
AWS
re:Invent

OPN306

Deploying PyTorch models for inference using TorchServe

Shashank Prasanna
Sr. Developer Advocate, AI/ML
AWS

Geeta Chauhan
AI Partner Engineering Lead
Facebook AI

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Part 1: Shashank Prasanna, AWS (18 mins)

- Challenges with machine learning deployment
- TorchServe: Open-source model serving library for PyTorch
- Key features and benefits
- How it works
- Demo: Getting started with TorchServe

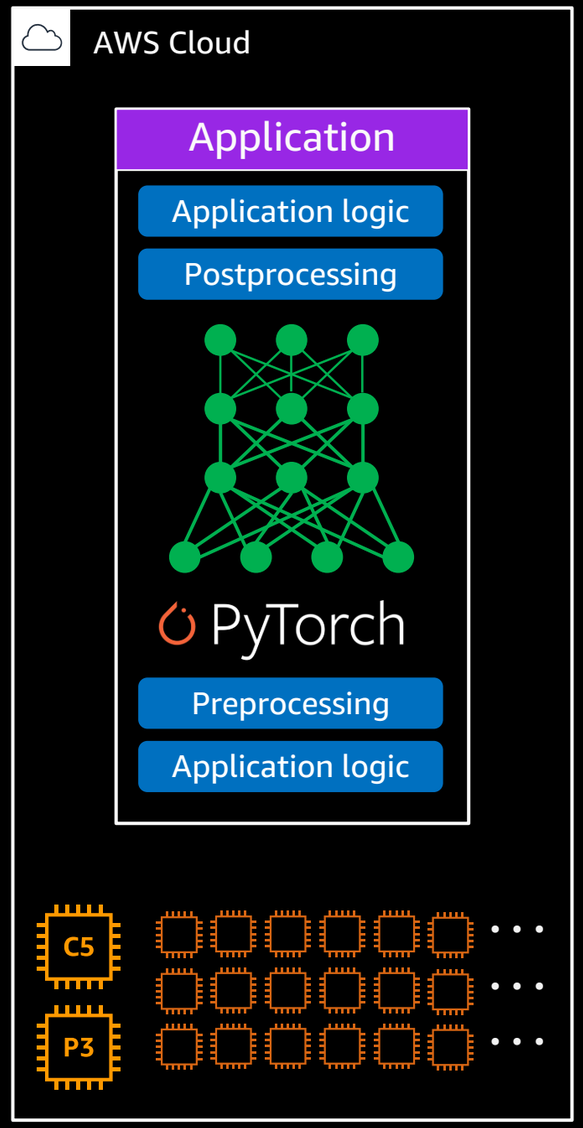
Part 2: Geeta Chauhan, Facebook AI (10 mins)

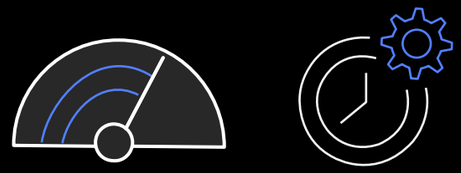
- Best practices for production deployment
- Future releases

Key takeaways and resources (2 mins)

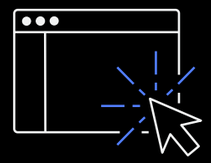
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Challenges with machine learning deployment

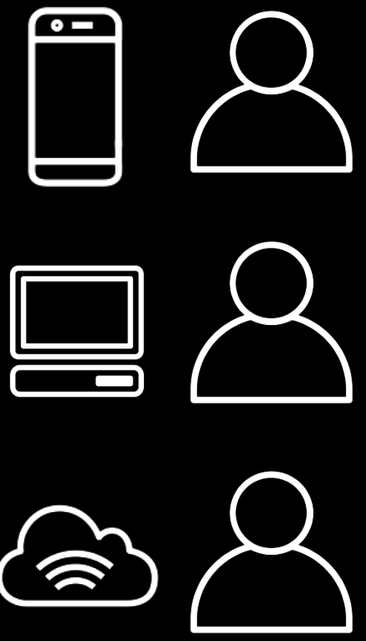


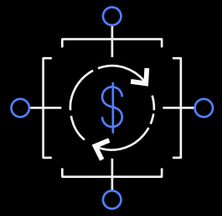


Performance

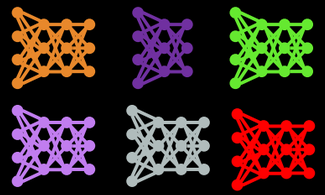


Ease of use





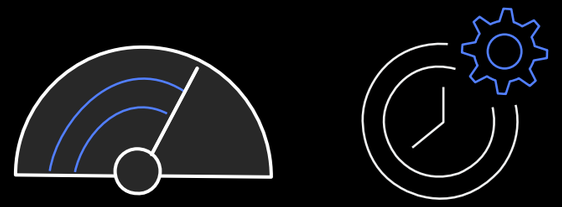
Cost efficiency



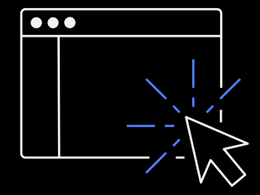
Deployment at scale

Challenges with machine learning deployment

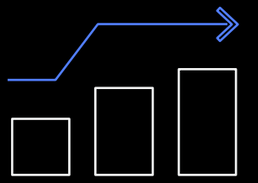
1. **Performance**
   Optimize for target throughput under desired latency



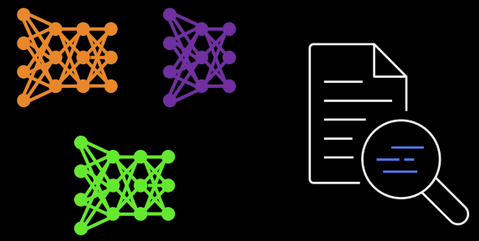
2. **Ease of deployment**
   Write custom service handlers for initialization, preprocessing, and postprocessing



