

# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



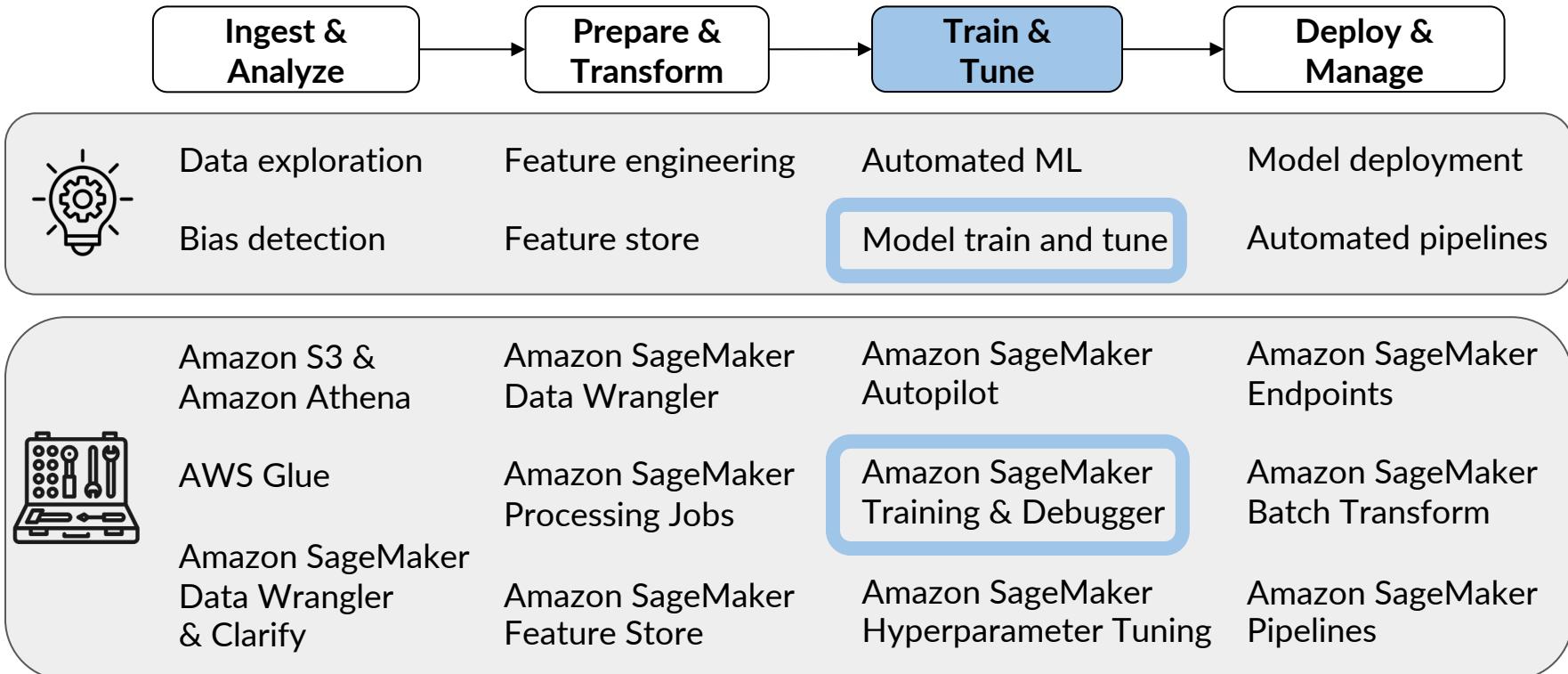
DeepLearning.AI



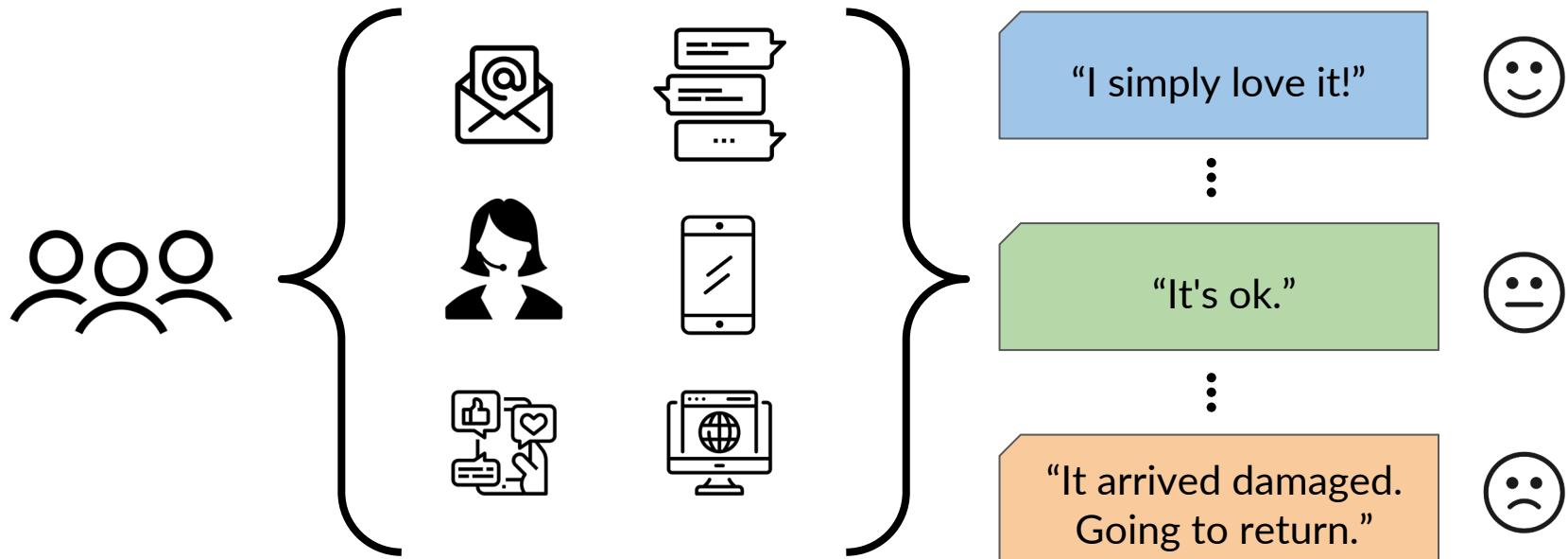
# Train and Debug a Custom Machine Learning Model

