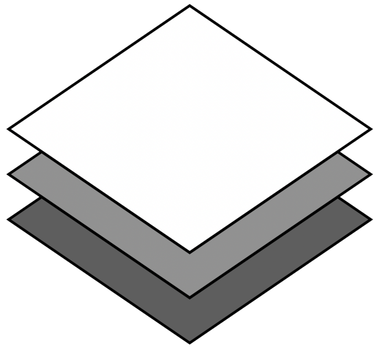
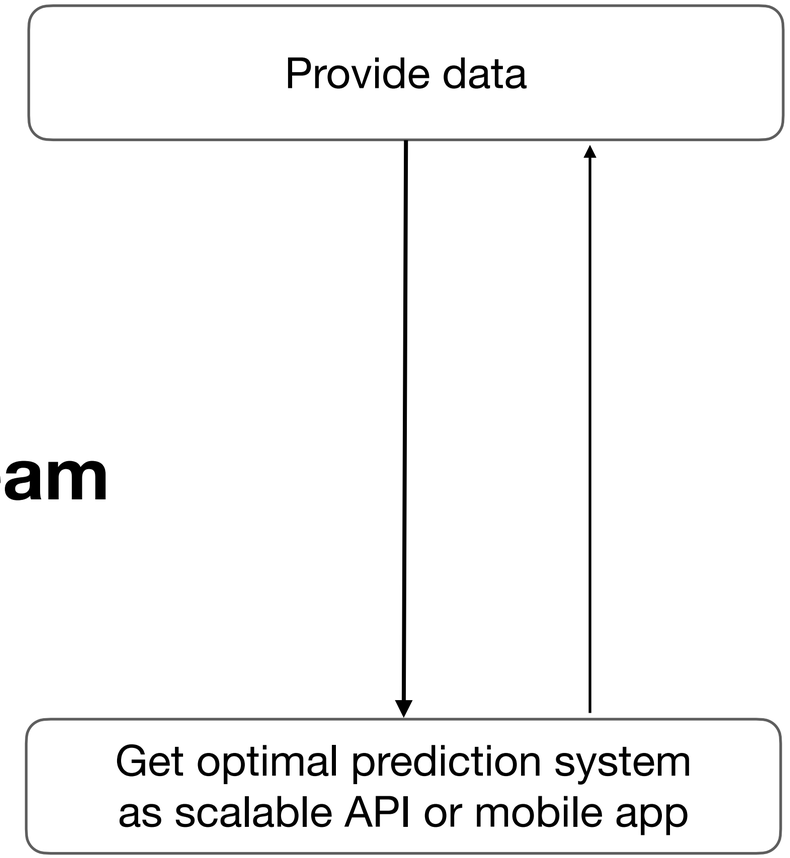
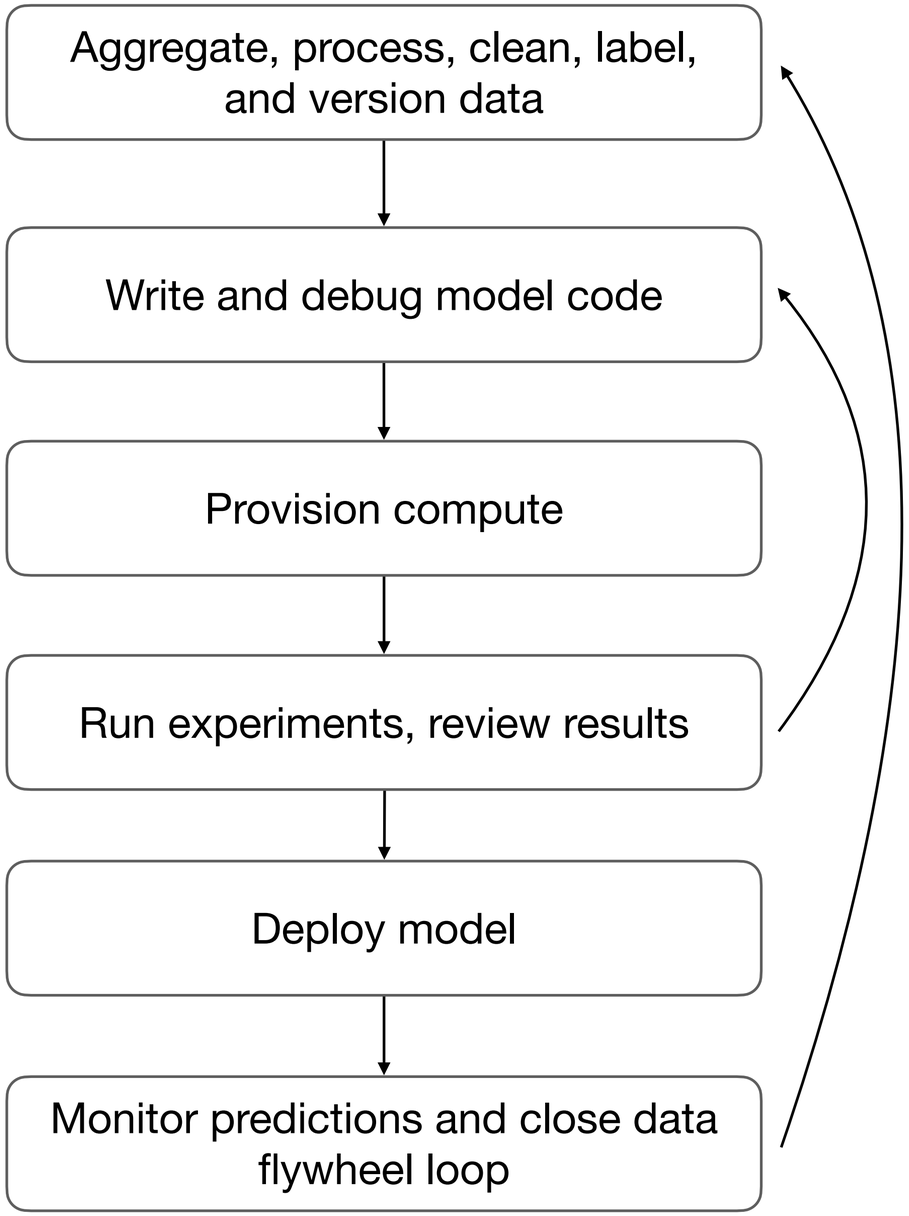


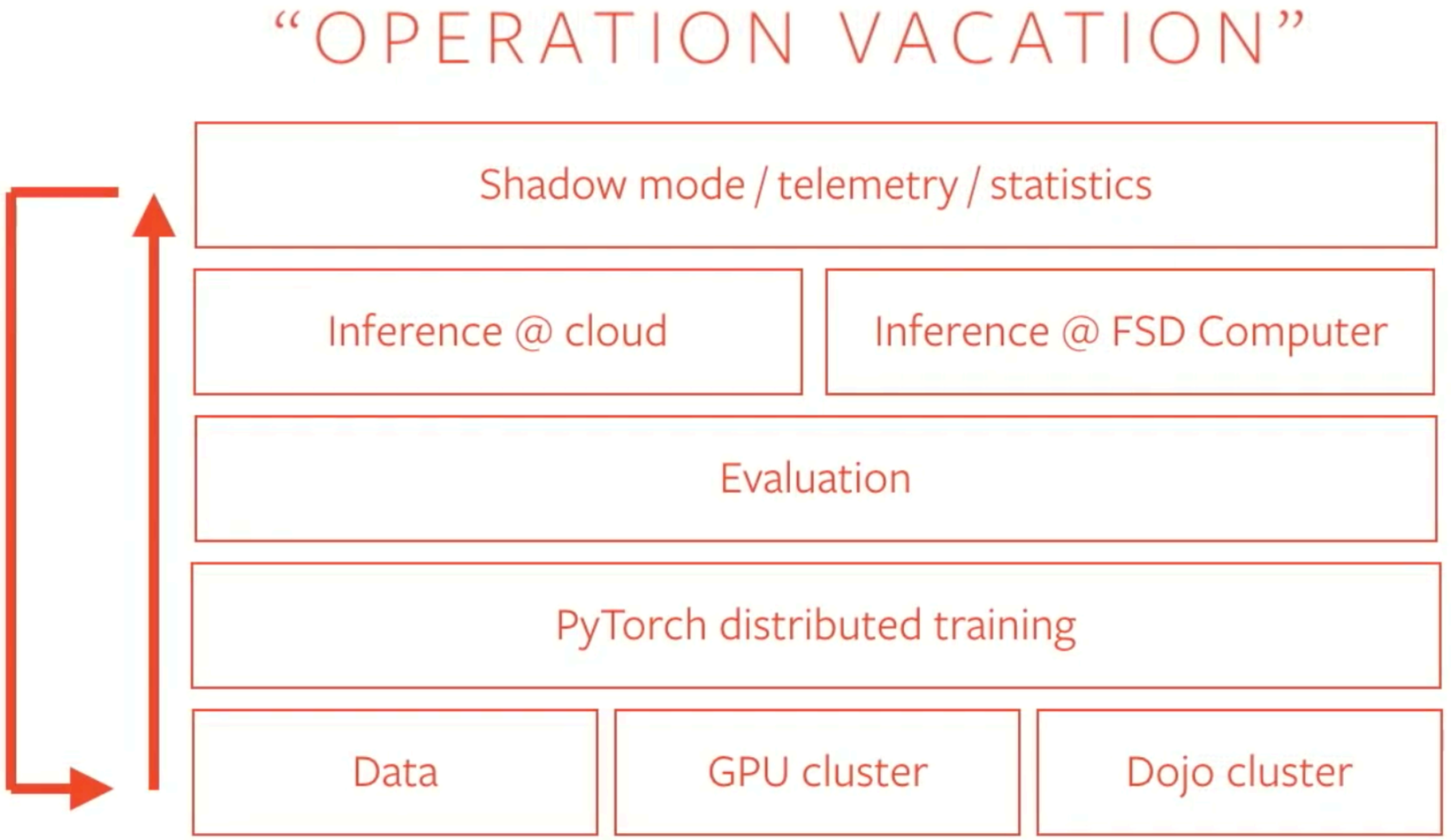
Infrastructure & Tooling

Dream



Reality





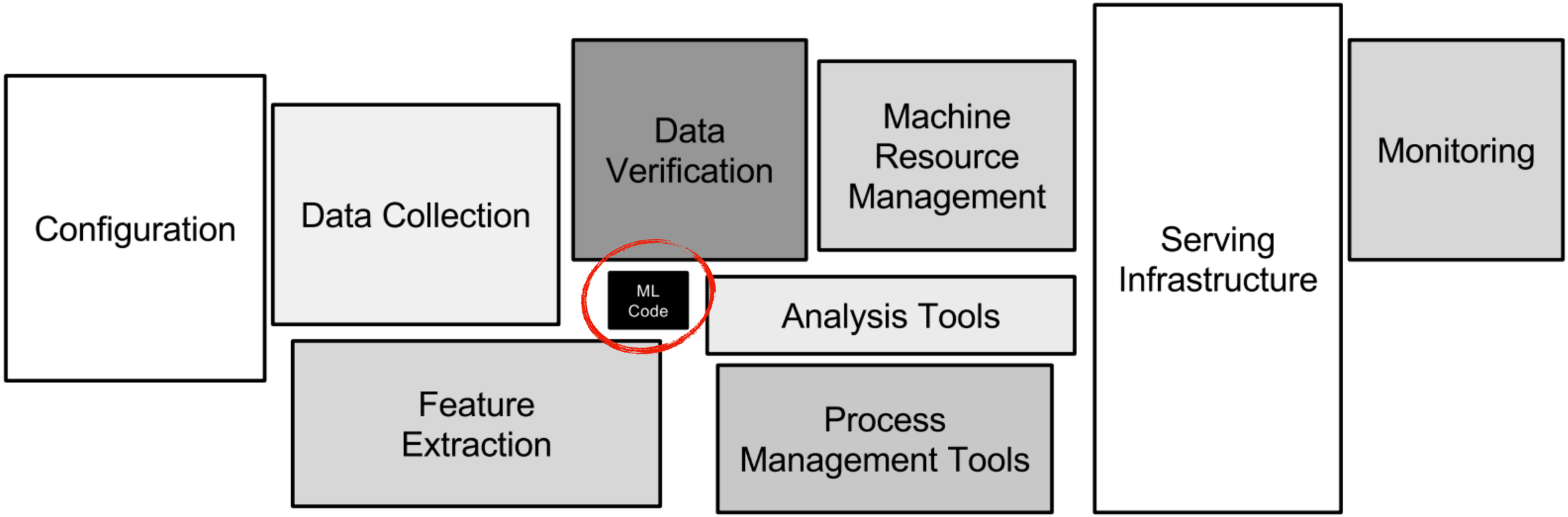
Goal: add data, see model improve

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
{dsculley,gholt,dgg,edavydov}@google.com
{toddphillips,ebner,vchaudhary,mwyoung}@google.com
Google, Inc

SE4ML: Software Engineering for Machine Learning (NIPS 2014



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

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

"All-in-one"

Data

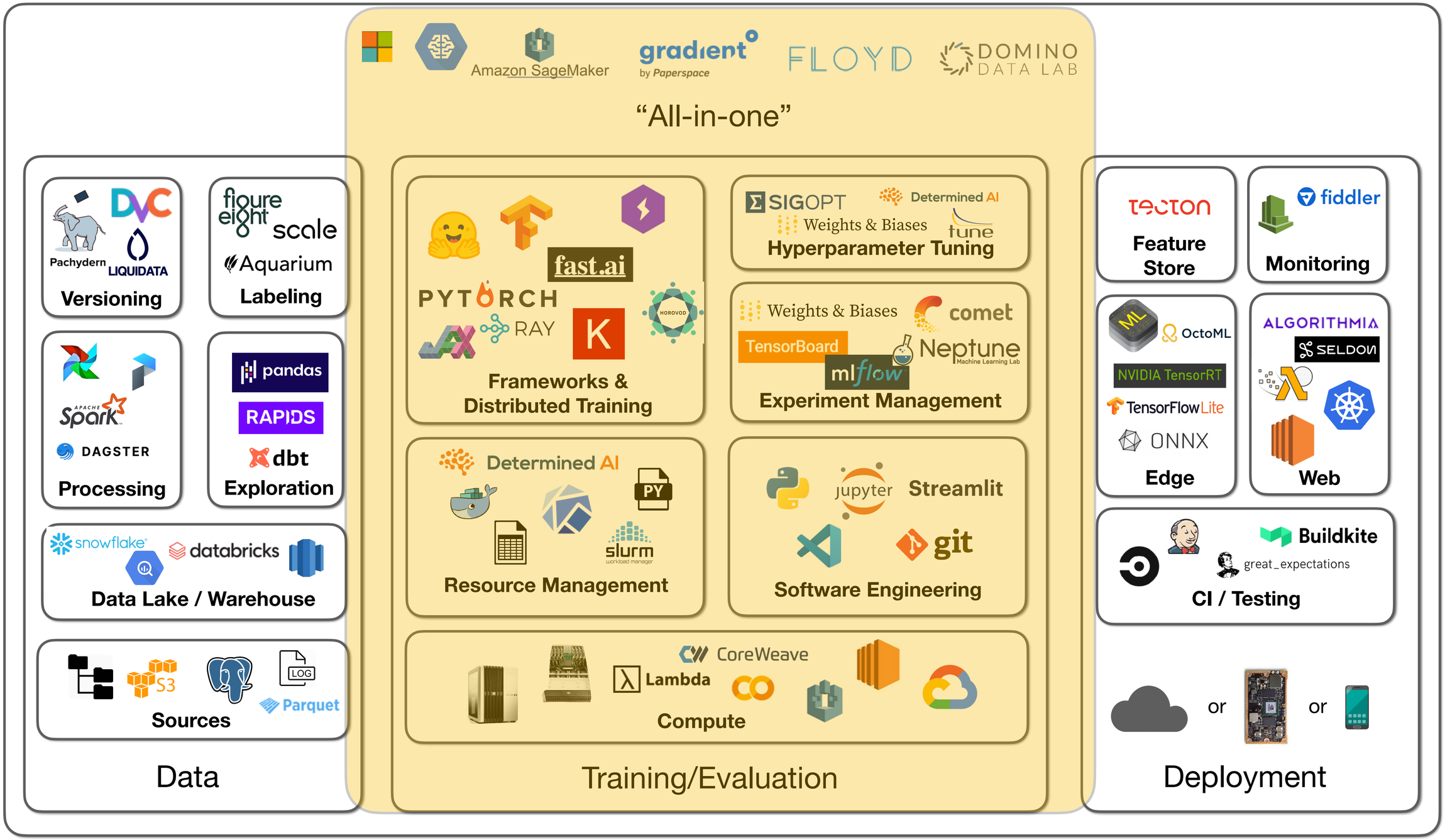
- Versioning: DVC, Pachydern, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, Parquet

Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- 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: NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations


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
LOG
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
HOROVOD
Frameworks & Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
Machine Learning Lab
mlflow
Experiment Management
Determined AI
PY
slurm
workload manager
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
ML
OctoML
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Questions?

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

"All-in-one"

Data

- Versioning: DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Apache Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, LOG, Parquet

Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- Hyperparameter Tuning: SIGOPT, Determined AI, Weights & Biases, tune
- Experiment Management: Weights & Biases, comet, TensorBoard, Neptune Machine Learning Lab, mlflow
- Resource Management: Determined AI, PY, slurm workload manager
- Software Engineering: jupyter, Streamlit, git
- Compute: CoreWeave, Lambda

Deployment

- Feature Store: tecton
- Monitoring: fiddler
- Edge: ML, OctoML, NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations

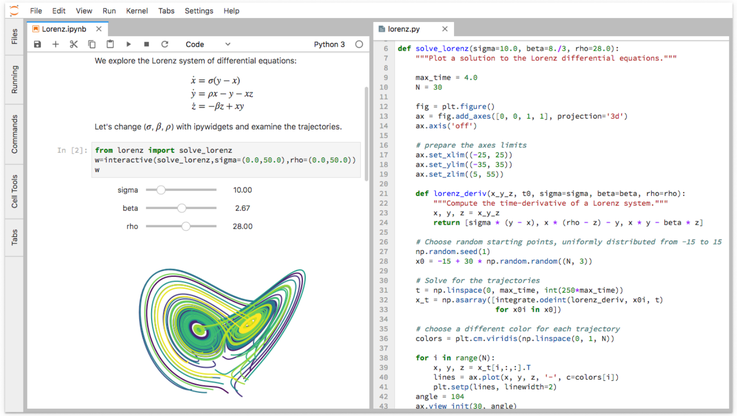
or or

Programming Language

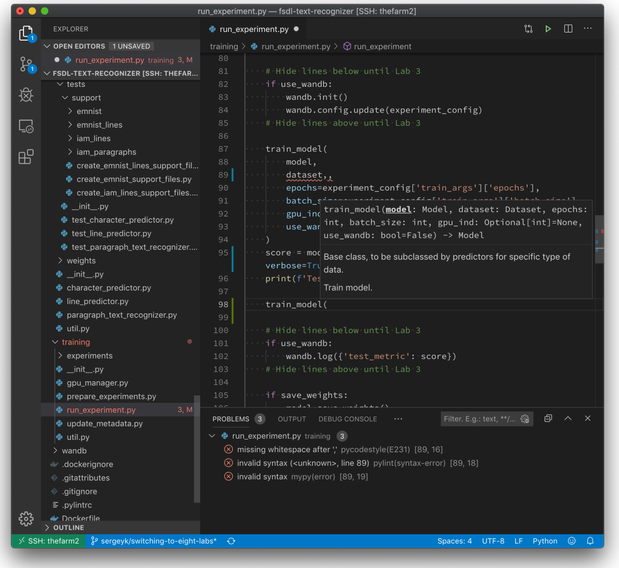
- Python, because of the libraries
  - Clear winner in scientific and data computing

