Copyright Notice

Welcome to Course 3

Advanced Model Training

Machine Learning Workflow

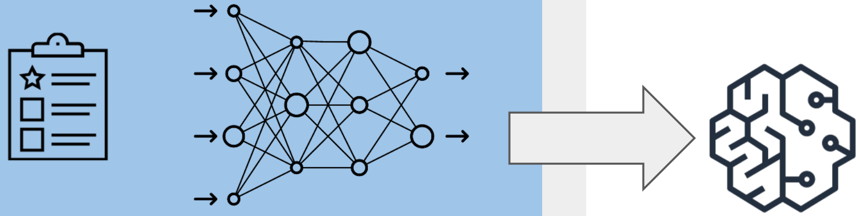
Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Model Tuning

Model Tuning

Model parameters

roberta-base
125M parameters



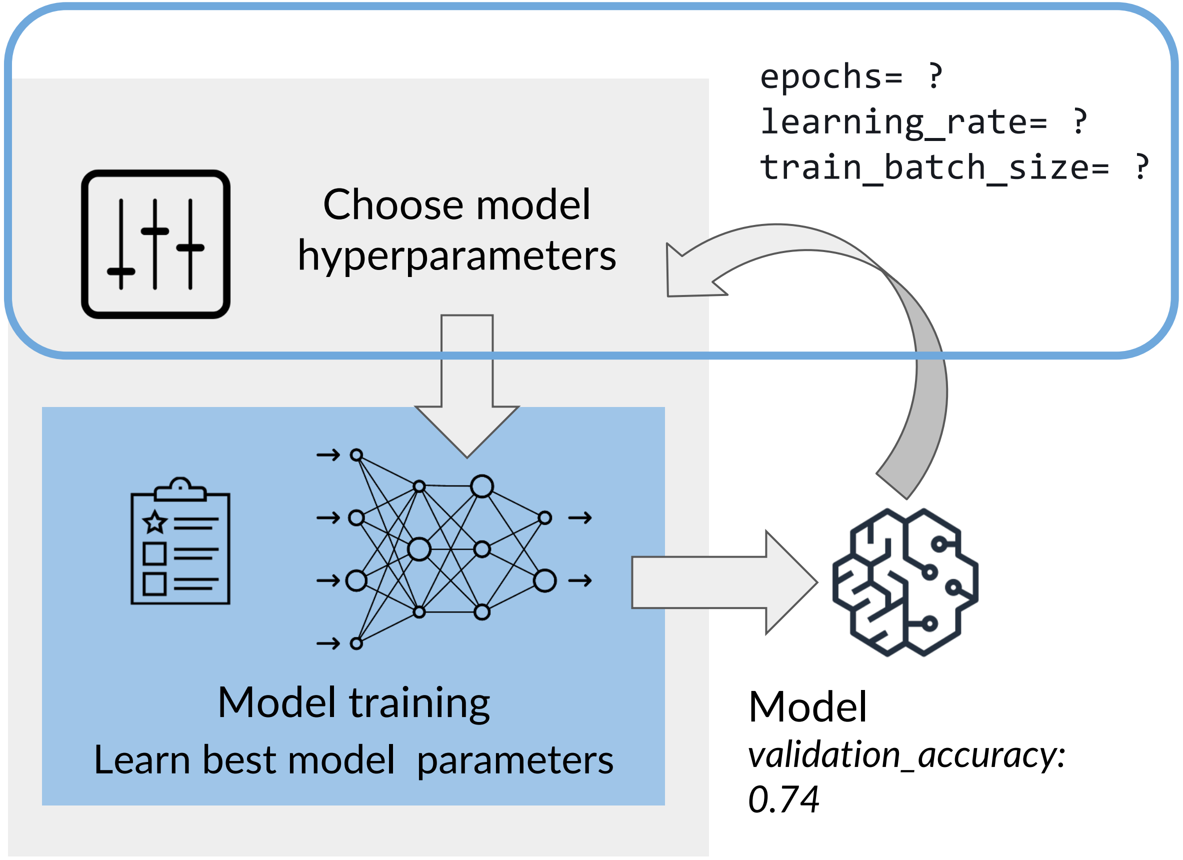
Model training
Learn best model parameters

