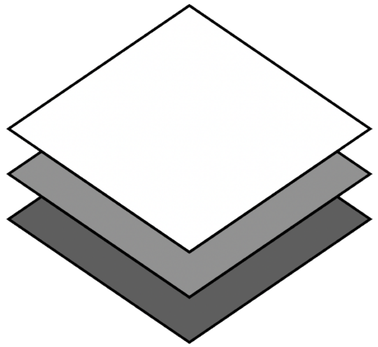
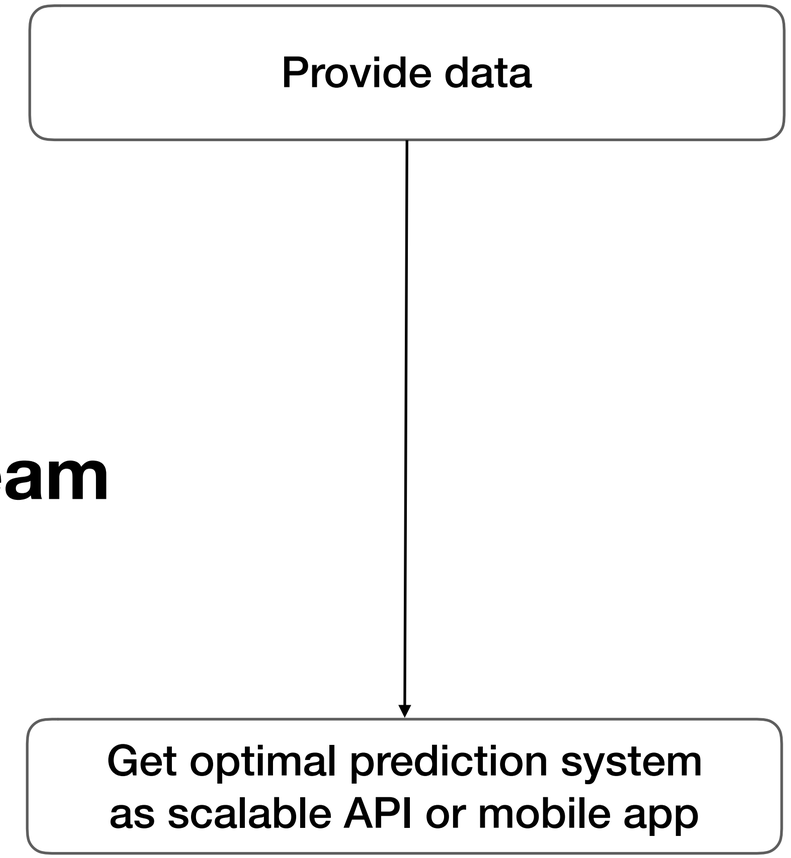


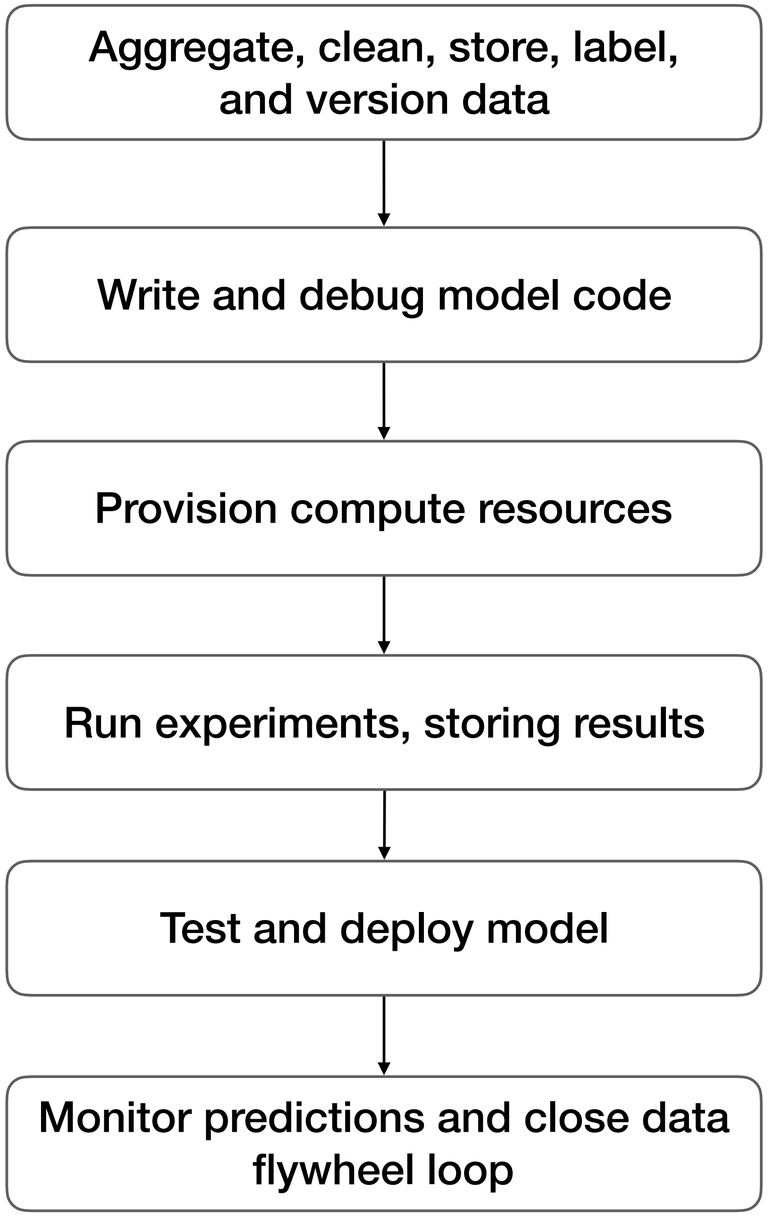
Infrastructure & Tooling

Josh Tobin, **Sergey Karayev**, Pieter Abbeel

Dream



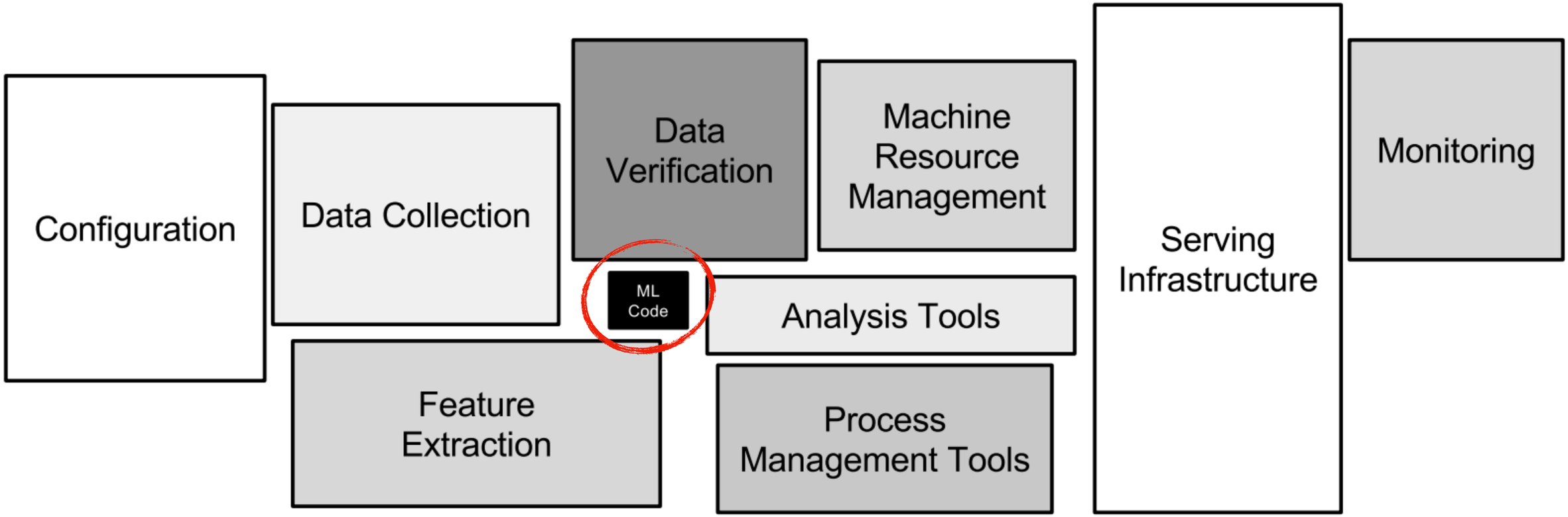
Reality



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

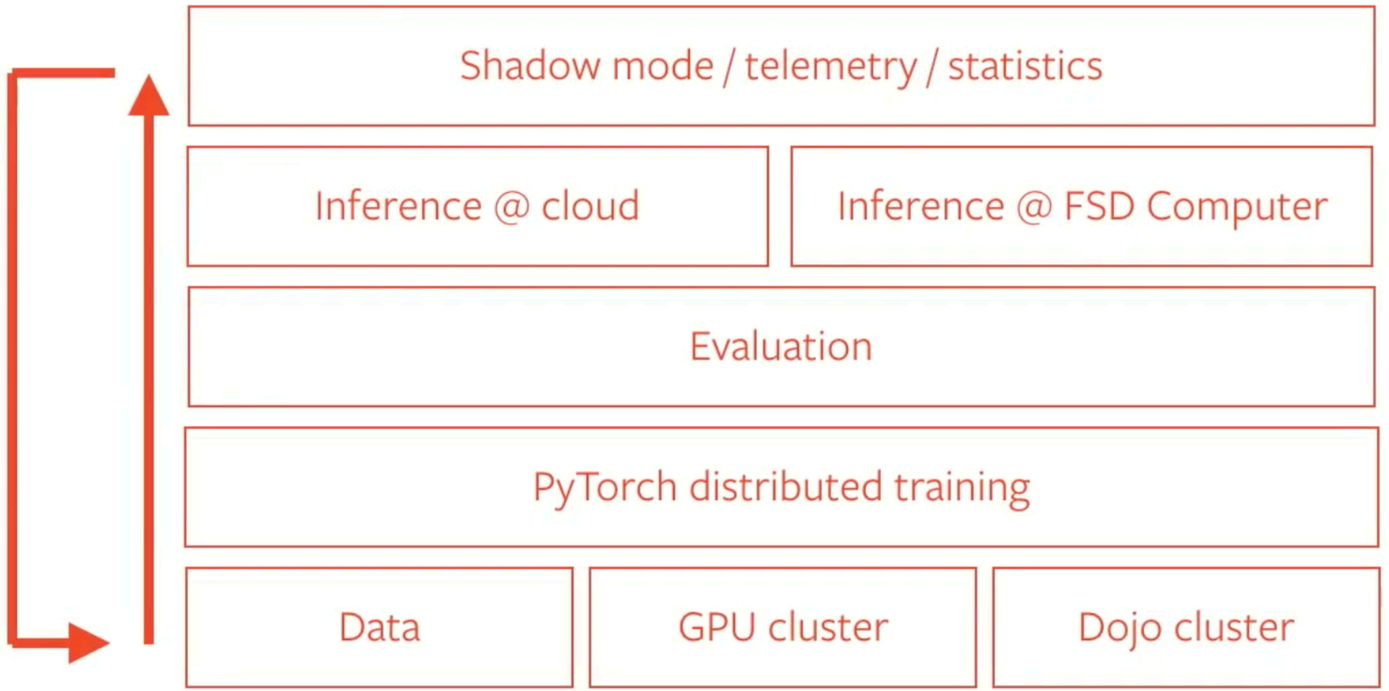
SE4ML: Software Engineering for Machine Learning (NIPS 2014



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

"OPERATION VACATION"



Goal: add data, see model improve

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

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

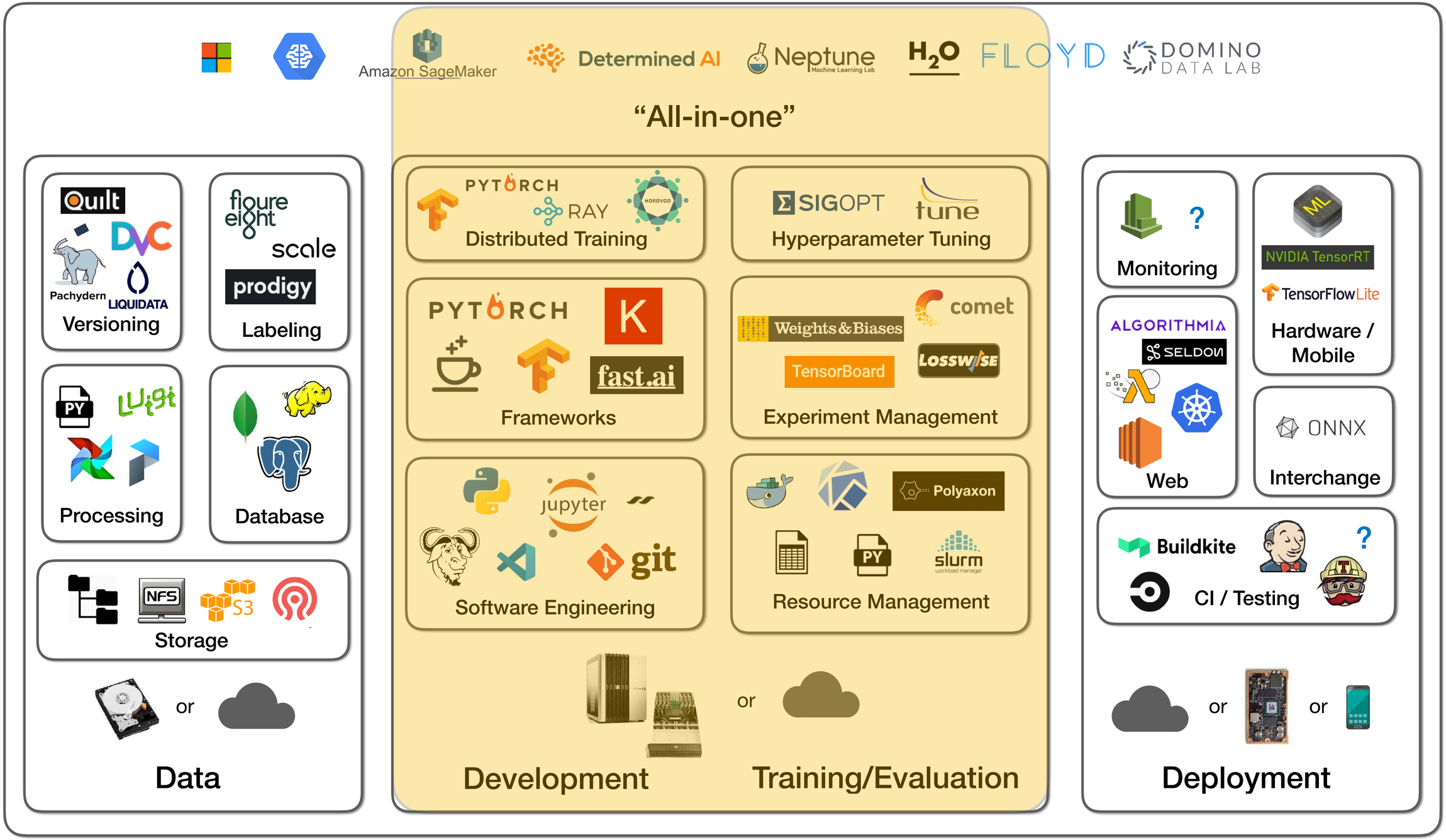
Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Interchange: ONNX

CI / Testing: Buildkite, ?

or or


Amazon SageMaker
Determined AI
Neptune
Machine Learning Lab
H2O
FLOYD
DOMINO DATA LAB
"All-in-one"
Quilt
DVC
Pachyderm
LIQUIDATA
Versioning
figure eight
scale
prodigy
Labeling
PY
Processing
Database
NFS
S3
Storage
or
Data
PYTORCH
RAY
HOROVOD
Distributed Training
SIGOPT
tune
Hyperparameter Tuning
PYTORCH
K
fast.ai
Frameworks
Weights & Biases
comet
TensorBoard
LOSSWISE
Experiment Management
jupyter
git
Software Engineering
Polyaxon
PY
slurm
workload manager
Resource Management
Development
or
Training/Evaluation
Monitoring
ML
NVIDIA TensorRT
TensorFlowLite
Hardware / Mobile
ALGORITHMIA
SELDON
Web
ONNX
Interchange
Buildkite
CI / Testing
or
or
Deployment

Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Interchange: ONNX

CI / Testing: Buildkite, ?

or

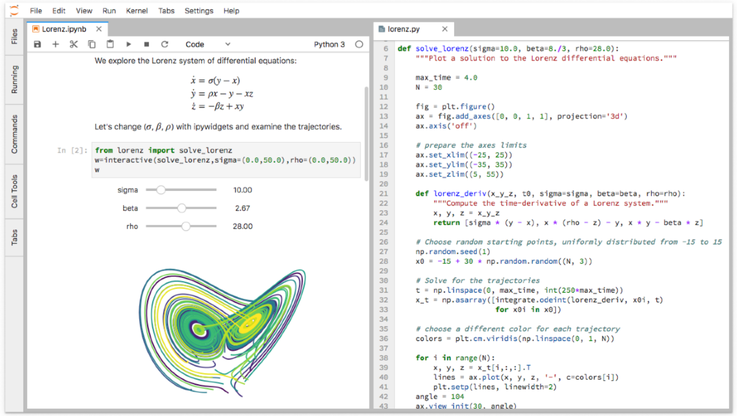
or

Programming Language

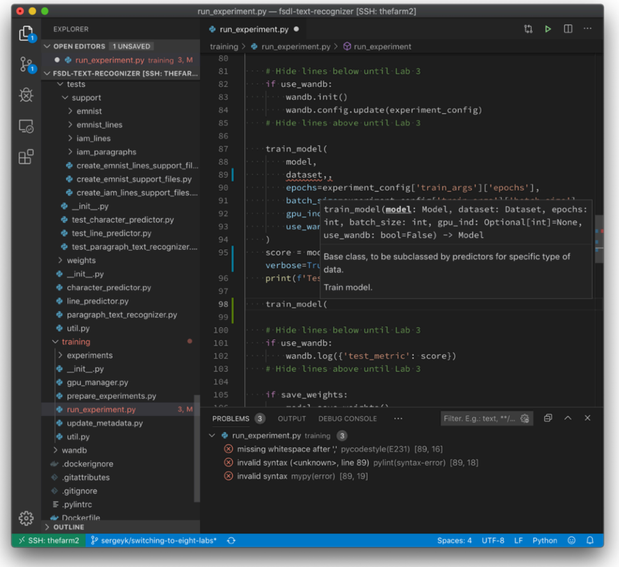
- Python, because of the libraries
  - Clear winner in scientific and data computing