Editors

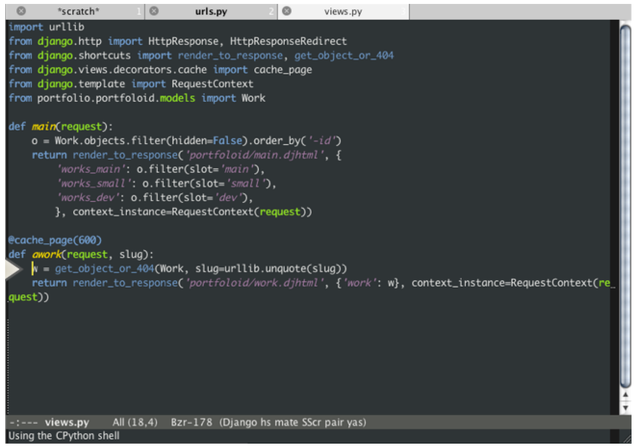
Vim



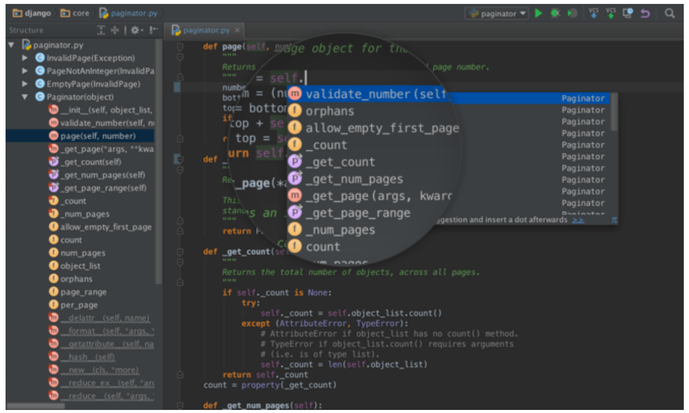
Jupyter



VS Code



Emacs



PyCharm

Visual Studio Code

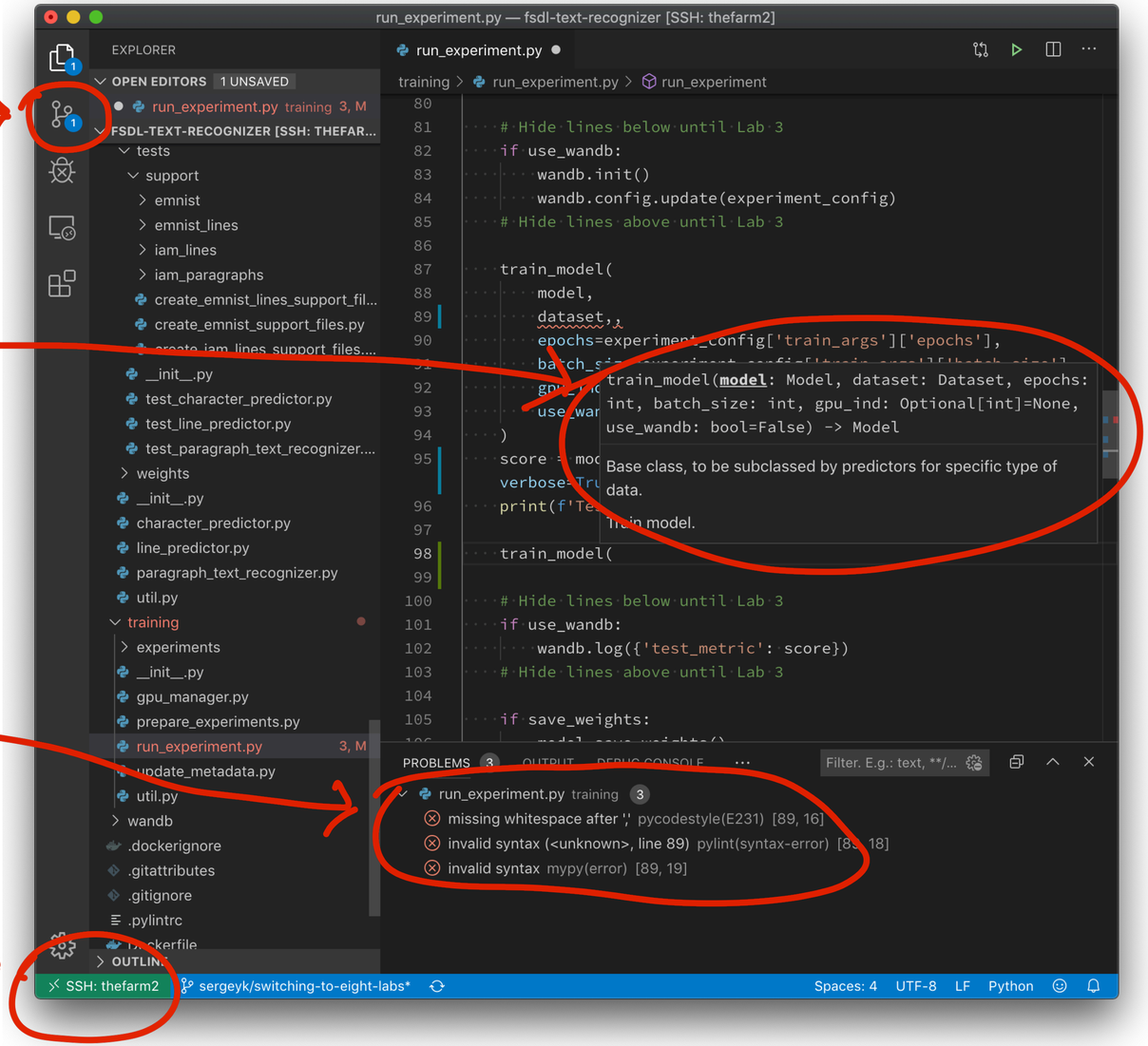
Built-in git staging and diffing

Peek documentation

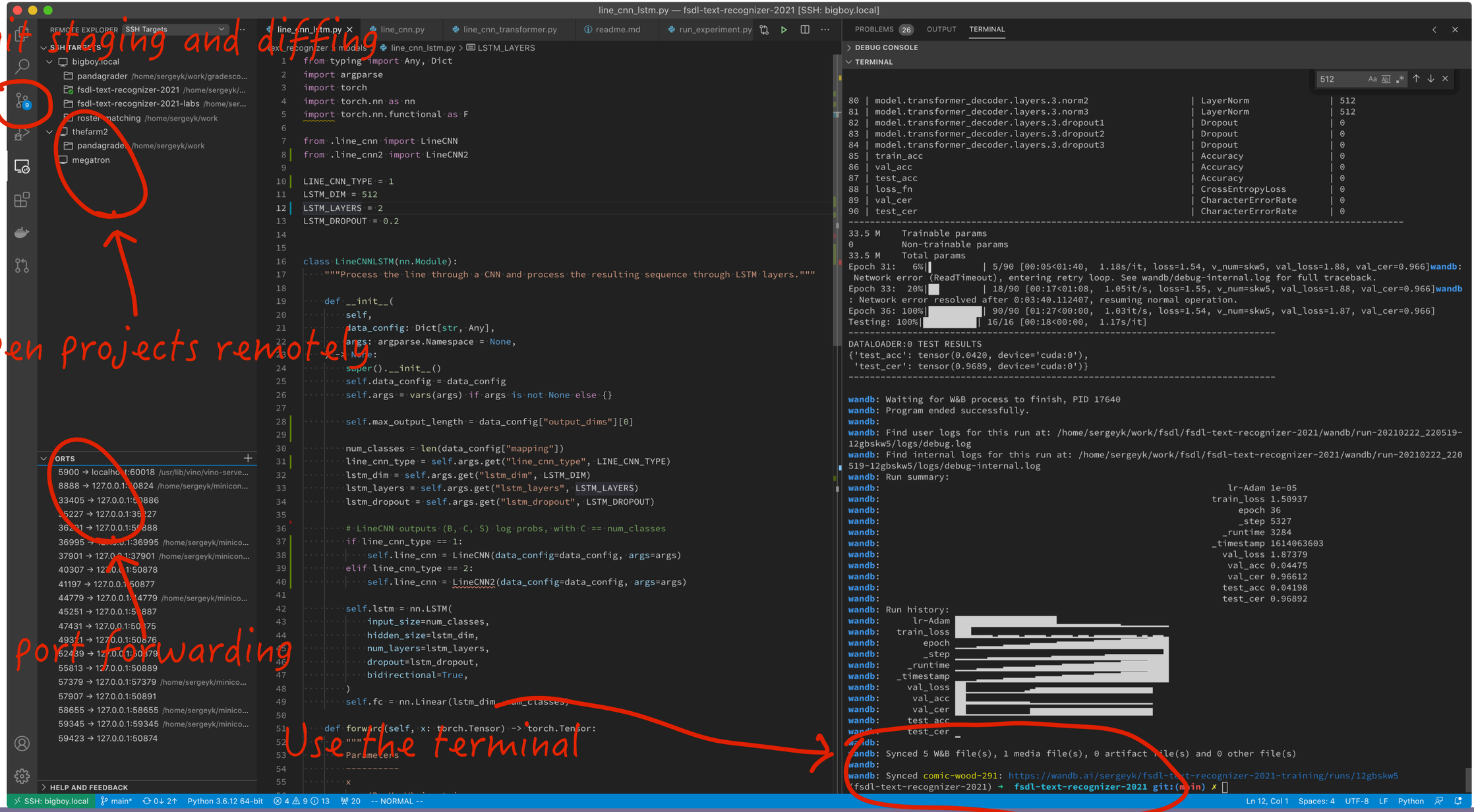
- VS Code makes for a very nice Python experience

Lint code as your write

Open whole projects remotely

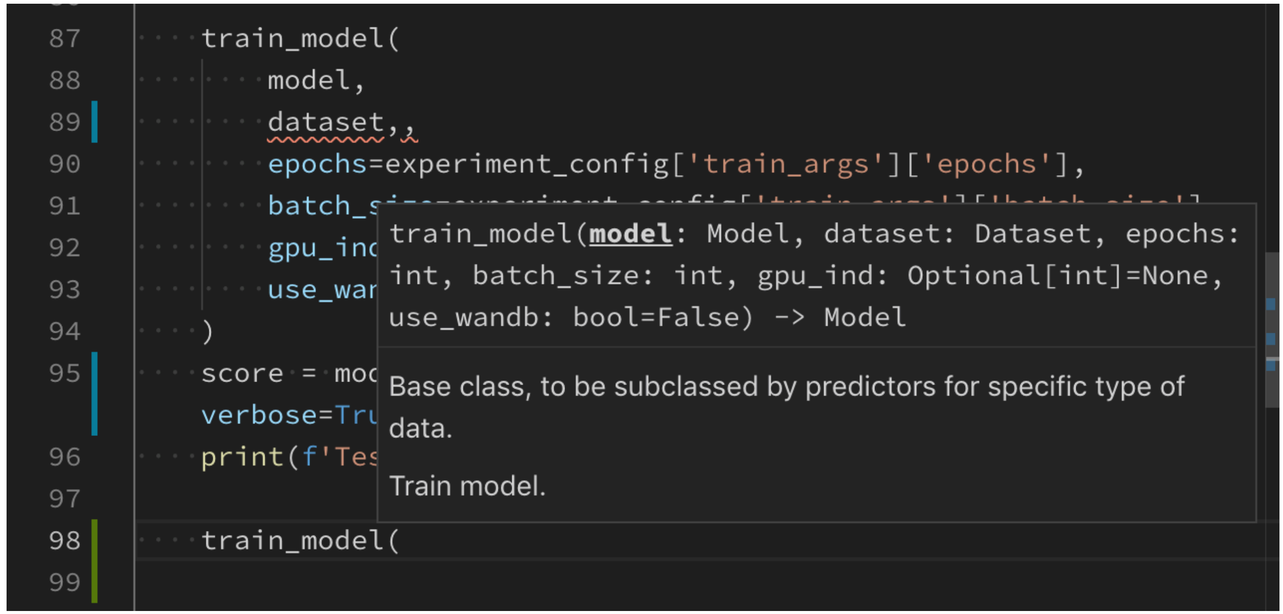


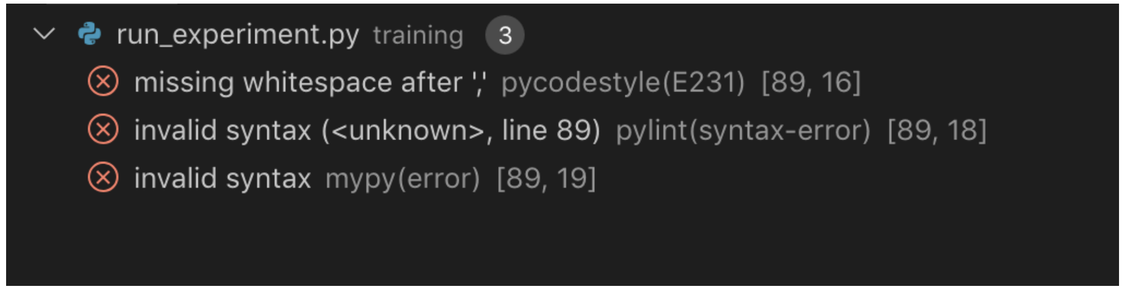
Visual Studio Code



Linters and Type Hints

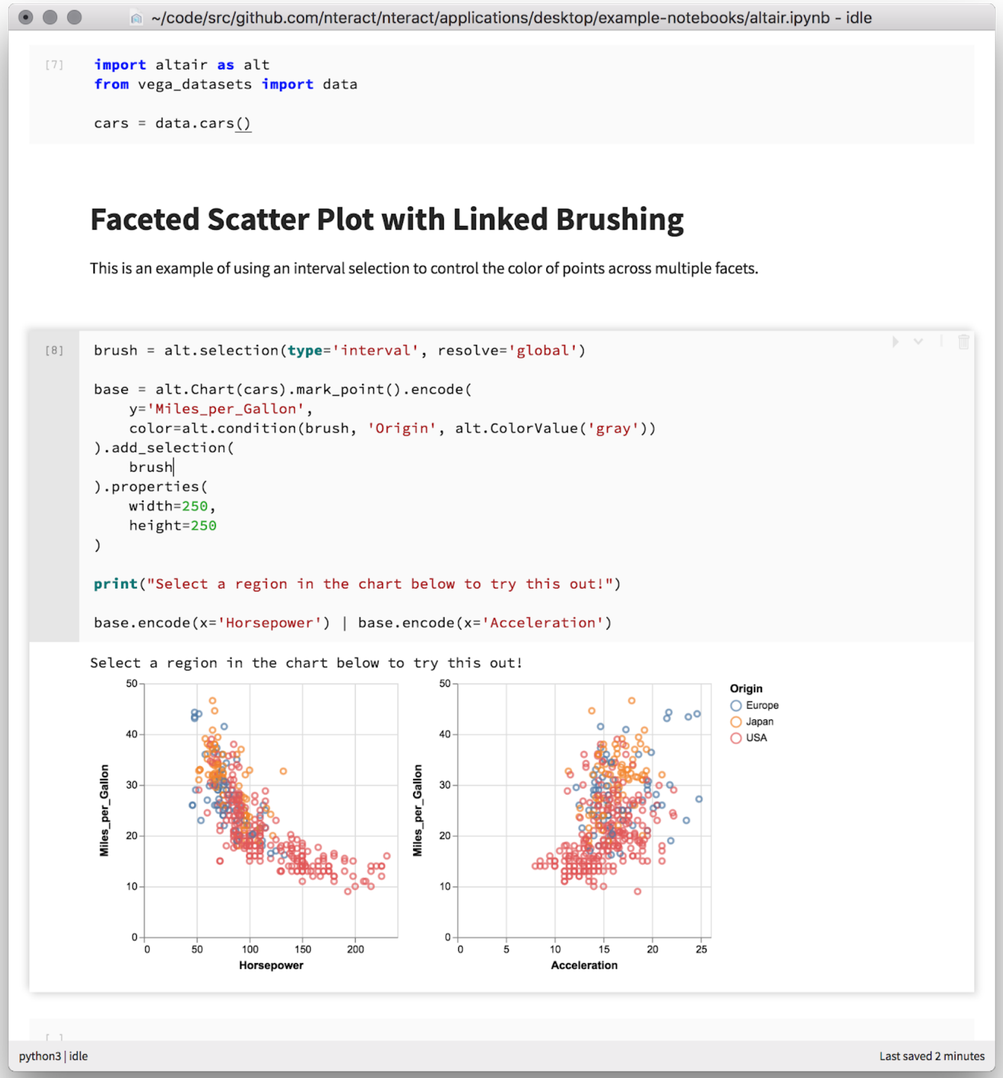
- Whatever code style rules can be codified, should be
- Static analysis can catch some bugs
- Static type checking both documents code and catches bugs
- Will see in Lab 7





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



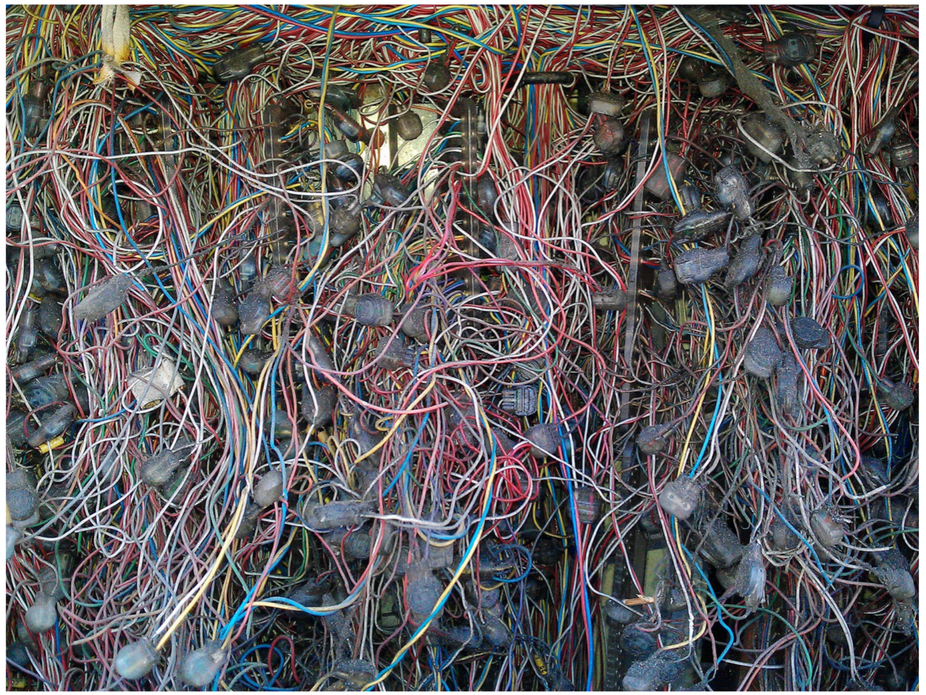
Problems with notebooks

- 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 ★

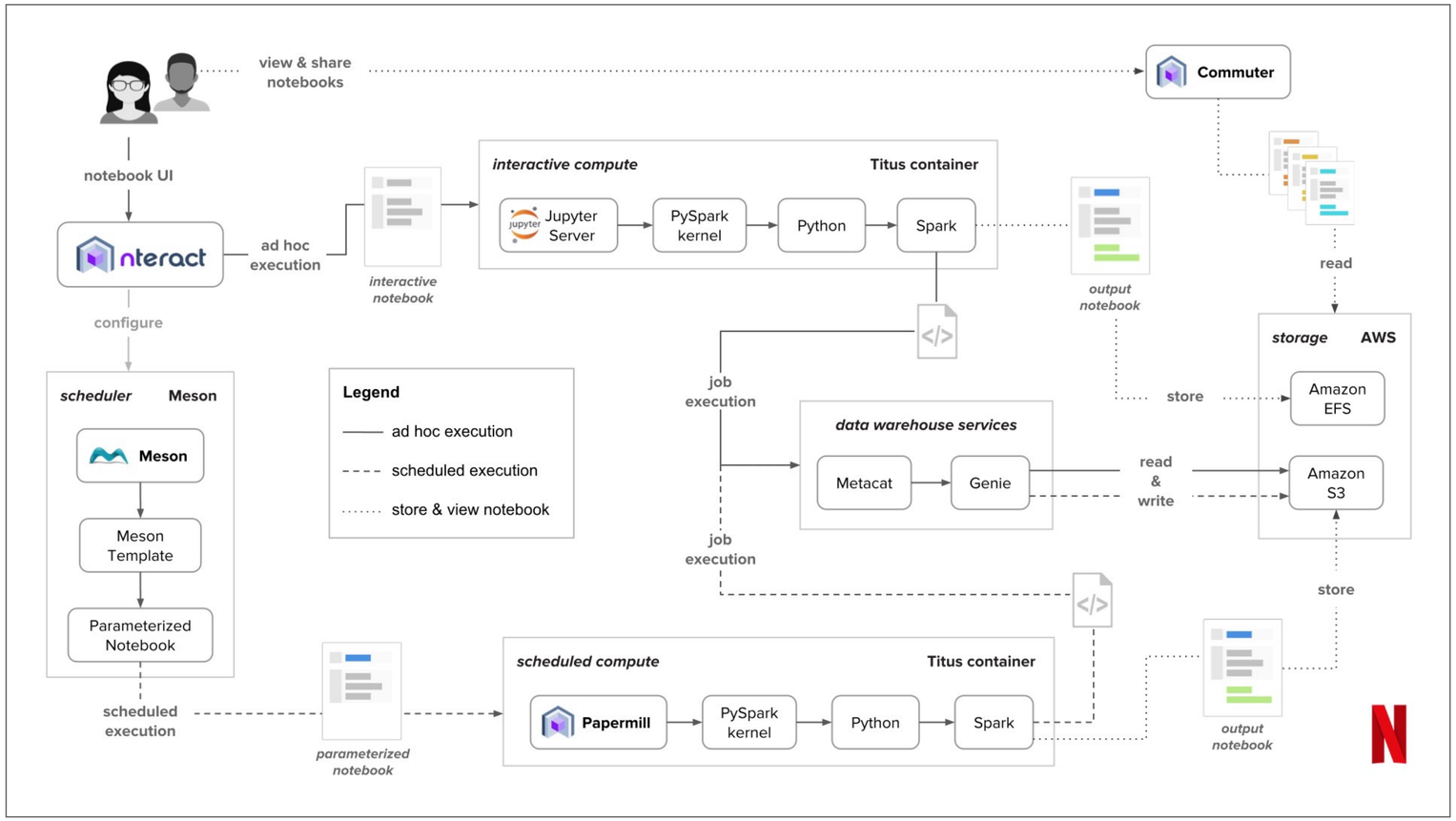


How it feels like managing jupyter notebooks (Complexity © https://www.flickr.com/photos/bitterjug/7670055210)

https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086

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!

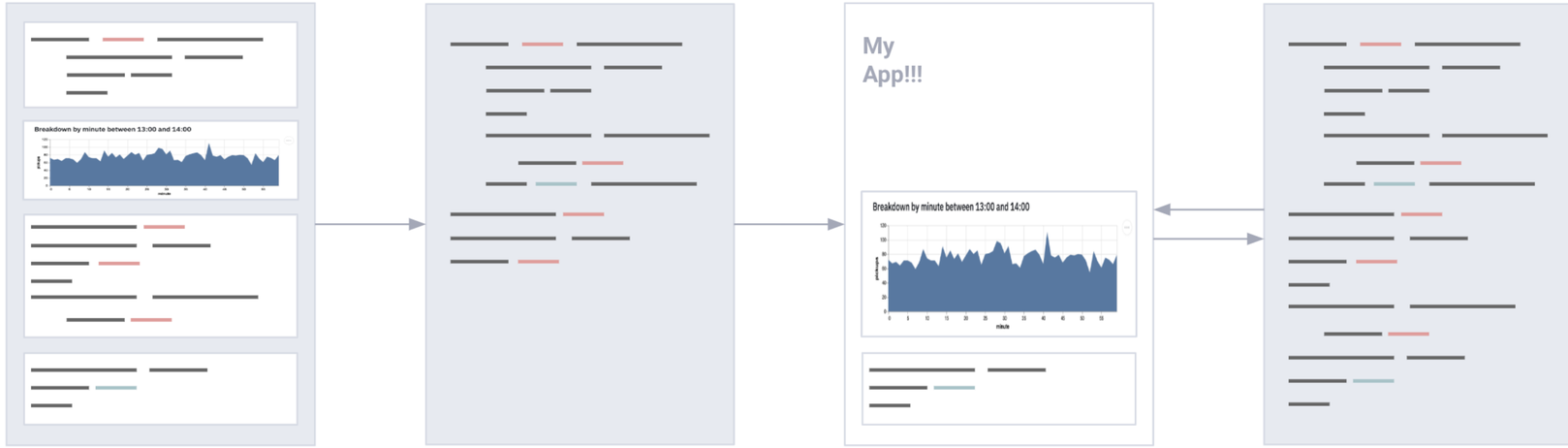
```
assert c2 > c
```

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

The unmaintainability trap



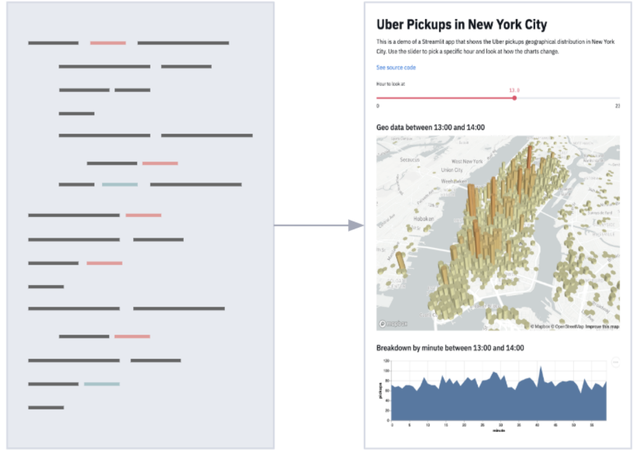
Step 1: Explore in a Jupyter notebook.

Step 2: Copy-paste into a Python script.

Step 3: Write Flask app. Think about HTTP requests, HTML, callbacks, JS...

Step 4: Uh oh. Need more features.😬

The Streamlit workflow



Step 1: Sprinkle in a few API calls into your existing Python script.

Step 2: Show off your beautiful, performant tool 🎉 .

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

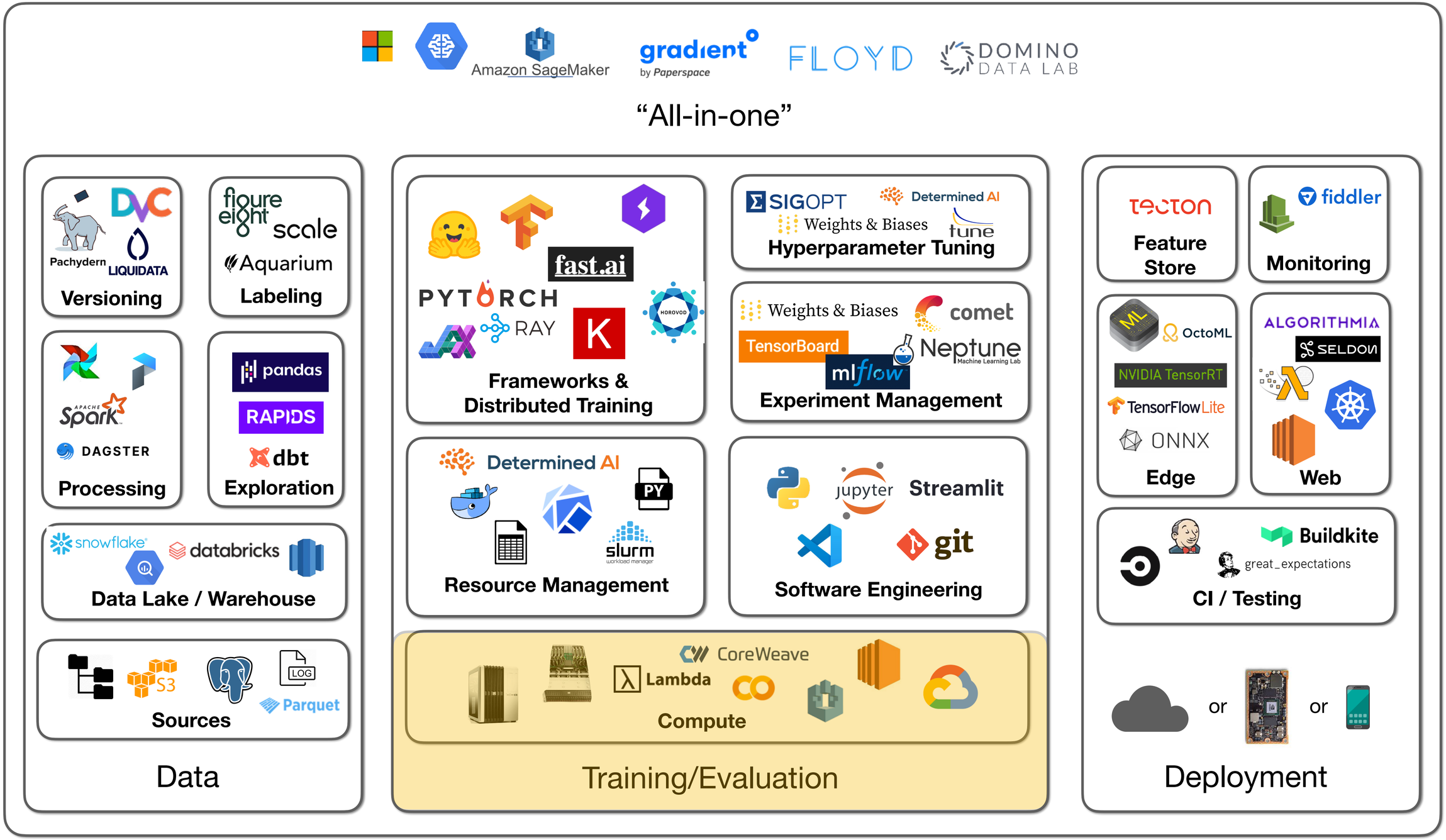
We achieve this by:

- 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

(How we do it in lab)

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

Questions?