Model
validation_accuracy: 0.74

Model Tuning

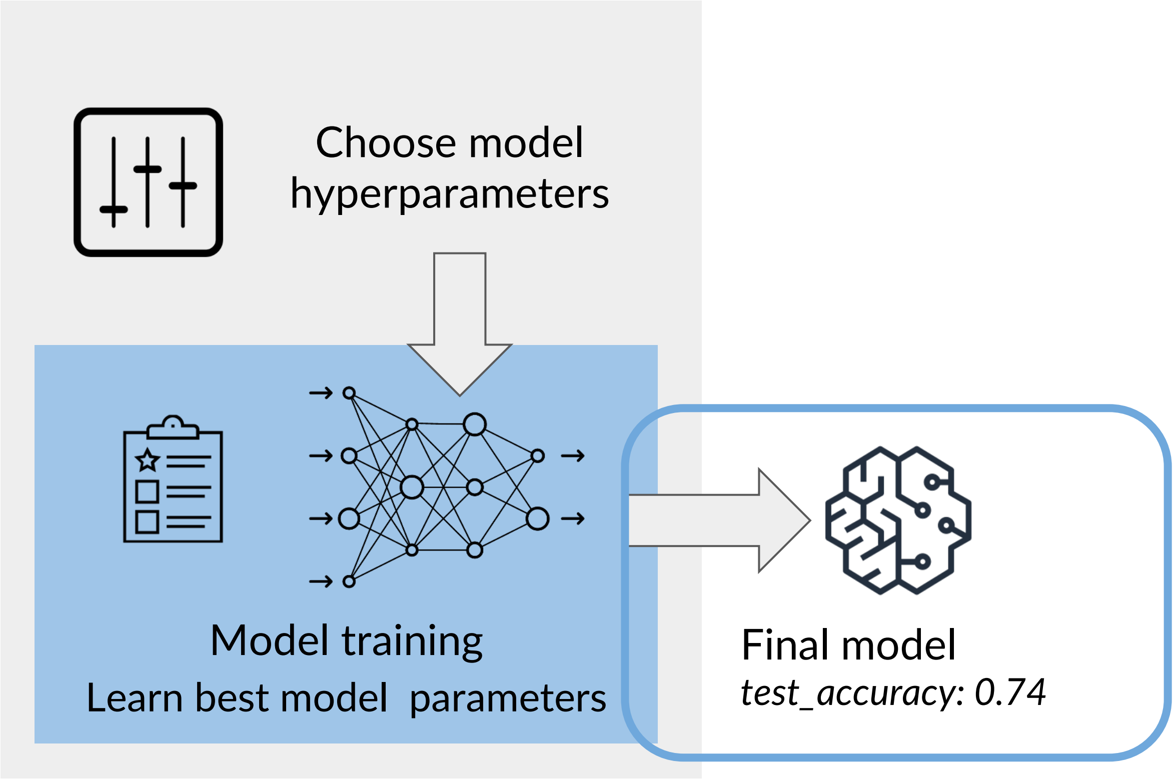
Model hyperparameters

Model parameters

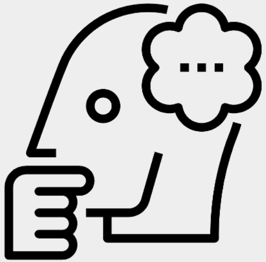


Model Evaluation

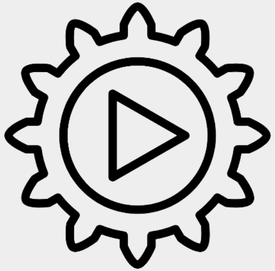
"If you can't measure it,
you can't improve it."
-- *Peter Drucker*



Manual vs. Automatic Model Tuning



Manual tuning



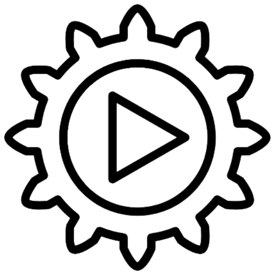
Automatic
model tuning

Popular Algorithms for Automatic Model Tuning

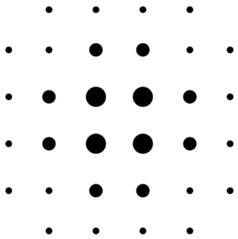
Automatic model tuning

- Grid search
- Random search
- Bayesian optimization
- Hyperband

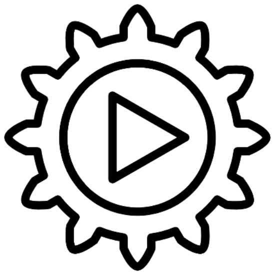
Grid Search



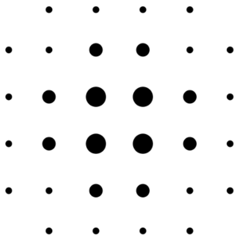
- Define sets of hyperparameters
- Test **every** combination
- Select the best performing hyperparameters



Grid Search



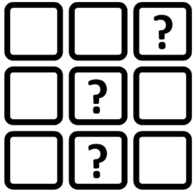
- Define sets of hyperparameters
- Test **every** combination
- Select the best performing hyperparameters



- \+ Explores all combinations
- \+ Works for small number of parameters
- \- Time-consuming
- \- Doesn't scale to large numbers of parameters

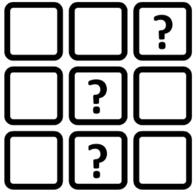
Random Search

- Define sets of hyperparameters
- Define search space & stop criteria
- Test **random** combinations within search space
- Select the best performing hyperparameters



Random Search

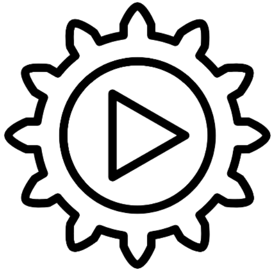
- Define sets of hyperparameters
- Define search space & stop criteria
- Test **random** combinations within search space
- Select the best performing hyperparameters



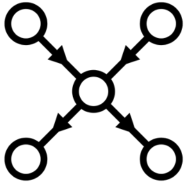
\+ Faster compared to grid search

\- Might miss better performing hyperparameters

Bayesian Optimization



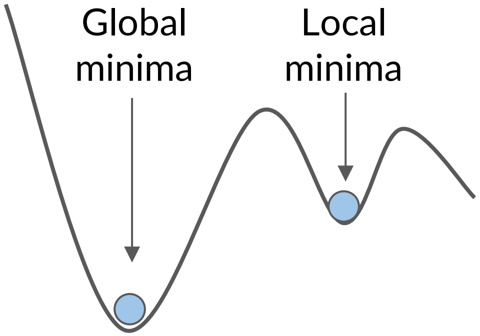
- Treat HPT like a **regression** problem (surrogate model)
- Start from random hyperparameters
- Narrow down search space around better performing hyperparameters



Bayesian Optimization

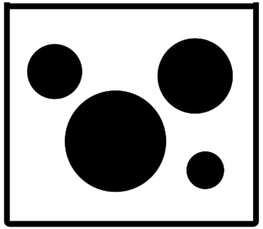
- Treat HPT like a **regression** problem (surrogate model)
- Start from random hyperparameters
- Narrow down search space around better performing hyperparameters

+ More efficient in finding best hyperparameters
- Requires sequential execution
- Might get stuck in local minima

