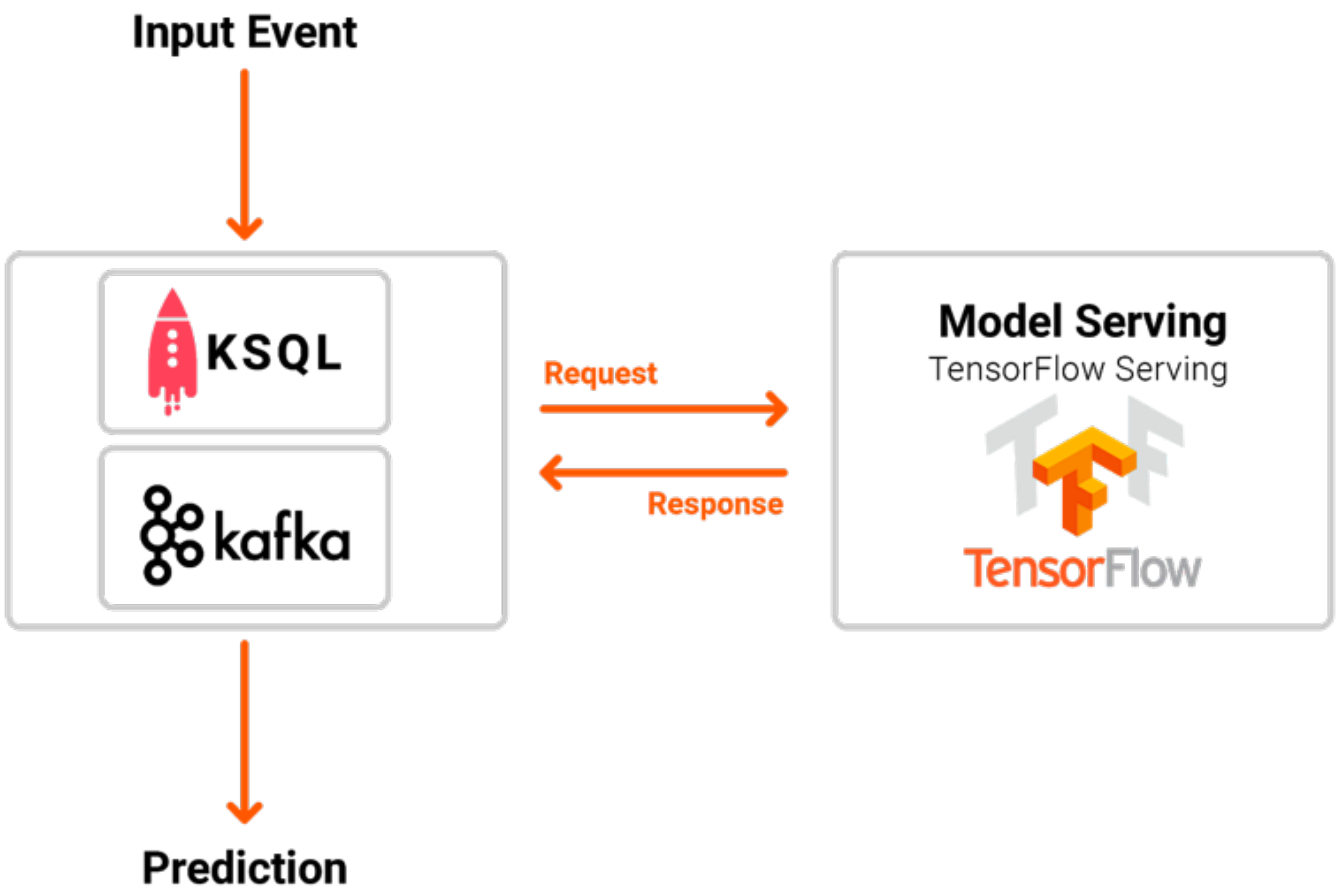
Internet of Thing



Model Serving

Stream Processing System





Model Serving

Embedded model

- Simple approach
- You treat the model artifact as a dependency that is built and packaged within the consuming application.
- You can treat the application artifact and version as being a combination of the application code and the chosen model.

Model Serving

Model deployed as a separate service

- The model is wrapped in a service that can be deployed independently of the consuming applications.
- This allows updates to the model to be released independently, but it can also introduce latency at inference time
- There will be some sort of remote invocation required for each prediction.