Amazon SageMaker
gradient by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
Pachydern
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
HOROVOD
Frameworks & Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
Machine Learning Lab
mlflow
Experiment Management
Determined AI
PY
slurm
workload manager
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
OctoML
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

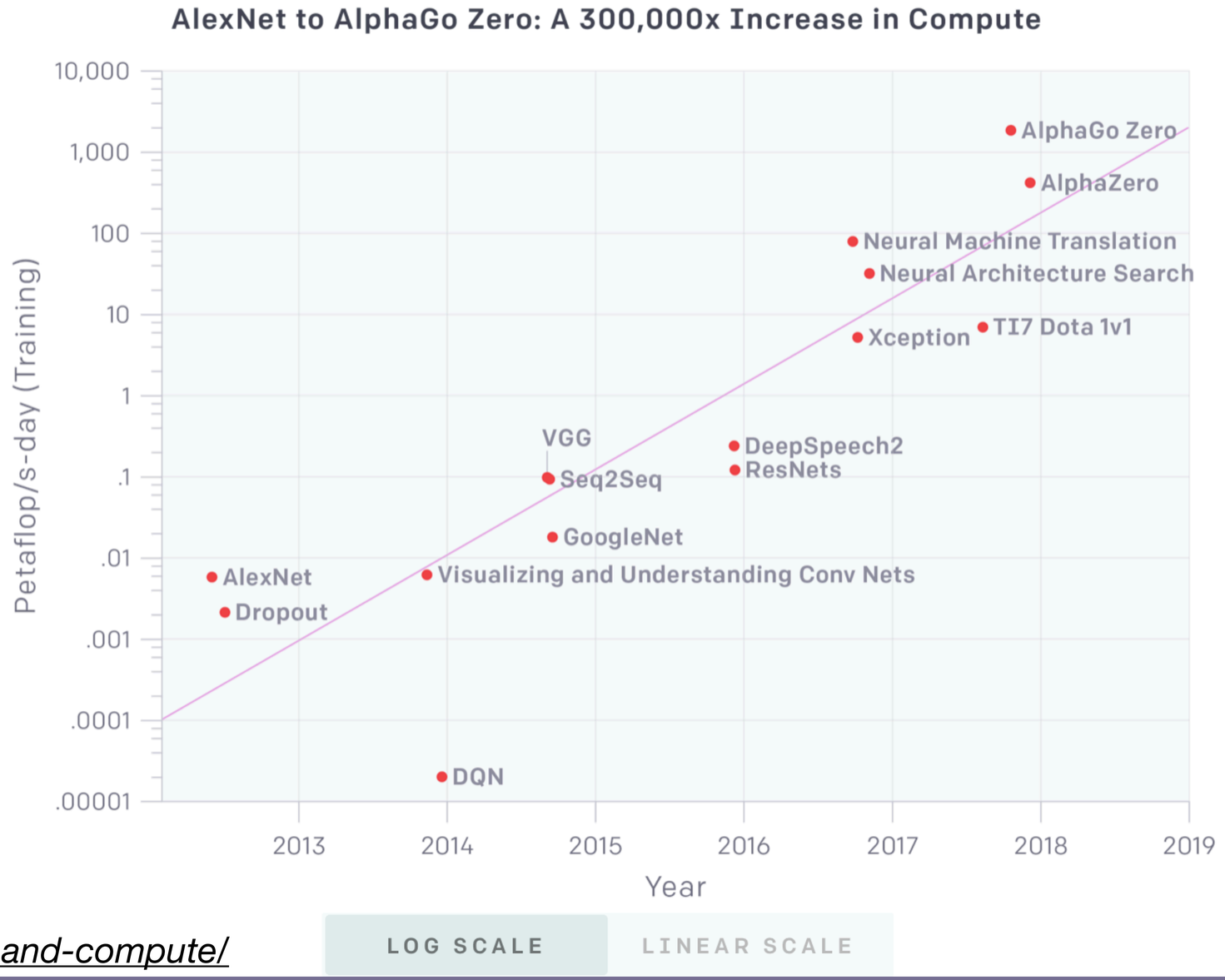

or


Training/Evaluation

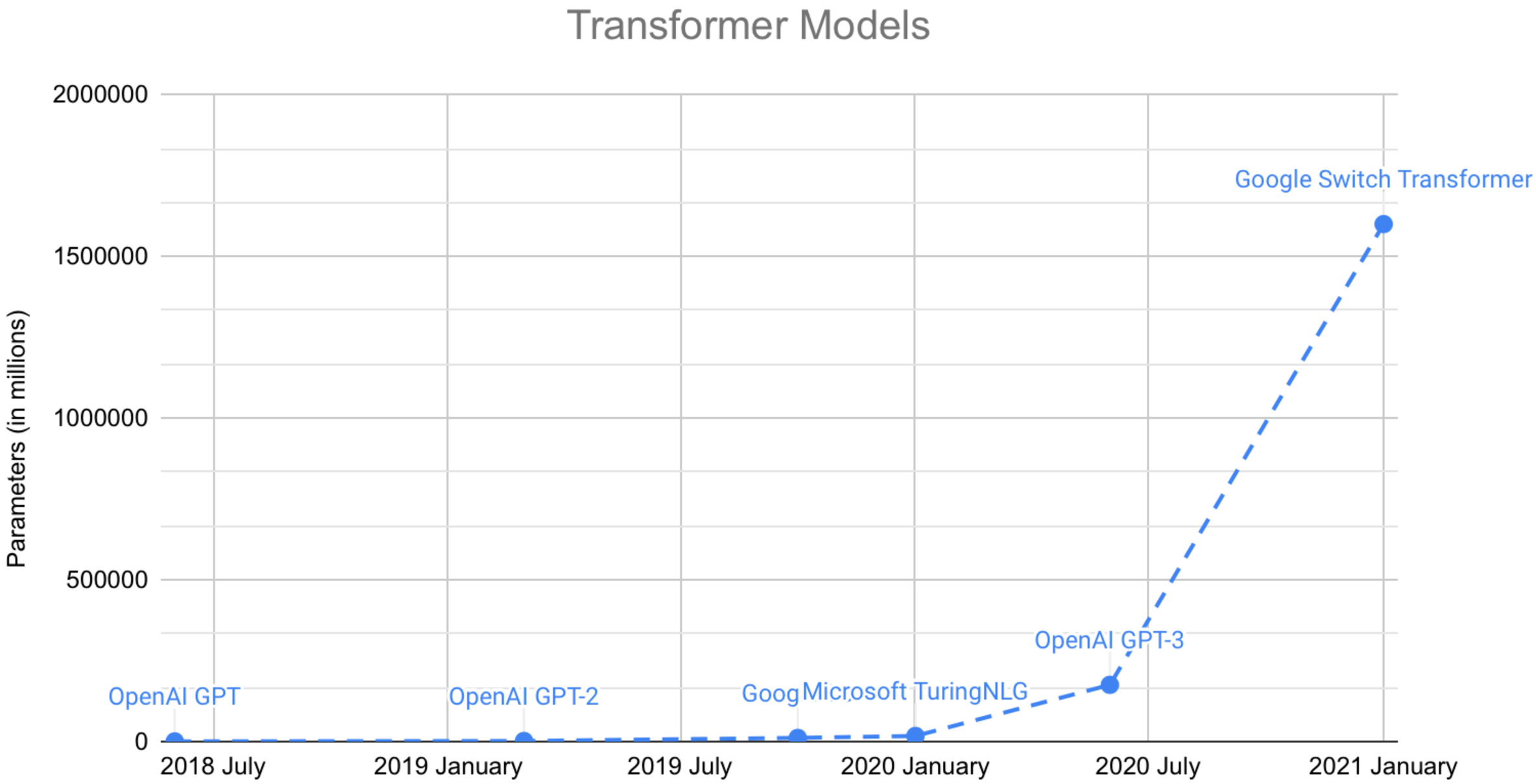
- **Function**
  - Model architecture / hyperparam search
  - Training large models
- **Desiderata**
  - Easy to launch experiments and review results
- **Solutions**
  - Desktop with 4 GPUs
  - Private cluster of GPU machines
  - Cloud cluster of GPU instances


or


Why compute matters



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


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

So,  or  ?

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

GPU Basics

- NVIDIA has been the only game in town
- Google TPUs are the fastest (on GCP only)

GPU Comparison Table

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

https://www.microway.com/knowledge-center-articles/comparison-of-nvidia-geforce-gpus-and-nvidia-tesla-gpus/

GPU Comparison Table

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- New NVIDIA architecture every year
  - Kepler —> Pascal —> Volta -> Turing -> Ampere
- Server version first, then “enthusiast”, then consumer.
  - For business, only supposed to use server cards.

GPU Comparison Table

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- **RAM**: should fit meaningful batches of your model

http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/

GPU Comparison Table

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- **RAM**: should fit meaningful batches of your model
- **32bit vs Tensor TFlops**
  - Tensor Cores are specifically for deep learning operations (mixed precision)
  - Good for convolutional/transformer models

http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/

GPU Comparison Table

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- **RAM**: should fit meaningful batches of your model
- **32bit vs Tensor TFlops**
  - Tensor Cores are specifically for deep learning operations (mixed precision)
  - Good for convolutional/transformer models
- Great speedups and bigger batches from 16bit mixed-precision

http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/

Kepler/Maxwell

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- 2-4x slower than Pascal/Volta
- Hardware: don’t buy, too old
- Cloud: K80's are cheap (providers are stuck with what they bought)

Pascal

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- Hardware: 1080 Ti still good if buying used, especially for recurrent
- Cloud: P100 is a mid-range option

Volta/Turing

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- Preferred choice right now due to 16bit mixed precision support
- Hardware:
  - 2080 Ti is ~1.3x as fast as 1080 Ti in 32bit, but ~2x faster in 16bit
  - Titan RTX is 10-20% faster yet. Titan V is just as good (but less RAM), if find used.
- Cloud: V100 is the ultimate for speed

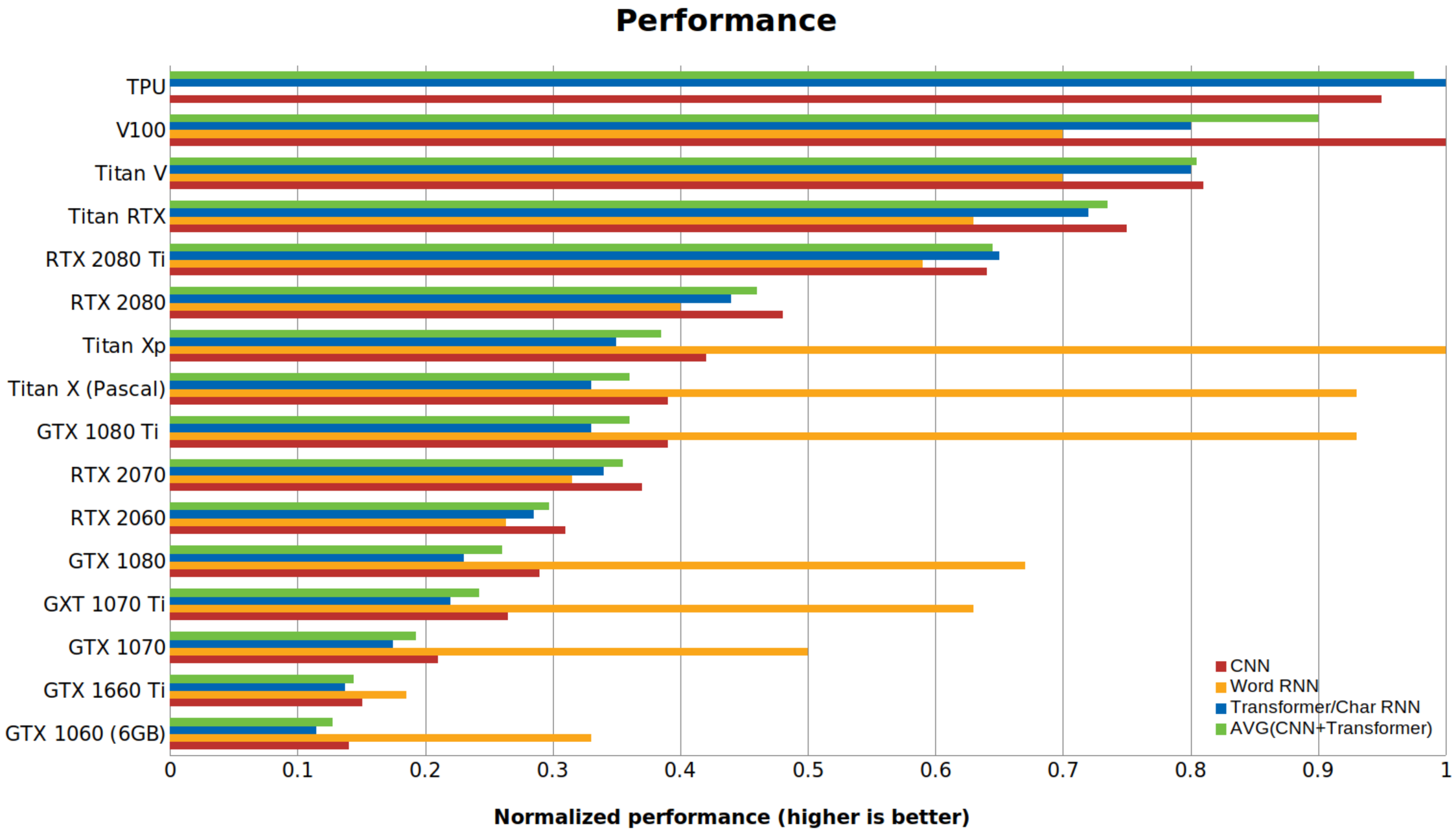
http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/ *https://lambdalabs.com/blog/titan-rtx-tensorflow-benchmarks/*

Ampere

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	-	AWS, GCP, MS
P100	2016H1	Pascal	Server	16	10	N/A	Yes	-	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	$	
V100	2017H1	Volta	Server	16	14	120	Yes	$$$$	AWS, GCP, MS
2080 Ti	2018H2	Turing	Consumer	11	13	107	Yes	$$	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$$	
3090	2021H1	Ampere	Enthusiast	24	?	285	Yes	$$	
A100	2020H1	Ampere	Server	40	19.5	312	Yes	$$$$	AWS, GCP

- Latest hardware, with the most Tensor cores.
- At least 30% speedup over Turing

http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/ *https://lambdalabs.com/blog/titan-rtx-tensorflow-benchmarks/*



https://timdettmers.com/2019/04/03/which-gpu-for-deep-learning/

A100 vs V100

	NVIDIA TESLA A100 SXM4-40GB	NVIDIA TESLA V100 SXM3-32GB
FP32 CUDA Cores	6912	5120
Clock Speed	1410 MHz	1530MHz
Theoretical FP32 TFLOPS	19.5 TFLOPS	15.7 TFLOPS
VRAM	40 GB HBM2e	32 GB HBM2
Memory Bandwidth	1,555 GBps	900 GBps
GPU Interconnect	12 NVLink Connections (600 GBps)	6 NVLink Connections (300 GBps)
Process Node	TSMC 7nm	TSMC 12nm FFN
TDP (W)	400 W	300 W
Power Efficiency	45.2 GFLOPS / W	48.6 GFLOPS / W

For training convnets with PyTorch, the Tesla A100 is...

- 2.2x faster than the V100 using 32-bit precision.*
- 1.6x faster than the V100 using mixed precision.

For training language models with PyTorch, the Tesla A100 is...

- 3.4x faster than the V100 using 32-bit precision.
- 2.6x faster than the V100 using mixed precision.

https://lambdalabs.com/blog/nvidia-a100-vs-v100-benchmarks/

https://lambdalabs.com/deep-learning/servers/hyperplane-a100

Great resource

Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning

2020-09-07 by Tim Dettmers — 1,618 Comments

TL;DR advice

Best GPU overall: RTX 3080 and RTX 3090.

GPUs to avoid (as an individual): Any Tesla card; any Quadro card; any Founders Edition card; Titan RTX, Titan V, Titan XP.

Additional Considerations for Ampere / RTX 30 Series

Summary:

- Ampere allows for sparse network training, which accelerates training by a factor of up to 2x.
- Sparse network training is still rarely used but will make Ampere future-proof.
- Ampere has new low-precision data types, which makes using low-precision much easy, but not necessarily faster than for previous GPUs.
- The new fan design is excellent if you have space between GPUs, but it is unclear if multiple GPUs with no space in-between them will be efficiently cooled.
- 3-Slot design of the RTX 3090 makes 4x GPU builds problematic. Possible solutions are 2-slot variants or the use of PCIe extenders.
- 4x RTX 3090 will need more power than any standard power supply unit on the market can provide right now.

The Most Important GPU Specs for Deep Learning Processing Speed

This section can help you build a more intuitive understanding of how to think about deep learning performance. This understanding will help you to evaluate future GPUs by yourself.

Tensor Cores

Summary:

- Tensor Cores reduce the used cycles needed for calculating multiply and addition operations, 16-fold — in my example, for a 32×32 matrix, from 128 cycles to 8 cycles.
- Tensor Cores reduce the reliance on repetitive shared memory access, thus saving additional cycles for memory access.
- Tensor Cores are so fast that computation is no longer a bottleneck. The only bottleneck is getting data to the Tensor Cores.

https://timdettmers.com/2020/09/07/which-gpu-for-deep-learning/#GPU_Recommendations

Cloud Providers

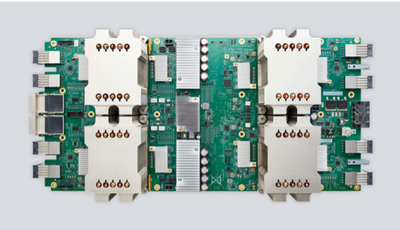
- Amazon Web Services, Google Cloud Platform, Microsoft Azure are the heavyweights.
- Heavyweights are largely similar in function and price.
  - AWS most expensive
  - GCP just about as expensive, and has TPUs
  - Azure reportedly has bad user experience
- Startups are Coreweave, Lambda Labs, and more

Amazon Web Services

Name	GPU	GPUs	GPU RAM	vCPU	RAM	On-demand
p2.16xlarge	K80	8	12	64	732	$14.40
p3.16xlarge	V100	8	16	64	488	$24.48
p3dn.24xlarge	V100	8	32	96	768	$31.22
p4dn.16xlarge	A100	8	40	96	1152	$32.78

- Three generations of GPUs
- Prices haven't moved in years

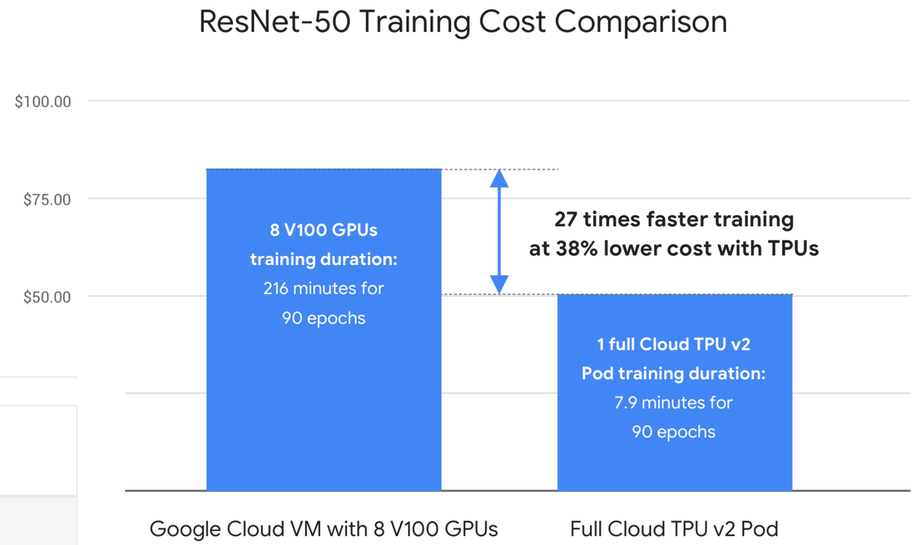
Google Cloud Platform



Cloud TPU v2

180 teraflops

64 GB High Bandwidth Memory (HBM)



US EUROPE ASIA PACIFIC

Version	On-demand	Preemptible
Cloud TPU v2	$4.50 / TPU hour	$1.35 / TPU hour
Cloud TPU v3	$8.00 / TPU hour	$2.40 / TPU hour

- Same lineup: K80, V100 (also P100), and A100
- In general, a little cheaper than AWS.
- Also has “Tensor Processing Units” (TPUs), the fastest option today.

Microsoft Azure

- Same lineup: K80, P100s, V100s (no A100s yet)
- Similar pricing

https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/

Lambda Labs Cloud

4-GPU Instance

For researchers, students, and hobbyists looking for extra power at low cost.

INSTANCE SPECIFICATIONS

GPU DETAILS
4x NVIDIA® Pascal Based GPUs (11 GB)

VCPUS
8

SYSTEM RAM
32 GB

TEMPORARY STORAGE
1.4 TB SATA SSD

NETWORK INTERFACE
1 Gbps (peak)

Dedicated 4-GPU instance pricing

$1.50 / hr

Launch an instance

8-GPU Instance NEW

For ML engineers and researchers looking for maximum training speed.

INSTANCE SPECIFICATIONS

GPU DETAILS
8x NVIDIA® V100 Tensor Core (16 GB) + NVLink™

VCPUS
92

SYSTEM RAM
448 GB

TEMPORARY STORAGE
6 TB NVMe

NETWORK INTERFACE
10 Gbps (peak)

Dedicated 8-GPU instance pricing

$12.00 / hr

Launch an instance

https://lambdalabs.com/service/gpu-cloud

Coreweave

The industry's best economics. Period.

Priced by the hour, billed by the minute

GPUs | Virtual Workstations | CPU Compute

Qty	GPU Type	VRAM	vCPU	RAM	Storage	Network	$/Hour	Optimized for
– 4 +	NVIDIA A100	40GB HBM2e	120	512 GB	1800 GB	40 Gbps	$12.00	
– 4 +	NVIDIA A6000	48GB GDDR6	32	512 GB	1800 GB	10 Gbps	$8.00	
– 8 +	NVIDIA V100 for NVLINK	16GB HBM2	32	256 GB	3600 GB	10 Gbps	$8.00	
– 8 +	NVIDIA V100 for PCIe	16GB HBM2	24	160 GB	512 GB	1 Gbps	$4.80	
– 4 +	NVIDIA P100 for NVLINK	16GB HBM2	16	128 GB	1800 GB	10 Gbps	$3.00	
– 4 +	NVIDIA Quadro RTX 6000	24GB GDDR6	32	240 GB	1800 GB	10 Gbps	$5.40	
– 4 +	NVIDIA Quadro RTX 5000	16GB GDDR6	32	240 GB	1800 GB	10 Gbps	$3.80	
– 4 +	NVIDIA Quadro RTX 4000	8GB GDDR6	12	64 GB	500 GB	1 Gbps	$1.40	

https://www.coreweave.com/pricing

On-prem Options

- Build your own
  - Up to 4 Turing or 2 Ampere GPUs is easy
- Buy pre-built
  - Lambda Labs, NVIDIA, and builders like Supermicro, Cirrascale, etc.