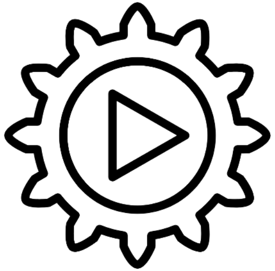


Hyperband

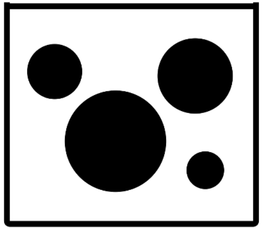
- **Bandit-based** approach
- Start from random hyperparameters
- Explore sets of hyperparameters for few iterations
- Choose best and explore longer
- Repeat until `max_iterations` reached or one candidate left



Hyperband



- **Bandit-based** approach
- Start from random hyperparameters
- Explore sets of hyperparameters for few iterations
- Choose best and explore longer
- Repeat until `max_iterations` reached or one candidate left

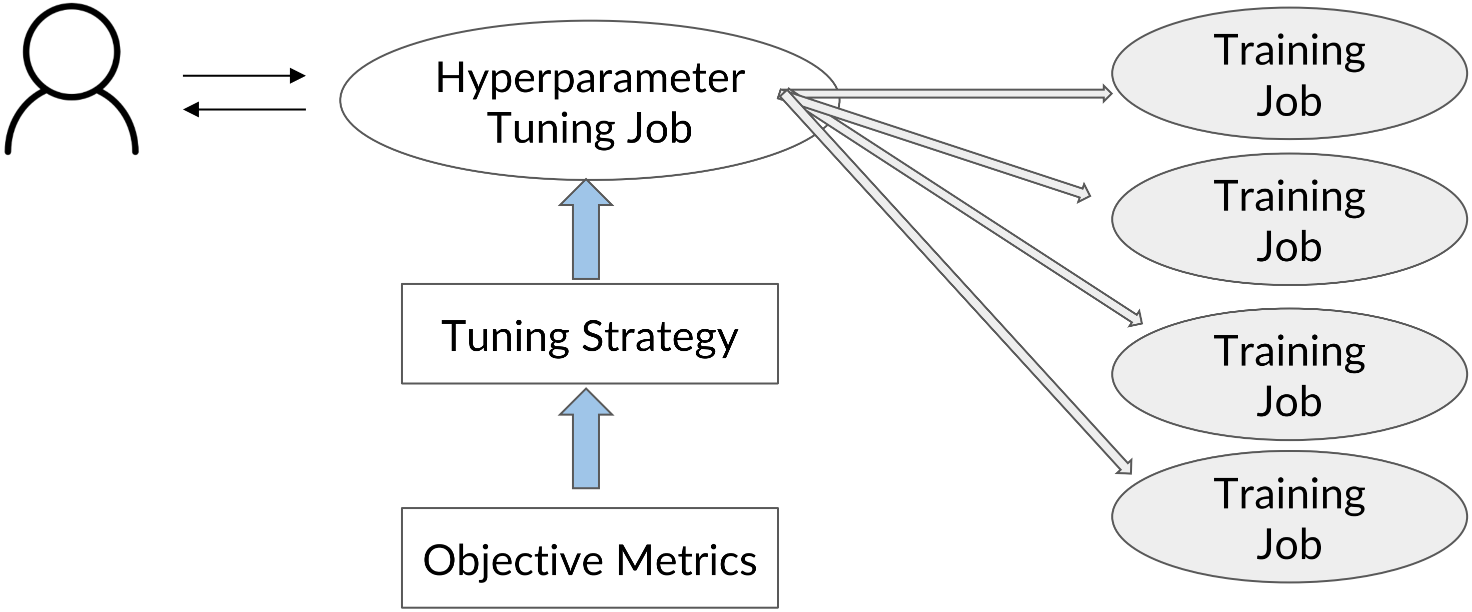


\+ Spends time efficiently (explore-exploit theory)

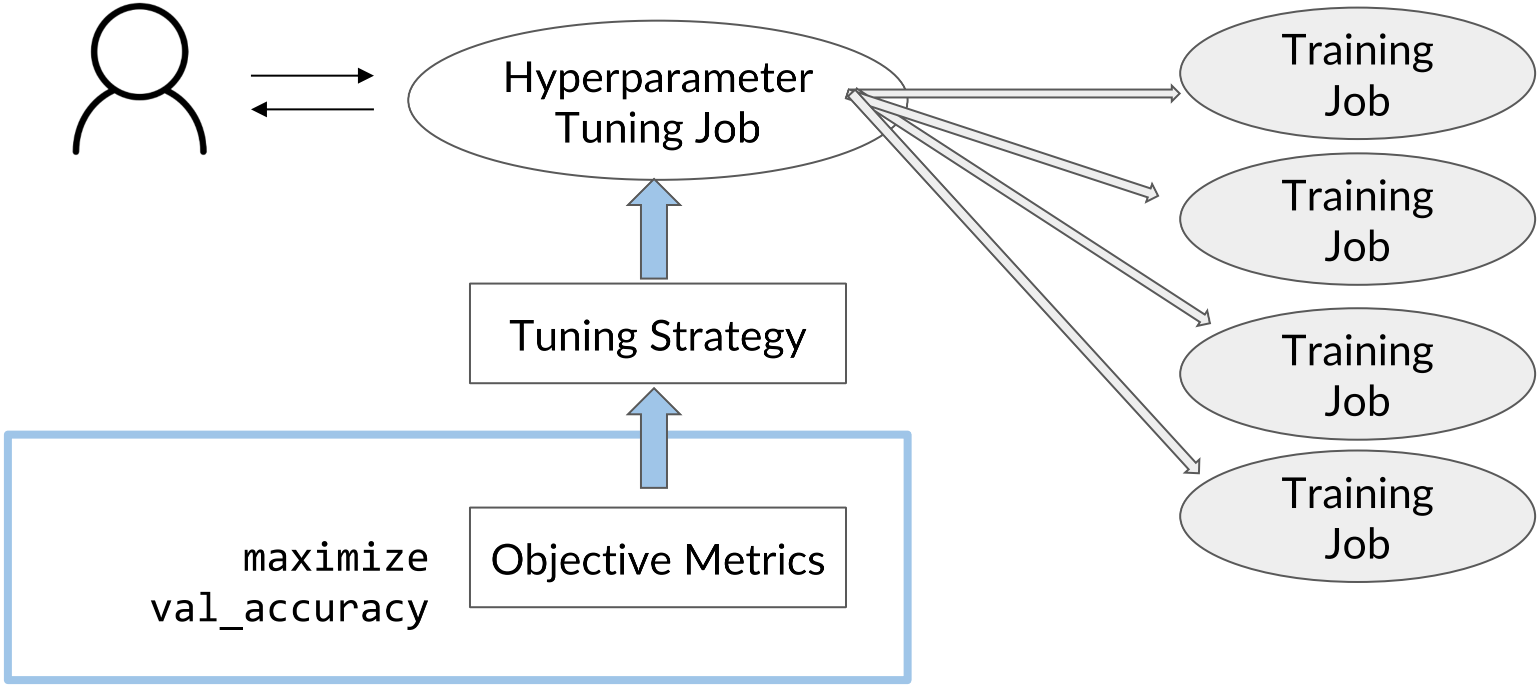
\- Might discard good candidates early that converge slowly

