Software Engineering for AI-Enabled Systems

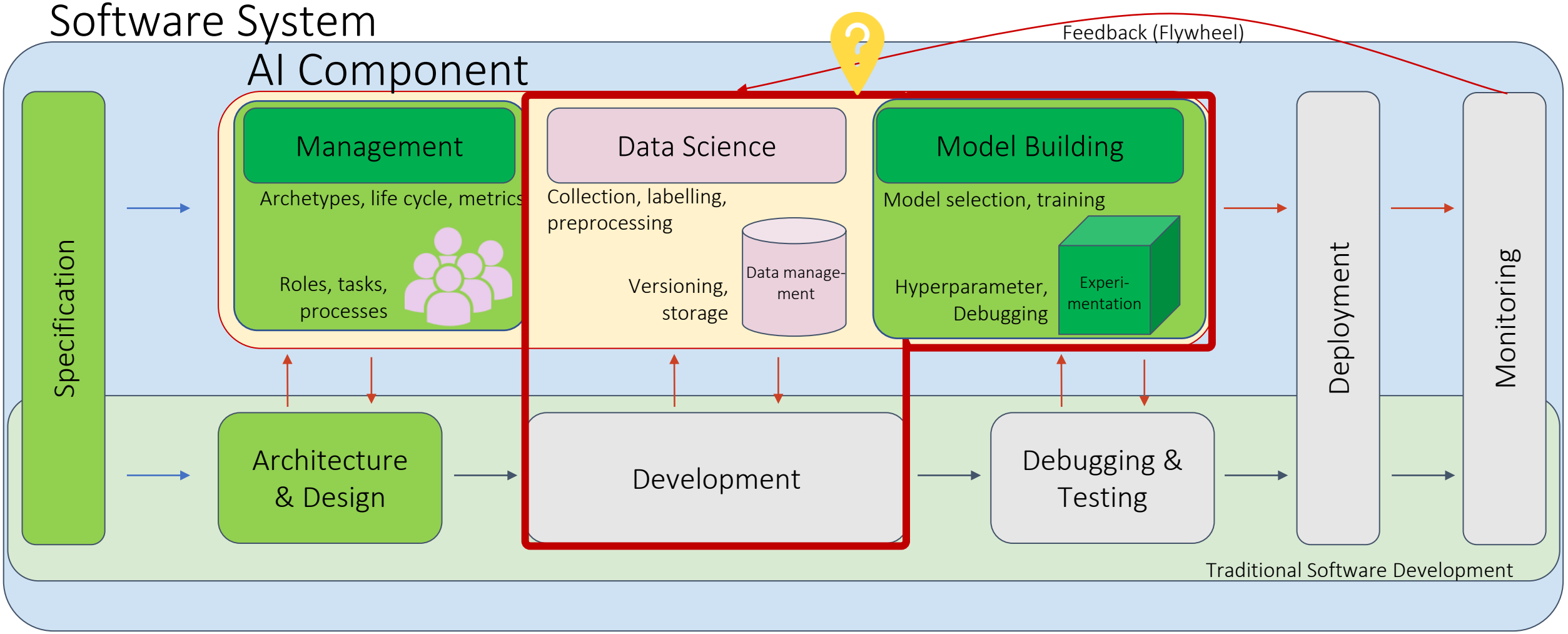
SOFTWARE SYSTEME

UNIVERSITÄT LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund
Software Systems

HeyGen

Software System
AI Component



How to develop an AI system, including the data science process, coding, and experimentation?

- How to store and version data?
- How to use different storage approaches?
- How to collect, label, and preprocess data?
- What tools and frameworks exist to automatically track and store experimentation runs?
- How to make experiments reproducible and how to automate the experimentation runs?

Topic 1:
Developing an AI System

TL;DR:

- AI Project Life Cycle
- Model-centric vs. data-centric approach
- Notebooks
- Data management and versioning
- Experimentation

Learning Goals

Know how to approach an AI project from data management over labelling to experimentation and development

Select from alternatives to realize any stage

Know important tools and concepts of stages and how they connect

Defining an automation of the process in a form of a pipeline

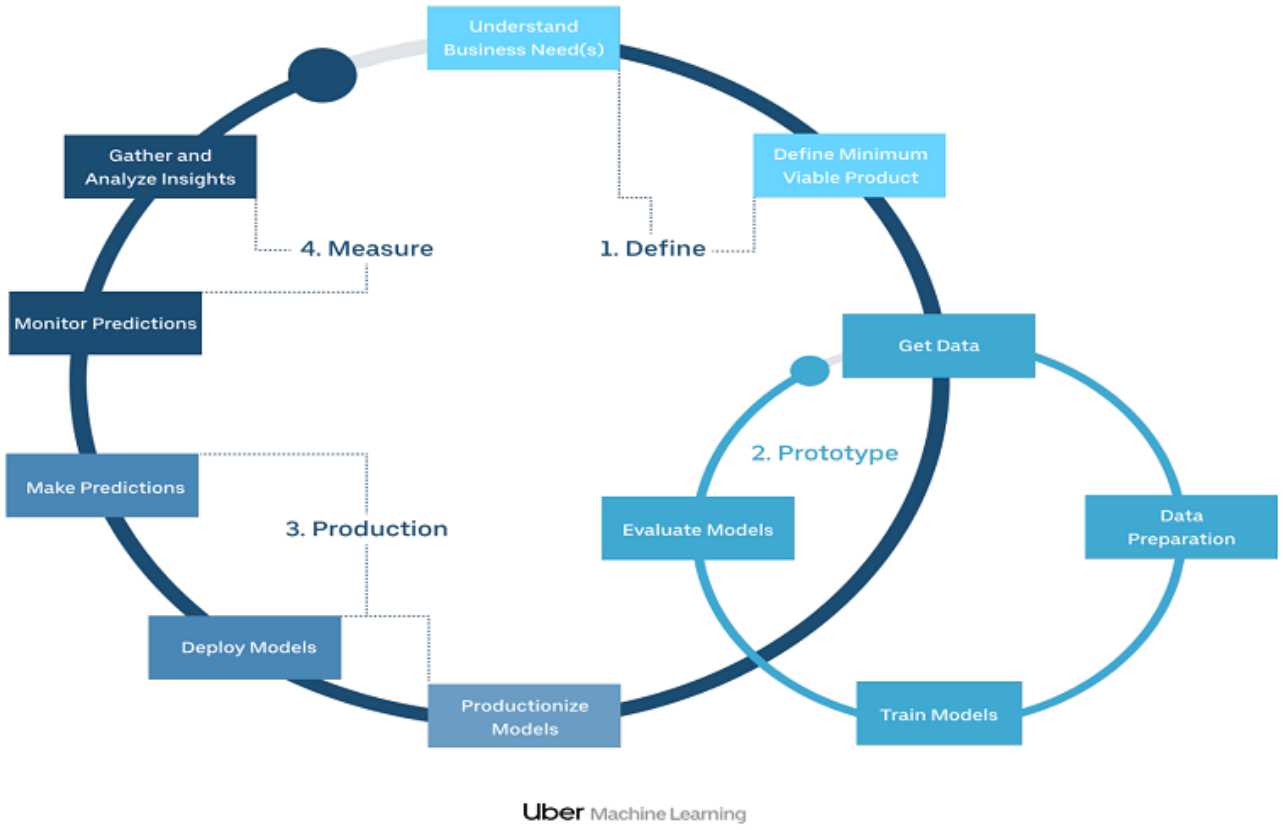
Model-Centric vs. Data-Centric Development

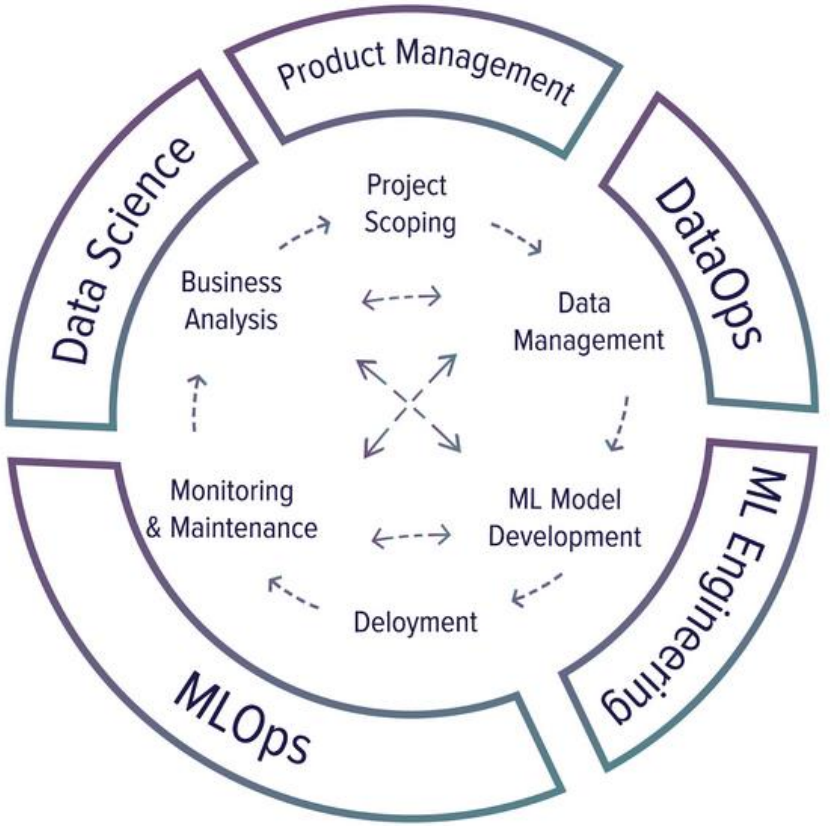
Model-centric development: Focuses development time and resources on the algorithms to make the model more accurate. Focus of research and industrial practice. Great success with Deep Learning, but reached a stale.

Data-centric development: Focuses development time and resources on the data set to make the model more accurate via larger, more curated, better distributed data. Switch now in industrial practice. Great success when there is little progress with algorithms.

	Steel defect detection	Solar panel	Surface inspection
Baseline	76.2%	75.68%	85.05%
Model-centric	+0% (76.2%)	+0.04% (75.72%)	+0.00% (85.05%)
Data-centric	+16.9% (93.1%)	+3.06% (78.74%)	+0.4% (85.45%)

End-to-End Pipeline at Uber





Credit: Chip Huyen

Steps to Develop an AI System

1. Revisit requirements with focus on
   a. The problem that is trying to be solved
   b. The environment in which the system operates
   c. How success of the AI is measured
   d. How data is being collected and under which terms
2. Select the appropriate implementation of the intelligence (topic 6 (validity); partially outside scope)
   a. Do we have hard or easy problems?
   b. Do we need to interpret the results or are they consumed by software?
3. Implement the AI component (here ML) (this topic!)
   a. How to store and preprocess the data?
   b. How to experiment to systematically find the best model?
   c. How to evaluate the model correctly?
   d. How to test and debug the model?
4. Deploy the model to production (topic 8/9)
   a. What deployment modes exist and how to handle multiple model versions?
   b. How to monitor the model and log errors?
   c. How to obtain feedback from users?

(1) Requirements: Environment

Clarify the following items to design the AI:

- Input to the system: device, software, or user origin; stream, query, pull, push
- Form of input: Image, video, text, sensor data, user query
- Deployment location: desktop, server, sensor, indoor, outdoor, setup process required
- Input modification: preprocessing of raw data, error correction, re-collection available or not
- Usage scenario: type of ML model, UX design or interpretability required
- HW resources: available computation power, energy usage
- Non-functional: inference time, bandwith

(1) Requirements: Success / Goal

Specify evaluation metric and thresholds to be reached

- Detection accuracy, prediction accuracy
- Maximum number of mistakes per day
- Success/Unsuccessful rate per user interaction
- Cost of mistake

(1) Requirements: Obtain Data

Without data, there is no ML intelligence (heuristics and code may work). Start early on with data collection (best even before the project):

Bootstrap:

- Scrape data from the web (images, GitHub, Google BigQuery, government portals, statistics collection portals, competitions, data repositories)
- Collect own data (take pictures, e.g., from production line, sensor data, interviews, Amazon Mechanical Turk, surveys, experiments)
- Find/Buy existing data sets (curated data sets of companies, cooperate with authorities, e.g., medical facilities)
- Assess the amount of data required for the task
- Plan how to label the data if necessary (tools, humans, outsourcing, etc.)

(1) Requirements: Obtain Data

Usage:

- Collect data from deployed intelligence (input-output pairs, context, etc.)
- Get explicit user feedback of AI-enabled SW system
- Develop explicit data collection software (see Tesla beta program, voluntary fine-tuning of speech recognition, calibration steps, etc.)
- Be transparent to the user for collecting the data

(2) Select the appropriate implementation

Internally, we do something like that: `prediction = model.getPrediction(context)`

How to implement `getPrediction()` in an appropriate form?

(recap) Criteria:

- Minimize mistakes of the implementation
- Interpretability of the implementation
- Match the skills of your team (especially when the production is maintained by others)
- Match deployment environment and resource constraints

(2) Code Implementation

Approach: Codify the model (e.g., in form of explicit rules or heuristics)

Applicable if project:

- Is in an experimentation state and we do not know the difficulty of the problem
- Requires a cheap and quick starting point
- Must provide a fallback or safeguard (e.g., hardcode max values for suggestions)
- Divide the problem in easy parts (code) and hard parts (ML)
- Fast and efficient performing intelligence

Not suggested if:

- Problem changes quickly
- Heuristics are too complex

(2) Lookup Tables

Approach: Lookup tables match a specific context to a prediction

Applicable if:

- We need a quick fallback scenario (i.e., override other intelligence implementations)
- Some contexts are often triggered with a clear result (i.e., we cache the results)
- Pre-processing of use case scenarios is possible
- Ships and deploy predictions to other environments
- Lock and freeze intelligence (e.g., specific models) by storing their I/O pairs

Not applicable if:

- Meaning of context changes over time (e.g., for language)
- Many contexts such that the table gets too large

(2) ML Models

Approach: Algorithm extracts patterns from data and encodes them usually in a mathematical formula

Applicable in most cases… (see lectures on validity and ethics)

Can handle:

- Preprocess data to (complex, aggregated) features
- Many features in arbitrary ways
- Scale to millions or billions of data points
- Find even feint or rare patterns and events

Not applicable if general intelligence or common sense is required!

(3) Develop and Evaluate in an iterative Process

Brief overview (more on that later):

- Specify evaluation/testing data and store it elsewhere
  - Consider randomness vs. controlling factors to ensure non-bias
  - Mind data distribution of input features, rare cases, and cost of mistakes
- Establish experimentation tracking (see experimentation lecture)
- Perform model evaluation in environment as close as production
- Automate reporting and analysis of the results, including versioning of model, data, and environment
- Debug and test your AI component (see extra lecture)

(4) Deploy Model to Production

Aspects need to be considered early on (see extra lectures):

- Deployment targets (cloud, desktop, edge) with HW constraints
- Automation of deployment (when triggered, which tools)
- Handling multiple model versions (how to update a model without disrupting the service and how to test a new model in production)
- Monitor live data (tools and frameworks) and obtain feedback

AI-Development Best Practices by Google

https://developers.google.com/machine-learning/guides/rules-of-ml

Rule #1: Don’t be afraid to launch a product without machine learning (topic 2).

Rule #2: First, design and implement metrics (topics 2&3).

Rule #3: Choose machine learning over a *complex* heuristic.

Rule #4: Keep the first model simple and get the infrastructure right (topic 6).

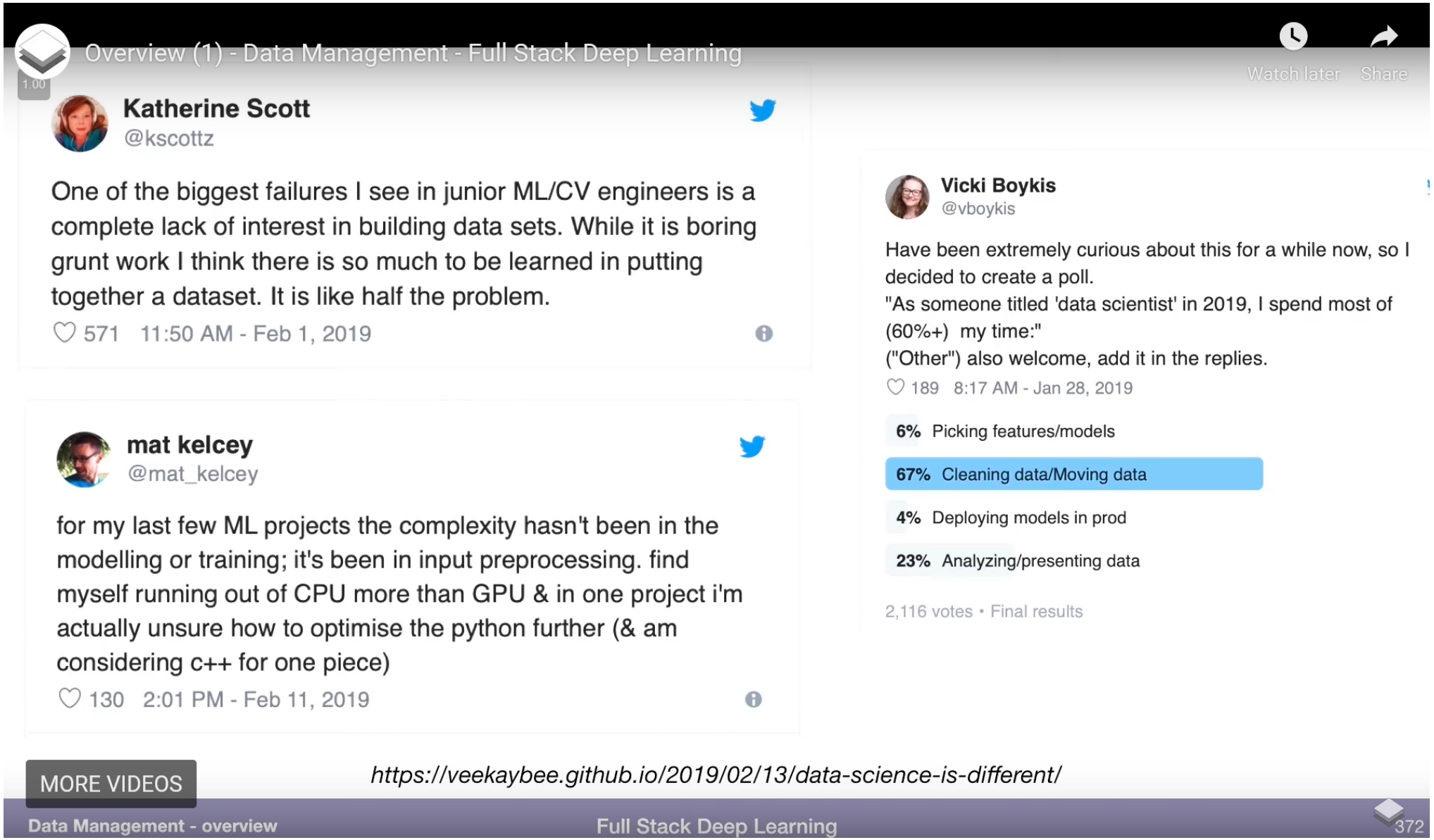
Rule #5: Test the infrastructure independently from the machine learning (topic 7).

Rule #6: Be careful about dropped data (when copying pipelines).

Topic 2:
Data Management & Feature Engineering

TL;DR:

- Data collection and cleaning
- Data labelling
- Storage types
- Versioning
- Data pipeline


Overview (1) - Data Management - Full Stack Deep Learning
1.00
Watch later
Share
Katherine Scott
@kscottz
One of the biggest failures I see in junior ML/CV engineers is a complete lack of interest in building data sets. While it is boring grunt work I think there is so much to be learned in putting together a dataset. It is like half the problem.
♡ 571 11:50 AM - Feb 1, 2019
mat kelcey
@mat_kelcey
for my last few ML projects the complexity hasn't been in the modelling or training; it's been in input preprocessing. find myself running out of CPU more than GPU & in one project i'm actually unsure how to optimise the python further (& am considering c++ for one piece)
♡ 130 2:01 PM - Feb 11, 2019
Vicki Boykis
@vboykis
Have been extremely curious about this for a while now, so I decided to create a poll.
"As someone titled 'data scientist' in 2019, I spend most of (60%+) my time:"
("Other") also welcome, add it in the replies.
♡ 189 8:17 AM - Jan 28, 2019
6% Picking features/models
67% Cleaning data/Moving data
4% Deploying models in prod
23% Analyzing/presenting data
2,116 votes • Final results
MORE VIDEOS
https://veekaybee.github.io/2019/02/13/data-science-is-different/
Data Management - overview
Full Stack Deep Learning
372

Data Assessment: What data do I need?

Find out what is needed to train / develop the AI component

Determine what is already out there

- Which quality does this data have?
- What (legal/economic/technological) constraints do we need to consider?
- Which amount of data do we have to collect to build a model?
- What are the costs in terms of time, money, and effort to collect the data?

Data collection

Existing data sources

No competitive advantage, since everyone has access to it

- Government data
- Competitions (e.g. Kaggle)
- Reports
- Google data set search

Expensive, time-consuming, requires infrastructure

Collecting data from Web pages: Scraping

- Write tool to obtain data from Web pages, social networks, Repositories (e.g., GitHub, StackOverflow), etc.
- Query existing APIs for data access (e.g., StackOverflow)

Install monitoring / logging capabilities

Requires organization agreement, implementation by different teams

- Monitor quality of produces goods; integrate logging capabilities into Web pages to log user behavior
- Install cameras to obtain visual data; distribute sensors to collect environmental data

Data Cleaning

Goal: Improve data quality (validity, accuracy, completeness, consistency, uniformity)

What is needed: Know the sources of bad / errornous data & perform actions to fix them.

Data Cleaning: Validity

How data conforms to defined business rules or constraints.

Data-type constraint: values must be of a specific type (e.g., boolean, int, date)

Range constraints: numbers or dates must be within a range (e.g., age)

Mandatory constraint: empty values are disallowed

Unique constraint: no duplicates allowed in a dataset

Set-membership constraint: values come from a set of values (e.g., gender)

Foreign-key constraint: value must exist in another data source

Regular expression pattern: value must conform to a specific pattern (e.g., email, plz, etc.)

Cross-field validation: Conditions span multiple dimensions/rows (e.g., end data must be after a start date)

Data Cleaning: Further Quality Aspects

Accuracy: How close is the value to the ground truth (e.g., is the address the correct address and not only a valid address? How precise is the information? Does the address contain only the country or city and street?) ?

Completeness: How much data is missing?

Consistency: How much contradiction is within the data (or across another data set)? For example, multiple addresses for customers, date of birth not consistent with driver license, etc.

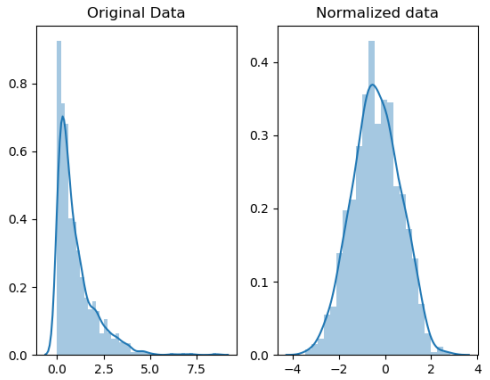
Uniformity: What degree of similarity does the data have in terms of unit of measurements? Or, how much conversion do we need to have the data in a uniform state?