---

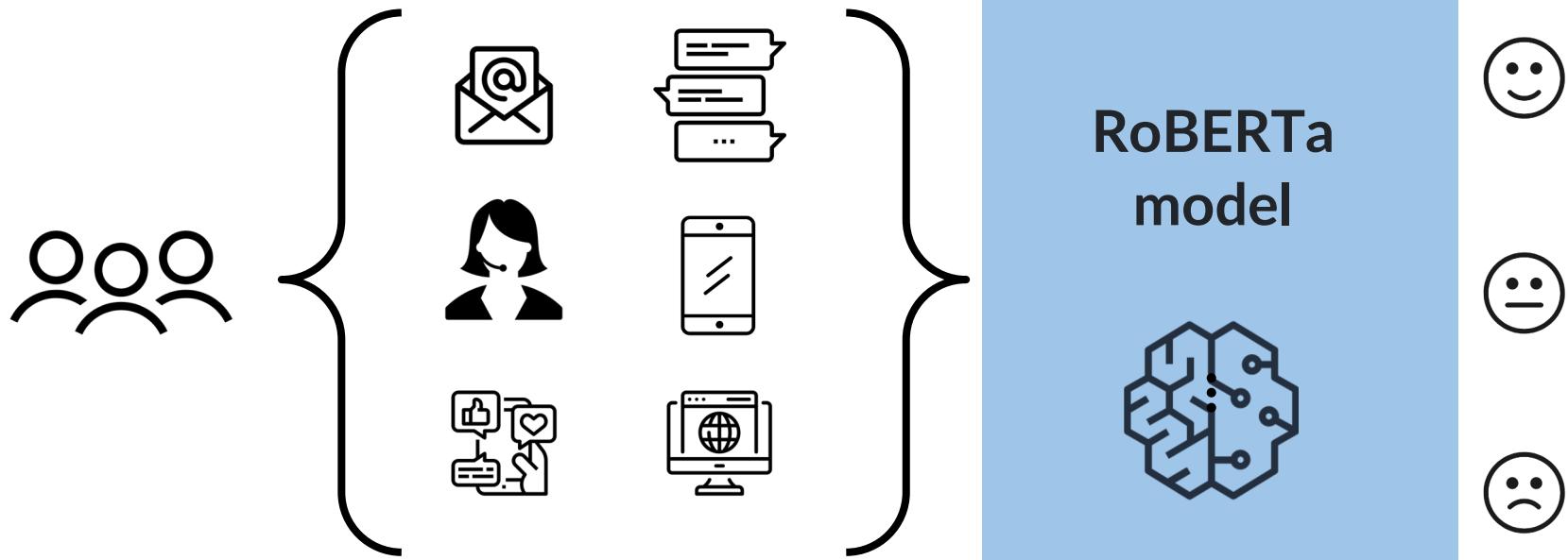
# Machine Learning Workflow



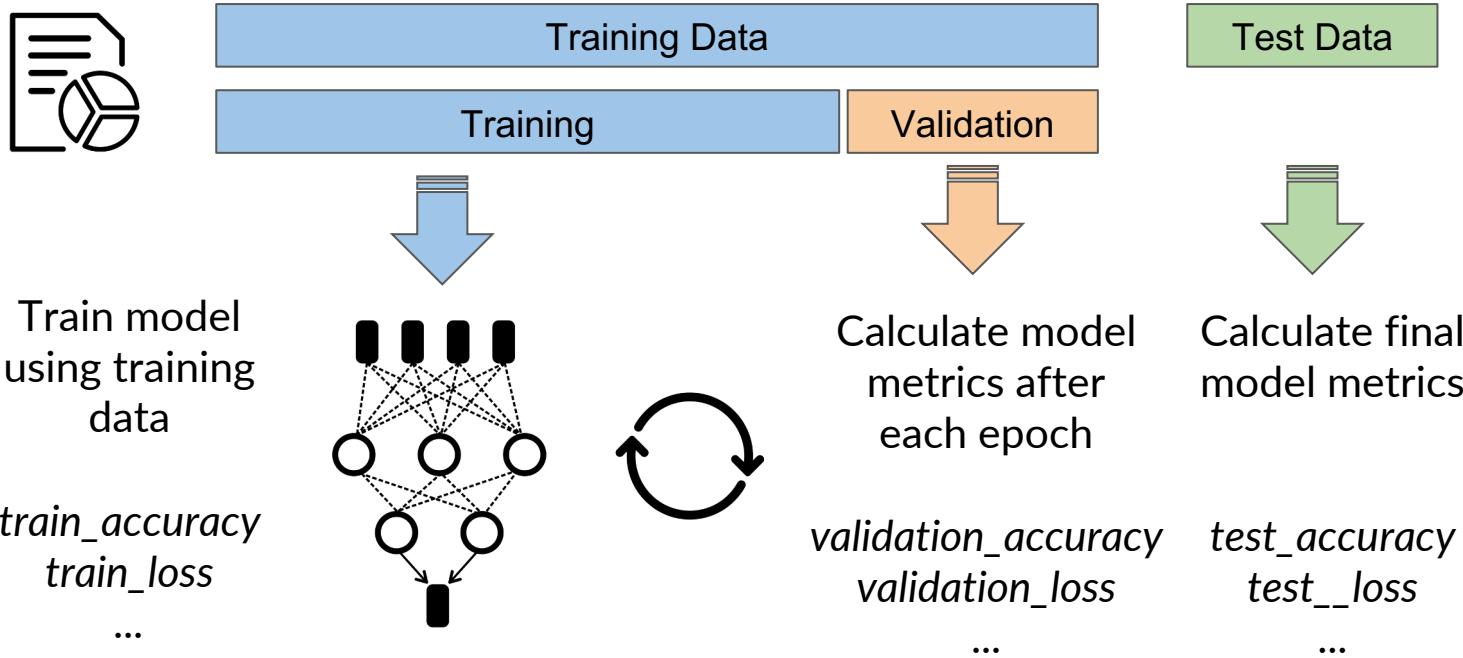
# Multi-class classification for sentiment analysis of product reviews



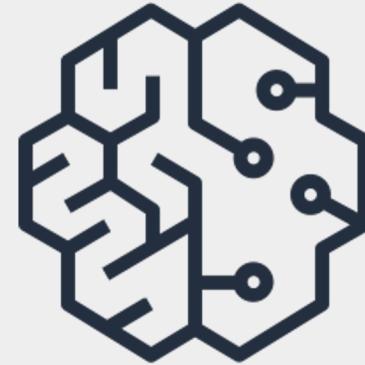
# Multi-class classification for sentiment analysis of product reviews



# Model training - Fit the model to your data



# Pre-trained models

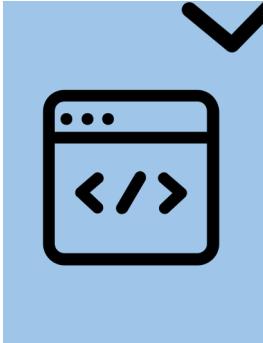


# Built-in algorithms versus pre-trained models

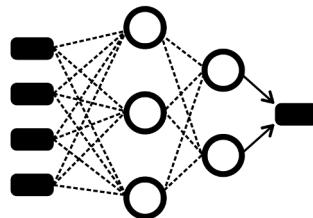
Data



Code



Model

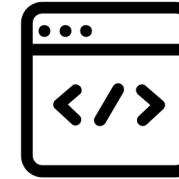


**Built-in algorithms**

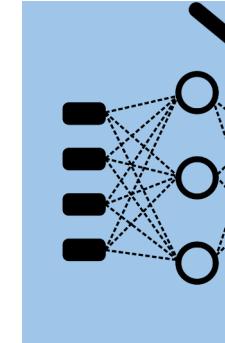
Data



Code

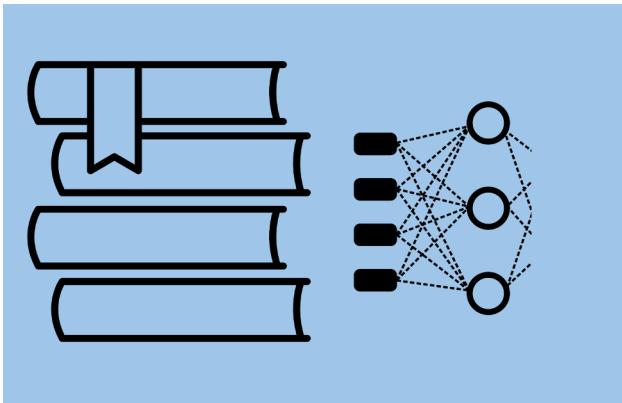


Model



**Pre-trained models**

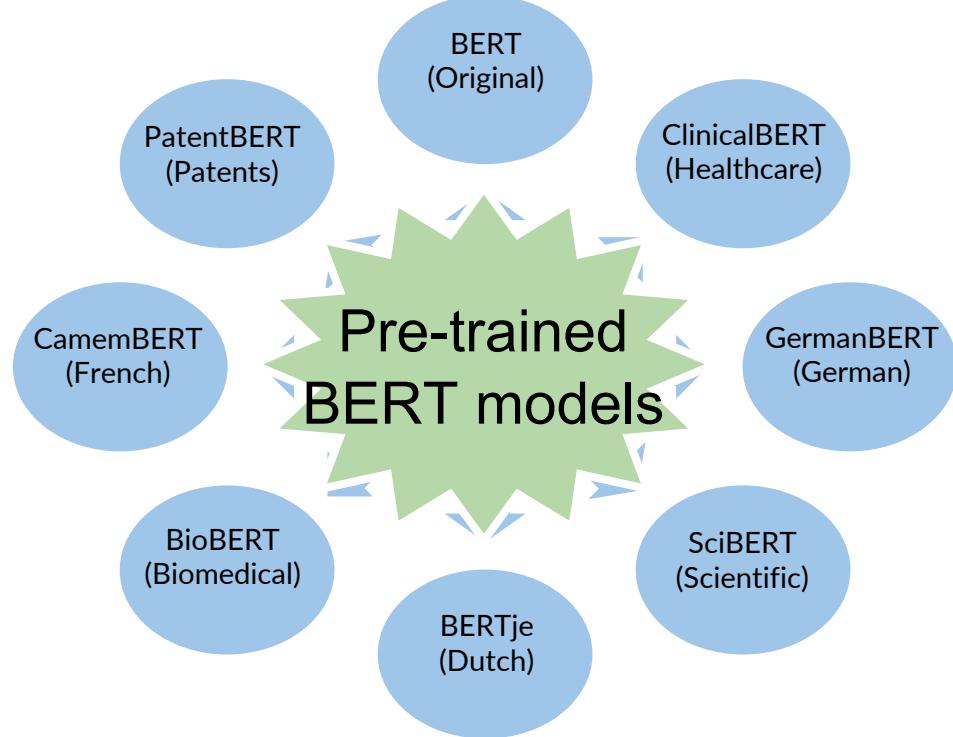
# Model pre-training and fine-tuning



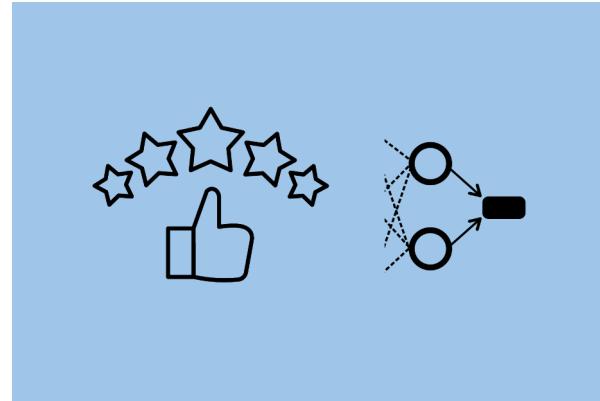
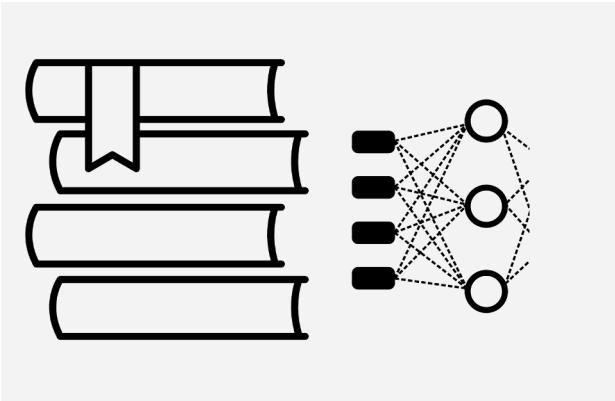
Unsupervised Learning