Tune a BERT-based Text Classifier

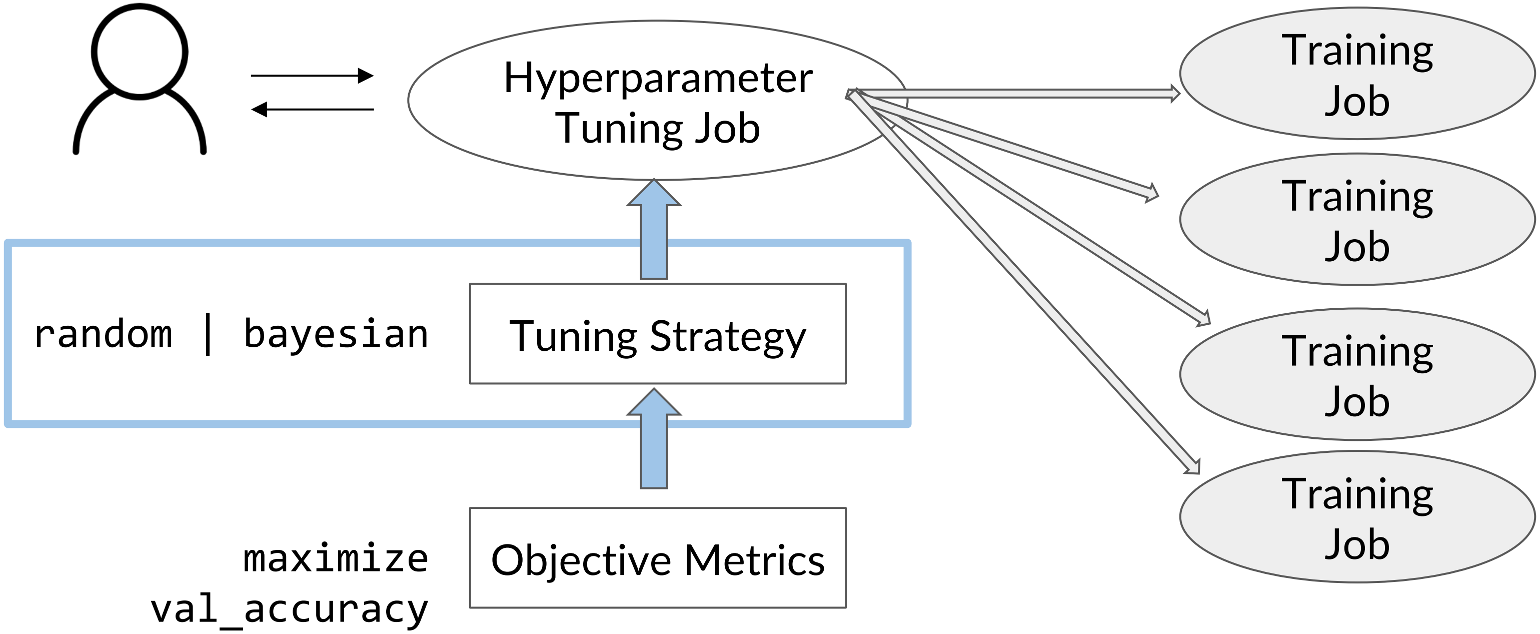
Amazon SageMaker Hyperparameter Tuning (HPT)