Editors

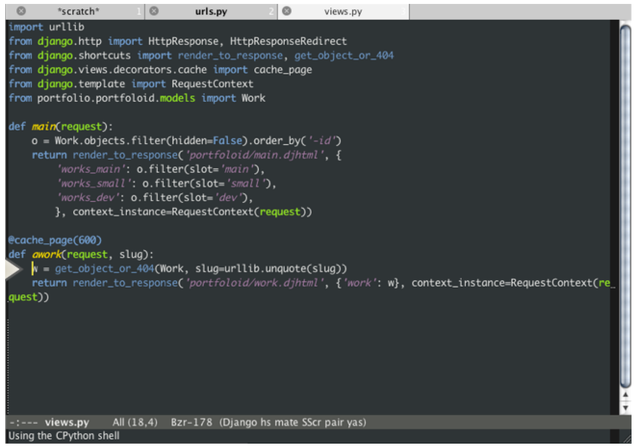
Vim



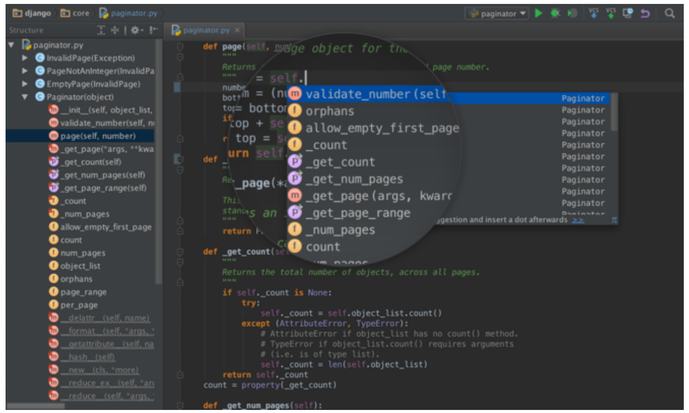
Jupyter



VS Code



Emacs



PyCharm

Visual Studio Code

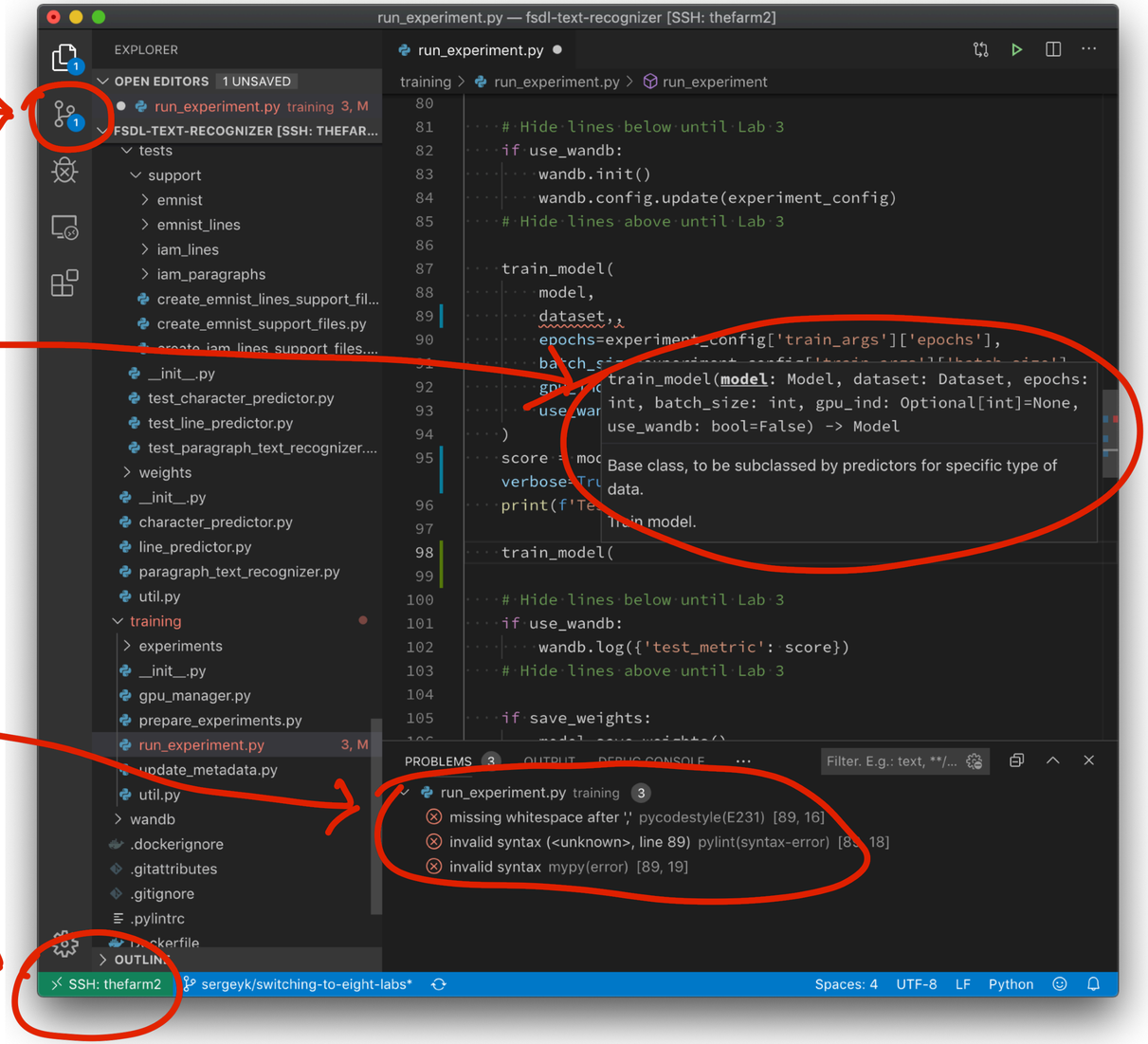
Built-in git staging and diffing

Peek documentation

- VS Code makes for a very nice Python experience

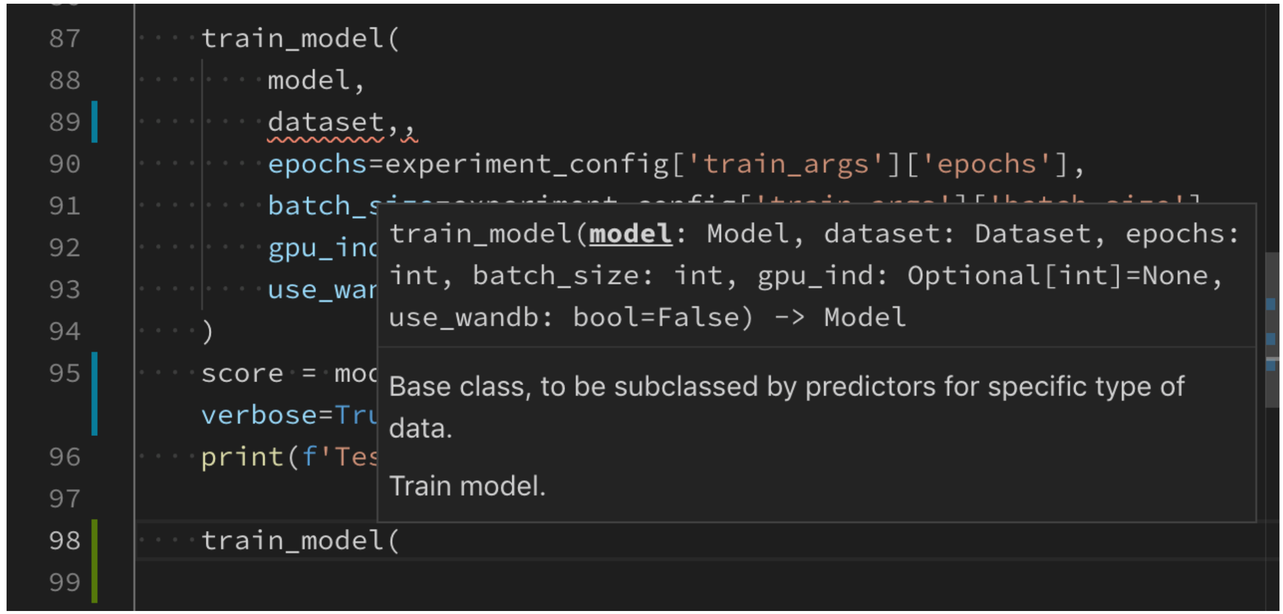
Lint code as your write

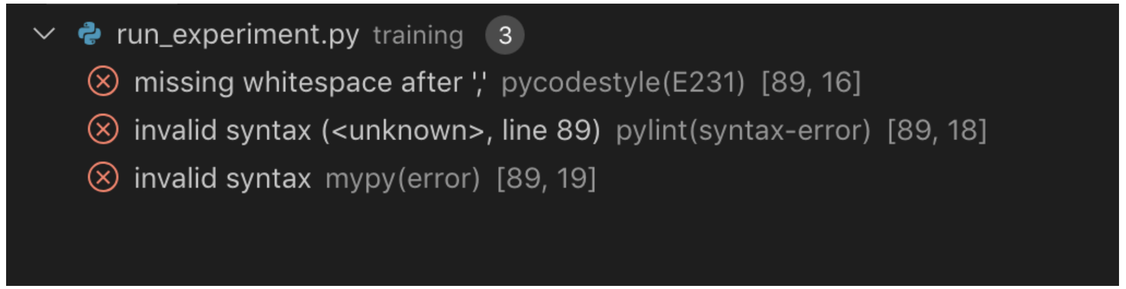
Open whole projects remotely



Linters and Type Hints

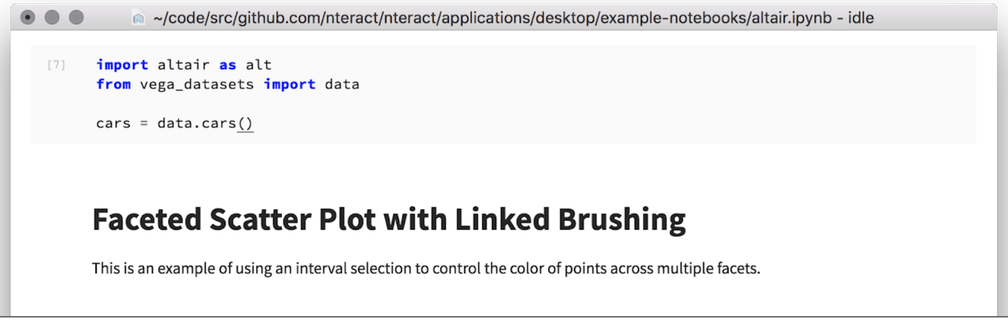
- Whatever code style rules can be codified, should be
- Static analysis can catch some bugs
- Static type checking both documents code and catches bugs
- Will see in Lab 7

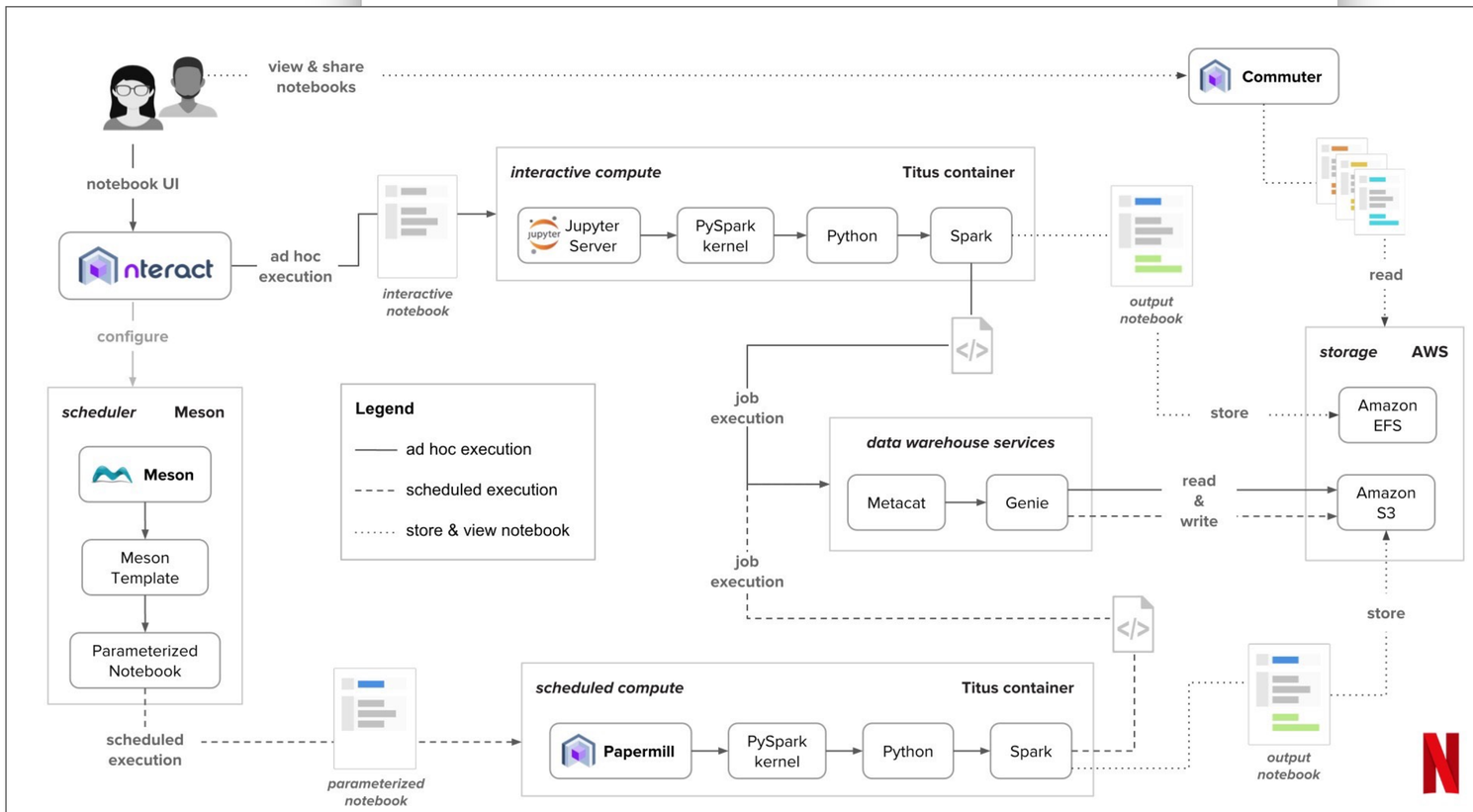




Jupyter Notebooks

- 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
  - Counter-point: Netflix based all ML workflows on them





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

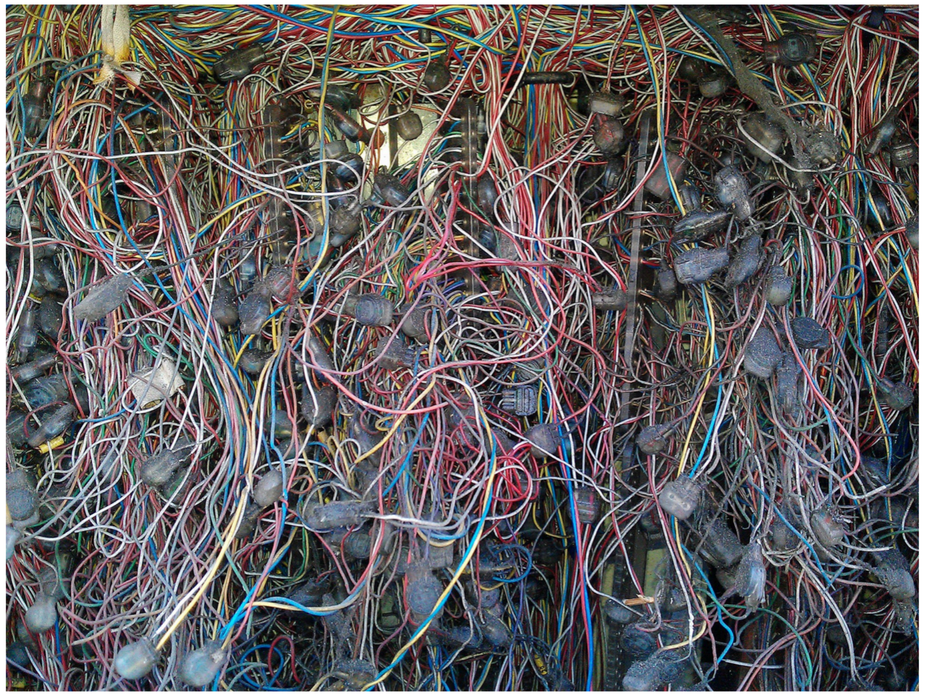
Problems with notebooks

- 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 ★

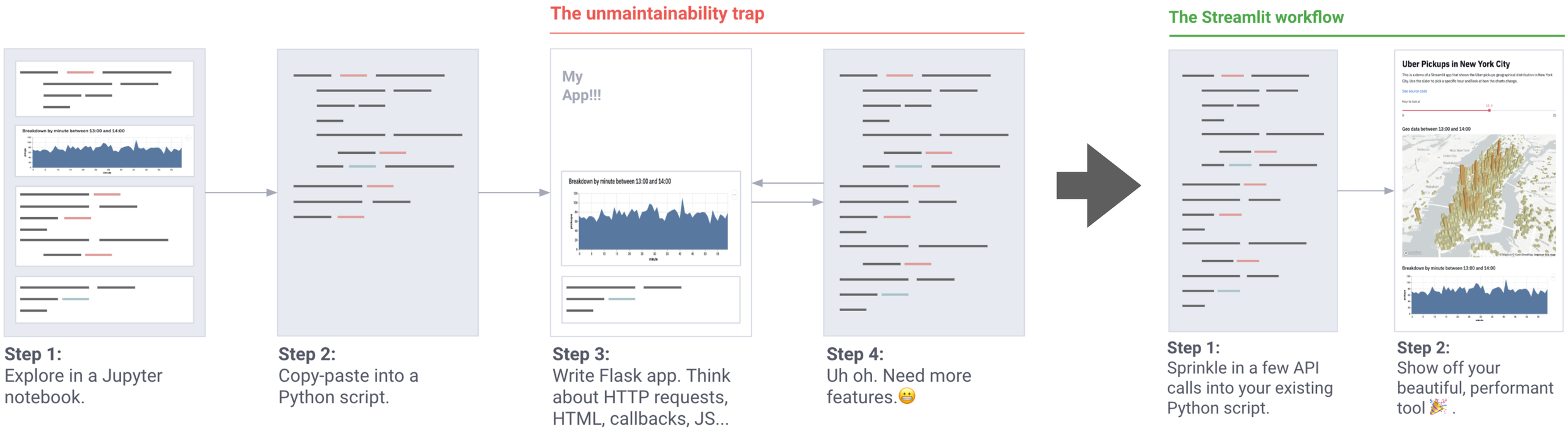


How it feels like managing jupyter notebooks (Complexity © https://www.flickr.com/photos/bitterjug/7670055210)

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

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

Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

PyTorch, Ray, Horovod — Distributed Training

PyTorch, K, fast.ai — Frameworks

jupyter, git — Software Engineering

or

Training/Evaluation

SigOpt, tune — Hyperparameter Tuning

Weights & Biases, comet, TensorBoard, Losswise — Experiment Management

Polyaxon, PY, slurm — Resource Management

Deployment

Monitoring

ALGORITHMIA, SELDON — Web

ML, NVIDIA TensorRT, TensorFlow Lite — Hardware / Mobile

ONNX — Interchange

Buildkite, CI / Testing

or

or

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

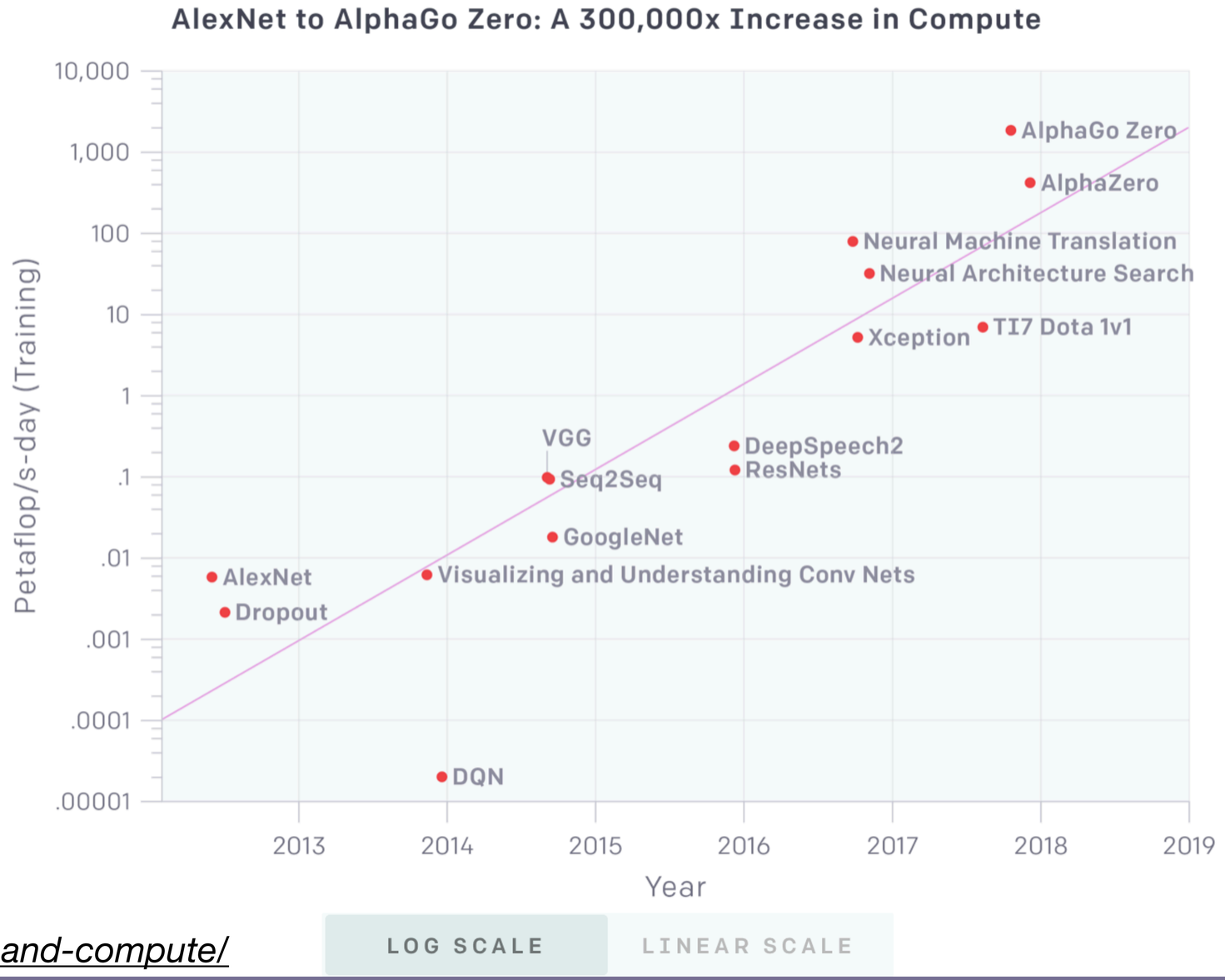

or


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/

But creativity matters, too!

