Iowa State University

Translational AI Center

Introduction to MLOps

Infrastructure and Tooling

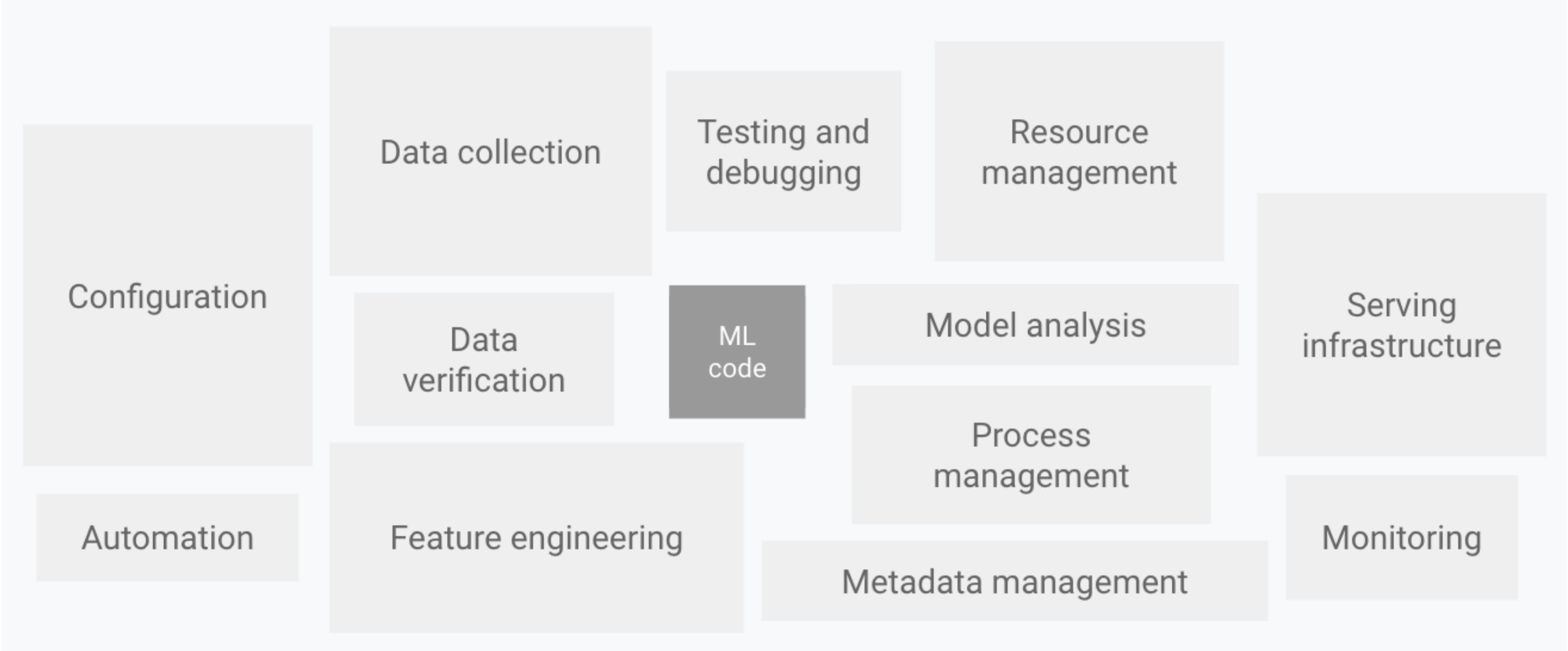
Objectives

- Introduction to MLOps
  - Dream vs. Reality for ML Practitioners
  - The 3 buckets of ML Infrastructure
  - Software Engineering
  - Compute Hardware
  - Resource Management
  - Frameworks and Distributed Training
  - All-in-one Solutions
  - Maturity Model

What is MLOps

- Apply DevOps principles to ML systems (MLOps).
- *MLOps* is an ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops).
- Practicing MLOps means that you advocate for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment and infrastructure management.

MLOps - Overview



DevOPs

- DevOps is a popular practice in developing and operating large-scale software systems. This practice provides benefits such as shortening the development cycles, increasing deployment velocity, and dependable releases. To achieve these benefits, you introduce two concepts in the software system development:
- Continuous Integration (CI)
- Continuous Delivery (CD)

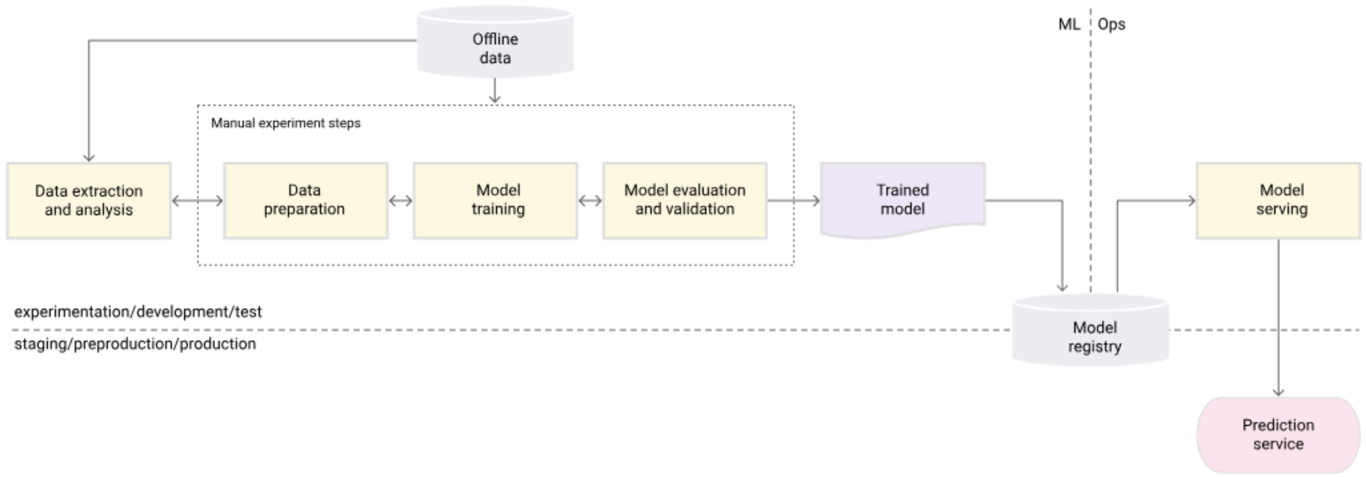
MLOPS vs Devops

- However, ML systems differ from other software systems in the following ways:
- **Team skills:** In an ML project, the team usually includes data scientists or ML researchers, who focus on exploratory data analysis, model development, and experimentation. These members might not be experienced software engineers who can build production-class services.
- **Development:** ML is experimental in nature. You should try different features, algorithms, modeling techniques, and parameter configurations to find what works best for the problem as quickly as possible. The challenge is tracking what worked and what didn't, and maintaining reproducibility while maximizing code reusability.
- **Testing:** Testing an ML system is more involved than testing other software systems. In addition to typical unit and integration tests, you need data validation, trained model quality evaluation, and model validation.
- **Deployment**: In ML systems, deployment isn't as simple as deploying an offline-trained ML model as a prediction service. ML systems can require you to deploy a multi-step pipeline to automatically retrain and deploy model. This pipeline adds complexity and requires you to automate steps that are manually done before deployment by data scientists to train and validate new models.
- **Production:** ML models can have reduced performance not only due to suboptimal coding, but also due to constantly evolving data profiles. In other words, models can decay in more ways than conventional software systems, and you need to consider this degradation. Therefore, you need to track summary statistics of your data and monitor the online performance of your model to send notifications or roll back when values deviate from your expectations.

MLOps Maturity Model

- Helps to clarify the Development Operations (DevOps) principles and practices necessary to run a successful MLOps environment.
- Encompasses five levels of technical capability
  - Level 0: No MLOps
  - Level 1: DevOps but no MLOps
  - Level 2: Automated Training
  - Level 3: Automated Model Deployment
  - Level 4: Full MLOps Automated Operations