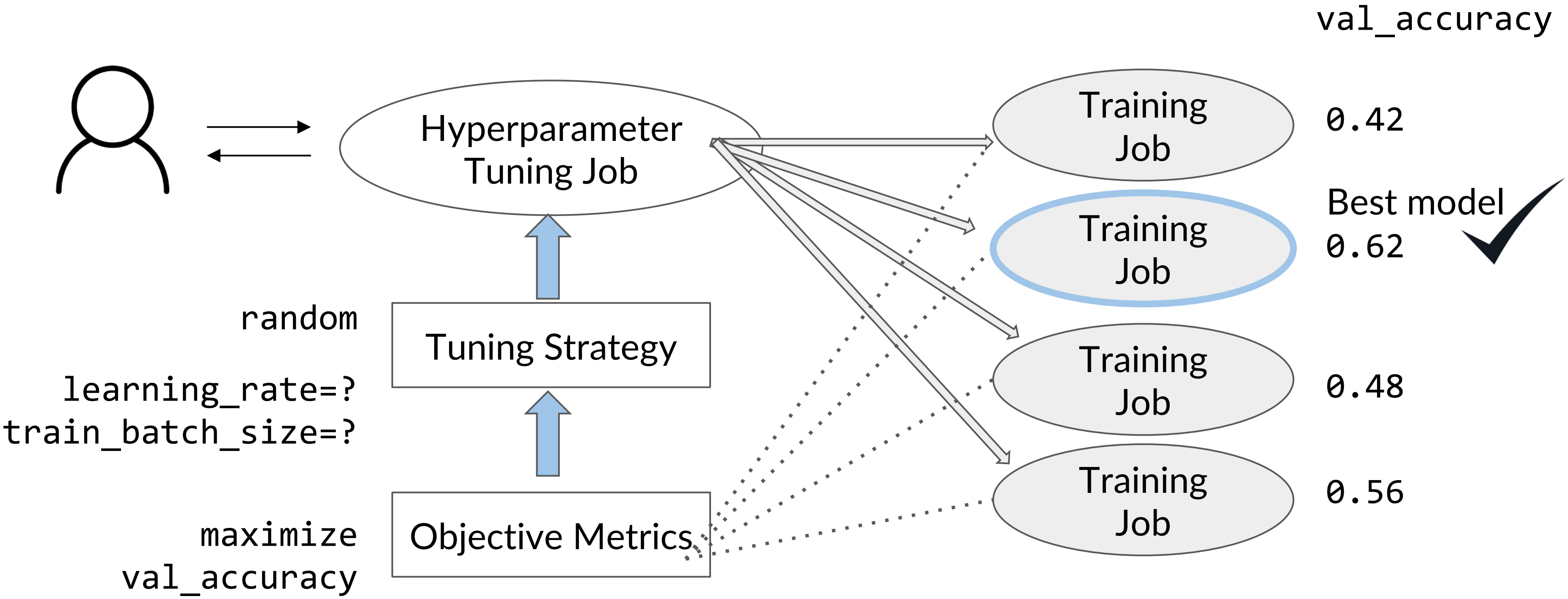
Amazon SageMaker Hyperparameter Tuning (HPT)



Amazon SageMaker Hyperparameter Tuning (HPT)



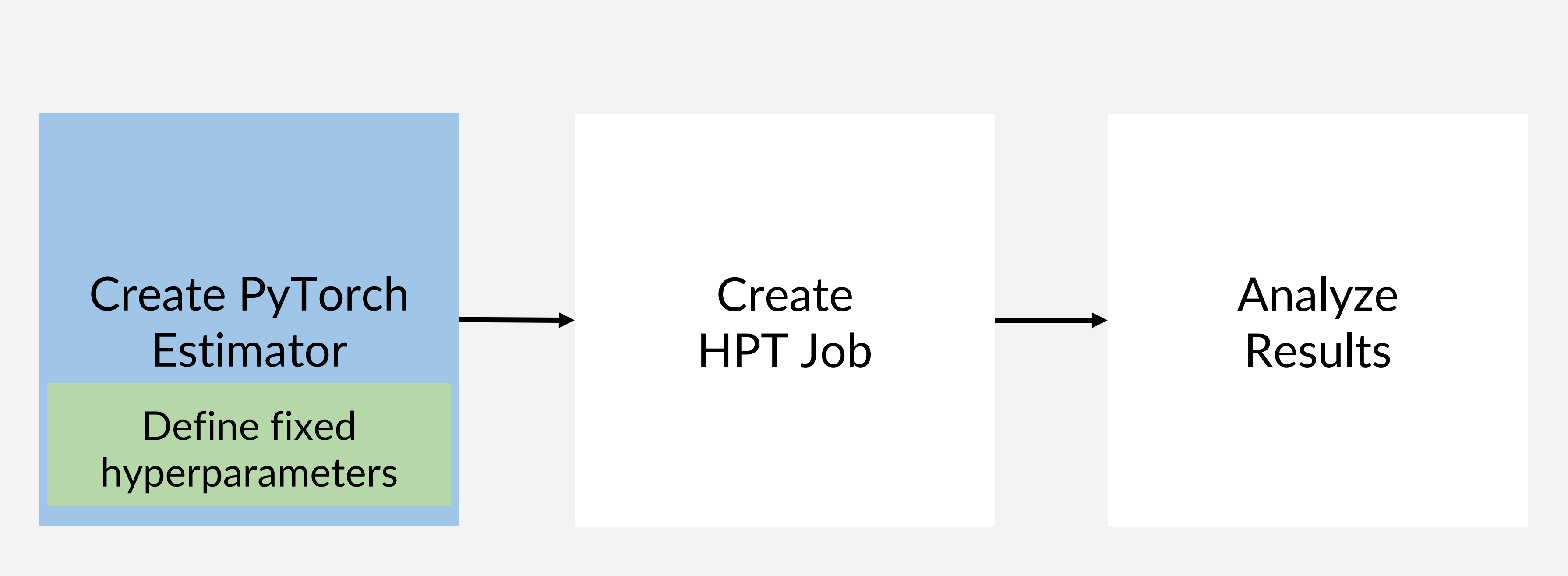
Tune BERT Text Classifier



Steps

Create PyTorch Estimator → Create HPT Job → Analyze Results

Steps


Create PyTorch Estimator
Define fixed hyperparameters
Create HPT Job
Analyze Results

Define Fixed Hyperparameters

Create PyTorch estimator

Define fixed hyperparameters

```
hyperparameters={
        'epochs': 3,
        'train_steps_per_epoch': 50,
        'validation_batch_size': 64,
        'validation_steps_per_epoch': 50,
        'freeze_bert_layer': False,
        'seed': 42,
        'max_seq_length': 64,
        'backend': 'gloo',
        'run_validation': True,
        'run_sample_predictions': False
}
```