Data Cleaning Workflow

Inspect data: Summary statistics (aka. data profiling); distribution of data in a column (or type of data; frequency of value; range of value; etc.) + visualizations (e.g., histogram of age distribution)

Clean data: Either remove or correct

- Look for irrelevant data (rows: data ranges are not relevant; columns: features/attributes are irrelevant)
- Look for duplicates (e.g., when data comes from different sources, but point to the same object)
- Convert to appropriate types (enumerations can be numbers; but watch out for scale of measurement)
- Uniform entries (e.g., remove white spaces; pad strings to the same length; fix typos in enumerations)
- Fix typos (check whether similar but different values have also other columns with similar values)
- Standardize (e.g., lower/upper case strings; unit of measurement; date format)
- Scale data (transform it to a certain range; percentage 0-100; -1 to 1)
- Normalize data (transform data such that it is normally distributed)
- Missing values: drop row when rare and random; impute (i.e.; recalculate base on other values, e.g., median); hot-deck (fill in with random data; k-nearest neighbor; etc.); flag it to know that the data point had missing data
- Outliers: innocent until proven guilty (safe to keep unless good arguments against)



Verify the actions and consistency (version data!)

Python Cleaning / Data Exploration

```
airbnb.head()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitu
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941

```
airbnb.info()
```

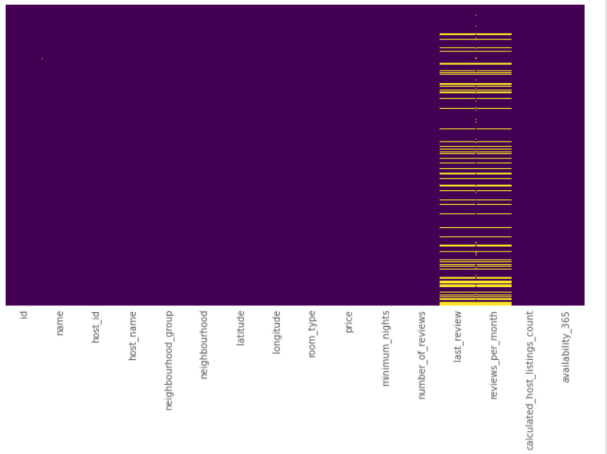
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
```

Heatmap of missing values

Heatmaps are also useful to visualize your missing values, in particular at which point of the data do missing values exists.

```
plt.figure(figsize=(10, 6))
sns.heatmap(airbnb.isnull(), yticklabels=False, cmap='viridis', cbar=False)
```

```
<AxesSubplot:>
```




```
airbnb['price'].describe()
```

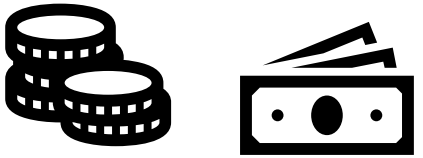
```
count    48892.000000
mean       152.714023
std        240.156106
min          0.000000
25%         69.000000
50%        106.000000
75%        175.000000
max      10000.000000
Name: price, dtype: float64
```

Data Labelling

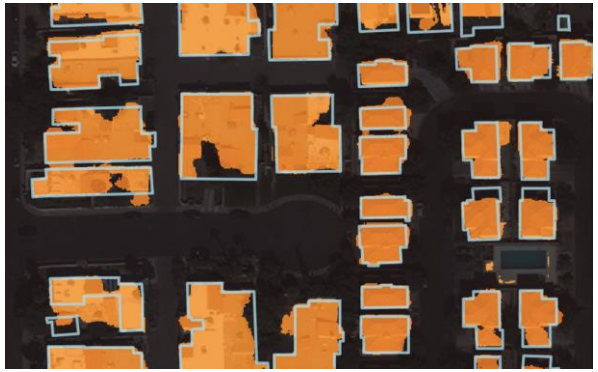
Supervised learning tasks require a ground truth, such that we can compute a loss/error from the prediction against the true value and propagate it back to the learning algorithm (how depends on the concrete ML algorithm).

1. Problem: Raw data is hugely available, but labelled data is scarce.
Example: Tons of images, but medical images companioned with the diagnosis and treatment is rare (privacy issues, only available for medical institutions, etc.)

2. Problem: Manual labelling is time-consuming and costly
Example: Label all interesting / relevant objects in a satellite image or camera of a car

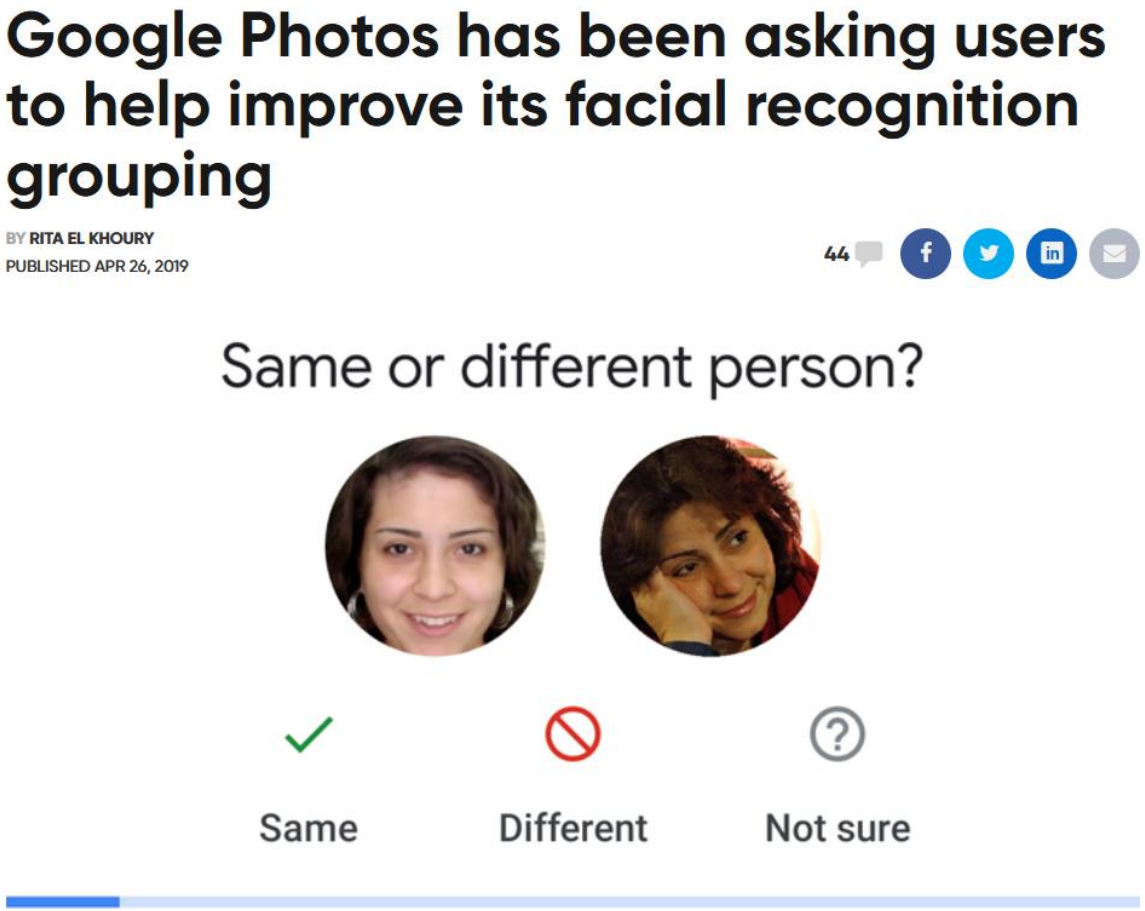


3. Problem: Labelling often requires expert knowledge or completed processes
Example: Medical diagnosis, click-through rate of recommended products



Data Labelling: Flywheel

Usually expensive (see earlier topics) and time consuming



Deploy model with high acurracy, but low recall (Idea: Always correct and does not annoy users)

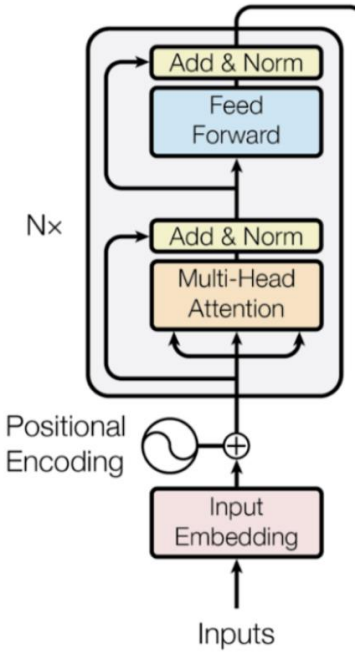
Ask users to obtain labelled data to improve recall (Users are motivated to improve their cataloge, the AI company gets free labels)

With the new labelled data, re-train, re-deploy the model and obtain more customers.

Pre-training + Fine-tuning

Idea: Pre-train a model using self-supervised learning (next slide) to learn internal features and semantic abstractions in the model without the need of any labelled data. Fine-tune the model on the actual learning task with the few labelled data that are available.

Example: BERT

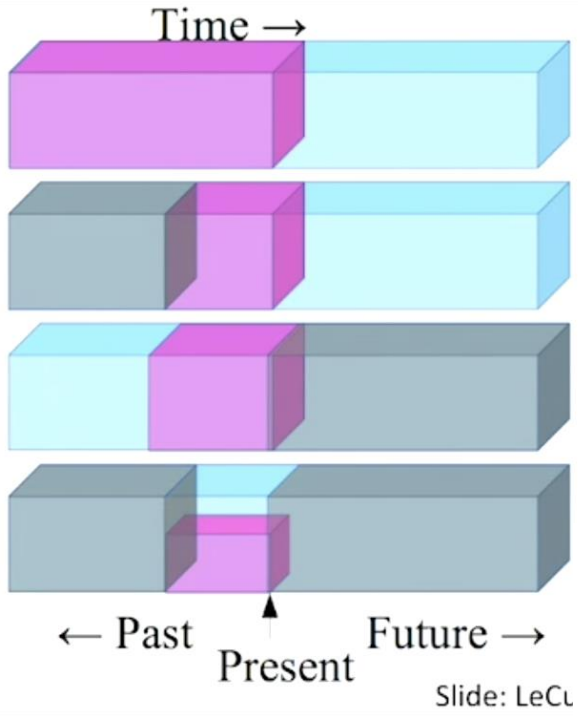


Random mask (remove) 15% of tokens in each sequence

Predict next sentence in a binary classification task.

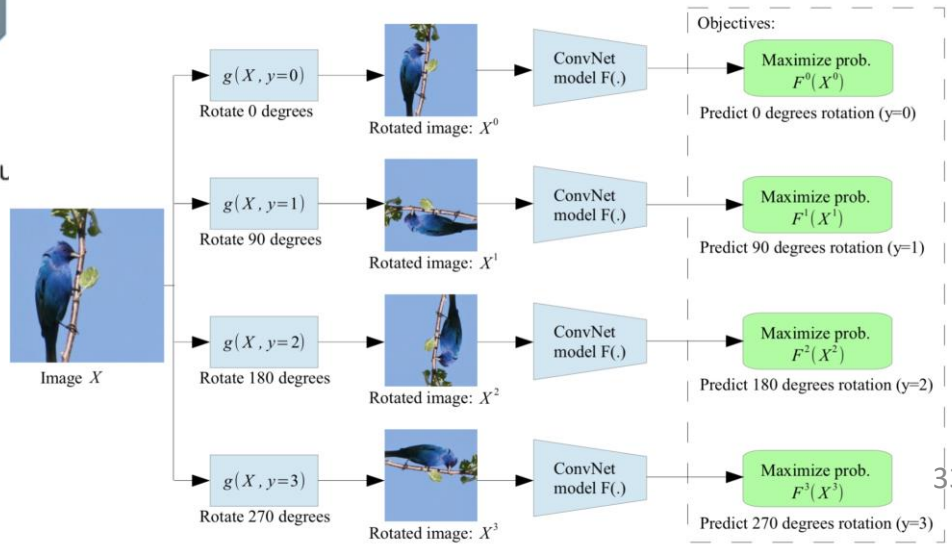
Self-Supervised Learning

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Idea: Frame a supervised learning task in a way that it predicts a subset of the input data.

Goal: Use the produced intermediate representation of the learned model to build on it for the actual task

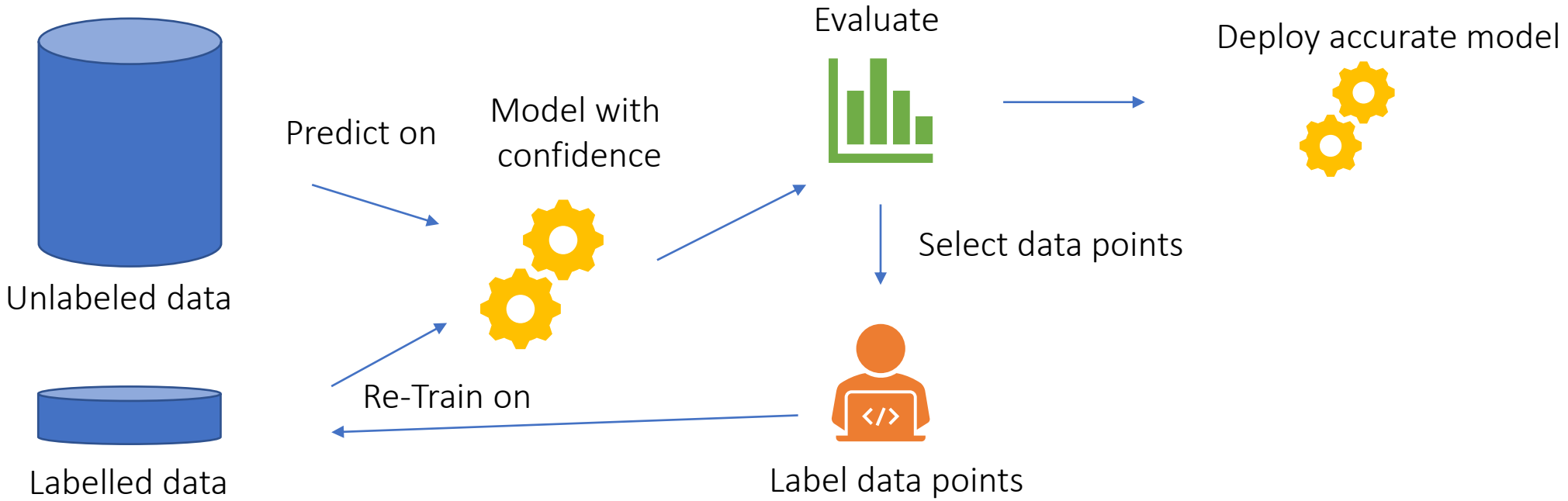


To identify the same image with different rotations, the model must learn to recognize high level object parts, such as heads, noses, and eyes, and the relative positions of these parts, rather than local patterns.*

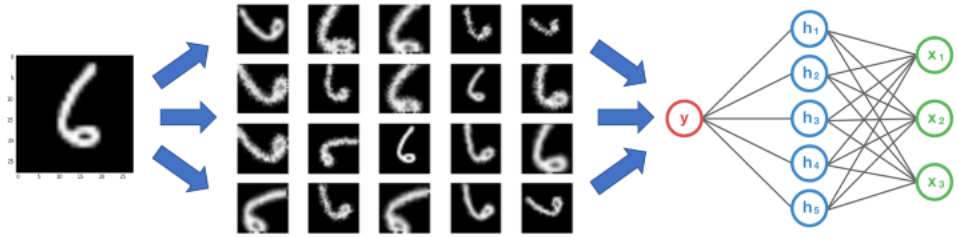
Active Learning

Idea: Learn which unlabeled data points are most valuable for being labelled next.

Realization: Use the small set of labelled data to train one or more models, preferable outputting confidence values with their predictions. The human selects those data points next for labelling where the confidence is low or 50:50. Alternatively, when multiple models are used, the human selects the data points for which the models disagree. Once the data points are selected, the human labels the data and retrains the model. Several iterations are performed until model accuracy is suitable.



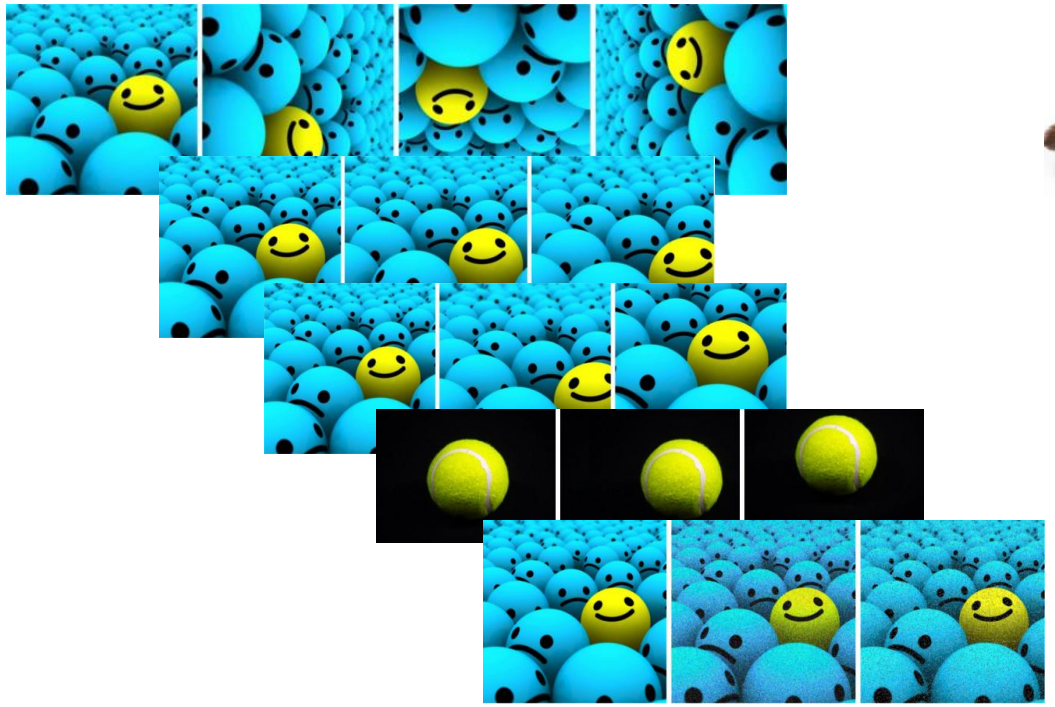
Data Augmentation

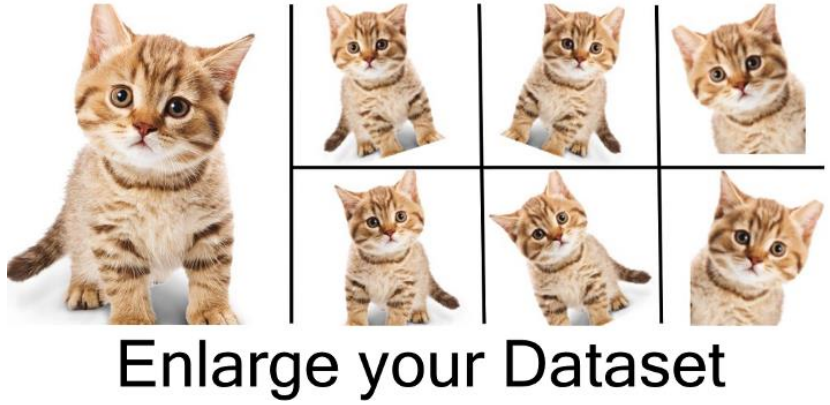


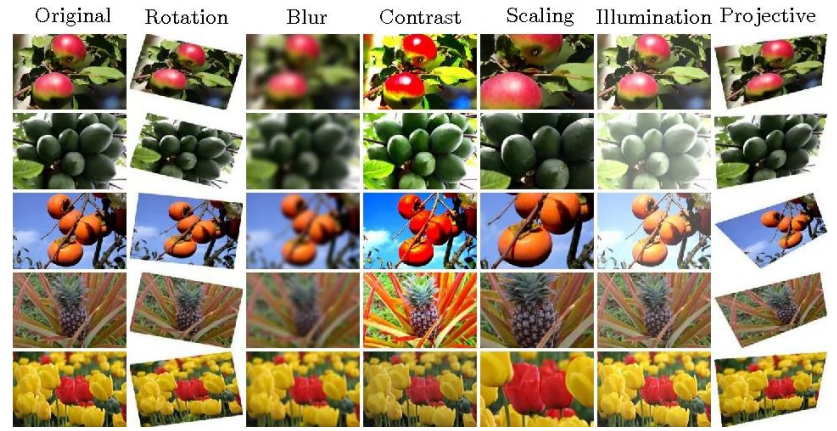
Idea: Generate more data by augmenting existing data points.

Techniques:

1. Flip&Rotation
2. Scale
3. Crop
4. Translation
5. Noise







Advanced: Condition GANs
synonym replacement,
word swap, back translation

Data Synthesis

Idea: Simulate real data (i.e., generate similar data with a notion of a similarity metric)

Benefits:
- Avoid real data usage restrictions (e.g., privacy constraints)
- Simulate unseen, but potential events
- High quality and ground truth
- Preserve multivariate relationships among input variables instead