Pre-Training



# Model pre-training and fine-tuning



Unsupervised Learning



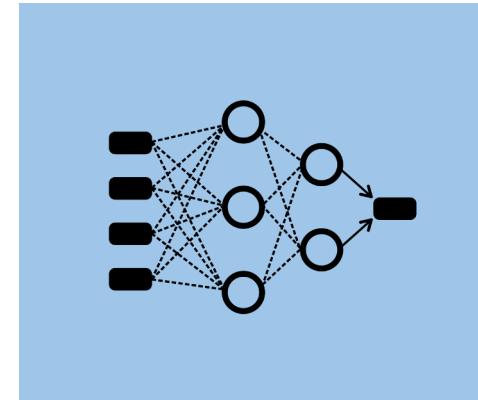
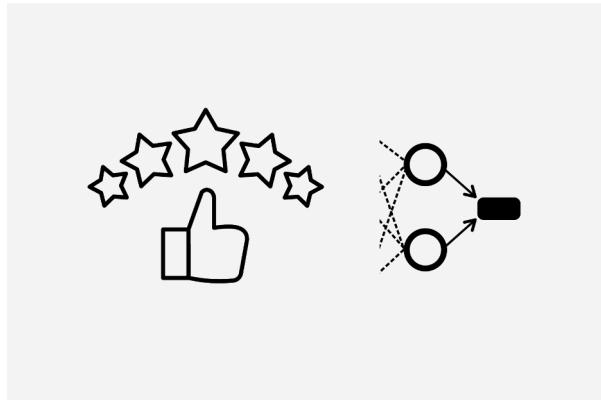
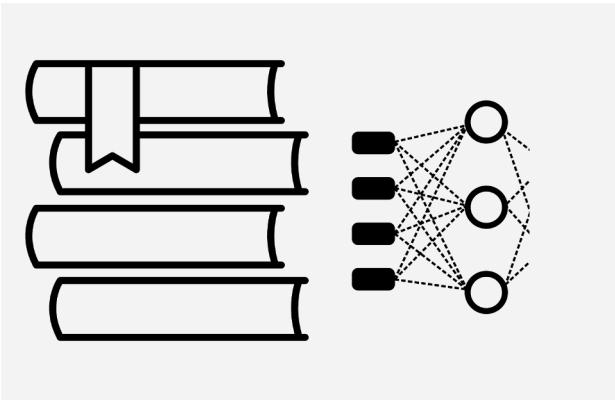
Pre-Training

Supervised Learning



Fine-Tuning

# Model pre-training and fine-tuning



Unsupervised Learning



Pre-Training

Supervised Learning



Fine-Tuning

Fine-Tuned  
Model



# Where to find pre-trained models

# Where to find pre-trained models

 PyTorch  
HUB

 mxnet

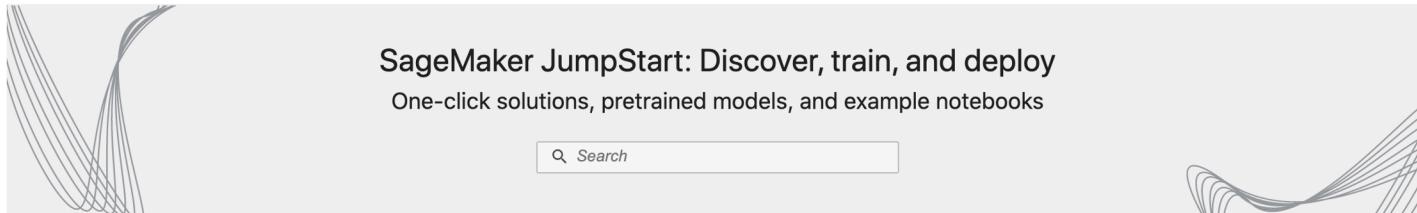
Gluon Model Zoo

 TensorFlow Hub



Hugging Face  
Model Hub

# Amazon SageMaker JumpStart



## › Solutions

Launch end-to-end machine learning solutions that tie SageMaker to other AWS services with one click.

[View all \(14\)](#)

## › Text models

Deploy and fine-tune pretrained transformers for various natural language processing use cases.

[View all \(39\)](#)

## › Vision models

Deploy and fine-tune pretrained models for image classification and object detection with one click.

[View all \(124\)](#)

# Amazon SageMaker JumpStart vision models

## ✓ Vision models

Deploy and fine-tune pretrained models for image classification and object detection with one click.

[View all \(124\)](#)



### Inception V3

Community Model · Vision

Task:	Image Classification
Dataset:	ImageNet
Fine-tunable:	Yes
Source:	TensorFlow Hub



### ResNet 18

Community Model · Vision

Task:	Image Classification
Dataset:	ImageNet
Fine-tunable:	Yes
Source:	PyTorch Hub



### SSD EfficientDet D0

Community Model · Vision

Task:	Object Detection
Dataset:	COCO 2017
Fine-tunable:	No
Source:	TensorFlow Hub



### SSD

Community Model · Vision

Task:	Object Detection
Dataset:	COCO 2017
Fine-tunable:	No
Source:	PyTorch Hub

# Amazon SageMaker JumpStart text models

## Text models

[View all \(39\)](#)



### BERT Large Cased

Community Model · Text

Task:	Extractive Question Answering
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub



### RoBERTa Base

Community Model · Text

Task:	Extractive Question Answering
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub



### BERT Base Uncased

Community Model · Text

Task:	Extractive Question Answering
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub



### BERT Base Uncased

Community Model · Text

Task:	Sentence Pair Classification
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub

# Amazon SageMaker JumpStart text models

## Text models

Deploy and fine-tune pretrained transformers for various natural language prc



### BERT Large Cased

Community Model · Text

Task:	Extractive Question Answering
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub



### RoBERTa Base

Community Model · Text

Task:	Extractive Question Answering
Pre-training Dataset:	English Text
Fine-tunable:	Yes
Source:	PyTorch Hub

## MODEL

### BERT Base Uncased

Text · Sentence Pair Classification

#### Get Started

##### Deploy Model

Deploy a pretrained model to an endpoint for inference. Deploying on SageMaker hosts the model on the specified compute instance and creates an internal API endpoint. JumpStart will provide you an example notebook to access the model after it is deployed. [Learn more.](#)

##### Deployment Configuration

Deploy

##### Fine-tune Model

Create a training job to fine-tune this pretrained model to fit your own data. Fine-tuning trains a pretrained model on a new dataset without training from scratch. It can produce accurate models with smaller datasets and less training time. [Learn more.](#)

# Amazon SageMaker JumpStart solutions

## Solutions

Launch end-to-end machine learning solutions that tie SageMaker to other AWS services with one click.

[View all \(14\)](#)



### Fraud Detection in Financial Transactions

Financial Services

Use Deep Graph Library (DGL) to train a graph neural network model to detect fraud in financial transactions.



### Explain Credit Decisions

Financial Services

Predict credit default in credit applications and provide explanations.



### Predictive Maintenance

Manufacturing

Use historical sensor readings to predict the remaining useful life for each sensor.



### Detect Malicious Users and Transactions

Financial Services

Automate the detection of potentially fraudulent activity in transactions.