Appendix: Recent novel results that used modest amounts of compute

- Attention is all you need: 0.089 pfs-days (6/2017)
- Adam Optimizer: less than 0.0007 pfs-days (12/2014)
- Learning to Align and Translate: 0.018 pfs-days (09/2014)
- GANs: less than 0.006 pfs-days (6/2014)
- Word2Vec: less than 0.00045 pfs-days (10/2013)
- Variational Auto Encoders: less than 0.0000055 pfs-days (12/2013)

Training Imagenet in 3 hours for $25; and CIFAR10 for $0.26

Written: 30 Apr 2018 by *Jeremy Howard*

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

https://www.fast.ai/2018/04/30/dawnbench-fastai/

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 current option (on GCP)
- Intel Nervana NNPs and AMD could start making sense in the future

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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- New NVIDIA architecture every year
  - Kepler —> Maxwell —> Pascal —> Volta -> Turing
- 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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- **RAM**: should fit meaningful batches of your model
  - Most important for recurrent 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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- **RAM**: should fit meaningful batches of your model
  - Most important for recurrent models
- **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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- **RAM**: should fit meaningful batches of your model
  - Most important for recurrent models
- **32bit vs Tensor TFlops**
  - Tensor Cores are specifically for deep learning operations (mixed precision)
  - Good for convolutional/transformer models
- Straight **16bit** is a bit less good but still better than 32bit
  - roughly 2x flops and 1.5x memory

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	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- 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/Volta

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

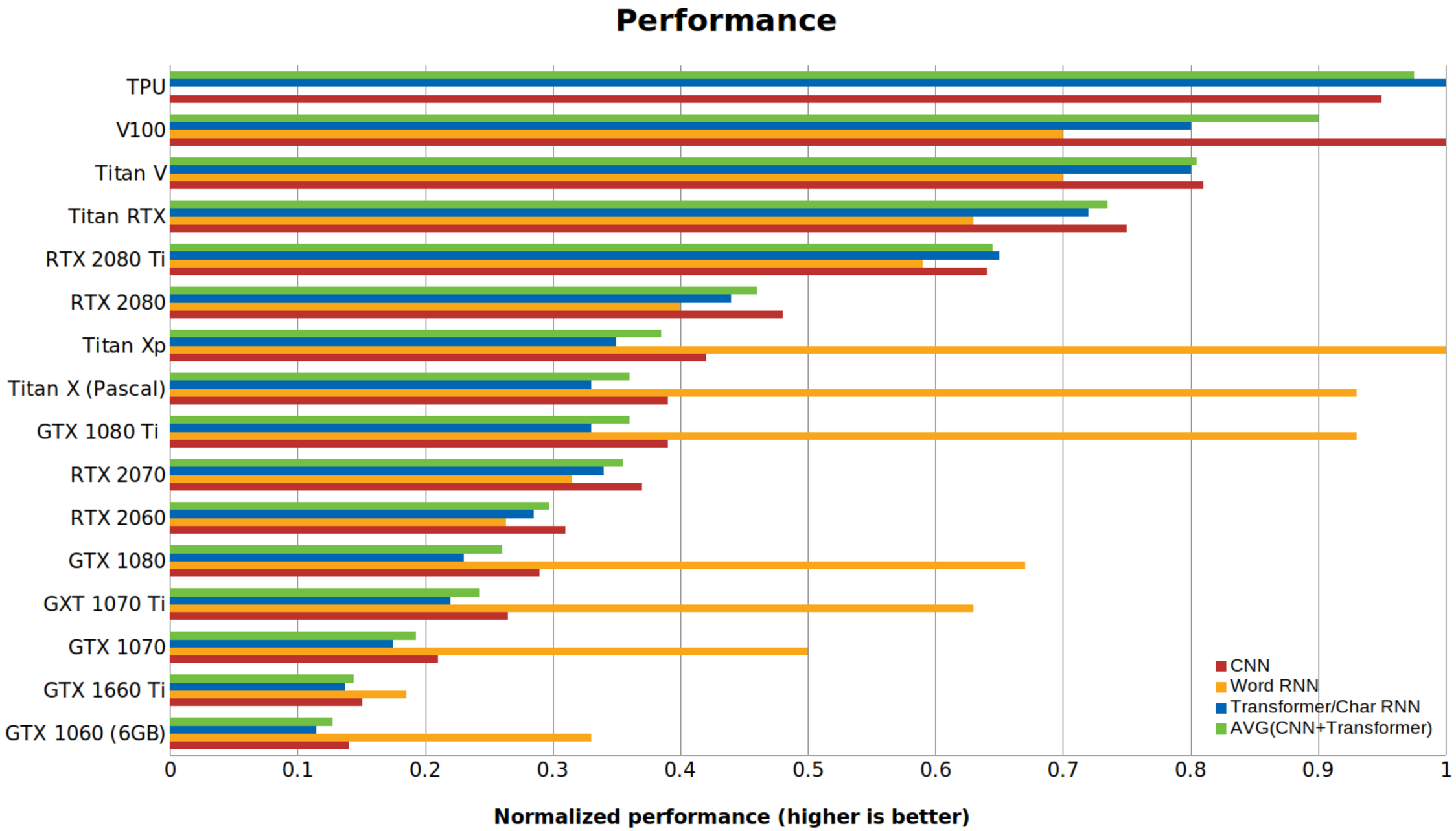
- Hardware: 1080 Ti still good if buying used, especially for recurrent
- Cloud: P100 is a mid-range option

Turing

Card	Release	Arch	Use-case	RAM (Gb)	32bit TFlops	Tensor TFlops	16bit	Cost	Cloud
K80	2014H2	Kepler	Server	24	5	N/A	No	used	AWS, GCP, MS
Titan X	2015H1	Maxwell	Enthusiast	12	6	N/A	No	used	
P100	2016H1	Pascal	Server	16	10	N/A	Yes	used	GCP, MS
1080 Ti	2017H1	Pascal	Consumer	11	13	N/A	No	used	
V100	2017H1	Volta	Server	16	14	120	Yes	$10000	AWS, GCP, MS
Titan V	2017H2	Volta	Enthusiast	12	14	110	Yes	used	
2080 Ti	2018H2	Turing	Consumer	11	13	60	Yes	$1000	
Titan RTX	2018H2	Turing	Enthusiast	24	16	130	Yes	$2500	
RTX 8000	2018H2	Turing	Enthusiast	48	16	160	Yes	$5500	

- 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/*



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

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 lets you connect GPUs to any instance, and has TPUs.
  - Azure reportedly has bad user experience
- Startups are Paperspace, Lambda Labs

Amazon Web Services

Name	GPU	GPUs	GPU RAM	vCPU	RAM	On-demand	Spot
p3.2xlarge	V100	1	16	8	61	$3.06	$1.05
p3.8xlarge	V100	4	64	32	244	$12.24	$4.20
p3.16xlarge	V100	8	128	64	488	$24.48	$9.64
p2.xlarge	K80	1 die	12	4	61	$0.90	$0.43
p2.8xlarge	K80	8 dies	96	32	488	$7.20	$3.40
p2.16xlarge	K80	16 dies	192	64	768	$14.40	$6.80

Amazon Web Services

Name	GPU	GPUs	GPU RAM	vCPU	RAM	On-demand	Spot
p3.2xlarge	V100	1	16	8	61	$3.06	$0.95
p3.8xlarge	V100	4	64	32	244	$12.24	$4.00
p3.16xlarge	V100	8	128	64	488	$24.48	$7.30
p2.xlarge	K80	1 die	12	4	61	$0.90	$0.27
p2.8xlarge	K80	8 dies	96	32	488	$7.20	$2.16
p2.16xlarge	K80	16 dies	192	64	768	$14.40	$4.32

- For a big (50-80%) discount, can get instances that can terminate at any time.
- Makes sense for running hyperparam search experiments, but need infrastructure to handle failures.

Google Cloud Platform

GPU	GPUs	GPU RAM	On-demand	Spot
V100	1-8	16	$2.48	$0.74
P100	1-4	64	$1.46	$0.43
K80	1-8	12	$0.45	$0.14

- GPUs can be attached to any instance.
  - In general, a little cheaper than AWS.
- Also has “Tensor Processing Units” (TPUs), the fastest option today.

Microsoft Azure

Name	GPU	GPUs	GPU RAM	vCPU	RAM	On-demand
`nc6 v3`	V100	1	16	6	112	$3.06
`nc12 v3`	V100	2	32	12	224	$6.12
`nc24 v3`	V100	4	64	24	448	$12.24
`nc6 v2`	P100	1	16	6	112	$2.07
`nc12 v2`	P100	2	32	12	224	$4.14
`nc24 v2`	P100	4	64	24	446	$8.28
`nc6`	K80	1	24	4	56	$0.90
`nc12`	K80	2	48	32	112	$7.20
`nc24`	K80	4	96	64	224	$14.40

- Pretty much equivalent to AWS
- Spot instances are not as well developed as in AWS or GCP.

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

On-prem Options

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

Building your own

- Quiet PC with 64GB RAM and 4x RTX 2080 Ti's: $7000
  - One day to build and set up
- Going beyond 4 GPUs is painful
- All you need to know: http://timdettmers.com/2018/12/16/deep-learning-hardware-guide/

Pre-built: NVIDIA

- NVIDIA DGX Station
  - 4x V100, 20-core CPU, 256GB RAM
  - $50,000

1. GPUs
4X NVIDIA Tesla® V100 32 GB/GPU
500 TFLOPS (Mixed Precision)
20,480 Total NVIDIA CUDA® Cores
2,560 Tensor Cores

2. SYSTEM MEMORY
256 GB RDIMM DDR4

3. GPU INTERCONNECT
NVIDIA NVLink™,
Fully Connected 4-Way

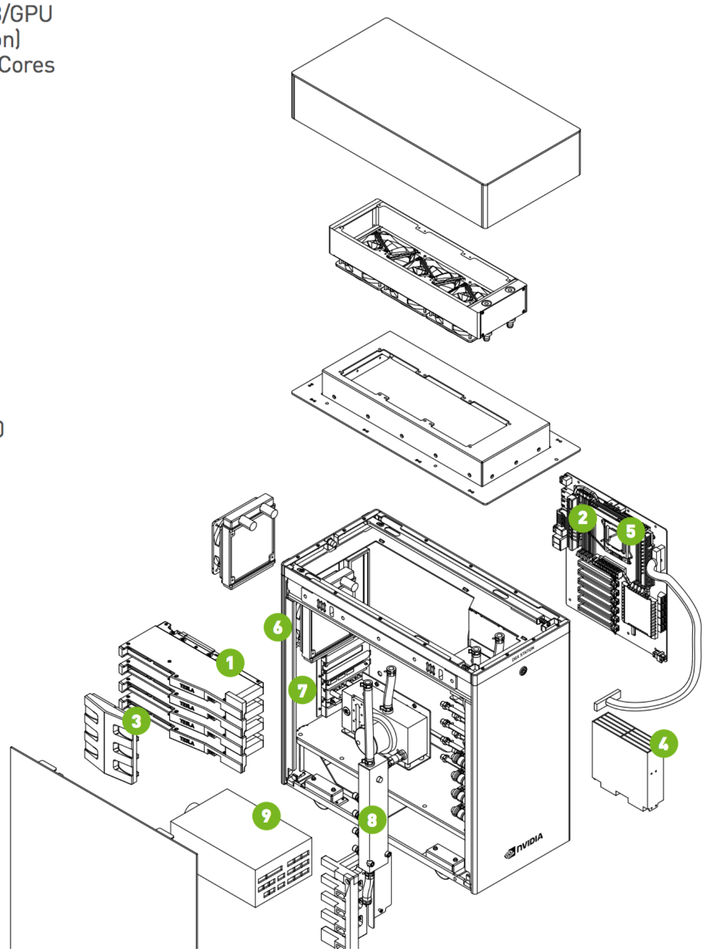
4. STORAGE
Data: 3 x 1.92 TB SSD RAID 0
OS: 1 x 1.92 TB SSD

5. CPU
Intel Xeon E5-2698 v4
2.2 GHz 20-Core

6. NETWORKING
2X 10 GbE

7. DISPLAYS
3X DisplayPort,
4K Resolution

8. COOLING
Water-Cooled



Pre-built: Lambda Labs

~20% more expensive than building yourself

BASIC

2x RTX 2080 Ti

In Stock

2-Way NVLink

Intel i9-9820X CPU (10 Cores)

64 GB Memory

2 TB NVMe (3,500 MB/s Read)

$5,902

Academic Discounts Available

CUSTOMIZE

BASIC

4x RTX 2080 Ti

In Stock

Intel i9-9820X CPU (10 Cores)

64 GB Memory

2 TB NVMe (3,500 MB/s Read)

$8,449

Academic Discounts Available

CUSTOMIZE

PREMIUM

4x Quadro RTX 6000

In Stock

Intel i9-9920X CPU (12 Cores)

128 GB Memory

2 TB NVMe (3,500 MB/s Read)

$20,657

Academic Discounts Available

CUSTOMIZE

RTX 6000 ==Titan RTX with better cooling

Pre-built: Lambda Labs

PREMIUM

8x Quadro RTX 6000

In Stock

2x Xeon Gold 5218 (16 Cores)

8x RTX 6000 GPUs

512 GB of Memory

1.92 TB NVMe SSD

4 TB SATA SSD

$42,138

Academic Discounts Available

CUSTOMIZE

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

$98,457

Academic Discounts Available

CUSTOMIZE

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 processe) definitely easier in the cloud

Recommendation for solo/startup

- Development
  - Build or Buy a 4x Turing-architecture PC
- Training/Evaluation
  - Use the same 4x GPU PC until architecture is dialed in
  - When running many experiments, either buy shared server machines or use cloud instances.

Recommendation for larger company

- Development
  - Buy a 4x Turing-architecture PC per ML scientist
  - Or, let them use V100 instances
- Training/Evaluation
  - Use cloud instances with proper provisioning and handling of failures

Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, Losswise

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Web: ALGORITHMIA, SELDON

Interchange: ONNX

CI / Testing: Buildkite, ?

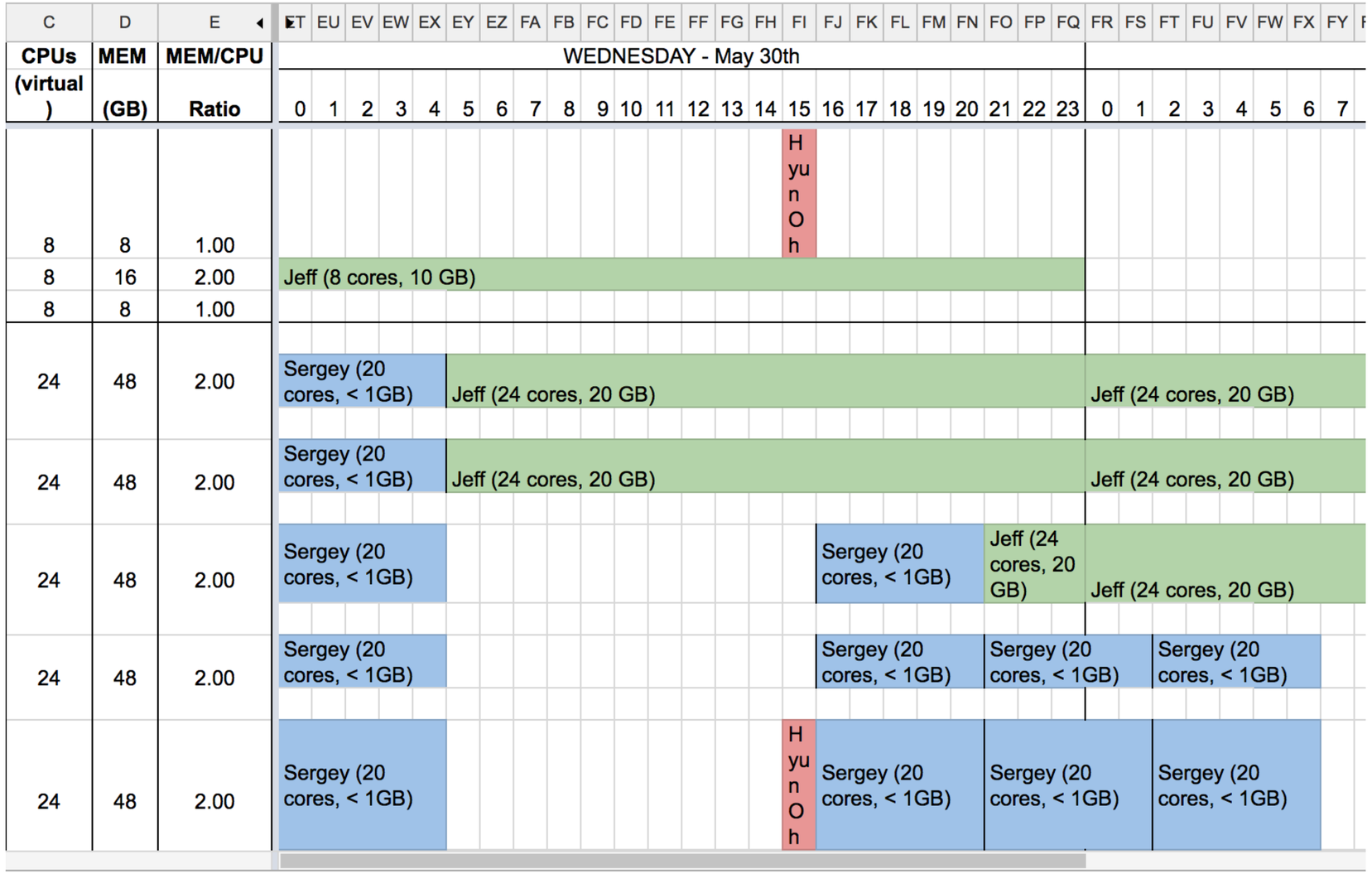
or or

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**
  - Spreadsheet
  - Python scripts
  - SLURM
  - Docker + Kubernetes
  - Software specialized for ML use cases