Building your own

- Quiet PC with 128GB RAM and 2x RTX 3090's: $8000
  - (If you can find them)
  - One day to build and set up
- Going beyond 4 2000-series or 2 3000-series is painful
- All you need to know: http://timdettmers.com/2018/12/16/deep-learning-hardware-guide/

Pre-built: Lambda Labs

AMD Threadripper 3960X
24 cores, 3.80 GHz

2x RTX 3090
24 GB VRAM per GPU

256 GB
System memory

3.84 TB NVMe
OS Drive

3.84 TB SSD (SATA)
Extra Storage

Customize your computer

Operating system

- Ubuntu 20.04 + Lambda Stack
- Ubuntu 18.04 + Lambda Stack + $0
- Windows 10 + $200
- No Operating System + $0

Processor

- **AMD Threadripper 3960X**
  24 cores, 3.80 GHz, 128 MB cache

~20% more expensive than building yourself

$11,446.00
Apply for a discount

Download quote | Add to cart

Pre-built: Lambda Labs

Premium

8x Tesla V100 Server

8-way NVLink

2x Xeon Gold 6248 (20 cores)

8x Tesla V100 GPUs (32 GB VRAM)

512 GB RAM

Customizable storage

100 Gbps InfiniBand

$94,870

Academic Discounts

COVID-19 Research Discounts

Customize

Lambda Hyperplane 4-A100

4 NVIDIA A100 Tensor Core GPUs with NVLink™ & Mellanox InfiniBand

SYSTEM SPECIFICATIONS

GPU DETAILS
4x NVIDIA Tesla A100 SXM4-40GB + NVLink

PROCESSOR
2x AMD EPYC™ Processors (Up to 64 cores)

SYSTEM RAM
512 GB

STORAGE
Up to 60TB NVMe

NETWORK INTERFACE
Up to 4x Mellanox InfiniBand HDR 200Gbps Cards

Hyperplane 4-A100 pricing starting at

$65,000

Get a quote

Cost Analysis

- Let’s first compare on-prem and equivalent cloud machines
- Then let’s also consider spot instances for experiment scaling

Quad PC vs Quad Cloud

Full-time workload

Work week load

GPU	Arch	RAM	Build Price	Cloud Price	Hours = Build	24/7 Weeks = Build	16/5 Weeks = Build
4x RTX 2080 Ti	Volta	12	$10000.00	0			
4x V100	Volta	16		$12.00	833	5	10

Verdict: not worth it. PC pays for itself in 5-10 weeks.

Quad PC vs Quad Cloud

481

Posted by u/cgnorthcutt 8 months ago

[P] I built Lambda's $12,500 deep learning rig for $6200

Project

See: http://l7.curtisnorthcutt.com/build-pro-deep-learning-workstation

Hi Reddit! I built a 3-GPU deep learning workstation similar to Lambda's 4-GPU (RTX 2080 TI) rig for half the price. In the hopes of helping other researchers, I'm sharing a time-lapse of the build, the parts list, the receipt, and benchmarking versus Google Compute Engine (GCE) on ImageNet. You save $1200 (the cost of an EVGA RTX 2080 ti GPU) per ImageNet training to use your own build instead of GCE. The training time is reduced by over half. In the post, I include 3 GPUs, but the build (increase PSU wattage) will support a 4th RTX 2080 TI GPU for $1200 more ($7400 total). Happy building!

https://l7.curtisnorthcutt.com/build-pro-deep-learning-workstation

Quad PC vs. Spot Instances

Length of trial in experiment (hours)	6
Number of trials in experiment	16
Total GPU hours for experiment	96
Cost of 4x RTX 2080 Ti machine	$10,000.00
Time to run experiment on 4x machine	24 hours
Time to run experiment on V100 spot instances	6 hours
Cost of provisioning enough pre-emptible V100s	$96.00
Number of experiments that equal cost of 4x	104

How to think about it: cloud enables quicker experiments.

Quad PC vs. Spot Instances

Length of trial in experiment (hours)	**6**	
Number of trials in experiment	16	
Total GPU hours for experiment	96	
Cost of 4x RTX 2080 Ti machine	$10,000.00	
Time to run experiment on 4x machine	24	hours
Time to run experiment on V100 spot instances	6	hours
Cost of provisioning enough pre-emptible V100s	$96.00	
Number of experiments that equal cost of 4x	104	

But at a pretty steep price.

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

Recommendation for hobbyist

- Development
  - Build a 4x Turing or 2x Ampere PC
- Training/Evaluation
  - Use the same PC, just always keep it running
  - To scale out, use Lambda or Coreweave cloud instances.

Recommendation for startup

- Development
  - Buy a 4x Turing or 2x Ampere PC per ML Scientist
- Training/Evaluation
  - To scale out, buy shared server machines or use cloud instances.

Recommendation for larger company

- Development
  - Buy 8x Turing or Ampere server rack per ML scientist
  - Or, go straight to the cloud with V100 instances
- Training/Evaluation
  - To scale out, use cloud instances with proper provisioning and handling of failures

Questions?

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

"All-in-one"

Data

- Versioning: DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Apache Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, LOG, Parquet

Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- Hyperparameter Tuning: SIGOPT, Determined AI, Weights & Biases, tune
- Experiment Management: Weights & Biases, comet, TensorBoard, Neptune Machine Learning Lab, mlflow
- Resource Management: Determined AI, PY, slurm workload manager
- Software Engineering: jupyter, Streamlit, git
- Compute: CoreWeave, Lambda

Deployment

- Feature Store: tecton
- Monitoring: fiddler
- Edge: ML, OctoML, NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations

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 workload manager
  - Docker + Kubernetes
  - Software specialized for ML use cases

Scripts or slurm

workload manager

- Problem we're solving: allocate free resources to programs
- Can be scripted pretty easily
- Even better, use old-school cluster job scheduler
  - Job defines necessary resources, gets queued

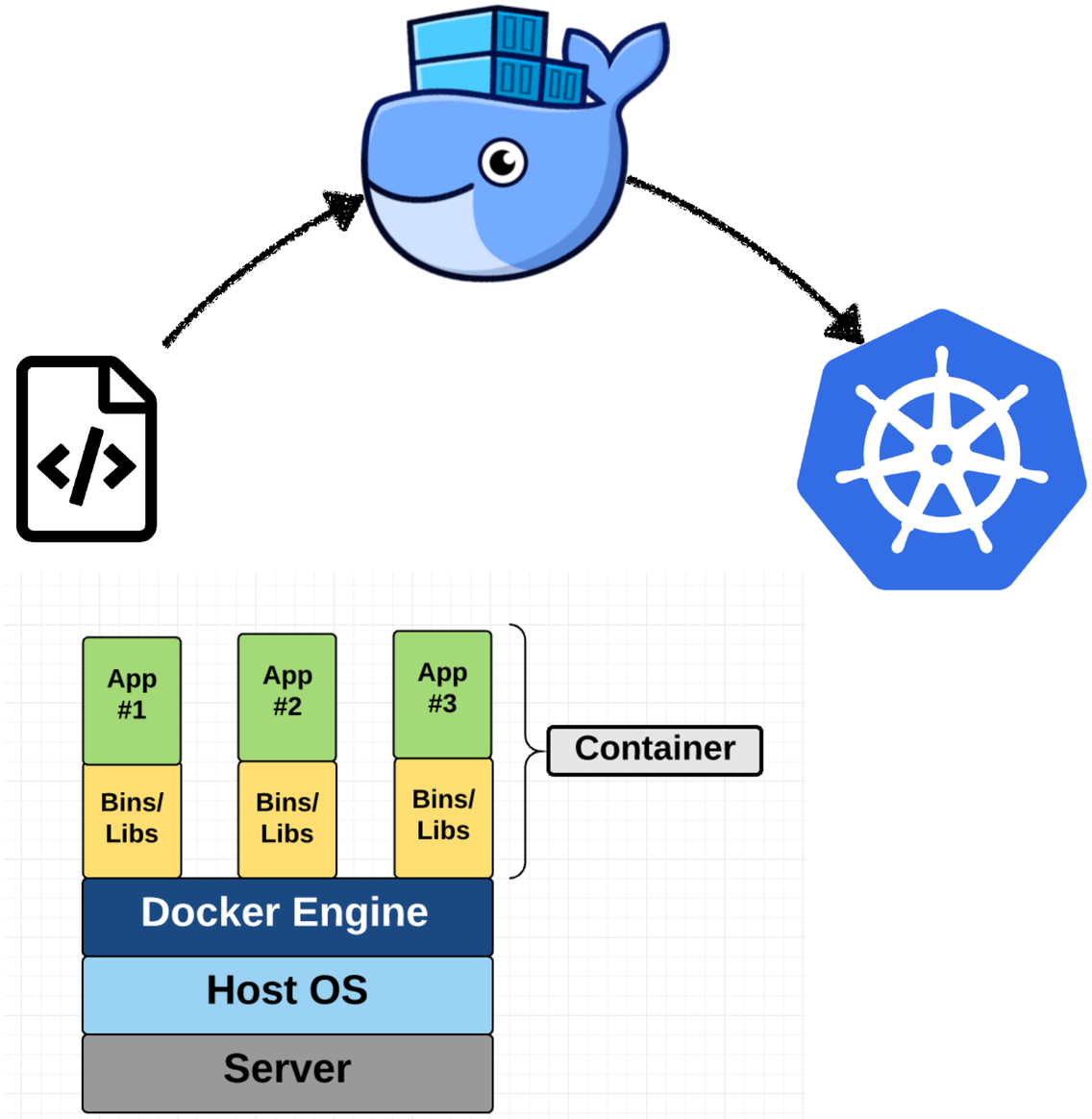
```python
class GPUManager(object):
    def __init__(self, verbose: bool=False):
        self.lock_manager = Redlock([{"host": "localhost", "port": 6379, "db": 0}, ])
        self.verbose = verbose

    def get_free_gpu(self):
        """
        If some GPUs are available, try reserving one by checking out an exclusive redis lock.
```

Job-file (schedule job with sbatch, check status with squeue -u <Username>):

```
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --mem=10000
#SBATCH -p gpu2    # K80 GPUs on Haswell node
#SBATCH --time=01:00:00

## with Theano (using configs from above)
module purge # purge if you already have modules loaded
module load modenv/eb
module load Keras

srun python mnist_cnn.py


## with Tensorflow
module purge
module load modenv/eb
module load Keras
module load tensorflow
# if you see 'broken pipe error's (might happen in interactive session after
module load h5py/2.6.0-intel-2016.03-GCC-5.3-Python-3.5.2-HDF5-1.8.17-serial

export KERAS_BACKEND=tensorflow      # configure Keras to use tensorflow

srun python mnist_cnn.py
```

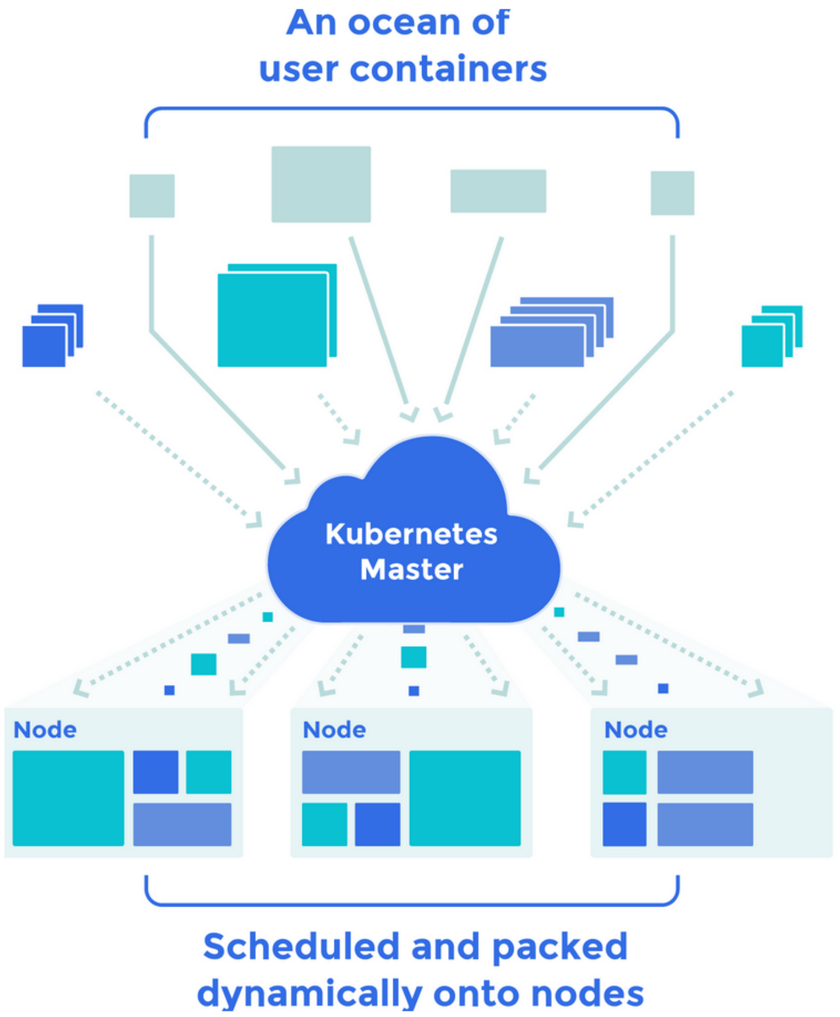
Docker + Kubernetes

- Docker is a way to package up an entire dependency stack in a lighter-than-a-VM package
- We will talk more about Docker in the Deployment lecture, and use it in lab



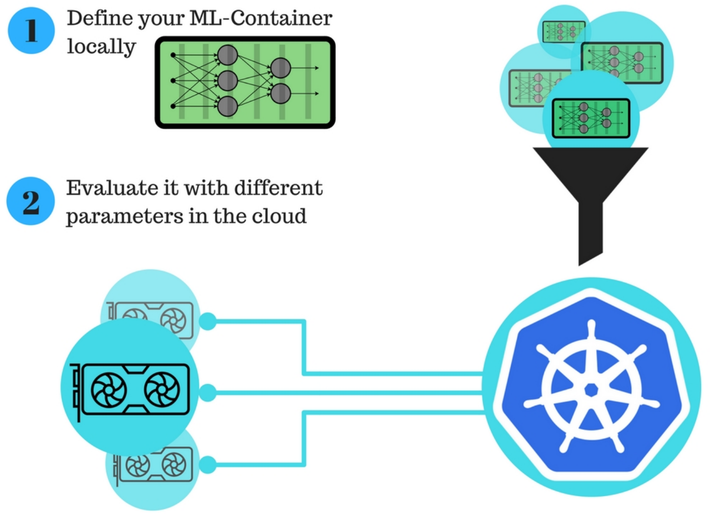
Docker + Kubernetes

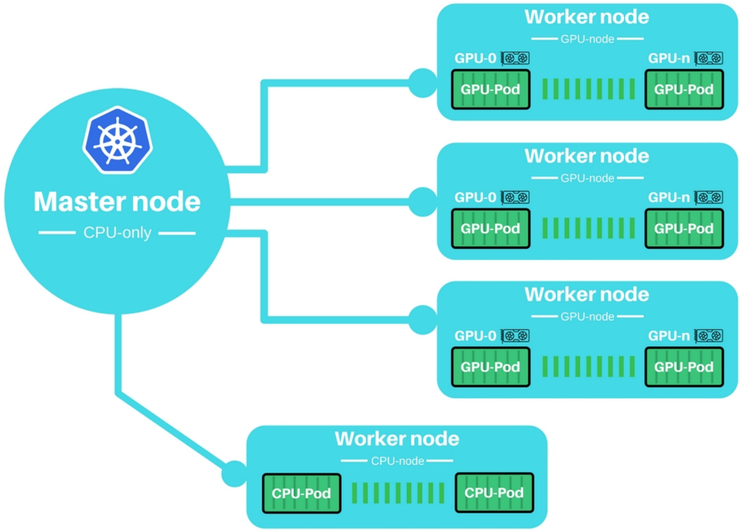
- Kubernetes is a way to run many docker containers on top of a cluster



Kubeflow

- Open source project from Google
- Spawn and manage Jupyter notebooks
- Manage multi-step ML workflows
- Plug-ins for hyperparameter tuning, model deployment





https://github.com/Langhalsdino/Kubernetes-GPU-Guide

This is an active area

- All-in-one solutions like SageMaker and Paperspace Gradient do this
- And some recently-announced startups have it as their goal...

Anyscale (makers of Ray, from Berkeley)

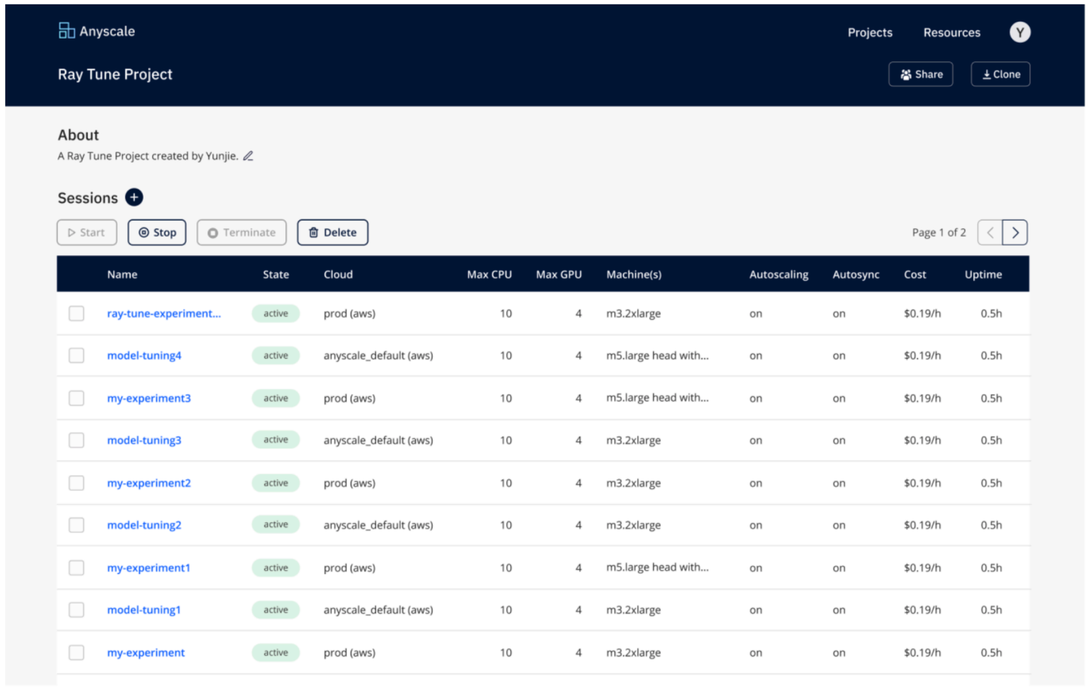
Serverless experience

without Serverless Limitations

Anyscale is cloud agnostic, supports stateless and stateful computations, GPUs and TPUs and autoscaling.

The serverless experience abstracts away servers, clusters making it easier for you to focus on building your application.

Autoscaling makes it easy to move from a single node to many with the click of a button.



Grid.ai

Seamlessly train hundreds of Machine Learning models on the cloud from your laptop.

Focus on machine learning, not infrastructure.

From the makers of PyTorch Lightning

```
python model.py --learning_rate 1e-6
```

```
grid train --grid_gpus 8 model.py --learning_rate "uniform(1-e6,1e-1,20)"
```

Questions?

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

"All-in-one"

Data

- Versioning: DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Apache Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, LOG, Parquet

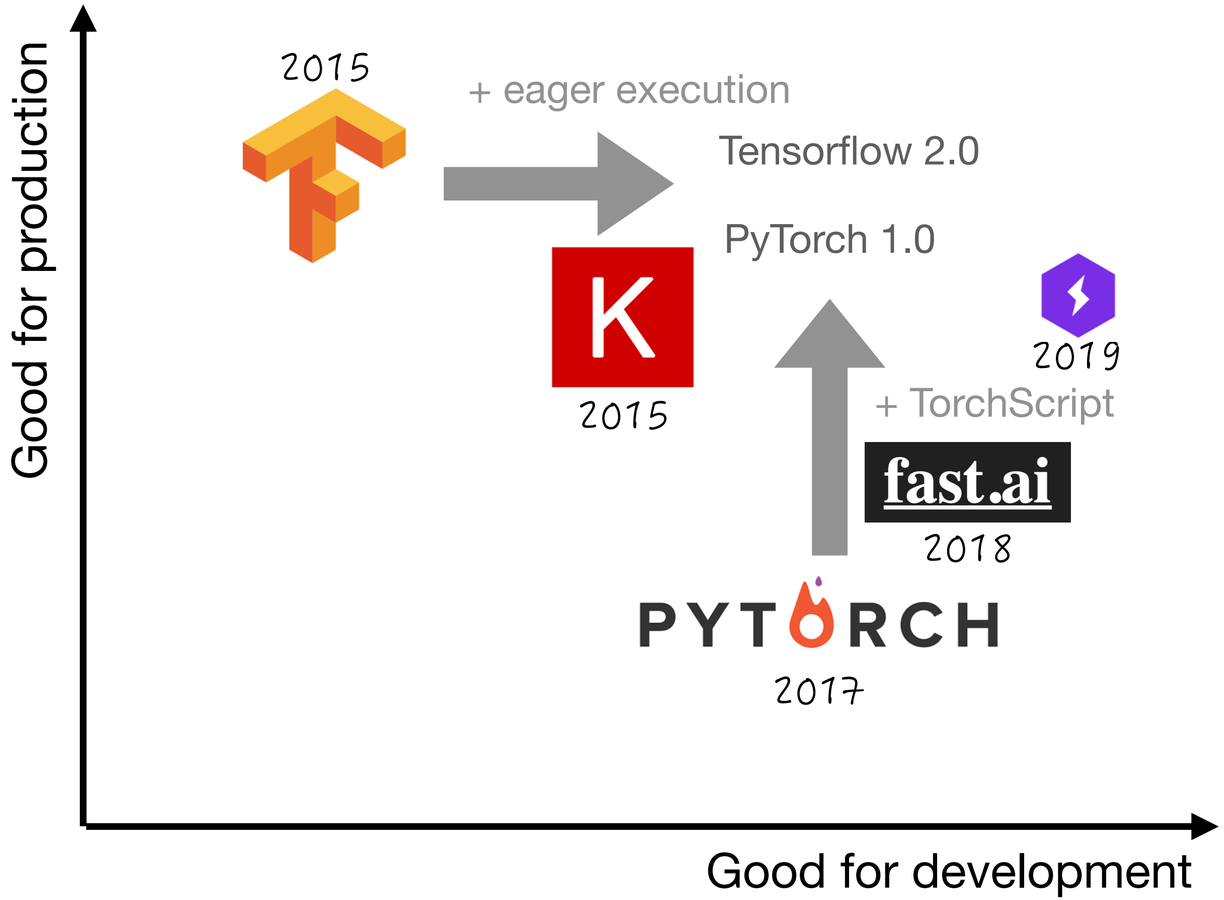
Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- Hyperparameter Tuning: SIGOPT, Determined AI, Weights & Biases, tune
- Experiment Management: Weights & Biases, comet, TensorBoard, Neptune Machine Learning Lab, mlflow
- Resource Management: Determined AI, PY, slurm workload manager
- Software Engineering: jupyter, Streamlit, git
- Compute: CoreWeave, Lambda