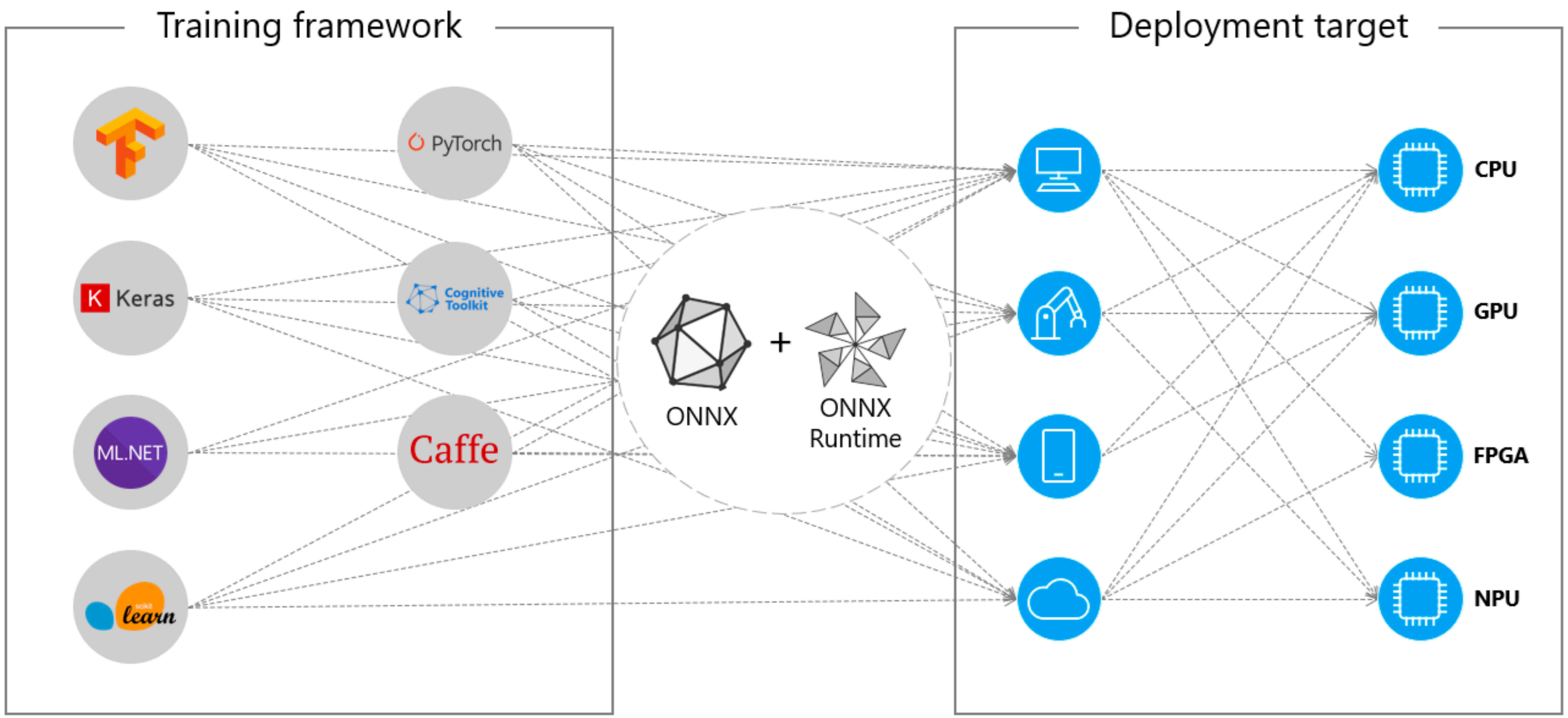
Model Serving

Model published as data

- The model is also treated and published independently,
- But the consuming application will ingest it as data at runtime.
- We have seen this used in streaming/real-time scenarios where the application can subscribe to events that are published whenever a new model version is released, and ingest them into memory while continuing to predict using the previous version.
- Software release patterns such as Canary Releases can also be applied in this scenario.

Export ML models to production environment

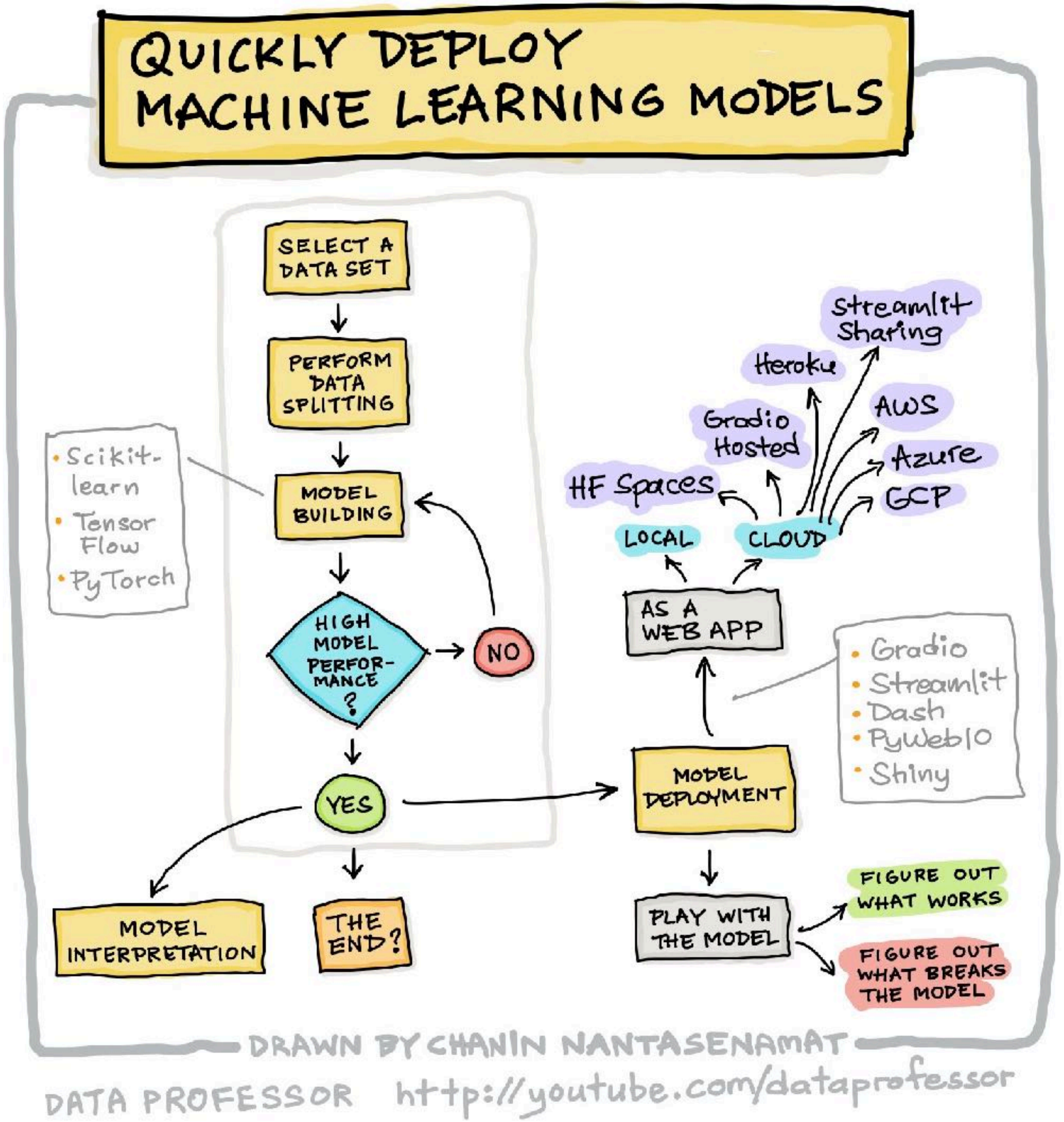
Open Neural Network Exchange



Testing and Quality in Machine Learning

- Regardless of which pattern you decide to use, there is always an implicit contract between the model and its consumers.
- The model will usually expect input data in a certain shape, and if Data Scientists change that contract to require new input or add new features, you can cause integration issues and break the applications using it.
- So testing becomes important.