# Where to find pre-trained models

 PyTorch  
HUB



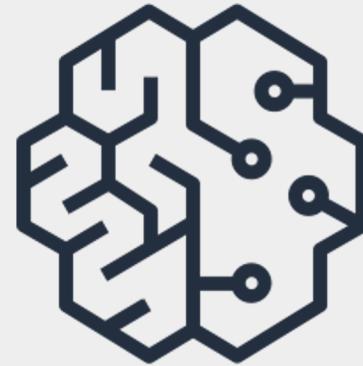
Gluon Model Zoo

  
TensorFlow Hub

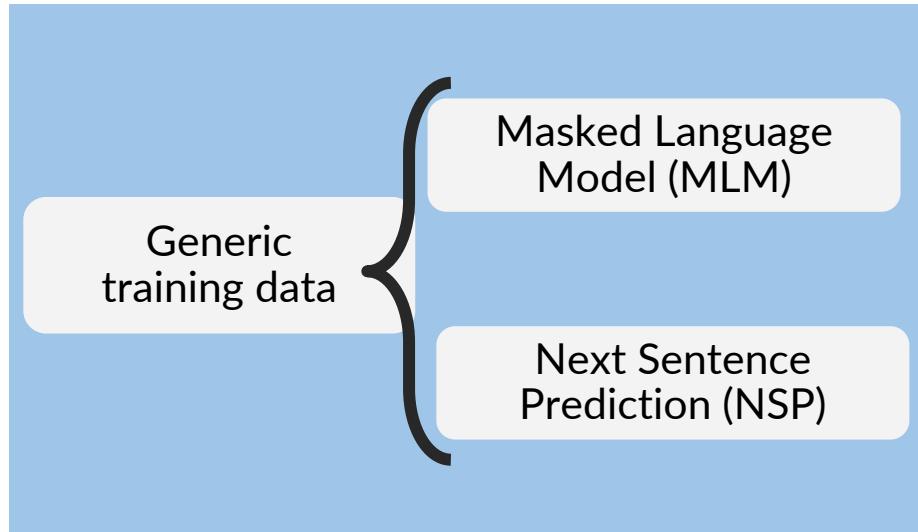


Hugging Face  
Model Hub

# Pre-trained BERT models

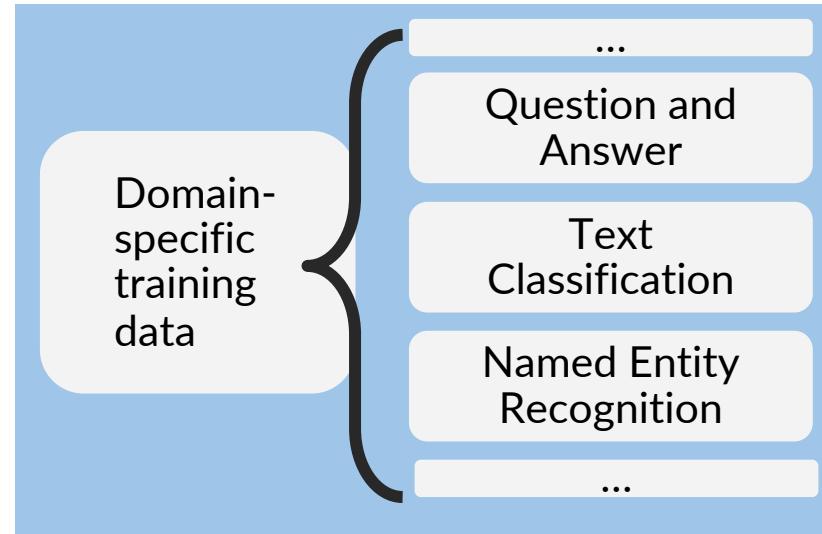


# BERT model pre-training and fine-tuning steps



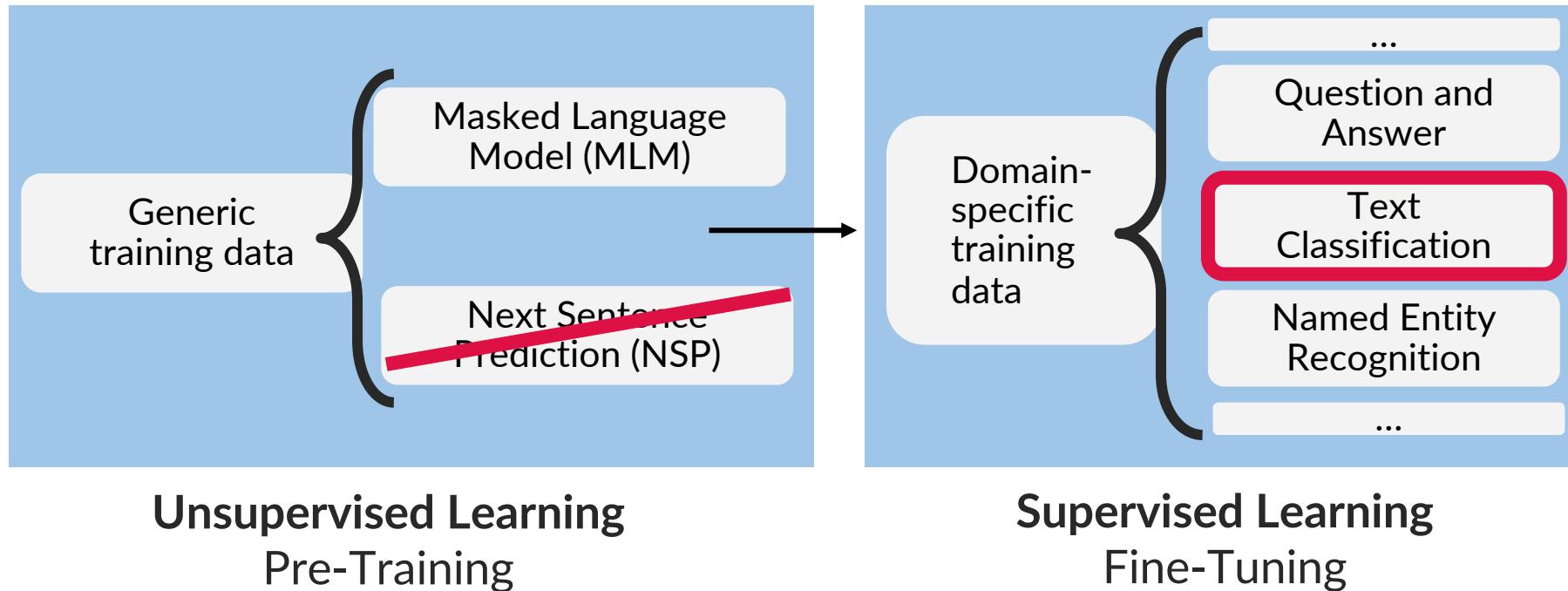
**Unsupervised Learning**  
Pre-Training