Deployment

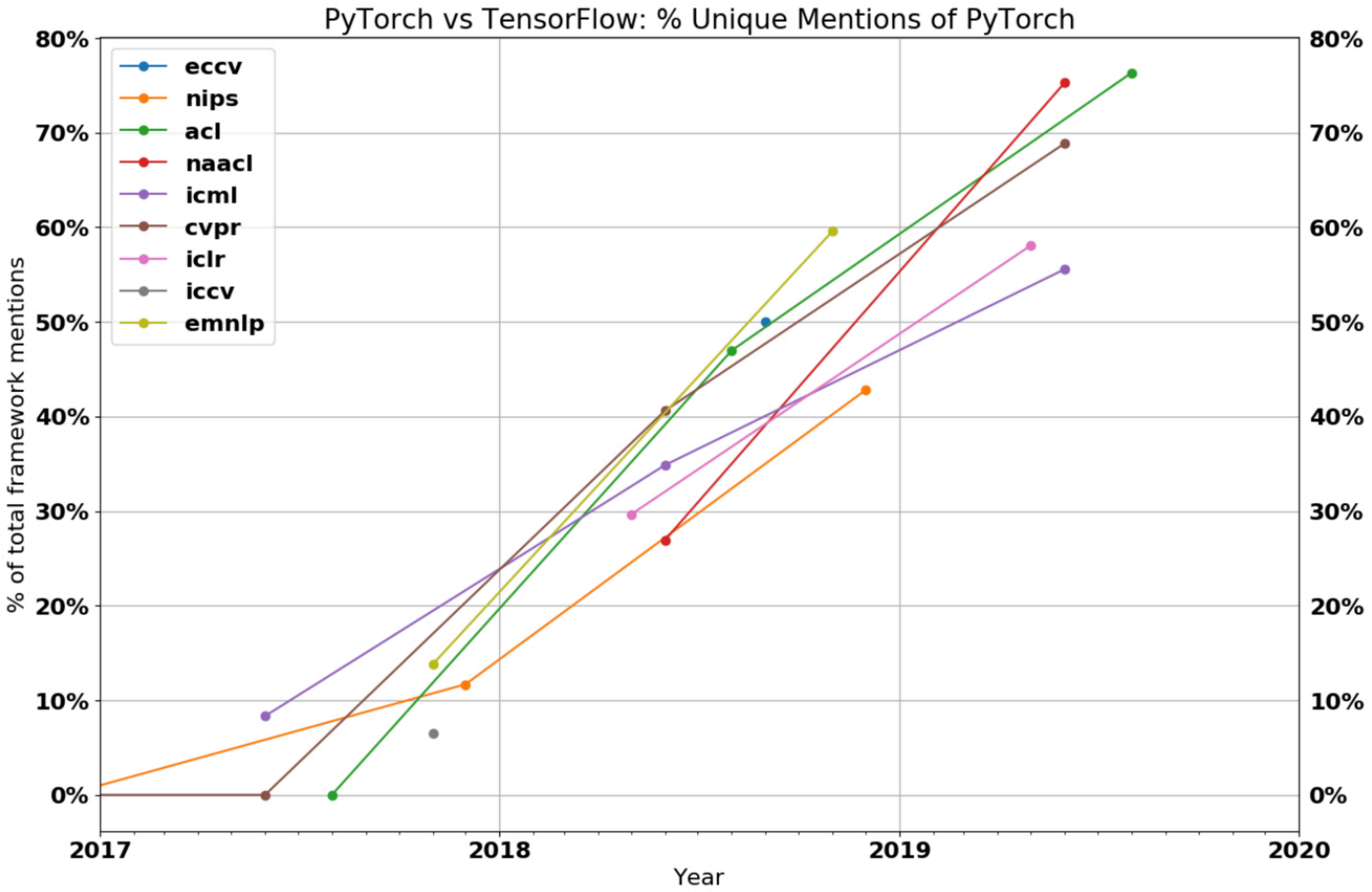
- Feature Store: tecton
- Monitoring: fiddler
- Edge: ML, OctoML, NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations

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

PyTorch dominates new development



https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

PyTorch and Tensorflow

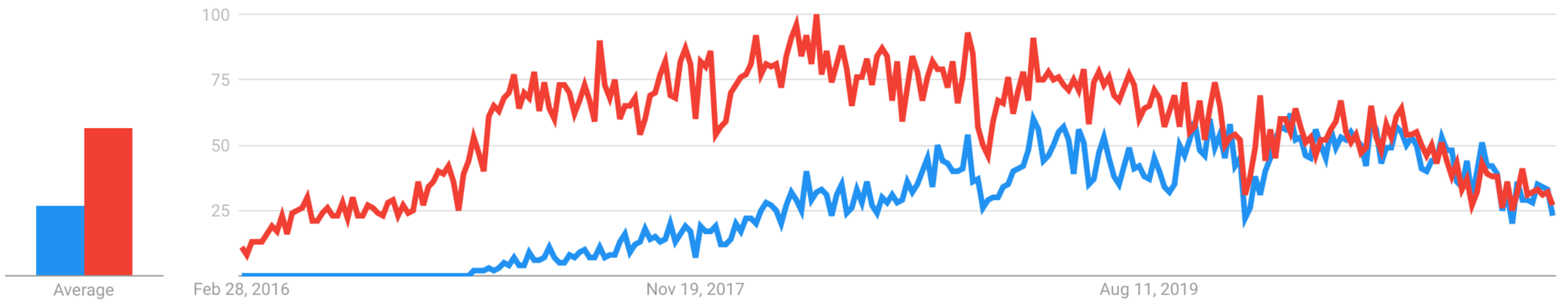
pytorch
Search term

tensorflow
Search term

+ Add comparison

United States | Past 5 years | All categories | Web Search

Interest over time



Why do we need frameworks?

- Deep Learning is not lot of code with a matrix math library (Numpy)
- BUT: auto-differentiation and CUDA are a lot of work
- ...Also all the layer types, optimizers, data interfaces, etc.

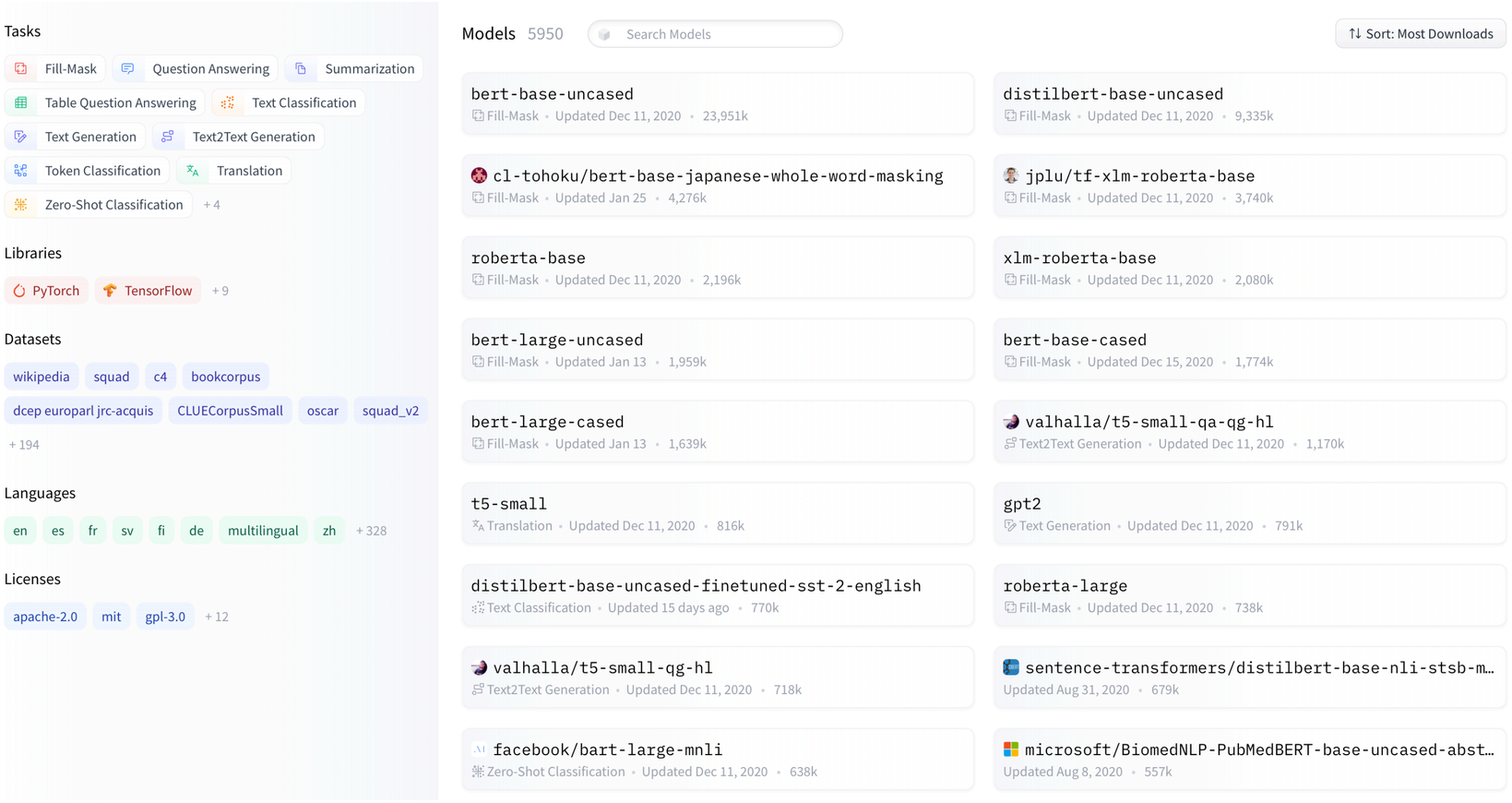
```python
class Linear:
  def __init__(self, input_dim: int, num_hidden: int = 1):
    self.weights = np.random.randn(input_dim, num_hidden) * np.sqrt(2. / input_dim)
    self.bias = np.zeros(num_hidden)

  def __call__(self, x):
    self.x = x
    output = x @ self.weights + self.bias
    return output

  def backward(self, gradient):
    self.weights_gradient = self.x.T @ gradient
    self.bias_gradient = gradient.sum(axis=0)
    self.x_gradient = gradient @ self.weights.T
    return self.x_gradient

  def update(self, lr):
    self.weights = self.weights - lr * self.weights_gradient
    self.bias = self.bias - lr * self.bias_gradient
```

- Recent project from Google that's gaining steam
- Numpy + auto-differentiation and compilation to GPU/TPU code
- Not just for deep learning!

https://github.com/google/jax#quickstart-colab-in-the-cloud

Hugging face

- Tons of NLP-focused model architectures (and pre-trained weights) for both PyTorch and Tensorflow

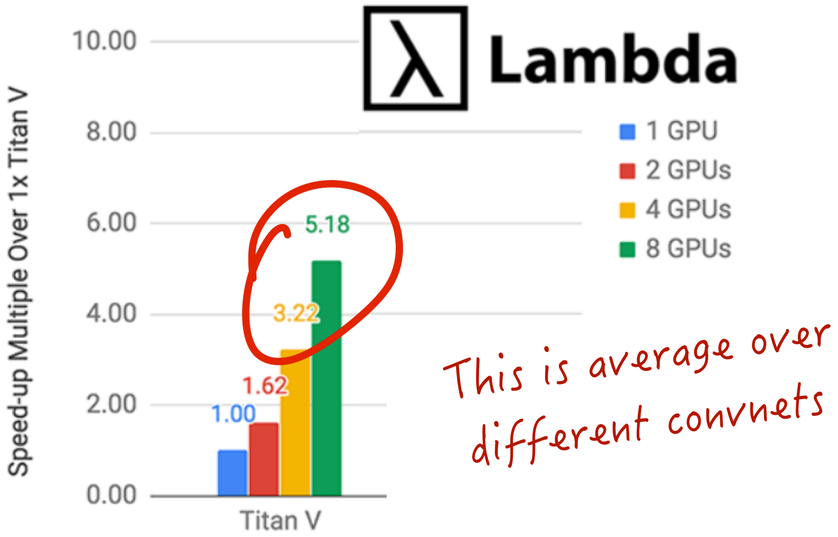


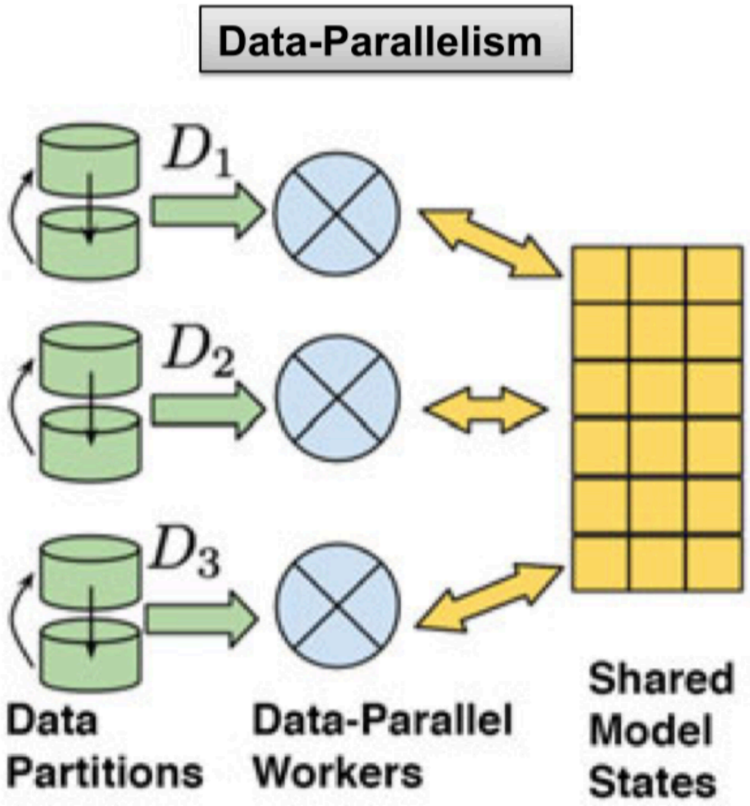
Distributed Training

- 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

Data Parallelism

- If iteration time is too long, try training in data parallel regime
- "For convolution, expect 1.9x/3.5x speedup for 2/4 GPUs.
  http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/



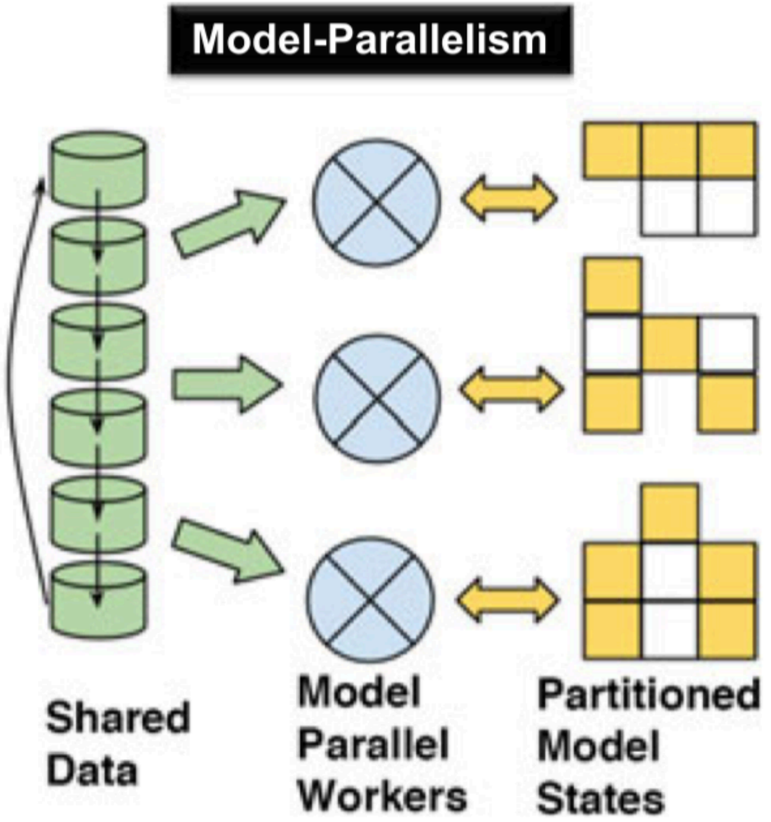


http://www.cs.cmu.edu/~pengtaox/papers/petuum_15.pdf

https://lambdalabs.com/blog/titan-v-deep-learning-benchmarks/

Model Parallelism

- Model parallelism is necessary when model does not fit on a single GPU
  - Introduces a lot of complexity and is usually not worth it. (But this is changing.)
  - Better to buy the largest GPU you can, and/or use gradient checkpointing



http://www.cs.cmu.edu/~pengtaox/papers/petuum_15.pdf

Data-parallel PyTorch

- Can be quite easy:

```
model = Model(input_size, output_size)
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  # dim = 0 [30, xxx] -> [10, ...], [10, ...], [10, ...] on 3 GPUs
  model = nn.DataParallel(model)

model.to(device)

for data in rand_loader:
    input = data.to(device)
    output = model(input)
    print("Outside: input size", input.size(),
          "output_size", output.size())
```

```
Let's use 2 GPUs!
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
    In Model: input size torch.Size([15, 5]) output size torch.Size([15, 2])
```

Data-parallel PyTorch-Lightning

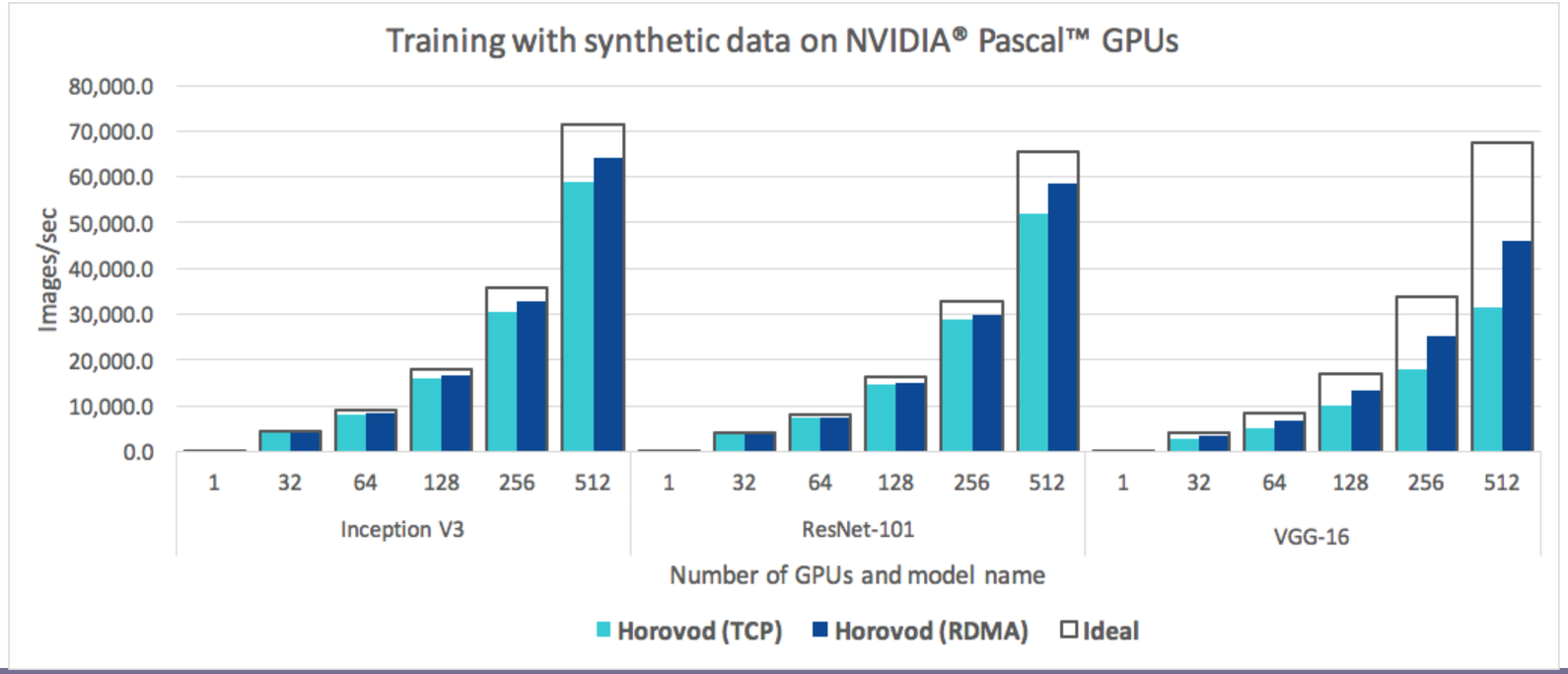
- Single node:
  python training/run_experiment.py --gpus=8 --accelerator=ddp

- And using SLURM to run on multiple nodes:
  python training/run_experiment.py --gpus=8 --accelerator=ddp --nodes=4

```
# (submit.sh)
#!/bin/bash -l

# SLURM SUBMIT SCRIPT
#SBATCH --nodes=4
#SBATCH --gres=gpu:8
#SBATCH --ntasks-per-node=8
#SBATCH --mem=0
#SBATCH --time=0-02:00:00

# activate conda env
source activate $1

# debugging flags (optional)
export NCCL_DEBUG=INFO
export PYTHONFAULTHANDLER=1

# on your cluster you might need these:
# set the network interface
# export NCCL_SOCKET_IFNAME=^docker0,lo

# might need the latest CUDA
# module load NCCL/2.4.7-1-cuda.10.0

# run script from above
srun python3 train.py
```

Horovod

- Distributed training framework for Tensorflow, Keras, and PyTorch
- Uses MPI (standard multi-process communication framework) instead of Tensorflow parameter servers
- Could be an easier experience for multi-node training



Ray (from Anyscale)

- Ray is open-source project for effortless, stateful distributed computing in Python
- From Berkeley!

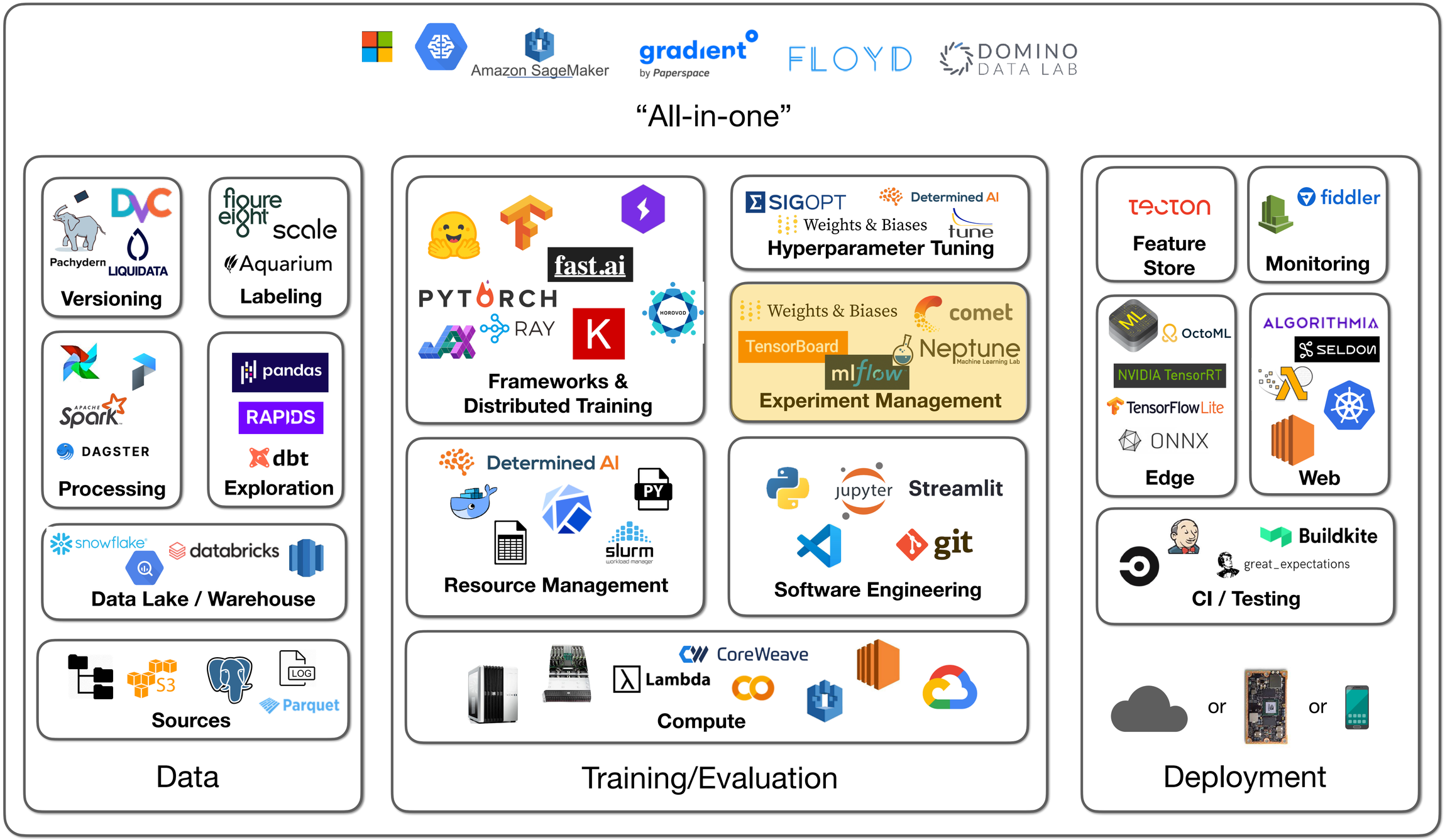
```
ray.init(args.address)

trainer1 = PyTorchTrainer(
    model_creator,
    data_creator,
    optimizer_creator,
    loss_creator,
    num_replicas=<NUM_GPUS_YOU_HAVE> * <NUM_NODES>,
    use_gpu=True,
    batch_size=512,
    backend="nccl")

stats = trainer1.train()
print(stats)
trainer1.shutdown()
print("success!")
```

Then, start a Ray cluster via autoscaler or manually.

```
ray up CLUSTER.yaml
python train.py --address="localhost:<PORT>"
```

https://ray.readthedocs.io/en/latest/walkthrough.html

Questions?