Continuous Delivery for Machine Learning


QUICKLY DEPLOY MACHINE LEARNING MODELS
SELECT A DATA SET
PERFORM DATA SPLITTING
MODEL BUILDING
• Scikit-learn
• Tensor Flow
• PyTorch
HIGH MODEL PERFORMANCE ?
NO
YES
MODEL INTERPRETATION
THE END?
MODEL DEPLOYMENT
AS A WEB APP
• Gradio
• Streamlit
• Dash
• PyWebIO
• Shiny
LOCAL
CLOUD
HF Spaces
Gradio Hosted
Heroku
Streamlit Sharing
AWS
Azure
GCP
PLAY WITH THE MODEL
FIGURE OUT WHAT WORKS
FIGURE OUT WHAT BREAKS THE MODEL
DRAWN BY CHANIN NANTASENAMAT
DATA PROFESSOR http://youtube.com/dataprofessor

Gradio powering clinical trials of machine learning models

4 Model's mistakes collected

1 Interdisciplinary team build/improve skin cancer classification model

2 They deploy it easily using Gradio

3 Model undergoes clinical validation at multiple sites

Testing Machine Learning Systems

Validating data

- Tests to **validate input data** against the expected schema, or to validate our **assumptions** about its valid values:
  - Values fall within expected ranges
  - Values are not null
- Unit tests to check **features** are calculated correctly:
  - Numeric features are scaled or normalized,
  - One-hot encoded vectors contain all zeroes and a single 1
  - Missing values are replaced appropriately

Testing Machine Learning Systems

Validating component integration

- Test the **integration** between different services:
  - Contract Tests to validate that the expected model interface is compatible with the consuming application.
- Test that the **exported model** still produces the same results:
  - Running the original and the productionized models against the same validation dataset, and comparing the results are the same.

Testing Machine Learning Systems

Validating the model quality

- ML **model performance** is non-deterministic.
- Collect and monitor **metrics** to evaluate a model's performance,
    - Error rates, accuracy
    - Precision, recall
    - AUC, ROC, confusion matrix
- **Threshold Tests** in our pipeline, to ensure that new models don't degrade against a known performance baseline.

Testing Machine Learning Systems

Validating model bias and fairness

- Check how the model performs against **baselines** for specific **data slices**:
  - Inherent bias in the training data where there are many more data points for a given value of a feature (e.g. race, gender, or region) compared to the actual distribution in the real world.
- A tool like **Facets** can help you visualize those slices and the distribution of values across the features in your datasets.

Testing Machine Learning Systems

Integration Test

- When models are **distributed or exported** to be used by a different application,
- The engineered features are **calculated differently** between training and serving time.
- Distribute a **holdout dataset** along with the model artifact, and allow the consuming application team to reassess the model's performance against the holdout dataset after it is integrated.
- This would be the equivalent of a broad **Integration Test** in traditional software development.

Governance process for ML Systems

Experiments Tracking

- To capture and display information that will allow humans to decide if and which model should be promoted to production.
- It is common that you will have multiple experiments being tried in parallel, and many of them might not ever make it to production.
- The code for many of these experiments will be thrown away, and only a few of them will be deemed worthy of making it to production.
- Different Git branches to track the different experiments in source control.
- Tools such as DVC can fetch and display metrics from experiments running in different branches or tags, making it easy to navigate between them.

Governance process for ML Systems

MLflow Tracking web UI

Model Deployment

Multiple models

- More than one model performing the same task.
  - Train a model to predict demand for each product.
- Deploying the models as a separate service might be better for consuming applications to get predictions with a single API call.
- You can later evolve how many models are needed behind that Published Interface.