Spreadsheets

- People “reserve” what resources they need to use
- Awful, but still surprisingly common



Scripts or slurm

workload manager

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

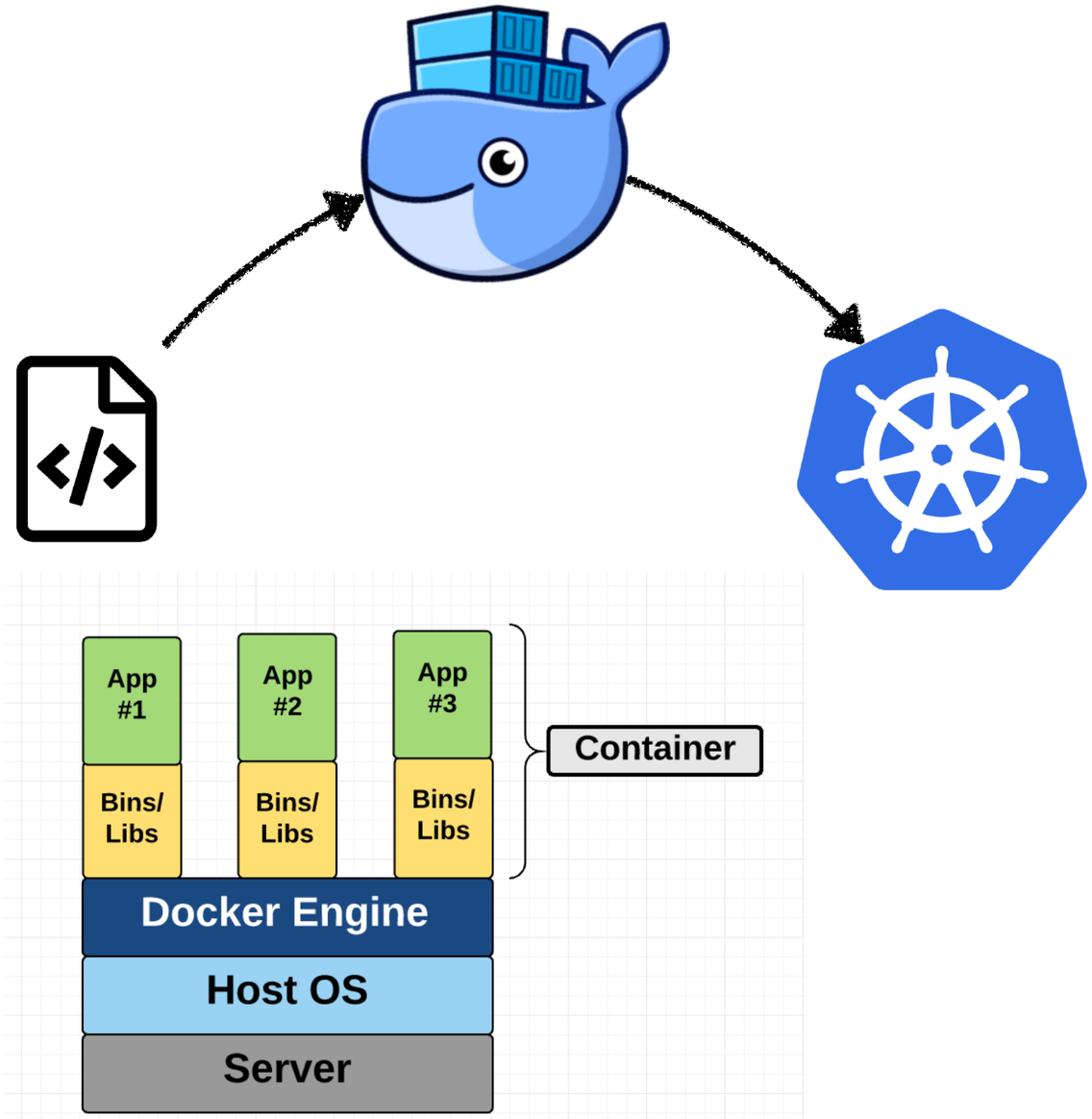
```
class GPUManager(object):
    def __init__(self, verbose: bool=False):
        self.lock_manager = Redlock([{"host": "localhost", "port": 6379, "db": 0}, ])
        self.verbose = verbose

    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
```

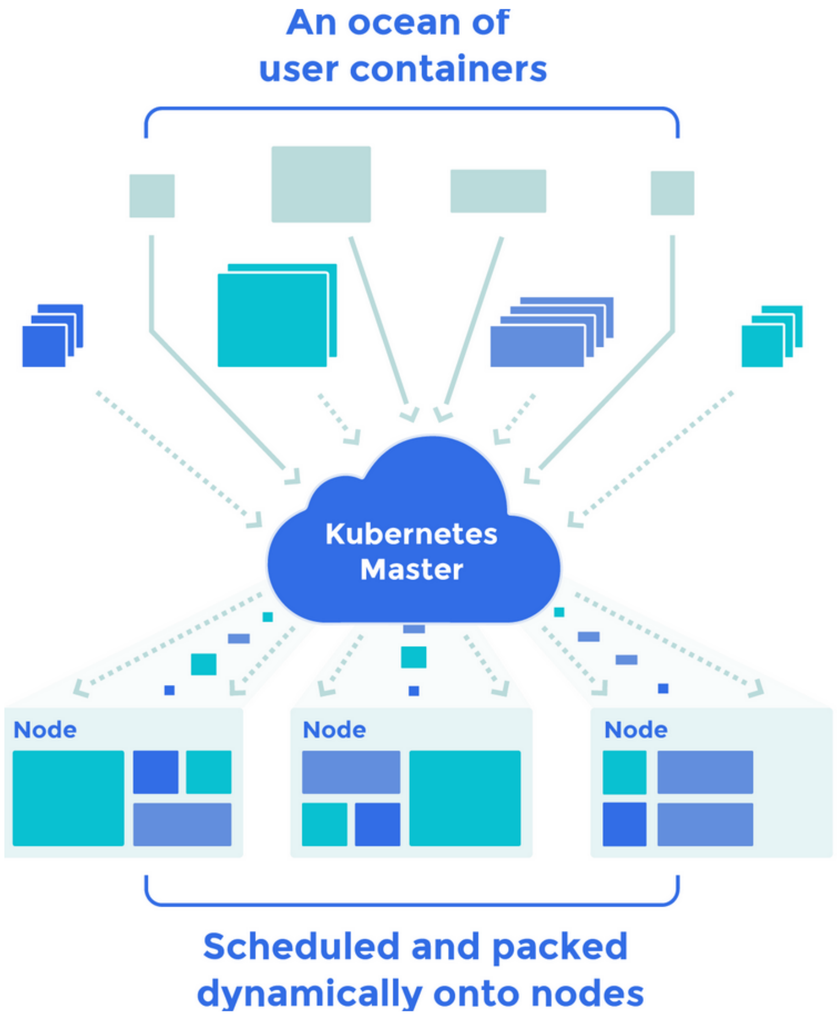
Docker + Kubernetes

- 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 tomorrow, and use it in lab



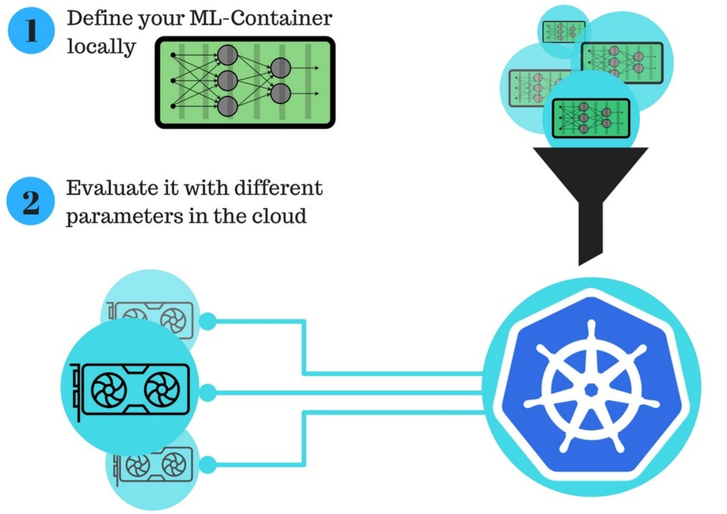
Docker + Kubernetes

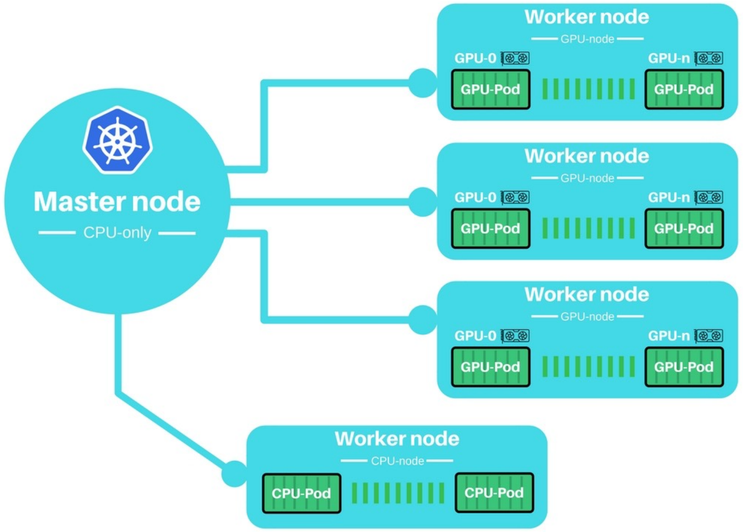
- Kubernetes is a way to run many docker containers on top of a cluster
- (Is actually what you are doing the labs in: https://github.com/jupyterhub/zero-to-jupyterhub-k8s)



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

Polyaxon Product

An enterprise-grade platform for agile, reproducible, and scalable machine learning.

Open Source project with paid features

Product

Tracking »

Automatically track key model metrics, hyperparams, visualizations, artifacts and resources, and version control code and data.

Orchestration »

Maximize the usage of your cluster by scheduling jobs and experiments via our CLI, dashboard, SDKs, or REST API.

Optimization »

Use our optimization algorithms to effectively run parallel experiments and find the best model.

Insights »

Visualize, search, and compare experiment results, hyperparams, training data and source code versions, so you can quickly analyze what worked and what didn't.

Model management »

Consistently develop, validate, deliver, and monitor models to create a competitive advantage.

Collaboration »

Instead of ad-hoc scripts, collaborate with the rest of your team and enable knowledge distribution practices in your organization.

Management »

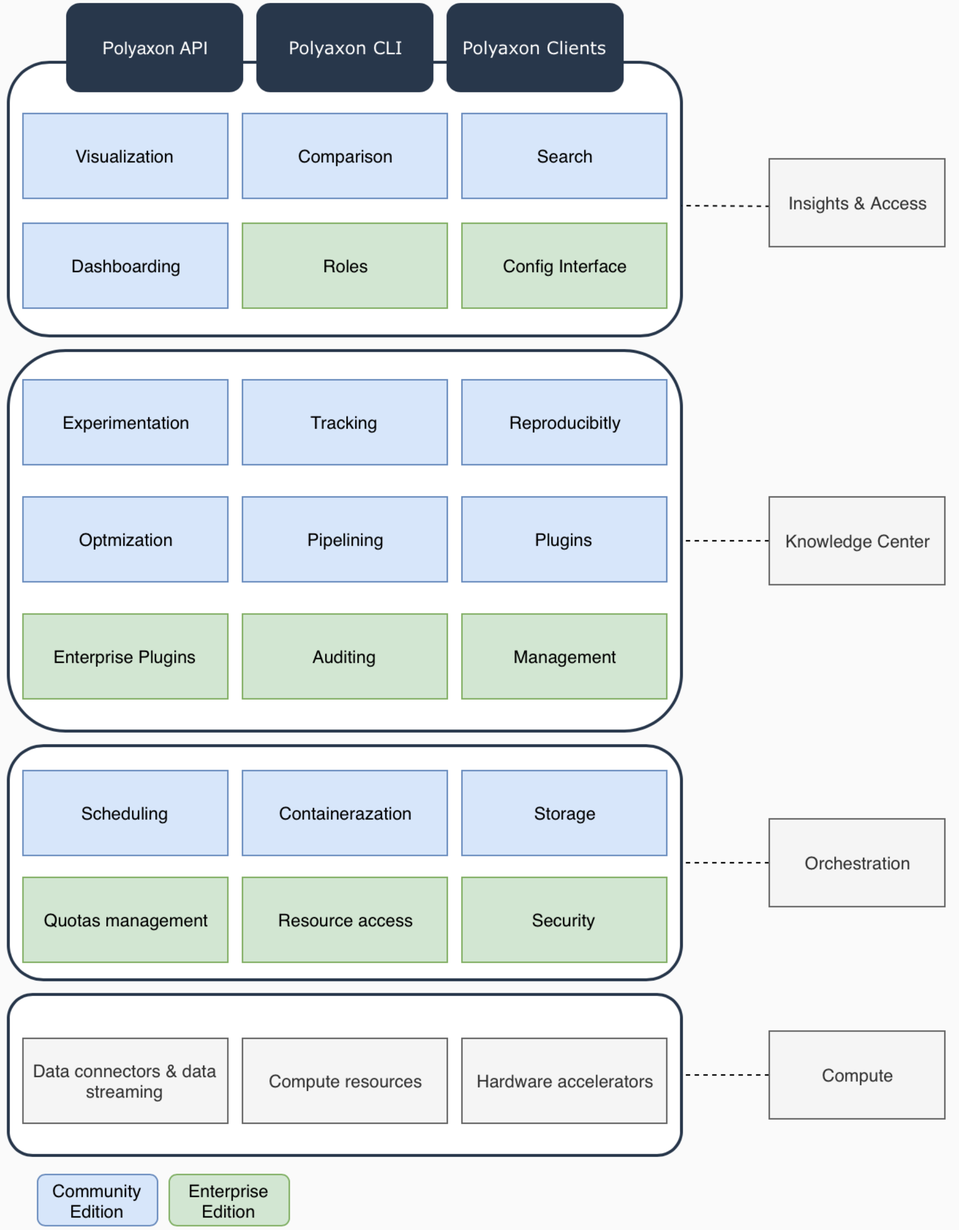
Reflect your organization's structure and manage permissions and quotas for your resources and cluster access.

Compliance »

Reproduce results and meet regulatory compliance without any added work.

Scalability »

Scale your resources as needed, and run jobs and experiments on any platform (AWS, Microsoft Azure, Google Cloud Platform, and on-premises hardware).



Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

or

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: comet, Weights & Biases, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

Deployment

Monitoring: ?

Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

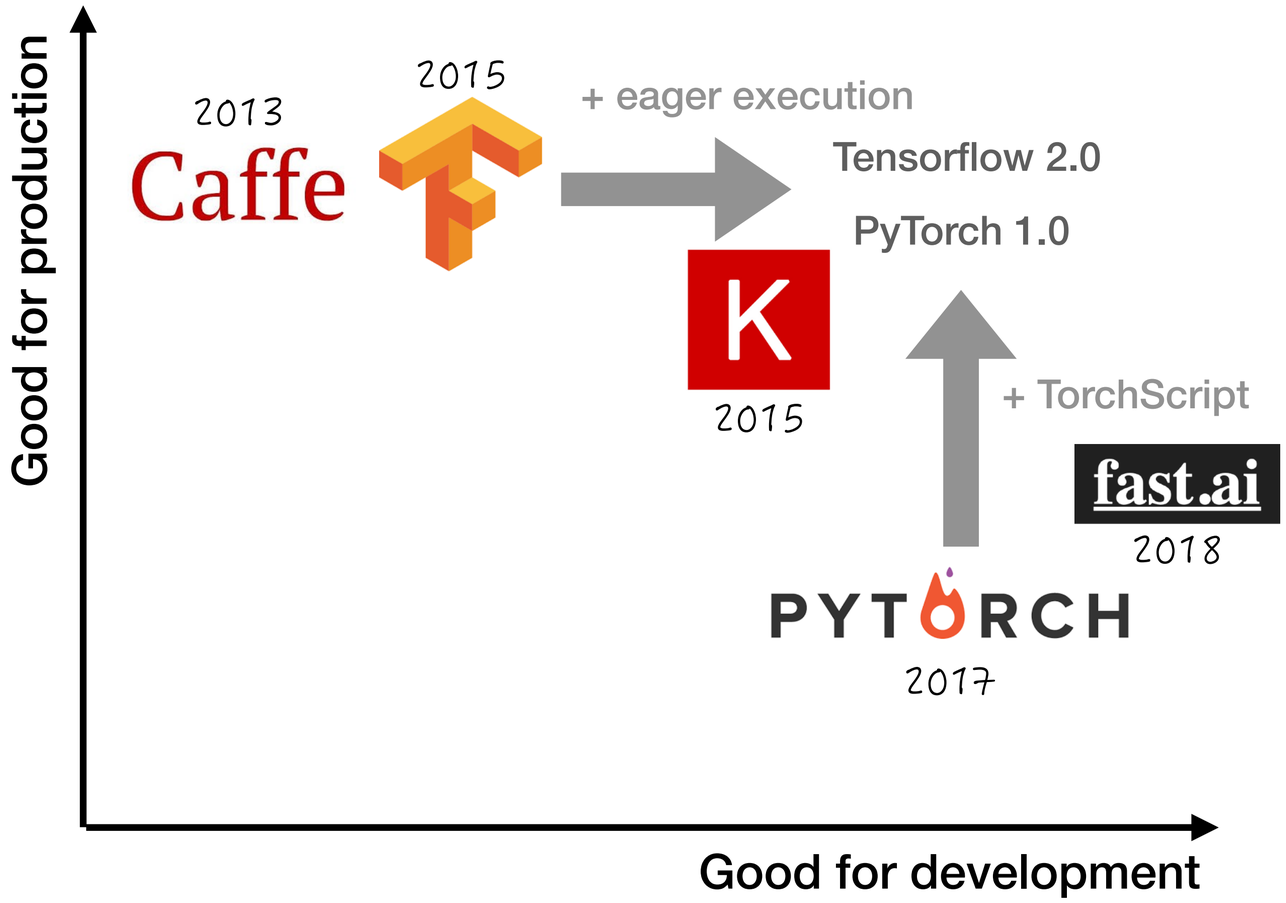
Interchange: ONNX

CI / Testing: Buildkite, ?