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
LOG
Parquet
Sources
Data
fast.ai
PYTORCH
RAY
HOROVOD
Frameworks & Distributed Training
SIGOPT
Determined AI
Weights & Biases
tune
Hyperparameter Tuning
Weights & Biases
comet
TensorBoard
Neptune
Machine Learning Lab
mlflow
Experiment Management
Determined AI
PY
slurm
workload manager
Resource Management
jupyter
Streamlit
git
Software Engineering
CoreWeave
Lambda
Compute
Training/Evaluation
tecton
Feature Store
fiddler
Monitoring
ML
OctoML
NVIDIA TensorRT
TensorFlow Lite
ONNX
Edge
ALGORITHMIA
SELDON
Web
Buildkite
great_expectations
CI / Testing
or
or
Deployment

Experiment Management

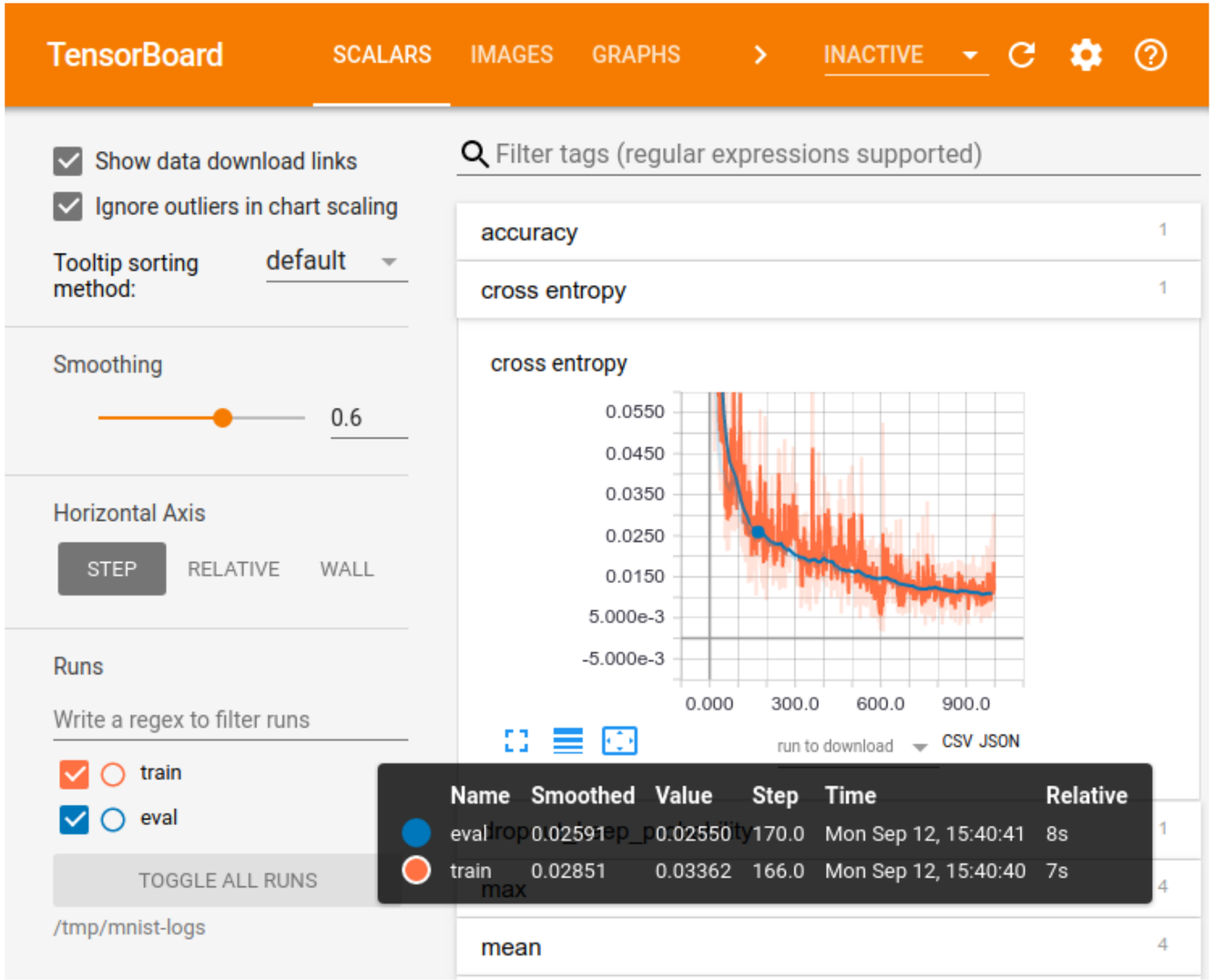
- Even running one experiment at a time, can lose track of which code, parameters, and dataset generated which trained model.
- When running multiple experiments, problem is much worse.

Dataset	Split (trian/dev/test)	0.7/0.2/0.1	0.7/0.2/0.1	0.7/0.2/0.1	0.7/0.15/0.15	0.7/0.15/0.15
	Class ratio (train/dev/test)	0.42/0.42/0.42	0.42/0.42/0.42	0.42/0.42/0.42	/0.3/0.3	/0.3/0.3
	train/dev/test size	4871/1392/696	4871/1392/696	5315/1518/760	5315/1139/1139	5315/1139/1139
Training hyperparameters	Learning rate	1.00E-05	1.00E-05	1.00E-05	1.00E-05	1.00E-05
	epoch	3	2	1	5	6
	batch size	32	32	32	32	32
Results	accuracy	0.88304595	0.8650862069	0.8687747	0.86997364	0.65
	f1	0.82495437	0.8108753316	0.82383946	0.81827954	0.44
	precision	0.878865	0.7848381601	0.8407407	0.8556561	0.56
	recall	0.7780239	0.8389705882	0.8076923	0.78442625	0.36
	tp	1398	1402	1460	1334	1130
	tn	1692	1663	1707	1543	1504
	fp	1113	1142	1161	1108	1148
	fn	1189	1185	1190	1154	1357
	loss	0.59637538	0.594134	0.594134	0.6037084	0.594134
Test results	accuracy	0.90747	0.90747	0.88026	0.88314	0.75847
	f1	0.85636	0.85636	0.83108	0.83469	0.5915
	precision	0.90934	0.90934	0.86689	0.87027	0.77626
	recall	0.8099	0.8099	0.79846	0.80226	0.48604

https://towardsdatascience.com/tracking-ml-experiments-using-mlflow-7910197091bb

Tensorboard

- A fine solution for single experiments
- Gets unwieldy to manage many experiments, and to properly store past work



MLFlow tracking

An open source platform for the machine learning lifecycle

- Self-hosted solution from DataBricks

/Shared/experiments/cryptocurrency/analysis-forecasting-1

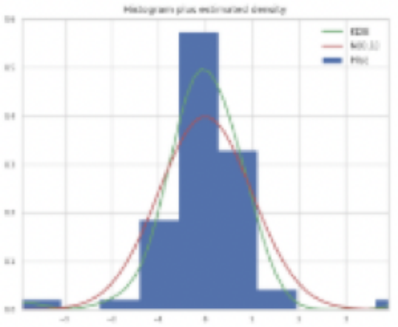
Experiment ID: 450992 Artifact Location: dbfs:/databricks/mlflow/450992

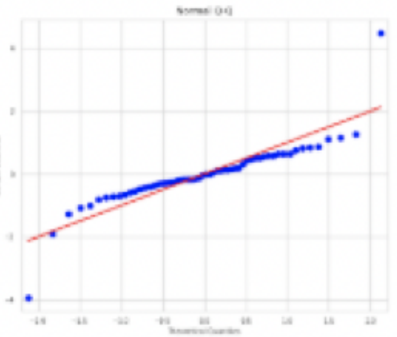
Search Expression: metrics.rmse < 1 and params.model = "tree" State: Active Search Clear

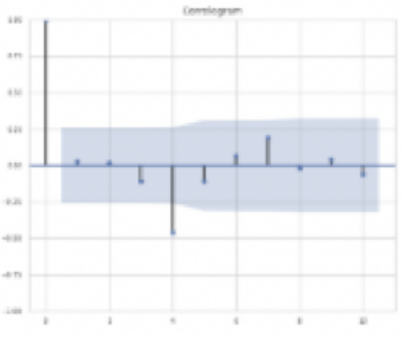
Params: alpha, lr Metrics: rmse, r2

8 matching runs Compare Delete Download CSV

	Date	User	Run Name	Source	Version	Tags	Parameters		Metrics
							param-ps	param-qs	metric
	2019-07-13 08:20:35	hafidzz	arima_param	cryptocurrency-price-forecasting-after			2	2	-219.8
	2019-07-13 08:20:34	hafidzz	arima_param	cryptocurrency-price-forecasting-after			1	2	-221.7
	2019-07-13 08:20:33	hafidzz	arima_param	cryptocurrency-price-forecasting-after			0	2	-223.2
	2019-07-13 08:20:32	hafidzz	arima_param	cryptocurrency-price-forecasting-after			2	1	-221.2
	2019-07-13 08:20:32	hafidzz	arima_param	cryptocurrency-price-forecasting-after			1	1	-223.2
	2019-07-13 08:20:31	hafidzz	arima_param	cryptocurrency-price-forecasting-after			0	1	-225.1
	2019-07-13 08:20:30	hafidzz	arima_param	cryptocurrency-price-forecasting-after			2	0	-222.9
	2019-07-13 08:20:30	hafidzz	arima_param	cryptocurrency-price-forecasting-after			1	0	-224.7

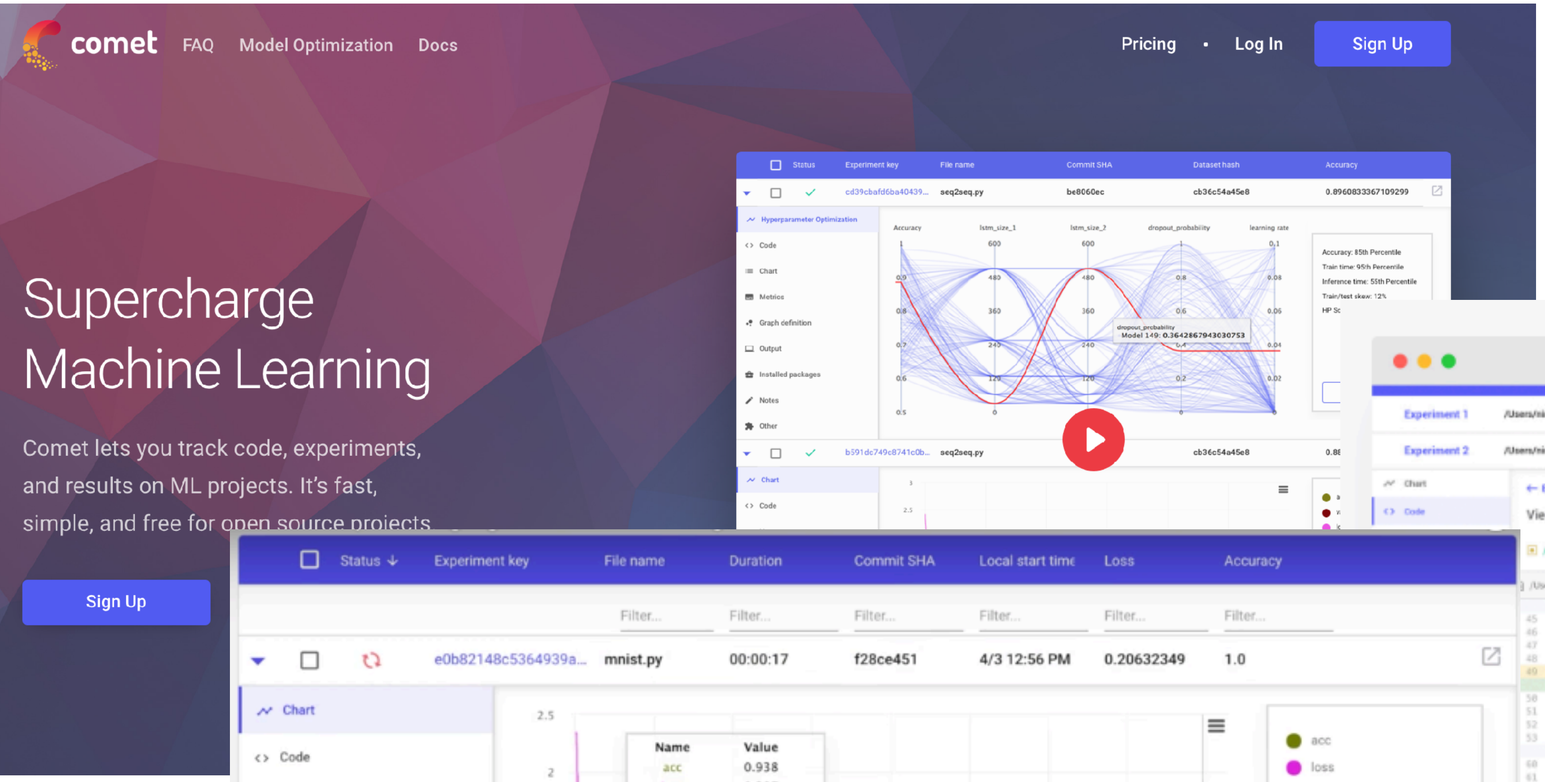


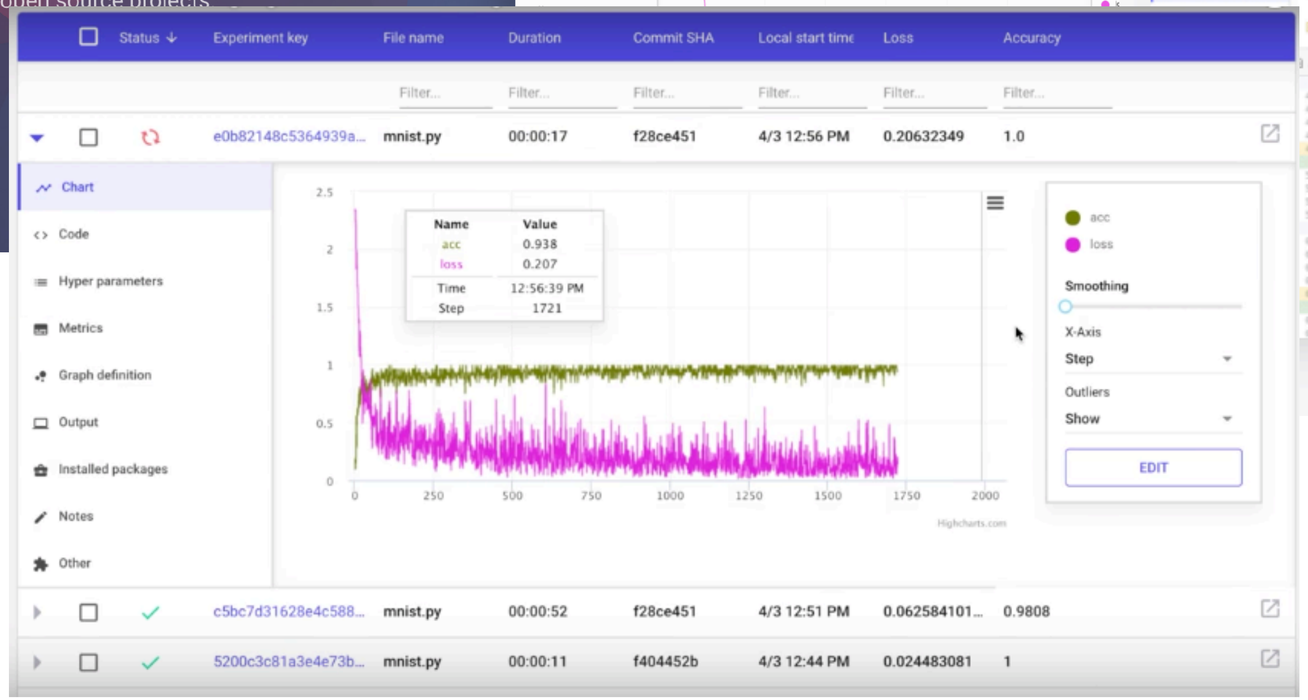


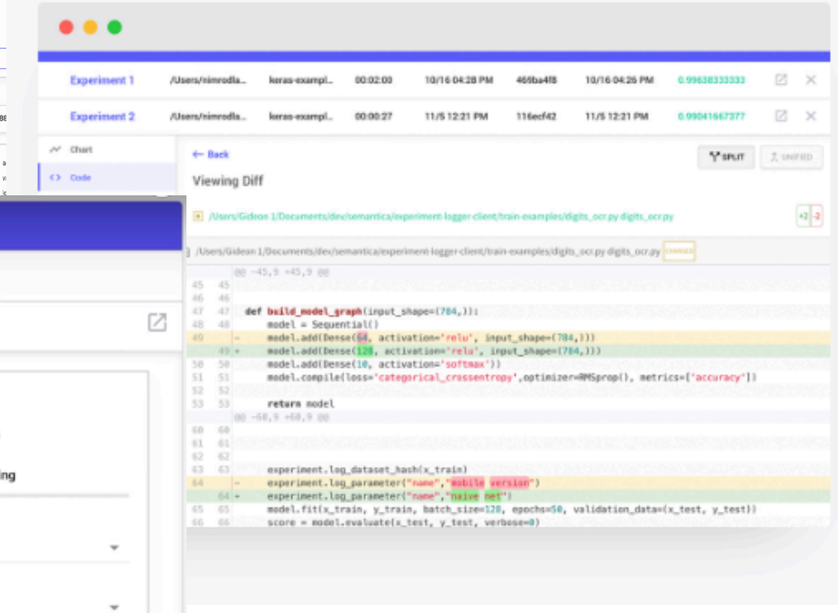


https://towardsdatascience.com/tracking-ml-experiments-using-mlflow-7910197091bb

Comet.ml



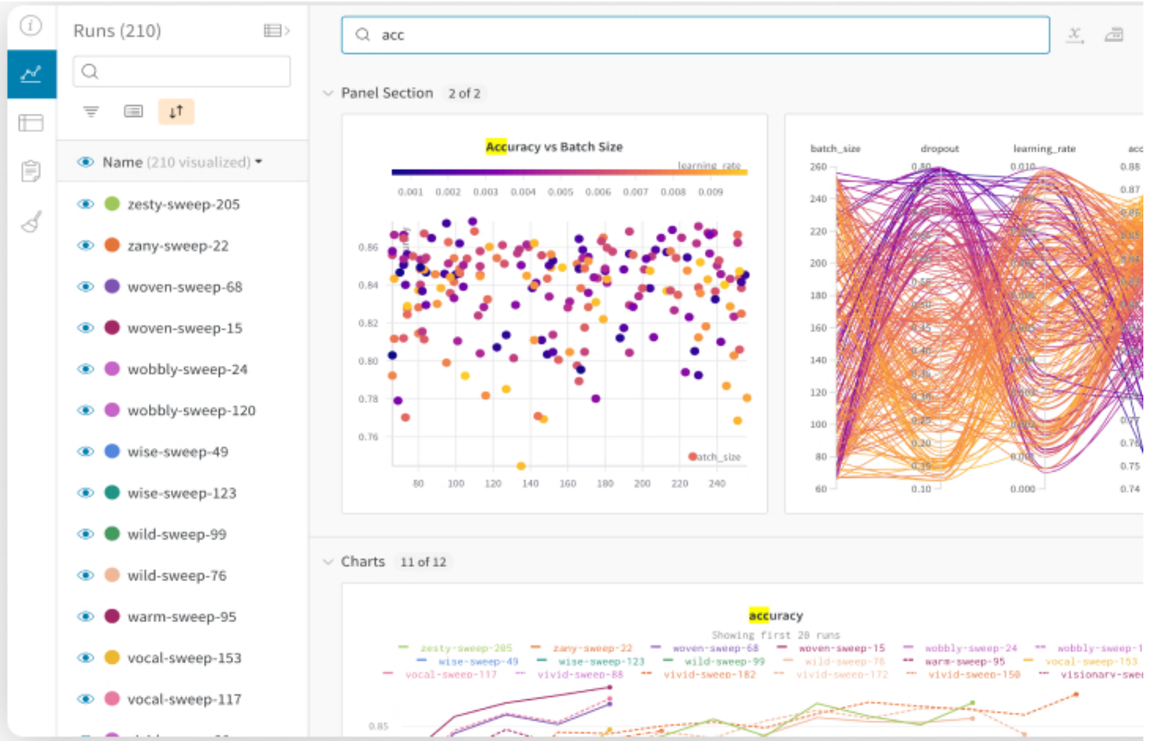


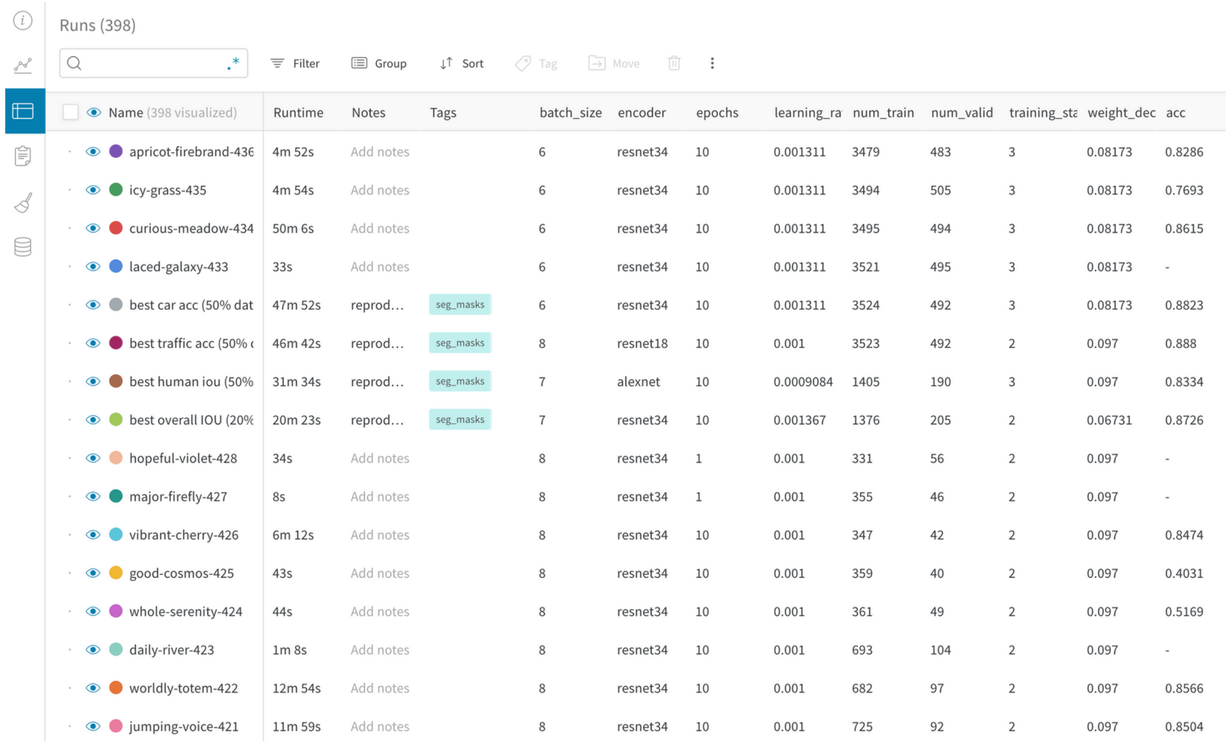


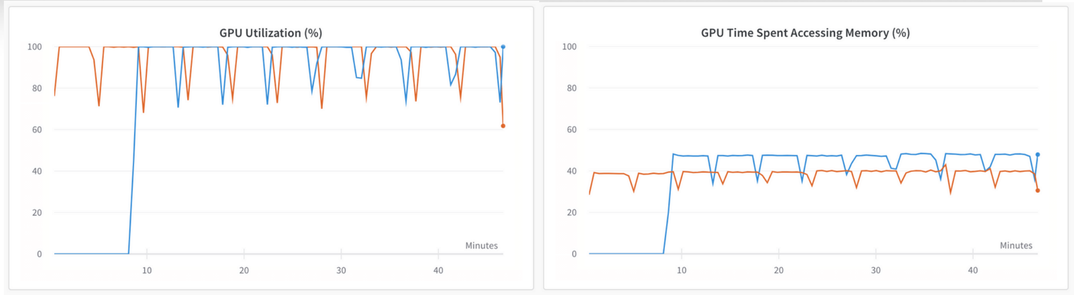
Compare Experiments

Comet lets you compare different experiments and see the differences in code, hyper-params, and many other data points.

