AGENDA

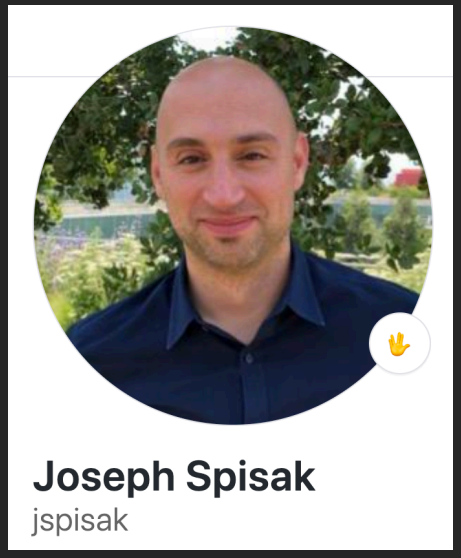
01
WHAT IS PYTORCH?

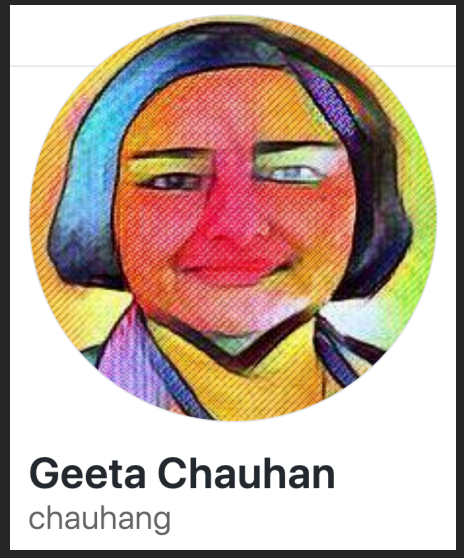
02
RECOMMENDER SYSTEMS USING DLRM

03
INTERPRETABILITY & RECOMMENDER SYSTEMS

04
USING MMF FOR KNOWLEDGE BASED RECOMMENDER SYSTEMS

05
CLOSING THOUGHTS & MATERIALS





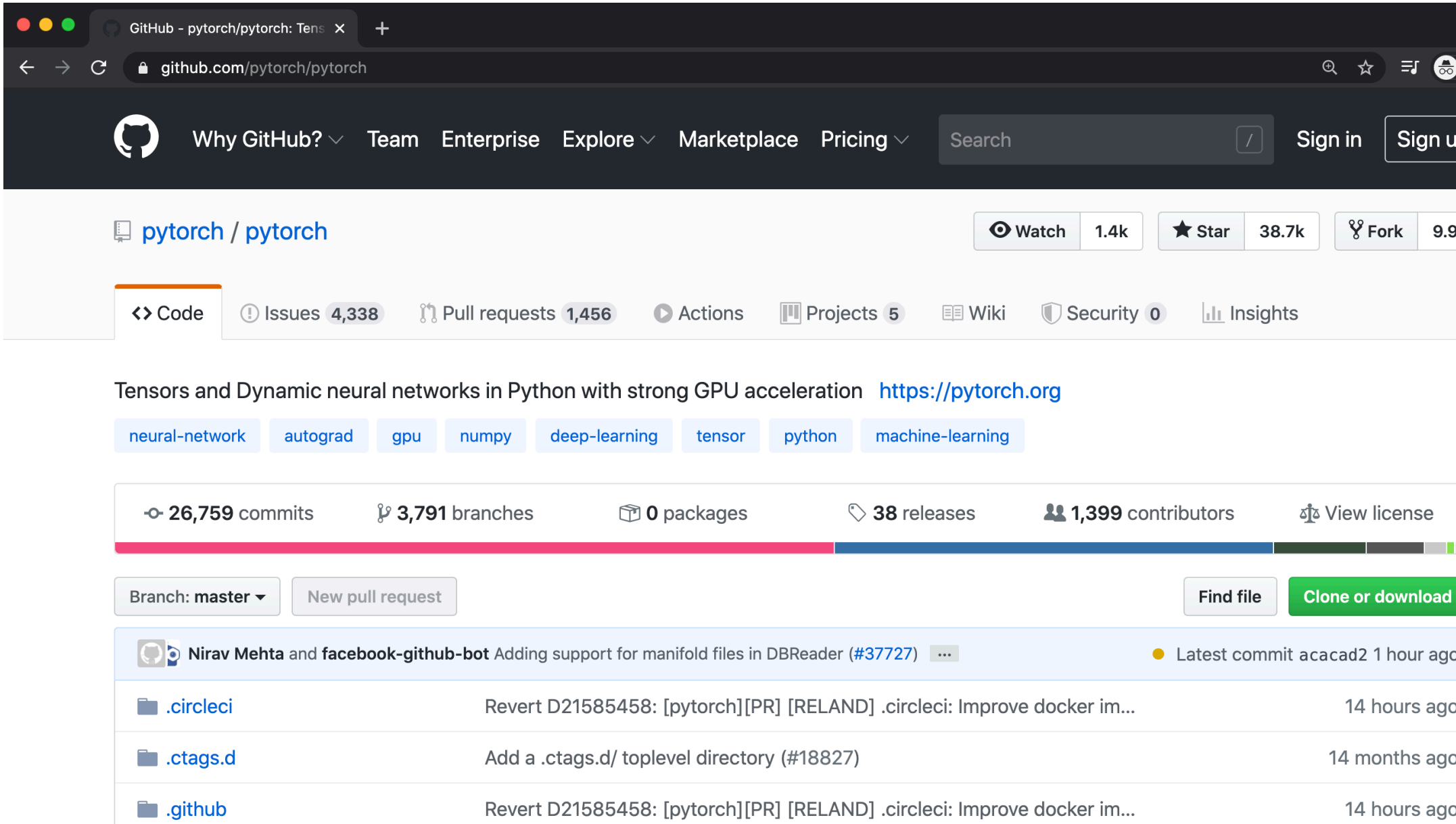
PYTORCH MISSION

RESEARCH
PROTOTYPING

+

PRODUCTION
DEPLOYMENT

OPEN SOURCE, COMMUNITY BASED DEEP LEARNING PLATFORM



torch.nn

NEURAL NETWORKS

torch.optim

OPTIMIZERS

torch.data

DATASETS &
DATA LOADERS

torch.autograd

AUTO
DIFFERENTIATION

torch.vision

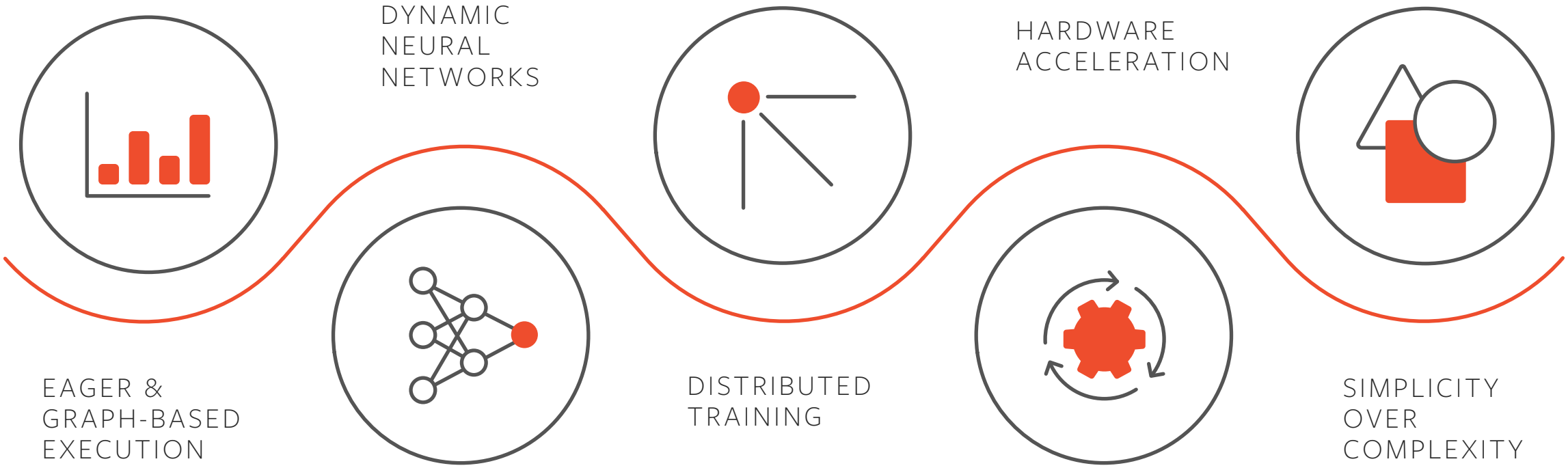
MODELS, DATA

torch.jit

TORCH SCRIPT
DEPLOYMENT

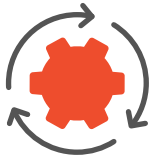
PROJECT PRINCIPLES

DESIGN PRINCIPLES & TENETS



DEVELOPER EXPERIENCE

FULL FREEDOM TO ITERATE AND EXPLORE THE DESIGN SPACE



CLEAN AND INTUITIVE APIS

A RICH ECOSYSTEM OF TOOLS AND LIBRARIES

```python
import torch

class Net(torch.nn.Module):
    def __init__(self):
        self.fc1 = torch.nn.Linear(8, 64)
        self.fc2 = torch.nn.Linear(64, 1)


    def forward(self, x):
        x = torch.relu(self.fc1.forward(x))
        x = torch.dropout(x, p=0.5)
        x = torch.sigmoid(self.fc2.forward(x))
        return x
```

PYTHON

```cpp
#include <torch/torch.h>

struct Net : torch::nn::Module {
  Net() : fc1(8, 64), fc2(64, 1) {
    register_module("fc1", fc1);
    register_module("fc2", fc2);
  }

  torch::Tensor forward(torch::Tensor x) {
    x = torch::relu(fc1->forward(x));
    x = torch::dropout(x, /*p=*/0.5);
    x = torch::sigmoid(fc2->forward(x));
    return x;
  }

  torch::nn::Linear fc1, fc2;
};
```

C++

```python
net = Net()

data_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./data'))

optimizer = torch.optim.SGD(net.parameters())

for epoch in range(1, 11):
    for data, target in data_loader:
        optimizer.zero_grad()
        prediction = net.forward(data)
        loss = F.nll_loss(prediction, target)
        loss.backward()
        optimizer.step()
    if epoch % 2 == 0:
        torch.save(net, "net.pt")
```

PYTHON

```cpp
Net net;

auto data_loader = torch::data::data_loader(
  torch::data::datasets::MNIST("./data"));

torch::optim::SGD optimizer(net.parameters());

for (size_t epoch = 1; epoch <= 10; ++epoch) {
  for (auto batch : data_loader) {
    optimizer.zero_grad();
    auto prediction = net.forward(batch.data);
    auto loss = torch::nll_loss(prediction,
                                batch.label);
    loss.backward();
    optimizer.step();
  }
  if (epoch % 2 == 0) {
    torch::save(net, "net.pt");
  }
}
```