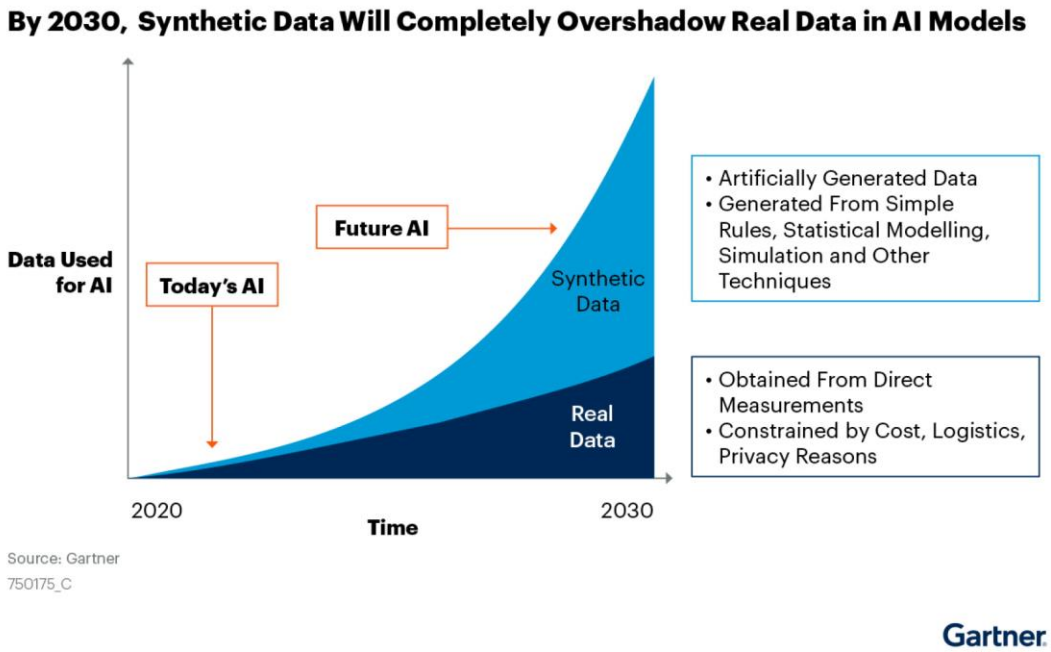
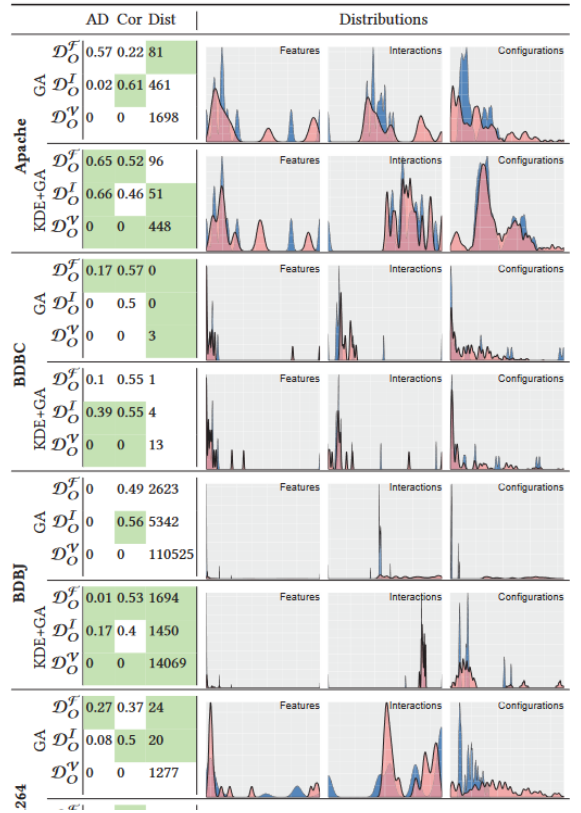
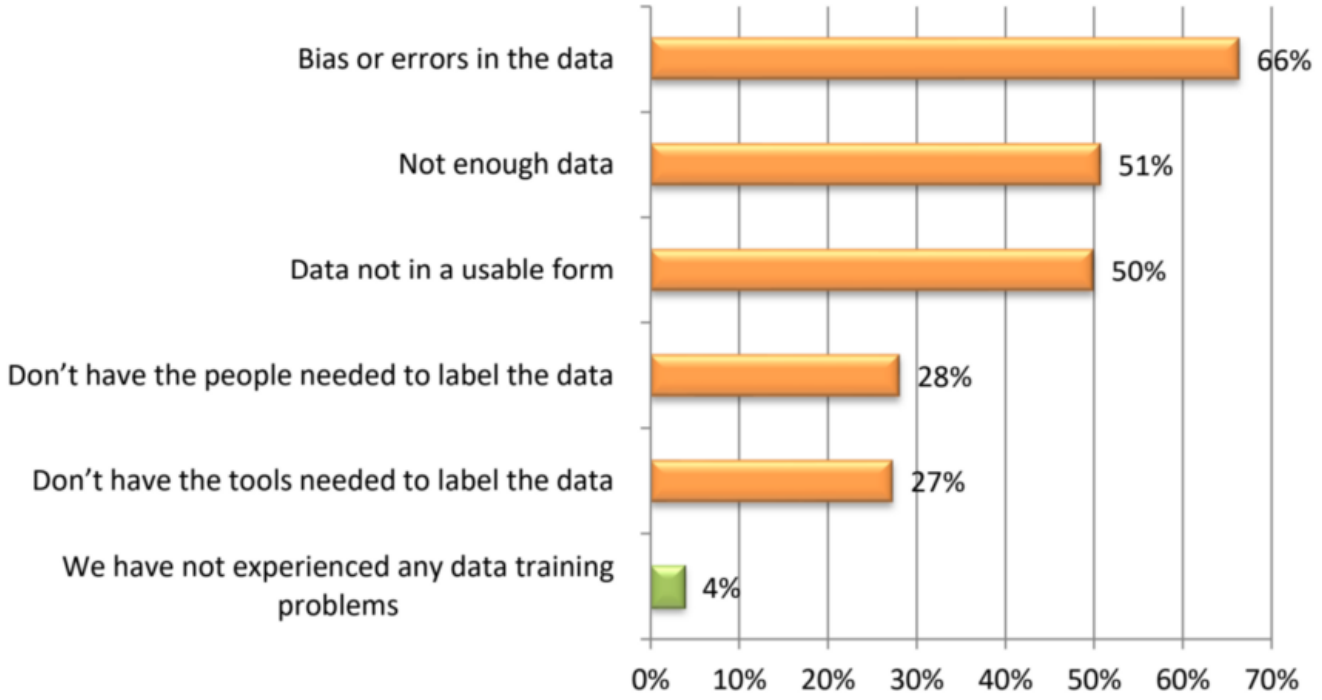


Table 3: Comparing output distributions with input distributions using kernel density estimation (KDE) and genetic algorithm (GA). AD=Anderson-Darling test; Cor=Pearson's correlation coefficient; Dist=Euclidean distance




TO COMPLETE YOUR REGISTRATION, PLEASE TELL US WHETHER OR NOT THIS IMAGE CONTAINS A STOP SIGN:
STOP
NO
YES
ANSWER QUICKLY—OUR SELF-DRIVING CAR IS ALMOST AT THE INTERSECTION.
SO MUCH OF "AI" IS JUST FIGURING OUT WAYS TO OFFLOAD WORK ONTO RANDOM STRANGERS.

Data Labelling and Quality is a Major Challenge

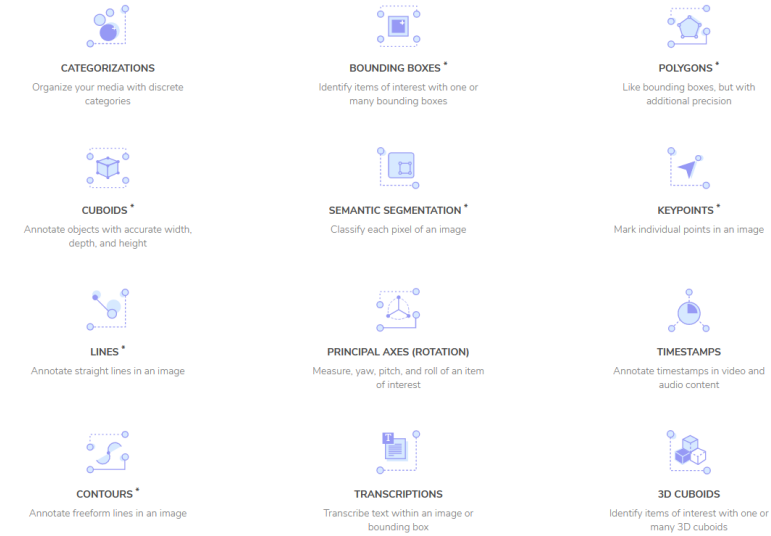


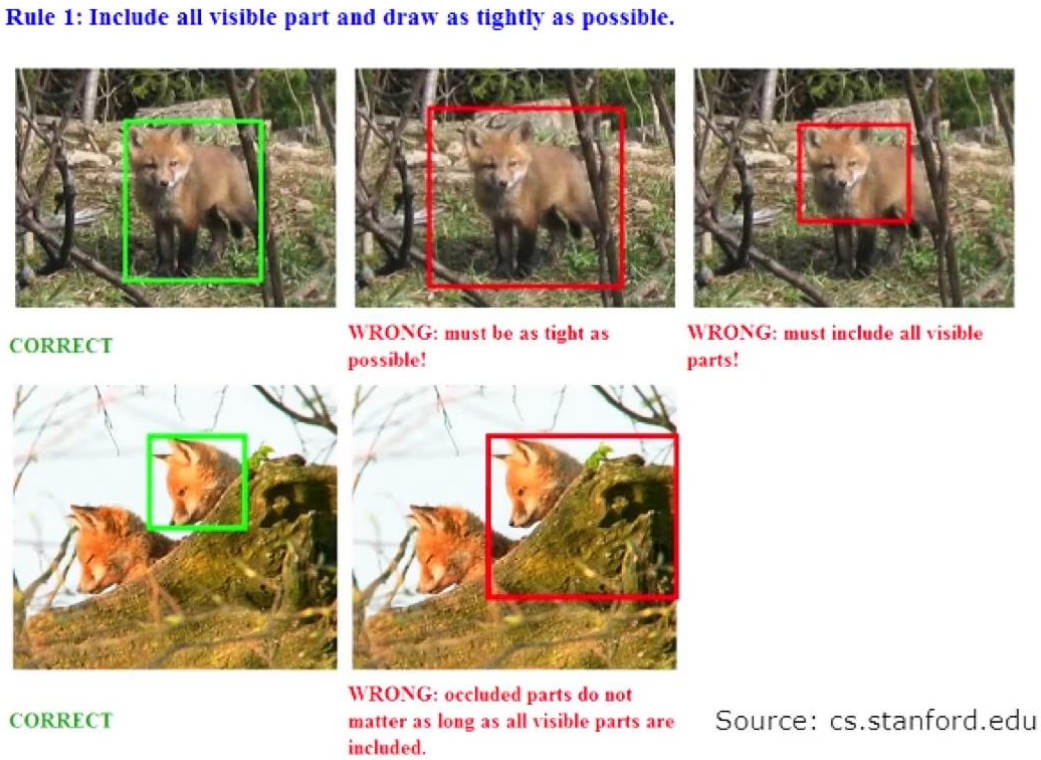
Study by Dimensional Research (2019) reported 96% of organizations run into problems of training-data quality and quantity

Labelling Process

Tool support available (e.g., Hive)

Be consistent in annotations!





Example taken from fullstackdeeplearning.com

Annotators

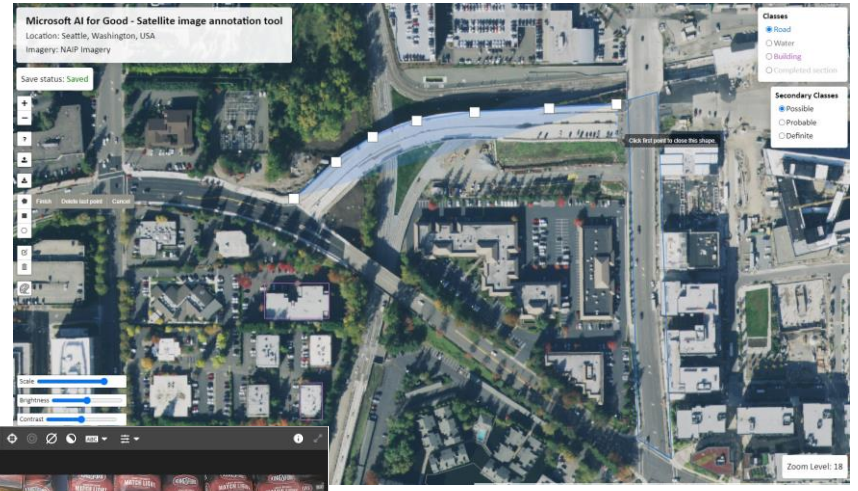
Direct employment: Direct control, high quality, but expensive and not scalable

Crowdsource: Easy to scale and not so expensive, but no quality control, high variation in results, requires additional quality assurance checks

Outsource to other companies: Expensive, but scalable and robust; select the appropriate company be letting them label a small data set and compare their results against your own (gold standard) label

Examples

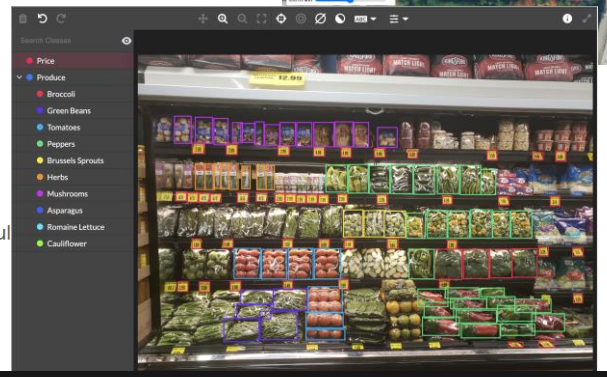
Microsoft Satellite Annotation Tool:



Appen

Have Confidence to Launch AI Projects with World-Class Training Data

With over 25 years of experience, we are the Data for AI Lifecycle leader. With 1,000s of successful projects in data sourcing, data annotation and model evaluation by humans, we enable the most innovative companies with AI projects to succeed.



Scale.AI

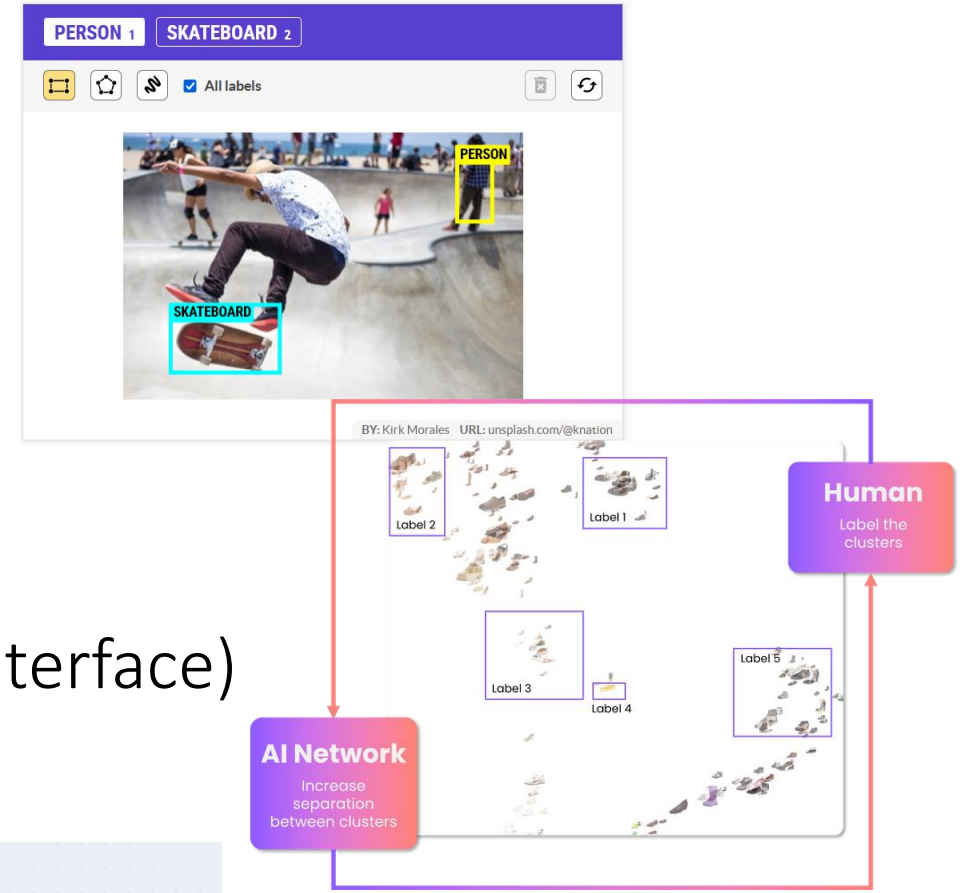
Please label any people, places, or organizations in the following text

They were delighted by the Hilton's (Organization) location: surrounded by vibrant nightlife, and in close proximity to major landmarks in Paris. (Location) And as luck would have it, Paris (Person) Hilton happened to be staying in the room next door!

```
client.createAnnotationTask({
  callback_url: 'http://www.example.com/callback',
  instruction: 'Draw a box around each rooftop and pool.',
  attachment: 'http://i.imgur.com/XOJbalC.jpg',
  objects_to_annotate: ['pool', 'rooftop'],
  with_labels: true,
  min_width: 30,
  min_height: 30
}, (err, task) => {
    // do something with task
});
```

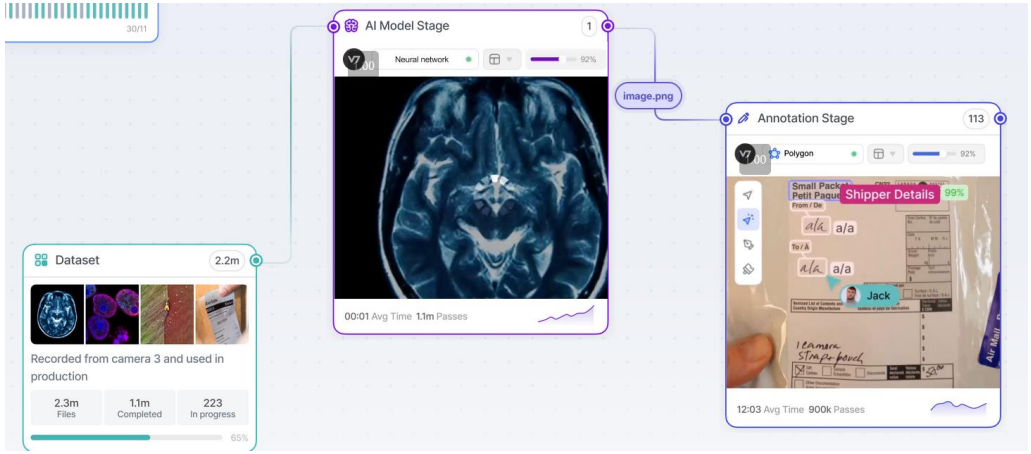
Annotation Tools

Prodigy (includes active learning)



Platform.AI (innovative labelling interface)

V7

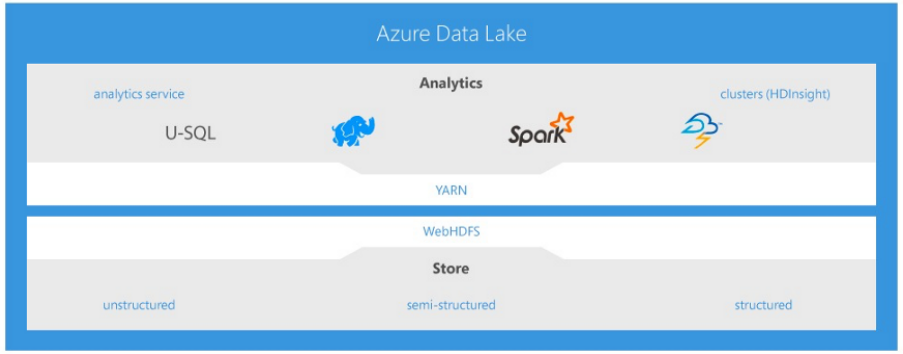


https://github.com/taivop/awesome-data-annotation

Data Storage

Different types of storage:

- File system (most basic form): local, distributed; not versioned (e.g., HDFS)
- Object storage: Database storing objects (or files) and providing an easy interface hiding the actual storage location; versioning support (e.g., S3, Ceph)
- Database systems: persistent and fast data storage to retrieve mostly structural data (with specialized systems, e.g., graph databases); works not for binary data; data will be retrieved often (e.g., Postgres)
- Data lake: unstructured place to store data without having a schema (could be a filesystem or object store); idea is to dump data now and transform it later; Hadoop file system often used



Where to store what

Binary data -> object store (version support; easy API access)

Metadata + structured data -> database

Raw data from diverse sources (scraping, logs, etc.) -> data lake

Trained models and artifacts reside as files -> use DVC (see next)

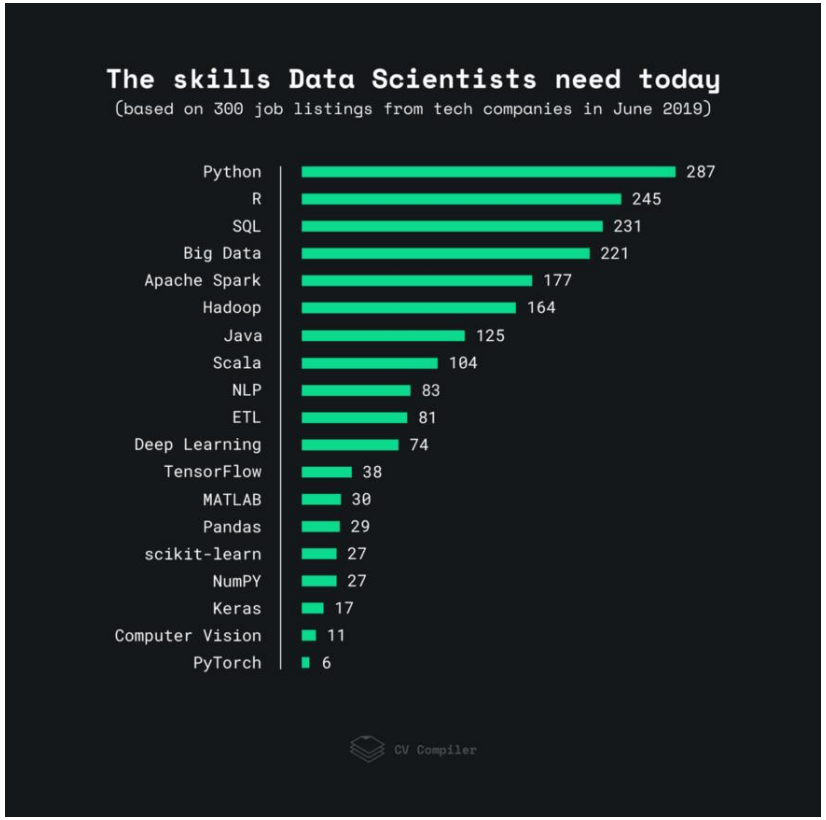
Data Access: SQL

SQL is a fundamental skill for data scientists (and often software engineers)

Declarative language for selecting, filtering, aggregating (and partially computing) data that is stored in a database

Used in Google's BigQuery data warehouse

> *BigQuery is an enterprise data warehouse that solves problems by enabling super-fast SQL queries using the processing power of Google's infrastructure. Simply move your data into BigQuery and let us handle the hard work.*
>
> *— BigQuery Documentation [2]*



Data Versioning

Problem: Model = BinaryOf(Data,Code); we version the code, but what is with the data? Without versioning, we cannot reproduce (an older) model anymore

Idea: Version also the data

Approaches:
Take snapshots of data (e.g., in Zip files) for every trained model or experimentation run

- Does not scale; organizational chaos; requires extra functionality (extraction, automatic referencing to code); is not the same as a Git hash (we have this for a reason); loses information about evolution history

Code repos with extensions (e.g., Git LFS / Git annex)

- LFS: Metadata is stored in Git, but file content is stored in S3 bucket (object storage);
  Annex: contents of files are stored in a distributed key/value store

Specialized solutions (see next)

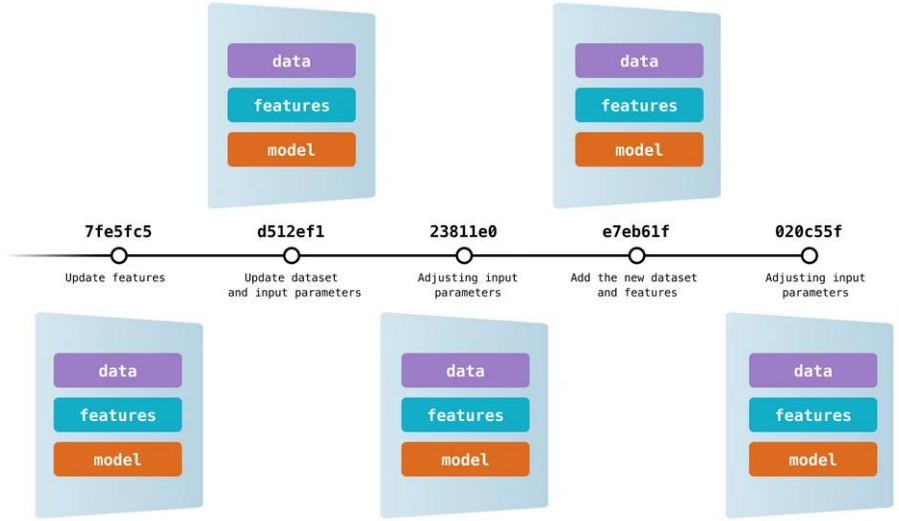
DVC

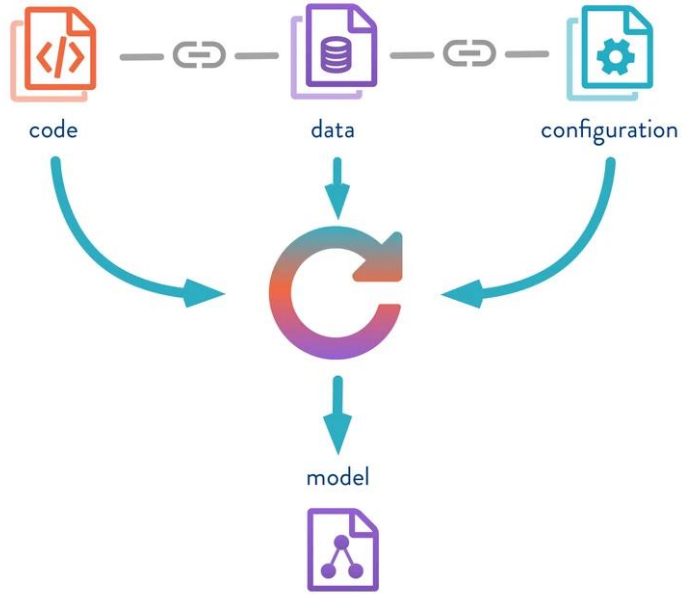
Open-source command-line tool for data versioning on top of Git repos