3. **Cost efficiency**
   Maximize instance utilization with multiple model hosting to reduce cost / inference



4. **Host and manage models in production at scale**
   Host secure, scalable, and reliable RESTful endpoints that scale to millions of users simultaneously
   A / B testing, versioning, and model monitoring



TorchServe: Open-source model serving library for PyTorch

High-performance serving framework for PyTorch

- Low-latency prediction APIs out of the box
- Inference optimized for CPUs and GPUs



TorchServe

Easy to use

- Zero code changes for image classification, object detection, image segmentation, and text classification with default handlers

Fully customizable and flexible

- Multi-model serving
- Model versioning for A/B testing
- Server-side request batching
- Monitoring and logging
- Low-latency prediction APIs out of the box

https://github.com/pytorch/serve

Three easy ways to deploy with TorchServe

1

Host a server locally on Amazon EC2

- `pip install torchserve`
- `conda install torchserve`

2

Host server locally on Amazon EC2 or on Kubernetes with Amazon EKS and AWS Deep Learning Containers

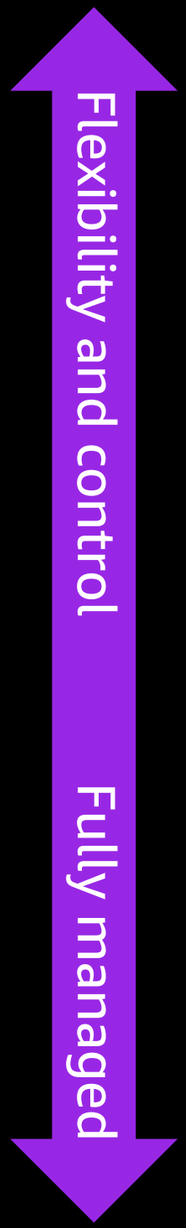
```
docker pull <account>.dkr.ecr.<region>.amazonaws.com/pytorch-inference:<version>
```

github.com/aws/deep-learning-containers/blob/master/available_images.md

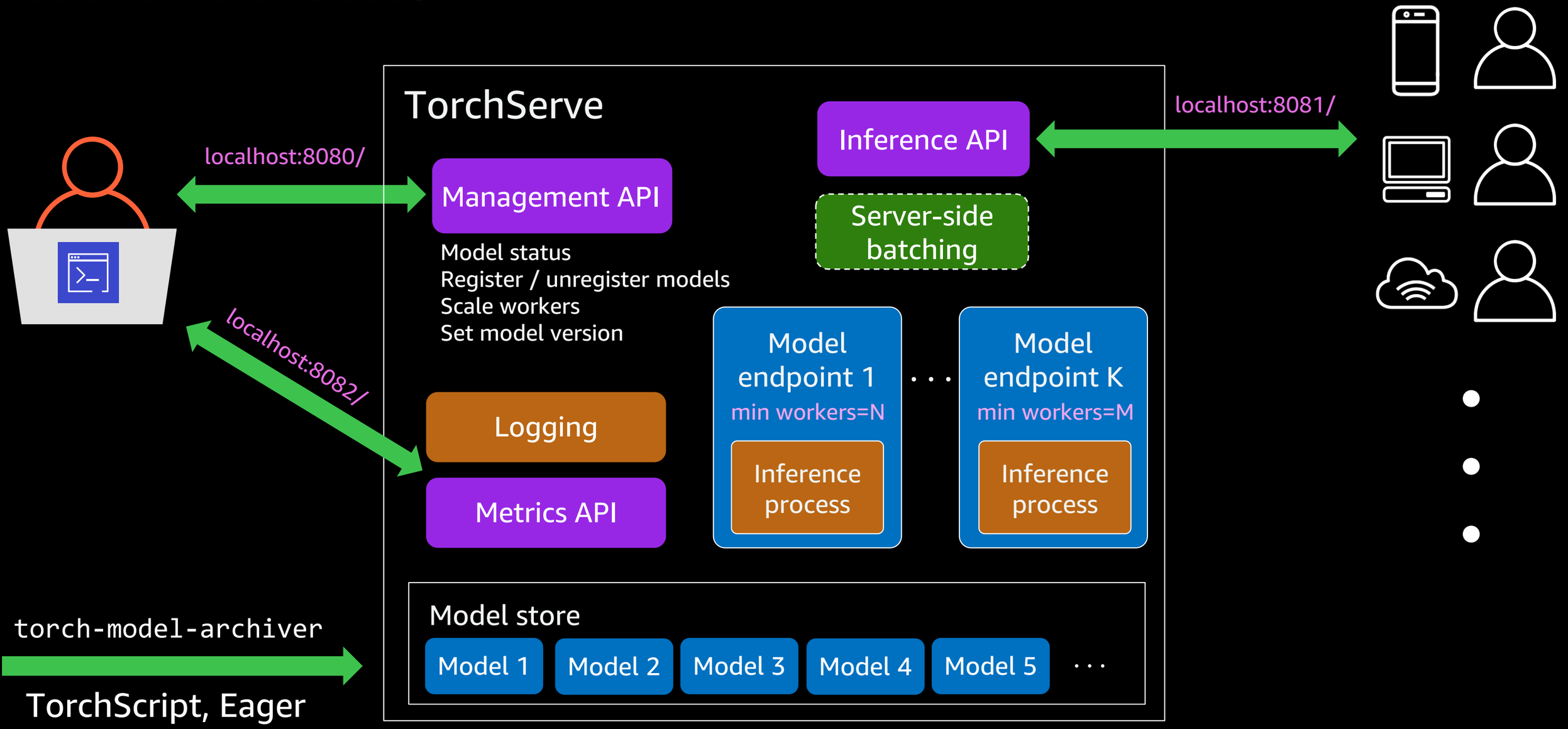
3

Fully managed endpoint with Amazon SageMaker

```
model = PyTorchModel(...)
model.deploy(initial_instance_count=N, instance_type=<CPU/GPU>)
```