MLOps – Level 0



Dream vs. Reality for ML Practitioners

Dream

Provide data

Get optimal prediction system as scalable API or mobile app

Reality

Aggregate, process, clean, label, and version data

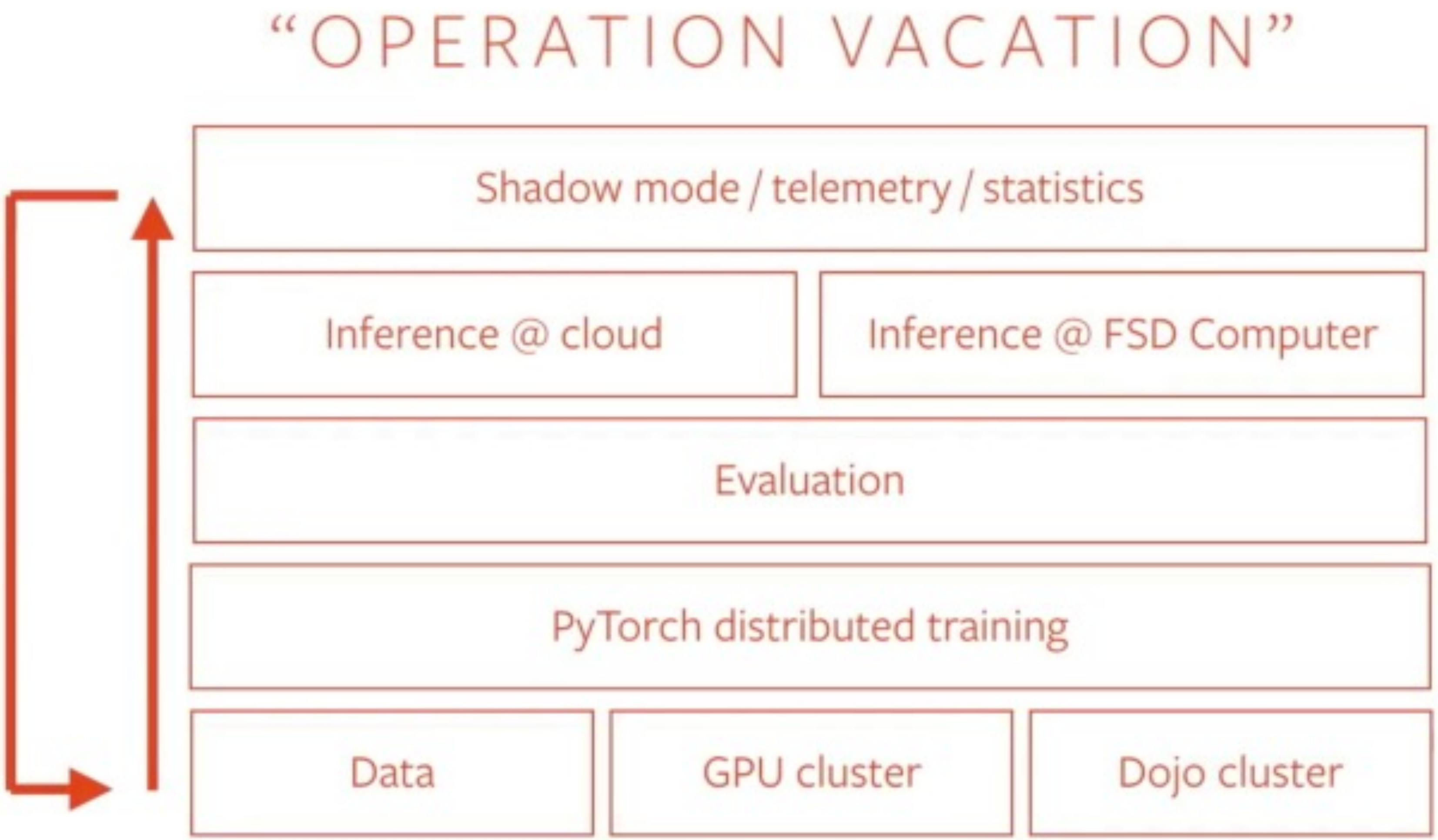
Write and debug model code

Provision compute

Run experiments, review results

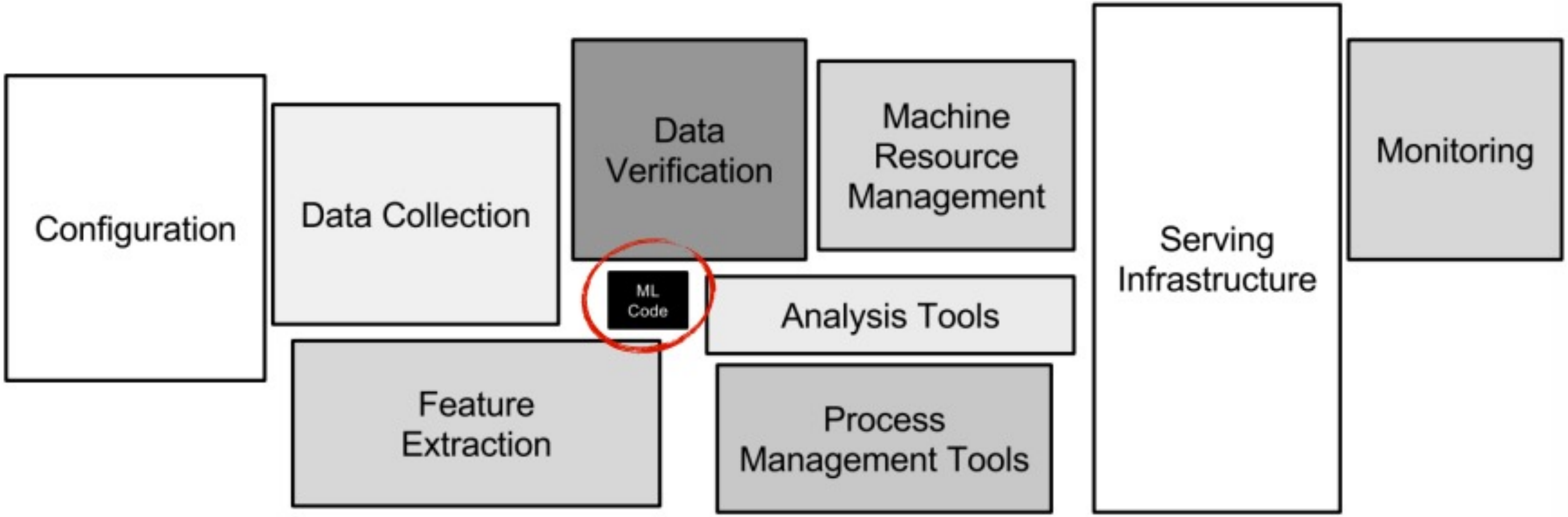
Deploy model

Monitor predictions and close data flywheel loop



Goal: add data, see model improve

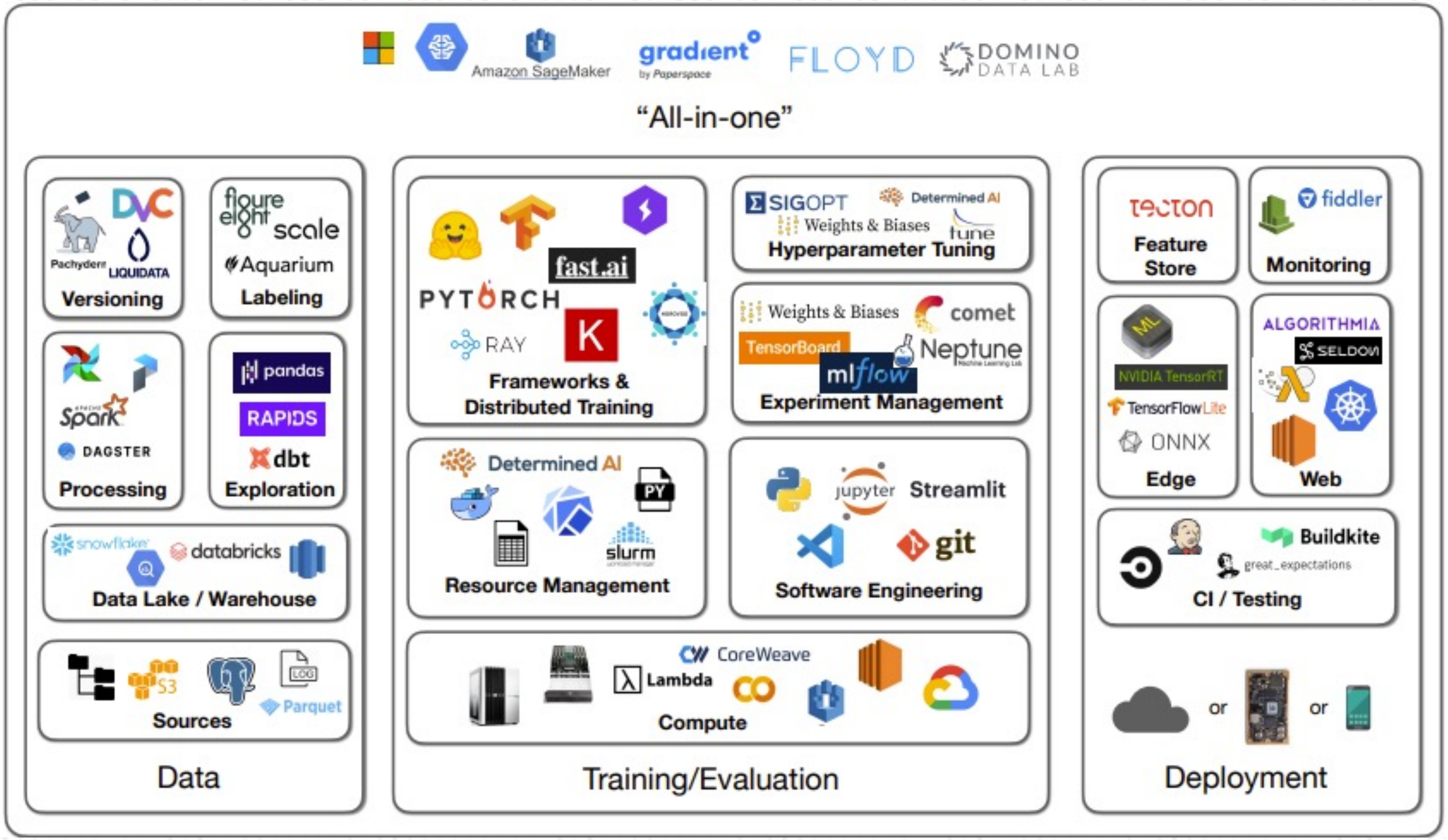
Andrej Karpathy at PyTorch Devcon 2019 - https://www.youtube.com/watch?v=oBklltKXtDE

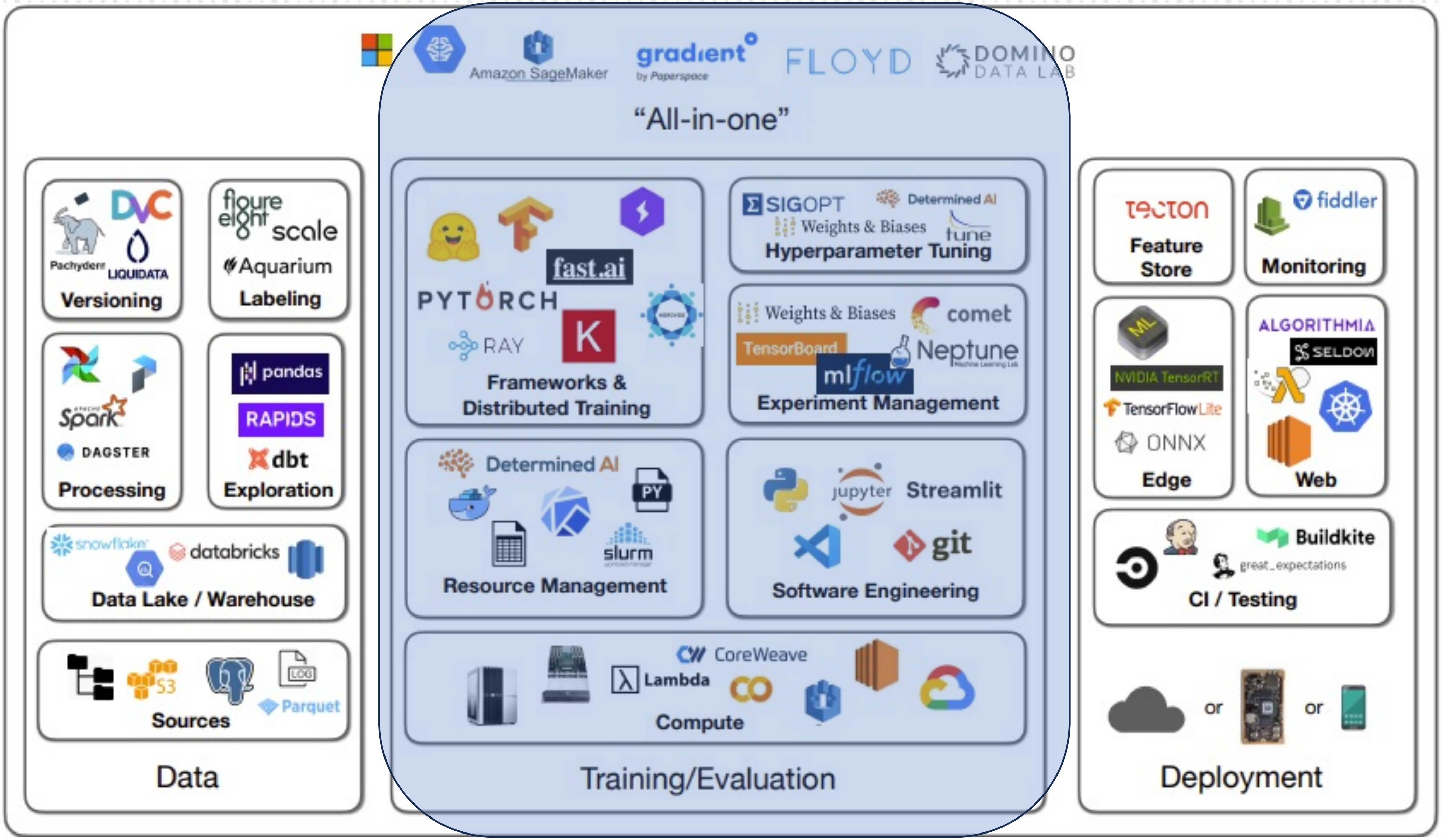


Machine Learning:
The High-Interest Credit Card of Technical Debt

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov,
Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young