Data files (models, training data, etc.) replaced by meta-data files, pointing to the actual files

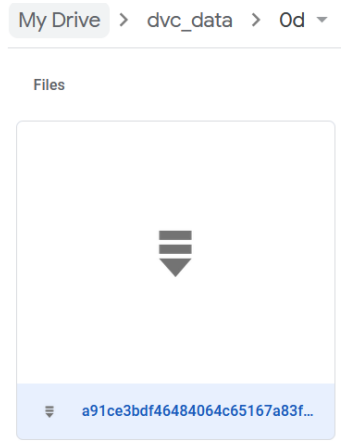
Storage is either on cloud or on premise

Includes a pipeline definition for data handling, which gets also versioned





DVC – Data Versioning



pip install dvc[gdrive] #or alternative ways, depending on the cloud storage type

dvc init #initializes repo within a git repo
git status #show added files to git repo
git commit -m "Initialize DVC“ #commit dvc files to git repo

#define where to store your data
dvc remote add --default myGDrive gdrive://XXXXX
#here, we use Google‘s GDrive
dvc remote modify myGDrive gdrive_client_id YYYYYY
dvc remote modify myGDrive gdrive_client_secret XXXXXXXX

#add remote config setting to the git repo
git add .dvc/config
git commit -m "Configure remote storage“

dvc add data/data.xml #add data to dvc versioning
git add data/data.xml.dvc data/.gitignore #add meta-data files
git commit -m "Add raw data“
dvc push #push dvc files to remote server

Supported storage types

The following are the types of remote storage (protocols) supported:

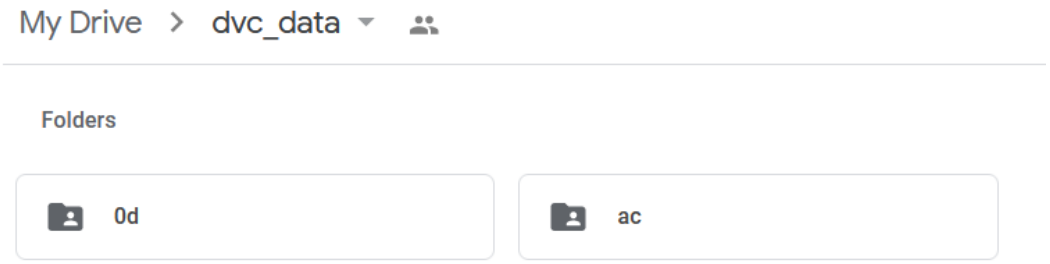
- Amazon S3
- S3-compatible storage
- Microsoft Azure Blob Storage
- Google Drive
- Google Cloud Storage
- Aliyun OSS
- SSH
- HDFS
- WebHDFS
- HTTP
- WebDAV
- local remote

DVC – Data Versioning II

Overwrite the data file with the a new version

dvc add data/data.xml
git add data/data.xml.dvc
git commit –m “updated training data set”

dvc push



#To get prior versions, check out the corresponding versions of the dvc file and apply dvc checkout to download the data file version that is linked to the dvc file version

git checkout HEAD^1 data/data.xml.dvc #reverts to the previous versions of the dvc file
dvc checkout #reverts to the previous versions of the data file

DVC – Project: Building a Pipeline

dvc-init starts a project and creates the cache for storing large files

dvc.yaml defines pipelines for ML experiments

.dvc files are placeholders for large files and directories that get tracked and stored in the cache

Pipelines are resembled from stages, which represent certain data transformation processes

```
stages:
  transpose:
    cmd: ./trans.r rows.txt > columns.txt
    deps:
      - rows.txt
    outs:
      - columns.txt
```

Cmd will be executed when a stage gets reproduced (dvc repro)

Enable to check whether data has been changed, and the command must be executed again (dvc status)

Enable to track new artifacts when produced in this stage (get automatically versioned; also dvc add)

DVC – Tracking Parameters

```yaml
stages:
  preprocess:
    cmd: bin/cleanup raw.txt clean.txt
    deps:
      - raw.txt
    params:
      - threshold # track specific param (from params.yaml)
      - passes
      - myparams.yaml: # track specific params from custom file
          - epochs
      - config.json: # track all parameters in this file
    outs:
      - clean.txt
```

Checks whether parameters in project or specific files have changed.

If so, it will reproduce (i.e., re-execute) this stage and all following stages.

DVC – Tracking Experiment Data

```yaml
stages:
  build:
    cmd: python train.py
    deps:
      - features.csv
    outs:
      - model.pt
    metrics:
      - accuracy.txt:
          cache: false
    plots:
      - auc.json:
          cache: false
```

Small file, so we let Git version it

Version metrics and plots; define meta-parameters and runs

```yaml
stages:
  build:
    foreach:
      uk:
        epochs: 3
        thresh: 10
      us:
        epochs: 10
        thresh: 15
    do:
      cmd: python train.py '${key}' ${item.epochs} ${item.thresh}
      outs:
        - model-${key}.hdfs
```

```
$ dvc metrics diff
Path          Metric    HEAD      workspace  Change
metrics.json  AUC       0.763981  0.801807   0.037826
```

```
$ dvc plots diff HEAD^ --targets logs.csv
file:///Users/usr/src/dvc_plots/index.html
```

Neptun

neptune.ai

Embedded into an experimentation tracking framework

```
pip install neptune-client
```

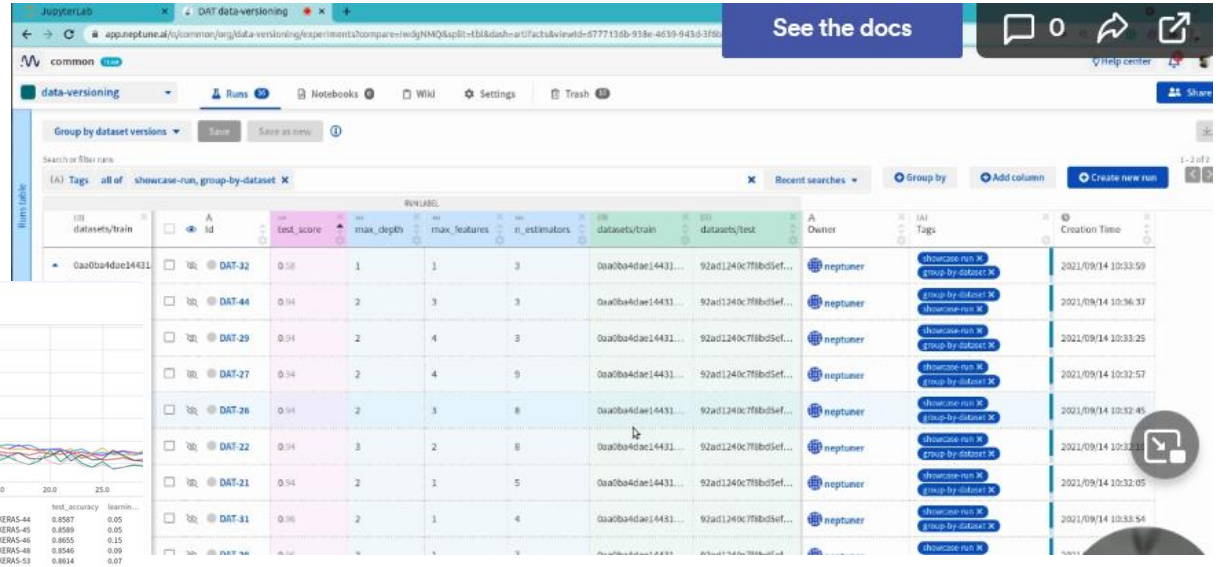
3. Add logging to your script

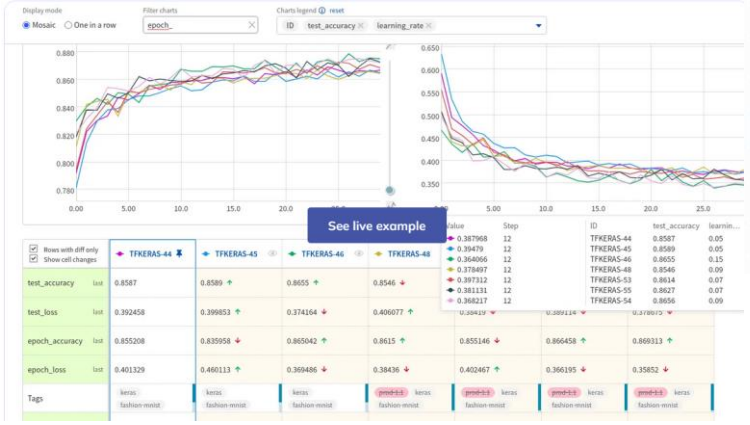
```
import neptune.new as neptune

run = neptune.init('Me/MyProject')
run['params'] = {'lr':0.1, 'dropout':0.4}
run['test_accuracy'] = 0.84
```

```
1 run["datasets/train"].track_files(TRAIN_DATASET_PATH)
2 run["datasets/test"].track_files(TEST_DATASET_PATH)
```

Experiment tracking and model registry for production teams

Log, store, query, display, organize, and compare all your model metadata in a single place

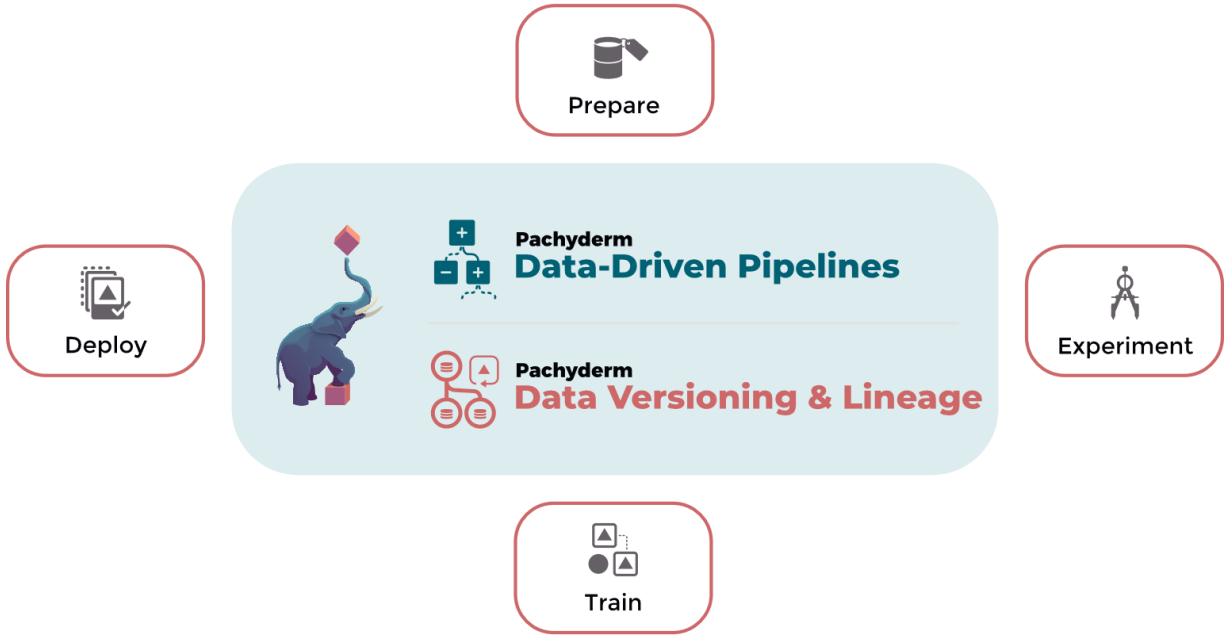




Pachyderm

Enterprise system that focuses on scaling DataOps. Written in Go, sitting on top of a Kubernetes cluster, it enables pipelines for data preprocessing, experimentation, training, deployment, and data versioning.

Community edition exists with limited features; integrates with other tools and platforms in the MLOps stack.



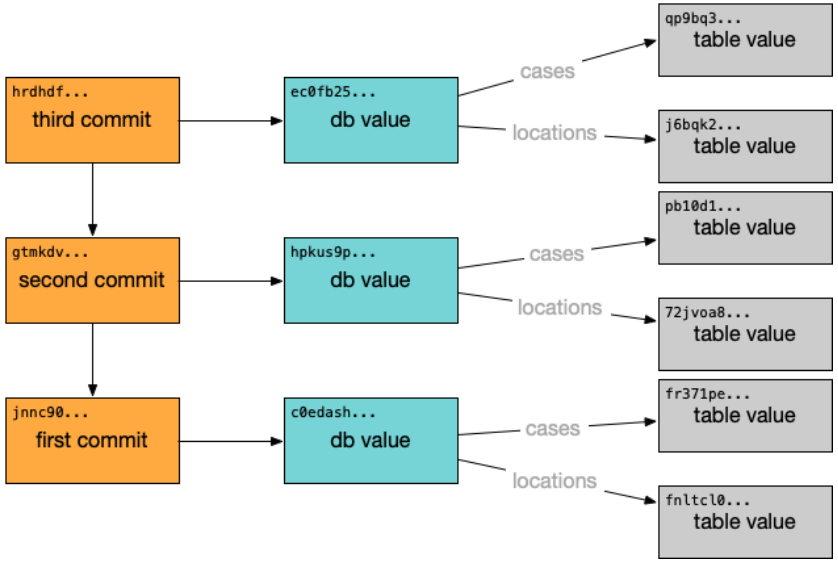
Dolt

"Dolt is the first and only SQL database that you can fork, clone, branch, merge, push and pull just like a Git repository."

The idea is to apply the git functionality onto relational data in tables. Typical git commands are implemented using SQL queries or functions.

Enables diffing of data to review changes made by others (customers, other teams, processes, etc.).

Enables branches of a table to include, for instance, exploration of preprocessing alternatives.



Dolt – Usage

#Download dolt and configure it
dolt config --global --add user.name <your name>
dolt config --global --add user.email <your email>

dolt init #init repository (similar to git init)

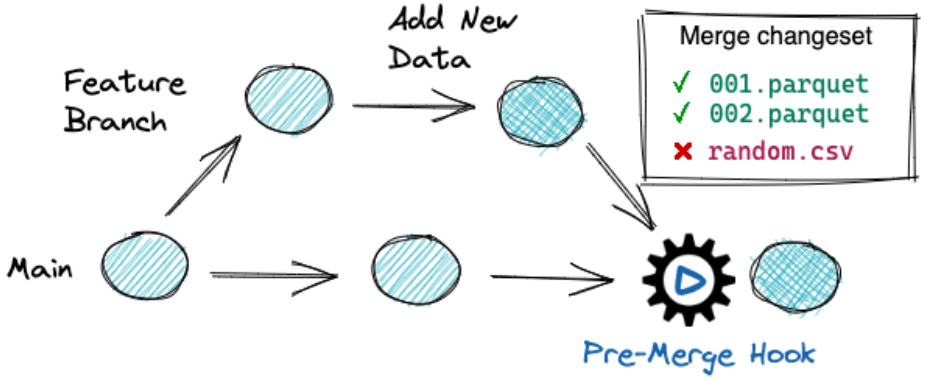
dolt table import -c adults adult.csv #importing a csv generates a table with the data stored in the file
dolt schema show adults #show the db schema

dolt status #similar to git status
dolt add .
dolt commit –m “initial data”

dolt checkout -b feature #create a feature branch
dolt sql #open sql shell make some changes
dolt diff #show differences
dolt add .
dolt commit –m “Changes XY made”
dolt checkout main #go back to main branch
dolt merge feature #and merge changes to main branch

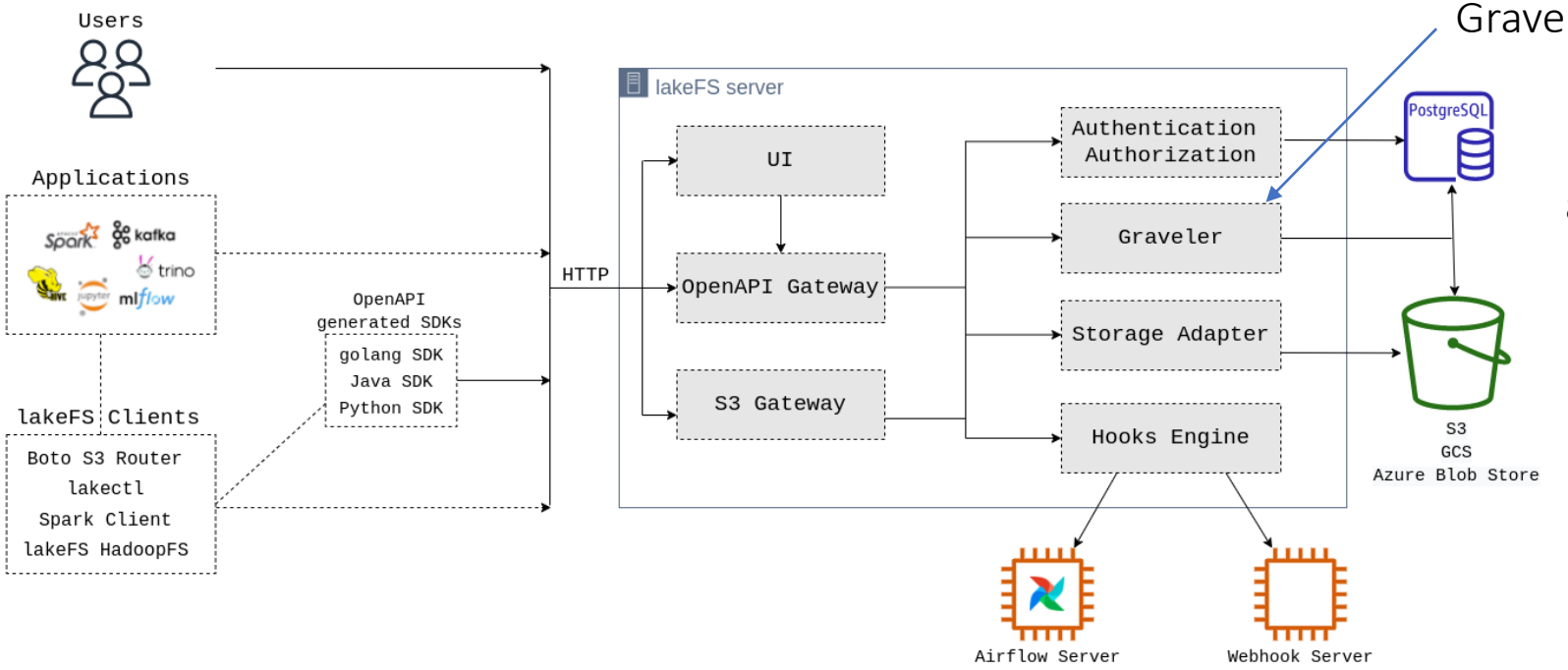
```
D:\Lehre\Leipzig\SS22\SE4AI\data>dolt table import -c adults adult.csv
Rows Processed: 32,561, Additions: 32,561, Modifications: 0, Had No Effect: 0
Import completed successfully.

D:\Lehre\Leipzig\SS22\SE4AI\data>dolt schema show adults
adults @ working
CREATE TABLE `adults` (
  `age` int unsigned NOT NULL,
  `workclass` varchar(16383) NOT NULL,
  `fnlwgt` int unsigned NOT NULL,
  `education` varchar(16383) NOT NULL,
  `education-num` int unsigned NOT NULL,
  `marital-status` varchar(16383) NOT NULL,
  `occupation` varchar(16383) NOT NULL,
  `relationship` varchar(16383) NOT NULL,
  `race` varchar(16383) NOT NULL,
  `sex` varchar(16383) NOT NULL,
  `capital-gain` int unsigned NOT NULL,
  `capital-loss` int unsigned NOT NULL,
  `hours-per-week` int unsigned NOT NULL,
  `native-country` varchar(16383) NOT NULL,
  `label` varchar(16383) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_bin;
```

lakeFS



Transform your data lake into a Git-like repository

The lakeFS open source project for data lakes allows data versioning, rollback, debugging, testing in isolation, and more – all in one.



Graveler handles versioning of data.

Meta data about versions, etc. is stored in a PostgreSQL database.

Data is stored in an object store, e.g., S3.

Final Thoughts: Checklist

What data do we need to reach our goal?

How to obtain it?

- Data sources, artefacts, monitorings, scrapings?

How to clean it?

- Missing data? Erroneous data? Outsiders? Balance? Trustworthy? Legal?

How to store it?

- Where / what is our data lake? What is the amount of data stored + access? What data needs versioned, what not? What is structured and binary and how often do I need to access it?

How to label?

- Hot to get the ground truth? What is the effort for that? How can we verify the correctness of this step?

Topic 3: Experimentation

Why Experimentation?

This is an empty slide, you fill it!

What to track?

Parameters: key-value inputs to the ML code (experimental settings, hyperparameters, model parameters)

Metrics: numeric values

Tags and notes: description about an individual experiment run (compare with git tag)

Artifacts: files, data, and models used or produced in a run

Source: code that was executed

Version: version of the code that was executed

Why using a framework?

```
import pandas as pd

df = pd.read_csv(file)
df_processed = tokenize(sanitize(df))

model = train_model (df_processed, learning_rate=lr, penalty=p, optimizer=opt)
score = compute_accuracy(model)

print("For penalty=%s, lr=%f, optimizer=%s: accuracy=%f" %(p, lr, opt, score))
pickle.dump(model, open("model.pl"))
```

```
For penalty=0.5, lr=0.1, optimizer=adam: accuracy= 0.76
For penalty=0.6, lr=0.1, optimizer=adam: accuracy= 0.78
For penalty=0.7, lr=0.1, optimizer=adam: accuracy= 0.73
For penalty=0.6, lr=0.2, optimizer=adam: accuracy= 0.81
…
```

Where to store? Which format?
What if ML library changes?
What if my code changes?
What if I update / tune my params?
What if I change my input data?