Create PyTorch Estimator

Create PyTorch estimator

Define fixed hyperparameters

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

estimator = PyTorchEstimator(
    entry_point='train.py',
    ...
    hyperparameters=hyperparameters,

)
```

Fixed hyperparameters in estimator

Steps

Create PyTorch Estimator → Create HPT Job (Define tunable hyperparameters) → Analyze Results

Define Tunable Hyperparameters

Create HPT job

Define tunable hyperparameters

```
from sagemaker.tuner import CategoricalParameter
from sagemaker.tuner import ContinuousParameter
from sagemaker.tuner import IntegerParameter

hyperparameter_ranges = {
    'learning_rate': ContinuousParameter(0.00001, 0.00005,
scaling_type='Linear'),
    'train_batch_size': CategoricalParameter([128, 256]),
}
```

Specify parameters

Specify hyperparameter types

Specify ranges

How to Choose Hyperparameter Types

Categorical	
`'train_batch_size': CategoricalParameter([128, 256])`	`'freeze_bert_layer': CategoricalParameter([True, False])`

How to Choose Hyperparameter Types

Categorical

```
'train_batch_size':
CategoricalParameter([128, 256])
```

```
'freeze_bert_layer':
CategoricalParameter([True, False])
```

Integer

```
'train_batch_size':
IntegerParameter(16, 1024, scaling_type='Logarithmic')
```

If you need to explore large ranges quickly

How to Choose Hyperparameter Types

Categorical

```
'train_batch_size':
CategoricalParameter([128, 256])
```

```
'freeze_bert_layer':
CategoricalParameter([True, False])
```

Integer

```
'train_batch_size':
IntegerParameter(16, 1024, scaling_type='Logarithmic')
```

Continuous

```
'learning_rate':
ContinuousParameter(0.00001, 0.00005, scaling_type='Linear')
```

Create Amazon SageMaker HPT job

Create HPT job

Define tunable hyperparameters

```
from sagemaker.tuner import HyperparameterTuner

tuner = HyperparameterTuner(
    estimator=...,
    hyperparameter_ranges=...,
    objective_type=...,
    objective_metric_name=...,
    strategy=...,
)

tuner.fit(inputs={...}, ...)
```

Pass in estimator

Configure Hyperparameter ranges

Run HPT job with .fit()

Steps

Create PyTorch Estimator → Create HPT Job → Analyze Results

Analyze Results

Analyze Results