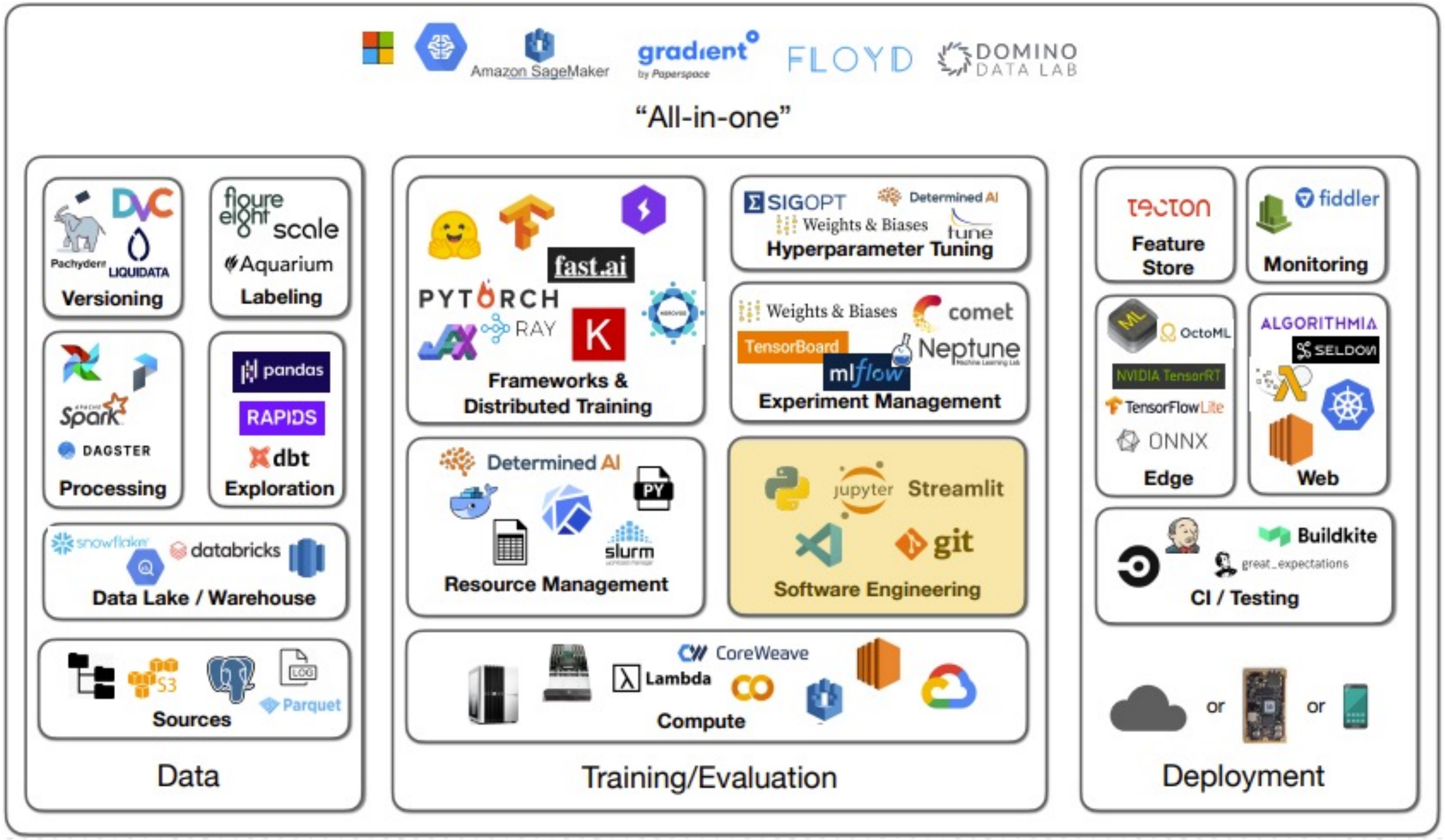
The 3 buckets of ML Infrastructure





Amazon SageMaker
gradient
by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
DVC
Pachyderm
LIQUIDATA
Versioning
figure eight
scale
Aquarium
Labeling
Spark
DAGSTER
Processing
pandas
RAPIDS
dbt
Exploration
snowflake
databricks
Data Lake / Warehouse
S3
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
K
Frameworks & Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
mlflow
Experiment Management
Determined AI
PY
slurm
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Software Engineering

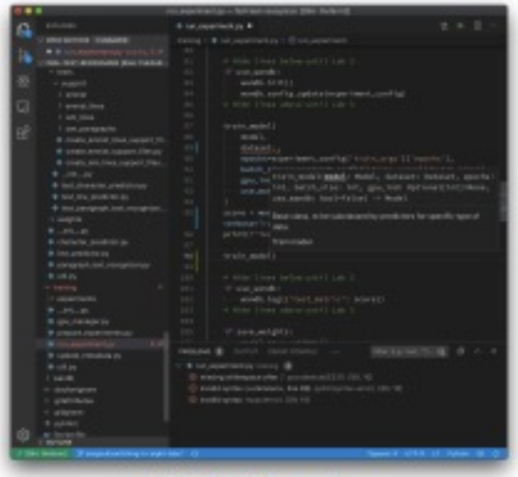


Programming Language

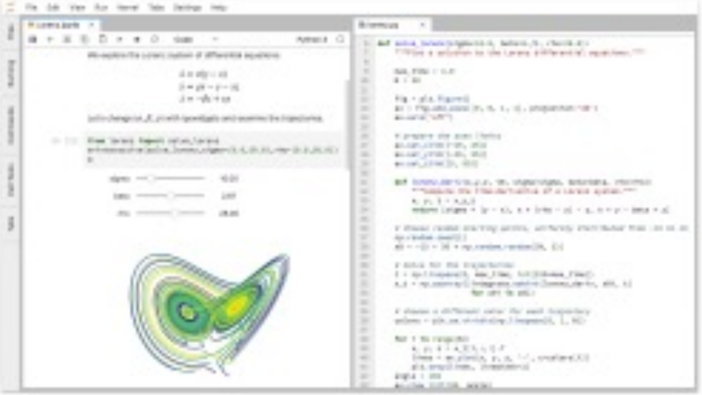
- Python, because of the libraries
    - Clear winner in scientific and data computing