or

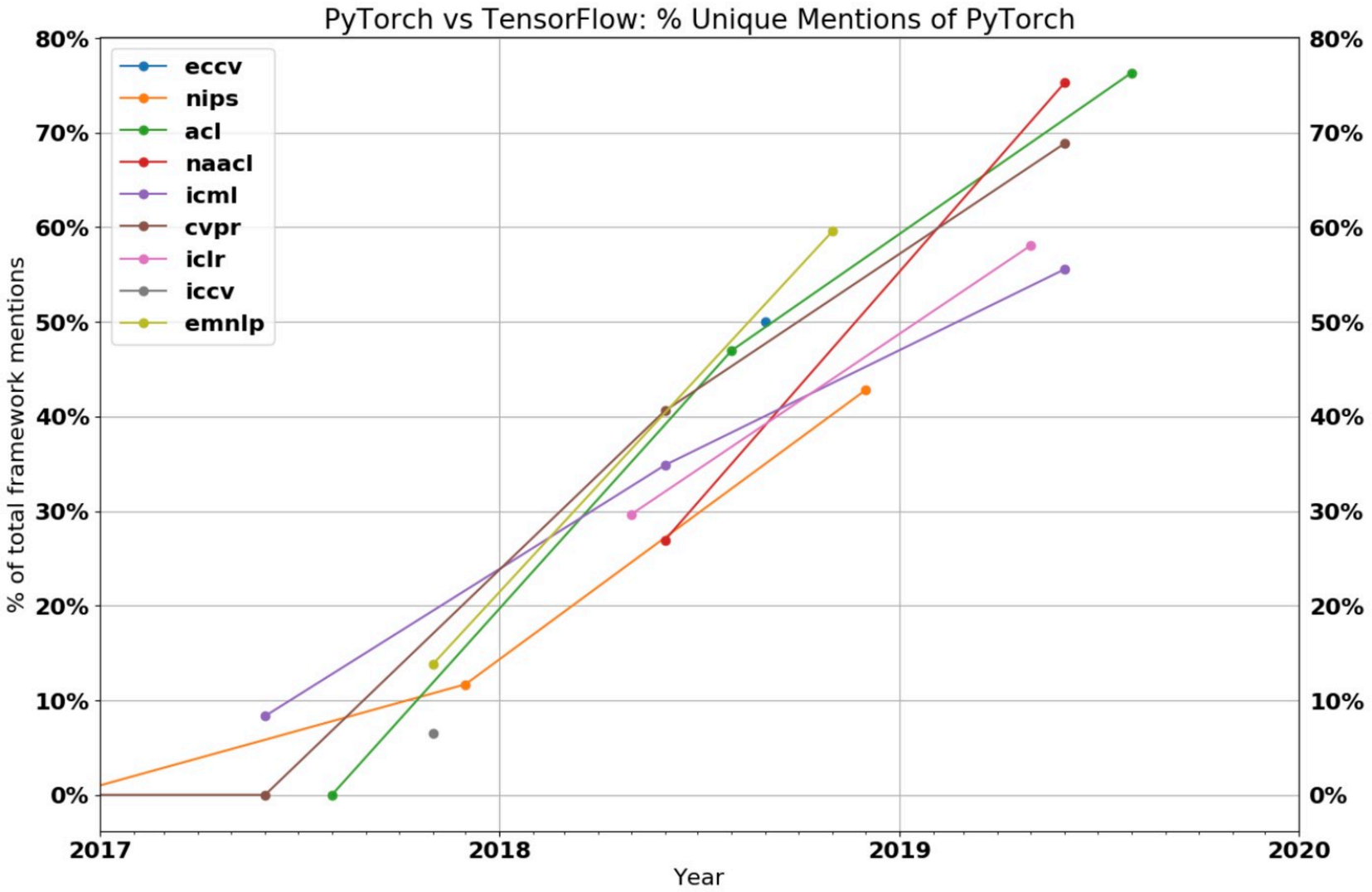
or

Deep Learning Frameworks



- Unless you have a good reason not to, use Tensorflow/Keras or PyTorch
- Both are converging to the same ideal point:
  - easy development via define-by-run
  - multi-platform optimized execution graph
- Today, most new projects use PyTorch
- For most problems, fast.ai library worth starting with for immediate best practices

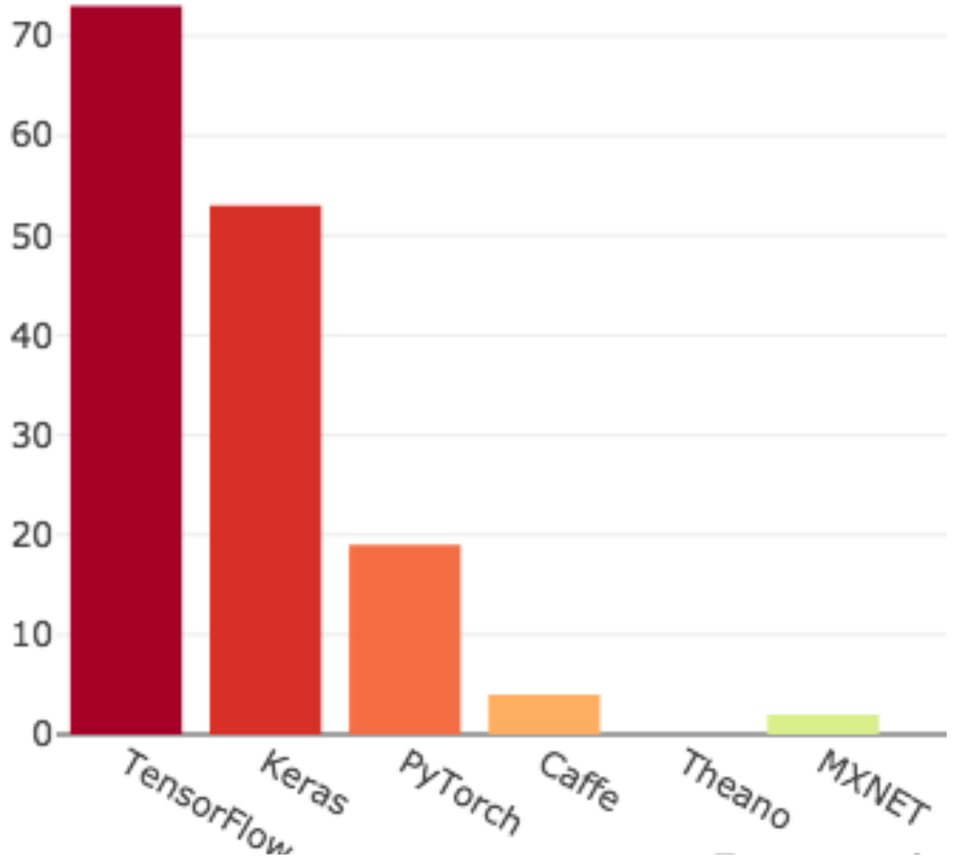
PyTorch dominates new development



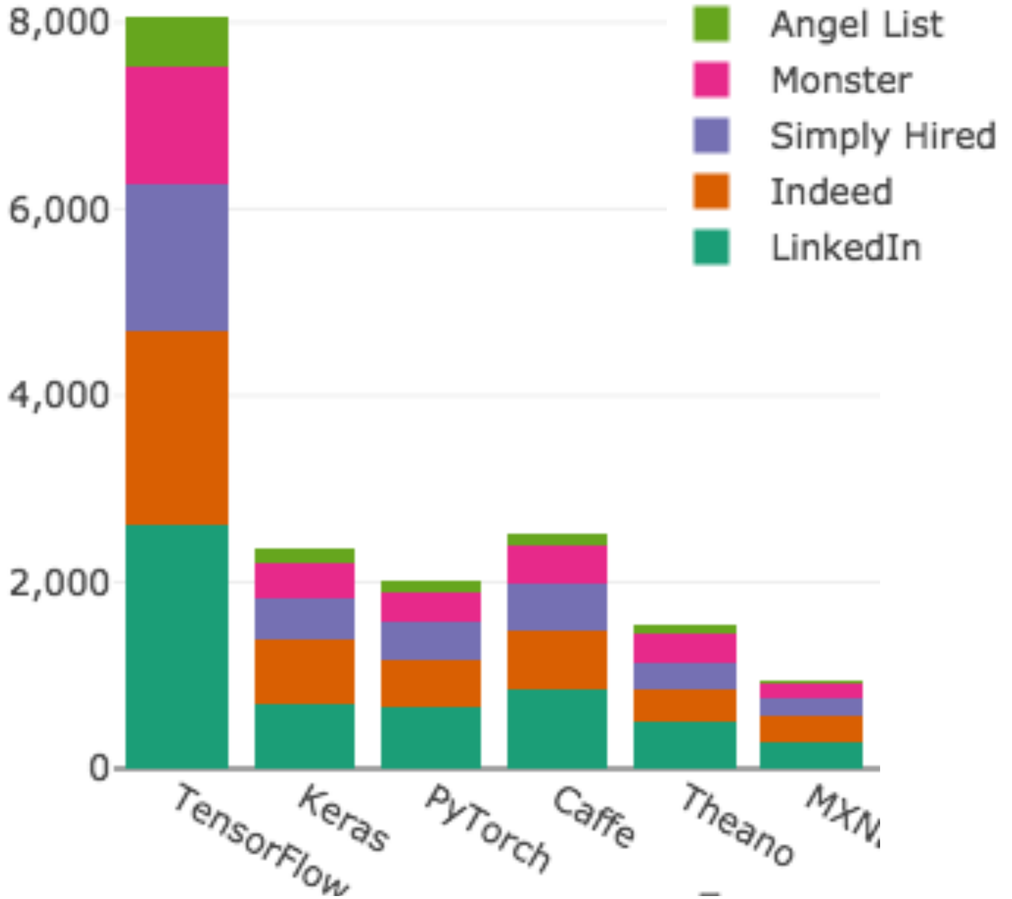
https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

Tensorflow/Keras still lead job posts

Google searches (2018)



Job posts (2018)



https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Interchange: ONNX

CI / Testing: Buildkite, ?

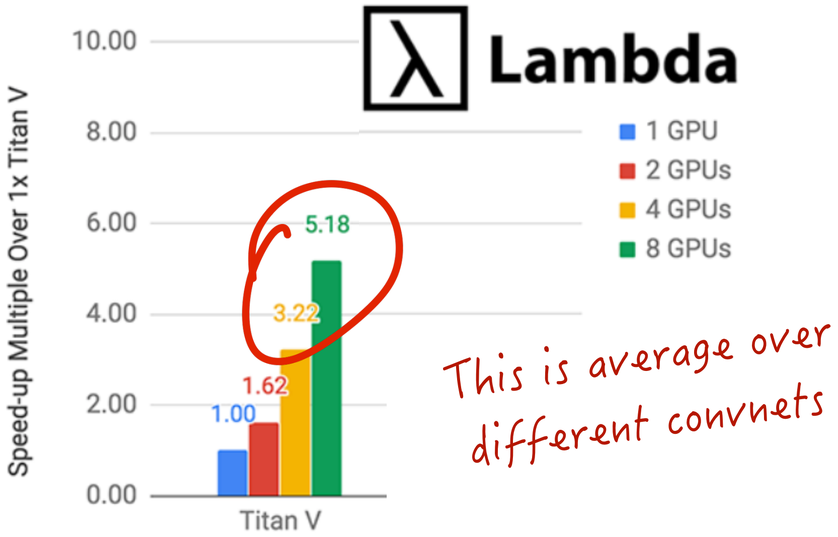
or or

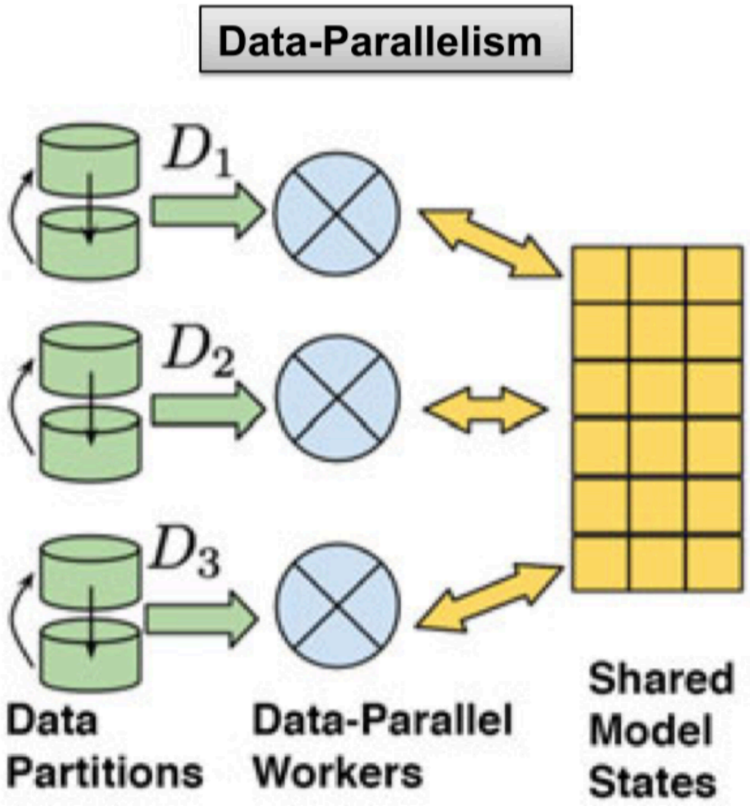
Distributed Training

- Using multiple GPUs and/or machines to train a single model.
- More complex than simply running different experiments on different GPUs
- Makes sense to reduce iteration time, especially 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/



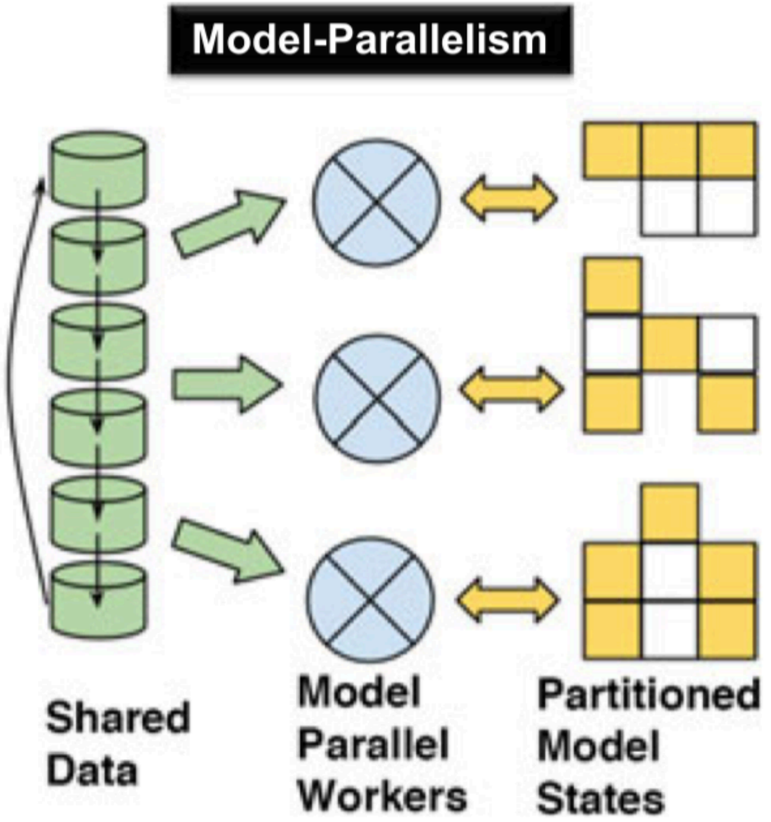


http://www.cs.cmu.edu/~pengtaox/papers/petuum_15.pdf

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
  - Better to buy the largest GPU you can



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

Data-parallel Tensorflow

- Can be quite easy:

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(32, 3, activation='relu',
input_shape=(28, 28, 1)),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(64, activation='relu'),
      tf.keras.layers.Dense(10, activation='softmax')
  ])

  model.compile(loss='sparse_categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
```

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

- If want to spread over multiple machines, or make it work with model-parallel, gets more complicated

In training code

```
torch.distributed.init_process_group(backend='YOUR BACKEND',
                                     init_method='env://')

model = torch.nn.parallel.DistributedDataParallel(model,
                                                  device_ids=[arg.local_rank],
                                                  output_device=arg.local_rank)
```

In consoles of the workers

```
# Node 1: *(IP: 192.168.1.1, and has a free port: 1234)*
$ python -m torch.distributed.launch --nproc_per_node=NUM_GPUS_YOU_HAVE
           --nnodes=4 --node_rank=0 --master_addr="192.168.1.1"
           --master_port=1234 YOUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3
           and all other arguments of your training script)
# Node 2:
$ python -m torch.distributed.launch --nproc_per_node=NUM_GPUS_YOU_HAVE
           --nnodes=4 --node_rank=1 --master_addr="192.168.1.1"
           --master_port=1234 YOUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3
           and all other arguments of your training script)
# Node 3:
$ python -m torch.distributed.launch --nproc_per_node=NUM_GPUS_YOU_HAVE
           --nnodes=4 --node_rank=2 --master_addr="192.168.1.1"
           --master_port=1234 YOUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3
           and all other arguments of your training script)
# Node 4:
$ python -m torch.distributed.launch --nproc_per_node=NUM_GPUS_YOU_HAVE
           --nnodes=4 --node_rank=3 --master_addr="192.168.1.1"
           --master_port=1234 YOUR_TRAINING_SCRIPT.py (--arg1 --arg2 --arg3
           and all other arguments of your training script)
```

Ray

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