```
df_results = tuner.analytics().dataframe()
```

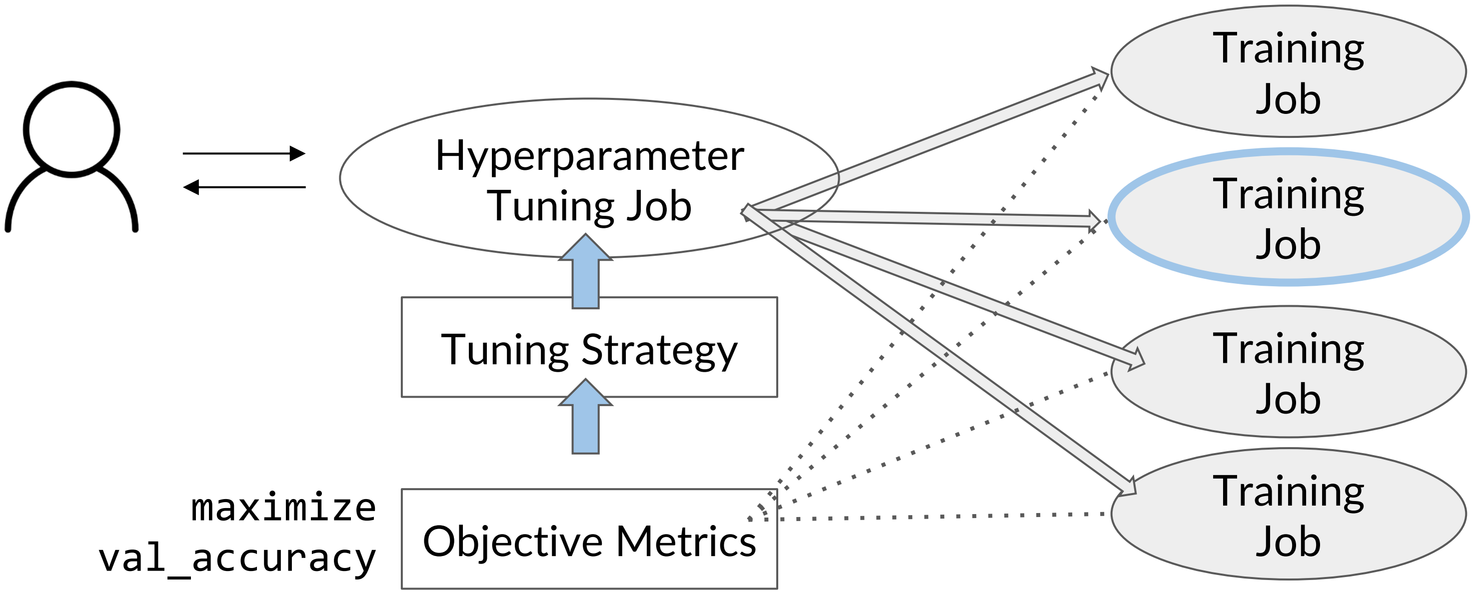
Analyze Results

Analyze Results

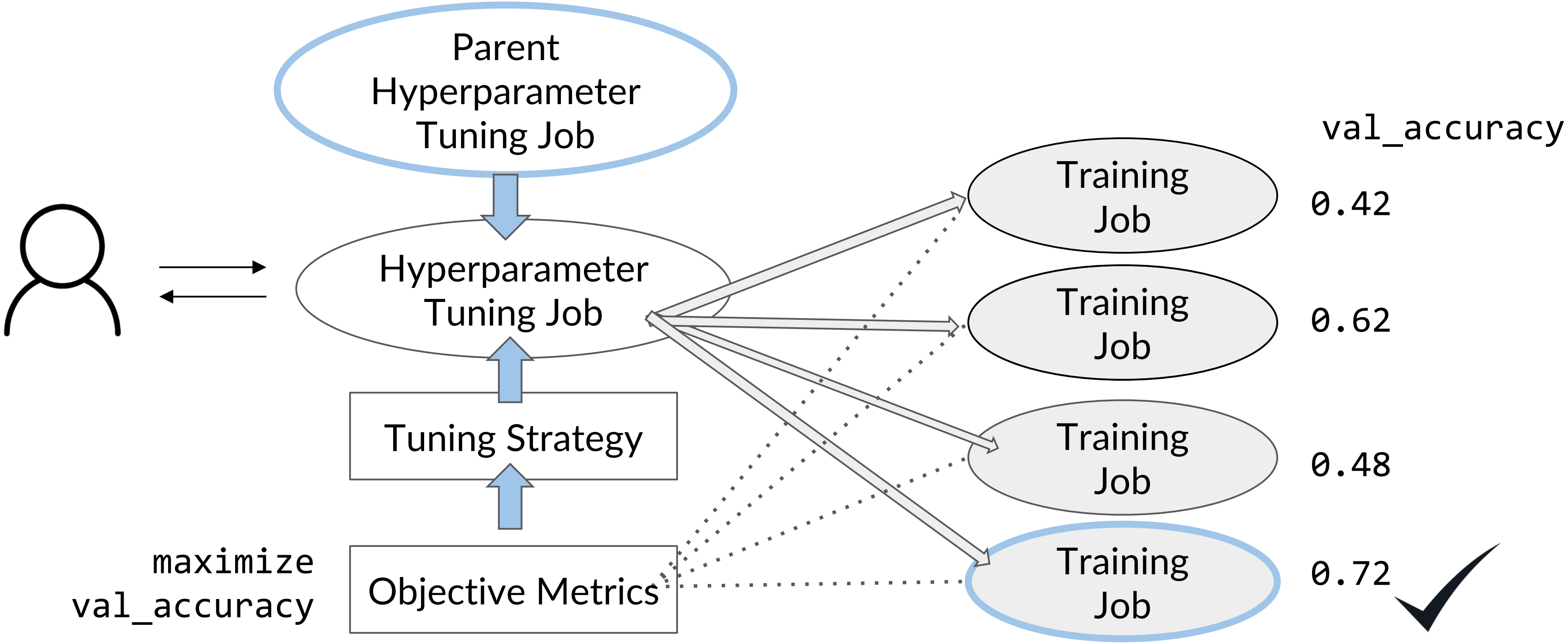
```
df_results = tuner.analytics().dataframe()
```

learning_rate	train_batch_size	TrainingJobName	TrainingJobStatus	FinalObjectiveValue
0.000021	"128"	pytorch-training-210225-1535-001-71394bc3	Completed	44.939999
0.000035	"128"	pytorch-training-210225-1535-002-cf437bad	Completed	41.580002

Warm Start HPT Job



Warm Start HPT Job



Warm Start HPT Job

- IDENTICAL_DATA_AND_ALGORITHM
  - Same input data and training data
  - Update hyperparameter tuning ranges and maximum number of training jobs
- TRANSFER_LEARNING
  - Updated training data and different version of training algorithm

Configure Warm Start

```
from sagemaker.tuner import WarmStartConfig
from sagemaker.tuner import WarmStartTypes

warm_start_config = WarmStartConfig(
     warm_start_type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
     parents=<PARENT_TUNING_JOB_NAME>)

tuner = HyperparameterTuner(
    ...
    warm_start_config=warm_start_config)