C++

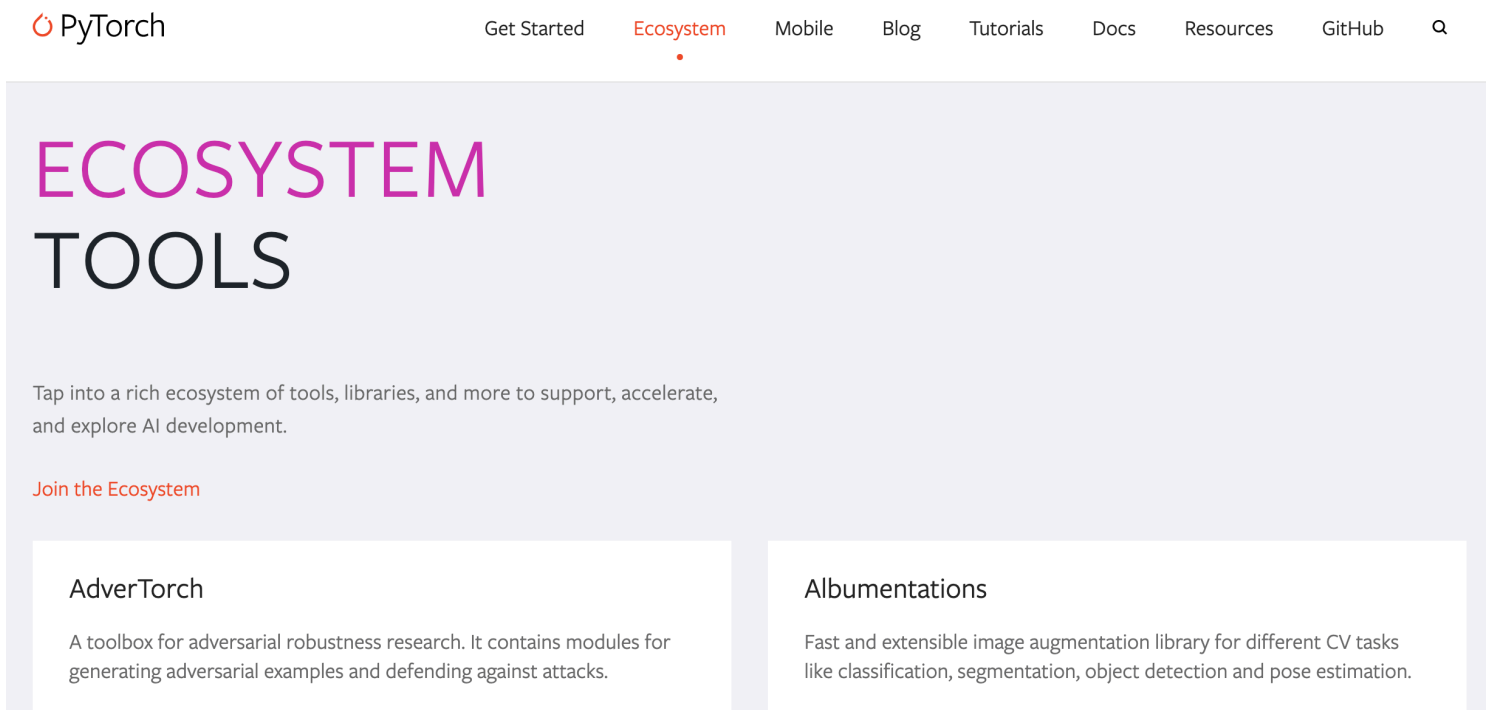
TOOLS & LIBRARIES

Ax | BoTorch

PyTorch3D

CrypTen

Captum



COMMUNITY

Alfredo Canziani
@alfcnz

Atcold/pytorch-Deep-Learning-Minicourse

ing with PyTorch. Contribute to
rning-Minicourse develpment
GitHub.

Smerity
@smerity

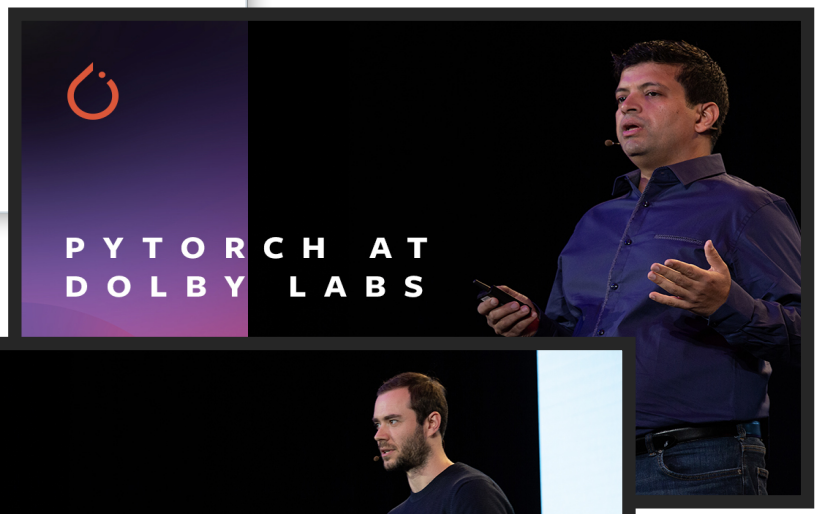
True to their mission, the @PyTorch community focused on solving the issues of eager mode w/o impacting
erability?
Want
ll without

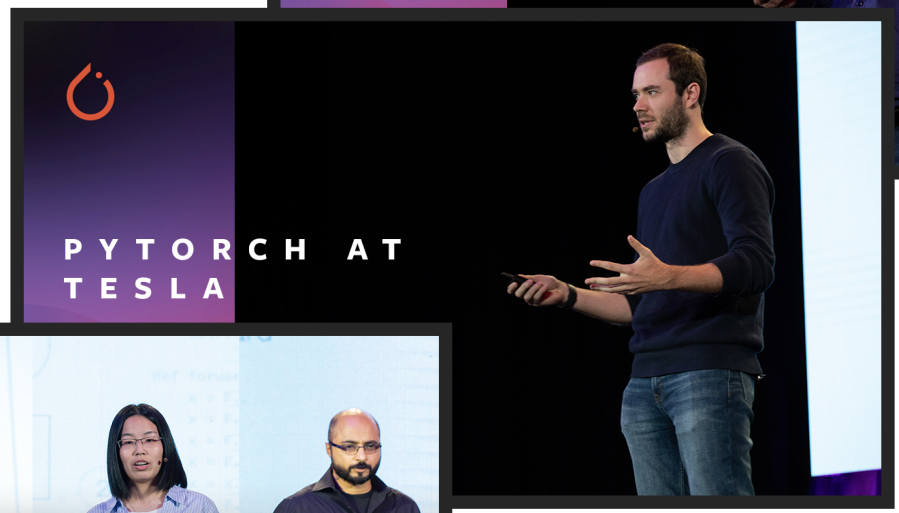
2 Oct 2018

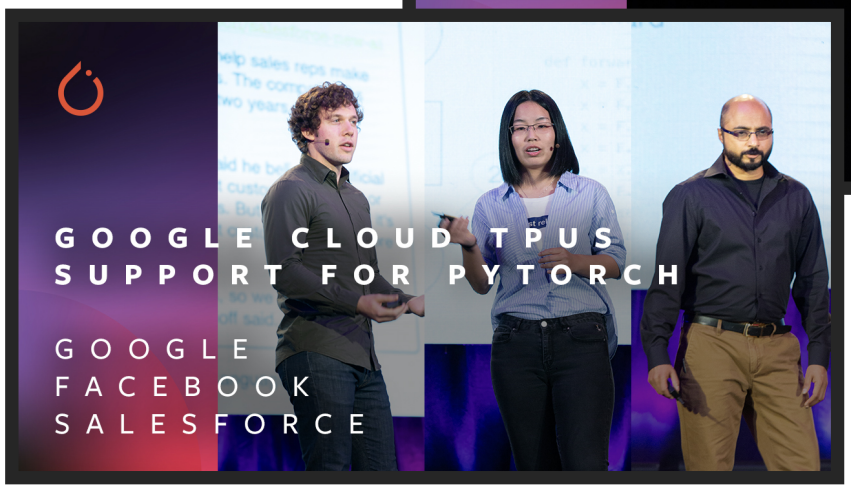
Jeremy Howard
@jeremyphoward

At the @PyTorch developer conference, I was part of a fascinating panel with @clattner_llvm, Yangqing Jia, and Noah Goodman, Expertly moderated by @soumithchintala. Here it is!

10 Oct 2018







O'REILLY® Ideas Learning Platform Conferences Shop

ON OUR RADAR AI DATA ECONOMY OPERATIONS SOFTWARE ARCHITECTURE SOFTWARE ENGINEERING WEB PROGRAMMING

AI

One simple graphic: Researchers love PyTorch

Preferred Networks

Home > News > Preferred Networks Migrates its Deep Learning Research Platform to PyTorch

News

tutorials that cover PyTorch, TensorFlow,
ntelligence conference in San Jose,

rise, which drew more than
age of several machine learning
dicated they used TensorFlow
re using PyTorch or Keras.

resentation delivered by
AutoPandas and automation
course of her presentation,

Research lab
(source: SMU Libraries D

ABOUT PROGRESS RESEARCH BLOG

JANUARY 30, 2020 • 1 MINUTE READ

OpenAI → PyTorch

We are standardizing OpenAI's deep learning framework on PyTorch. In the past, we implemented projects in many frameworks depending on their relative strengths. We've now chosen to standardize to make it easier for our team to create and share optimized implementations of our models.