How it works




```
torchserve --start --model-store <model_store>
```

Ease of use

github.com/pytorch/serve/blob/master/ts/torch_handler/image_classifier.py

Default model handlers that perform initialization, preprocessing, and post-processing for common cases:

- Image classification
- Image segmentation
- Object detection
- Text classification

Extend and customize

Automatically map from numeric classes to friendly strings when you include `index_to_name.json`

```python
import torch
import torch.nn.functional as F
from torchvision import transforms

from .vision_handler import VisionHandler
from ..utils.util  import map_class_to_label


class ImageClassifier(VisionHandler):
    """
    ImageClassifier handler class. This handler takes an image
    and returns the name of object in that image.
    """

    topk = 5
    # These are the standard Imagenet dimensions
    # and statistics
    image_processing = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    def set_max_result_classes(self, topk):
        self.topk = topk

    def get_max_result_classes(self):
        return self.topk

    def postprocess(self, data):
        ps = F.softmax(data, dim=1)
        probs, classes = torch.topk(ps, self.topk, dim=1)
        probs = probs.tolist()
        classes = classes.tolist()
        return map_class_to_label(probs, self.mapping, classes)
```

Management and inference APIs

Management API (default): http://localhost:8081

Query model status

```
curl "http://localhost:8081/models"
```

Register multiple models

```
curl -X POST "http://localhost:8081/models?url=densenet161.mar"
curl -X POST "http://localhost:8081/models?url=fastrcnn.mar"
```

Unregister model

```
curl -X DELETE http://localhost:8081/models/fastrcnn/
```

Scale workers

```
curl -v -X PUT "http://localhost:8081/models/fastrcnn?min_worker=20"
```

Inference API (default): http://localhost:8080

```
curl -X POST http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
```

Part 2:
Geeta Chauhan, Facebook AI

Best practices for production deployment

Future releases



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Best practices for production deployment



Responsible AI

- Fairness
- Human-centered design



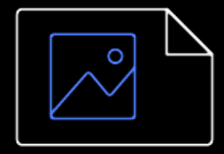
Optimizations

- Performance vs. latency
- TorchScript profiling
- Offline vs. real-time
- Cost



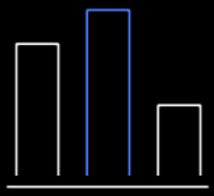
Deployment models

- Standalone
- Primary backup
- Orchestration
- Cloud vs. on-premises



Resilience

- Robust endpoint
- Auto-scaling
- Canary deployments
- A / B testing



Measurement

- Metrics
- Model performance
- Interpretability
- Feedback loop

Responsible AI

Fairness by design

- Measure skewness of data, model bias, data bias; identify relevant metrics
- Transparency, Explainable AI, inclusive design

Human-centered design

- Consider AI-driven decisions and their impact on people at the time of model design
- Provide ability to have human recourse instead of full automation – for example, mortgage applications AI rejecting people of certain category or race
- Computer vision models measure results based on demographics; for example, include support for different skin tones, age groups