# BERT model pre-training and fine-tuning steps



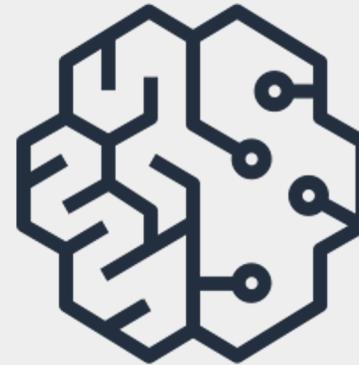
**Supervised Learning  
Fine-Tuning**

# RoBERTa model

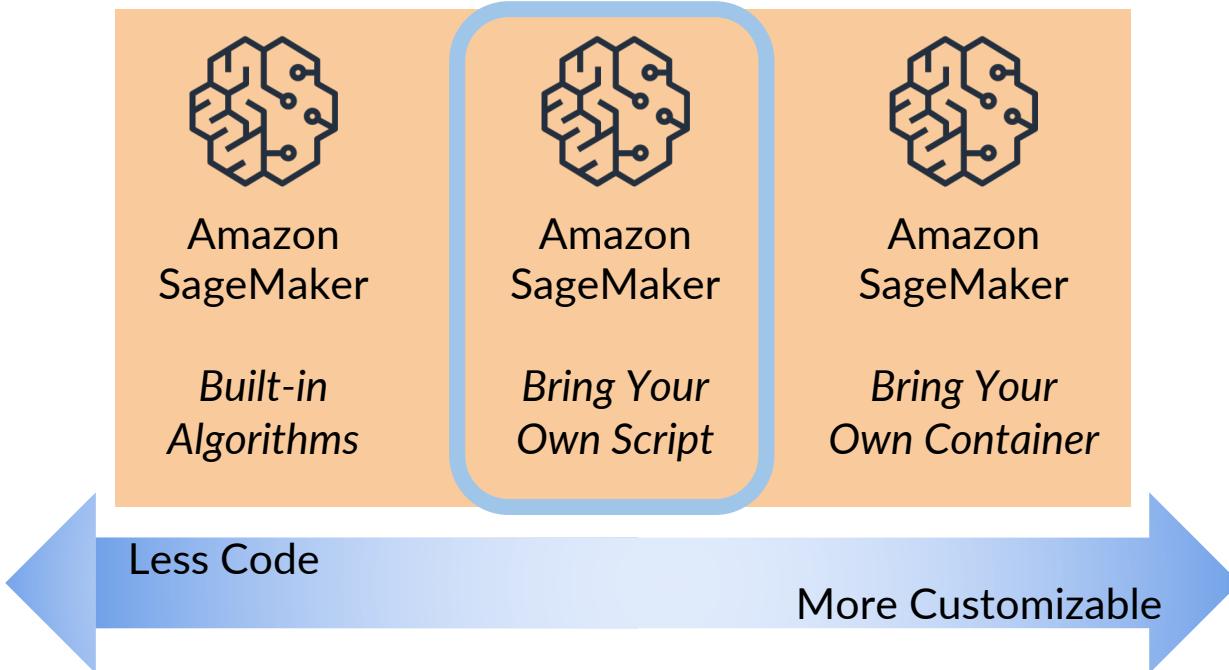


# Train a custom model

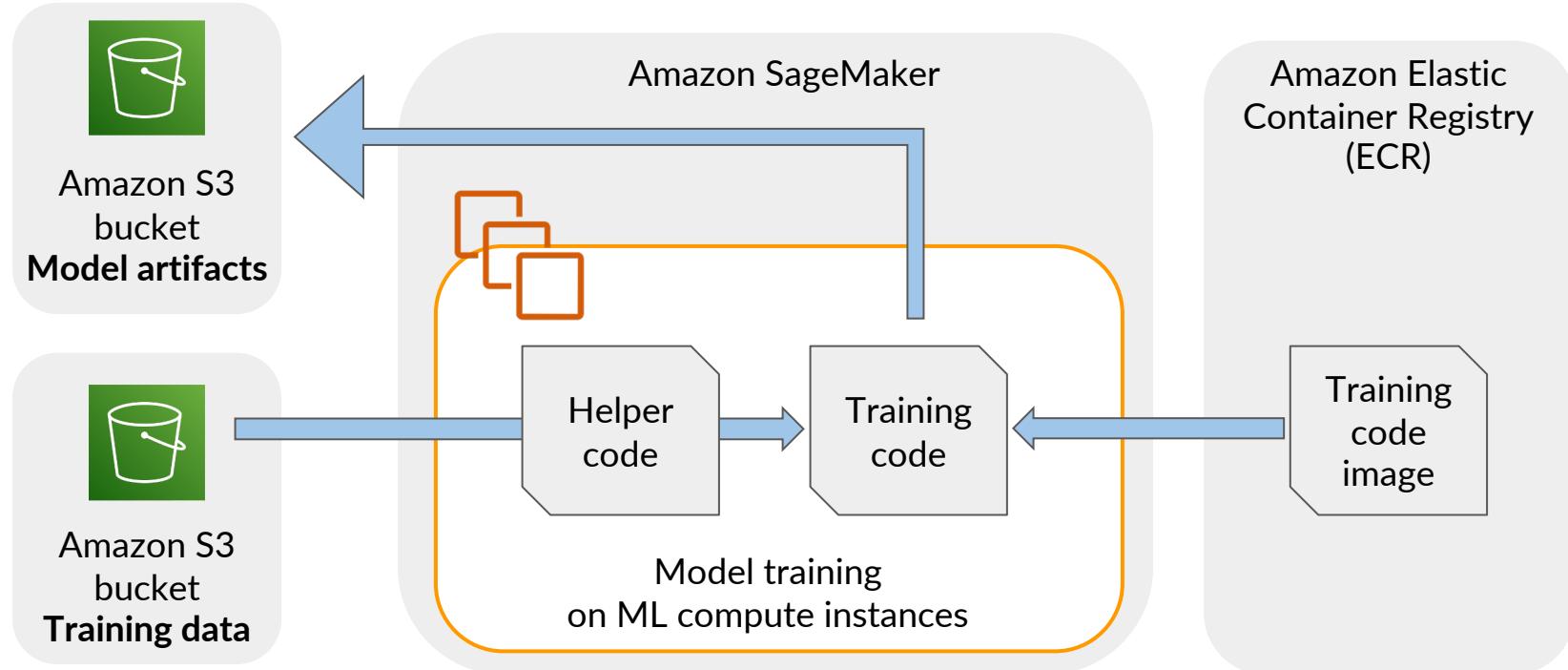
with Amazon SageMaker



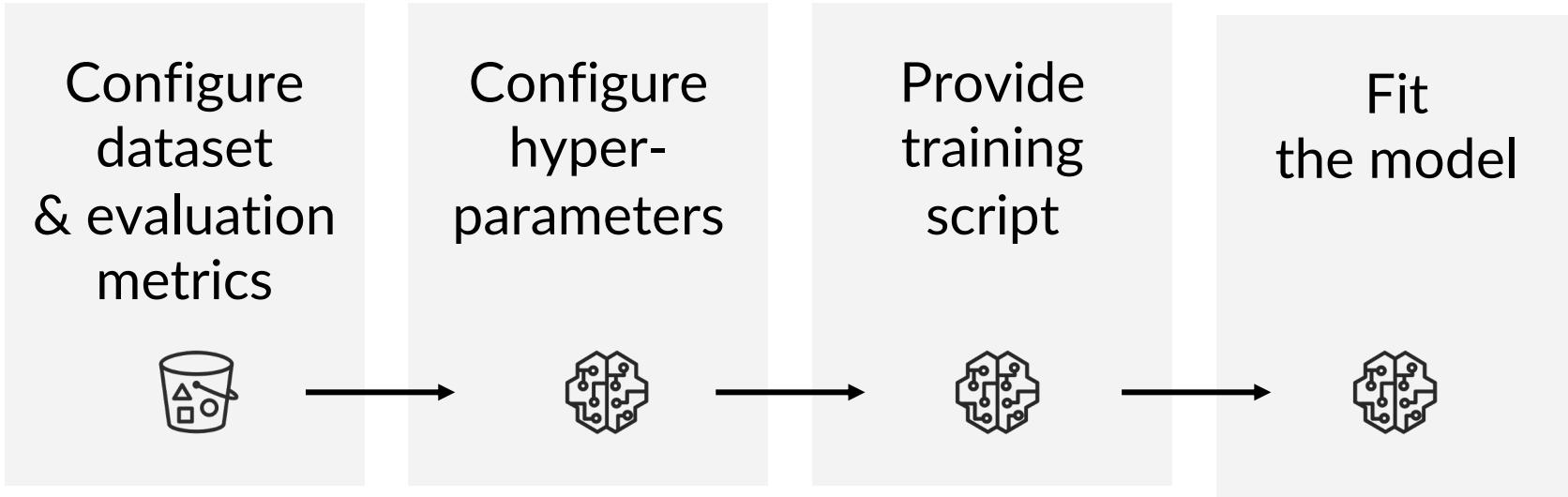
# Model training options



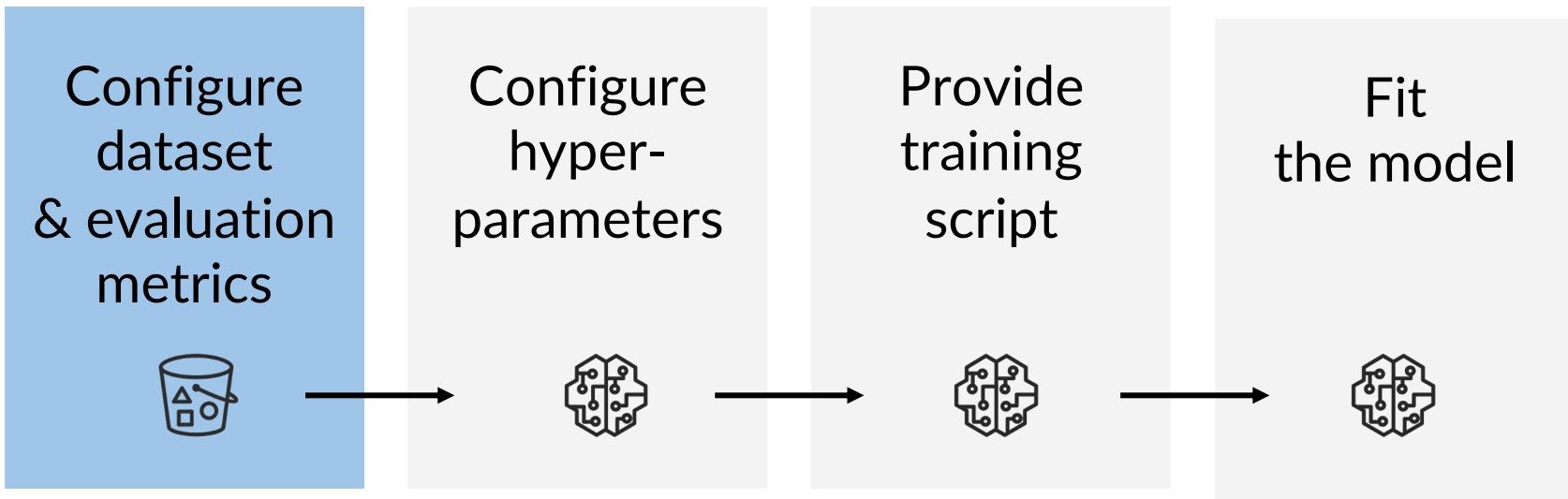
# Amazon SageMaker "Bring Your Own Script"



# Steps



# Steps



# Configure dataset and metrics

```
from sagemaker.inputs import TrainingInput  
  
s3_input_train_data = TrainingInput(s3_data="s3://...")  
s3_input_validation_data = TrainingInput(s3_data="s3://...")  
s3_input_test_data = TrainingInput(s3_data="s3://...")
```



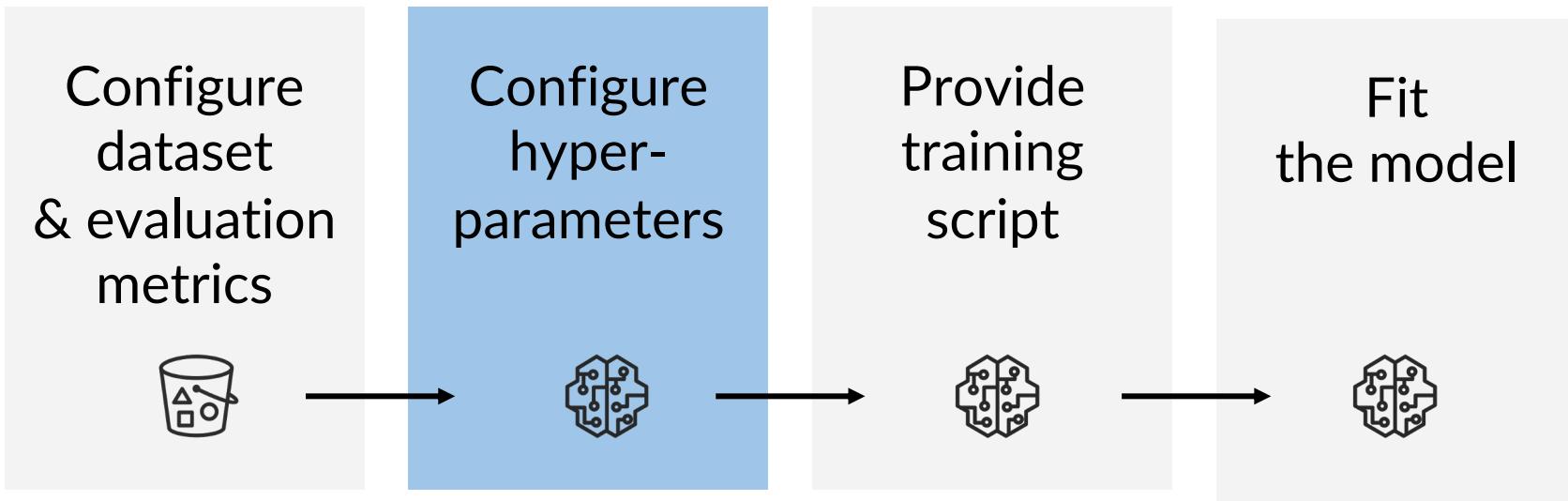
Amazon S3

```
metric_definitions = [  
    {'Name': 'validation:loss', 'Regex': 'val_loss: ([0-9\\.]+)'},  
    {'Name': 'validation:accuracy', 'Regex': 'val_acc: ([0-9\\.]+)'}  
]
```