Share

O: Toru
al
borate
ment of
PFN

~1,491

CONTRIBUTORS

50%+

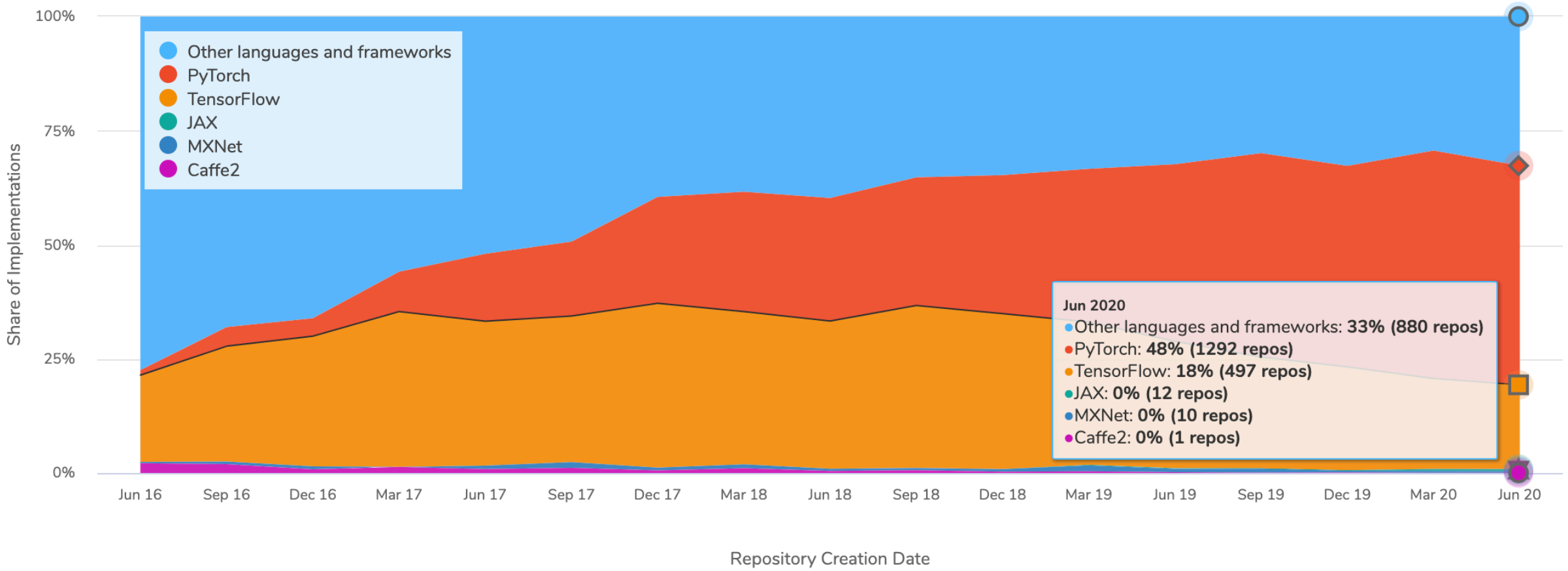
YOY GROWTH

32K+

PYTORCH FORUM USERS

GROWING USAGE IN OPEN SOURCE

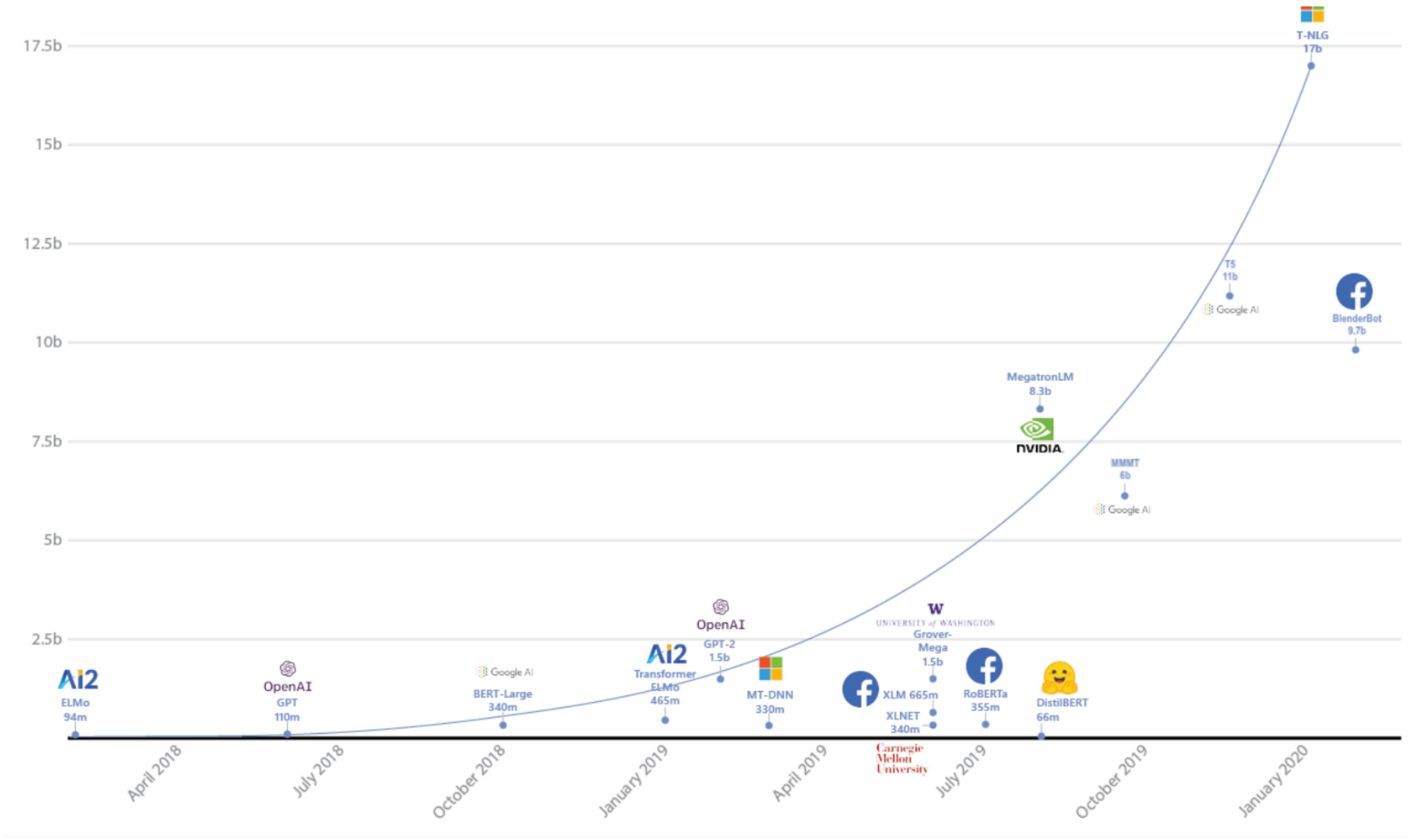
Paper Implementations grouped by framework



Source: https://paperswithcode.com/trends

MODEL SIZE AND COMPUTE NEEDS

PARAMETER COUNTS CONTINUE TO GROW



PRUNING

State-of-the-art deep learning techniques rely on over-parametrized models that are hard to deploy.

Identify optimal techniques to compress models by reducing the number of parameters without sacrificing accuracy.

```
new_model = LeNet()
for name, module in new_model.named_modules():
   # prune 20% of connections in all 2D-conv layers
   if isinstance(module, torch.nn.Conv2d):
      prune.l1_unstructured(module, name='weight', amount=0.2)
   # prune 40% of connections in all linear layers
   elif isinstance(module, torch.nn.Linear):
      prune.l1_unstructured(module, name='weight', amount=0.4)

 # to verify that all masks exist
print(dict(new_model.named_buffers()).keys())
```

QUANTIZATION

EXPERIMENTAL

Neural networks inference is expensive

IoT and mobile devices have limited resources

Quantizing models enables efficient inference at scale

```
model = ResNet50()
model.load_state_dict(torch.load("model.pt"))

qmodel = quantization.prepare(
        model, {"": quantization.default_qconfig})

qmodel.eval()
for batch, target in data_loader:
        model(batch)

qmodel = quantization.convert(qmodel)
```

EXAMPLE MODELS | W/ QUANTIZATION

	fp32 accuracy	int8 accuracy change	Technique	CPU inference speed up
ResNet50	76.1 Top-1, Imagenet	-0.2 75.9	Post Training	2x 214ms ➞102ms, Intel Skylake-DE
MobileNetV2	71.9 Top-1, Imagenet	-0.3 71.6	Quantization-Aware Training	4x 75ms ➞18ms OnePlus 5, Snapdragon 835
Translate / FairSeq	32.78 BLEU, IWSLT 2014 de-en	0.0 32.78	Dynamic (weights only)	4x for encoder Intel Skylake-SE

These models and more available on TorchHub - https://pytorch.org/hub/

TRAINING MODELS AT SCALE

EXPENSIVE & EXPERIMENTAL

Unlike data pipelines, majority of ML jobs are ad hoc

- Long running jobs complicate demand prediction & control

Cost & Efficiency

- ROI for jobs hard to estimate
- Sub-linear scaling + huge scale = an easy way to waste resources

PYTORCH ELASTIC

Enables users to write fault tolerant and elastic distributed PyTorch jobs

Use cases

Fault tolerance

Run on non-HPC cluster (e.g. cloud)

Mission critical production job

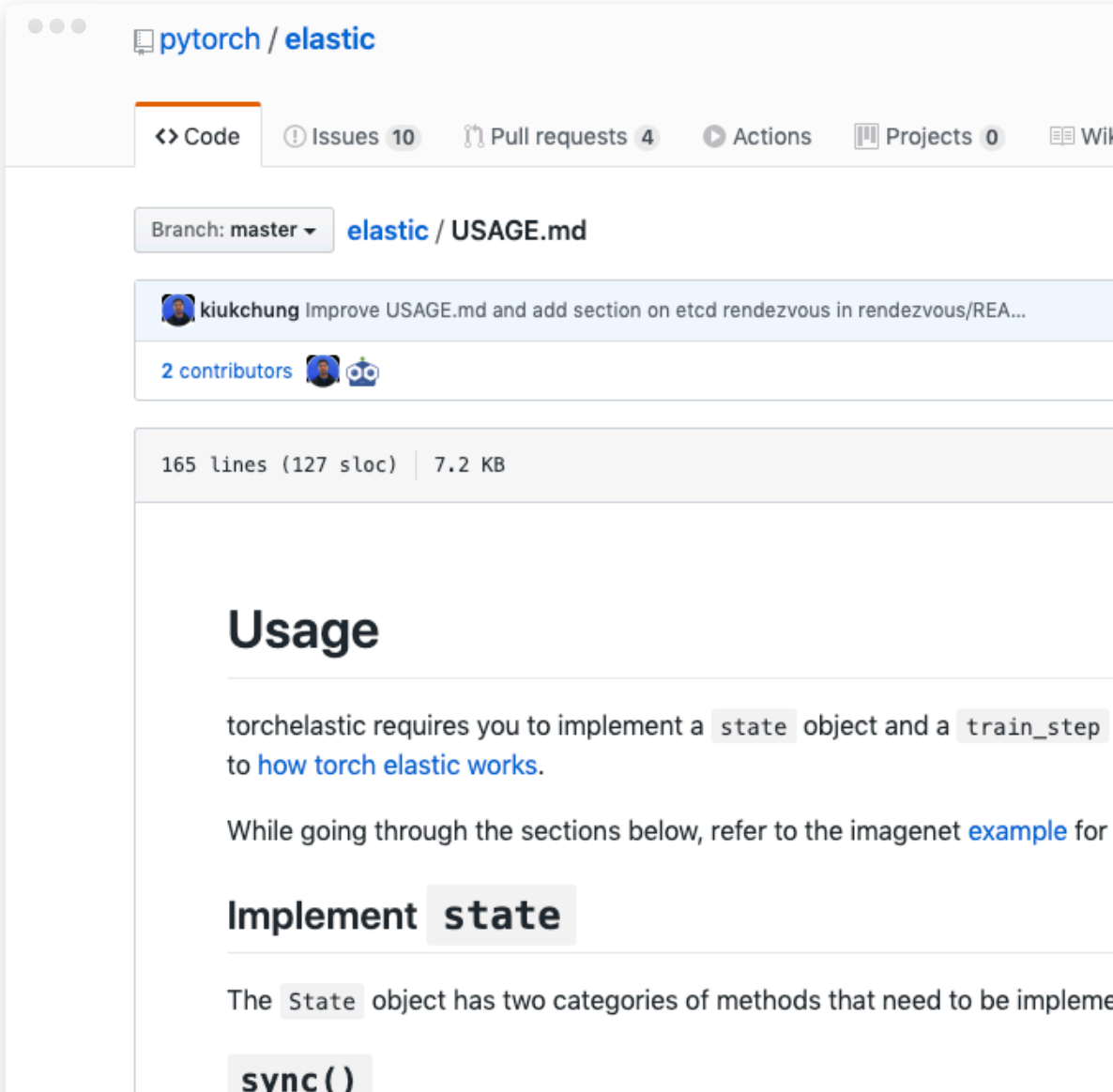
Dynamic Capacity Management:

Run on leased capacity that can be preempted (e.g. AWS spot instances)

Shared pools where the pool size can change dynamically based on demand (e.g. autoscaling)

Open sourced 0.1.0.rc - https://github.com/pytorch/elastic

Integrated with Classy Vision - https://classyvision.ai/tutorials/pet_aws



REALLY LARGE SCALE

Tens of billions of training examples

Some models larger than size of machine

Hundreds of experimental runs competing for resources

More than 1000+ different production models

PYTORCH RPC (REMOTE PROCEDURE CALL)

Enables applications to run functions remotely, and automatically handles autograd if necessary.

Use cases

Scale out applications

- Parameter Server Framework
- In RL, multiple observers streaming states and rewards to the agent for policy training

Distributed model parallelism

- Allow large models to span multiple machines
- Hogwild! training

PYTORCH RPC COMPONENTS

RPC

Run user code with given args on the specified destination

Remote Reference (RRef)

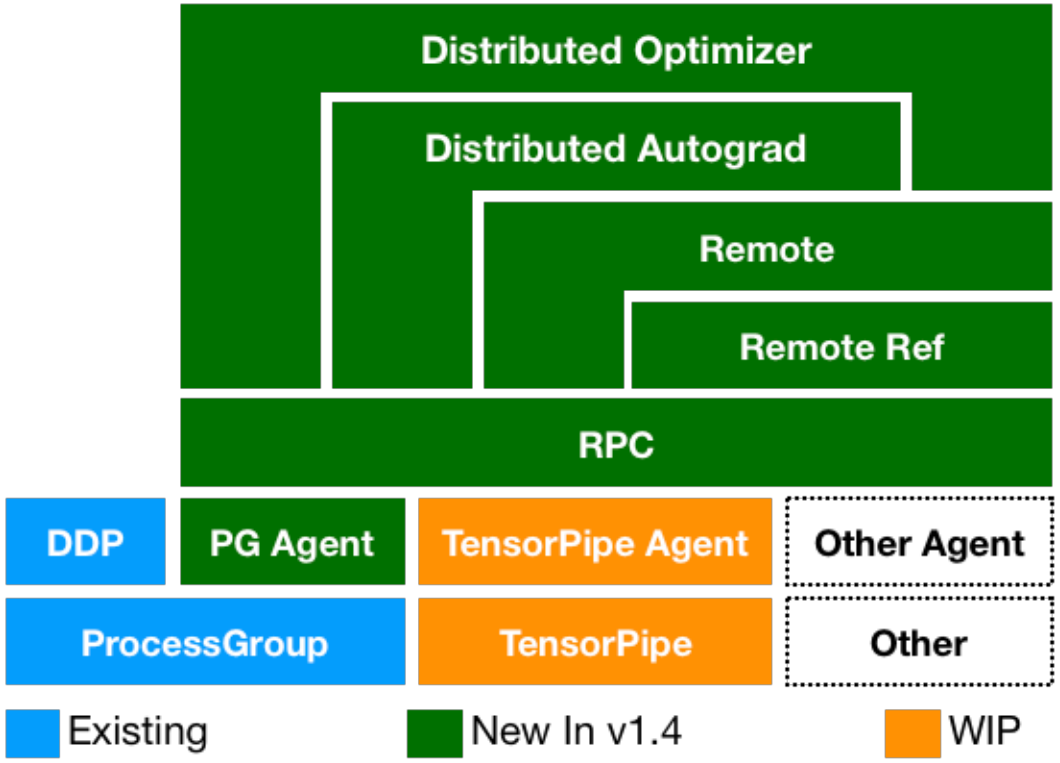
Tracks and maintains objects owned by a remote worker.

Distributed Autograd

Connects autograd graphs on different workers into one global graph and provides a similar backward API.

Distributed Optimizer

Automatically handles RRef of subnets and expose a similar API as local optimizers.



DEPLOYMENT AT SCALE

INFERENCE AT SCALE

Deploying and managing models in production is difficult.