Optimizations

Build with performance vs. latency goals in mind

Reduce size of the model: Quantization, pruning, mixed precision training

Reduce latency: TorchScript model; use SnakeViz profiler

Evaluate GPU vs. CPU for low latency

Evaluate REST vs. gRPC for your prediction service

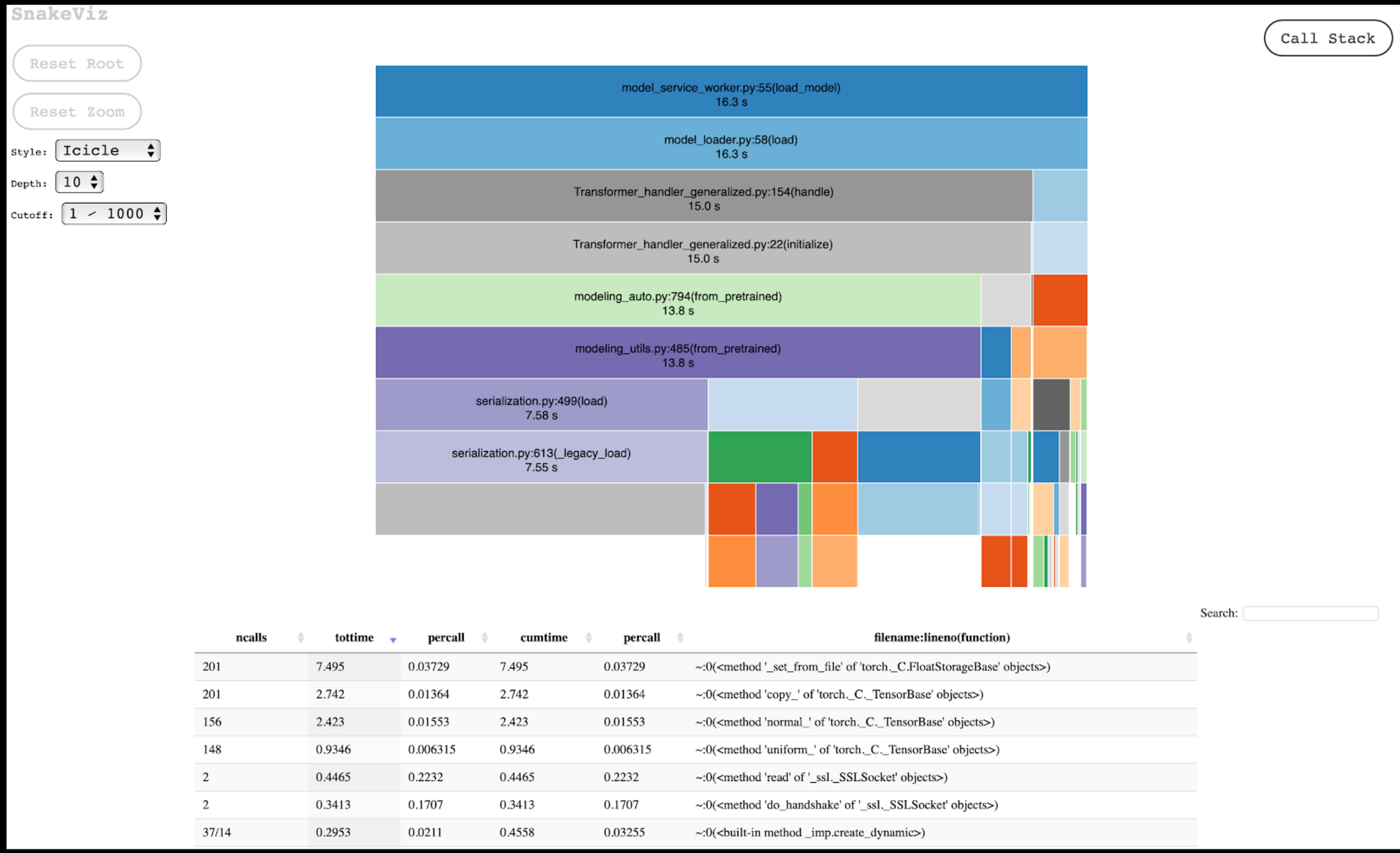
Quantization

	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

SnakeViz profiler

BERT MODEL PROFILING

Eager mode



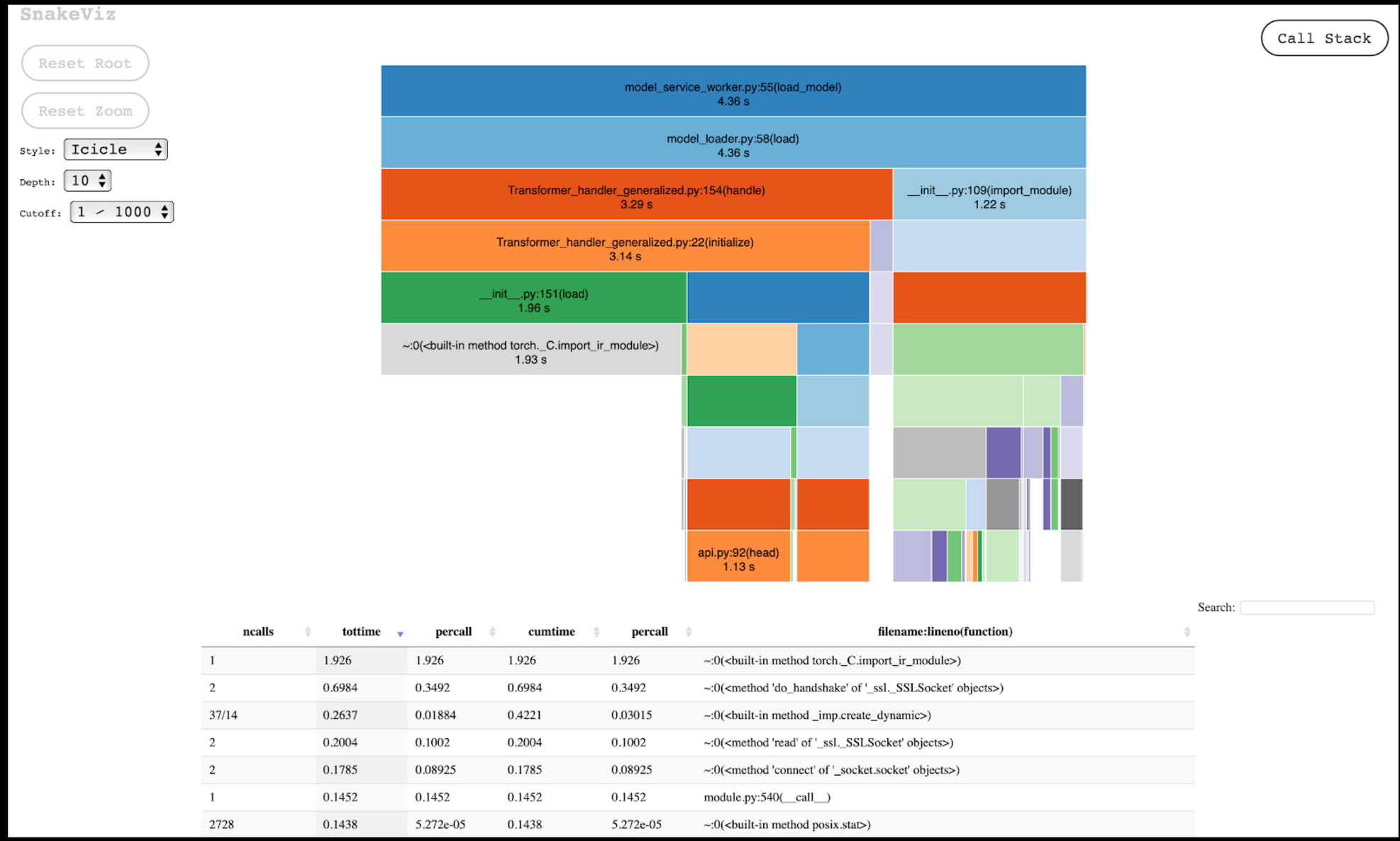
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
201	7.495	0.03729	7.495	0.03729	~:0(<method '_set_from_file' of 'torch._C.FloatStorageBase' objects>)
201	2.742	0.01364	2.742	0.01364	~:0(<method 'copy_' of 'torch._C._TensorBase' objects>)
156	2.423	0.01553	2.423	0.01553	~:0(<method 'normal_' of 'torch._C._TensorBase' objects>)
148	0.9346	0.006315	0.9346	0.006315	~:0(<method 'uniform_' of 'torch._C._TensorBase' objects>)
2	0.4465	0.2232	0.4465	0.2232	~:0(<method 'read' of '_ssl._SSLSocket' objects>)
2	0.3413	0.1707	0.3413	0.1707	~:0(<method 'do_handshake' of '_ssl._SSLSocket' objects>)
37/14	0.2953	0.0211	0.4558	0.03255	~:0(<built-in method _imp.create_dynamic>)