Model Deployment

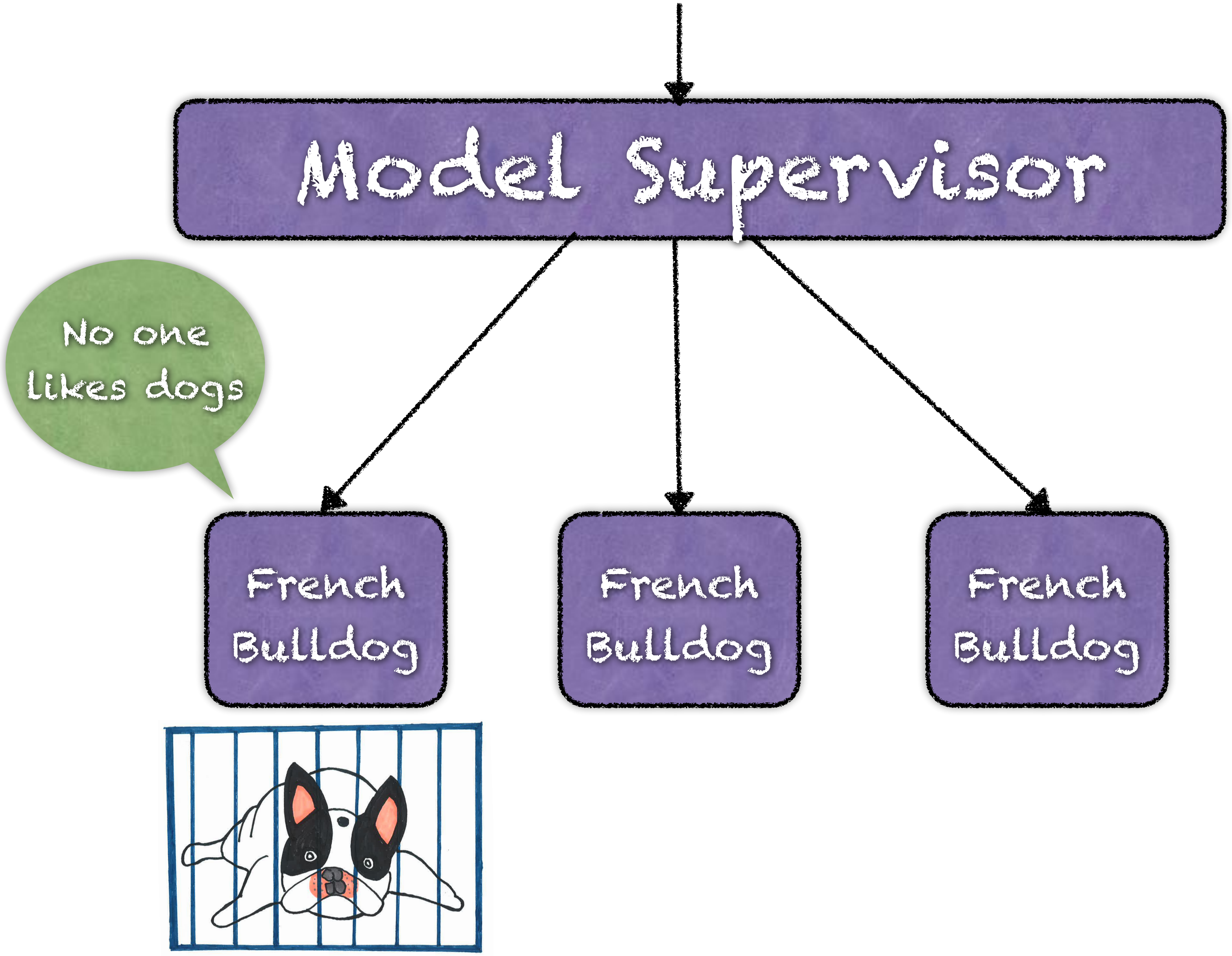
Shadow models

- Deploy the new model side-by-side with the current one, as a shadow model
- Send the same production traffic to gather data on how the shadow model performs before promoting it into the production.

Model Deployment

Competing models

- Multiple versions of the model in production — like an A/B test
  - Infrastructure and routing rules required to ensure the traffic is being redirected to the right models.
  - To gather enough data to make statistically significant decisions, which can take some time.
- Evaluating multiple competing models is Multi-Armed Bandits,
  - To define a way to calculate and monitor the reward associated with using each model.



Model Supervisor
No one likes dogs
French Bulldog
French Bulldog
French Bulldog

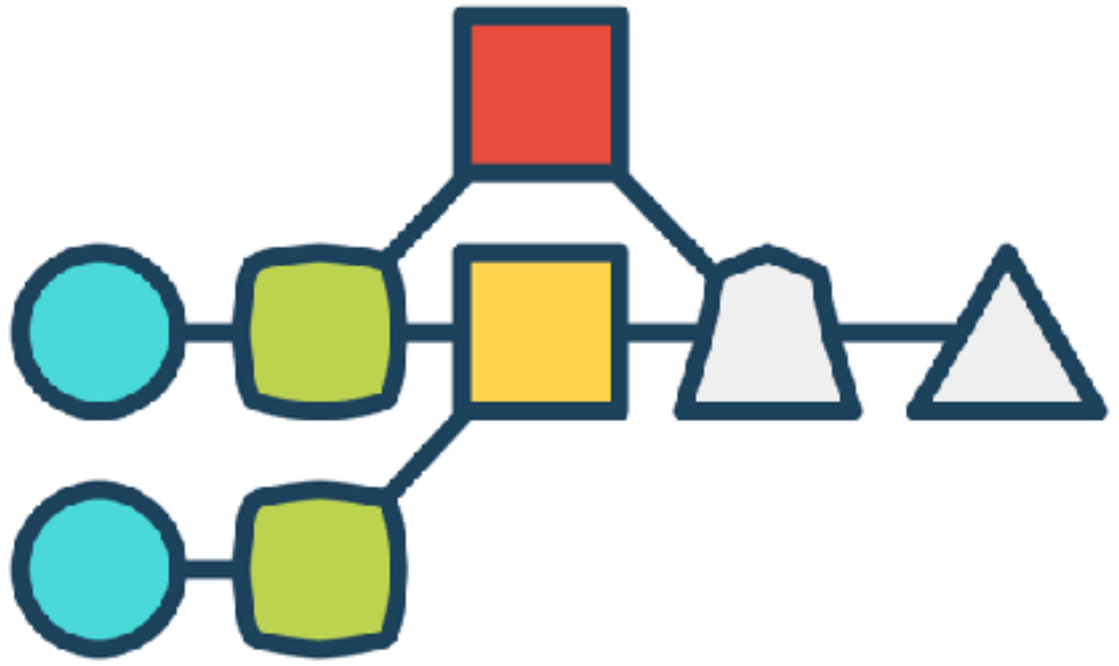
Model Deployment

Online learning models

- To use algorithms and techniques that can continuously improve its performance with the arrival of new data.
- Constantly learning in production.
- Extra complexities, as versioning the model as a static artifact won't yield the same results if it is not fed the same data.
- You will need to version not only the training data, but also the production data that will impact the model's performance.