Some of the pain points include:

Loading and managing multiple models, on multiple servers or end devices

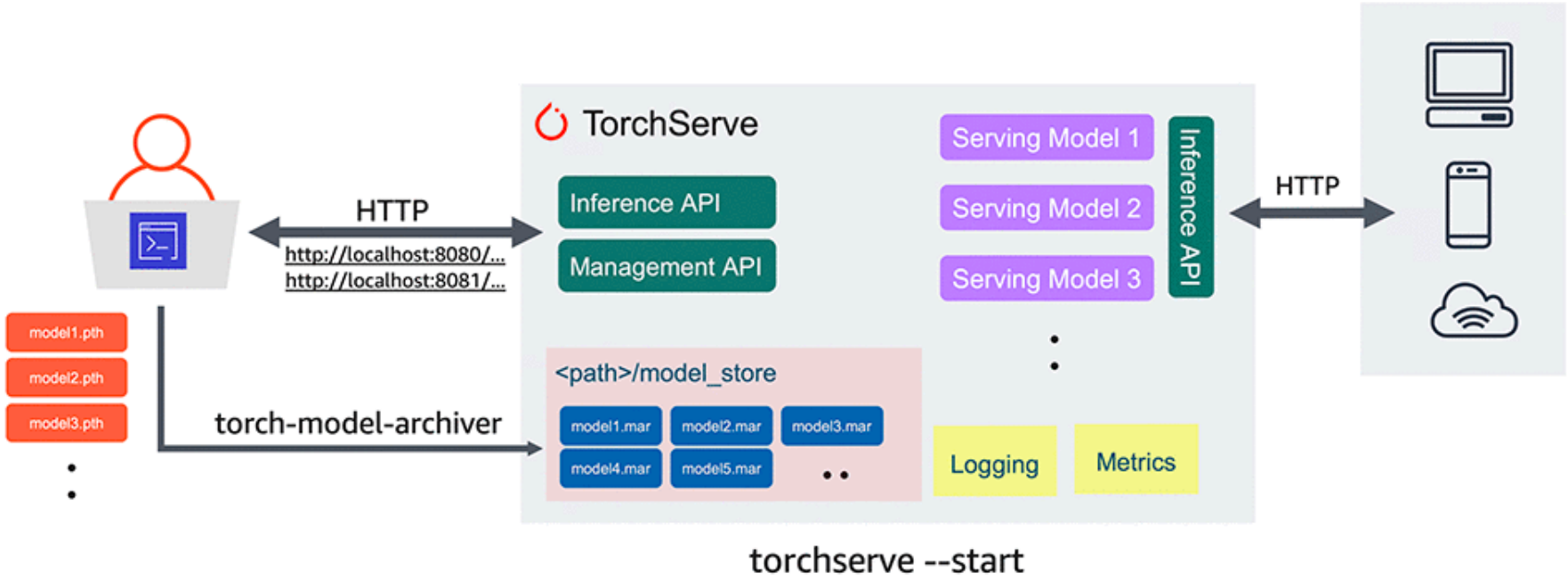
Running pre-processing and post-processing code on prediction requests.

How to log, monitor and secure predictions

What happens when you hit scale?

TORCHSERVE

EXPERIMENTAL



- Default handlers for common use cases (e.g., image segmentation, text classification) along with custom handlers support for other use cases
- Model versioning and ability to roll back to an earlier version
- Automatic batching of individual inferences across HTTP requests
- Logging including common metrics, and the ability to incorporate custom metrics
- Robust HTTP APIS - Management and Inference

THE LATEST | GET STARTED

PYTORCH 1.6 RELEASED - END OF JULY

- Automatic mixed precision (AMP) training is now natively supported and a stable feature - thanks for NVIDIA's contributions;
- Native TensorPipe support now added for tensor-aware, point-to-point communication primitives built specifically for machine learning;
- New profiling tools providing tensor-level memory consumption information; and
- Numerous improvements and new features for both distributed data parallel (DDP) training and the remote procedural call (RPC) packages.

DOMAIN LIBRARIES - SHIPPED WITH PYTORCH 1.6

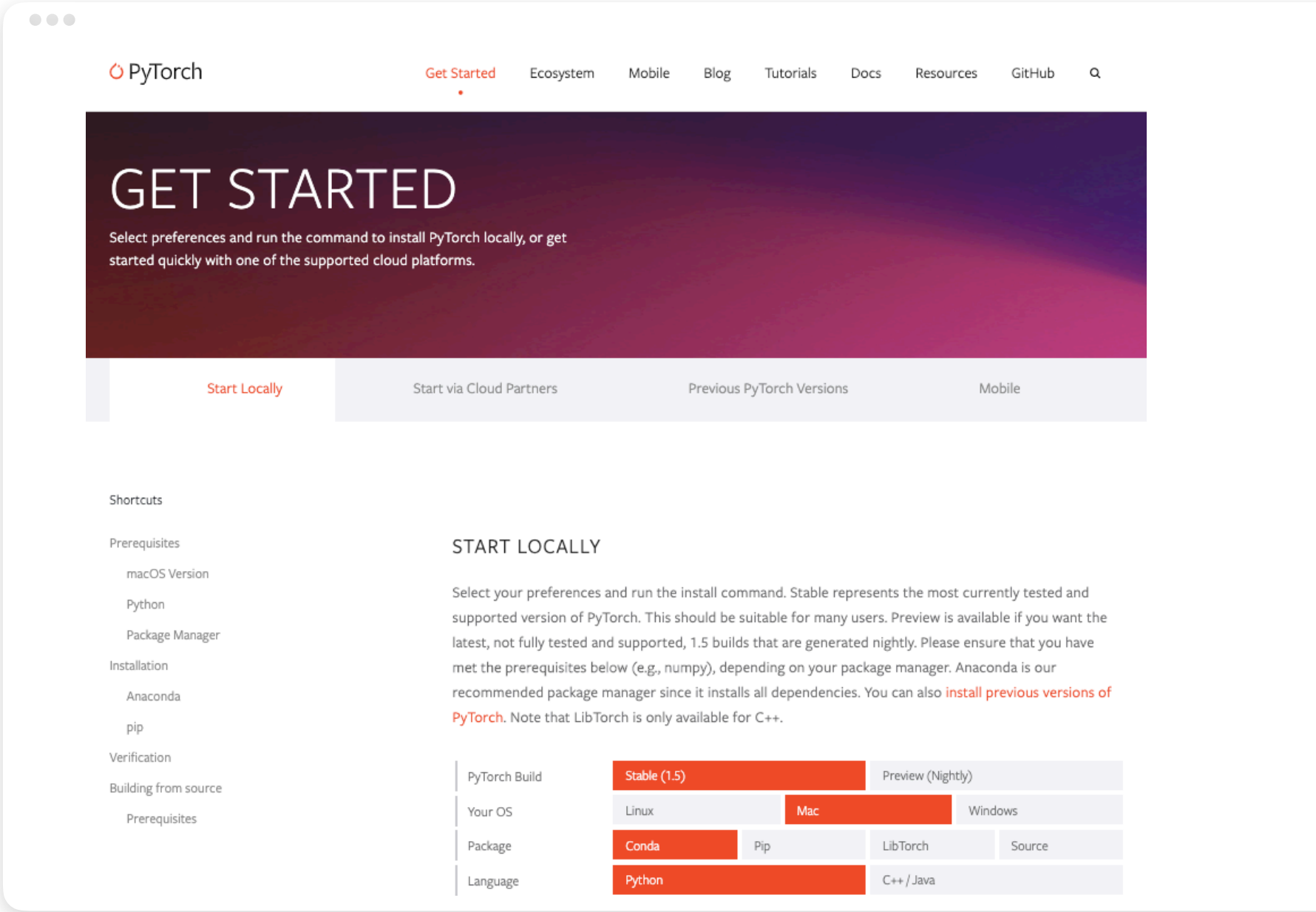
torchvision 0.7

- New pretrained models - Segmentation models FCN ResNet50 and DeepLabV3 ResNet50, trained on COCO.
- Added support for AMP (Automatic Mixed Precision) autocasting for torchvision models and operators

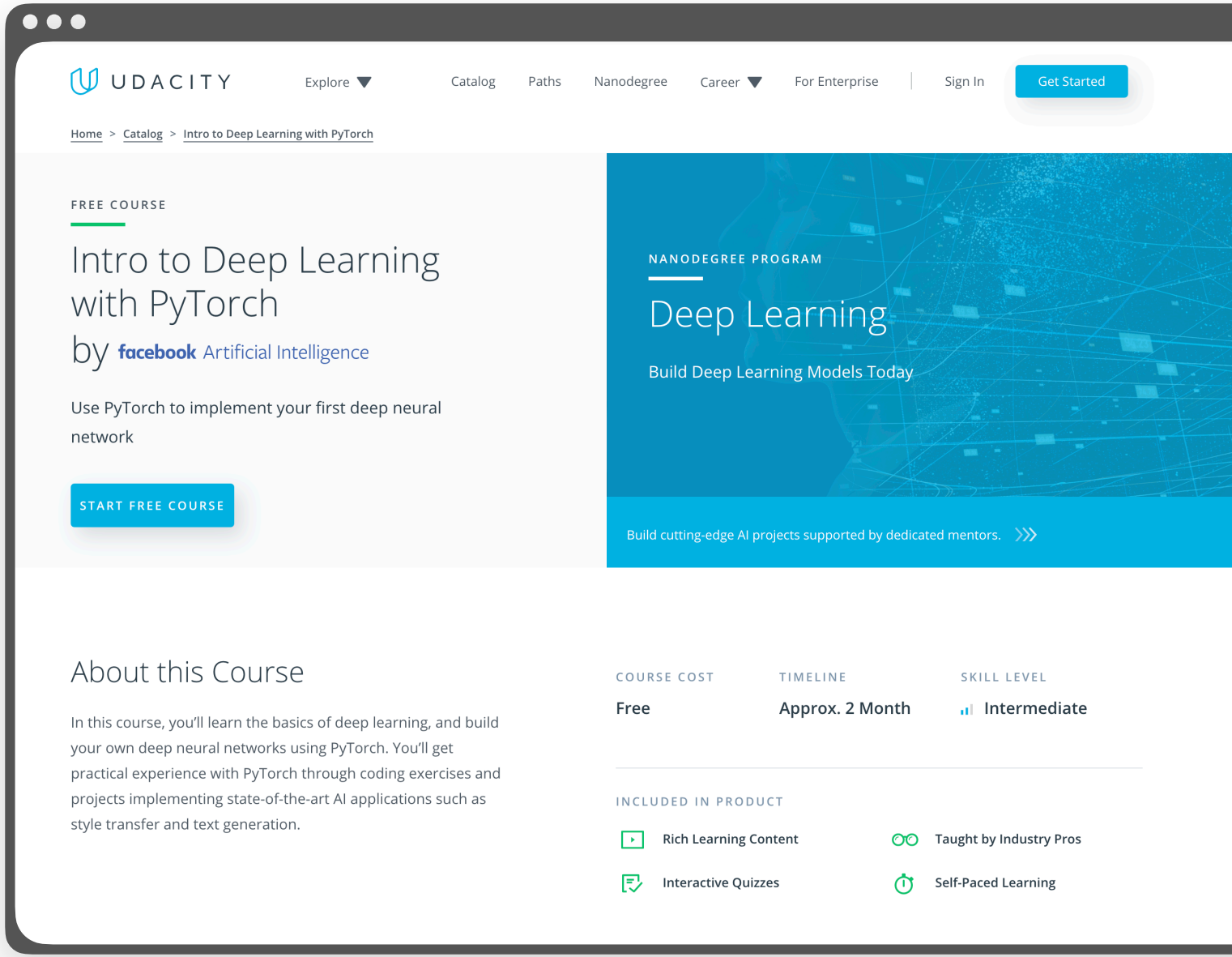
- **torchtext 0.7**
  - Experimental overhaul of torchtext's datasets - easier, compatible PyTorch's DataLoader and Sampler, and simpler out-of-the-box with preprocessing defaults.
  - A new MultiheadAttention module which provides more flexibility for innovative transformer variants.
  - Windows is now officially supported.
  - The SentencePiece models have been extended with torchscript support.
  - A pre-trained BERT example pipeline, along with a question answering fine-tuning example and two new Q&A datasets.
- **torchaudio 0.6**
  - Official support for Windows.
  - New model module (with wav2letter included), new functionals (contrast, cvm, dcshift, overdrive, vad, phaser, flanger, biquad), datasets (GTZAN, CMU), and a new optional sox backend with support for torchscript.