SnakeViz profiler

BERT MODEL PROFILING

TorchScript mode

4x speedup



Optimizations (cont'd)

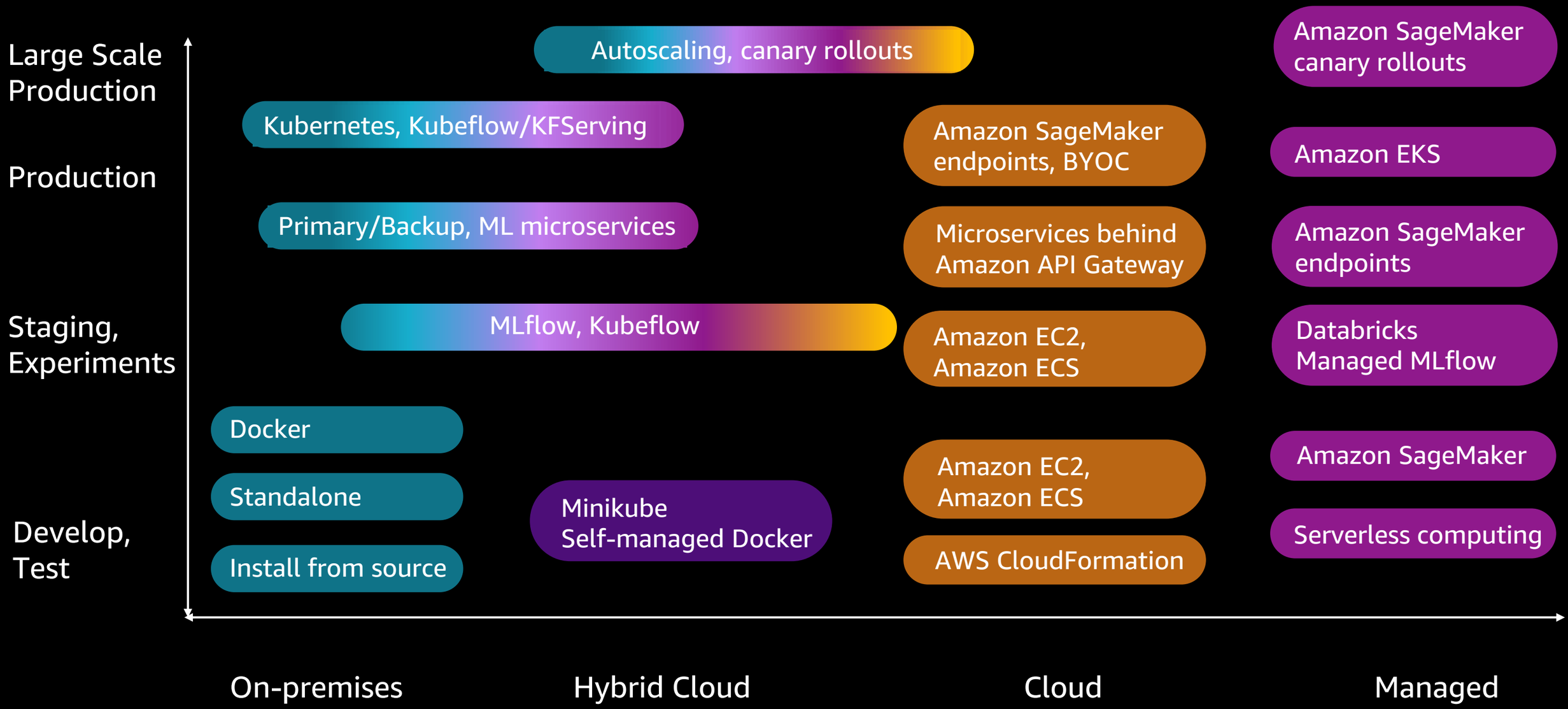
Offline vs. real-time predictions

- Offline: Dynamic batching
- Online: Asynchronous processing – push/pull
- Pre-computed predictions for certain elements

Cost optimizations

- Spot Instances for offline
- Autoscaling based on metrics, on-demand cluster

Deploying models in production



Resilience

Create robust endpoint for serving, for example, SageMaker endpoint

Auto-scaling with orchestration deployments, multi-node for EC2, and other scenarios

Canary deployments: Test new version of a model on small subset before making default