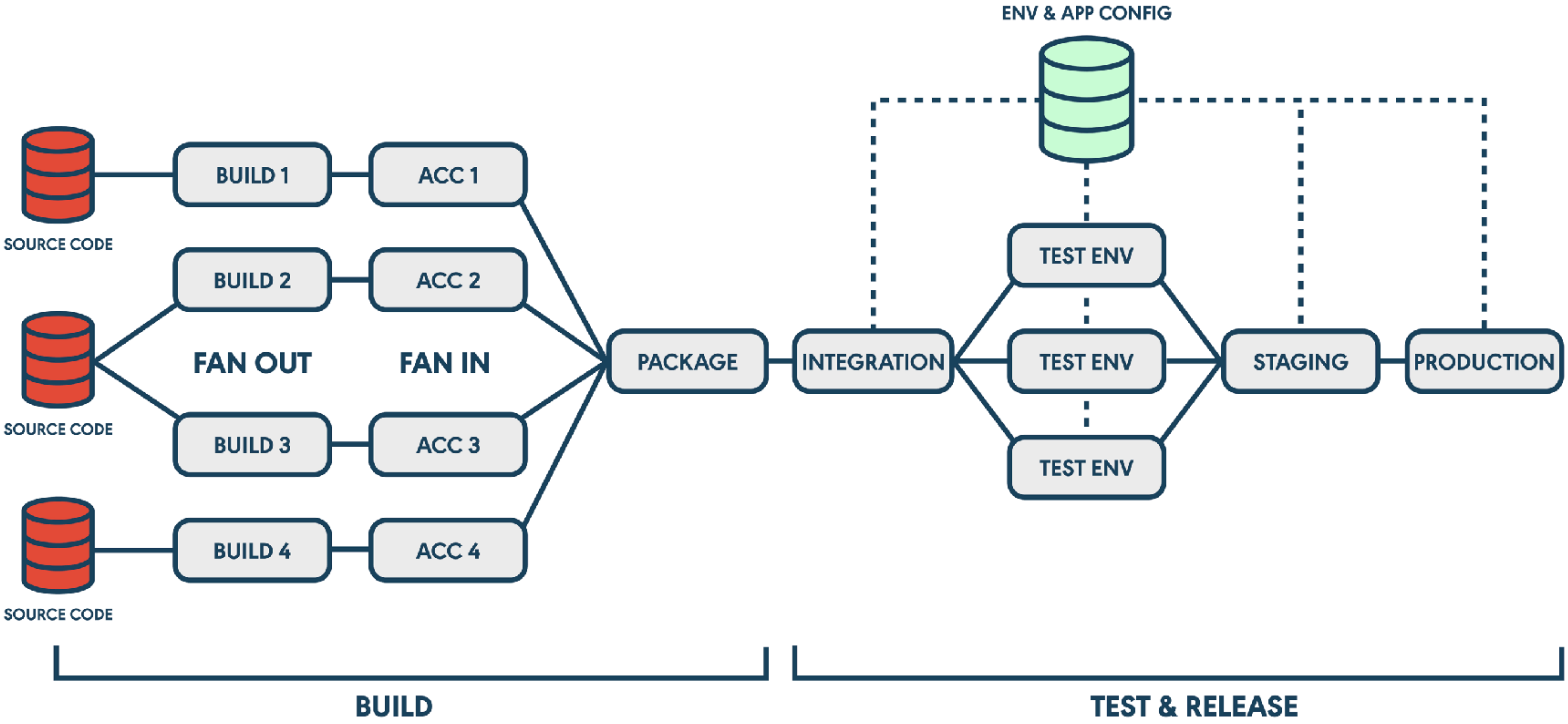
Orchestration in ML Pipelines

- Provisioning of infrastructure and the execution of the ML Pipelines to train and capture metrics from multiple model experiments
- Building, testing, and deploying Data Pipelines
- Testing and validation to decide which models to promote
- Provisioning of infrastructure and deployment of models to production



Continuous Integration and Delivery

GoCD

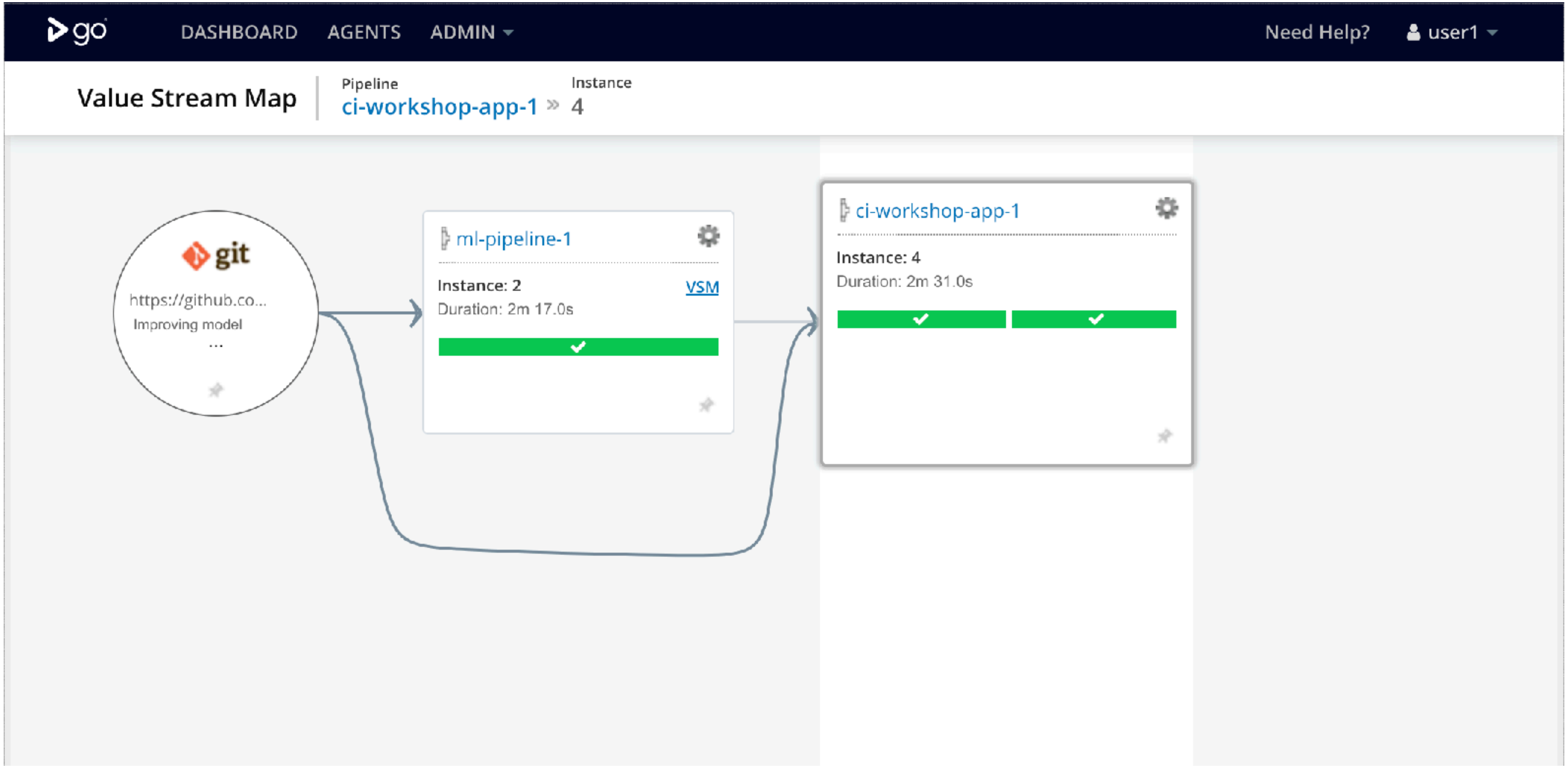


A Continuous Delivery Scenario for ML

1. **Machine Learning Pipeline**:
    - To train and evaluate ML models
    - To execute threshold test to decide if the model can be promoted or not
    - *dvc push* to publish it as an artifact
2. **Application Deployment Pipeline**:
    - To build and test the application code
    - To fetch the promoted model from the upstream pipeline using *dvc pull*
    - To package a new combined artifact that contains the model and the application as a Docker image
    - To deploy them to a production cluster

Continuous Delivery for Machine Learning

Combining Machine Learning Pipeline and Application Deployment Pipeline



ML Model Monitoring

How models perform in production and rollback mechanisms

- **Model inputs**:
  - What data is being fed to the models, identifying training-serving skew.
- **Model outputs**:
  - What predictions and recommendations are the models making from these inputs, to understand how the model is performing with real data.