```
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!")
```

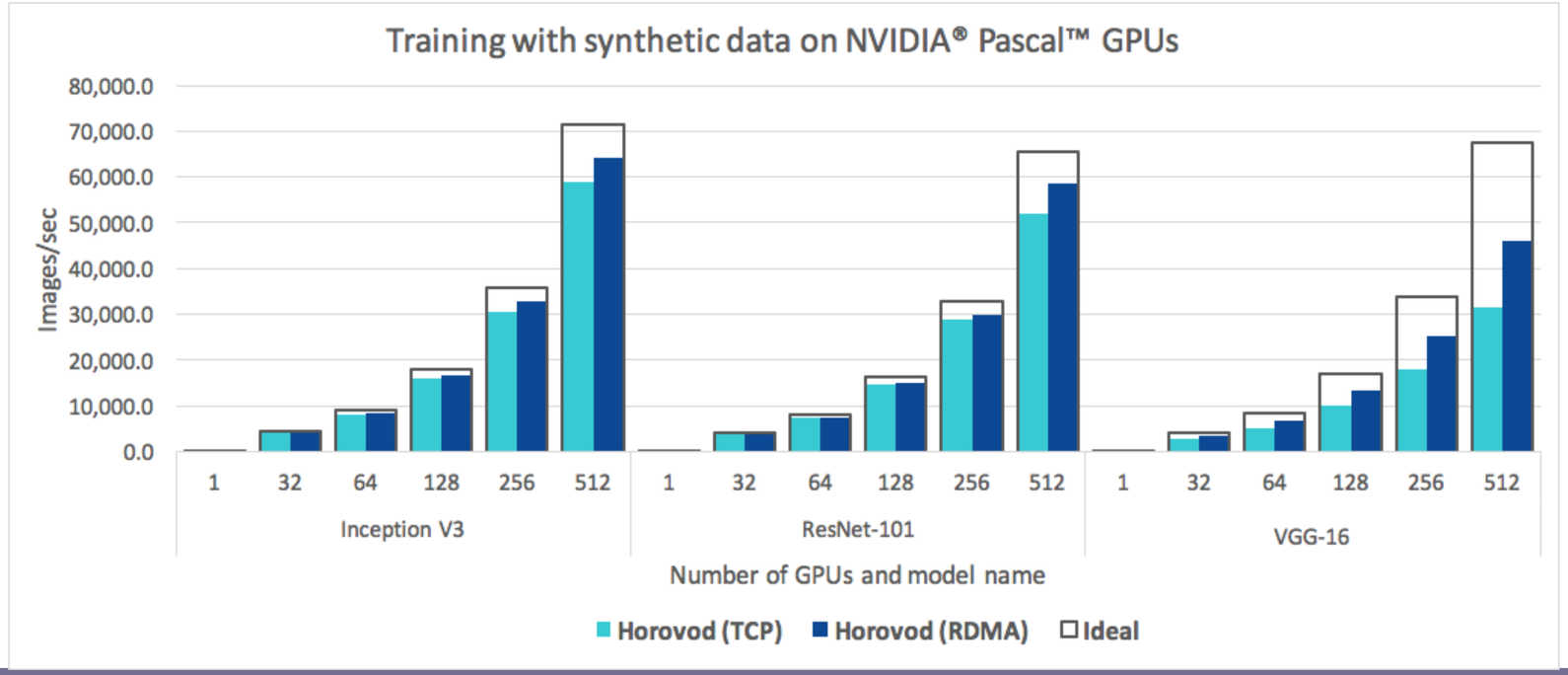
Then, start a Ray cluster via autoscaler or manually.

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

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

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



Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: Weights & Biases, comet, TensorBoard, Losswise

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Web: ALGORITHMIA, SELDON

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite

Interchange: ONNX

CI / Testing: Buildkite, ?

or or

Experiment Management

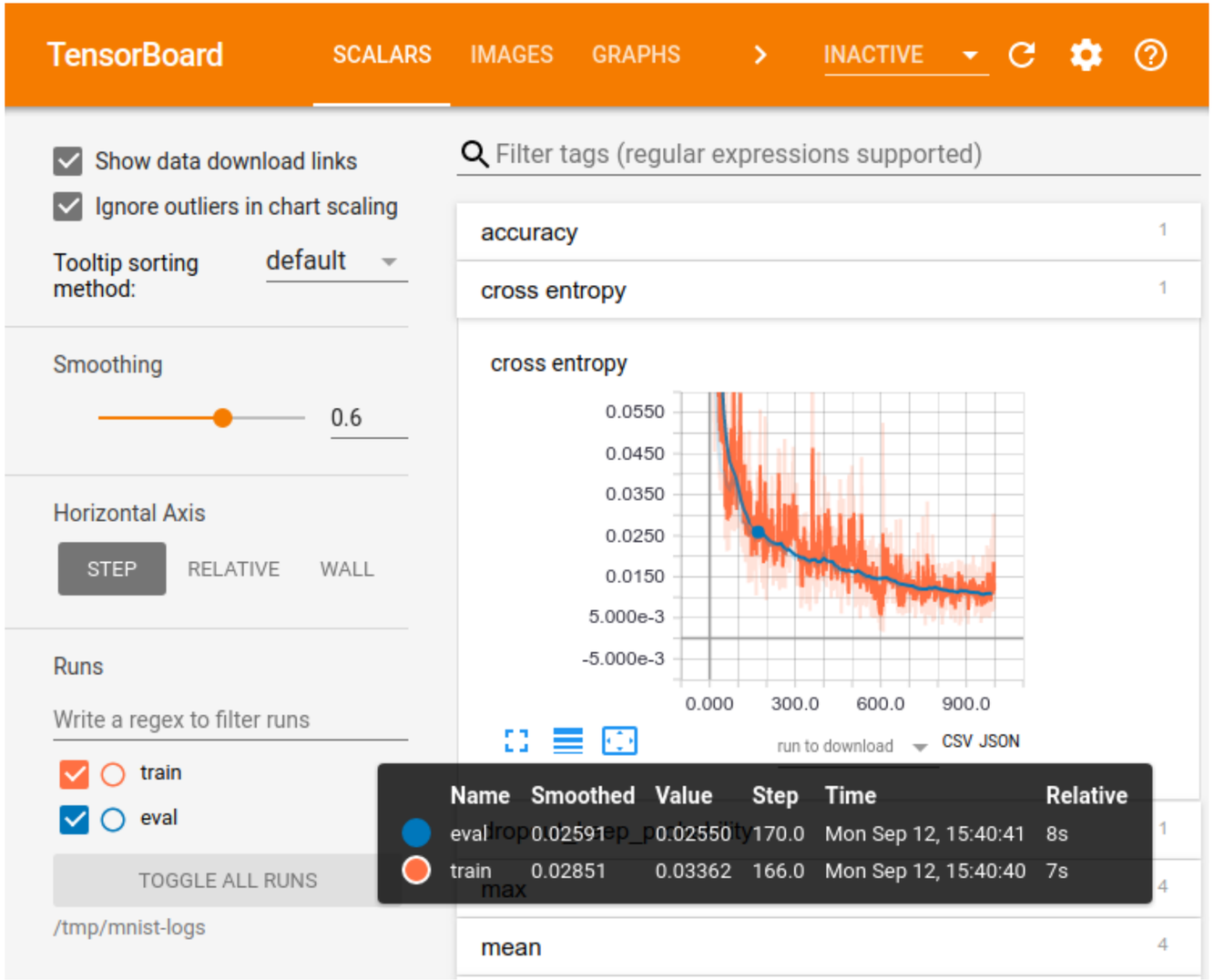
- Even running one experiment at a time, can lose track of which code, parameters, and dataset generated which trained model.
- When running multiple experiments, problem is much worse.

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



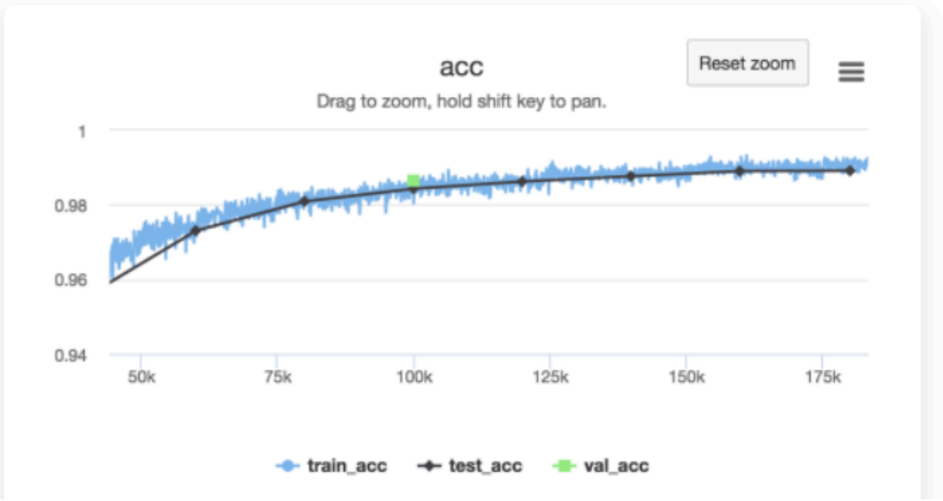
Losswise

Monitoring for ML

Real time graphs, comparison tables, completion estimates, and more.

Git tracking

Seamlessly correlate experimental results with your git branch and your git diff at the time of your experiment.



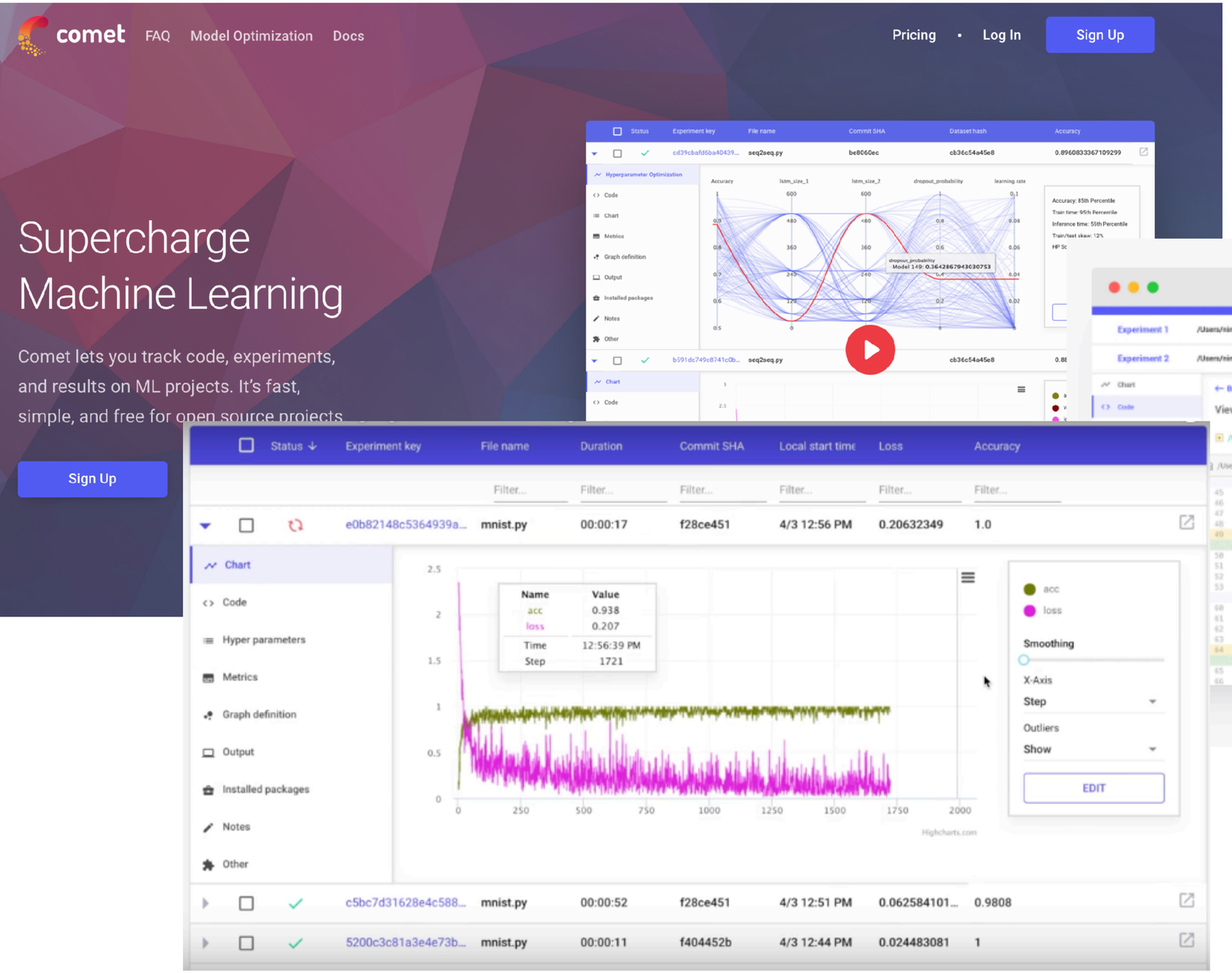
Summary statistics

Losswise provides real time tables showing you your latest model results, **including estimates of when your models will be done training.**

Sort and filter by min loss, max accuracy, or whatever you wish, to get a better understanding of how parameters affect your final results.

Estimated completion	Iter	Completion	max(test_acc)	min(test_loss)
September 14th 2017 at 7:04:17 am	150000	100.00%	0.97971	0.13376
September 18th 2017 at 10:42:29 pm	200000	100.00%	0.97968	0.16518
September 21st 2017 at 2:42:24 am	200000	100.00%	0.97917	0.16369
September 12th 2017 at 8:29:06 am	150000	100.00%	0.97891	0.10267
September 27th 2017 at 10:18:12 pm	200000	100.00%	0.97878	0.47146
September 26th 2017 at 6:03:40 pm	174090	87.05%	0.97815	0.15282
September 14th 2017 at 12:06:38 am	150000	100.00%	0.97704	0.13551
October 1st 2017 at 1:53:15 pm	165850	82.93%	0.97442	0.05283
September 18th 2017 at 8:49:50 pm	154010	77.01%	0.97369	0.18086
October 2nd 2017 at 1:49:02 pm	49490	24.75%	0.96503	0.09603

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

Weights & Biases

- What we use in Lab 3

MLFlow tracking

An open source platform for the machine learning lifecycle

- Self-hosted solution from DataBricks

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

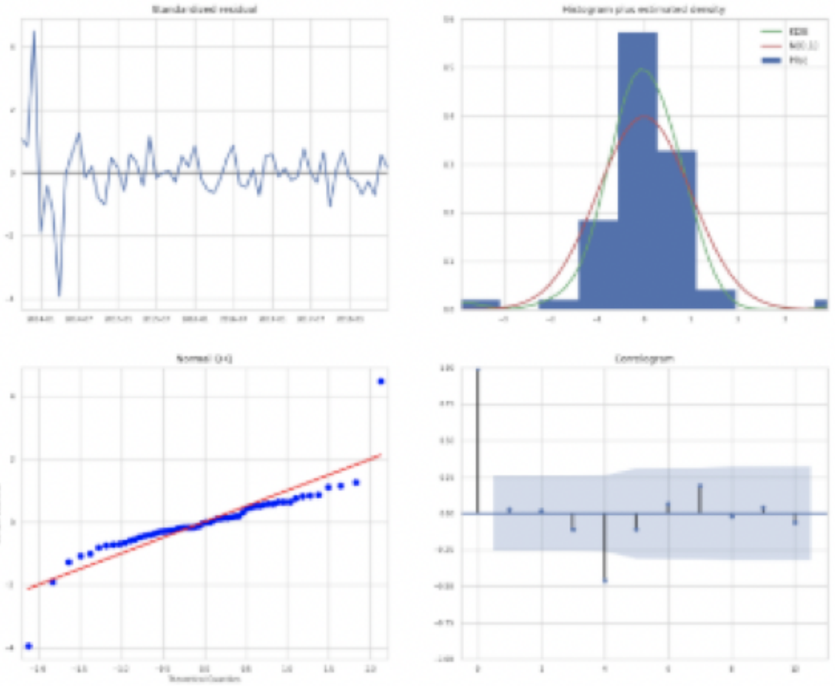
Experiment ID: 450992 Artifact Location: dbfs:/databricks/mlflow/450992

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

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

- Versioning: Quilt, DVC, Pachyderm, LIQUIDATA
- Labeling: figure eight, scale, prodigy
- Processing: PY, Luigi
- Database
- Storage: NFS, S3

 or 

Development

- Distributed Training: PYTORCH, RAY, HOROVOD
- Frameworks: PYTORCH, K, fast.ai
- Software Engineering: jupyter, git

Training/Evaluation

- Hyperparameter Tuning: SIGOPT, tune
- Experiment Management: Weights & Biases, comet, TensorBoard, Losswise
- Resource Management: Polyaxon, PY, slurm workload manager

or 

Deployment

- Monitoring: ?
- Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlow Lite
- Web: ALGORITHMIA, SELDON
- Interchange: ONNX
- CI / Testing: Buildkite, ?

 or  or 

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

Hyperas

Keras + Hyperopt: A very simple wrapper for convenient hyperparameter optimization

```
from hyperas.distributions import uniform

model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout({{uniform(0, 1)}}))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout({{uniform(0, 1)}}))
model.add(Dense(10))
model.add(Activation('softmax'))
```

```
best_run = optim.minimize(model=model,
                          data=data,
                          algo=tpe.suggest,
                          max_evals=10,
                          trials=Trials())
```

Algorithms

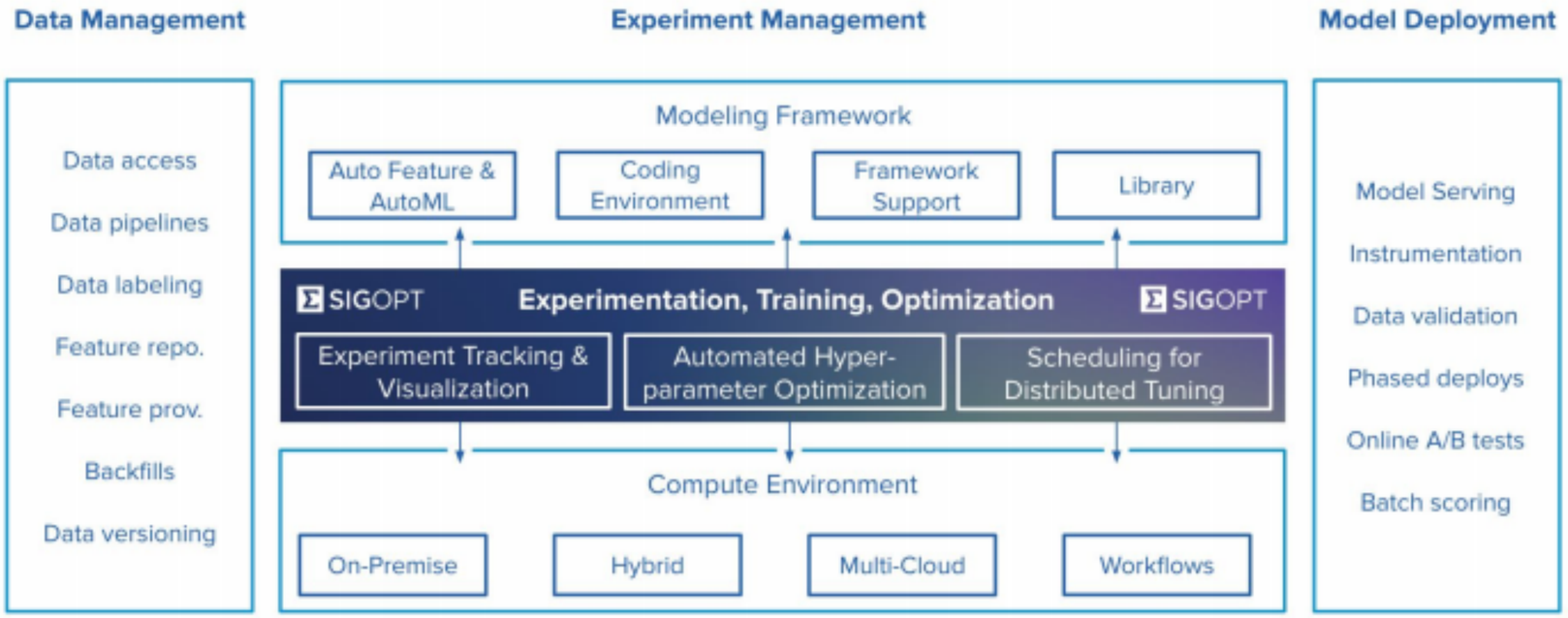
Currently two algorithms are implemented in hyperopt:

- Random Search
- Tree of Parzen Estimators (TPE)

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

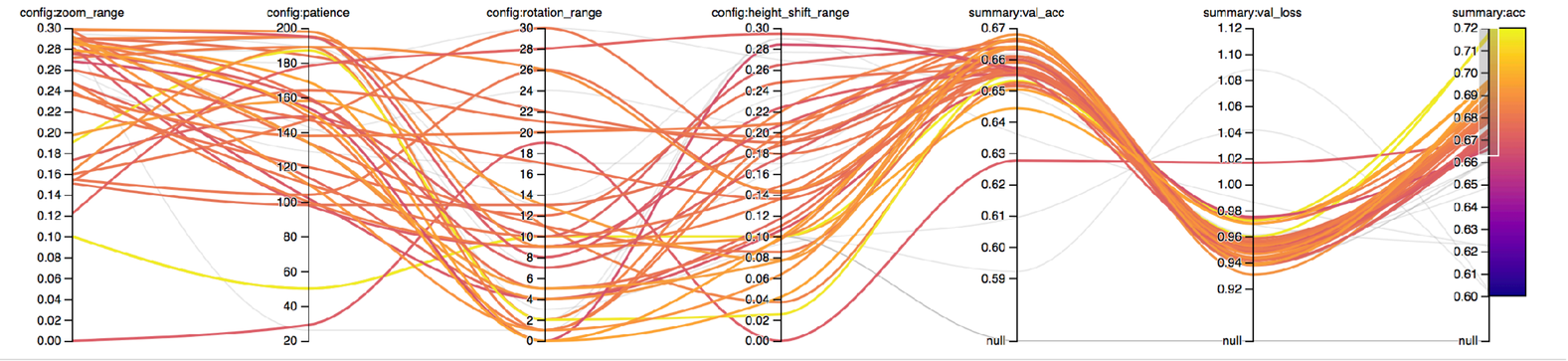
You'll reach optimal values up to 100x faster than other methods.

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()
```

- We will also see a solution from W&B in Lab 4



Can also visually inspect hyperparam results

```yaml
# Method can be bayes, random, grid
method: bayes

# Metric to optimize
metric:
  name: val_loss
  goal: minimize

# Should we early terminate runs
early_terminate:
  type: envelope

# Parameters to search over
parameters:
  learning-rate:
    min: 0.001
    max: 0.1
  optimizer:
    values: ["adam", "sgd"]
  dropout:
    min: 0.01
    max: 0.5
  epochs:
    value: 30
```

Questions?

Amazon SageMaker · Determined AI · Neptune Machine Learning Lab · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data

Versioning: Quilt, DVC, Pachyderm, LIQUIDATA

Labeling: figure eight, scale, prodigy

Processing: PY, Luigi

Database

Storage: NFS, S3

or

Development

Distributed Training: PYTORCH, RAY, HOROVOD

Frameworks: PYTORCH, K, fast.ai

Software Engineering: jupyter, git

Training/Evaluation

Hyperparameter Tuning: SIGOPT, tune

Experiment Management: comet, Weights & Biases, TensorBoard, LOSSWISE

Resource Management: Polyaxon, PY, slurm workload manager

or

Deployment

Monitoring: ?