Shadow inference: Deploy new version of model in parallel

A / B testing of different versions of model

Measurement

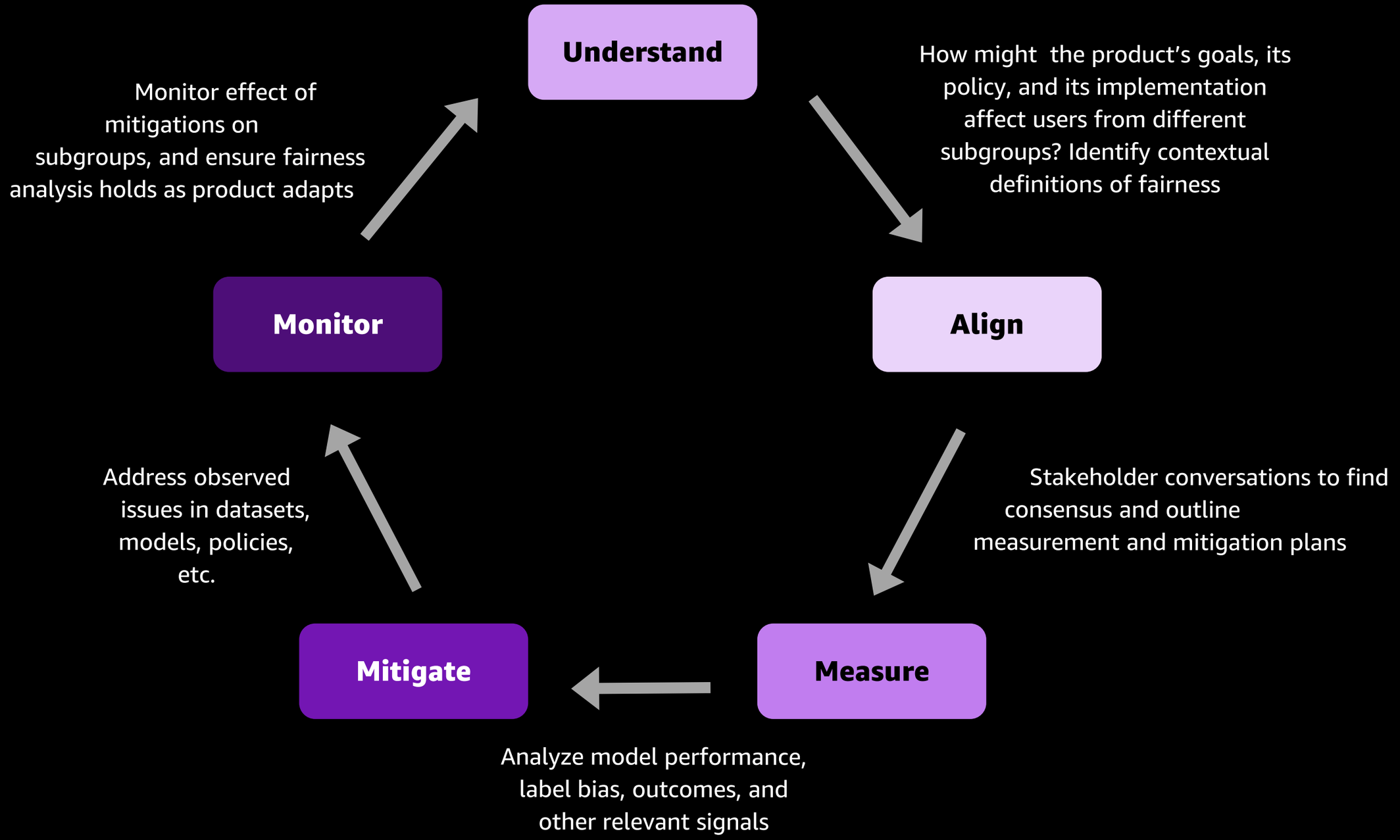
Define model performance metrics, such as accuracy, while designing the AI service; use-case specific

Add custom metrics as appropriate

Use CloudWatch or Prometheus dashboards for monitoring model performance

Model interpretability analysis via Captum

Deploy with a feedback loop, if model accuracy drops over time or new version, analyze issues like concept drift, stale data, etc.


Understand
How might the product's goals, its policy, and its implementation affect users from different subgroups? Identify contextual definitions of fairness
Align
Stakeholder conversations to find consensus and outline measurement and mitigation plans
Measure
Analyze model performance, label bias, outcomes, and other relevant signals
Mitigate
Address observed issues in datasets, models, policies, etc.
Monitor
Monitor effect of mitigations on subgroups, and ensure fairness analysis holds as product adapts

Future releases

Ensemble model support

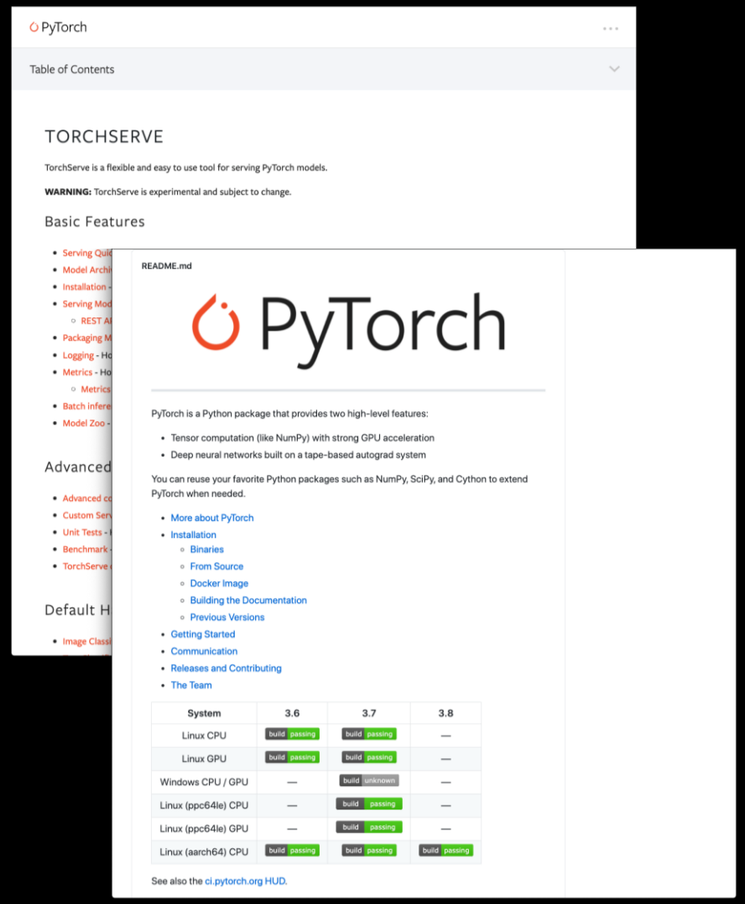
Improved memory and resource usage for better scalability