ML Model Monitoring

How models perform in production and rollback mechanisms

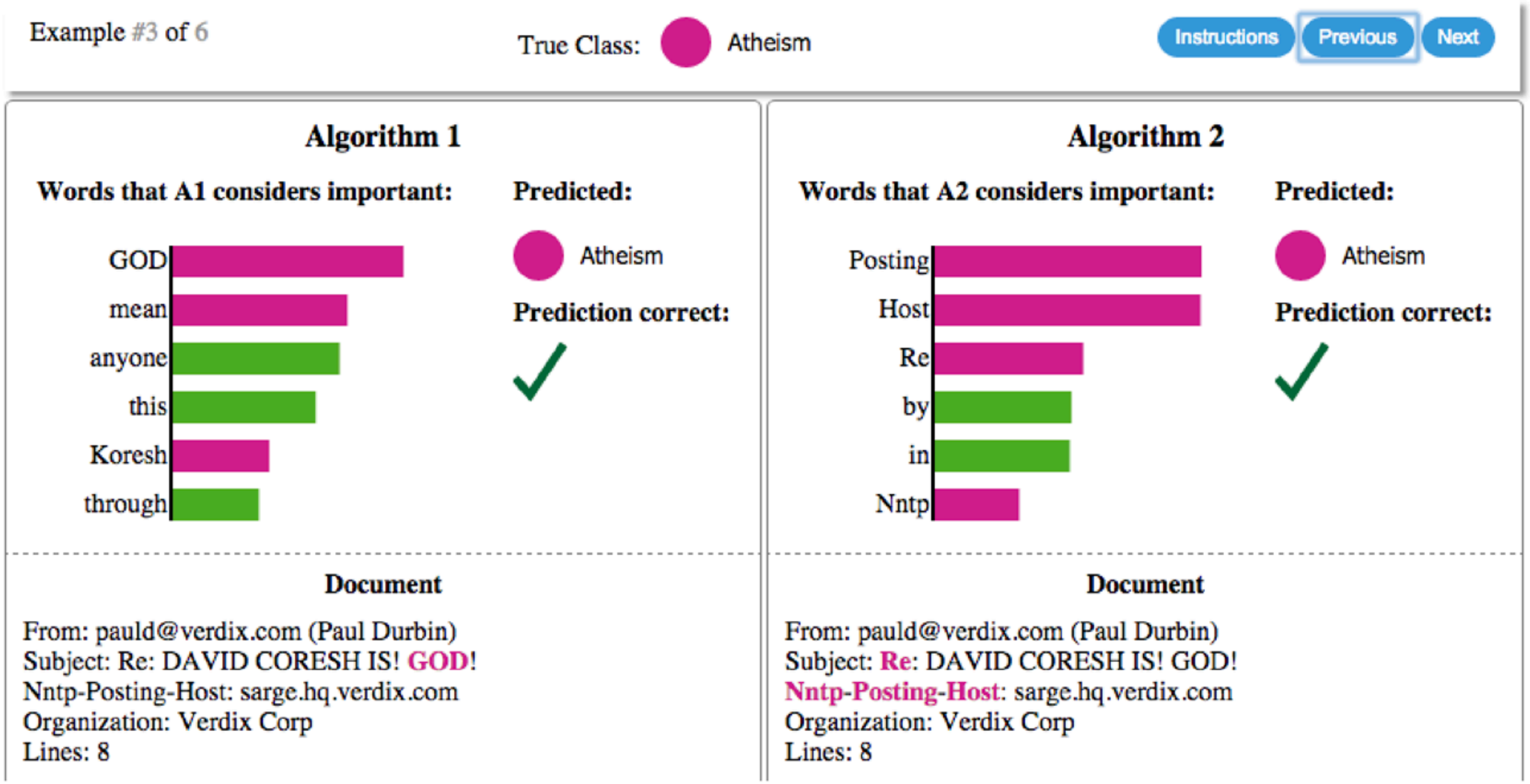
- **Model interpretability outputs**:
  - Metrics such as model coefficients, ELI5, or LIME outputs that allow further investigation to understand how the models are making predictions to identify potential overfit or bias that was not found during training.

hi there, i am here looking for some help. my friend is a interic
graphics software on pc. any suggestion on which software tc
sophisticated software(the more features it has,the better)

y=0 (probability **0.000**) top features			y=1 (probability **0.100**) top features			y=2 (probability **0.900**) top features		
Contribution?	**Feature**	**Value**	**Contribution?**	**Feature**	**Value**	**Contribution?**	**Feature**	**Value**
+0.301	<BIAS>	1.000	+0.427	<BIAS>	1.000	+0.289	hue	0.670
+0.064	color_intensity	8.500	+0.033	proline	630.000	+0.272	<BIAS>	1.000
+0.004	malic_acid	4.600	+0.022	od280/od315_of_diluted_wines	1.920	+0.095	color_intensity	8.500
-0.018	alcalinity_of_ash	25.000	+0.009	alcalinity_of_ash	25.000	+0.083	flavanoids	0.960
-0.044	total_phenols	1.980	+0.006	total_phenols	1.980	+0.067	proline	630.000
-0.055	flavanoids	0.960	-0.003	proanthocyanins	1.110	+0.056	malic_acid	4.600
-0.100	proline	630.000	-0.010	alcohol	13.400	+0.038	total_phenols	1.980
-0.153	hue	0.670	-0.028	flavanoids	0.960	+0.010	alcohol	13.400
			-0.060	malic_acid	4.600	+0.009	alcalinity_of_ash	25.000
			-0.137	hue	0.670	+0.003	proanthocyanins	1.110
			-0.160	color_intensity	8.500	-0.022	od280/od315_of_diluted_wines	1.920