Hardware / Mobile: ML, NVIDIA TensorRT, TensorFlowLite

Web: ALGORITHMIA, SELDON

Interchange: ONNX

CI / Testing: Buildkite, ?

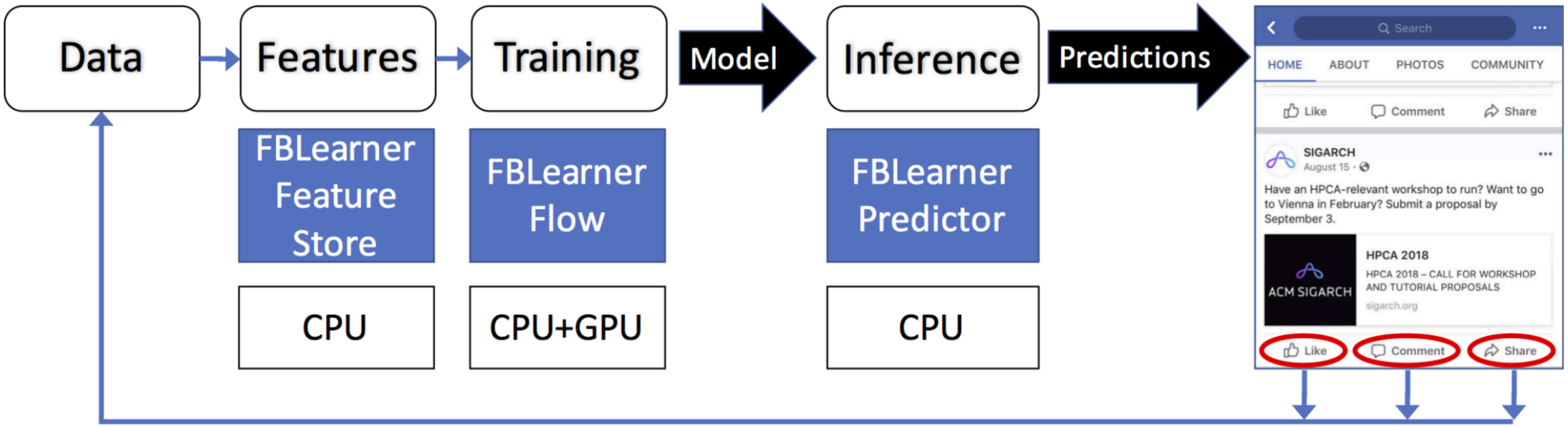
or or

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

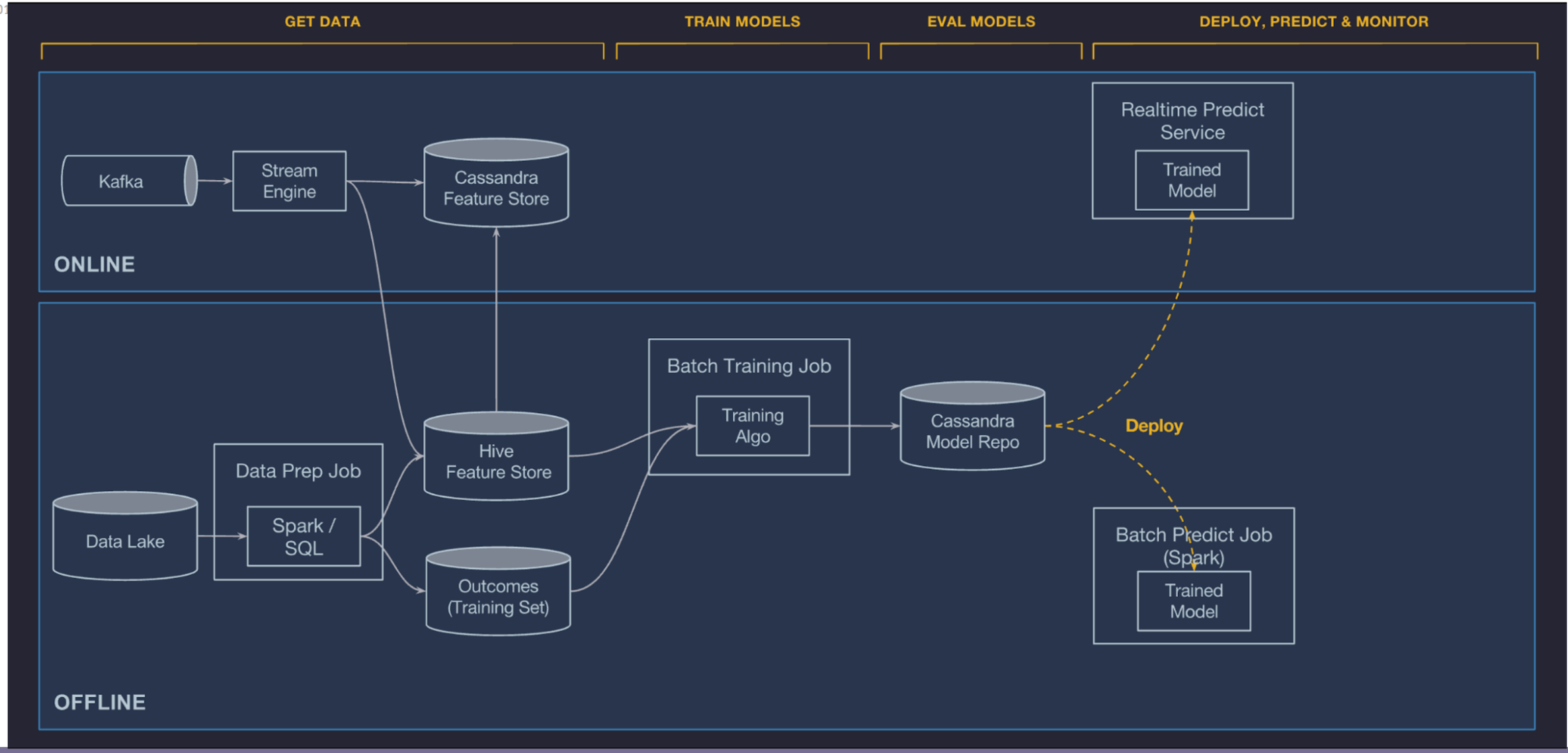
Introducing FBLearner Flow: Facebook's AI backbone



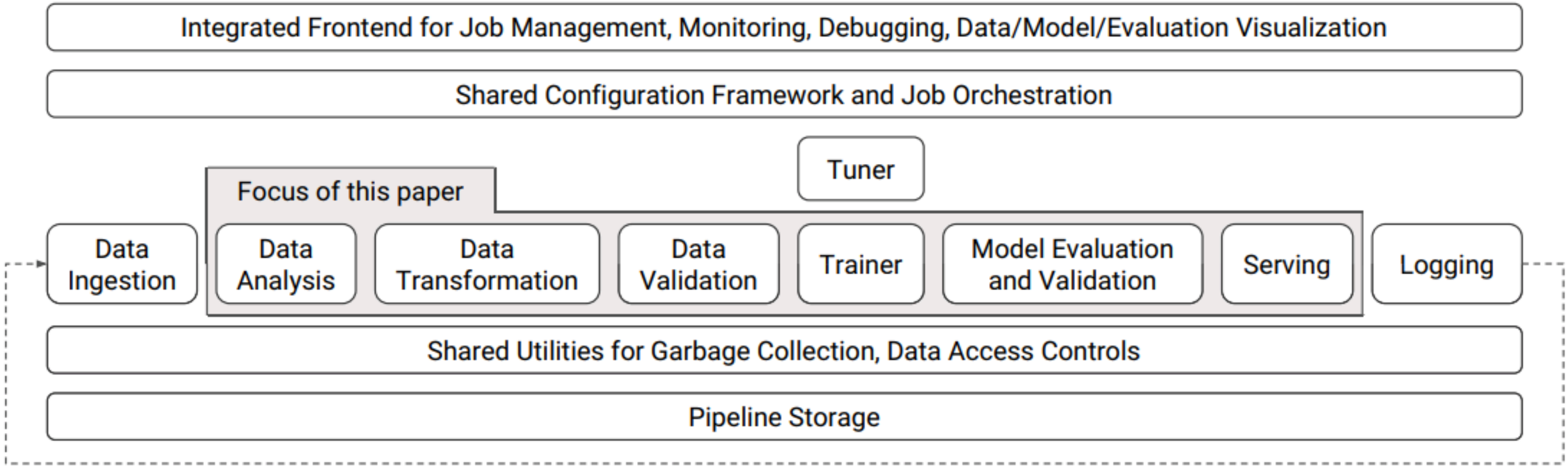
Meet Michelangelo: Uber's Machine Learning Platform

By Jeremy Hermann & Mike Del Balso

September 5, 201

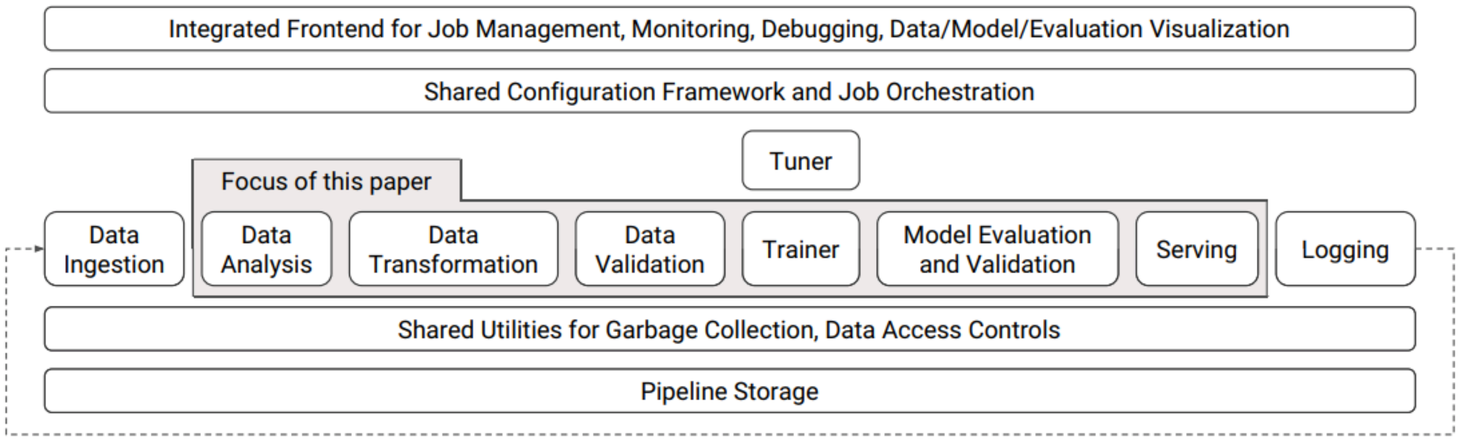


TFX: A TensorFlow-Based Production-Scale Machine Learning Platform



KDD 2017 Applied Data Science Paper KDD'17, August 13–17, 2017, Halifax, NS, Canada

TFX: A TensorFlow-Based Production-Scale Machine Learning Platform



TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines

When you're ready to move your models from research to production, use TFX to create and manage a production pipeline.

See tutorials

Tutorials show you how to use TFX with complete, end-to-end examples.

See the guide

Guides explain the concepts and components of TFX.

TensorFlow Data Validation

Get started →

TensorFlow Transform

Get started →

TensorFlow Model Analysis

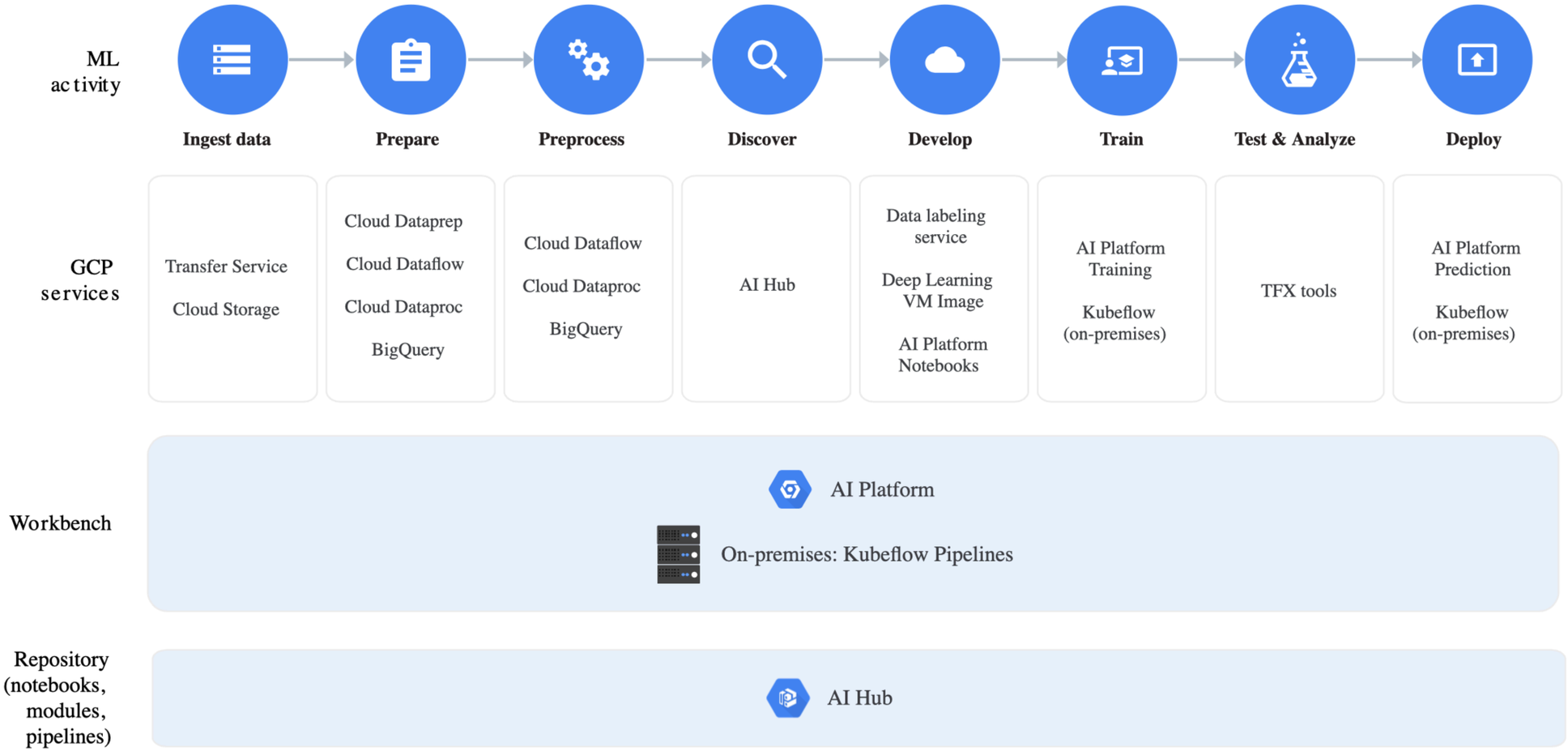
Get started →

TensorFlow Serving

Get started →

Google Cloud AI Platform

Create your AI applications once, then run them easily on both GCP and on-premises.



40% markup over corresponding EC2 instances

BUILD

Collect & prepare training data

Data labeling & pre-built notebooks for common problems

Choose & optimize your ML algorithm

Built-in, high-performance algorithms and hundreds of ready to use algorithms in **AWS Marketplace**

TRAIN

Set up & manage environments for training

One-click training using Amazon EC2 On-Demand or Spot instances

Train & tune model

Train once, run anywhere & model optimization

DEPLOY

Deploy model in production

One-click deployment

Scale & manage the production environment

Fully managed with auto-scaling for 75% less

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

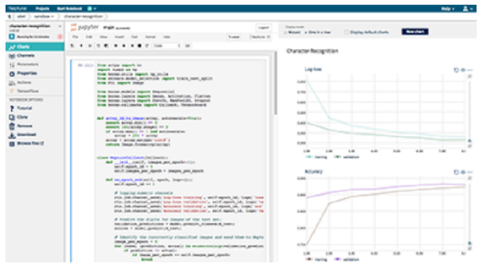
Run on your laptop or entirely in the cloud

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

READ MORE

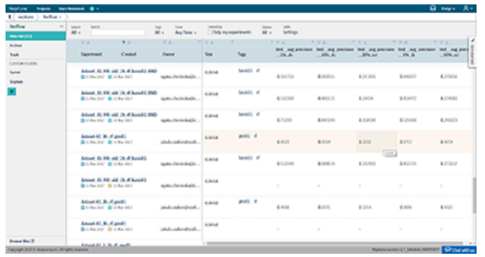
Use Jupyter Notebooks in the cloud

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



Quickly find and compare your best experiments

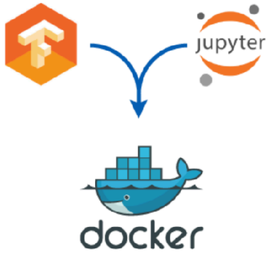
Use your private "Kaggle leaderboard" like a dashboard to filter, explore and sort through your experiments. Find your top machine learning models based on your favorite metric.



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.