C++ backend for lower latency

AWS Inferentia AI accelerator support

Enhanced profiling tools

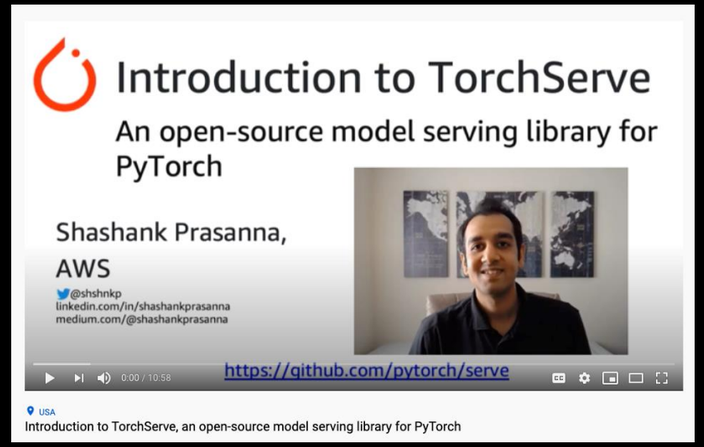
Resources



pytorch.org/serve/
github.com/pytorch/pytorch

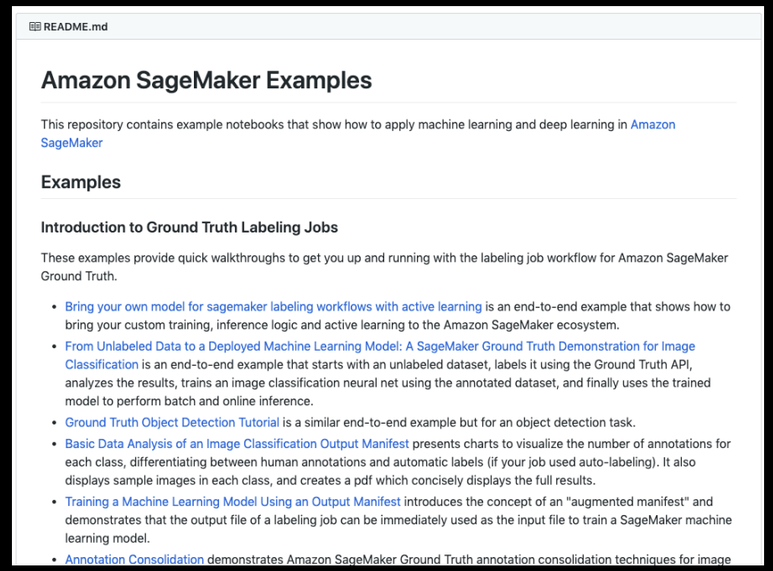


aws.amazon.com/blogs/machine-learning/deploying-pytorch-models-for-inference-at-scale-using-torchserve/



youtube.com/watch?v=AIrrI8WOIuk

Resources: Amazon SageMaker


README.md

Amazon SageMaker Examples

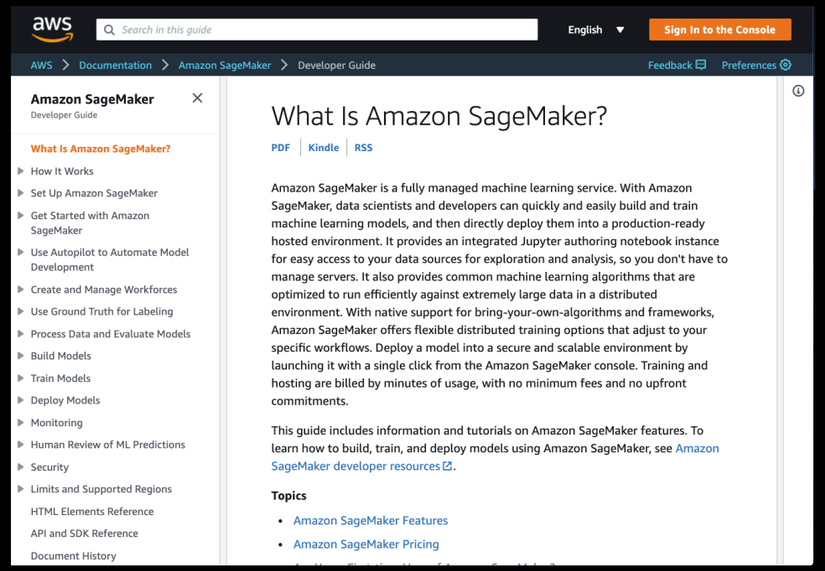
This repository contains example notebooks that show how to apply machine learning and deep learning in Amazon SageMaker

Examples

Introduction to Ground Truth Labeling Jobs

These examples provide quick walkthroughs to get you up and running with the labeling job workflow for Amazon SageMaker Ground Truth.