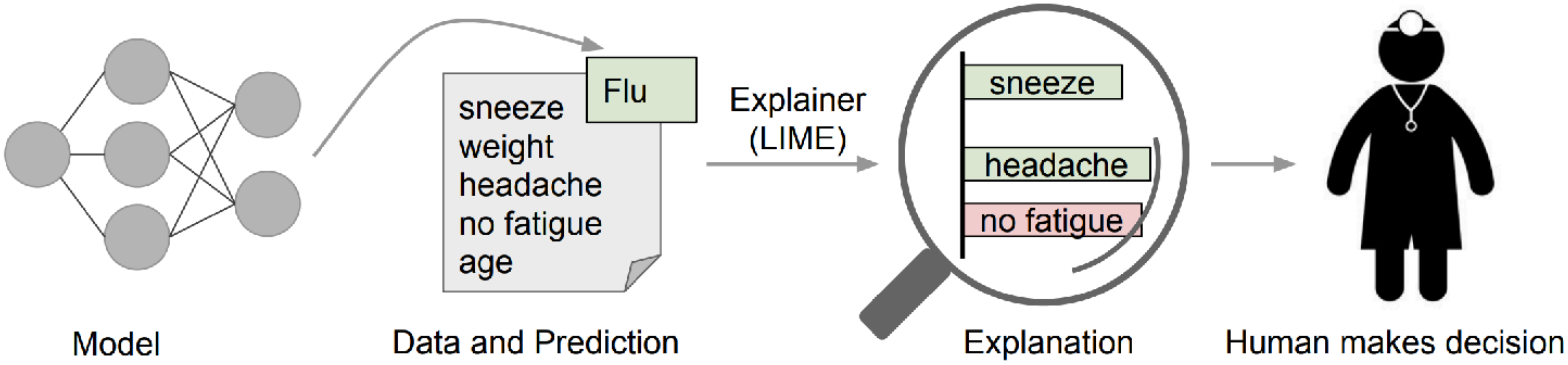
“Why Should I Trust You?”

Explaining the Predictions of Any Classifier



Explaining individual predictions

A model predicts that a patient has the flu, and LIME highlights the symptoms in the patient's history that led to the prediction



ML Model Monitoring

How models perform in production and rollback mechanisms

- **Model outputs and decisions**:
  - What predictions our models are making given the production input data, and also which decisions are being made with those predictions.
  - Sometimes the application might choose to ignore the model and make a decision based on pre-defined rules (or to avoid future bias).

ML Model Monitoring

How models perform in production and rollback mechanisms

- **User action and rewards**:
  - Based on further user action, we can capture reward metrics to understand if the model is having the desired effect.
  - For example, if we display product recommendations, we can track when the user decides to purchase the recommended product as a reward.

A pipeline for model monitoring
ELK

- **Elasticsearch**: an open source *search* engine.
- **Logstash**: an open source data collector for unified *logging* layer.
- **Kibana**: an open source web UI that makes it easy to explore and *visualize* the data indexed by Elasticsearch.