Amazon  
CloudWatch Logs

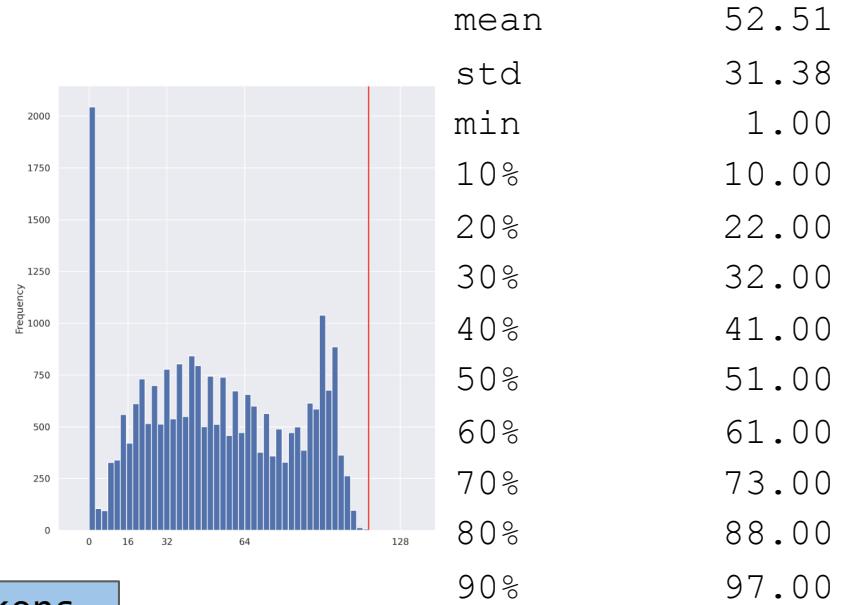
# Steps



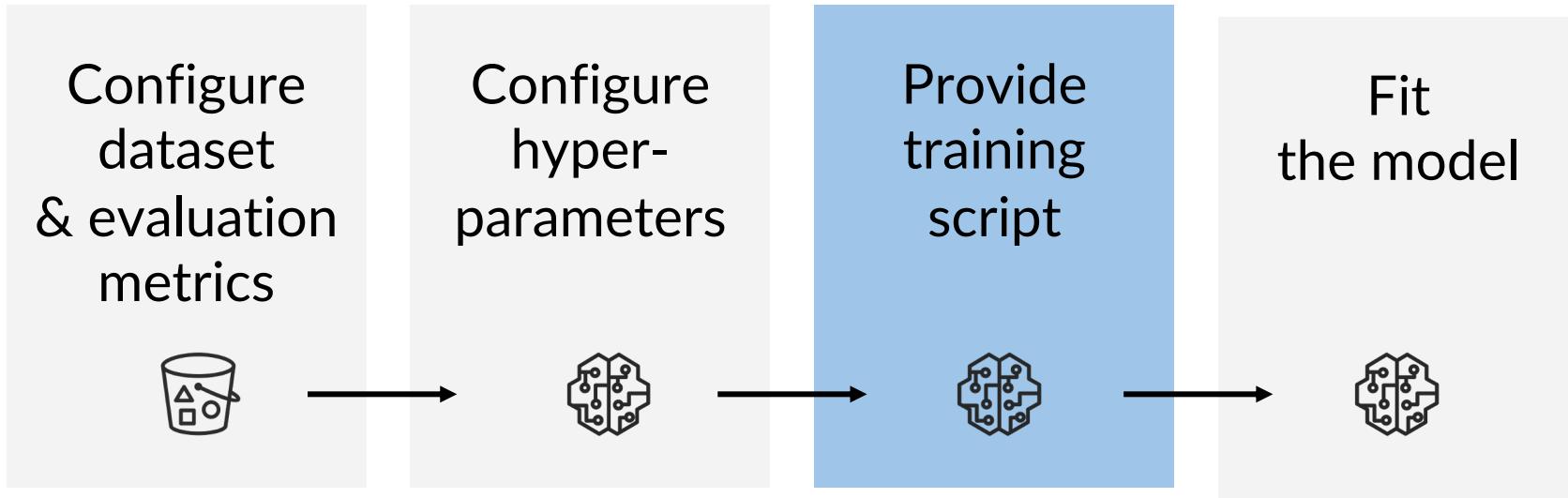
# Configure model hyperparameters

```
hyperparameters={  
    'epochs': 3,  
    'learning_rate': 2e-5,  
    'train_batch_size': 256,  
    'train_steps_per_epoch': 50,  
    'validation_batch_size': 256,  
    'validation_steps_per_epoch': 50,  
    ...  
    'max_seq_length': 128  
}
```

Max. number of input tokens  
passed to BERT model



# Steps



# Provide training script `src/train.py`

```
from transformers import RobertaModel, RobertaConfig  
from transformers import RobertaForSequenceClassification  
...  
  
config = RobertaConfig.from_pretrained('roberta-base', num_labels=3,  
                                         id2label={ 0: -1, 1: 0, 2: 1},  
                                         label2id={-1: 0, 0: 1, 1: 2})
```

Import Hugging Face transformer libraries  
(`pip install transformers`)

Download model config

Specify number of labels and `id2label` mappings

# Provide training script *src/train.py*

```
model = RobertaForSequenceClassification.from_pretrained(  
    'roberta-base',  
    config=config)
```

...

Download  
pre-trained model

```
model = train_model(model, ...)
```

Fine-tune model



Hugging Face  
Model Hub

# Provide training script *src/train.py*

```
def train_model(model, train_data_loader, df_train, val_data_loader, df_val, args):  
    loss_function = nn.CrossEntropyLoss()  
    optimizer = optim.Adam(params=model.parameters(), lr=args.learning_rate)  
    ...
```

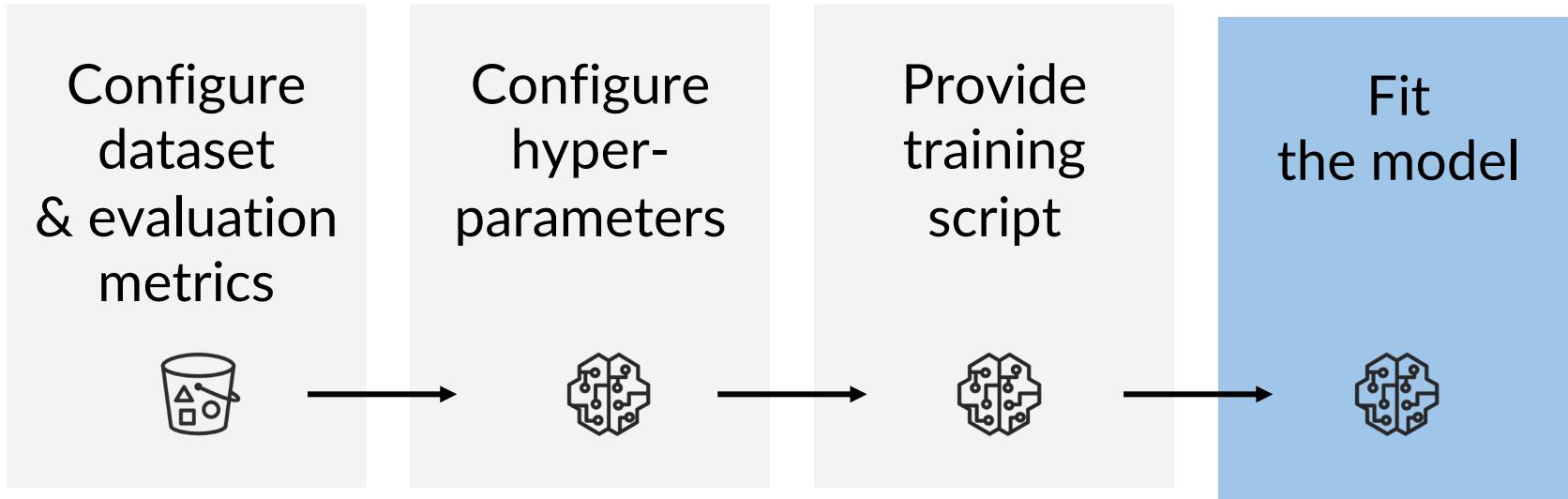
Define loss function

Creating optimizer

# Provide training script `src/train.py`

```
for epoch in range(args.epochs):
    print('EPOCH -- {}'.format(epoch))
    for i, (sent, label) in enumerate(train_data_loader):
        if i < args.train_steps_per_epoch:
            model.train()                                Put model in train mode
            optimizer.zero_grad()                         Clear gradients
            sent = sent.squeeze(0)
            output = model(sent)[0]
            _, predicted = torch.max(output, 1)           Get prediction result
            loss = loss_function(output, label)
            loss.backward()                               Compute gradients via
            optimizer.step()                            backpropagation
            ...
return model                                         Update the parameters
```

# Steps



# Fit the model

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
```

```
estimator = PyTorchEstimator(  
    entry_point='train.py',  
    source_dir='src',  
    role=role,  
    instance_count=1,  
    instance_type='ml.c5.9xlarge',  
    ...)
```

Model training script

Directory containing additional assets such as requirements.txt

Compute instance types and count

# Fit the model

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
```

```
estimator = PyTorchEstimator(
```

```
...
```

```
framework_version=<PYTORCH_VERSION>,  
hyperparameters=hyperparameters,  
metric_definitions=metric_definitions)
```

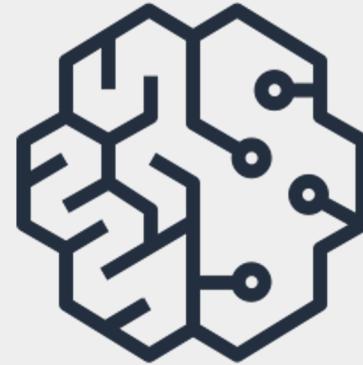
PyTorch version

Pass in hyperparameters  
and metrics

```
estimator.fit(...)
```

Start model training

# Debug and profile models



# Debug and profile models

Detect common  
training errors

"Are my gradient values  
becoming too large or too small?"



# Detect common training errors

- Vanishing gradients
- Exploding gradients
- Bad initialization
- Overfitting
- ...

# Debug and profile models



Detect common  
training errors

"Are my gradient values  
becoming too large or too small?"

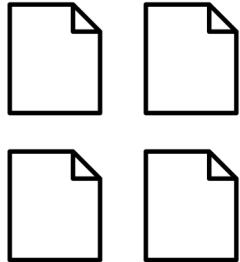
Monitor and profile  
system resource  
utilization

"How much GPU, CPU, network, and  
memory does my model training  
consume?"

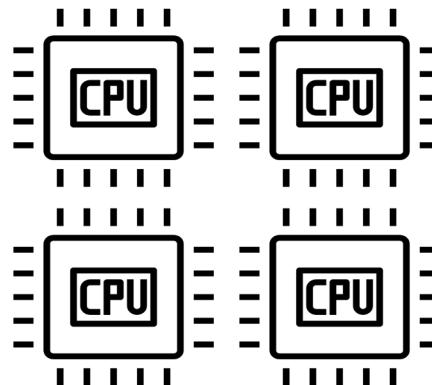
# Monitor and profile system resource utilization

Potential bottlenecks:

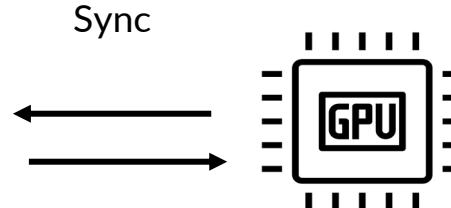
I/O usage



CPU and memory usage



GPU usage



Data storage

Data loading

Data preprocessing

Batch

Training

# Debug and profile models



Detect common training errors

"Are my gradient values becoming too large or too small?"

Monitor and profile system resource utilization

"How much GPU, CPU, network, and memory does my model training consume?"