- Bring your own model for sagemaker labeling workflows with active learning is an end-to-end example that shows how to bring your custom training, inference logic and active learning to the Amazon SageMaker ecosystem.
- From Unlabeled Data to a Deployed Machine Learning Model: A SageMaker Ground Truth Demonstration for Image Classification is an end-to-end example that starts with an unlabeled dataset, labels it using the Ground Truth API, analyzes the results, trains an image classification neural net using the annotated dataset, and finally uses the trained model to perform batch and online inference.
- Ground Truth Object Detection Tutorial is a similar end-to-end example but for an object detection task.
- Basic Data Analysis of an Image Classification Output Manifest presents charts to visualize the number of annotations for each class, differentiating between human annotations and automatic labels (if your job used auto-labeling). It also displays sample images in each class, and creates a pdf which concisely displays the full results.
- Training a Machine Learning Model Using an Output Manifest introduces the concept of an "augmented manifest" and demonstrates that the output file of a labeling job can be immediately used as the input file to train a SageMaker machine learning model.
- Annotation Consolidation demonstrates Amazon SageMaker Ground Truth annotation consolidation techniques for image


aws

Search in this guide

English

Sign In to the Console

AWS > Documentation > Amazon SageMaker > Developer Guide

Feedback Preferences

Amazon SageMaker
Developer Guide

What Is Amazon SageMaker?
How It Works
Set Up Amazon SageMaker
Get Started with Amazon SageMaker
Use Autopilot to Automate Model Development
Create and Manage Workforces
Use Ground Truth for Labeling
Process Data and Evaluate Models
Build Models
Train Models
Deploy Models
Monitoring
Human Review of ML Predictions
Security
Limits and Supported Regions
HTML Elements Reference
API and SDK Reference
Document History

What Is Amazon SageMaker?

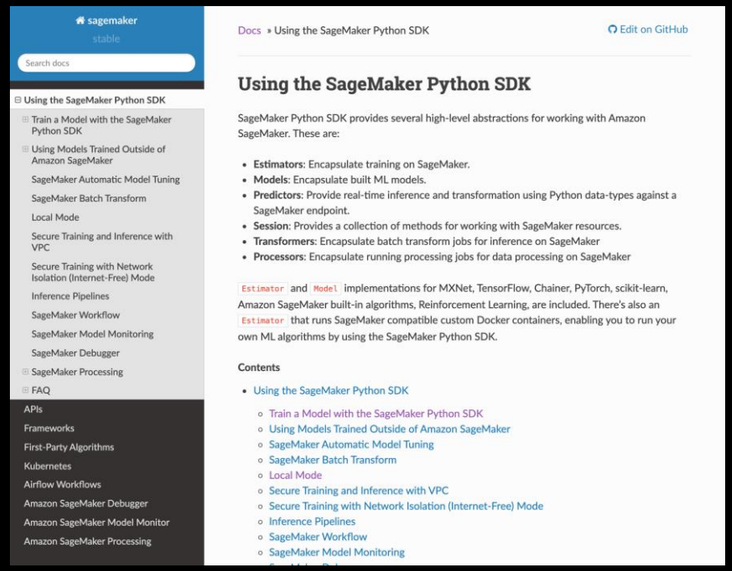
PDF | Kindle | RSS

Amazon SageMaker is a fully managed machine learning service. With Amazon SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, Amazon SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a single click from the Amazon SageMaker console. Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

This guide includes information and tutorials on Amazon SageMaker features. To learn how to build, train, and deploy models using Amazon SageMaker, see Amazon SageMaker developer resources.

Topics

- Amazon SageMaker Features
- Amazon SageMaker Pricing


sagemaker
stable

Search docs

Using the SageMaker Python SDK
Train a Model with the SageMaker Python SDK
Using Models Trained Outside of Amazon SageMaker
SageMaker Automatic Model Tuning
SageMaker Batch Transform
Local Mode
Secure Training and Inference with VPC
Secure Training with Network Isolation (Internet-Free) Mode
Inference Pipelines
SageMaker Workflow
SageMaker Model Monitoring
SageMaker Debugger
SageMaker Processing
FAQ
APIs
Frameworks
First-Party Algorithms
Kubernetes
Airflow Workflows
Amazon SageMaker Debugger
Amazon SageMaker Model Monitor
Amazon SageMaker Processing

Docs » Using the SageMaker Python SDK

Edit on GitHub

Using the SageMaker Python SDK

SageMaker Python SDK provides several high-level abstractions for working with Amazon SageMaker. These are:

- **Estimators**: Encapsulate training on SageMaker.
- **Models**: Encapsulate built ML models.
- **Predictors**: Provide real-time inference and transformation using Python data-types against a SageMaker endpoint.
- **Session**: Provides a collection of methods for working with SageMaker resources.
- **Transformers**: Encapsulate batch transform jobs for inference on SageMaker
- **Processors**: Encapsulate running processing jobs for data processing on SageMaker

`Estimator` and `Model` implementations for MXNet, TensorFlow, Chainer, PyTorch, scikit-learn, Amazon SageMaker built-in algorithms, Reinforcement Learning, are included. There's also an `Estimator` that runs SageMaker compatible custom Docker containers, enabling you to run your own ML algorithms by using the SageMaker Python SDK.

Contents

- Using the SageMaker Python SDK
  - Train a Model with the SageMaker Python SDK
  - Using Models Trained Outside of Amazon SageMaker
  - SageMaker Automatic Model Tuning
  - SageMaker Batch Transform
  - Local Mode
  - Secure Training and Inference with VPC
  - Secure Training with Network Isolation (Internet-Free) Mode
  - Inference Pipelines
  - SageMaker Workflow
  - SageMaker Model Monitoring

https://github.com/awslabs/amazon-sagemaker-examples

https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html

https://sagemaker.readthedocs.io/en/stable/overview.html

Resources: Related AWS re:Invent 2020 talk

re:Invent 2020 talk

AIM301: How to choose the right instance type for ML inference



Blog post
https://towardsdatascience.com/a-complete-guide-to-ai-accelerators-for-deep-learning-inference-gpus-aws-inferentia-and-amazon-7a5d6804ef1c

Thank you!

Shashank Prasanna

Sr. Developer Advocate, AI/ML
AWS

Geeta Chauhan

AI Partner Engineering Lead
Facebook AI

Twitter: @shshnkp

LinkedIn: linkedin.com/in/shashankprasanna

Medium: medium.com/@shashankprasanna

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.