Tools and Frameworks for Experiment Tracking

DVC (covered)

MLFlow

Weights and Biases (wand)

Neptune.AI (covered)

Sacred

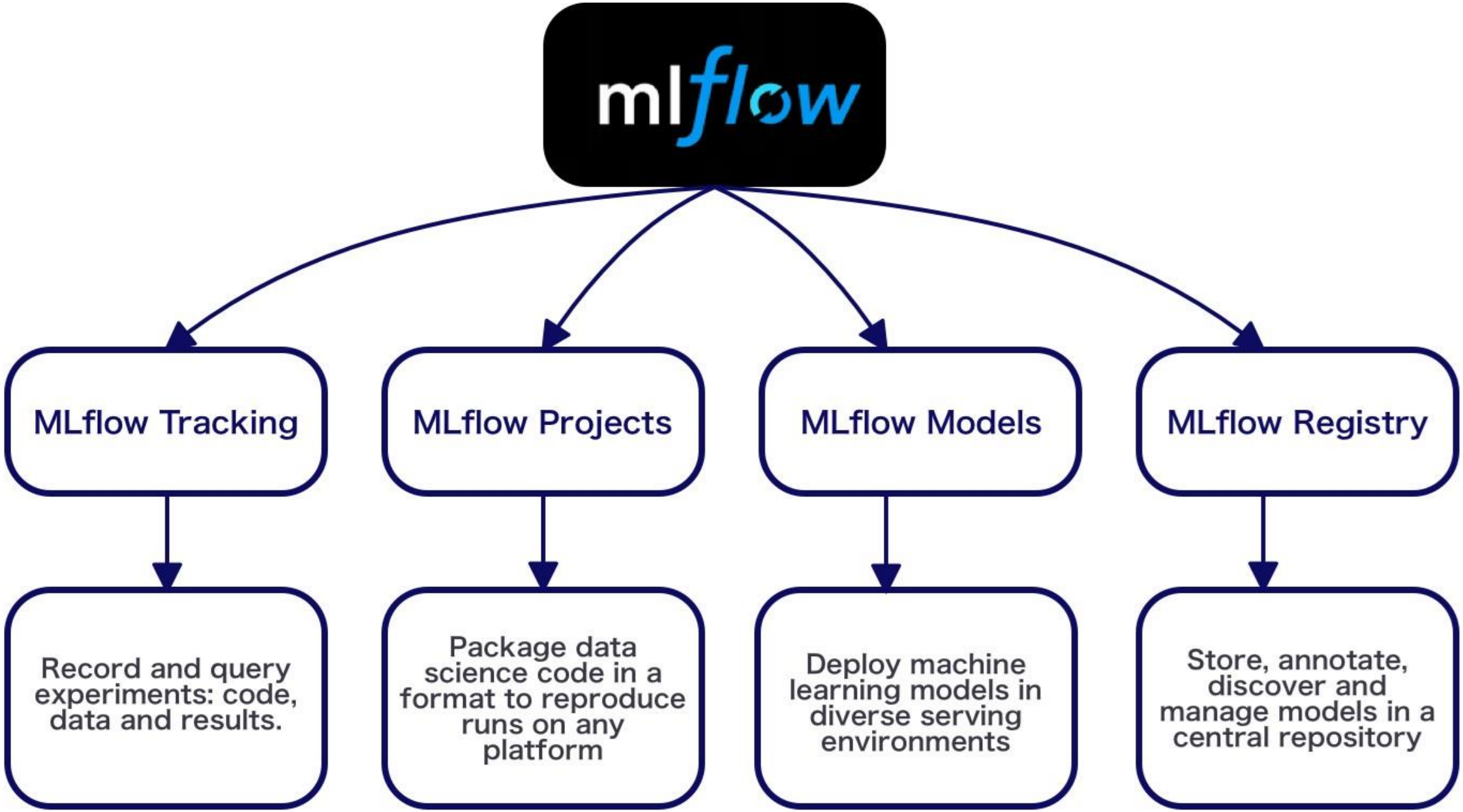
and many more popping up every month…

MLflow

Open source machine learning platform

- Goals: Experiment tracking, model logging & versioning, model deployment
- Extensible for new algorithms/frameworks: APIs, REST APIs + CLI
- Integrates many popular ML libraries and frameworks
- Supports different languages
- Same execution model everywhere (cloud or local)
- Highly scalable (single person projects to large companies)
- Modular and easy to integrate

pip install mlflow



Governance, experimentation analysis, progress history, etc.

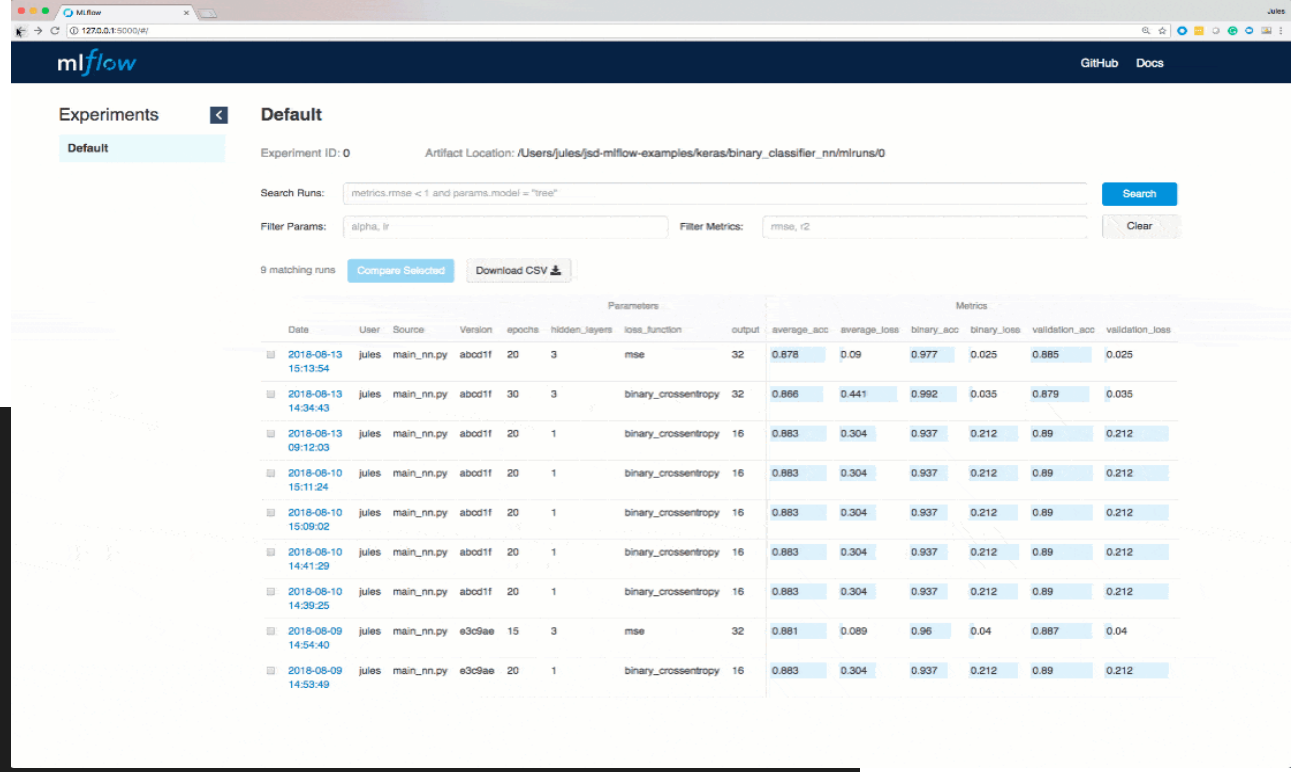
Reproducibility, platform independence

Automation capabilities, target platform independence

Scaling for multi-model, multi-project orgs

Credit: Towards Data Science

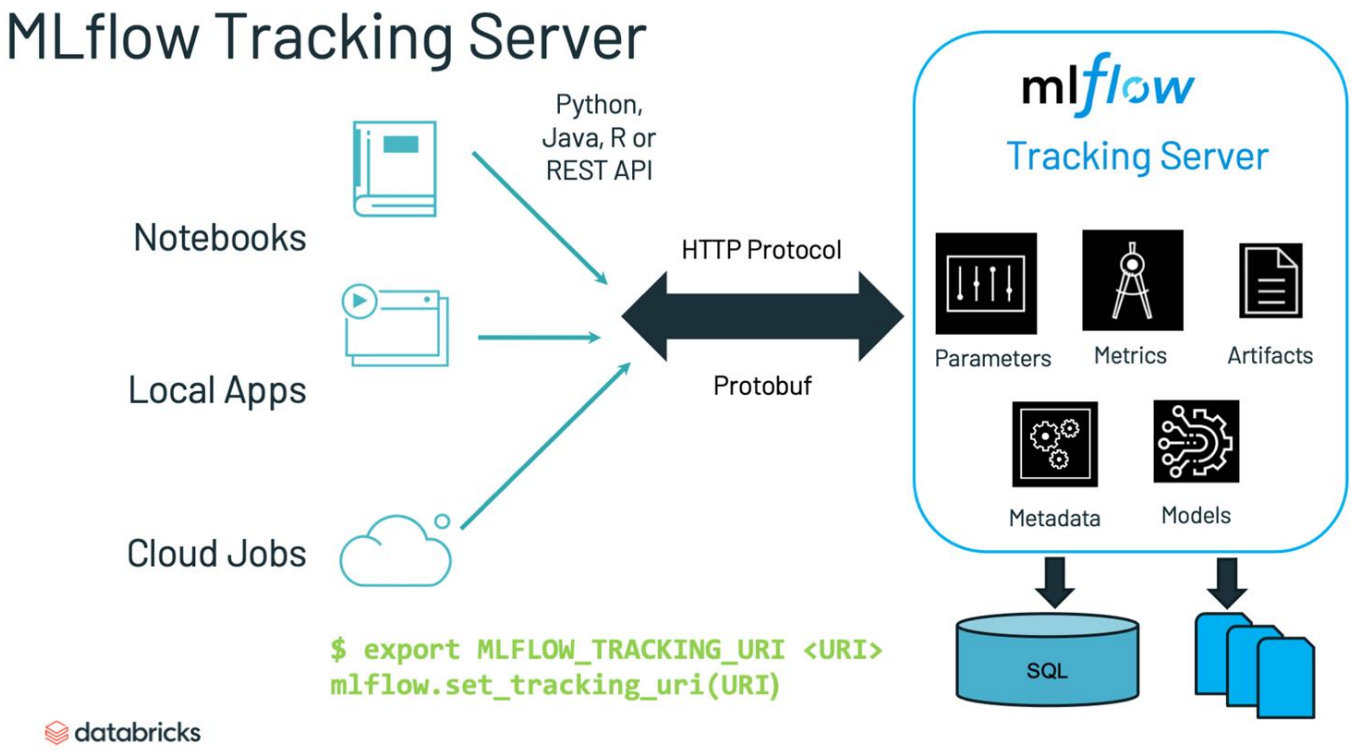
Tracking with MLFlow




```
import pandas as pd
import mlflow

df = pd.read_csv(file)
df_processed = tokenize(sanitize(df))

model = train_model (df_processed, learning_rate=lr, penalty=p, optimizer=opt)
score = compute_accuracy(model)

with mflow.start_run():
    mlflow.log_param("input_file", file)
    mlflow.log_param("learning_rate", lr)
    mlflow.log_param("penalty", p)
    mlflow.log_param("optimizer", opt)
    mlflow.log_metric("score", score)
    mlflow.sk_learn.log_model(model)
```

Creates an experimentation run
Logs all defined parameters and metrics, no matter how often this is executed.
With mlflow UI, we can inspect all runs visually to analyze and compare models.

FLflow Tracking Server



Entity store for metadata with options:

- FileStore (local) in mlruns directory
- SQLStore (SQLAlchemy)
- MLflow Plugins Scheme (customized)
- Managed MLflow on Databricks

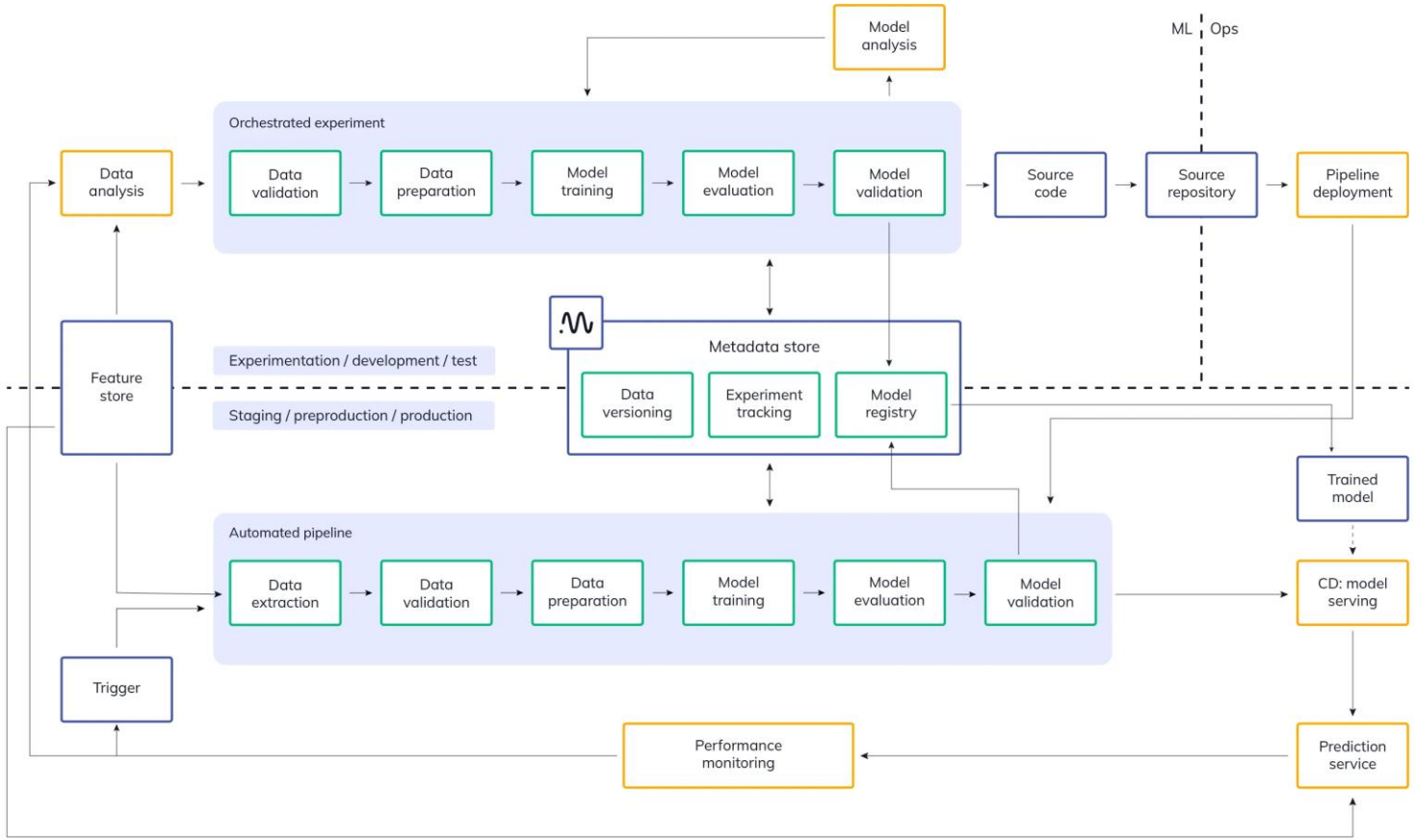
Artifact store for file artifacts:

- Local filesystem (mlruns directory)
- S3 bucket, Azure, Google cloud store
- DBFS artifact repo

Automatic logging supported for major ML libraries using mlflow.autolog() e.g., mlflow.tensorflow.autolog()

neptune.ai

Meta-data store for AI experiments




```
pip install neptune-client
```

For more help, see Installation and setup.

Step 2: Connect Neptune to your code

```
import neptune.new as neptune

run = neptune.init(project="corp_space/fraud_detection")
```

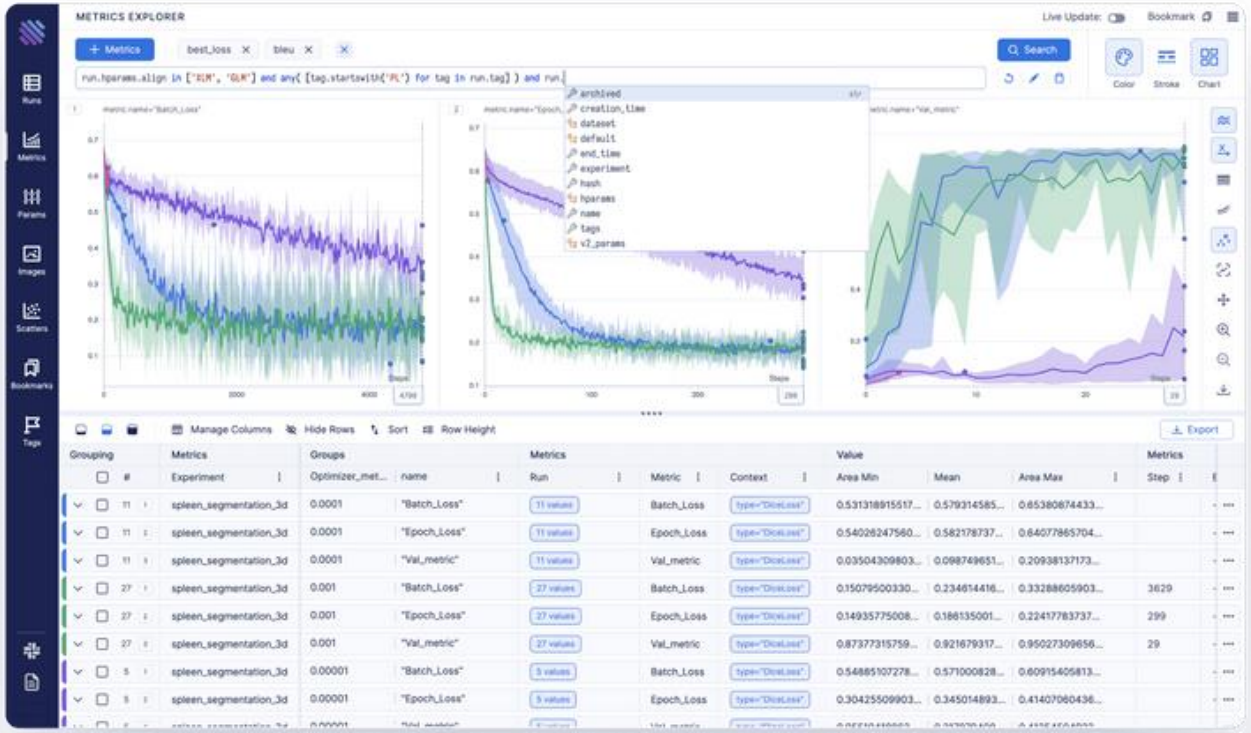
Step 3: Log metadata

```
run["parameters"] = {"lr": 0.001, "optim": "Adam"} # parameters
run["f1_score"] = 0.66 # metrics
run["roc_curve"].upload("roc_curve.png") # charts
run["model"].upload("model.h5") # models
```

Aim: Experiment Tracker

AimStack

Stores, compares AI experiments
Open-source framework

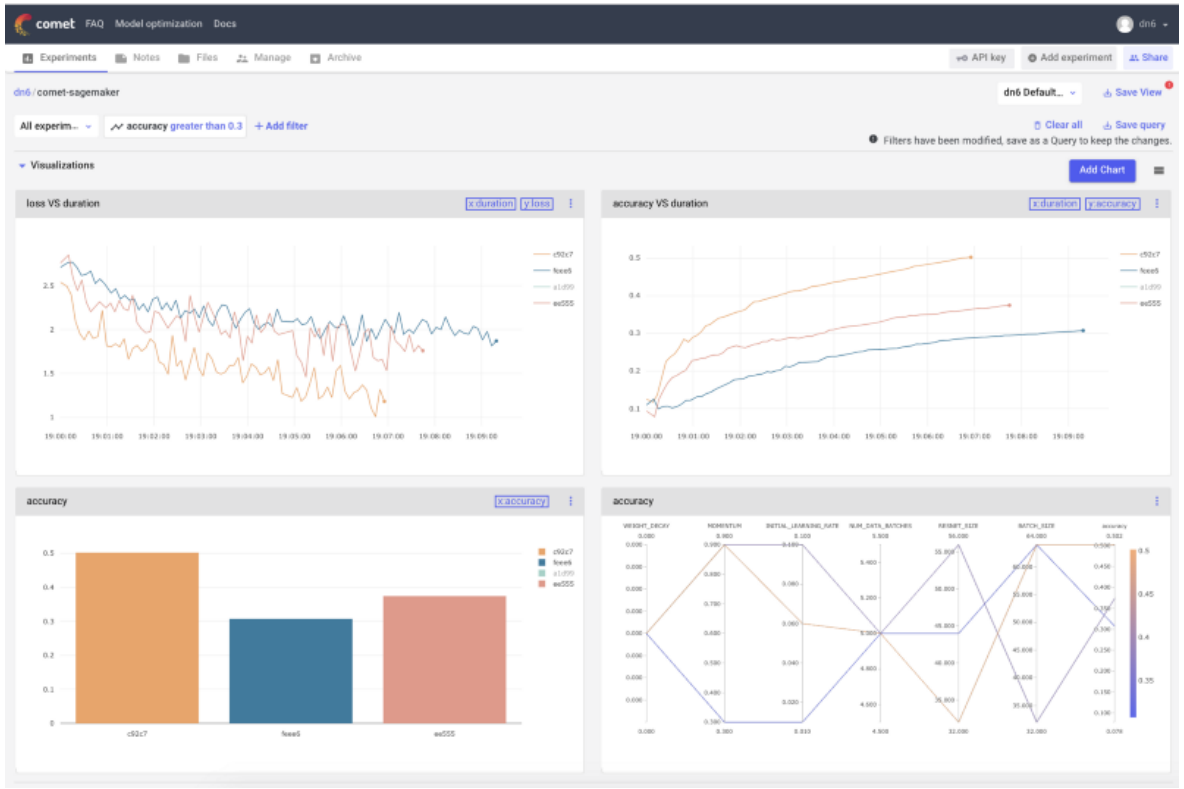



```python
from aim import Run
# Initialize a new run
run = Run()
# Log run parameters
run["hparams"] = {
    "learning_rate": 0.001,
    "batch_size": 32,
}
# Log metrics
for i in range(10):
    run.track(i, name='loss', step=i, context={ "subset":"train" })
    run.track(i, name='acc', step=i, context={ "subset":"train" })
```

```
aim up
```

Comet

Meta machine learning platform to track, compare, and explain AI experiments
Built-in support for other frameworks, such as Mlflow, kubeflow, etc.
Requires online account



Guild.AI

guild.yml

```yaml
- model: egan
  description: Anomaly detector using Efficient GAN
  reference: https://arxiv.org/abs/1802.06222
  operations:
    prepare-mnist:
      description: Prepare MNIST data set for training
      main: prepare_mnist
    prepare-kdd99:
      description: Prepare KDD99 data set for training
      main: prepare_kdd99
    train:
      description: Train model
      main: train
      flags:
        epochs:
          description: Training epochs
          default: 1000
        w:
          description: Weight for the sum of the mapping loss function
          default: 0.1
        m:
          description: Mode/method for discriminator loss
          default: fm
          choices: [cross-e, fm]
        d:
          description: Degree for the L norm
          default: 1
```

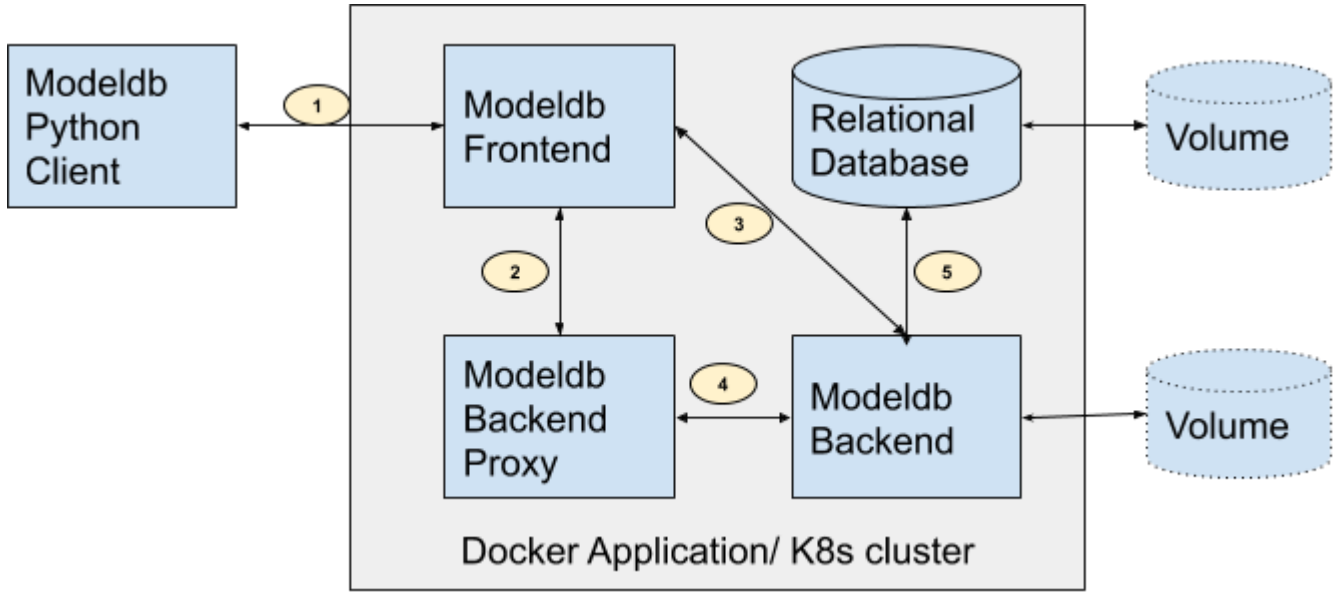
~/SCM/guild.ai/examples/guild.yml* 1:1 LF UTF-8 YAML master Fetch GitHub Git (9)

```python
import numpy as np

# Hyperparameters
x = 0.1
noise = 0.1

# Simulated training loss
loss = (np.sin(5 * x) * (1 - np.tanh(x ** 2)) + np.random.randn() * noise)

print("loss: %f" % loss)
```