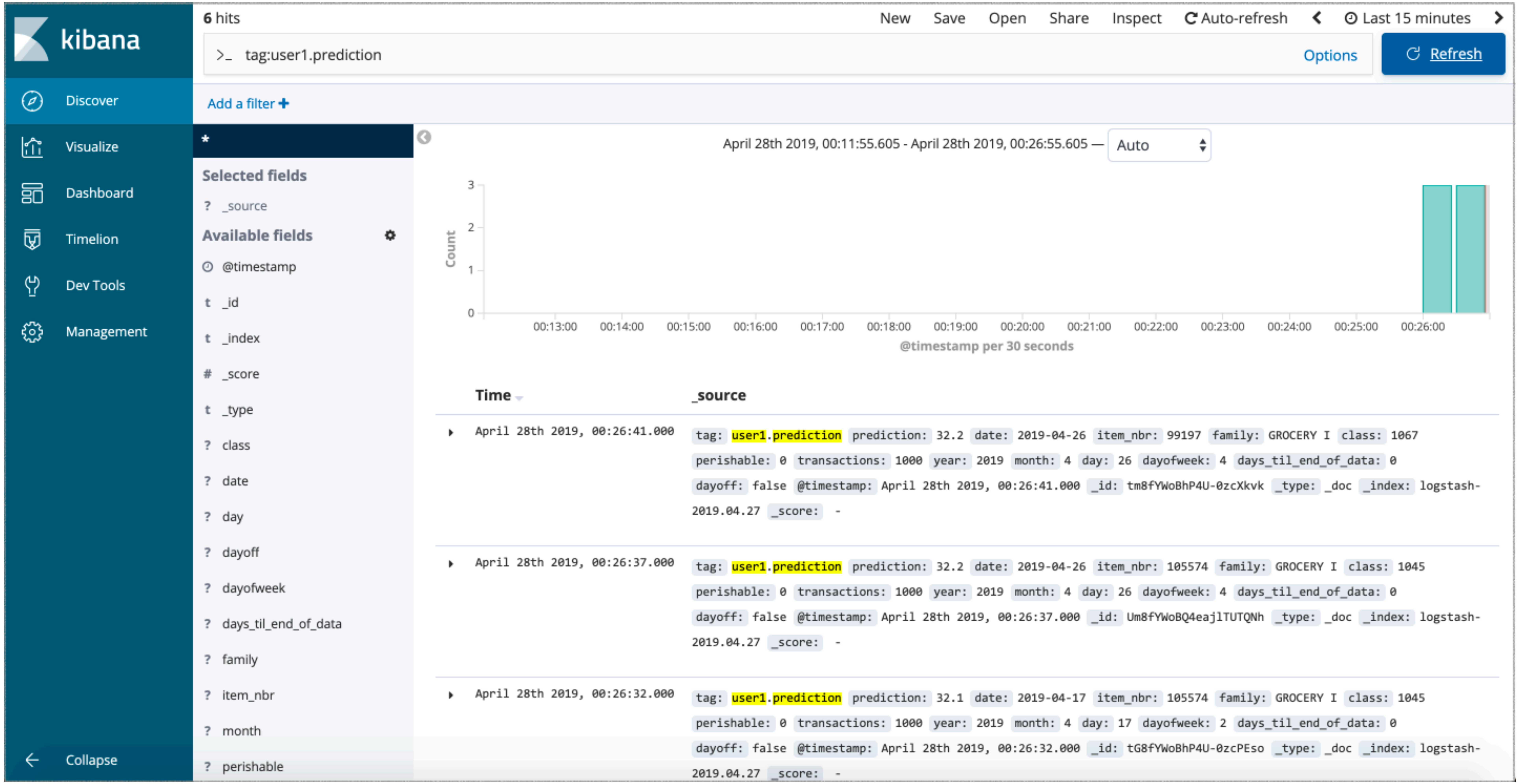
A pipeline for model monitoring
ELK

Logging

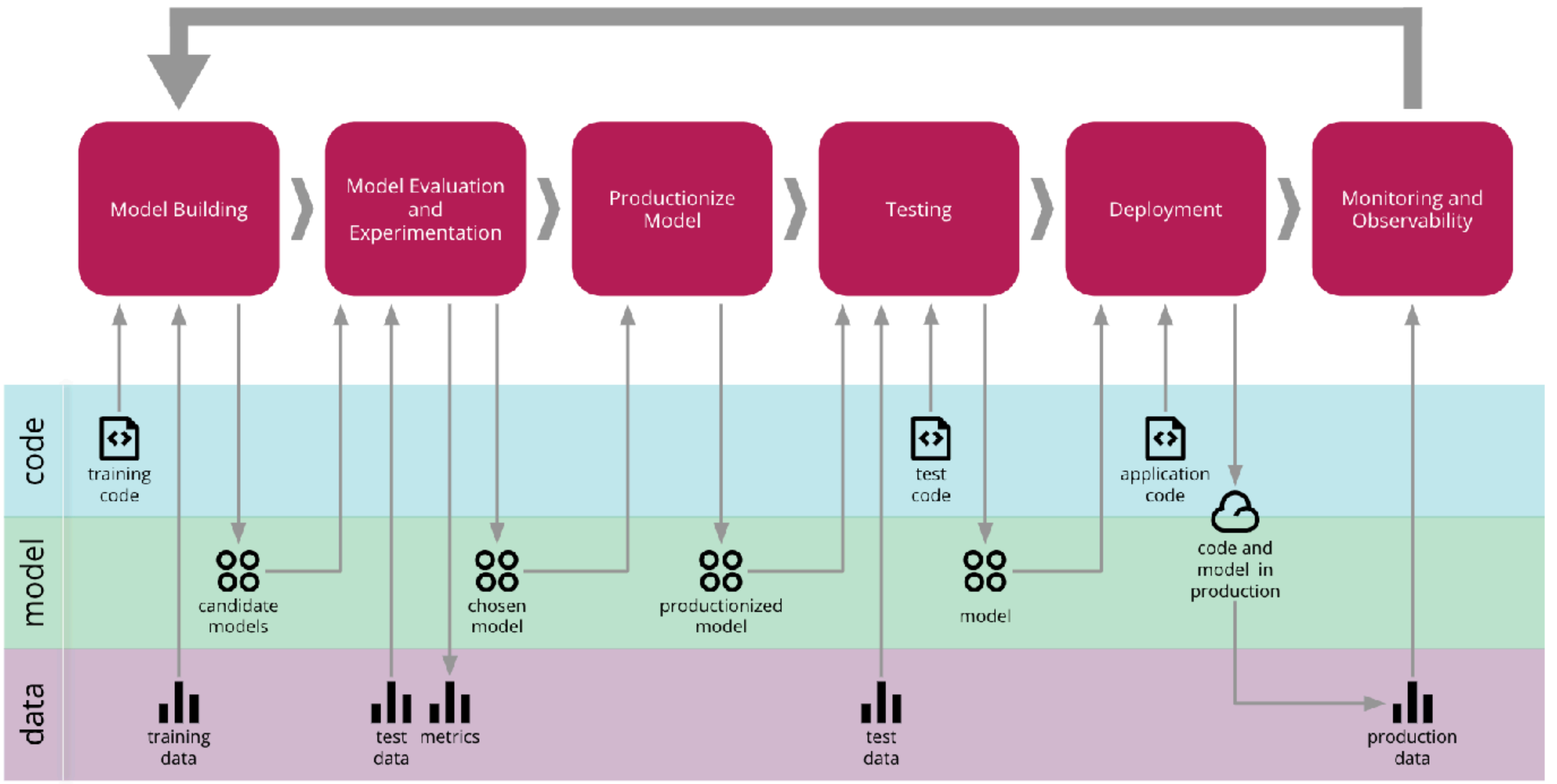
predict_with_logging.py...

```
df = pd.DataFrame(data=data, index=['row1'])
df = decision_tree.encode_categorical_columns(df)
pred = model.predict(df)
logger = sender.FluentSender(TENANT, host=FLUENTD_HOST, port=int(FLUENTD_PORT))
log_payload = {'prediction': pred[0], **data}
logger.emit('prediction', log_payload)
```

A pipeline for model monitoring



An End-to-End ML Building Process



Continuous Delivery for Machine Learning

Machine Learning Systems

Next class: Foundations of Neural Networks and Learning

PyTorch: An Imperative Style, High-Performance Deep Learning Library

Adam Paszke
University of Warsaw
adam.paszke@gmail.com

Sam Gross
Facebook AI Research
sgross@fb.com

Francisco Massa
Facebook AI Research
fmassa@fb.com

Adam Lerer
Facebook AI Research
alerer@fb.com

James Bradbury
Google
jekbradbury@gmail.com

Gregory Chanan
Facebook AI Research
gchanan@fb.com

Trevor Killeen
Self Employed
killeent@cs.washington.edu

Zeming Lin
Facebook AI Research
zlin@fb.com

Natalia Gimelshein
NVIDIA
ngimelshein@nvidia.com

Luca Antiga
Orobix
luca.antiga@orobix.com

Alban Desmaison
Oxford University
alban@robots.ox.ac.uk

Andreas Köpf
Xamla
andreas.koepf@xamla.com

Edward Yang
Facebook AI Research
ezyang@fb.com

Zach DeVito
Facebook AI Research
zdevito@cs.stanford.edu

Martin Raison
Nabla
martinraison@gmail.com

Alykhan Tejani
Twitter
atejani@twitter.com

Sasank Chilamkurthy
Qure.ai
sasankchilamkurthy@gmail.com

Benoit Steiner
Facebook AI Research
benoitsteiner@fb.com

Lu Fang
Facebook
lufang@fb.com

Junjie Bai
Facebook
jbai@fb.com

Soumith Chintala
Facebook AI Research
soumith@gmail.com

TensorFlow: A System for Large-Scale Machine Learning

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *Google Brain*

https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

This paper is included in the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16).

November 2–4, 2016 • Savannah, GA, USA

ISBN 978-1-931971-33-1