Weights & Biases







- What we use in lab

The View from the Driver's Seat

Semantic segmentation for scene parsing on Berkeley Deep Drive 100K

Stacey Svetlichnaya

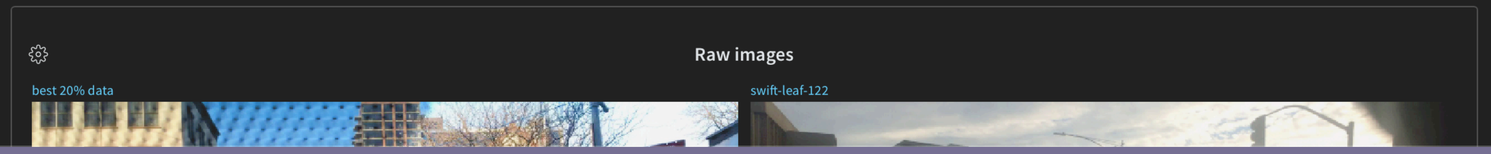
0 10

Understand a dashboard scene with semantic segmentation

A self-driving car must functionally understand the road and its environment the way a human would from the driver's seat. One promising computer vision approach is **semantic segmentation**: parse visual scenes from a car dashboard camera into relevant objects (cars, pedestrians, traffic signs), foreground (road, sidewalk), and background (sky, building). Semantic segmentation annotates an image with object types, labeling meaningful subregions as a tree, bus, cyclist, etc. For a given car dashboard photo, this means labeling *every pixel* as belonging to a subregion.

Below you can see examples in two columns: raw images, the model's predictions, and the ground truth (correct labeling). Buildings are orange, cars are pink, road is cobalt blue, and pedestrians are beige. In the left column, the model can't differentiate between a pedestrian and a rider on a bicycle (magenta and cyan in ground truth, beige in prediction). Note how the hazy conditions in the right column make the model predictions blurry around the boundaries between dashboard and road, or vehicle and road).

Example segmentation maps



Weights & Biases

- Publish reports with embedded charts, figures, etc

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

"All-in-one"

Data

- Versioning: DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Apache Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, LOG, Parquet

Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- Hyperparameter Tuning: SIGOPT, Determined AI, Weights & Biases, tune
- Experiment Management: Weights & Biases, comet, TensorBoard, Neptune Machine Learning Lab, mlflow
- Resource Management: Determined AI, PY, slurm workload manager
- Software Engineering: jupyter, Streamlit, git
- Compute: CoreWeave, Lambda

Deployment

- Feature Store: tecton
- Monitoring: fiddler
- Edge: ML, OctoML, NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations

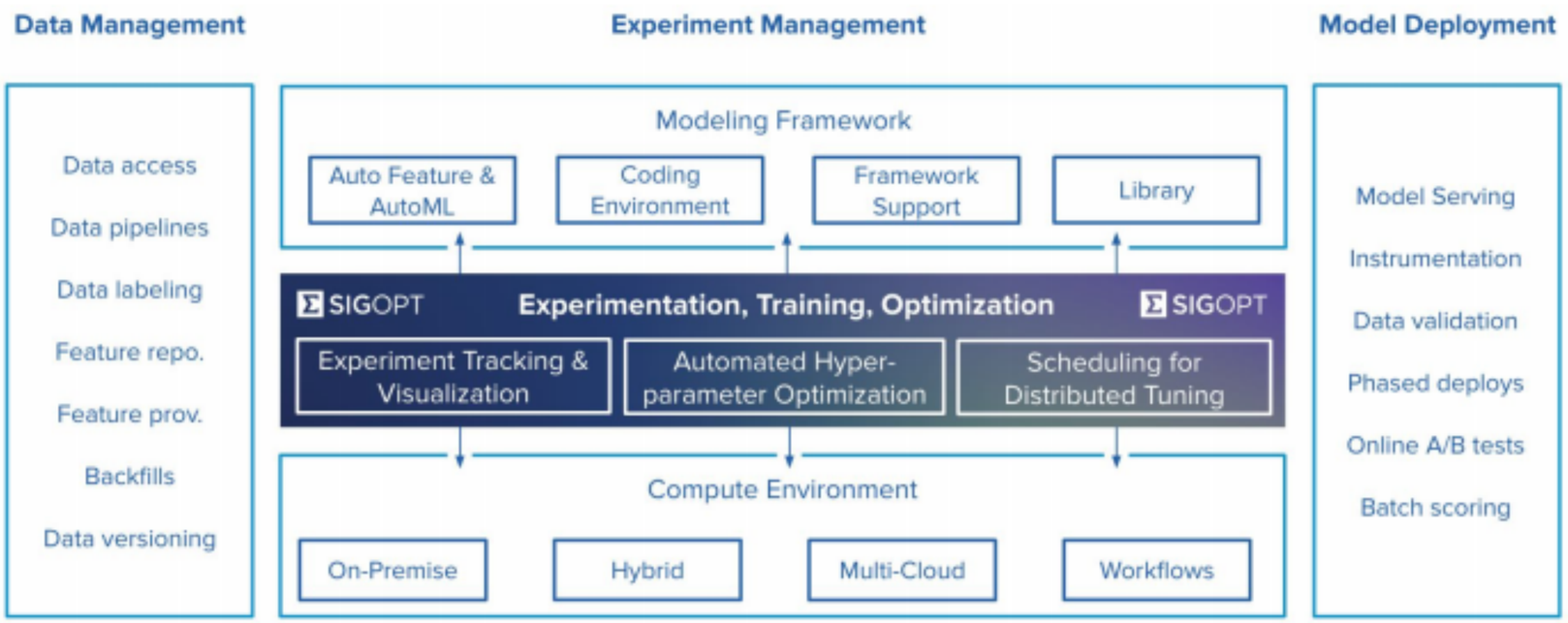
Hyperparameter Optimization

- Useful to have software that helps you search over hyper parameter settings.
- Could be as simple as being able to provide `—lr=(0.0001, 0.1) --num_layers=[128, 256, 512]` to training script
- Would be even better if settings were selected intelligently, and underperforming runs stopped early.

SIGOPT

Improve ML models 100x faster

SigOpt's API tunes your model's parameters through *state-of-the-art* Bayesian optimization.



1 Provide parameters

Ping our API with your model's parameters. We don't need the model itself. You keep it private.

2 Use our values

Our API suggests new values for these parameters. Use them to evaluate your model within your current infrastructure.

3 Send model output

We use your model's output to calculate the next best configuration.

4 Repeat until optimized

You'll reach optimal values up to 100x faster than other methods.

Ray Tune

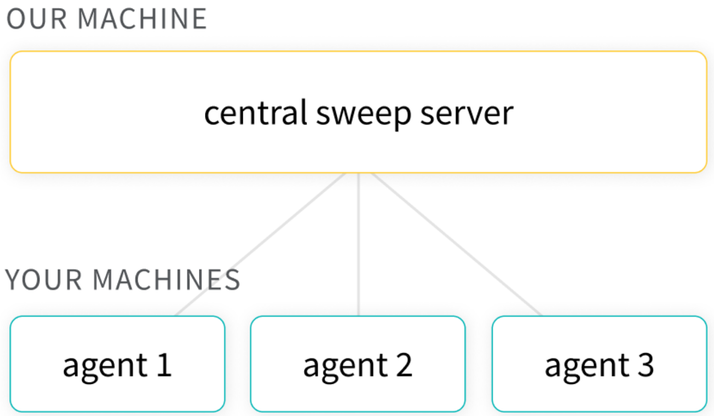
- "Choose among scalable SOTA algorithms such as Population Based Training (PBT), Vizier's Median Stopping Rule, HyperBand/ASHA."

- These redirect compute resources toward promising areas of search space

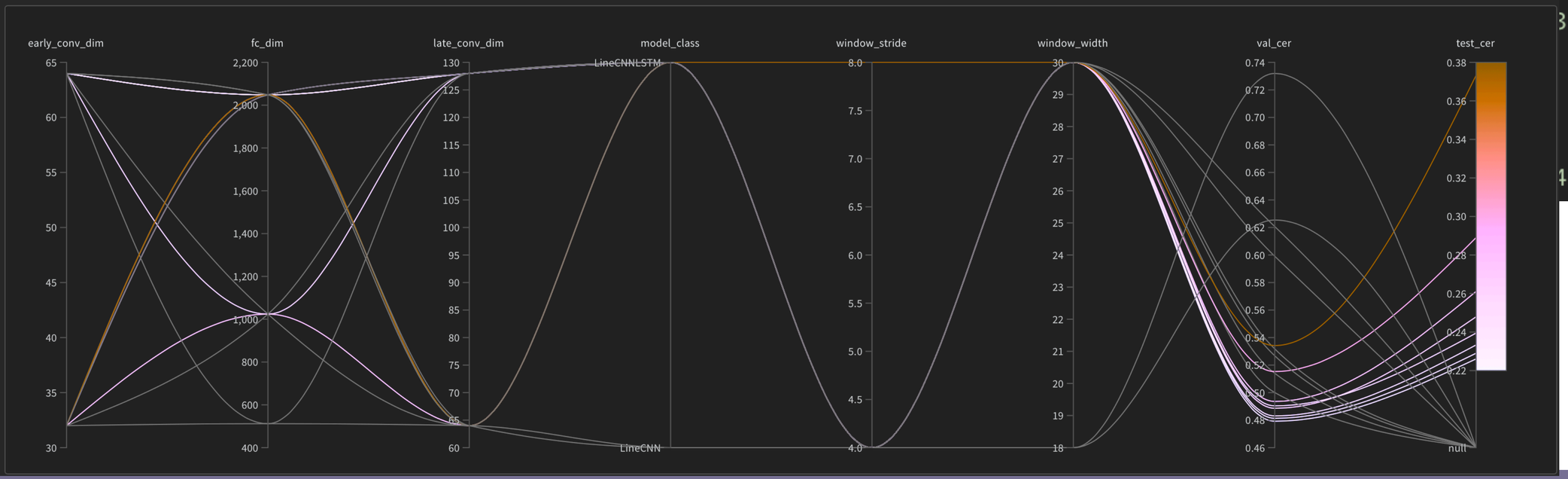
```python
import torch.optim as optim
from ray import tune
from ray.tune.examples.mnist_pytorch import get_data_loaders, ConvNet, train, test


def train_mnist(config):
    train_loader, test_loader = get_data_loaders()
    model = ConvNet()
    optimizer = optim.SGD(model.parameters(), lr=config["lr"])
    for i in range(10):
        train(model, optimizer, train_loader)
        acc = test(model, test_loader)
        tune.track.log(mean_accuracy=acc)


analysis = tune.run(
    train_mnist, config={"lr": tune.grid_search([0.001, 0.01, 0.1])})

print("Best config: ", analysis.get_best_config(metric="mean_accuracy"))

# Get a dataframe for analyzing trial results.
df = analysis.dataframe()
```

Weights & Biases

- What we use in Lab




```
8   program: training/run_experiment.py
9   method: bayes
10  metric:
11    goal: minimize
12    name: val_cer
13  early_terminate:
14    type: hyperband
15    min_iter: 20
16  parameters:
17    early_conv_dim:
18      values: [32, 64]
19    late_conv_dim:
20      values: [64, 128]
21    window_width:
                                                    30]
                                                    4, 2048]
```



Questions?

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

"All-in-one"

Data

- Versioning: DVC, Pachydern, LIQUIDATA
- Labeling: figure eight, scale, Aquarium
- Processing: Spark, DAGSTER
- Exploration: pandas, RAPIDS, dbt
- Data Lake / Warehouse: snowflake, databricks
- Sources: S3, Parquet

Training/Evaluation

- Frameworks & Distributed Training: fast.ai, PYTORCH, RAY, HOROVOD, K
- 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, OctoML, NVIDIA TensorRT, TensorFlow Lite, ONNX
- Web: ALGORITHMIA, SELDON
- CI / Testing: Buildkite, great_expectations

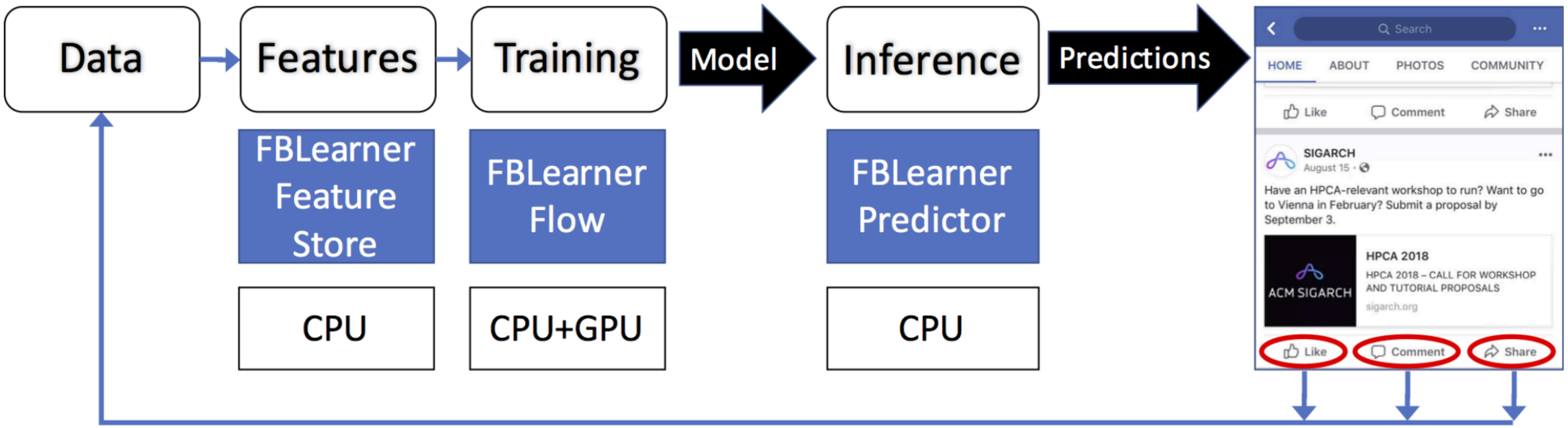
or · or

All-in-one Solutions

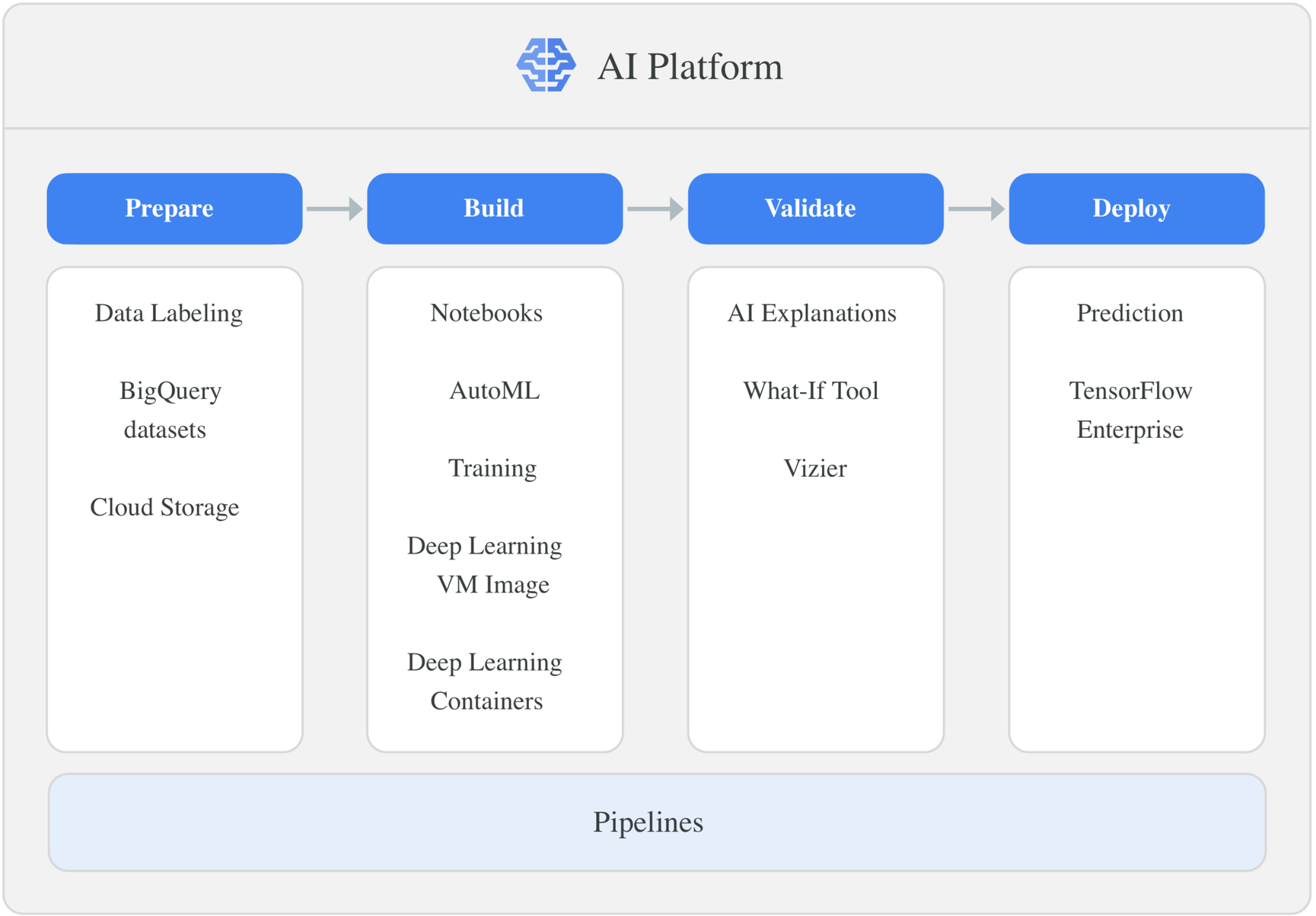
- Single system for everything
  - development (hosted notebook)
  - scaling experiments to many machines (sometimes even provisioning)
  - tracking experiments and versioning models
  - deploying models
  - monitoring performance

POSTED ON MAY 9, 2016 TO AI RESEARCH, APPLIED MACHINE LEARNING, CORE DATA

Introducing FBLearner Flow: Facebook's AI backbone



Google Cloud



Amazon SageMaker

Machine learning for every data scientist and developer

Amazon SageMaker helps data scientists and developers to prepare, build, train, and deploy high-quality machine learning (ML) models quickly by bringing together a broad set of capabilities purpose-built for ML.

Neptune
Machine Learning Lab

Build machine learning models

Run numerous experiments, easily compare them and find the best model.
Monitor the training process using charts instead of just console logs

READ MORE

Use your favorite tools

Write code in your preferred IDE and Jupyter Notebooks.
Neptune integrates with your favorite languages, libraries and frameworks.

READ MORE

TensorFlow™ Keras Caffe PYTORCH jupyter

Reproduce

Seamlessly track your experiments.
Never lose your work and always be able to reproduce your results.

READ MORE

Collaborate

Share your experiments and source code and compare different machine learning models with your teammates to achieve better results.

READ MORE

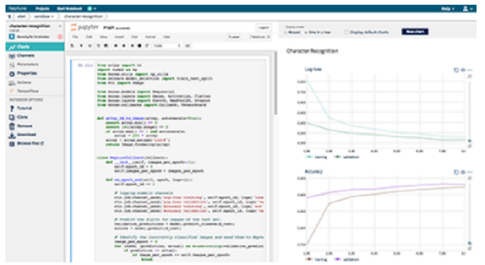
Run on your laptop or entirely in the cloud

Easily run experiments on your laptop or in the cloud - Neptune integrates with public clouds and manages compute environments (Dockers).

READ MORE

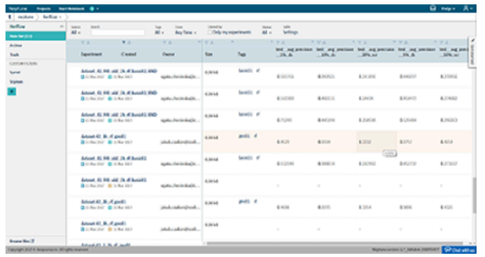
Use Jupyter Notebooks in the cloud

Prototype interactively using Jupyter Notebooks in the cloud. Neptune saves your code and outputs automatically.



Quickly find and compare your best experiments

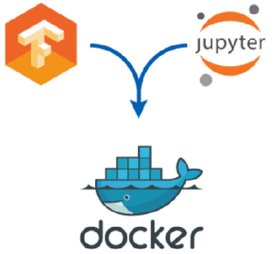
Use your private "Kaggle leaderboard" like a dashboard to filter, explore and sort through your experiments. Find your top machine learning models based on your favorite metric.



Use pre-configured compute environments

Tired of manually configuring your remote machines or installing missing libraries?
Don't waste your time on DevOps work! Use one of Neptune's pre-configured environments (Docker images) and focus on data science instead.

```
$ neptune send --environment tensorflow
$ neptune send --environment keras
$ neptune send --environment theano
```



Send your training processes to the cloud

Move training processes off your desktop and onto powerful machines in the cloud. Experiment more at the same time.

```
$ neptune send train_cnn.py
```

FLOYD

Version Control and Full Reproducibility

All your work is saved and versioned automatically. Your code, environments, data, parameters and results are preserved for exact reproducibility.

Jupyter Notebook

Develop interactively using Jupyter Notebook from y

`floyd run --mode jupyter`

Run your first GPU-powered notebook now

Deploy Trained Models

Preview Deploy your trained models as REST API wit

`floyd run --mode serve`

Set up your model-serving API in one command

Training Metrics

Real-time training metrics for your jobs to help you optimize your code.

Standard GPU

Tesla K80 · 12 GB Memory

61 GB RAM · 100 GB SSD

*Machine cost includes bandwidth and high-performance SSD

High Performance GPU

Tesla V100 · 16 GB Memory

61 GB RAM · 100 GB SSD

*Machine cost includes bandwidth and high-performance SSD

Zero Setup Deep Learning

Fully configured CPU and GPU instances primed for deep learning. Includes CUDA, cuDNN and all popular frameworks. Simply run:

`floyd run --env theano|`

See all environments

Amazing Hardware

- `Amazing Hardware`
  Multiple tiers of CPU and GPU instances, equipped with Nvidia Tesla K80 GPUs, Nvidia Tesla V100 GPUs, or better.
- `Dedicated Machines`
  FloydHub machines are not spot instances. With guaranteed SLAs you will never lose your training results again.
- `Per Second Billing`
  Forget about expensive hourly rates. Pay only for what you use, per second.