Determines automatically that these could be hyperparameters to track

ModelDB

Open-source system for ML model versioning, metadata, and experiment management.




```python
from verta import Client
client = Client("http://localhost:3000")

proj = client.set_project("My first ModelDB project")
expt = client.set_experiment("Default Experiment")

# log the first run
run = client.set_experiment_run("First Run")
run.log_hyperparameters({"regularization" : 0.5})
# ... model training code goes here
run.log_metric('accuracy', 0.72)

# log the second run
run = client.set_experiment_run("Second Run")
run.log_hyperparameters({"regularization" : 0.8})
# ... model training code goes here
run.log_metric('accuracy', 0.83)
```

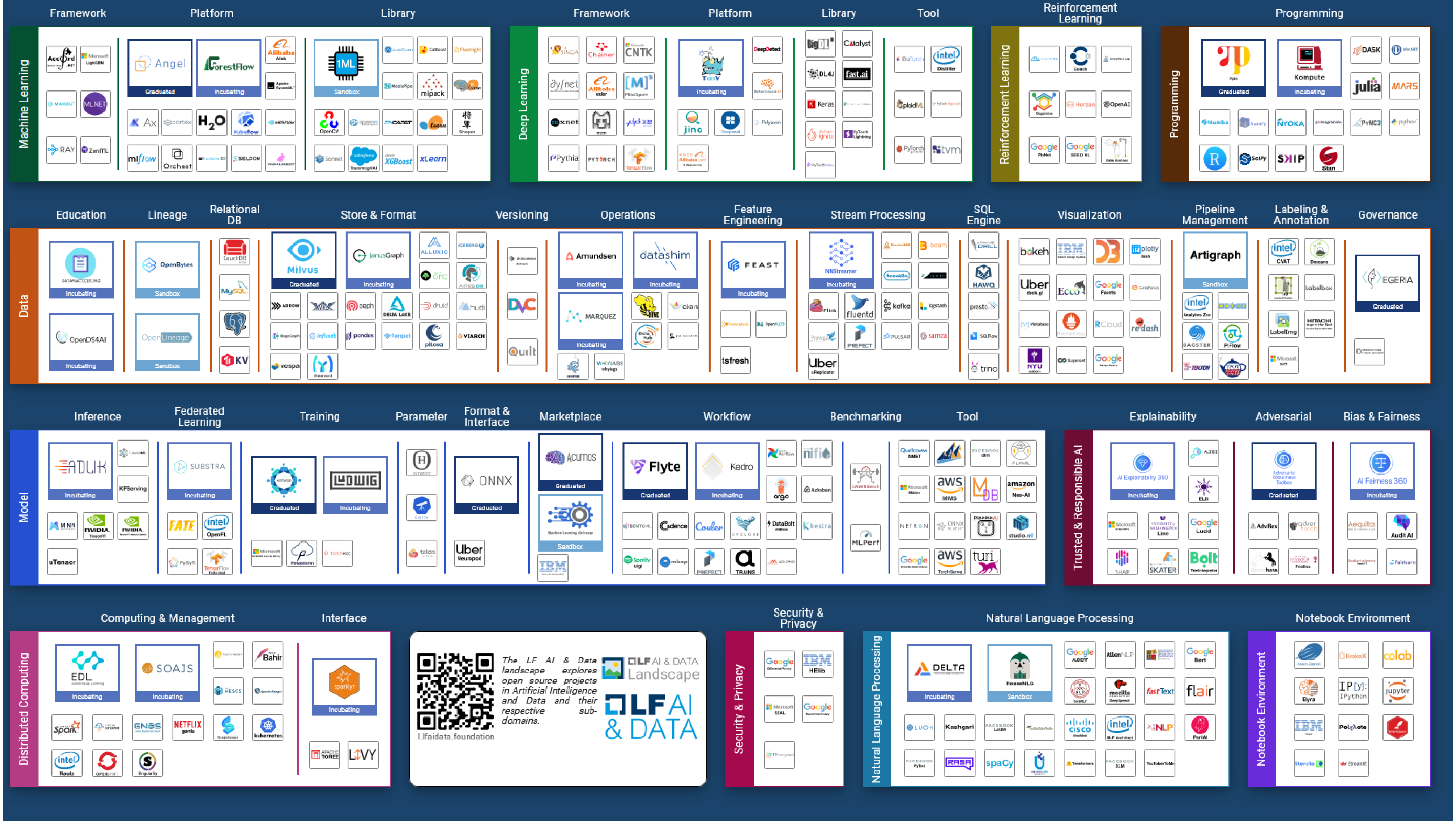
Topic 4:
Coding & Tools, and Automation

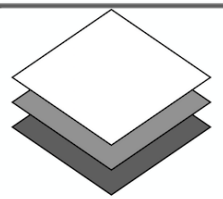
Full Stack Deep Learning @full_stack_dl · 3h

🛠️ Tooling Tuesday 🛠️

📢📢📢 This week: @streamlit 📢📢📢

"In my experience, every non-trivial ML project is eventually stitched together with bug-ridden and un-maintainable internal tools" -- @myelbows (co-founder / CEO of streamlit)


Machine Learning
Framework
Platform
Library
Deep Learning
Framework
Platform
Library
Tool
Reinforcement Learning
Programming
Data
Education
Lineage
Relational DB
Store & Format
Versioning
Operations
Feature Engineering
Stream Processing
SQL Engine
Visualization
Pipeline Management
Labeling & Annotation
Governance
Model
Inference
Federated Learning
Training
Parameter
Format & Interface
Marketplace
Workflow
Benchmarking
Tool
Trusted & Responsible AI
Explainability
Adversarial
Bias & Fairness
Distributed Computing
Computing & Management
Interface
Security & Privacy
Natural Language Processing
Notebook Environment
Graduated
Incubating
Sandbox
The LF AI & Data landscape explores open source projects in Artificial Intelligence and Data and their respective sub-domains.
l.lfaidata.foundation
LF AI & DATA Landscape
LF AI & DATA



Credits to fullstackdeeplearning.com

Amazon SageMaker · gradient by Paperspace · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Pachyderm, DVC, LIQUIDATA

Labeling: figure eight, scale, Aquarium

Processing: Spark, DAGSTER

Exploration: pandas, RAPIDS, dbt

Data Lake / Warehouse: snowflake, databricks

Sources: S3, LOG, Parquet

Training/Evaluation

Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, K, HOROVOD

Hyperparameter Tuning: SIGOPT, Determined AI, Weights & Biases, tune

Experiment Management: Weights & Biases, comet, TensorBoard, Neptune, mlflow

Resource Management: Determined AI, PY, slurm

Software Engineering: jupyter, Streamlit, git

Compute: CoreWeave, Lambda

Deployment

Feature Store: tecton

Monitoring: fiddler

Edge: ML, NVIDIA TensorRT, TensorFlow Lite, ONNX

Web: ALGORITHMIA, SELDON

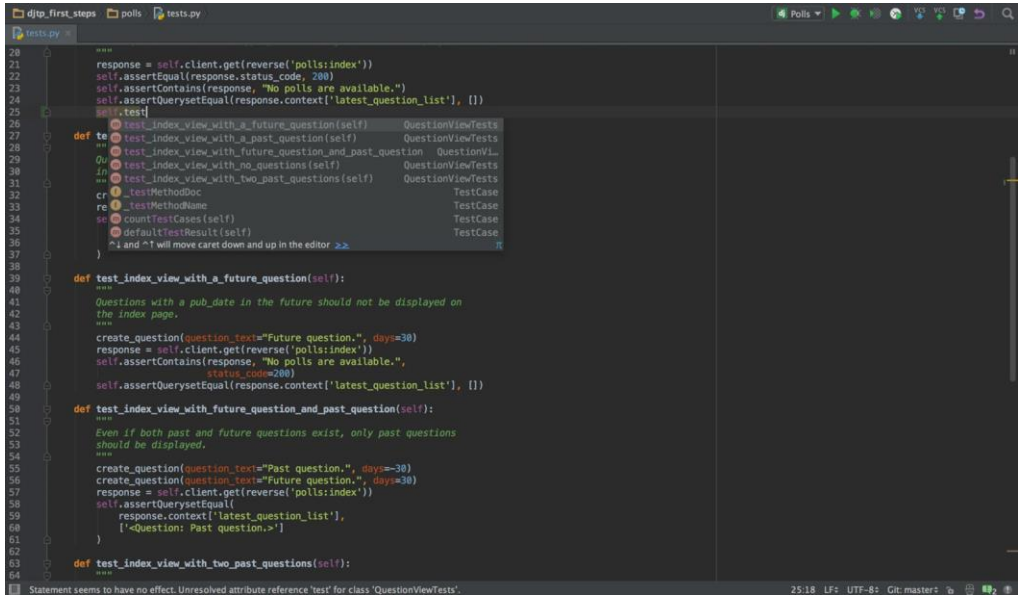
CI / Testing: Buildkite, great_expectations

or or

Features of Traditional IDEs and Development

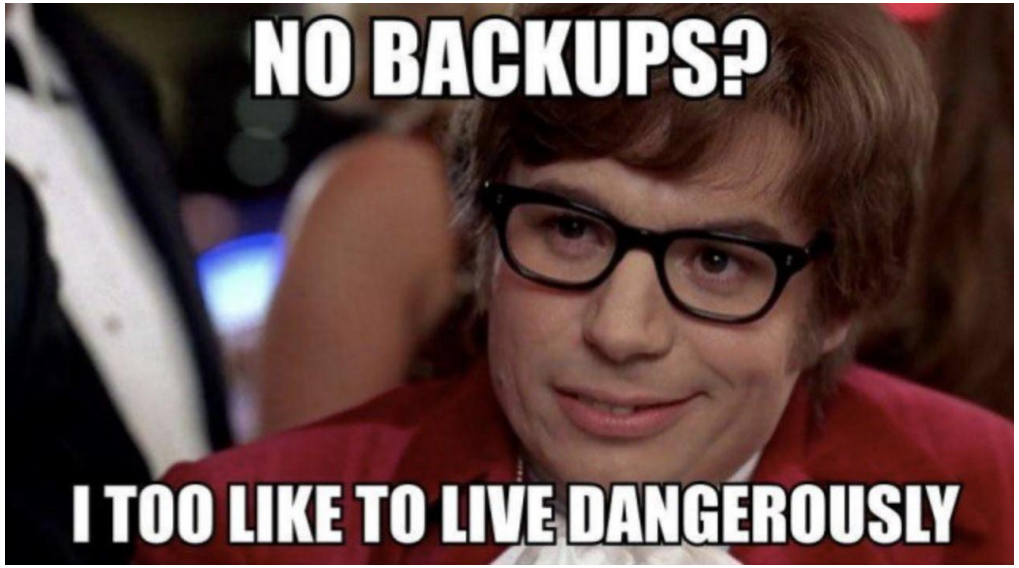
IDEs support:

- Documentation support and API lookup
- Syntax highlighting
- Auto completion / code suggestions
- Integrated VCS
- Search and refactoring capabilities
- Integration with unit tests
- Production of distributable source code files (e.g., via a bui
- Multiple developers working on the same document



Typical IDEs: PyCharm, Visual Studio Code, Spider, Eclipse+PyDev

Pain Points of Notebooks



Andrej Karpathy ✓
@karpathy

so I accidentally held down something and deleted all cells in this jupyter notebook I've been building for ~2 months, and the "undo delete cell" isn't bringing them back. Lol.

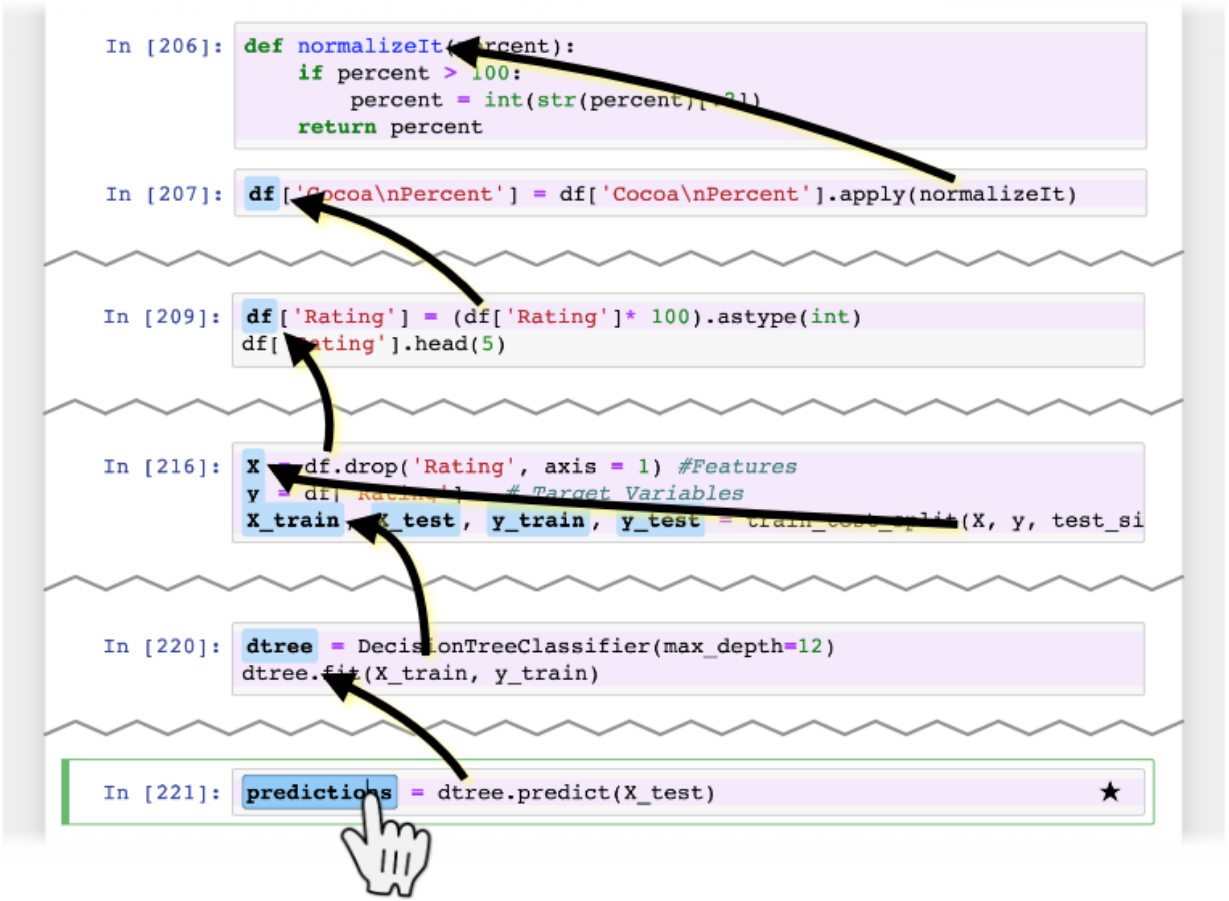
9:27 AM · Aug 30, 2020 · Tw

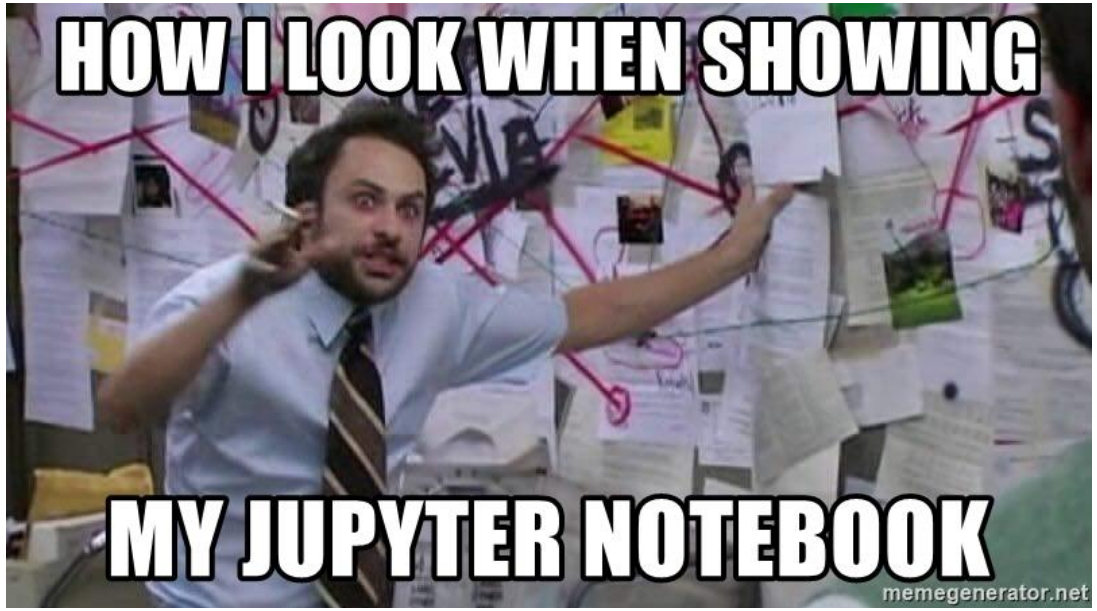
Andrej Karpathy ✓
@karpathy

I'm still shook. Some jupyter hotkey, somehow held down with my left palm, just iteratively deletes everything and undo doesn't bring them back (it creates an empty cell only). Hug your favorite notebooks and keep them safe ❤️

9:32 AM · Aug 30, 2020 · Twitter Web App

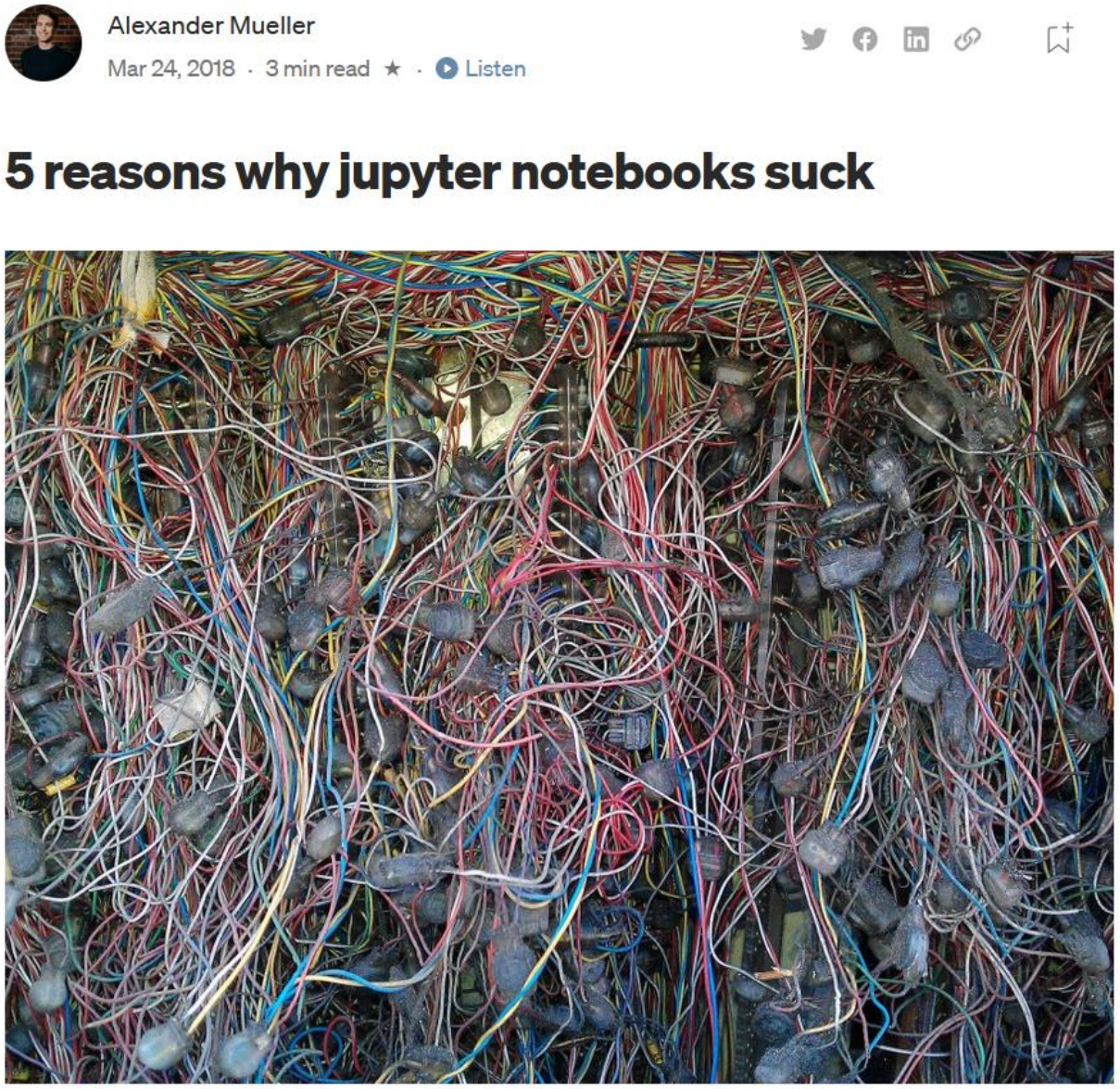
Problem: Complex data- and control-flow





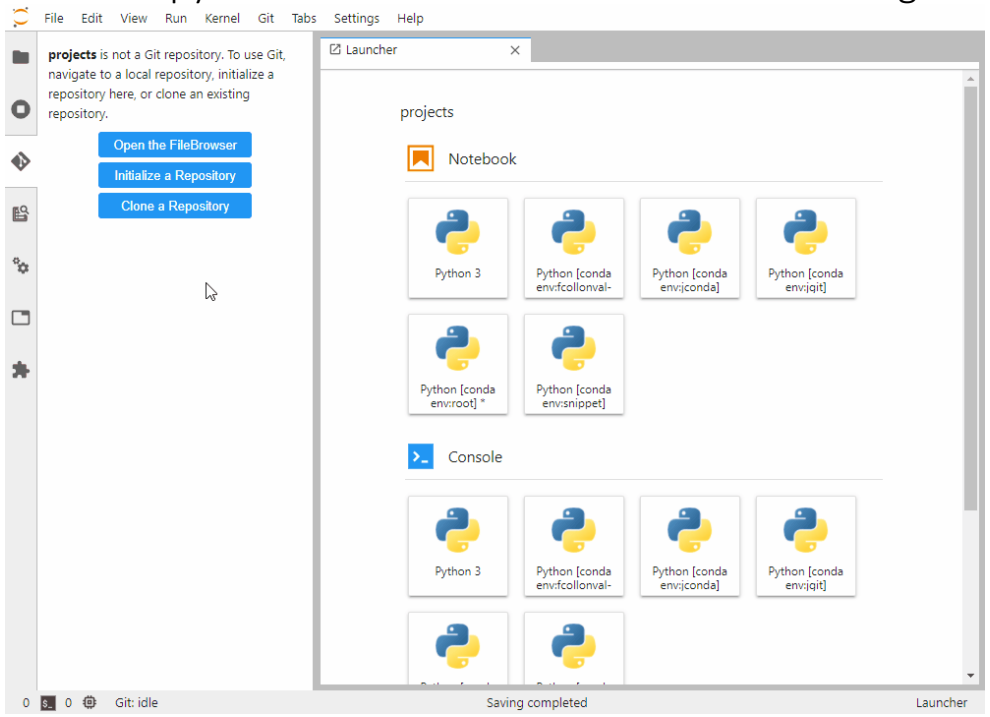
Version control
Primitive IDE
Hard to test
Out-of-order execution
Hard to run long or distributed task

All of these problems have solutions!