GET STARTED

PYTORCH.ORG

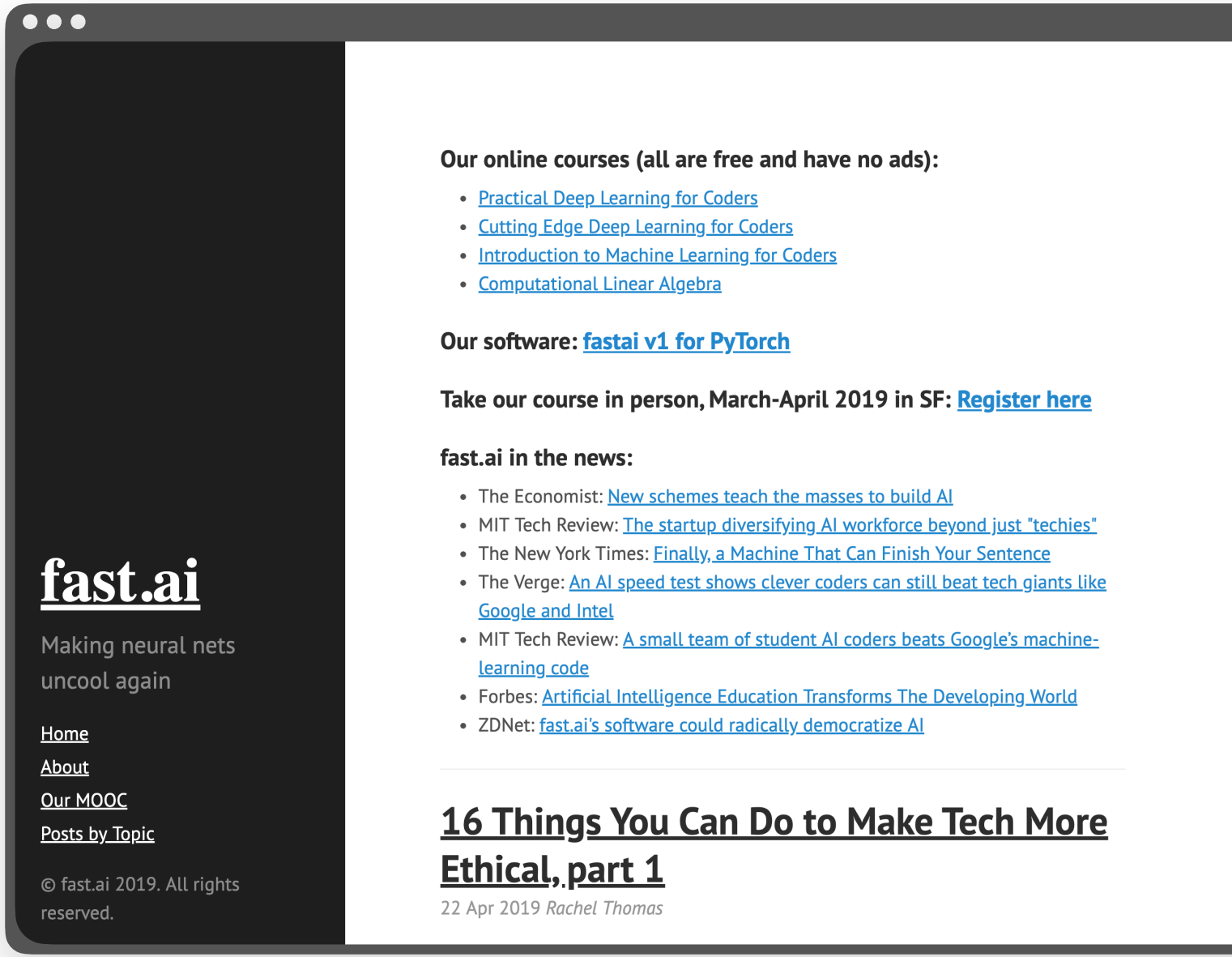


EDUCATIONAL RESOURCES




O'REILLY
Natural Language Processing with PyTorch
Build Intelligent Language A
Using Deep Learning
Deep Learning with PyTorch
Eli Stevens
Luca Antiga
MEAP
MANNING

EDUCATIONAL RESOURCES

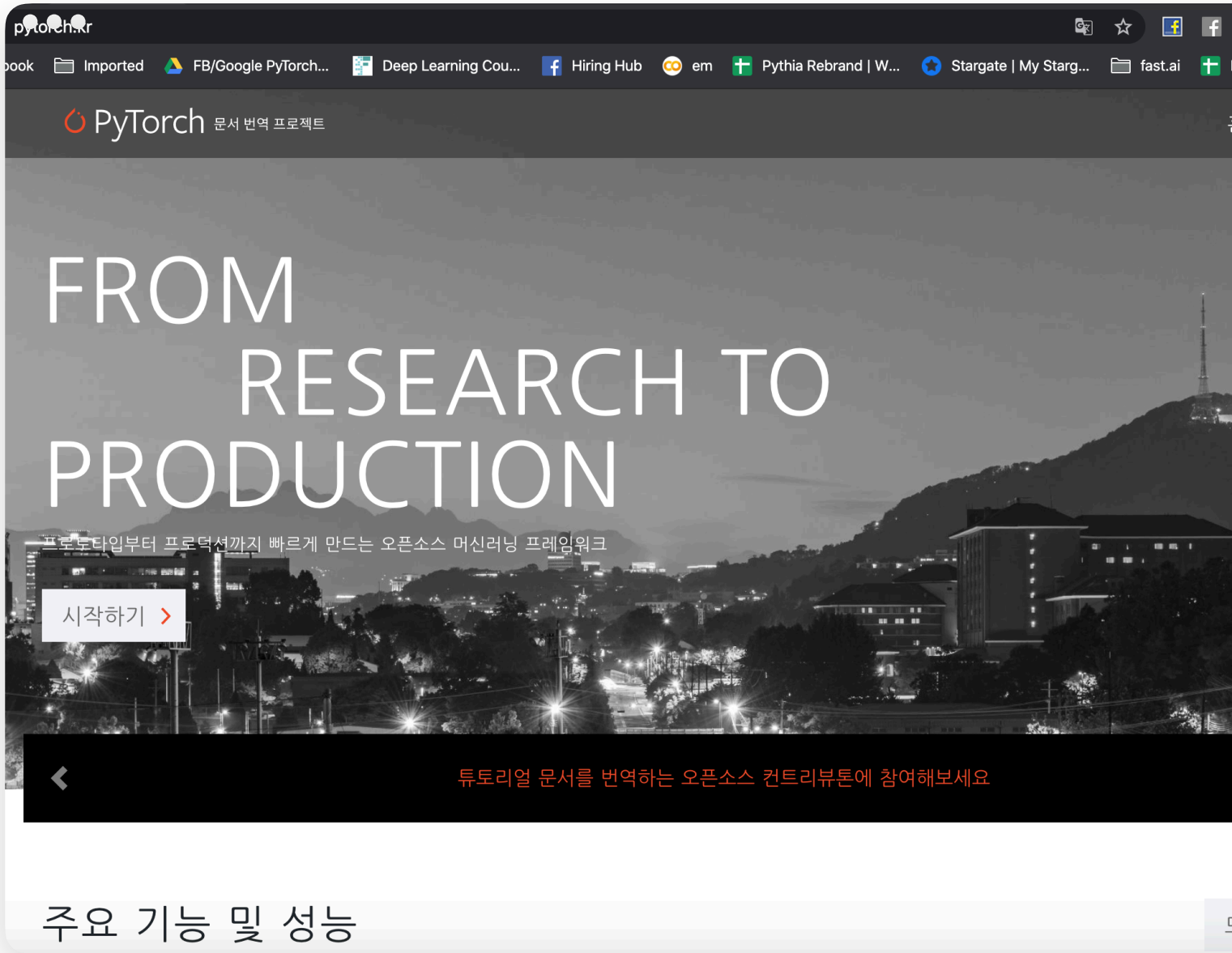


ALSO IN CHINESE :)

https://pytorch.apachecn.org/

AND KOREAN :) :)