View pricing

Iterate in Parallel

Evaluating multiple models? Parameter sweeping? Scale parallel on the cloud. Run concurrent jobs and FloydHub schedules, manages and tracks

gradient
by Paperspace

DEVELOP, TRAIN, & DEPLOY

Notebooks
1-Click Jupyter Notebooks

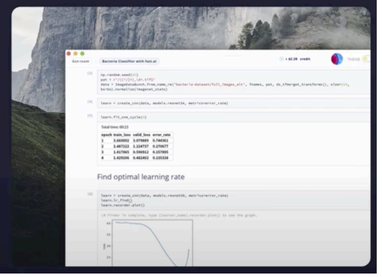
Experiments
Develop, train, and track results

Datasets
Connect, version, and track data

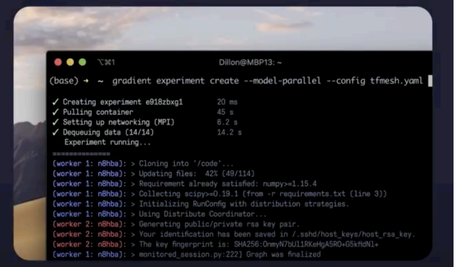
Models
Store, analyze and version models

Inference
Deploy models as API endpoints

01
Develop
1-click Jupyter Notebooks with free GPUs and zero management



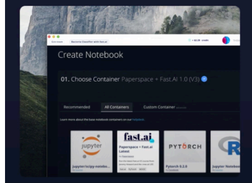
02
Train
Run single experiments to large-scale distributed training



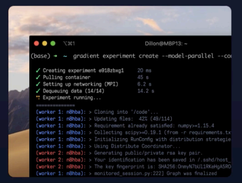
Run anywhere
Gradient runs in the public cloud or on your private servers

Azure aws Google Cloud Platform NVIDIA DGX

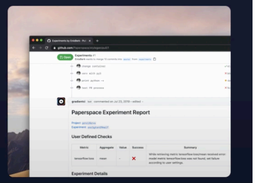
01
Interface



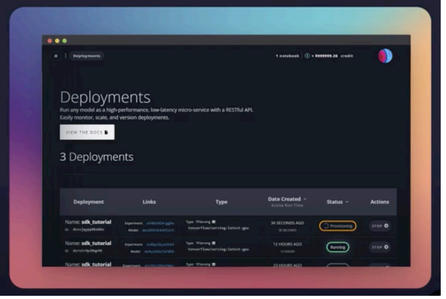
02
CLI



03
GitHub



03
Deploy
Model Serving with scalable API endpoints

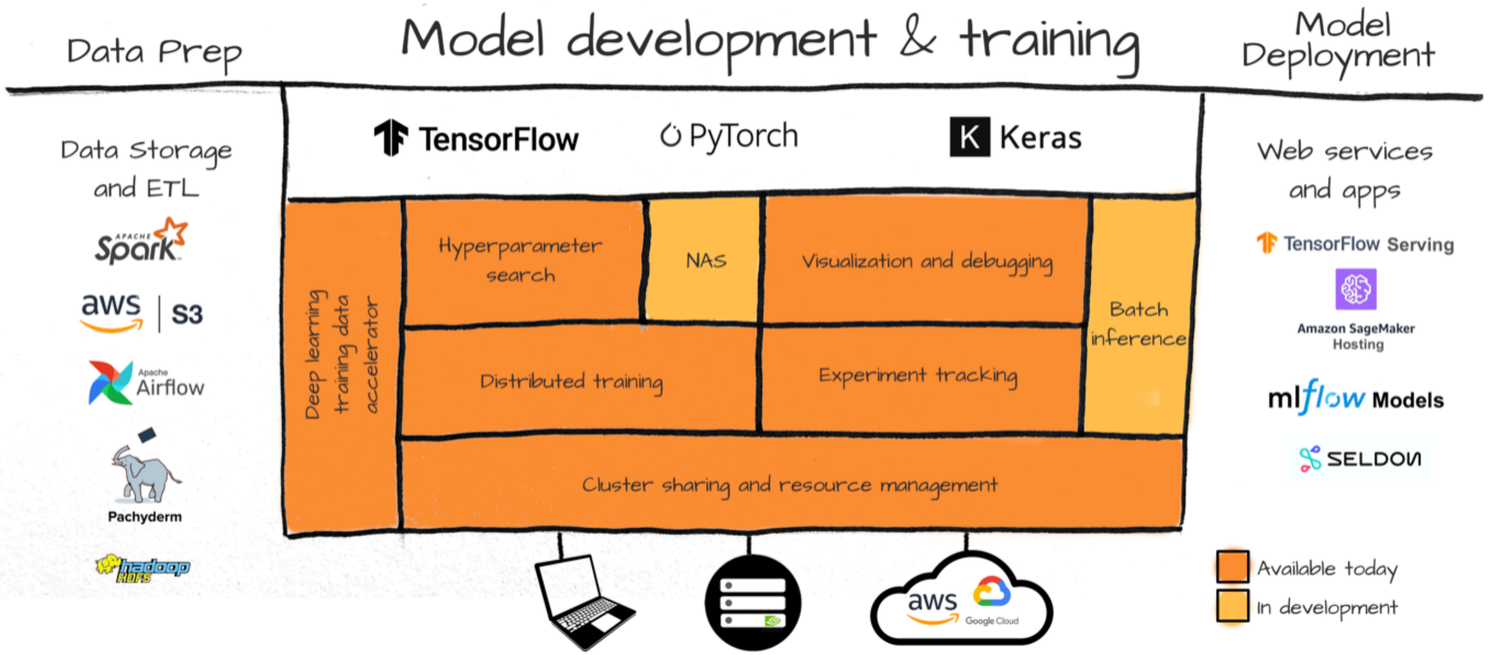


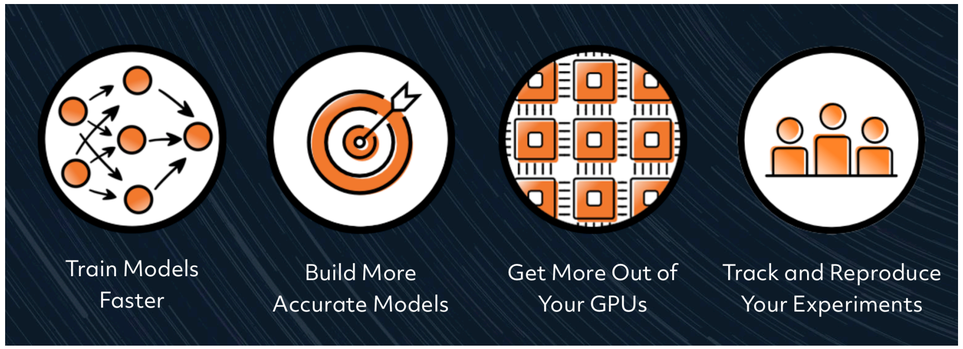
04
Scale
Advanced Features & Pipelines

Auto-scaling **Monitoring**

Versioning **Continuous**

Determined AI



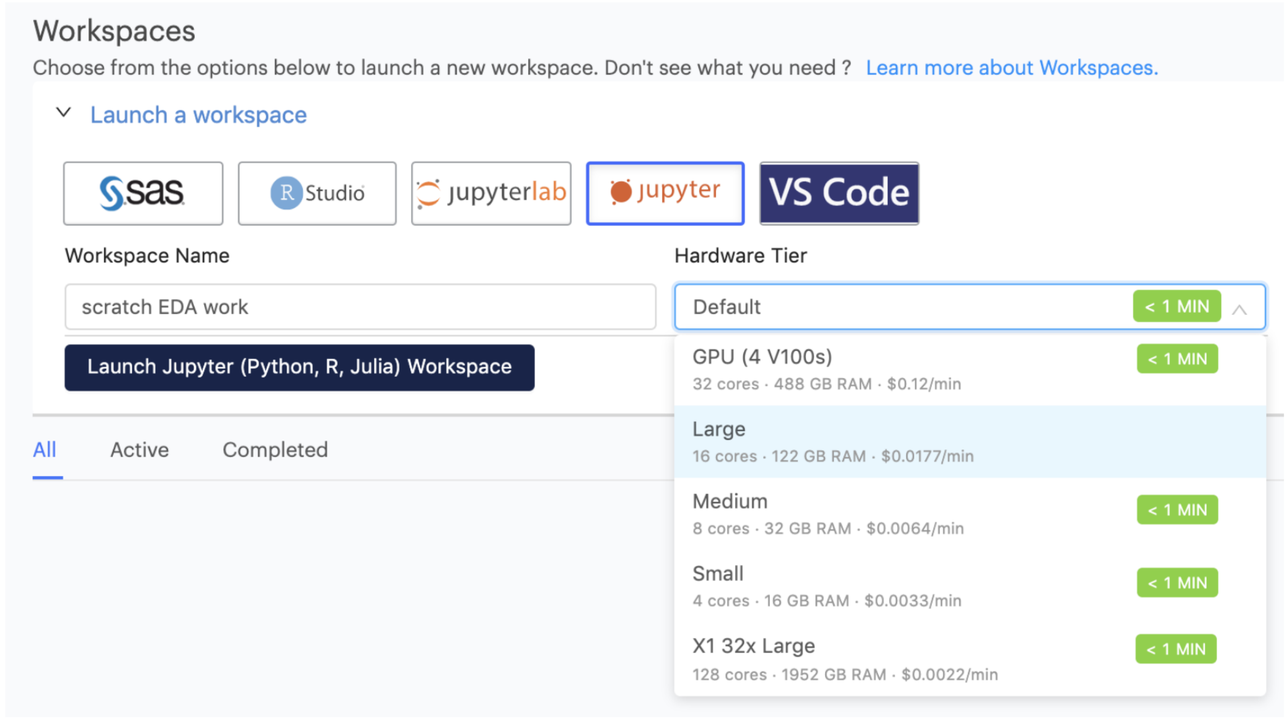


- Open source!
- In-house distributed training module
- HyperBand based hyperparameter tuning
- Smart GPU scheduling, including spot instances
- Experiment tracking

Domino Data Lab

One place for your data science tools, apps, results, models, and knowledge.

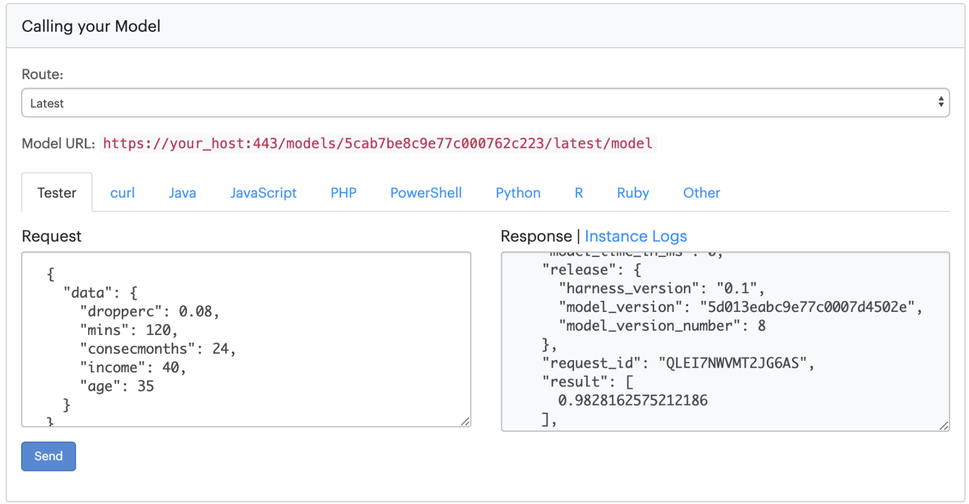
Provision compute



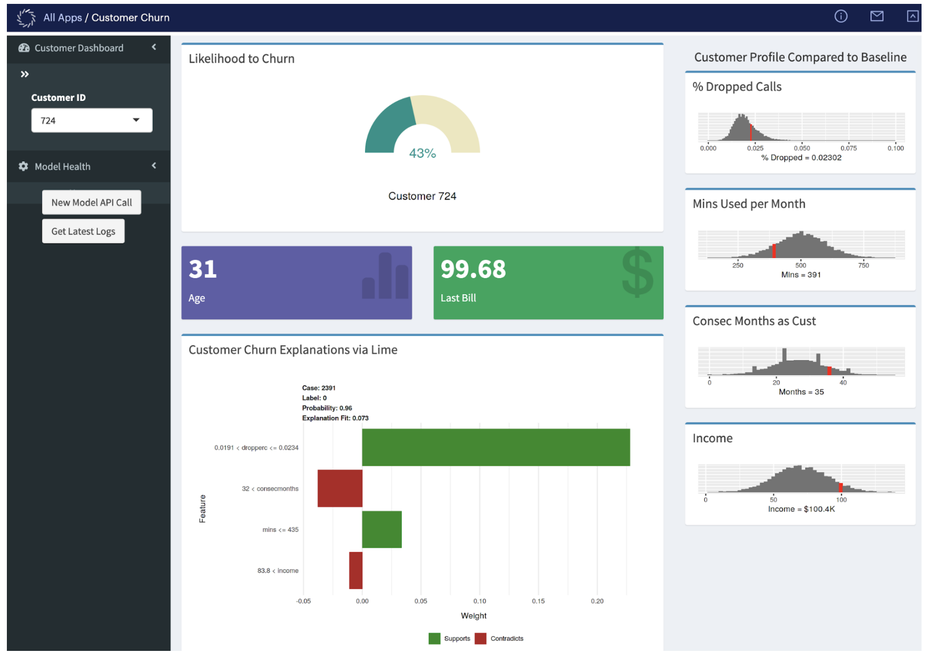
Track experiments

Domino Data Lab

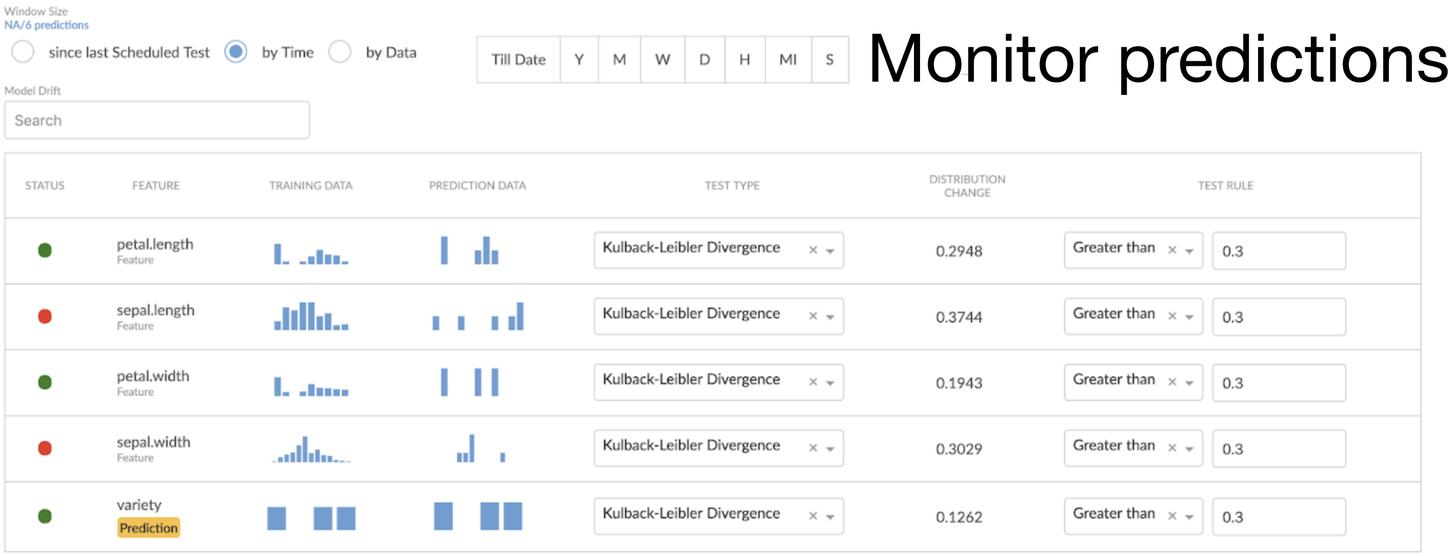
Deploy REST API



Publish applets

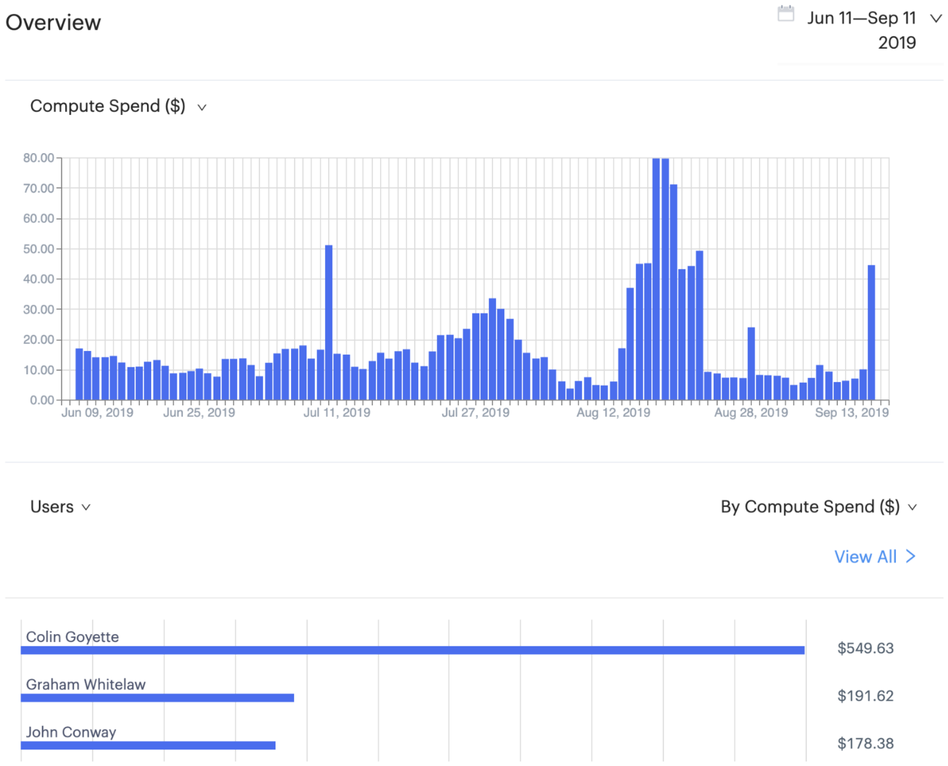


Monitor predictions

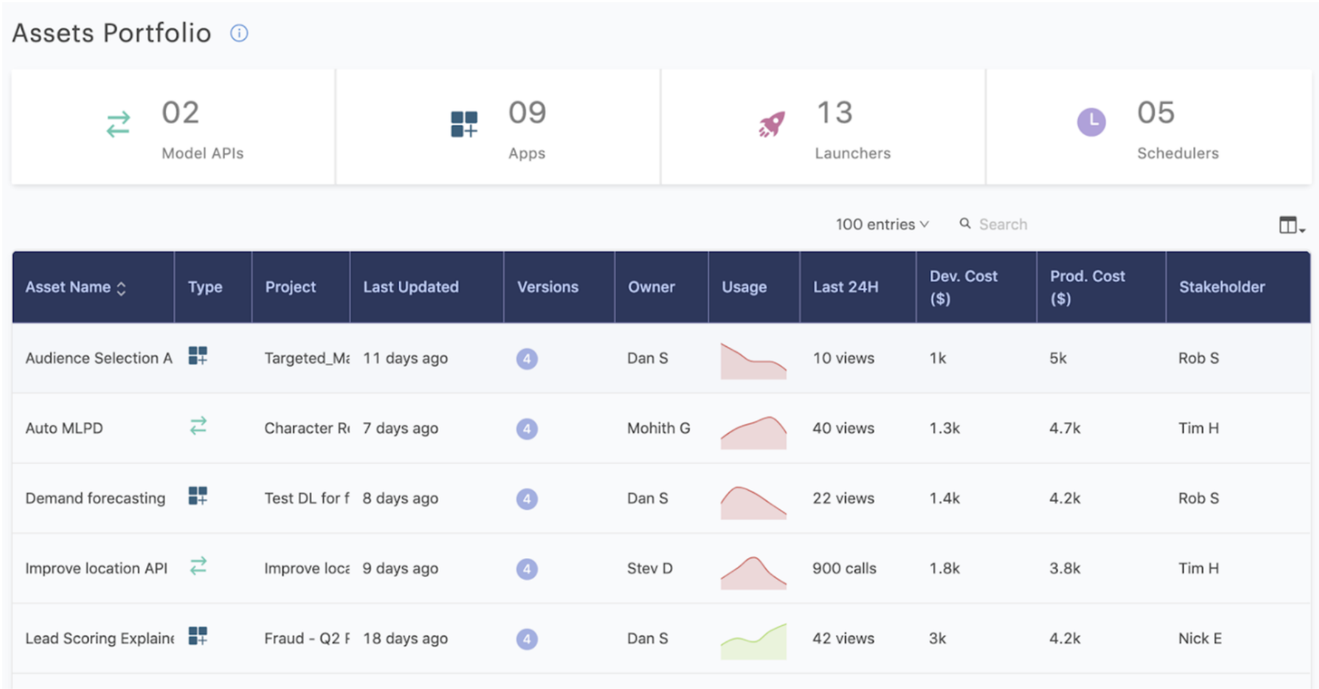


Domino Data Lab

Monitor spend



All projects in one place



Natural place to go for most MLOps companies

Weights & Biases

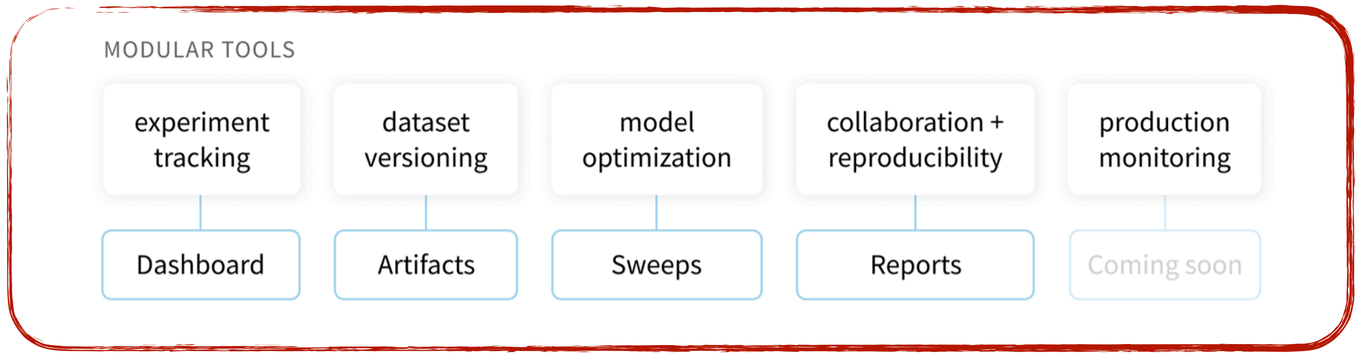
Track machine learning experiments, visualize metrics, and share results

---

Weights & Biases helps you keep track of your machine learning projects. Use our tools to log hyperparameters and output metrics from your runs, then visualize and compare results and quickly share findings with your colleagues.

Here is a quick overview of our lightweight, modular tools:

1. **Dashboard**: Track experiments, visualize results
2. **Reports**: Save and share reproducible findings
3. **Artifacts**: Dataset and model versioning, pipeline tracking
4. **Sweeps**: Optimize models with hyperparameter tuning



	W&B	Paperspace Gradient	Floyd	Determined.ai	Domino Data Lab	Amazon SageMaker	GC ML Engine
Hardware	N/A	Paperspace	GCP	Agnostic	Agnostic	AWS	GCP
Resource Management	No	Yes	Yes	Yes	Yes	Yes	Yes
Hyperparam Optimization	Yes	No	No	Yes	Yes	Yes	Yes
Storing Models	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reviewing Experiments	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Deploying Models as REST API	No	No	Yes	No	Yes	Yes	Yes
Monitoring	No	No	No	No	Yes	Yes	Yes

Questions?

Tooling Tuesdays

https://twitter.com/full_stack_dl

Thank you!