Version Control with Notebooks

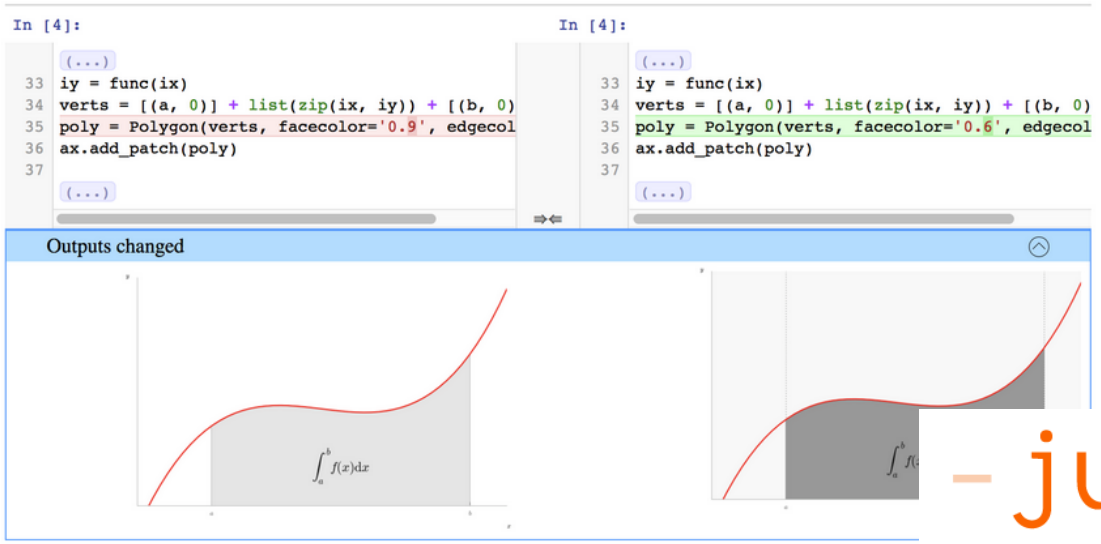
A JupyterLab extension for version control using Git



nbdime – diffing and merging of Jupyter Notebooks

Version: 3.1.1.dev

nbdime provides tools for diffing and merging Jupyter notebooks.



– jupy

+ text

Jupyter Notebooks as Markdown Documents, Julia, Python or R Scripts

Using py files, but showing them as notebooks

Notebooks as Novel Development? Maybe..

Managing Messes in Computational Notebooks

Andrew Head
UC Berkeley
andrewhead@berkeley.edu

Fred Hohman
Georgia Institute of Technology
fredhohman@gatech.edu

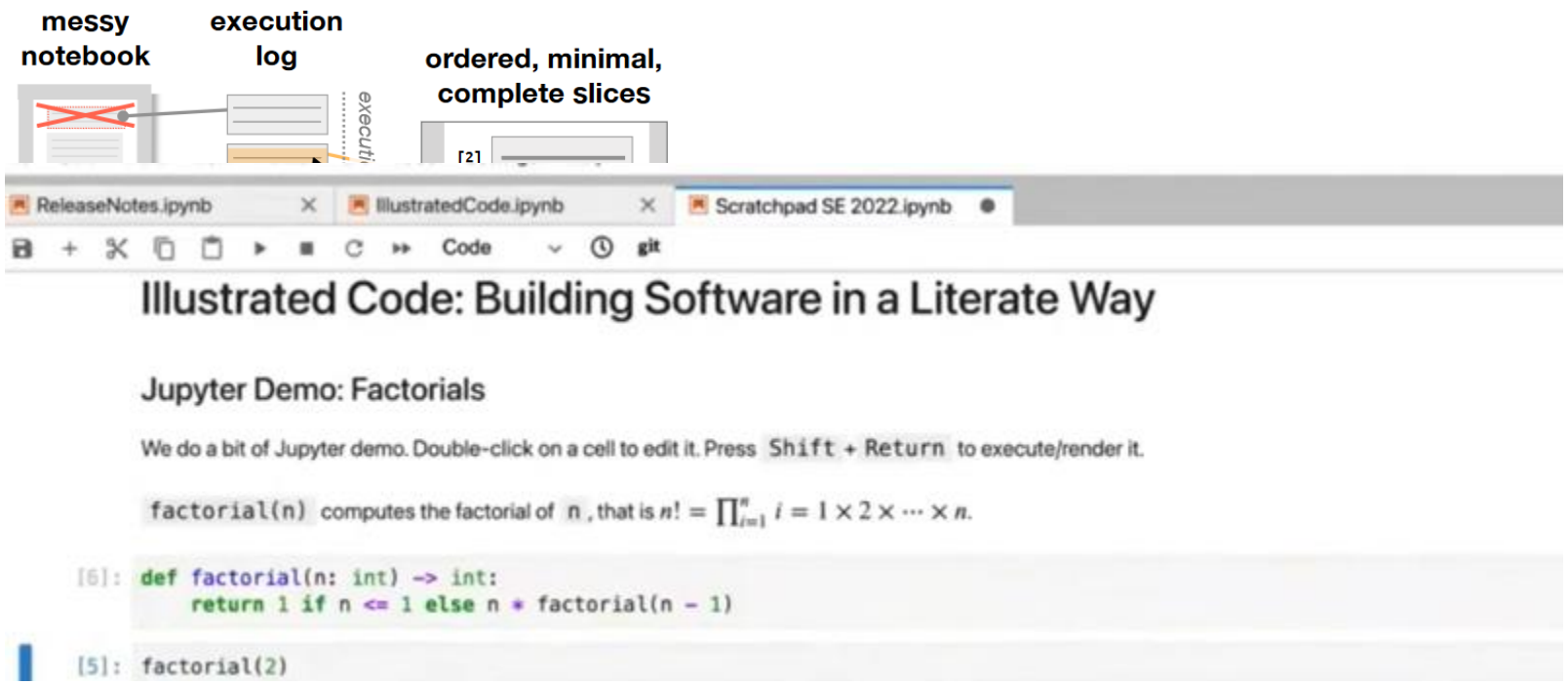
Titus Barik
Microsoft
titus.barik@microsoft.com

Steven M. Drucker
Microsoft Research
sdrucker@microsoft.com

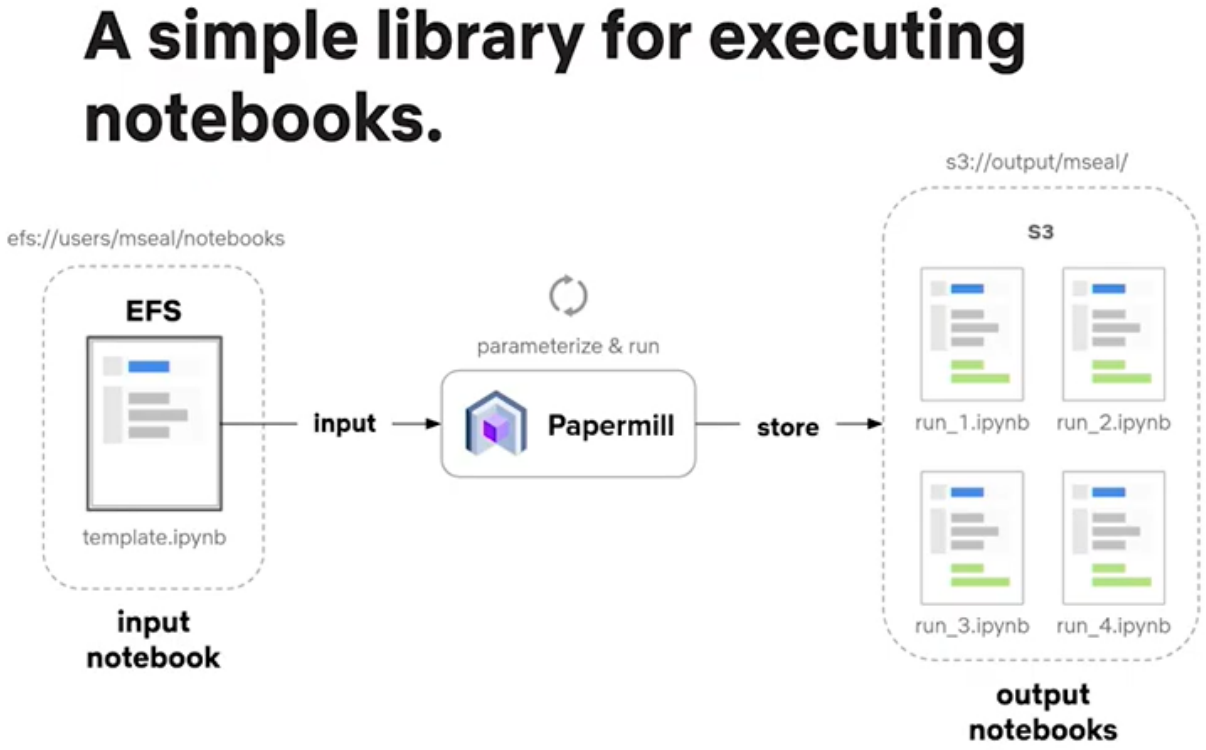
Robert DeLine
Microsoft Research
rob.deline@microsoft.com

STRACT

a analysts use computational notebooks to write code analyzing and visualizing data. Notebooks help analysts atively write analysis code by letting them interleave e with output, and selectively execute cells. However, as lysis progresses, analysts leave behind old code and out-, and overwrite important code, producing cluttered and



Papermill: Programmatically Working with NBs



Enables scaling, remote execution, and ***testing*** of notebooks

```python
import json
from pathlib import Path

import papermill as pm

def test_my_notebook()
    # execute notebook
    pm.execute_notebook('my-nb.ipynb', 'output.ipynb')

    # ensure notebook creates model
    assert Path('model.pickle').is_file()

    # check it stores metrics
    metrics = json.loads(Path('metrics.json').read_text())
    assert list(metrics.keys()) == ['accuracy', 'precision', 'recall']
```

nbs-myths-0.py hosted with ❤ by GitHub

Unit-Tests for NBs: testbook

Welcome to testbook

testbook is a unit testing framework for testing code in Jupyter Notebooks.

Previous attempts at unit testing notebooks involved writing the tests in the notebook itself. However, testbook will allow for unit tests to be run against notebooks in separate test files, hence treating `.ipynb` files as `.py` files.

Here is an example of a unit test written using testbook

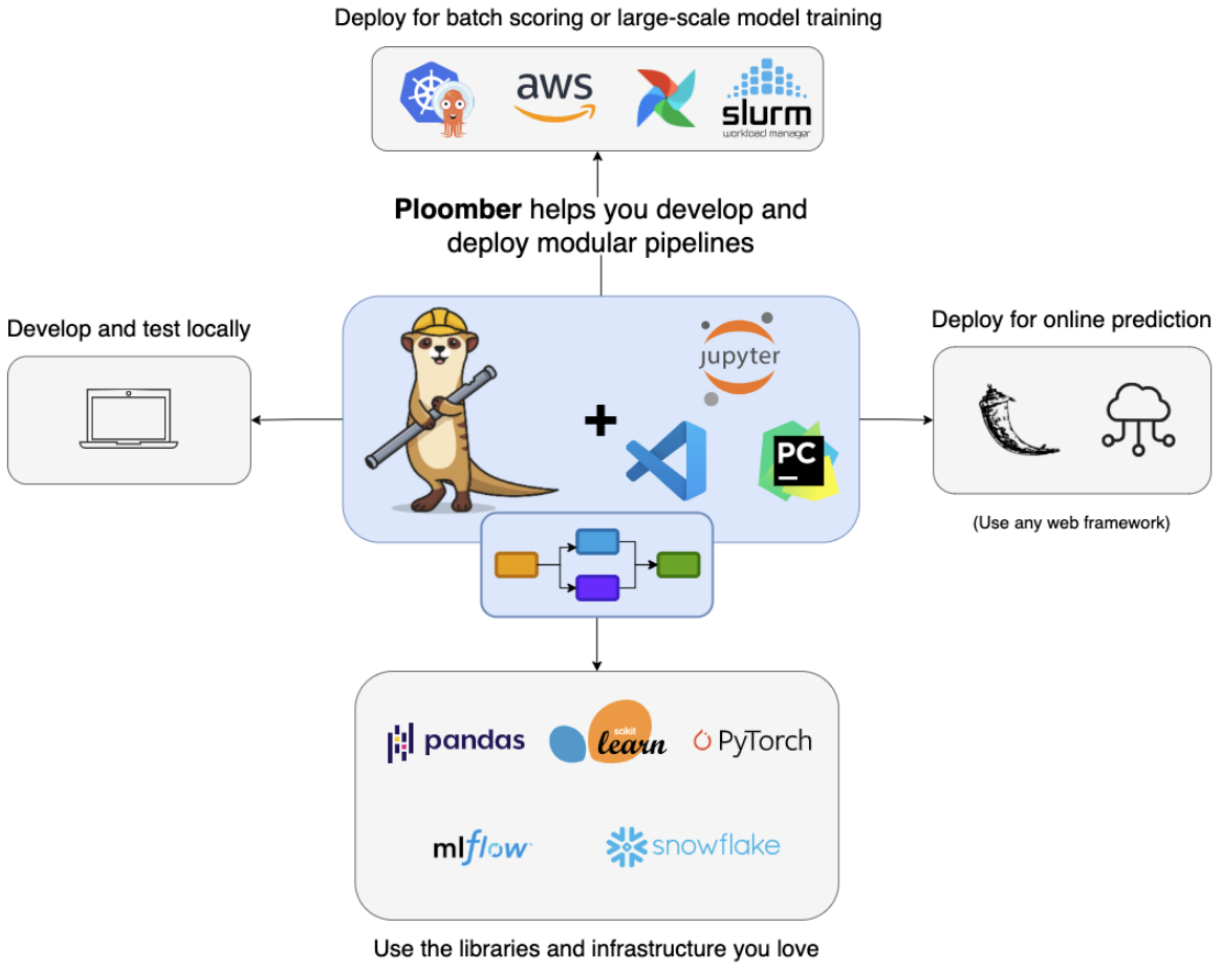
Consider the following code cell in a Jupyter Notebook:

```
def func(a, b):
    return a + b
```

You would write a unit test using `testbook` in a Python file as follows:

```
from testbook import testbook


@testbook('/path/to/notebook.ipynb', execute=True)
def test_func(tb):
    func = tb.ref("func")

    assert func(1, 2) == 3
```

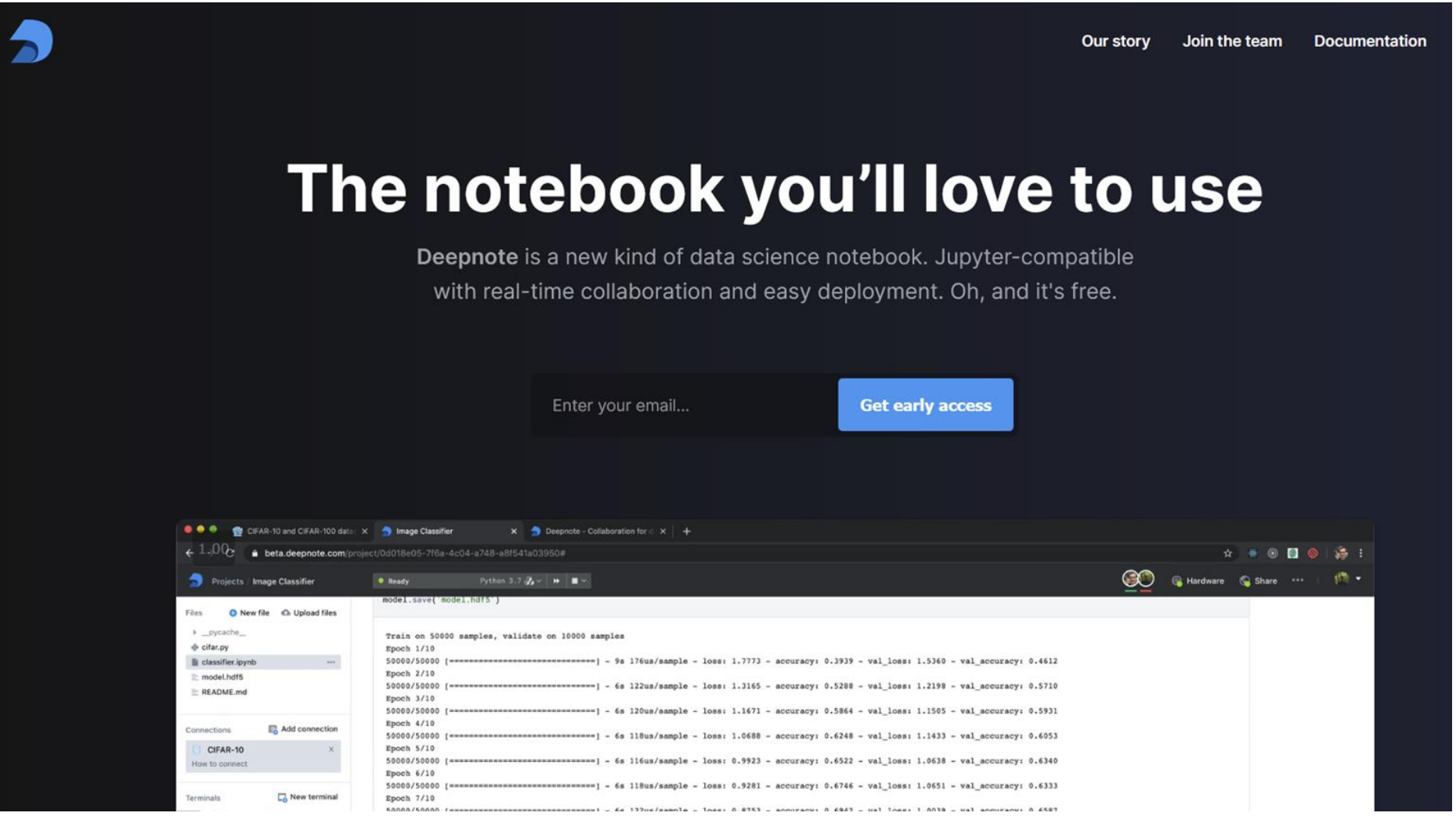
Ploomber: Modularize Notebooks as Pipelines



Idea: Build pipeline with source code files in the background and notebooks for user interaction

```yaml
tasks:
  - source: get.py
    product:
      data: output/raw.csv
      nb: output/get.ipynb

  - source: clean.py
    product:
      data: output/clean.csv
      nb: output/clean.ipynb

  - source: plot.py
    product: output/report.ipynb
```

Tools: Better Notebooks



Deepnote.com

- Versioning of notebooks
- Python scripts visualized as notebooks
- Real-time collaboration
- Shell support
- GPU-included cloud support
- Standardized environments for reproducibility

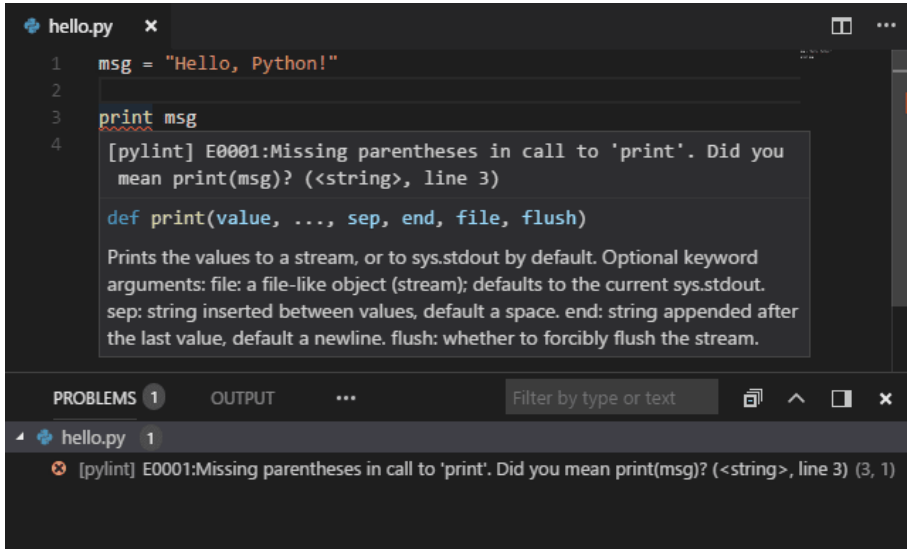
Tools for Notebook-Driven Development: NBDEV

Python programming environment to develop complete python packages, including tests and a rich documentation system (developed by Fast.AI)

Fosters *exploratory programming with SE features*:

- Find out how an unknown API works / behaves
- Experiment with an algorithm to meet functional and non-functional requirements
- Design, refactor, and debug code for different design and input decisions
- Generate docs from Jupyter notebooks
- Automate the publishing to pypi and conda packages
- Two-way sync of notebooks and code
- Write tests directly in cells
- Merge and conflict resolution
- Integrates with GitHub actions
- Integration with GitHub pages
- Latex math support and so on…

Linters & Type Hints



Linters ensure certain static code properties

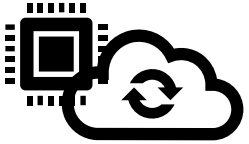
- Code guidelines for clean code
- Static analysis for certain “compile-time” bugs
- Pylint, flake8, mypy, pycodestyle, etc.

```python
from typing import List

def should_use(annotations: List[str]) -> bool:
    print("They're awesome!")
    return True
```

Type hints

- Language feature
- Enables to define the type of a variable / parameter
- Makes it possible to rune linters and checking for type errors
- Easier to develop when knowing the type (document)

(Scaling) Infrastructure of MLOps



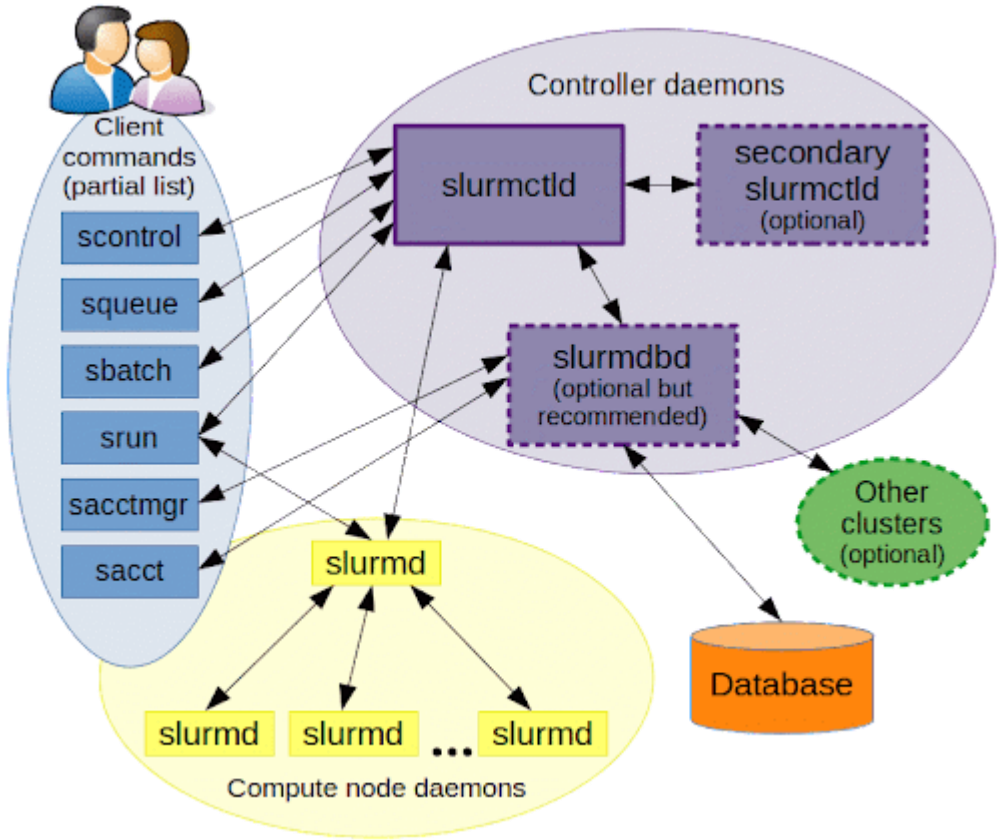
Local initial experiments with
small number of GPUs.