https://pytorch.kr/



JOIN THE PYTORCH DEVELOPER COMMUNITY

PyTorch.org

Twitter.com/pytorch

Youtube.com/pytorch

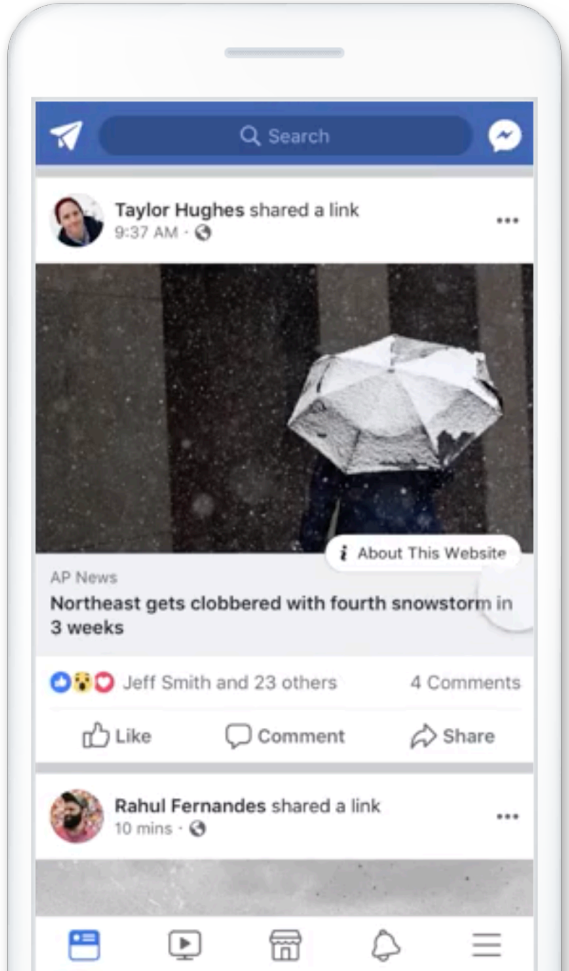
Facebook.com/pytorch

Medium.com/pytorch

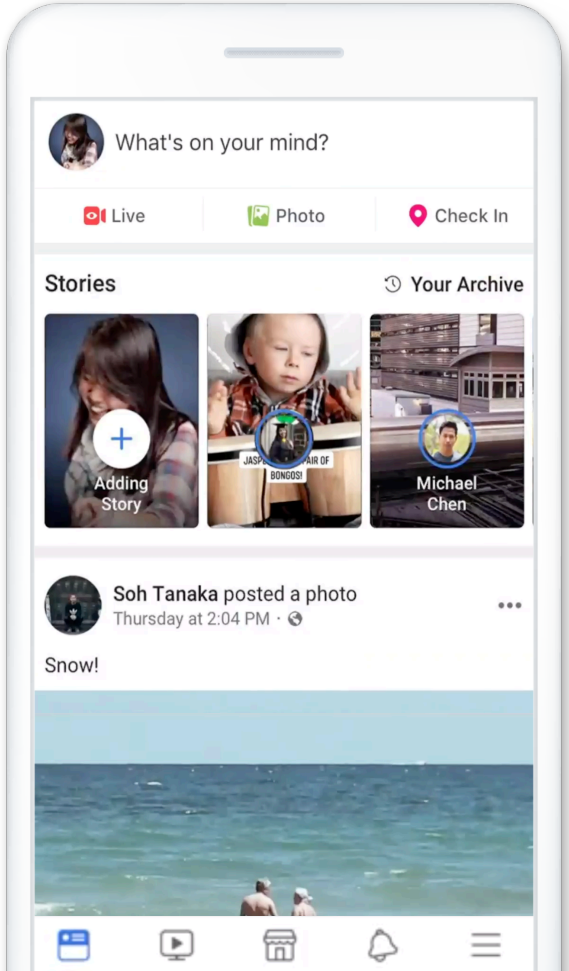
RECOMMENDER SYSTEMS

RECOMMENDATION USE CASES

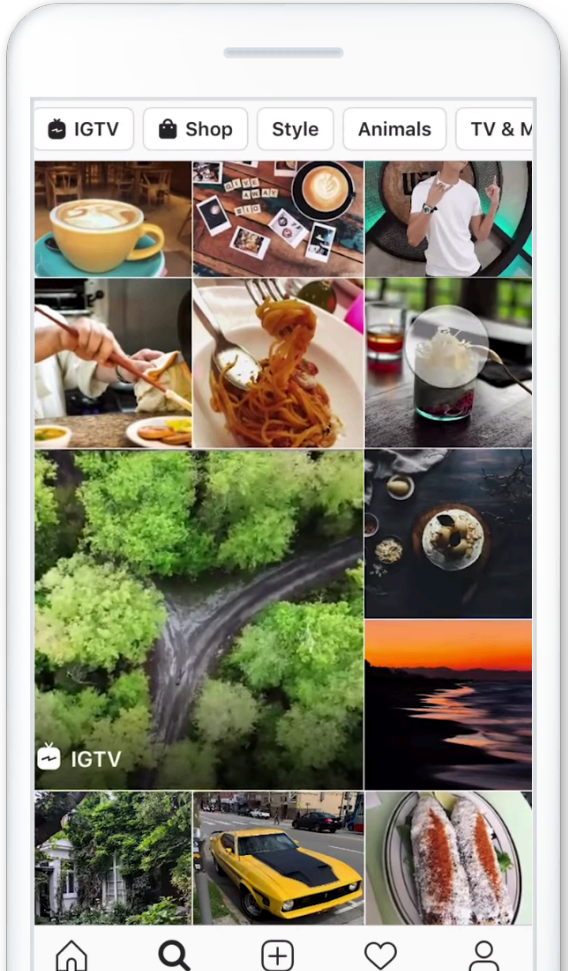
News Feed Ranking



Stories Ranking



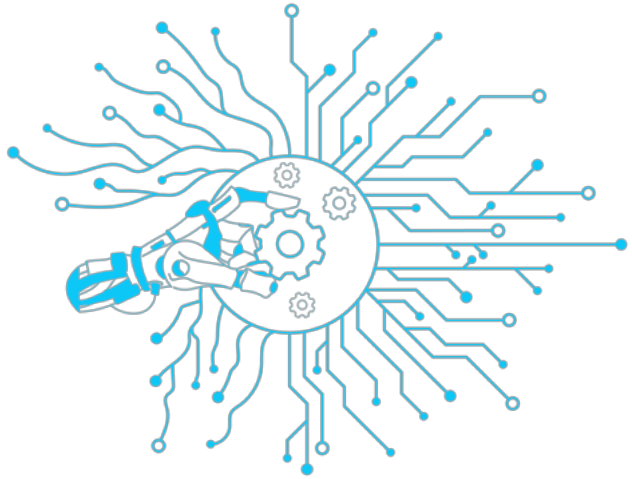
Instagram Explore



50%

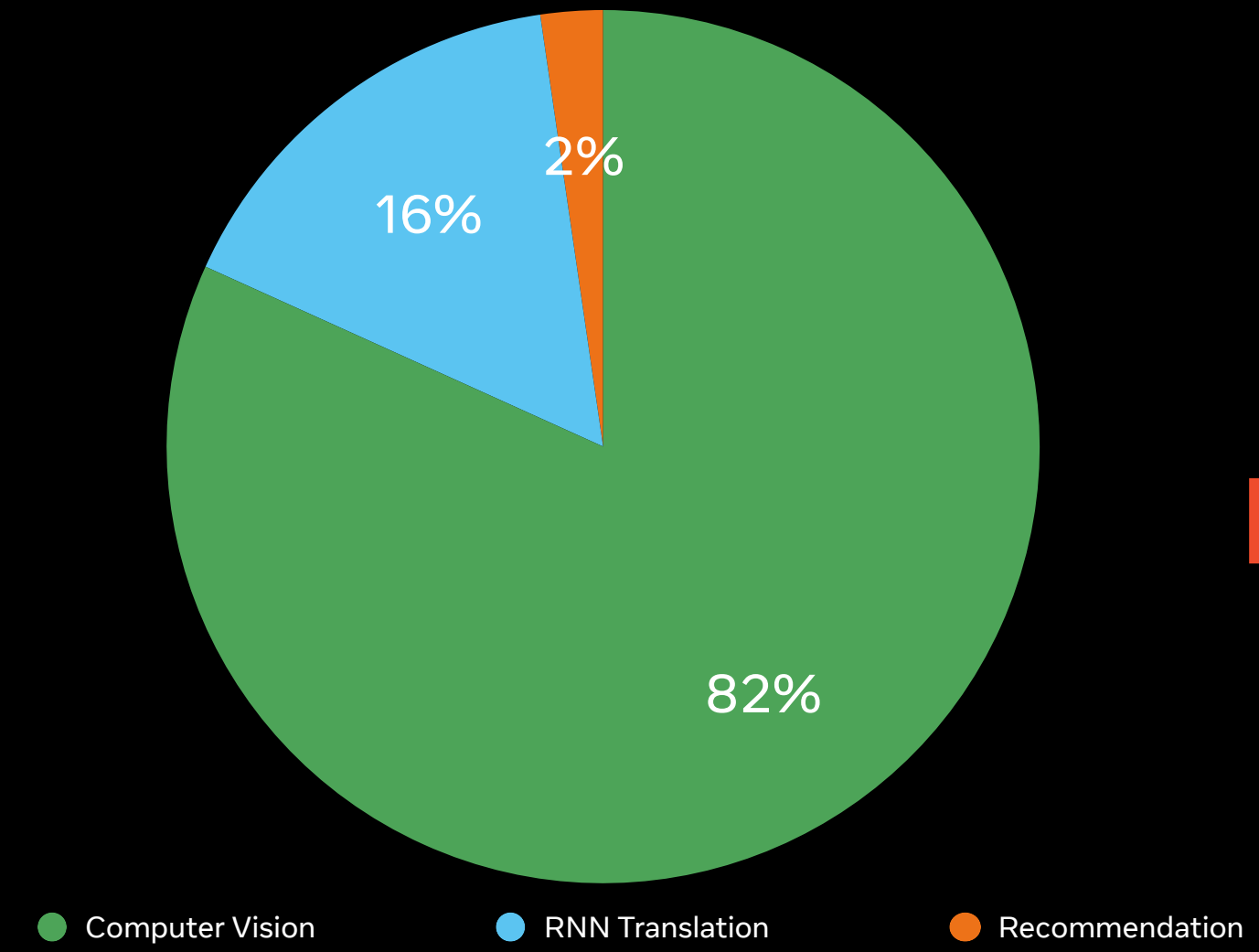
of all AI Training Cycles

80%



of all AI Inference Cycles

PUBLICATIONS - ML SYSTEMS COMMUNITY



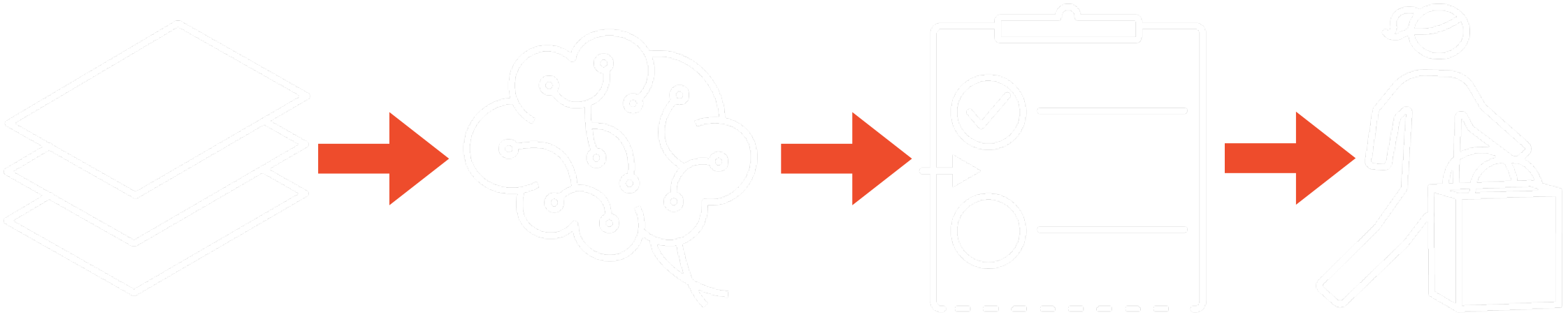
Why The Disconnect?

Candidate Items

Neural Recommendation Model

Recommended Items (Ranked)

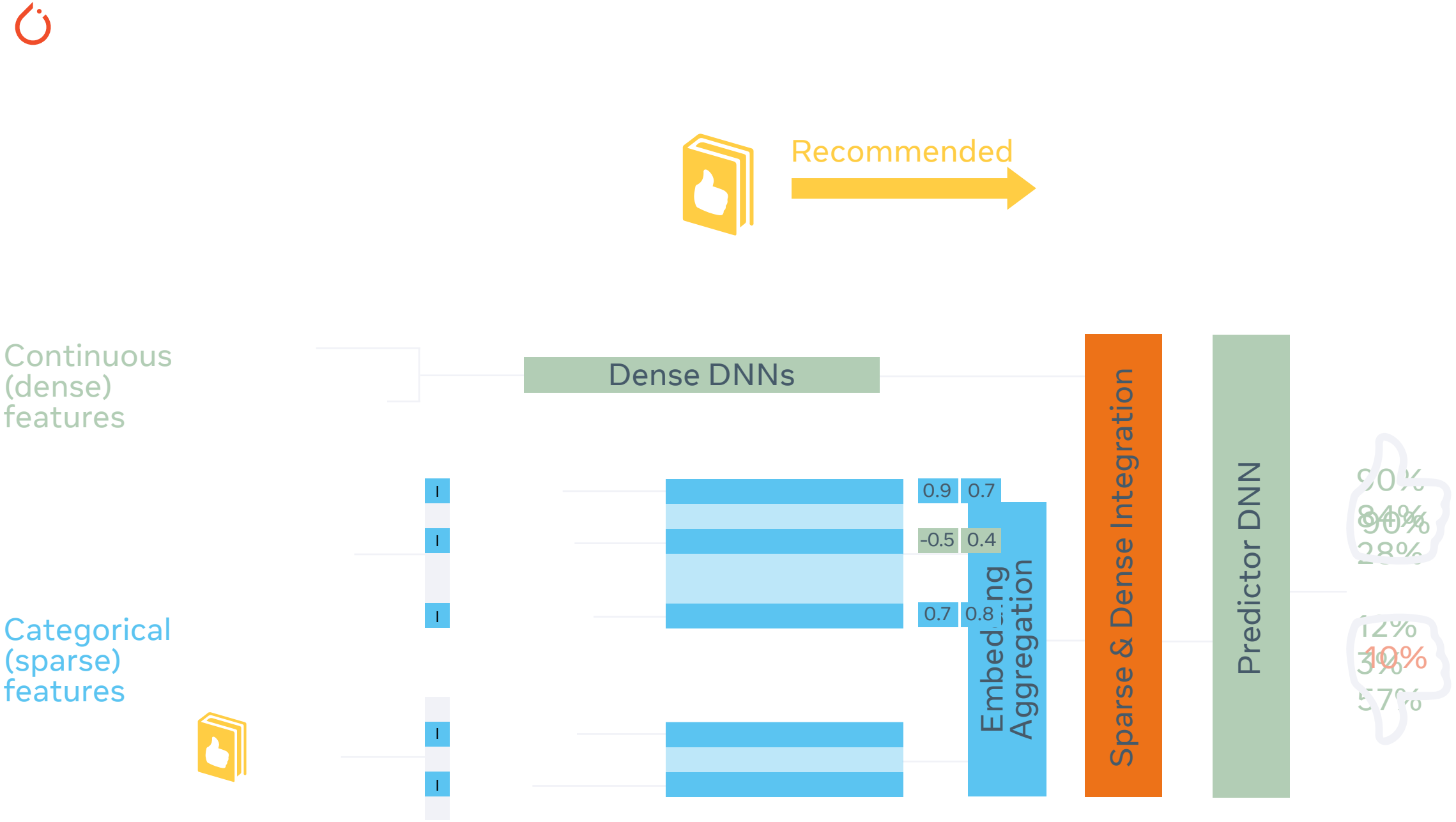
User



Hundreds of Thousands

Billions of Parameters

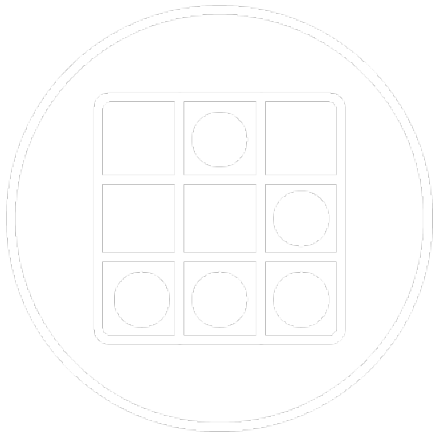
Billions


Recommended
Continuous (dense) features
Dense DNNs
Categorical (sparse) features
0.9
0.7
-0.5
0.4
0.7
0.8
Embedding Aggregation
Sparse & Dense Integration
Predictor DNN
90%
84%
90%
28%
12%
10%
3%
57%