Analyze errors and take action

"Stop the model training if the model starts overfitting!"

# Analyze errors and take action



- Stop model training when an issue is found



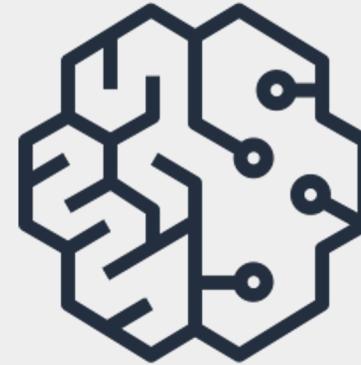
- Send a notification via email when an issue is found



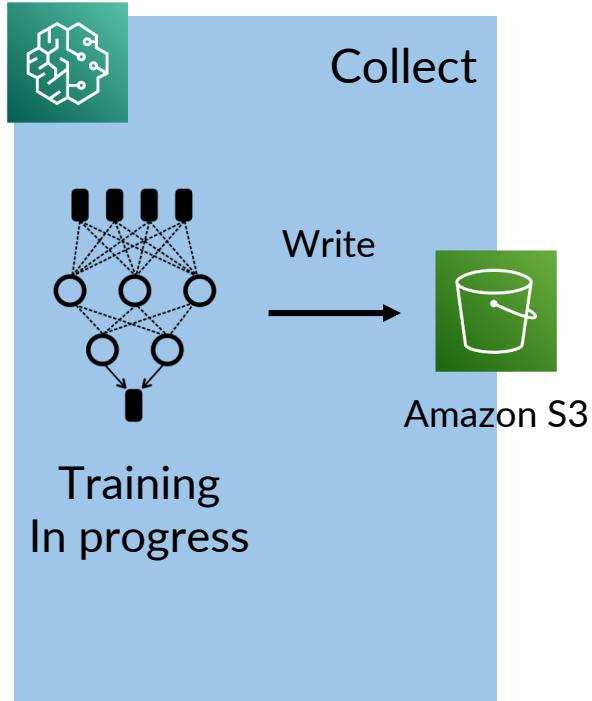
- Send a notification via text message when an issue is found

# Debug and profile models

with Amazon SageMaker  
Debugger



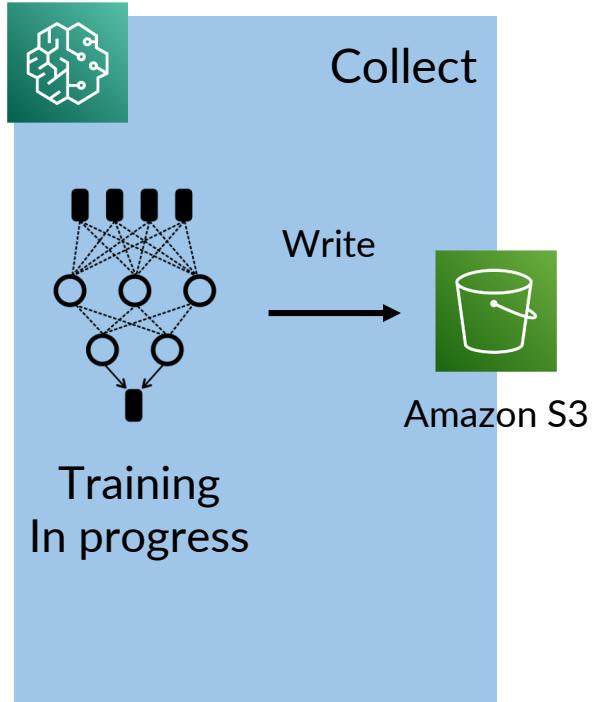
# Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- System metrics
- Framework metrics
- Output tensors

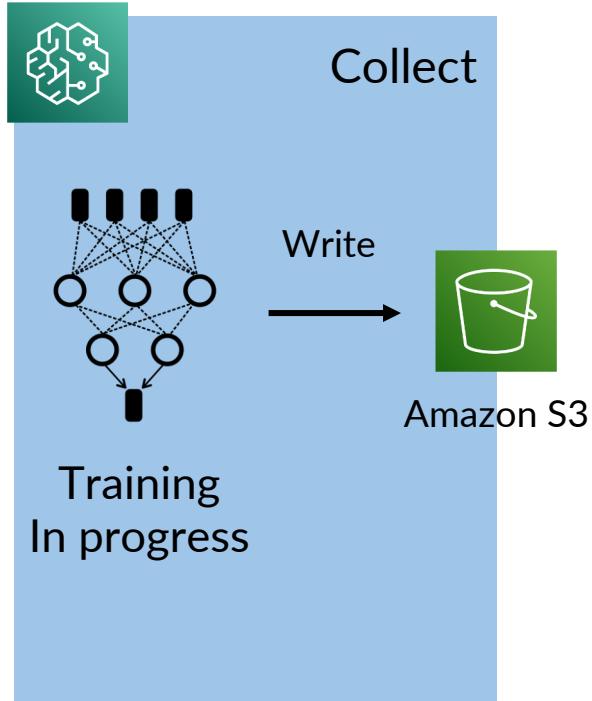
# Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- System metrics
  - CPU and GPU (and memory) utilization
  - Network metrics
  - Data input and output (I/O) metrics

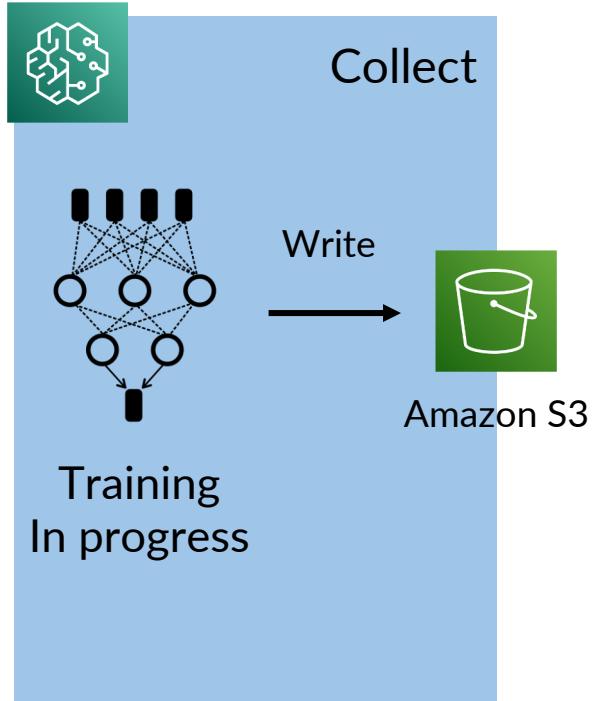
# Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- Framework metrics
  - Convolutional operations in forward pass
  - Batch normalization operations in backward pass
  - Data loader processes between steps
  - Gradient descent algorithm operations

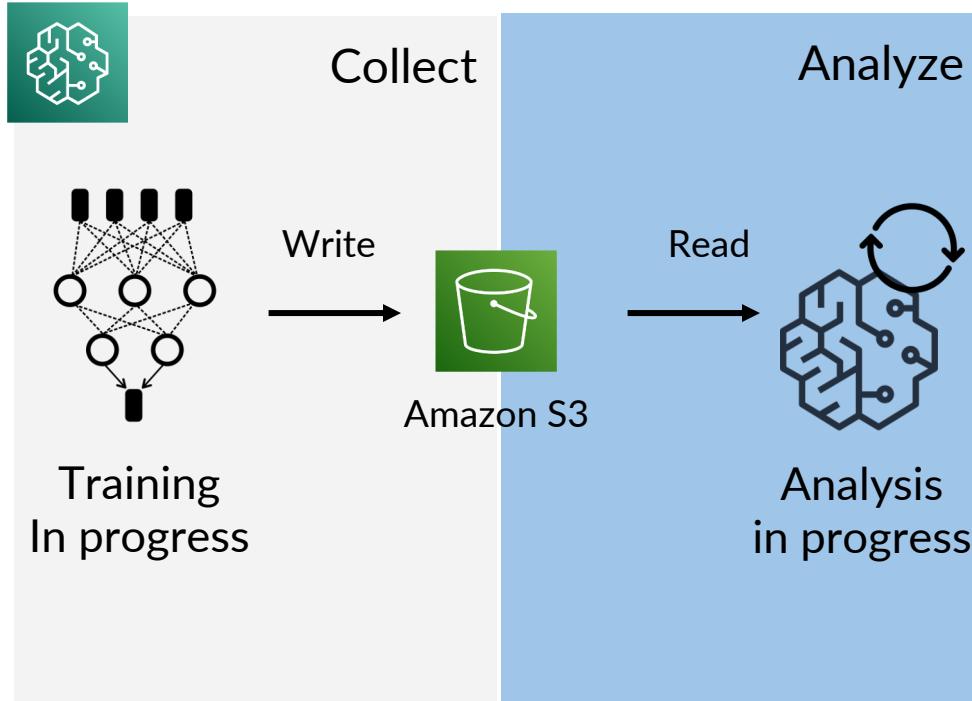
# Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- Output tensors
  - Scalar values (accuracy and loss)
  - Matrices (weights, gradients, input layers, and output layers)