Individual experiments in notebooks
Difficult to reproduce
No larger models to train
Limited number of experiments due to combinatorial explosion

Expensive, but vast amount of
resources, with scaling opportunities

User management required
Selection and allocation of resources
Everything costs money
Orchestration challenges
Configuration management
Secrets
Runs large number of experiments in parallel

Slurm: Job Scheduler

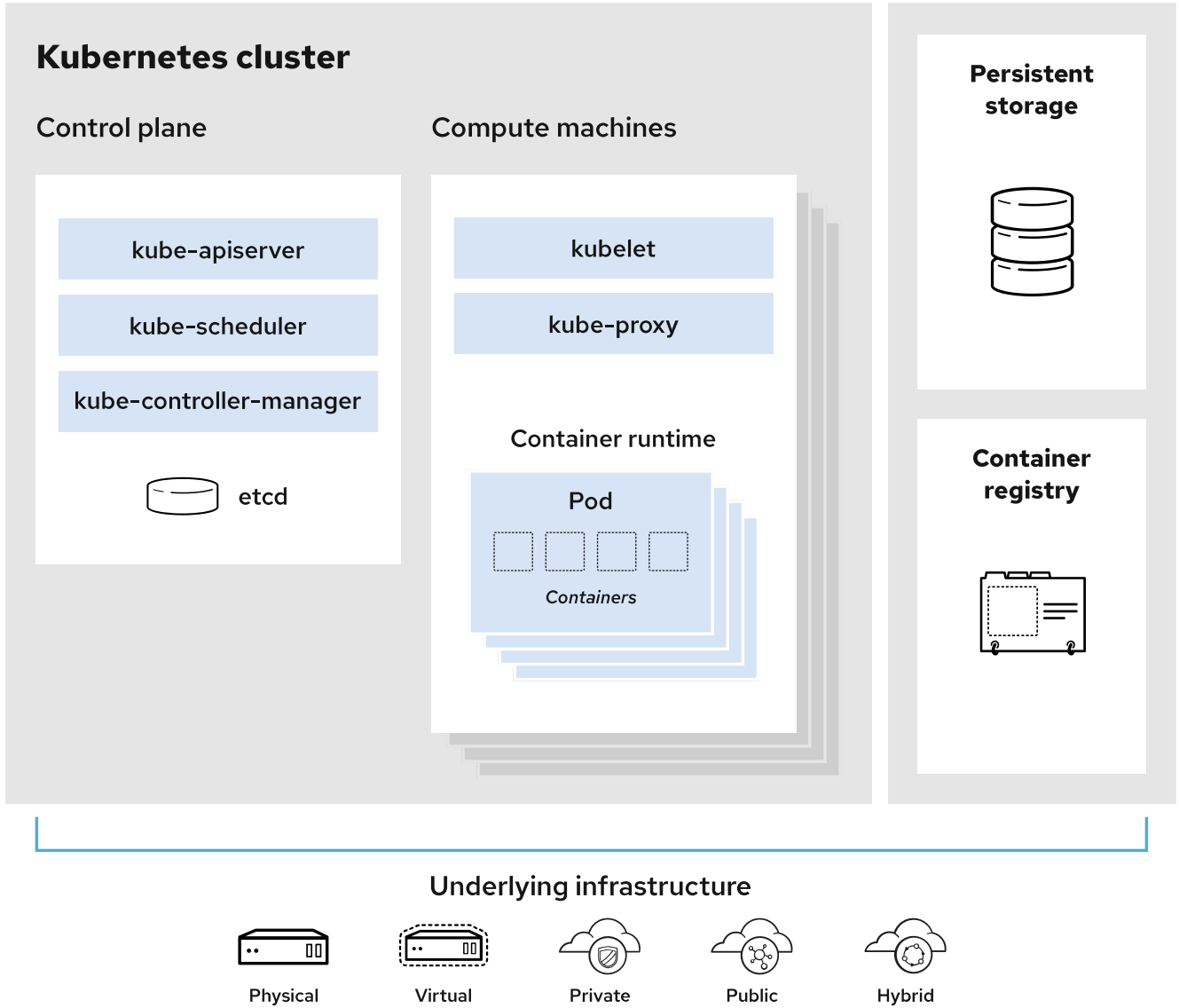


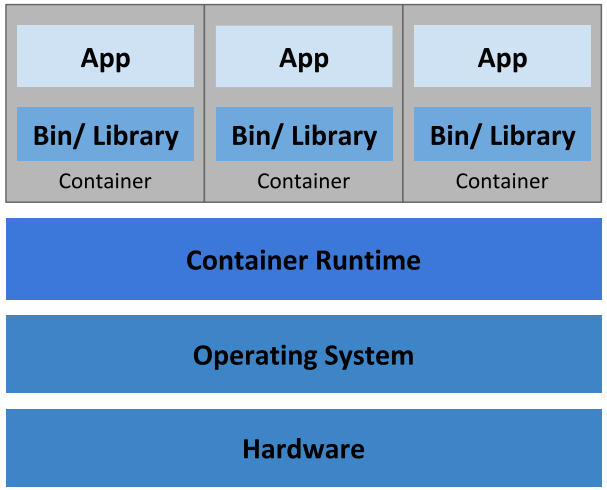

```
#!/bin/bash
#SBATCH -J myjob_4GPUs
#SBATCH -o myjob_4GPUs_%j.out
#SBATCH -e myjob_4GPUs_%j.err
#SBATCH --mail-user=florin@mit.edu
#SBATCH --mail-type=ALL
#SBATCH --gres=gpu:4
#SBATCH --gpus-per-node=4
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --mem=0
#SBATCH --time=24:00:00
#SBATCH --exclusive

## User python environment
HOME2=/nobackup/users/$(whoami)
PYTHON_VIRTUAL_ENVIRONMENT=wmlce-1.7.0
CONDA_ROOT=$HOME2/anaconda3

## Activate WMLCE virtual environment
source ${CONDA_ROOT}/etc/profile.d/conda.sh
conda activate $PYTHON_VIRTUAL_ENVIRONMENT
ulimit -s unlimited

## Creating SLURM nodes list
export NODELIST=nodelist.$
srun -l bash -c 'hostname' |  sort -k 2 -u | awk -vORS=, '{print $2":4"}' | sed 's/,$//' > $NODELIST

## Number of total processes
echo " "
echo " Nodelist:= " $SLURM_JOB_NODELIST
echo " Number of nodes:= " $SLURM_JOB_NUM_NODES
echo " GPUs per node:= " $SLURM_JOB_GPUS
echo " Ntasks per node:= "  $SLURM_NTASKS_PER_NODE


####    Use MPI for communication with Horovod - this can be hard-coded during installation as well.
export HOROVOD_GPU_ALLREDUCE=MPI
export HOROVOD_GPU_ALLGATHER=MPI
export HOROVOD_GPU_BROADCAST=MPI
export NCCL_DEBUG=DEBUG

echo " Running on multiple nodes/GPU devices"
echo ""
echo " Run started at:- "
date
```

Finds machines for your scripts
Controls access to machines
User management with budgets
Define requirements on resources
Synchronization features, etc.

Kubernetes + Docker

How to manage containers? How to scale? How to discover services? How to manage resources? How to manage configuration and secrets?





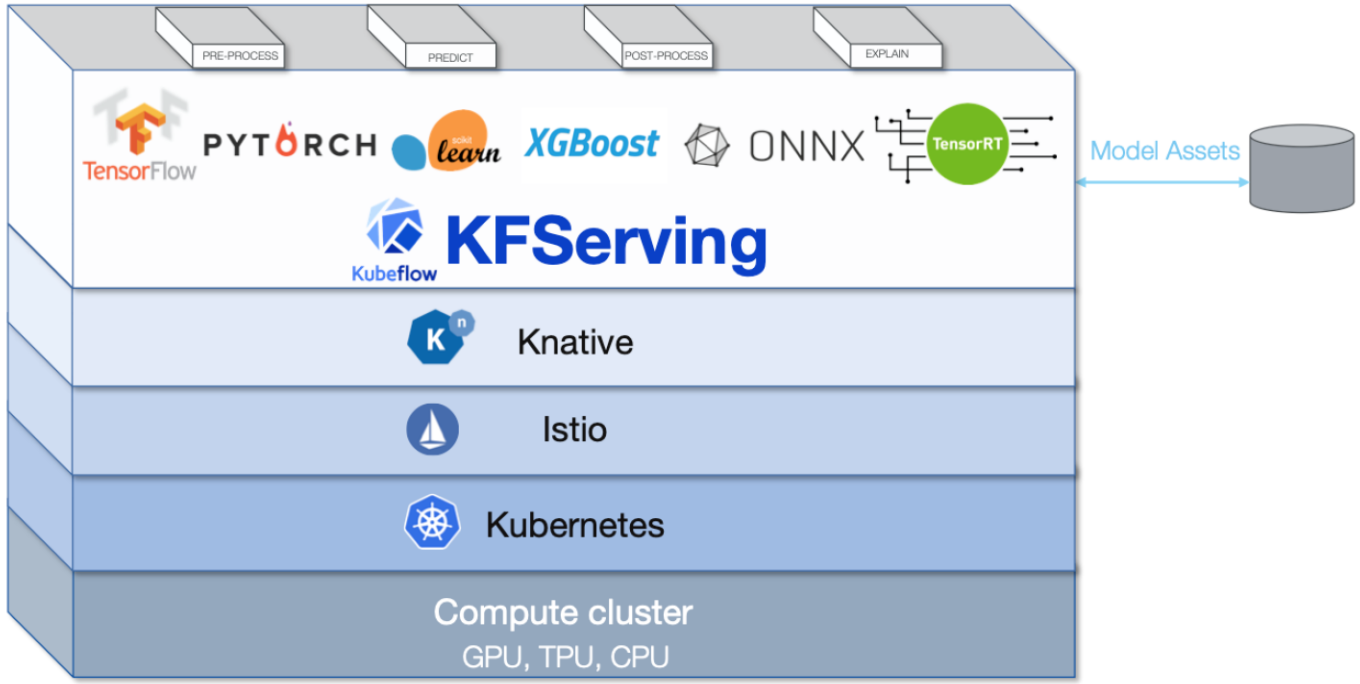
Container Deployment

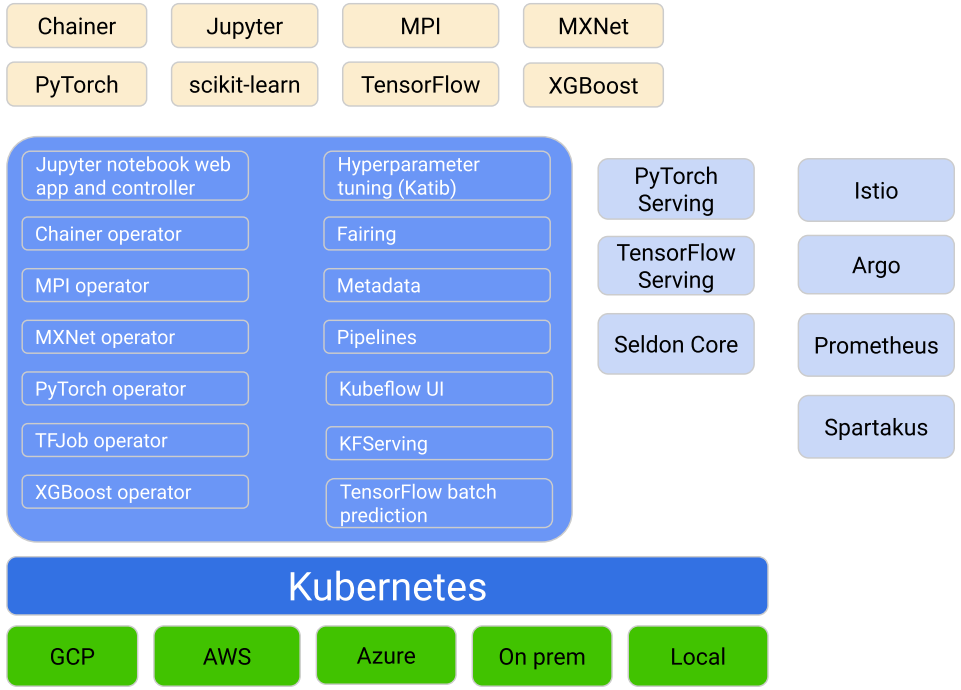
But, no natural support for ML / AI processes. All tools must be installed manually and connected.

Placing ML Tools on Top of Kubernetes

Kubeflow

Manage Jupyter notebooks
Plugins for hyperparameter tuning and model deployment
Make workflows portable, scalable, and easily deploy it on a K8s cluster

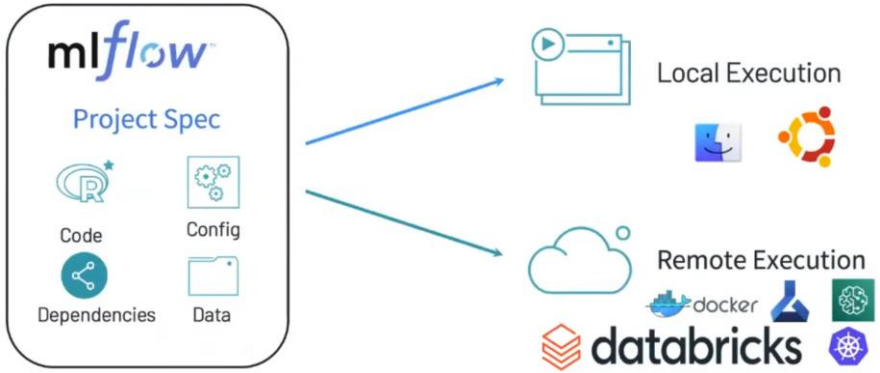




ML*flow* Projects

Goal: Package an ML project in way that it can be run on any platform (portability)

Problem: Diverse set of tools and environments to run your project on (see similarities to development environments in software development)



Solution: Bundles all ML code, libraries, dependencies, data, and configurations together

ML*flow* Project Structure

project_dir/

- MLproject

```
conda_env: conda.yaml

entry_points:
  main:
    parameters:
      training_data: path
      lambda: {type: float, default: 0.1}
    command: python main.py {training_data} {lambda}
```

Path to stored data

Optional start arguments

- conda.yaml / Dockerfile ← Specifies environment

```
channels:
- defaults
dependencies:
- python=3.7.3
- scitkit-learn=0.20.3
- pip:
  - mlflow
  - cloudpickle==0.8.0
name: mlflow-env
```

- main.py
- model.py

Execute project via: $mlflow run git://<project_dir>.git -P lambda=0.2

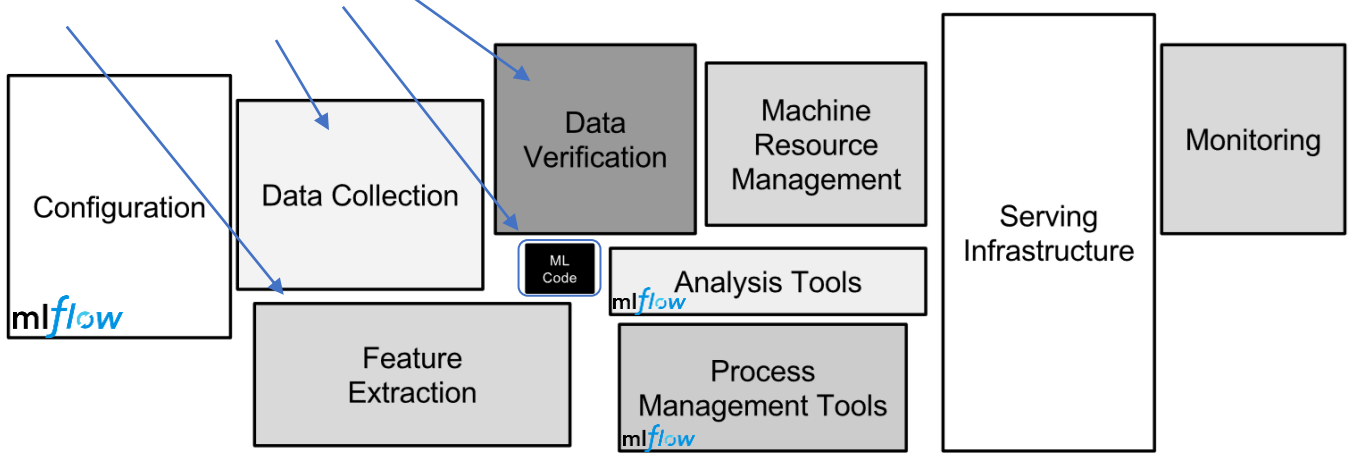
Execute project programmatically via: mlflow.run("git://<project_dir>", parameters={..})

Execute project locally via: mlflow run . -e main -P lambda=0.2

Building an ML*flow* Project

1. Create MLflow project file: specify entry point(s) + arguments
2. Create a conda.yaml: specify dependencies (pro tip: get this file from ML*flow* experiment tracking; see experimentation later)
3. Create a GitHub repository and upload files + data (dvc/LFS) + code
4. Test it locally with mlflow run git://… (it will clone it locally, install env, run the entry point)
5. Share the project with others as it is now "portable"

Multi-stage project entry points:




```yaml
name: multistep_example

conda_env: conda.yaml

entry_points:
  load_raw_data:
    command: "python load_raw_data.py"

  etl_data:
    parameters:
      ratings_csv: path
      max_row_limit: {type: int, default: 100000}
    command: "python etl_data.py --ratings-csv {ratings_csv} --max-row-limit {max_row_limit}"

  als:
    parameters:
      ratings_data: path
      max_iter: {type: int, default: 10}
      reg_param: {type: float, default: 0.1}
      rank: {type: int, default: 12}
    command: "python als.py --ratings-data {ratings_data} --max-iter {max_iter} --reg-param {reg_param} --rank {rank}"

  train_keras:
    parameters:
      ratings_data: path
      als_model_uri: string
      hidden_units: {type: int, default: 20}
    command: "python train_keras.py --ratings-data {ratings_data} --als-model-uri {als_model_uri} --hidden-units {hidden_units}"

  main:
    parameters:
      als_max_iter: {type: int, default: 10}
      keras_hidden_units: {type: int, default: 20}
      max_row_limit: {type: int, default: 100000}
    command: "python main.py --als-max-iter {als_max_iter} --keras-hidden-units {keras_hidden_units}
                             --max-row-limit {max_row_limit}"
```

ML*flow* Models

ML frameworks

Inference code

NxM combinations of models and serving technologies

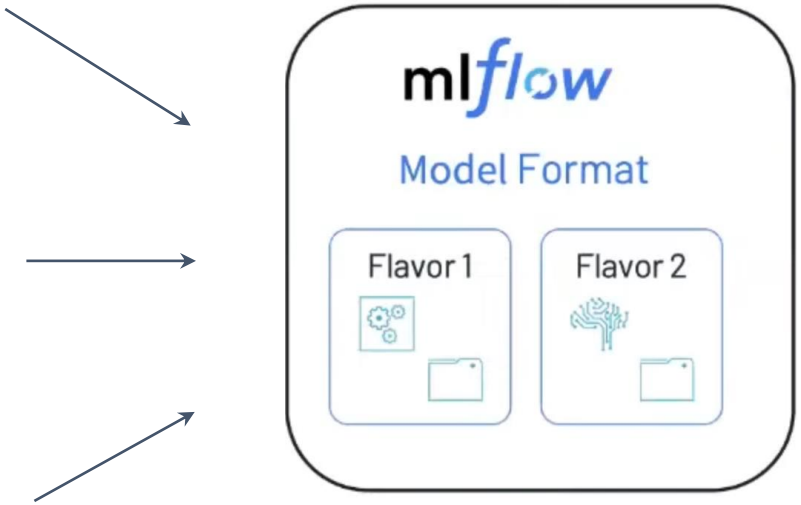
Batch / stream processing

Serving tools

Amazon SageMaker

MLflow Models

ML frameworks



Wrap a model in a function-style way to invoke inference in any Python-supported environment

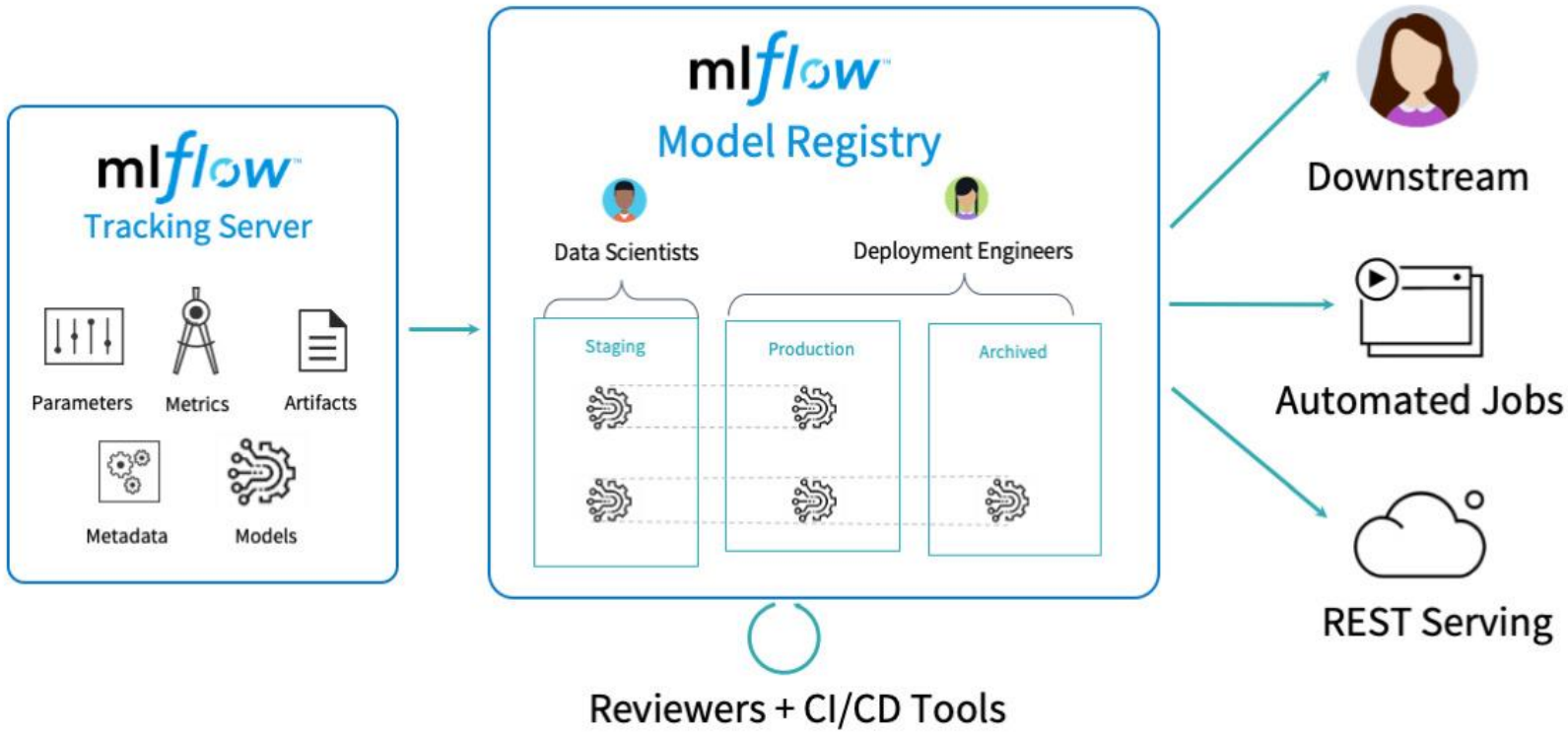
Inference code

Batch / stream processing

Serving tools

Amazon SageMaker

Model Registry



Credit: databricks