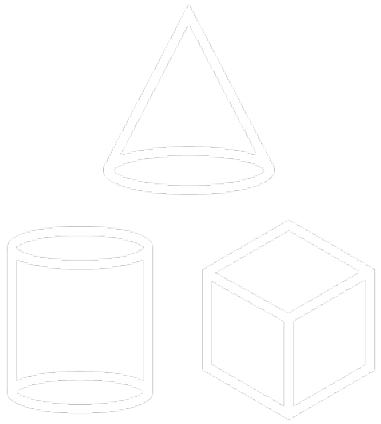
High Throughput

Low Latency

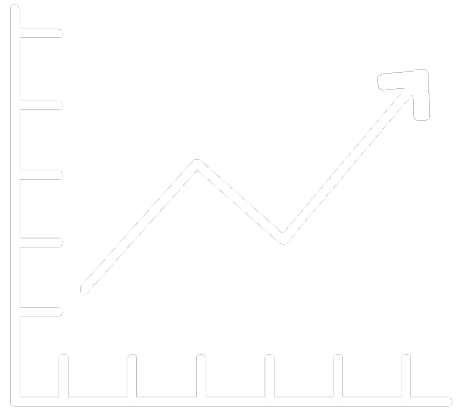
Latency-Bounded Throughput



Embedding Tables



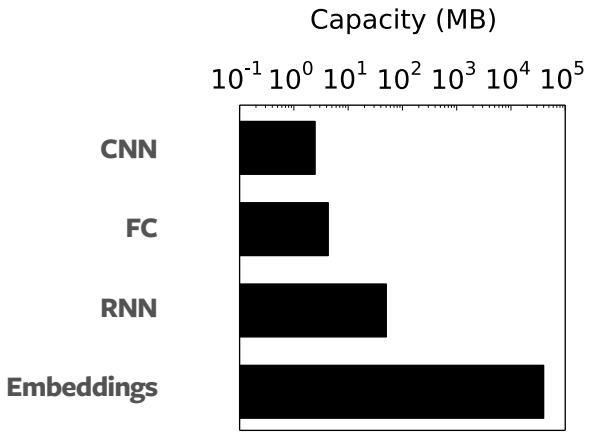
Model Heterogeneity



Performance Variance

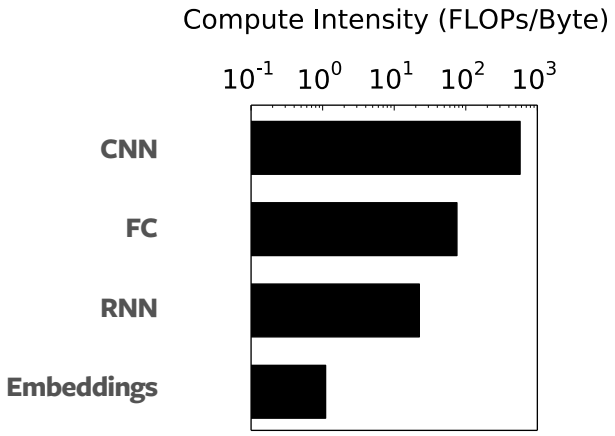
CHALLENGES: EMBEDDING TABLES

Storage Capacity



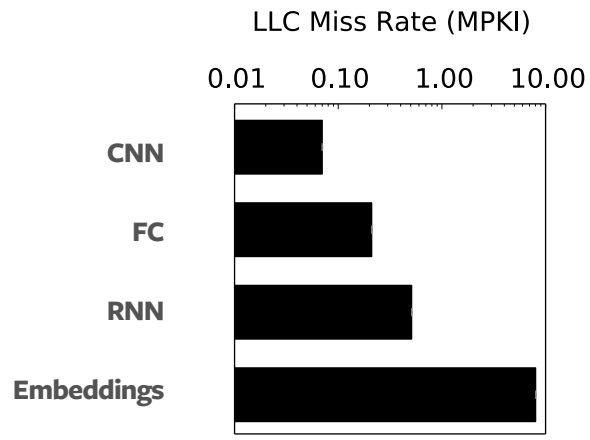
Orders of Magnitude
Larger

Compute Intensity



Orders of Magnitude
Fewer FLOPS/Byte

Memory Irregularity

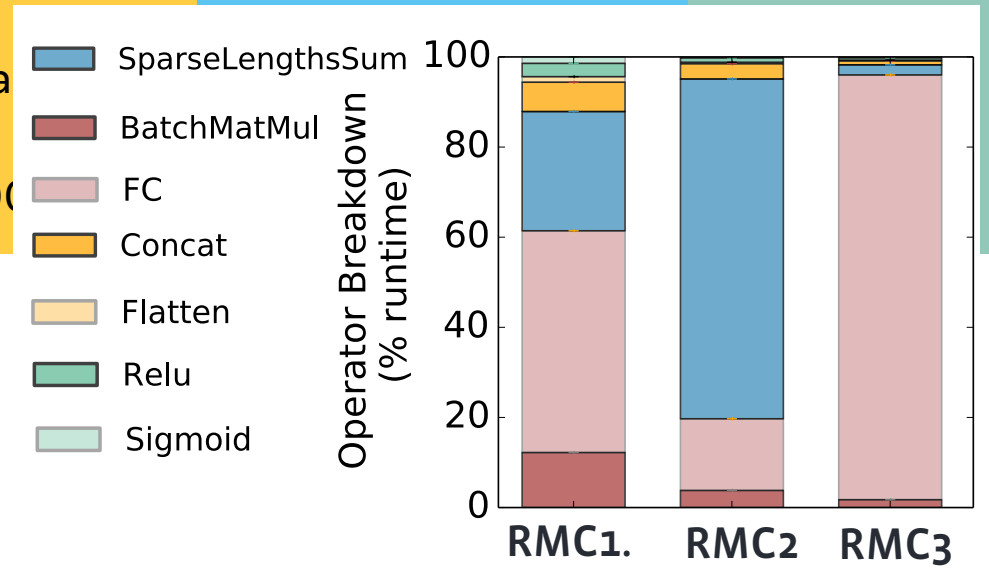


Sparse, Irregular
Memory Accesses

CHALLENGE: MODEL HETEROGENEITY

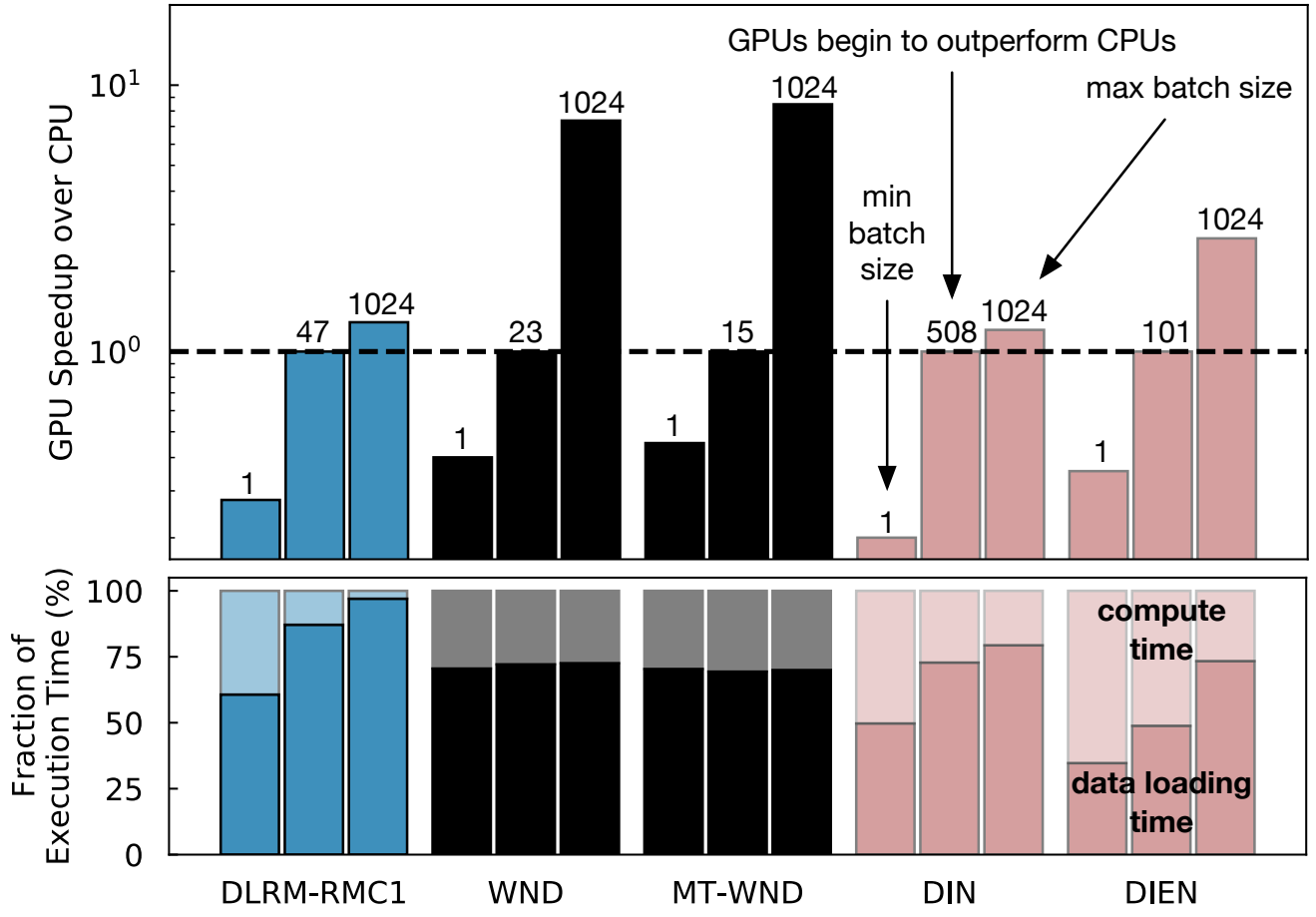
Three Facebook Recommendation Models

	RM1	RM2	RM3
FC sizes	Small	Medium	Large
Number of embedding tables	O(10)	O(50)	O(10)
Size of embeddings	Sma		
Number of lookups per table	O(10(		