# Amazon SageMaker Debugger



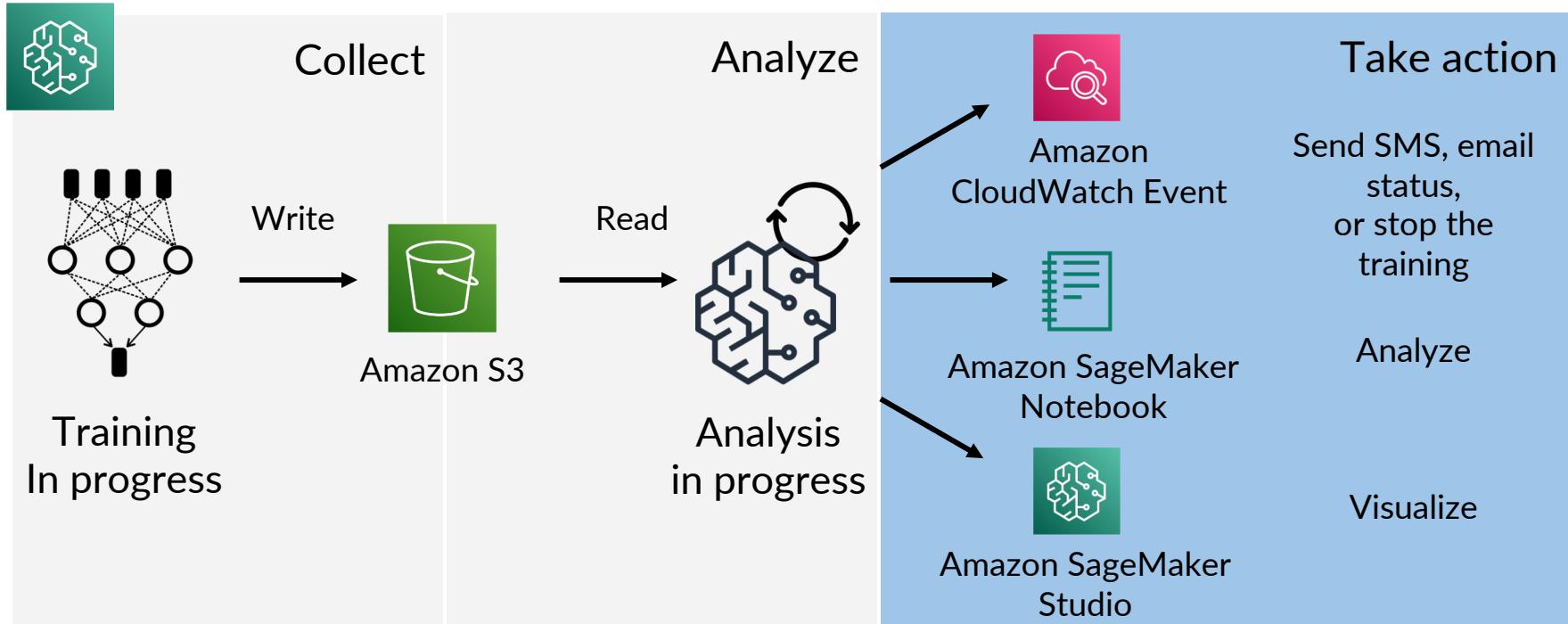
Continuous analysis  
through rules

# Amazon SageMaker Debugger: Built-in rules

Problem class	Rules
Datasets	Class imbalance Data not normalized Ratio of tokens in sequence
Loss and accuracy	Loss not decreasing Overfitting Underfitting Overtraining Classifier confusion
Weights	Poor initialization Updates too small

Problem class	Rules
Tensor	All values zero Variance of values too small Values not changing across steps
Activation function	Tanh saturation Sigmoid saturation Dying ReLU
Decision trees	Depth of tree too large Low feature importance
Gradients	Vanishing Exploding

# Amazon SageMaker Debugger



# Code example

# Debug model training: Configuration

```
from sagemaker.debugger import Rule, rule_configs  
rules=[  
    Rule.sagemaker(rule_configs.loss_not_decreasing()),  
    Rule.sagemaker(rule_configs.overtraining())  
    ...  
]
```

Select (built-in) rules to evaluate

Overtraining prevents model from overfitting

# Debug model training: Add config to estimator

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator  
estimator = PyTorchEstimator(  
    entry_point='train.py',  
    ...  
    rules=rules  
)
```

Pass rules  
in estimator

# Profile training job: Configuration

```
from sagemaker.debugger import ProfilerRule, rule_configs  
rules=[  
    ProfilerRule.sagemaker(rule_configs.LowGPUUtilization()),  
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),  
    ...]
```

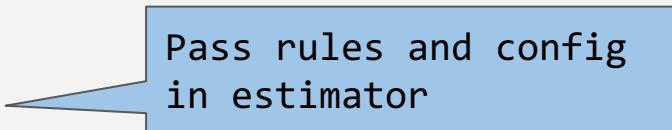
Select rules  
for profiling

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile  
profiler_config = ProfilerConfig(  
    system_monitor_interval_millis=500,  
    framework_profile_params=FrameworkProfile(num_steps=10))
```

Configure profiler

# Profile training job: Add config to estimator

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator  
estimator = PyTorchEstimator(  
    entry_point='train.py', ... ,  
    rules=rules,  
    profiler_config=profiler_config)
```



Pass rules and config in estimator

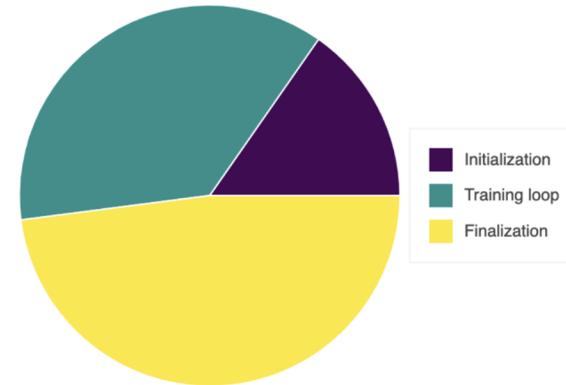
# Analyze results

# Profiling Report

## Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 05/06/2021 at 17:53:36 and ran for 323 seconds.

#	Job Statistics
0	Start time 17:53:36 05/06/2021
1	End time 17:58:59 05/06/2021
2	Job duration 323 seconds
3	Training loop start 17:54:26 05/06/2021
4	Training loop end 17:56:26 05/06/2021
5	Training loop duration 119 seconds
6	Initialization time 49 seconds
7	Finalization time 153 seconds
8	Initialization 15 %
9	Training loop 36 %
10	Finalization 47 %



# Profiling Report - Rules summary

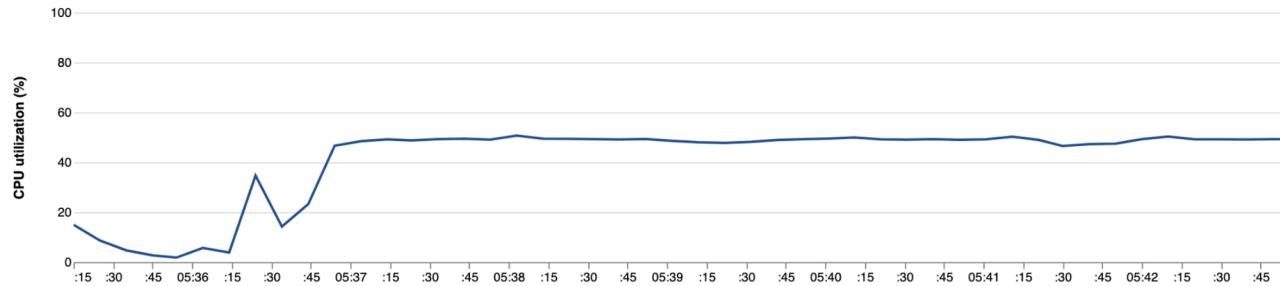
## Rules summary

The following table shows a profiling summary of the Debugger built-in rules. The table is sorted by the rules that triggered the most frequently. During your training job, the MaxInitializationTime rule was the most frequently triggered. It processed 295 datapoints and was triggered 0 times.

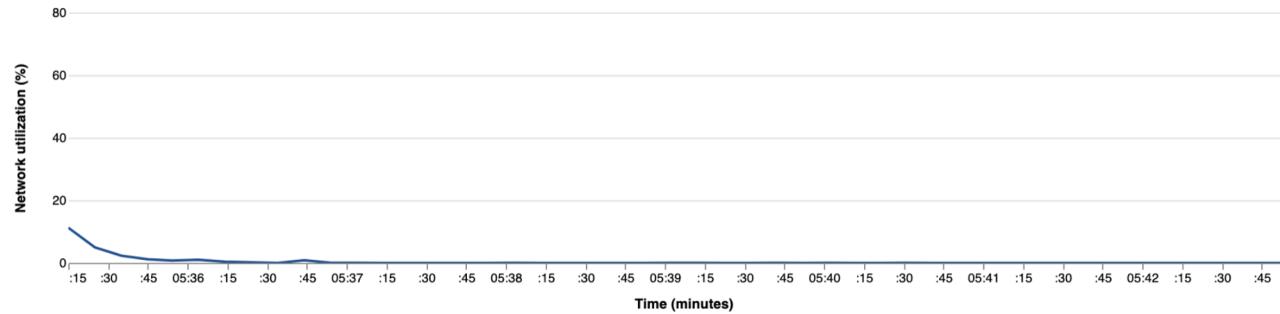
		Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
<b>MaxInitializationTime</b>		Checks if the time spent on initialization exceeds a threshold percent of the total training time. The rule waits until the first step of training loop starts. The initialization can take longer if downloading the entire dataset from Amazon S3 in File mode. The default threshold is 20 minutes.	Initialization takes too long. If using File mode, consider switching to Pipe mode in case you are using TensorFlow framework.	0	295	threshold:20
<b>GPUMemoryIncrease</b>		Measures the average GPU memory footprint and triggers if there is a large increase.	Choose a larger instance type with more memory if footprint is close to maximum available memory.	0	0	increase:5 patience:1000 window:10
<b>StepOutlier</b>		Detects outliers in step duration. The step duration for forward and backward pass should be roughly the same throughout the training. If there are significant outliers, it may indicate a system stall or bottleneck issues.	Check if there are any bottlenecks (CPU, I/O) correlated to the step outliers.	0	295	threshold:3 mode:None n_outliers:10 stddev:3
<b>BatchSize</b>		Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	0	930	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:1000 window:500

# Profiling Report - CPU/network utilization

CPU utilization over time



Network utilization over time



# Profiling Report - System utilization

System utilization over time