Editors

Vim

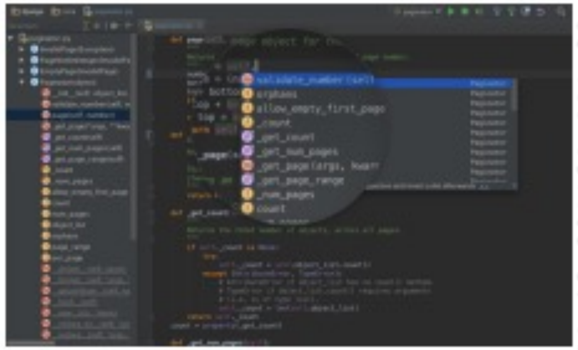


VS Code



Jupyter

Emacs



PyCharm

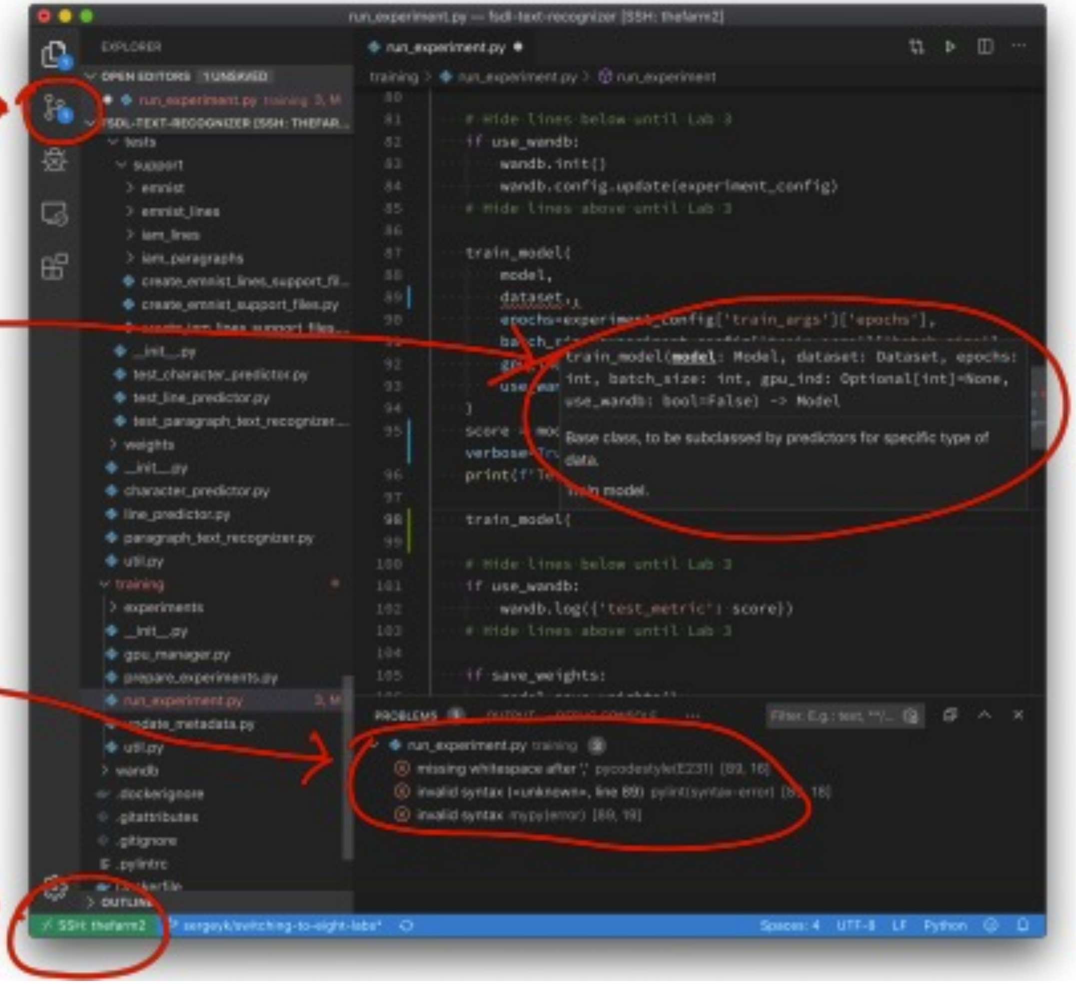
Visual Studio Code

Built-in git staging and diffing

Peek documentation

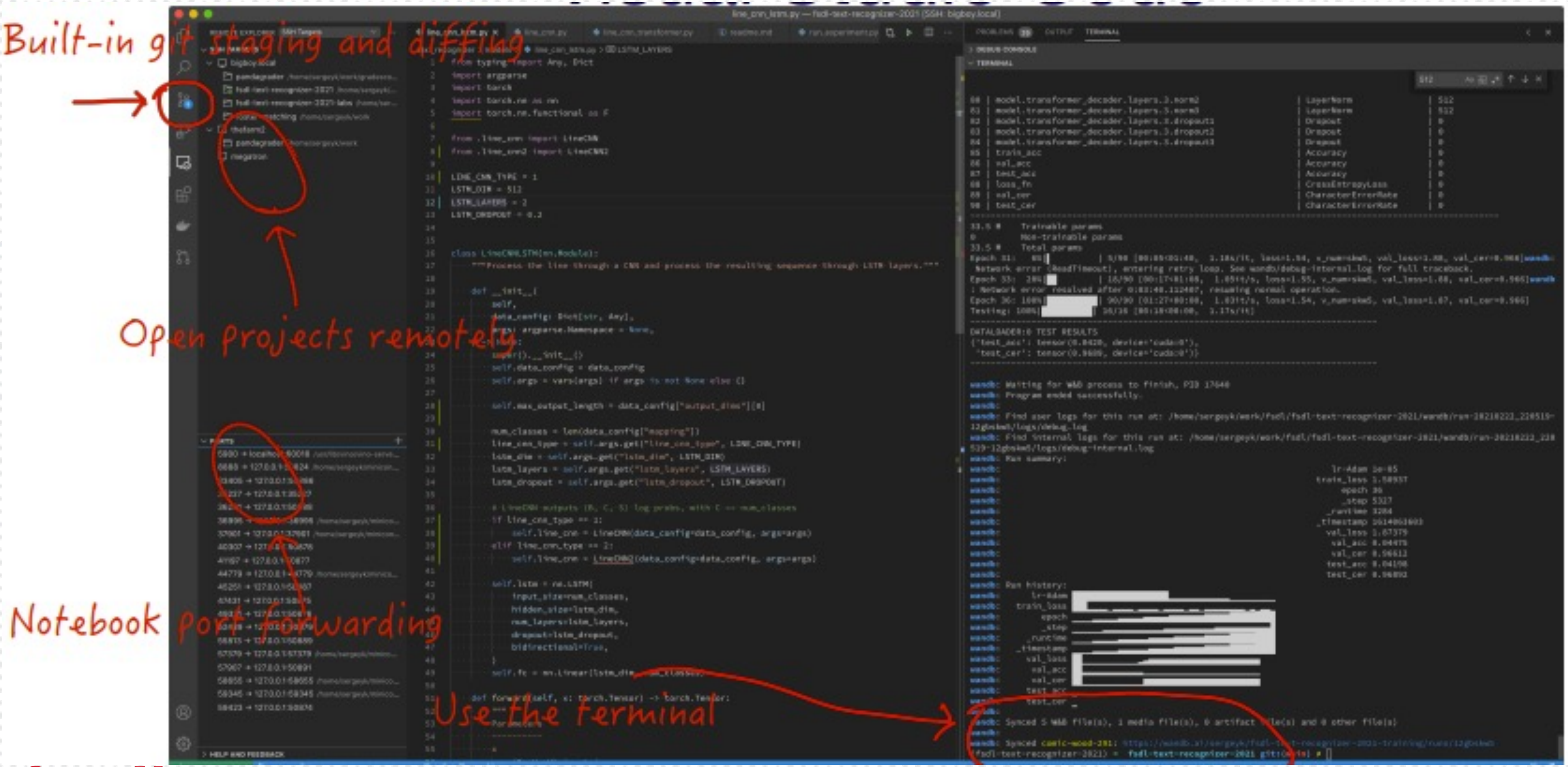
Lint code as your write

Open whole projects remotely



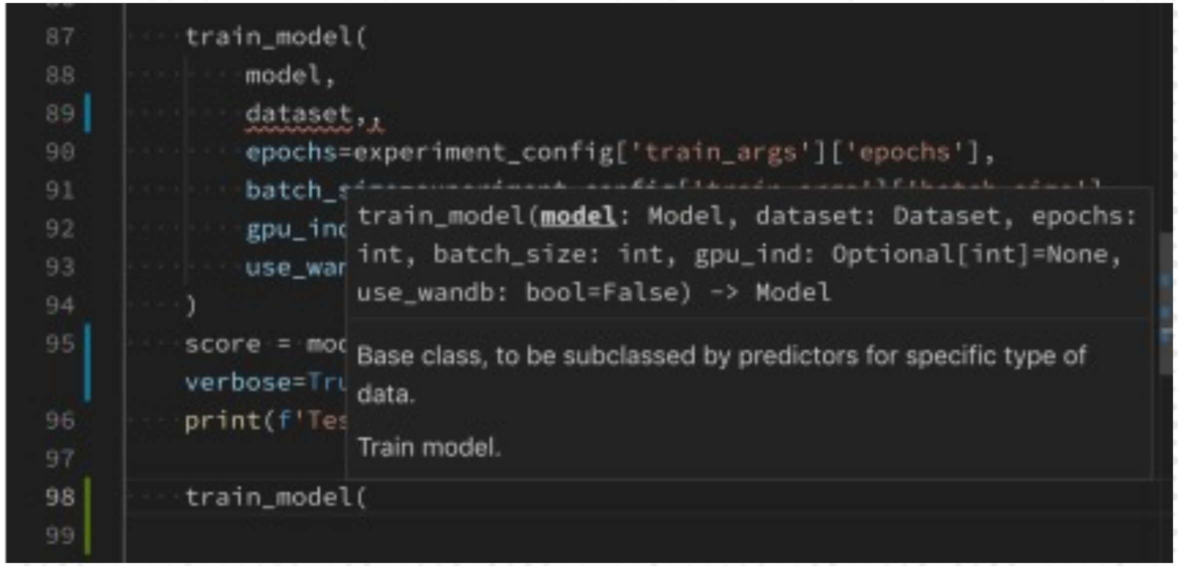
- VS Code makes for a very nice Python experience

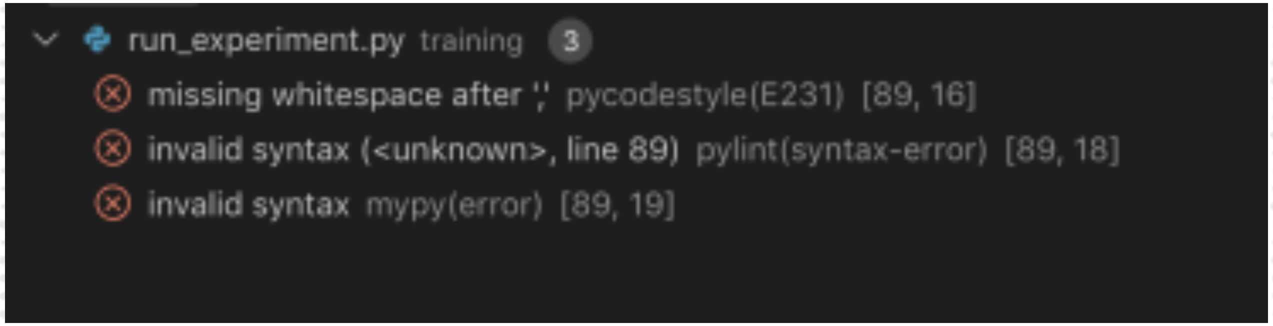
Visual Studio Code



Linters and Type Hints

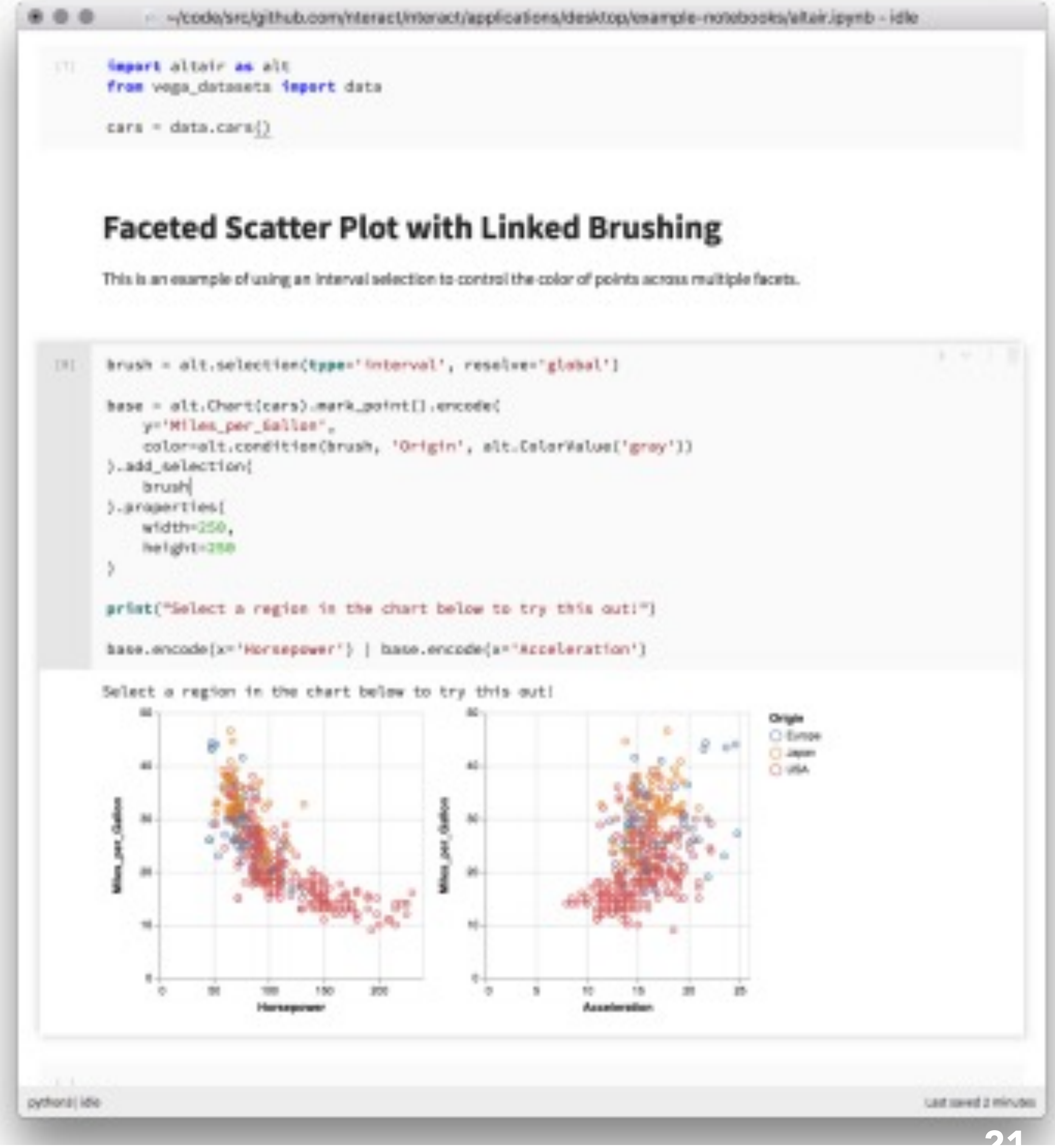
- Whatever code styles rules can be codified, should be
- Static analysis can catch some bugs
- Static type checking both documents code and catches bugs