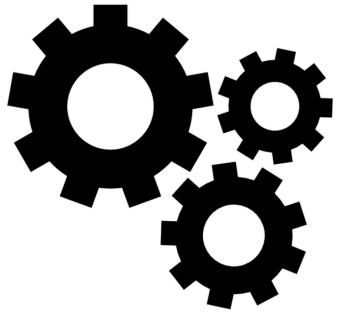
CHALLENGE: OPTIMAL SYSTEM CONFIG VARIES

Batch Sizes, Compute Platforms



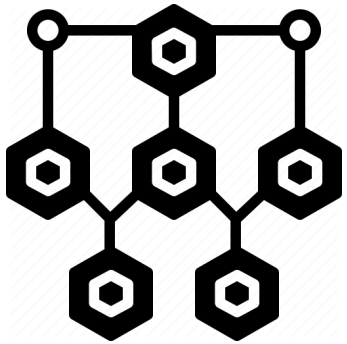
HOW TO BUILD RECOMMENDER SYSTEMS

Personalization API Builders
AutoML



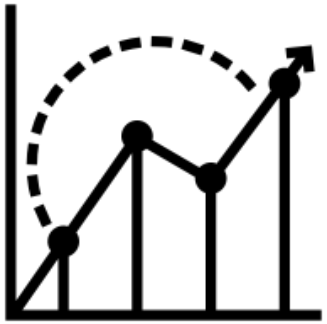
AWS Personalize

Frameworks & Libraries



Microsoft Recommenders
Nvidia Merlin

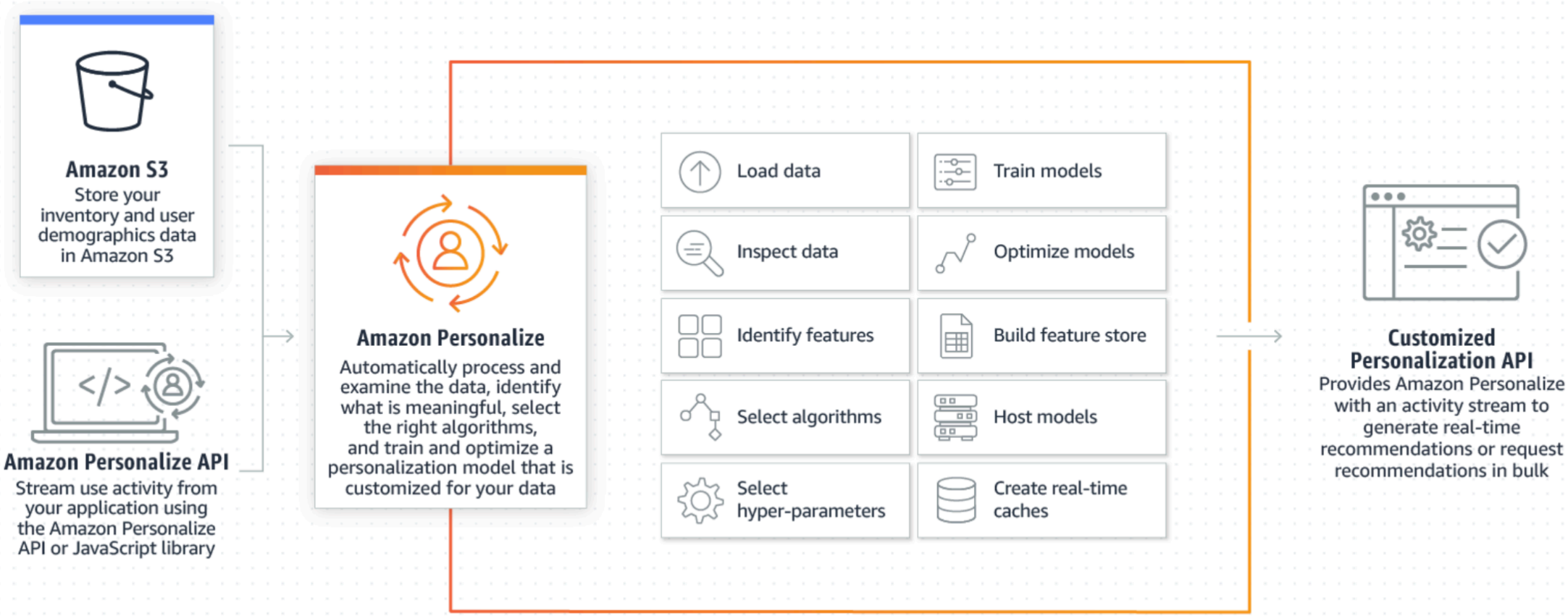
Models & Benchmarks



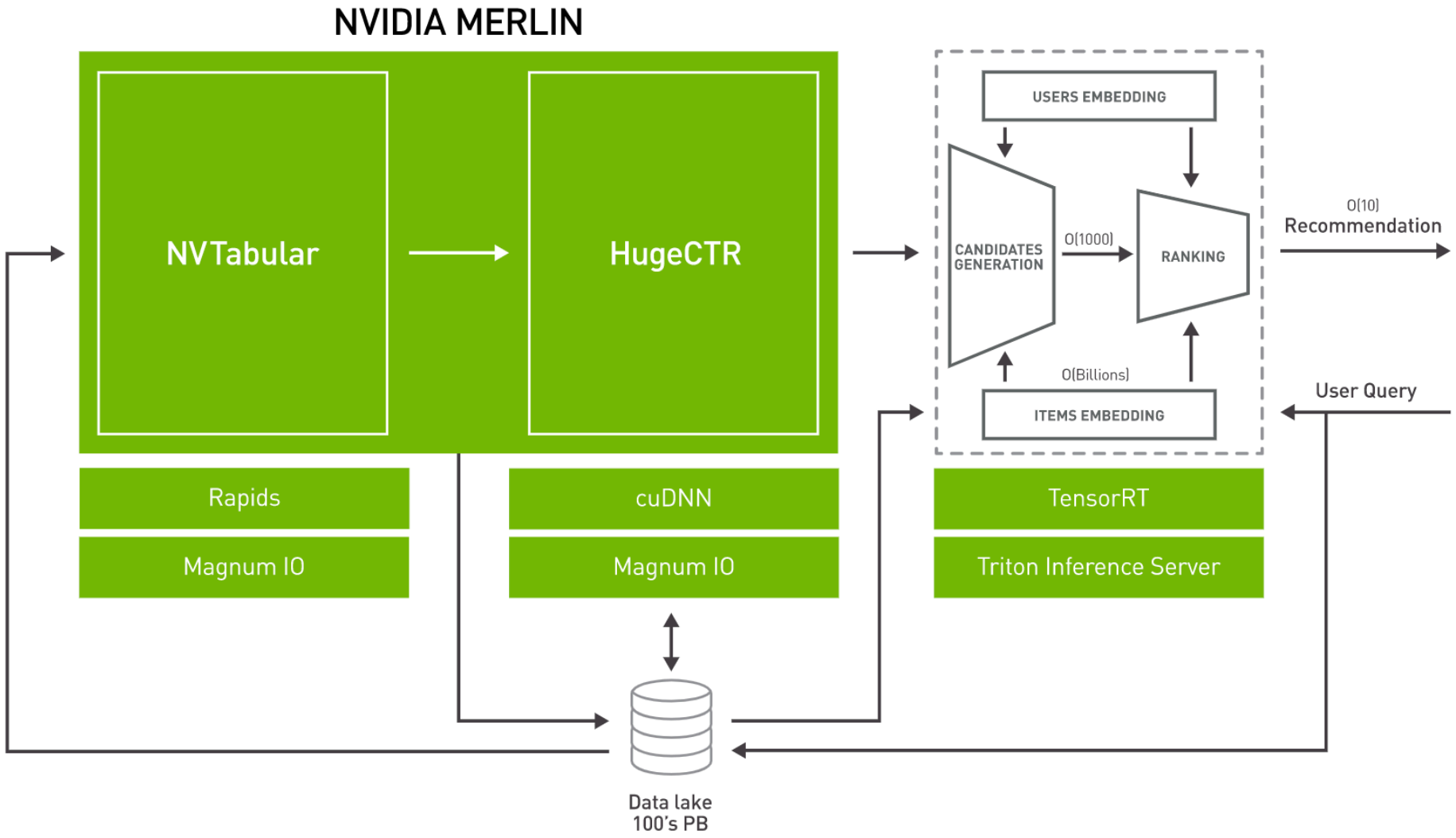
Facebook DLRM
MLPerf

Open Datasets - Criteo Ads Dataset

AWS PERSONALIZE API BUILDER - AUTOML BASED



NVIDIA MERLIN RECOMMENDER SYSTEMS FRAMEWORK

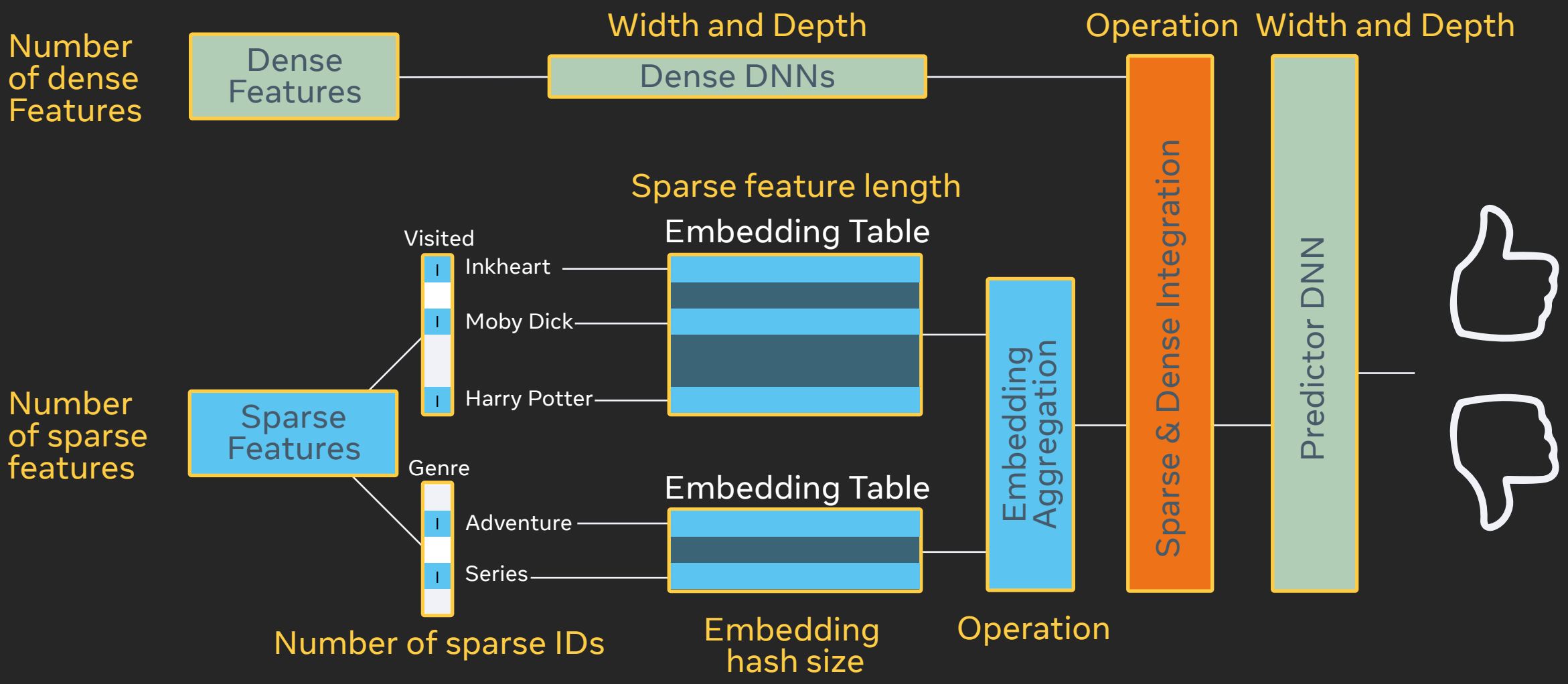


MICROSOFT RECOMMENDERS LIBRARY

Algorithm	Environment	Type	Description
Simple Algorithm for Recommendation (SAR)*	Python CPU	Collaborative Filtering	Similarity-based algorithm for implicit feedback dataset
Surprise/Singular Value Decomposition (SVD)	Python CPU	Collaborative Filtering	Matrix factorization algorithm for predicting explicit rating feedback in datasets that are not very large
Neural Collaborative Filtering (NCF)	Python CPU / Python GPU	Collaborative Filtering	Deep learning algorithm with enhanced performance for implicit feedback
Restricted Boltzmann Machines (RBM)	Python CPU / Python GPU	Collaborative Filtering	Neural network based algorithm for learning the underlying probability distribution for explicit or implicit feedback
FastAI Embedding Dot Bias (FAST)	Python CPU / Python GPU	Collaborative Filtering	General purpose algorithm with embeddings and biases for users and items
Alternating Least Squares (ALS)	PySpark	Collaborative Filtering	Matrix factorization algorithm for explicit or implicit feedback in large datasets, optimized by Spark MLLib for scalability and distributed computing capability
Vowpal Wabbit Family (VW)*	Python CPU (train online)	Collaborative, Content-Based Filtering	Fast online learning algorithms, great for scenarios where user features / context are constantly changing

DLRM: Deep Learning Recommendation Model

A Configurable Benchmark for E2E Models



RESEARCH TOOLS FOR RECOMMENDERS

- DLRM: Configurable Benchmark for E2E Models
- MLPerf includes DLRM + Criteo Ads open dataset (Kaggle, 1 TB)
- Ecosystem partners - Optimized models and workflows
    - Nvidia optimized DLRM on Ampere GPUs
    - Intel optimized DLRM with BFloat16
    - DLRM on TPUs with GCP, Reference architectures with Azure, AWS (coming soon)
- Papers with code - search for recommendation systems
- PyTorch 1.6 with support for building Heterogenous (DDP + RPC) models
    - DDP for Dense features, DMP for Sparse features

MLPerf

RECOMMENDATION SYSTEMS ARE IMPORTANT

1. Are Important
2. Are Underinvested
3. Have Unique Systems Challenges
4. New benchmarks and datasets are NOW available

REFERENCES

- DLRM: https://ai.facebook.com/blog/dlrm-an-advanced-open-source-deep-learning-recommendation-model/
- MLPerf DLRM: https://github.com/mlperf/inference/tree/master/v0.5/recommendation
- MLPerf Recommendation Benchmark paper: https://arxiv.org/abs/2003.07336
- DLRM Paper: https://arxiv.org/abs/1906.00091
- DeepRecSys paper: https://arxiv.org/abs/2001.02772
- Nvidia DLRM: https://ngc.nvidia.com/catalog/resources/nvidia:dlrm_for_pytorch
- Intel Optimized DLRM blog: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/intel-facebook-boost-bfloat16.html
- Microsoft Recommenders: https://azure.microsoft.com/en-us/blog/building-recommender-systems-with-azure-machine-learning-service/
- Nvidia Merlin: https://ngc.nvidia.com/catalog/resources/nvidia:dlrm_for_pytorch
- PyTorch XLA on Google TPUs: https://cloud.google.com/tpu/docs/tutorials/pytorch-pod
- AWS Personalize: https://aws.amazon.com/personalize/
- Papers with code: https://paperswithcode.com/task/recommendation-systems