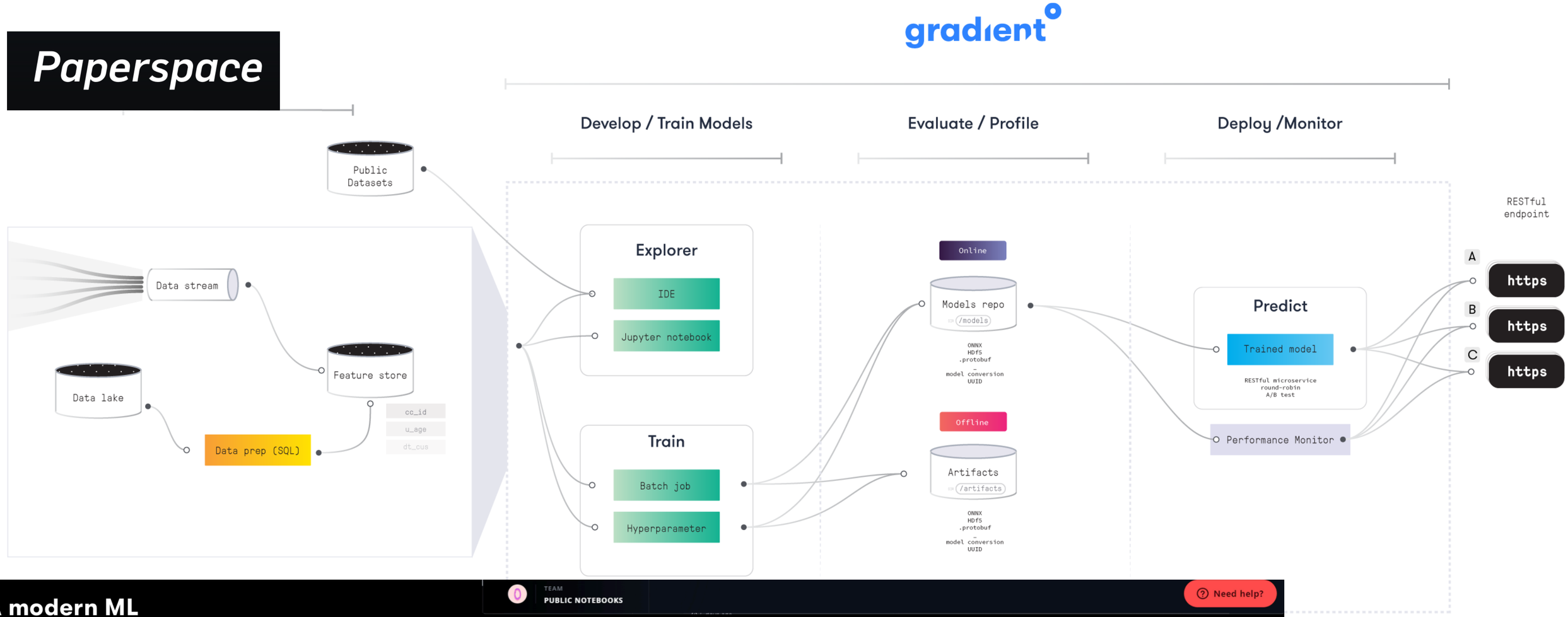
View pricing

Iterate in Parallel

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

Paperspace

gradient



A modern ML pipeline with Gradient

01 **Develop**

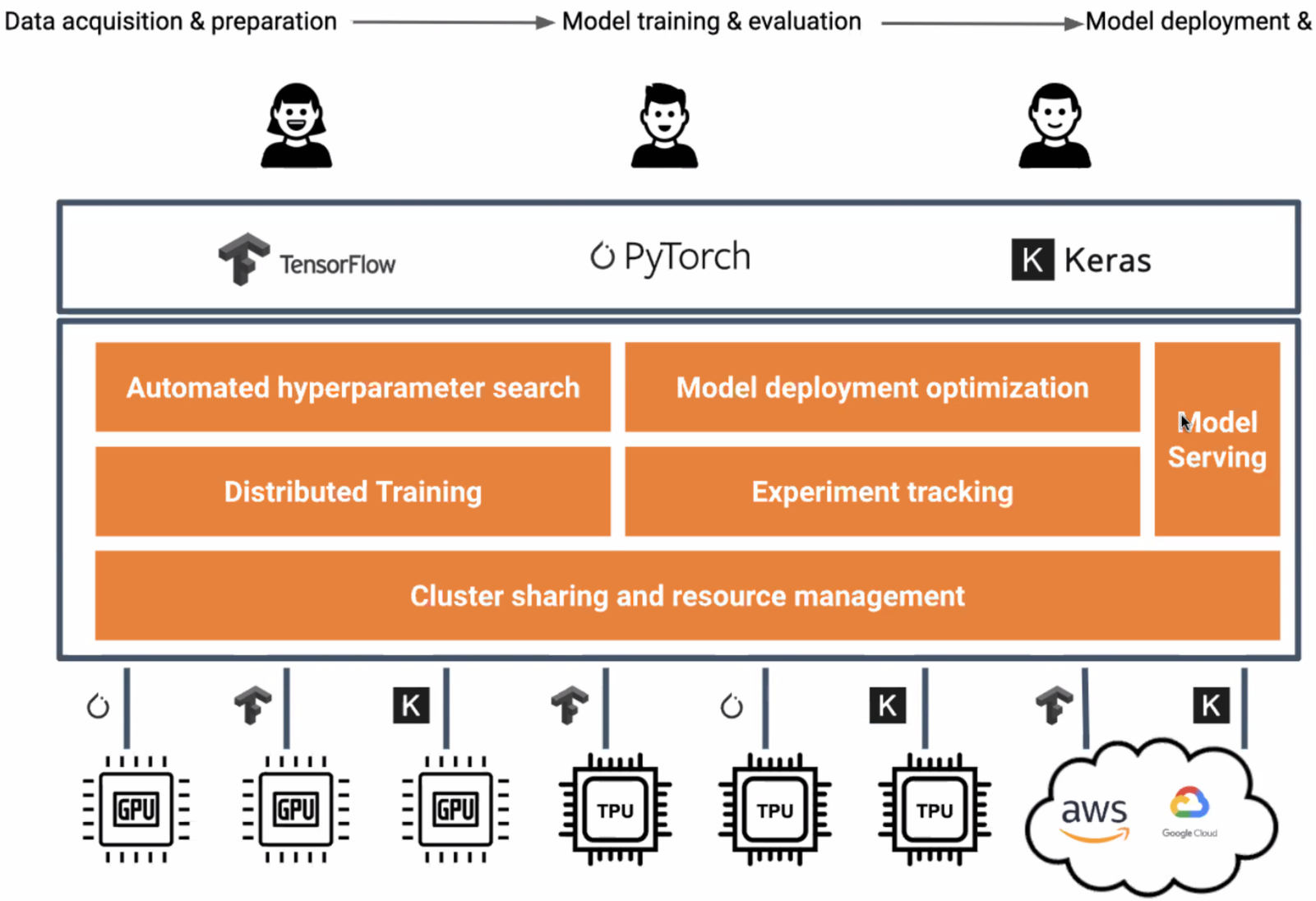
02 **Train**

03 **Measure**

04 **Deploy**

Easily deploy your models as API endpoint in seconds. Scale your deployment to respond to request volume. Deploy on GPUs or CPU instances.

Determined AI

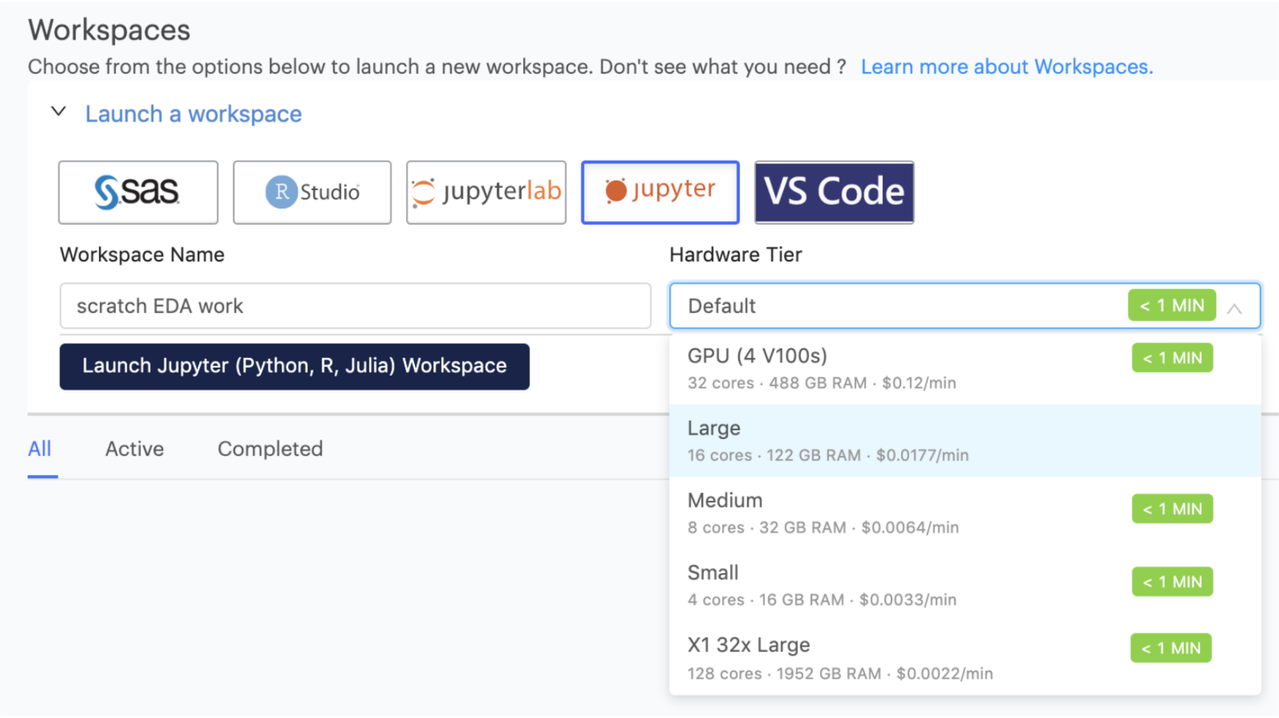


- HyperBand based hyperparameter tuning
- In-house distributed training module

Domino Data Lab

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

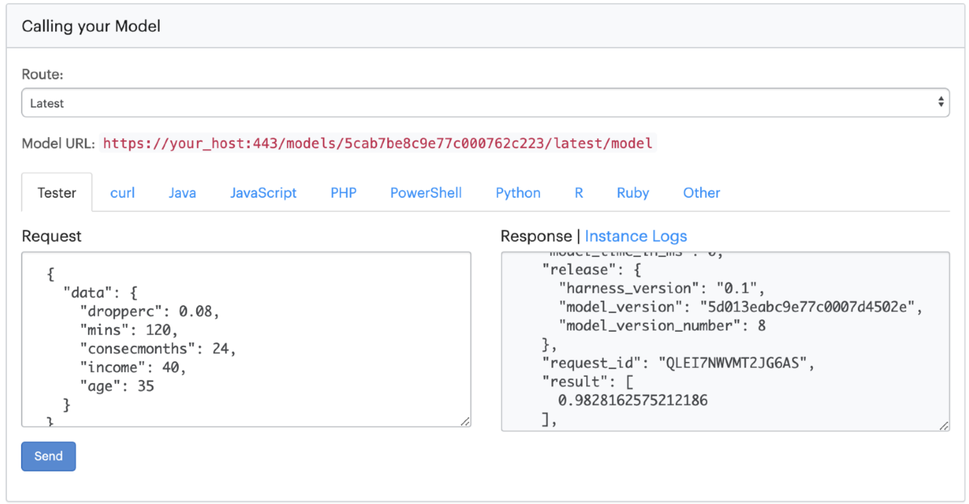
Provision compute



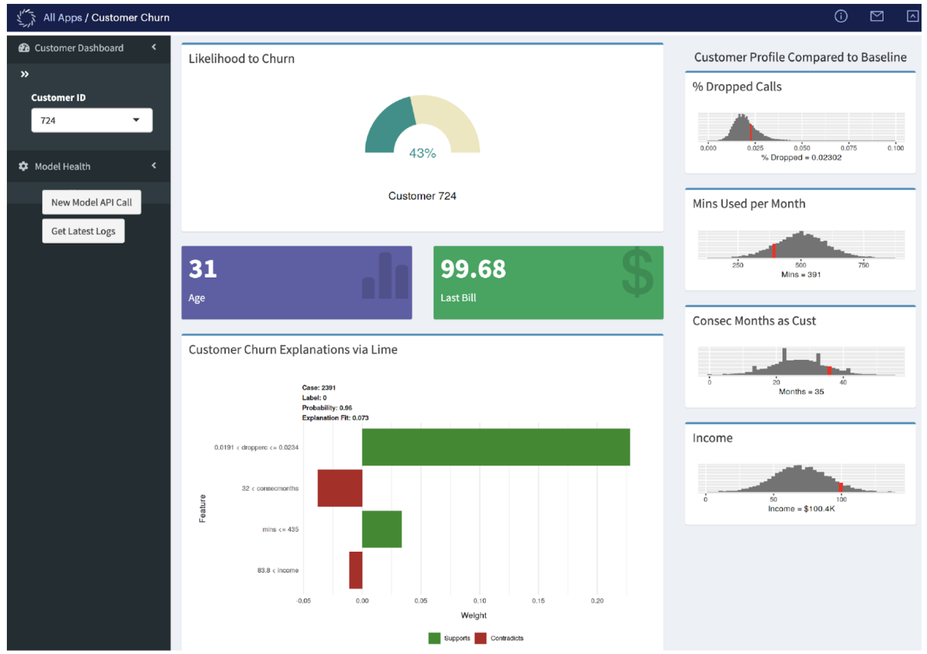
Track experiments

Domino Data Lab

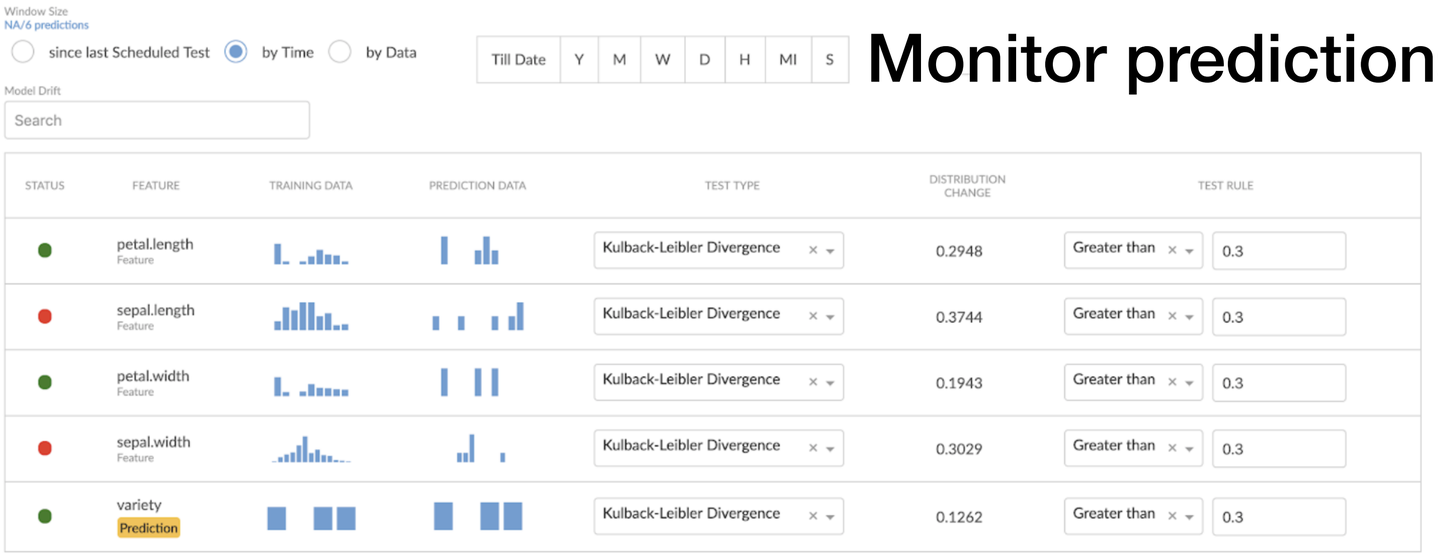
Deploy REST API



Publish applets

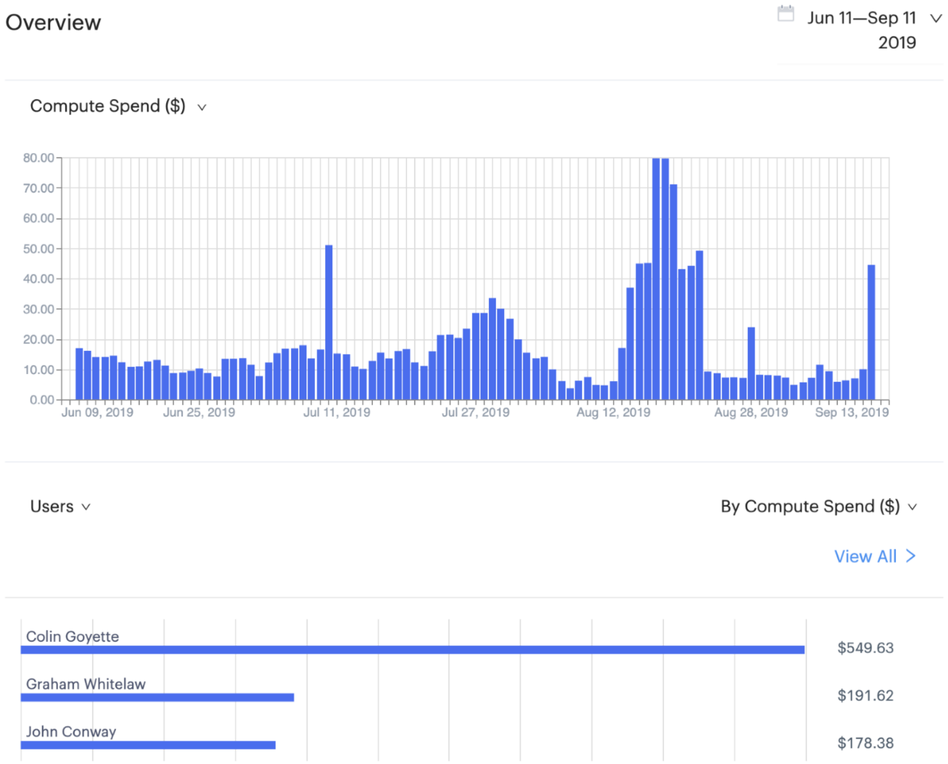


Monitor predictions

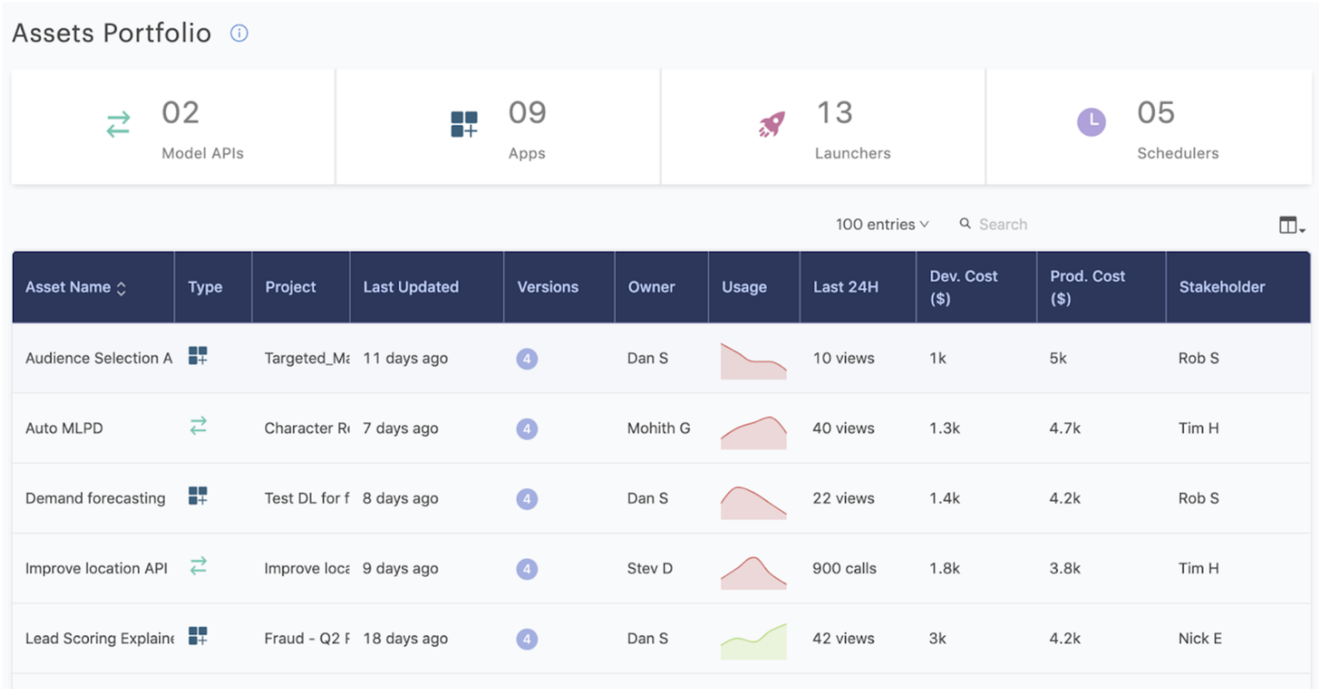


Domino Data Lab

Monitor spend



All projects in one place



	Neptune	Paperspace Gradient	W&B	Comet.ml	Floyd	Algorithmia	Determined.ai	Amazon SageMaker	GC ML Engine	Domino Data Lab
Hardware	GCP or agnostic	Paperspace	N/A	N/A	GCP	N/A	Agnostic	AWS	GCP	Agnostic
Resource Management	Yes	Yes	No	No	Yes	No	Yes* (but not provisioning)	Yes	Yes	Yes
Hyperparam Optimization	Yes	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes
Storing Models	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Reviewing Experiments	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Deploying Models as REST API	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Monitoring	No	No	No	No	No	Yes	No	Yes	Yes	Yes

Questions?

Thank you!