Jupyter Notebooks

- Notebooks have become fundamental to data science
  - Great as the "first draft" of a project
  - Jeremy Howard from fast.ai good to learn from (course.fast.ai videos)
- Difficult to make scalable, reproducible, well-tested

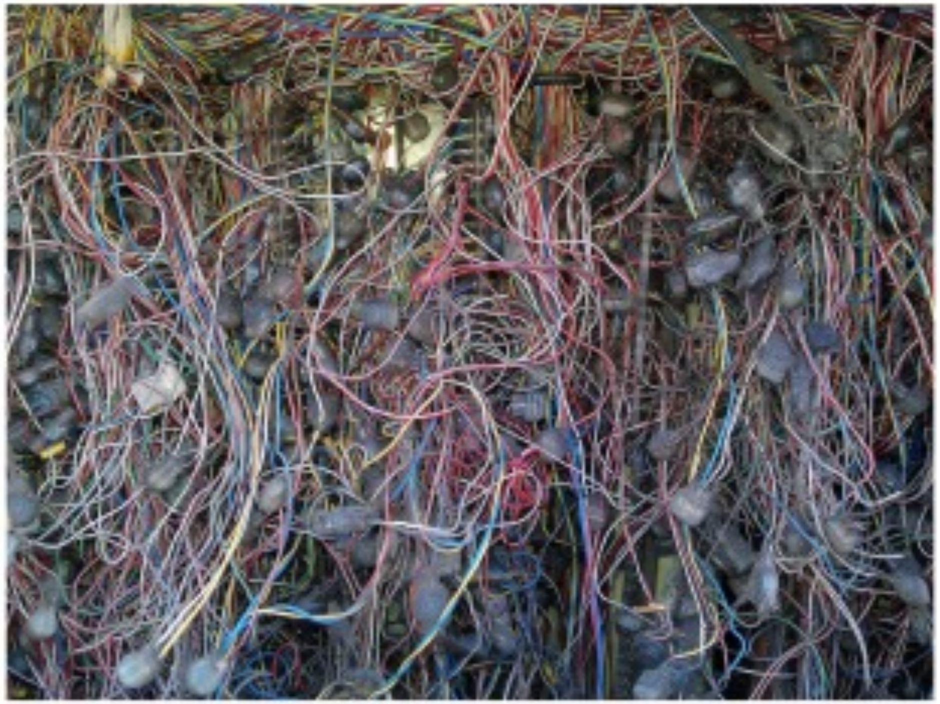


Problem with notebooks

- Hard to version
- Notebook "IDE" is primitive
- Very hard to test
- Out-of-order execution artifacts
- Hard to run long or distributed tasks

5 reasons why jupyter notebooks suck

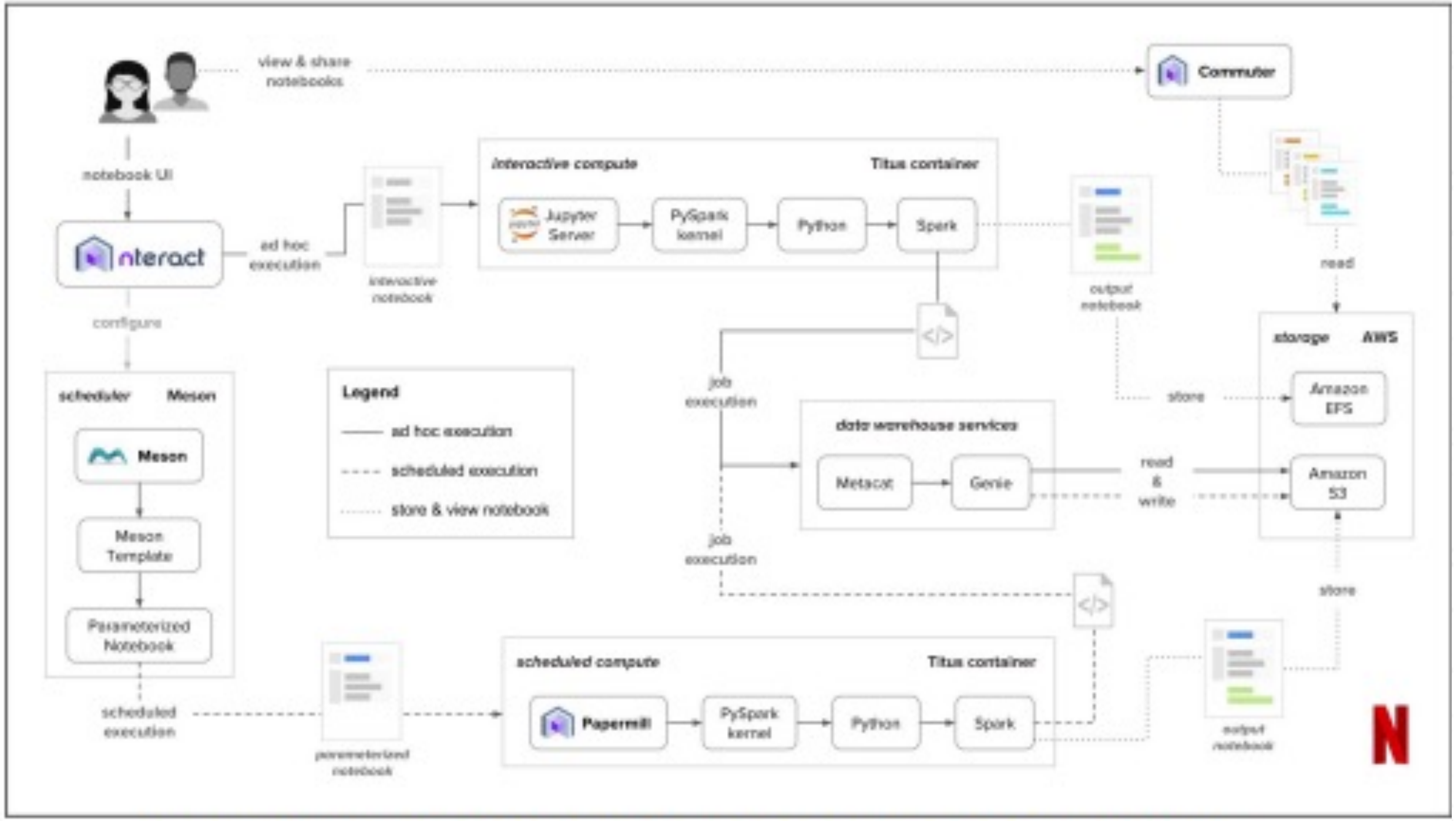
Alexander Mueller Follow
Mar 24, 2018 · 3 min read ★



How it feels like managing jupyter notebooks (Complexity © https://www.flickr.com/photos/bitterjug/7670055210)

Jupyter Notebooks

- Counter-points:
  - Netflix bases all ML workflows on them



https://medium.com/netflix-techblog/notebook-innovation-591ee3221233

NBDev

- Counter-points:
  - Jeremy Howard from fast.ai uses them for everything, with nbdev

Card

> API details.

```
#hide
from nbdev.showdoc import *
```

```
#export
from __future__ import print_function, division
import random

class Card:
    """Represents a standard playing card.

    Attributes:
      suit: integer 0-3
      rank: integer 1-13
    """
    suit_names = ["Clubs", "Diamonds", "Hearts", "Spades"]
    rank_names = [None, "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"]

    def __init__(self, suit=0, rank=2):
        self.suit, self.rank = suit, rank

    def __str__(self):
        """Returns a human-readable string representation."""
        return '%s of %s' % (Card.rank_names[self.rank], Card.suit_names[self.suit])

    def __eq__(self, other) -> bool:
        """Checks whether self and other have the same rank and suit."""
        return self.suit == other.suit and self.rank == other.rank

    def __lt__(self, other) -> bool:
        """Compares this card to other, first by suit, then rank."""
        t1 = self.suit, self.rank
        t2 = other.suit, other.rank
        return t1 < t2

    def __repr__(self): return self.__str__()

    def foo(): pass
```

`Card` is a class that represents a single card in a deck of cards. For example:

```
Card(suit=2, rank=11)
```

```
Jack of Hearts
```

```
c = Card(suit=1, rank=3)
assert str(c) == '3 of Diamonds'

c2 = Card(suit=2, rank=11)
assert str(c2) == 'Jack of Hearts'
```

You can do comparisons of cards, too!

```
assert c2 > c
```

https://github.com/fastai/nbdev

Streamlit

- New, but great at fulfilling a common ML need: interactive applets
- Decorate normal Python code
- Smart data caching, quick re-rendering
- In the works: sharing as easy as pushing a web app to Heroku

https://streamlit.io

Setting up environment

Conda + Pip-Tools Sample Project

Quick demo of setting up a deep learning Python environment.

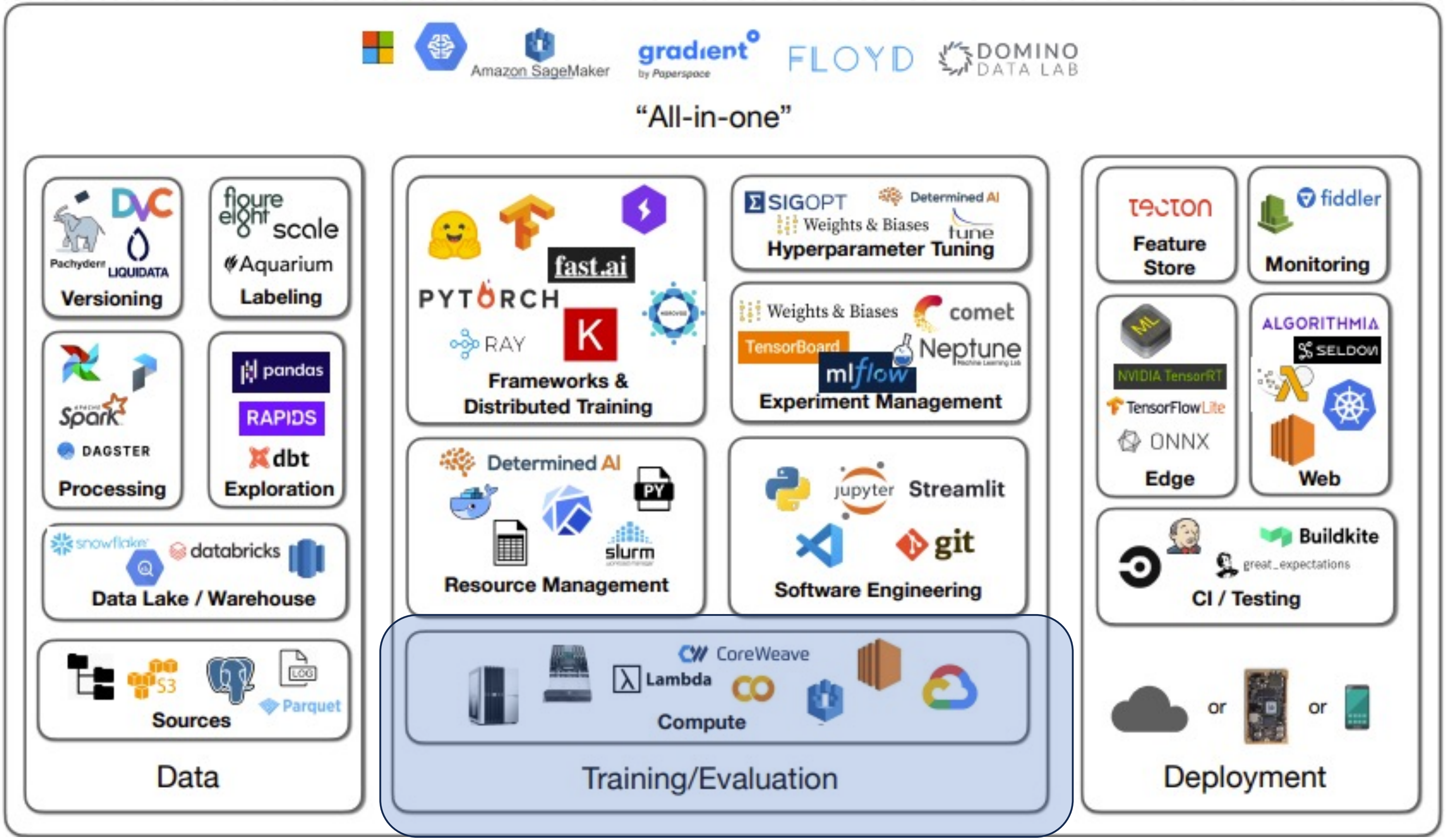
Our goals:

- Easily specify the exact Python, CUDA, CUDNN environment
- Humans should specify minimal constraints (`torch >= 1.7` and `numpy`), computer should figure out exact, mutually compatible versions (`torch==1.7.1; numpy==1.19.5`)
- Separate production (`torch`) from development (`black`) dependencies

We achieve this by:

- We specify our Python and CUDA versions in `environment.yml`
- We use the `conda` package manager to create our environment from this file
- We specify our requirements in `requirements/prod.in` and `requirements/dev.in`
- We use `pip-tools` to lock in mutually compatbile versions of all requirements
- We add a `Makefile` so we can simply run `make` to update everything

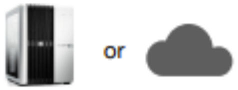
https://github.com/full-stack-deep-learning/conda-piptools


Amazon SageMaker
gradient
by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
DVC
Pachyderm
LIQUIDATA
Versioning
figure eight
scale
Aquarium
Labeling
Spark
DAGSTER
Processing
pandas
RAPIDS
dbt
Exploration
snowflake
databricks
Data Lake / Warehouse
S3
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
K
Frameworks &
Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
mlflow
Experiment Management
Determined AI
PY
slurm
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Compute needs

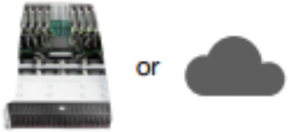
Development

- **Function**
  - Writing code
  - Debugging models
  - Looking at results
- **Desiderata**
  - Quickly compile models and run training
  - Nice-to-have: use GUI
- **Solutions**
  - Desktop with 1-4 GPUs
  - Cloud instance with 1-4 GPUs

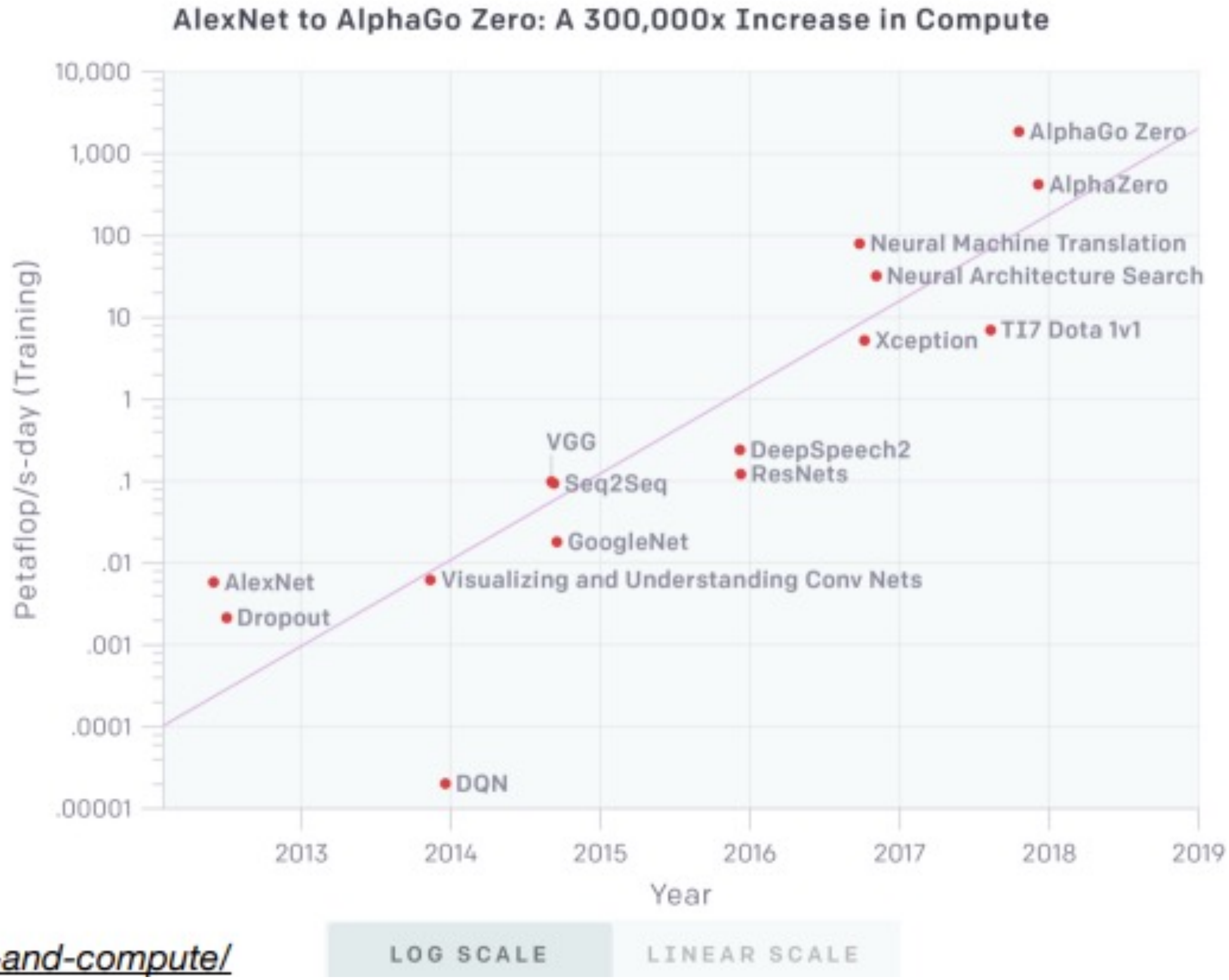


Training/Evaluation

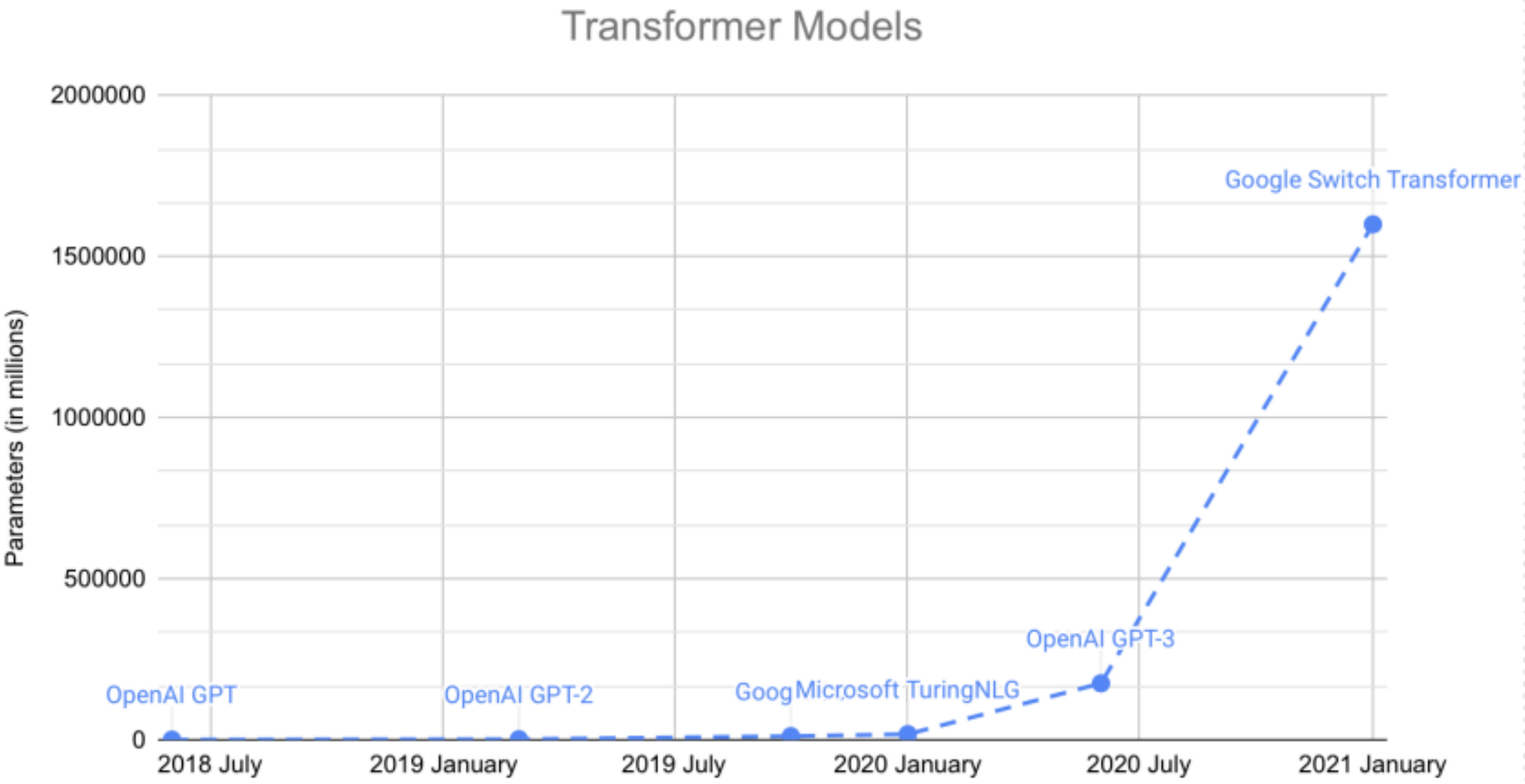
- **Function**
  - Model architecture/hyperparam search
  - Training large models
- **Desiderata**
  - Easy to launch experiments and review results
- **Solutions**
  - Desktop with 4 GPUs
  - Private cluster of GPU machines
  - Cloud cluster of GPU machines

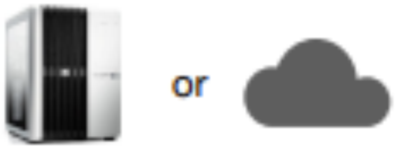


Why compute matters



https://openai.com/blog/ai-and-compute/


Transformer Models
2000000
1500000
1000000
500000
0
Parameters (in millions)
Google Switch Transformer
OpenAI GPT-3
Goog Microsoft TuringNLG
OpenAI GPT
OpenAI GPT-2
2018 July
2019 January
2019 July
2020 January
2020 July
2021 January

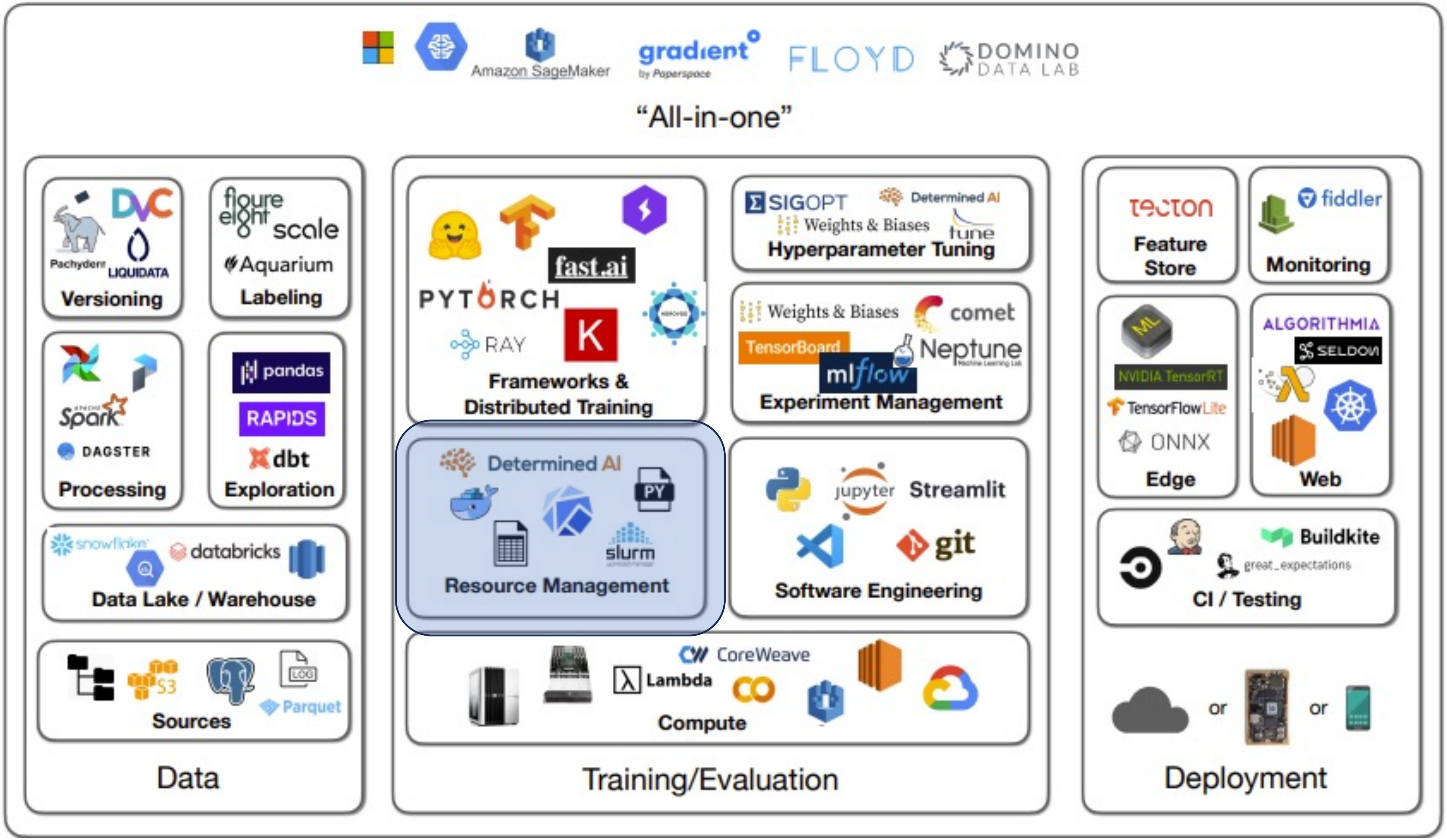
So, 

?

- GPU Basics
- Cloud Options
- On-prem Options
- Analysis and Recommendations

In Practice

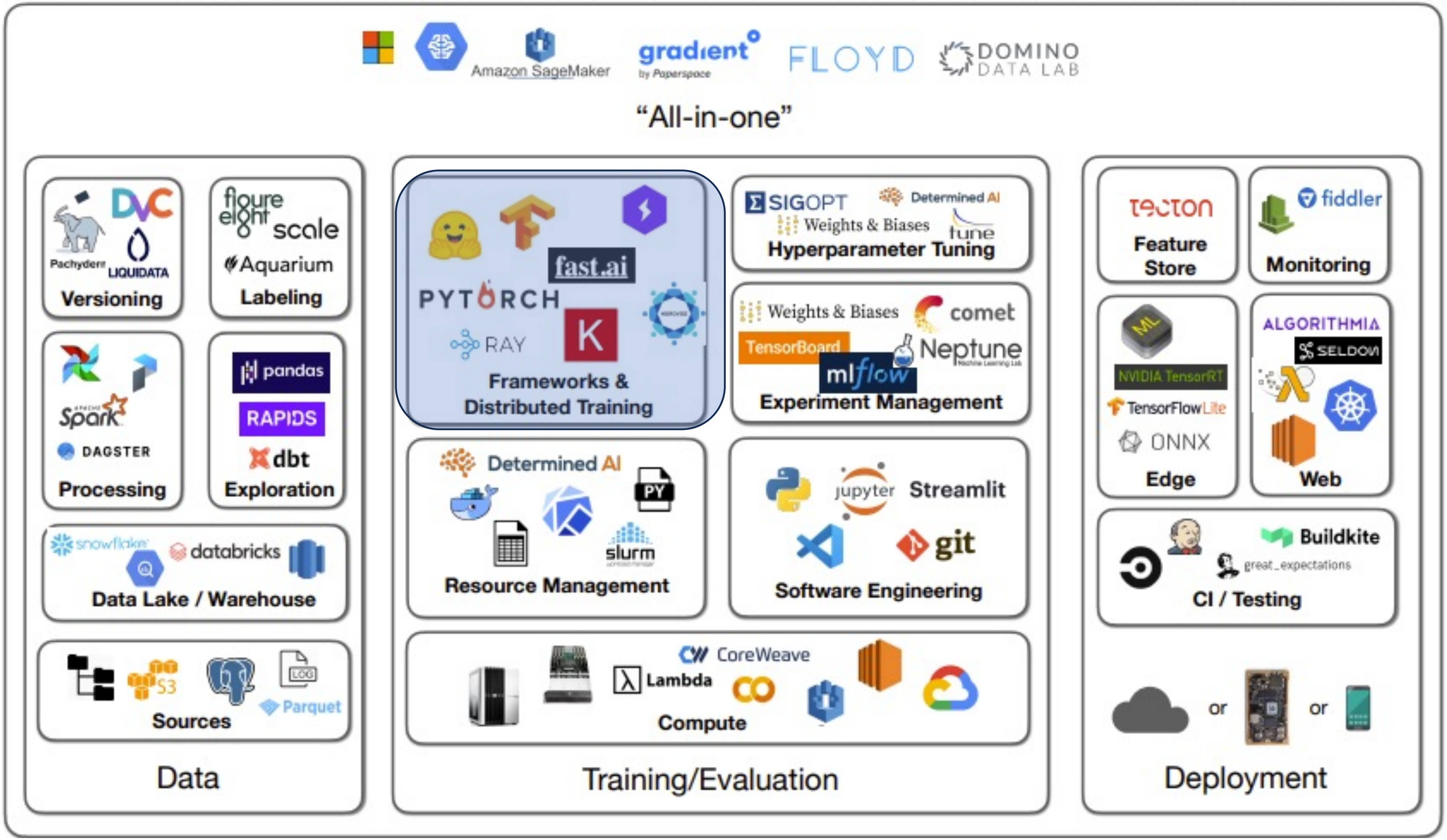
- Even though cloud is expensive, it's hard to make on-prem scale past a certain point
- Dev-ops (declarative infra, repeatable processes) definitely easier in the cloud
- Maintenance is also a big factor



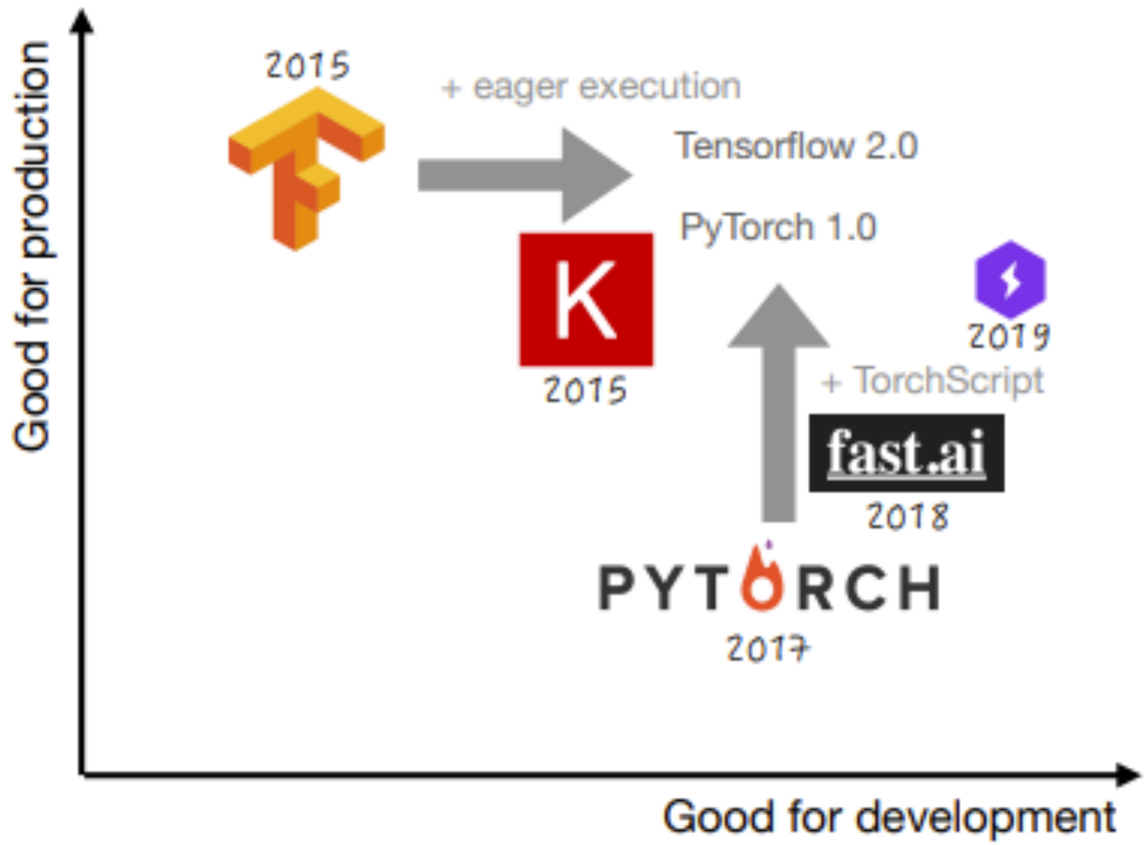
Amazon SageMaker
gradient
by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
Pachyderm
LIQUIDATA
Versioning
figure eight
scale
Aquarium
Labeling
Spark
DAGSTER
Processing
pandas
RAPIDS
dbt
Exploration
snowflake
databricks
Data Lake / Warehouse
S3
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
Frameworks &
Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
mlflow
Experiment Management
Determined AI
PY
slurm
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Resource Management

- Function
  - Multiple people…
  - using multiple GPUs/machines…
  - running different environments
- Goal
  - Easy to launch a batch of experiments with proper dependencies and resource allocations
- Solutions
  - Python scripts
  - SLURM
  - Docker + Kubernetes
  - Software specialized for ML use cases


Amazon SageMaker
gradient by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
DVC
Pachyderm
LIQUIDATA
Versioning
figure eight
scale
Aquarium
Labeling
Spark
DAGSTER
Processing
pandas
RAPIDS
dbt
Exploration
snowflake
databricks
Data Lake / Warehouse
S3
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
K
Frameworks & Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
mlflow
Experiment Management
Determined AI
PY
slurm
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Deep Learning Frameworks

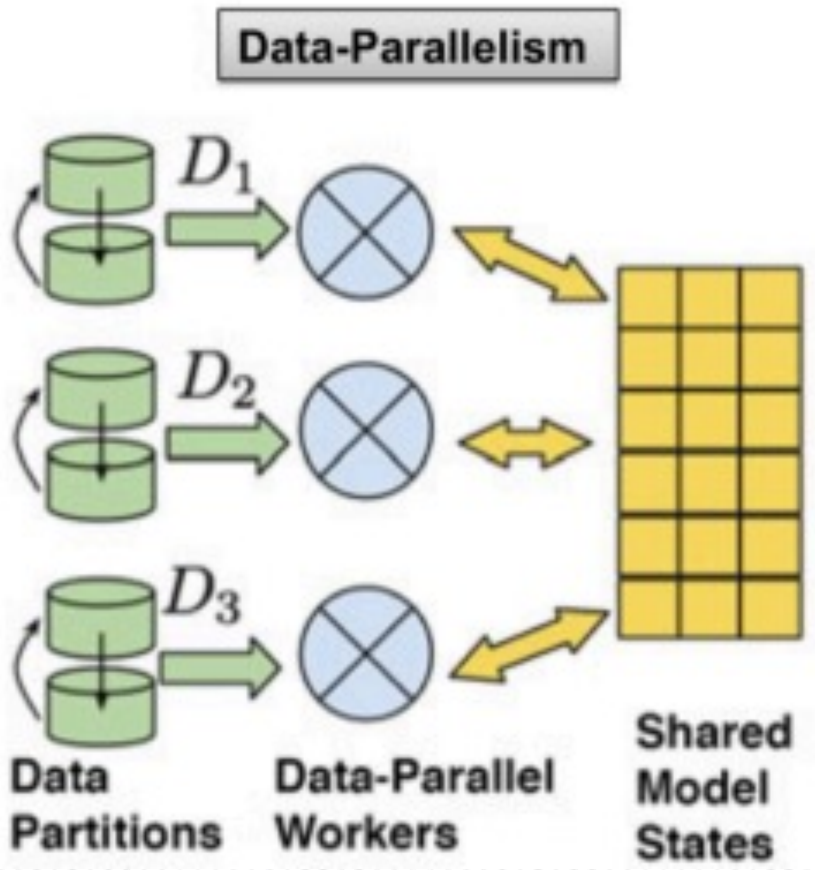


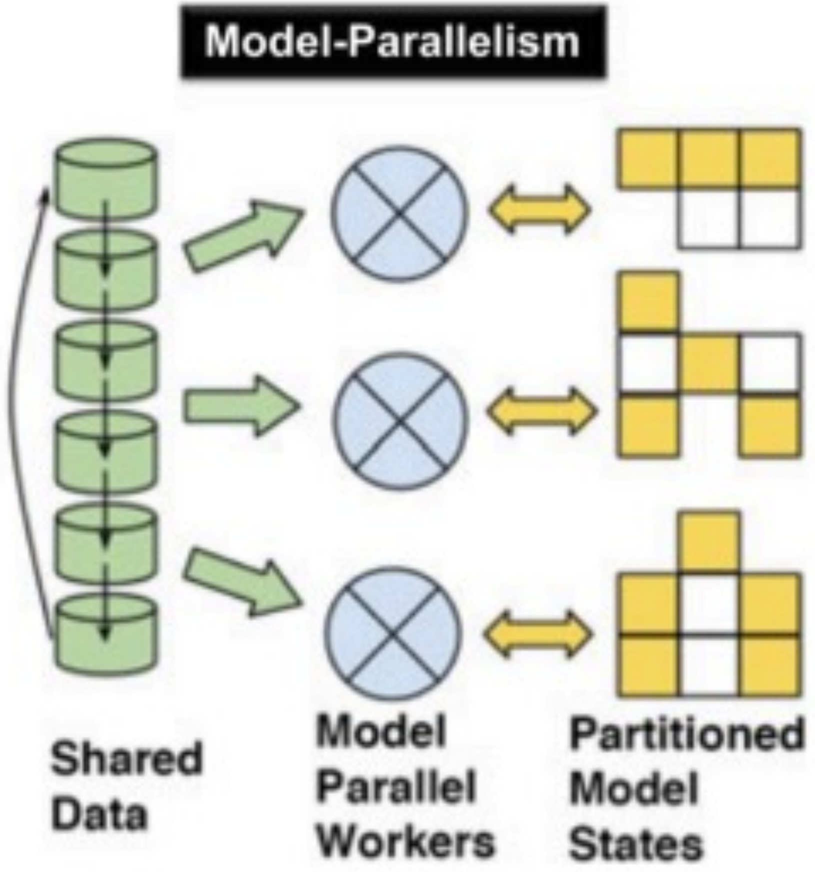
- Unless you have a good reason not to, use Tensorflow/Keras or PyTorch
- Both have converged to the same point:
  - easy development via define-by-run
  - multi-platform optimized execution graph
- Today, most new projects use PyTorch, because of its more Python dev-friendly experience
- fast.ai library builds on PyTorch with best practices
- PyTorch-Lightning adds a powerful training loop

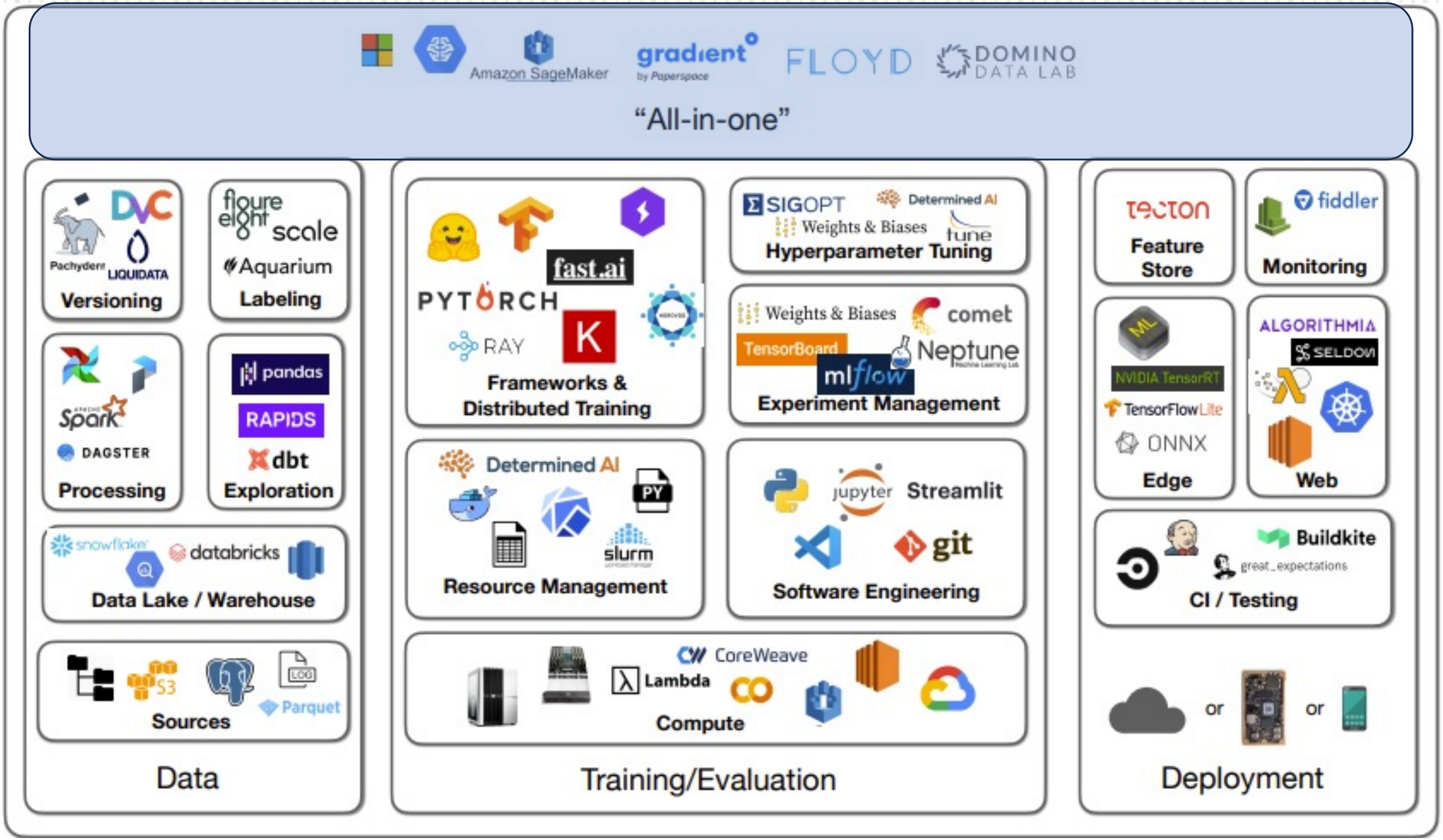
Distributed Training

- Using multiple GPUs and/or machines to train a single model.
- More complex than simply running different experiments on different GPUs
- A must-do on big datasets and large models

Parallelism






Amazon SageMaker
gradient
by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
Pachyderm
DVC
LIQUIDATA
Versioning
figure eight
scale
Aquarium
Labeling
Spark
DAGSTER
Processing
pandas
RAPIDS
dbt
Exploration
snowflake
databricks
Data Lake / Warehouse
S3
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
K
Frameworks &
Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
mlflow
Experiment Management
Determined AI
PY
slurm
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

All-in-one Solutions

- Single system for everything
  - development (hosted notebook)
  - scaling experiments to many machines (sometimes even provisioning)
  - tracking experiments and versioning models
  - deploying models
  - monitoring performance