tuner.fit(...)
```

IDENTICAL_DATA_AND_ALGORITHM or TRANSFER_LEARNING

Specify parent tuning job

Pass warm start config in HyperparameterTuner

Best Practices - SageMaker HyperParameter Tuning

- Select a small number of hyperparameters
- Select a small range for hyperparameters
- Enable warm start
- Enable early stop to save tuning time and costs
- Select a small number of concurrent training jobs

Best Practices - Monitoring Training Resources

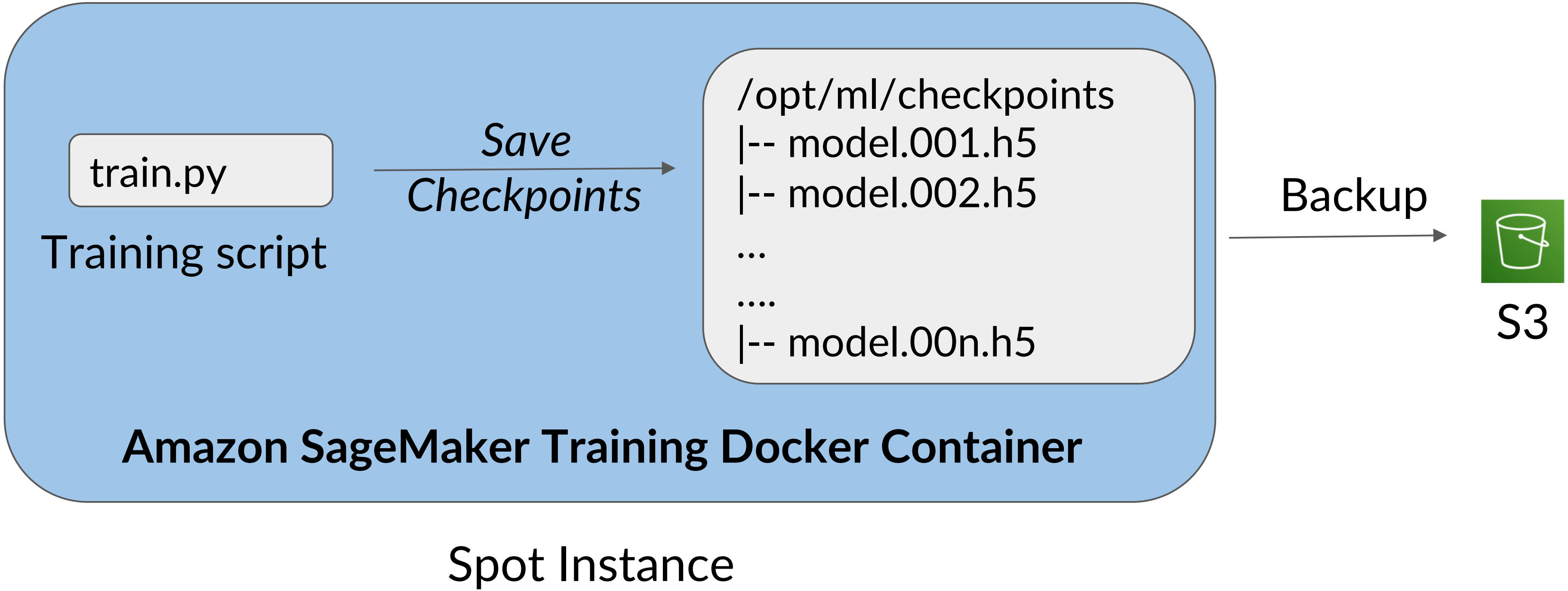
- Right size compute resources
- Requires empirical testing
- Amazon CloudWatch Metrics
- Insights from Amazon SageMaker Debugger

Checkpointing

Machine Learning Checkpointing

- Save state of ML models during training
- Checkpoints : Snapshots of the model
  - Model architecture
  - Model weights
  - Training configurations
  - Optimizer
- Frequency and number of checkpoints

Amazon SageMaker Managed Spot



Amazon SageMaker Managed Spot

train.py

Training script

Save Checkpoints

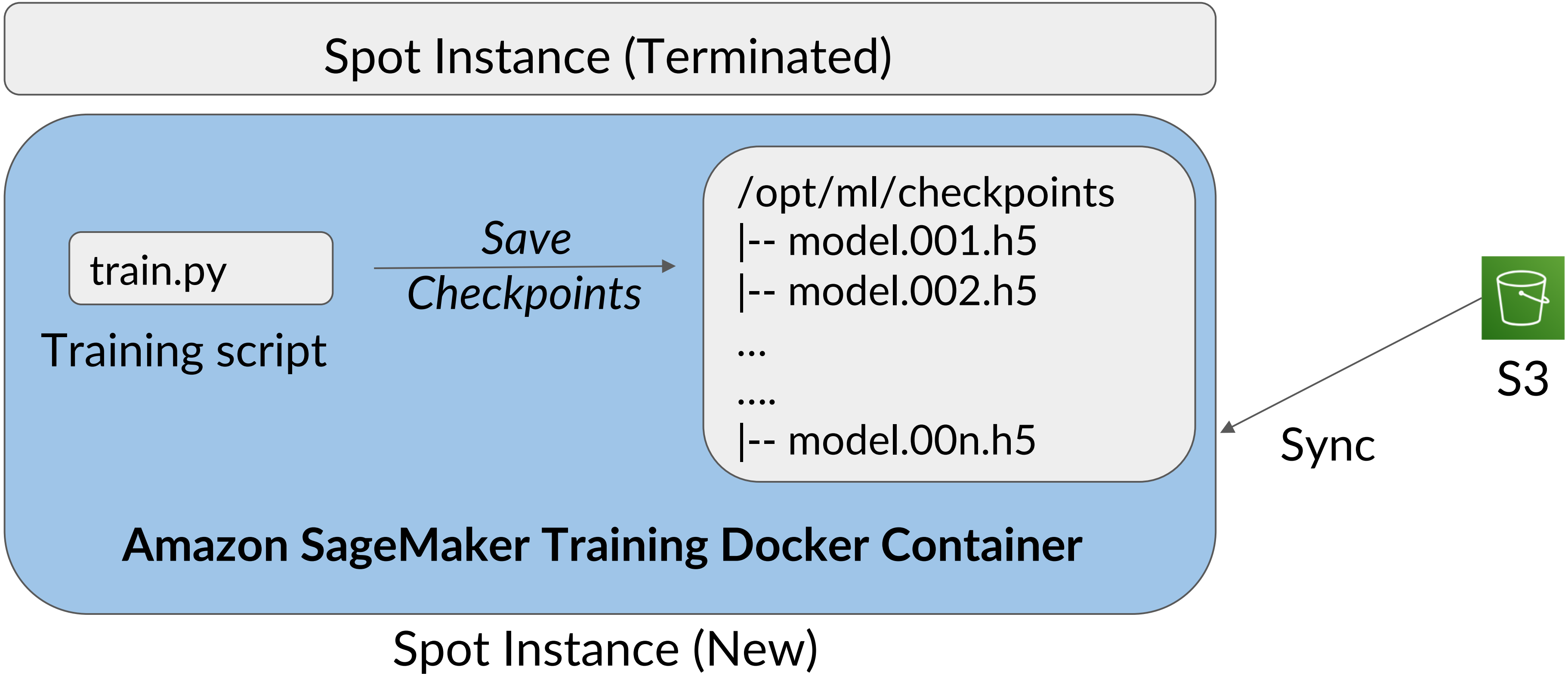
/opt/ml/checkpoints
|-- model.001.h5
|-- model.002.h5
...
....
|-- model.00n.h5

Amazon SageMaker Training Docker Container

S3

Spot Instance (Terminated)

Amazon SageMaker Spot Training



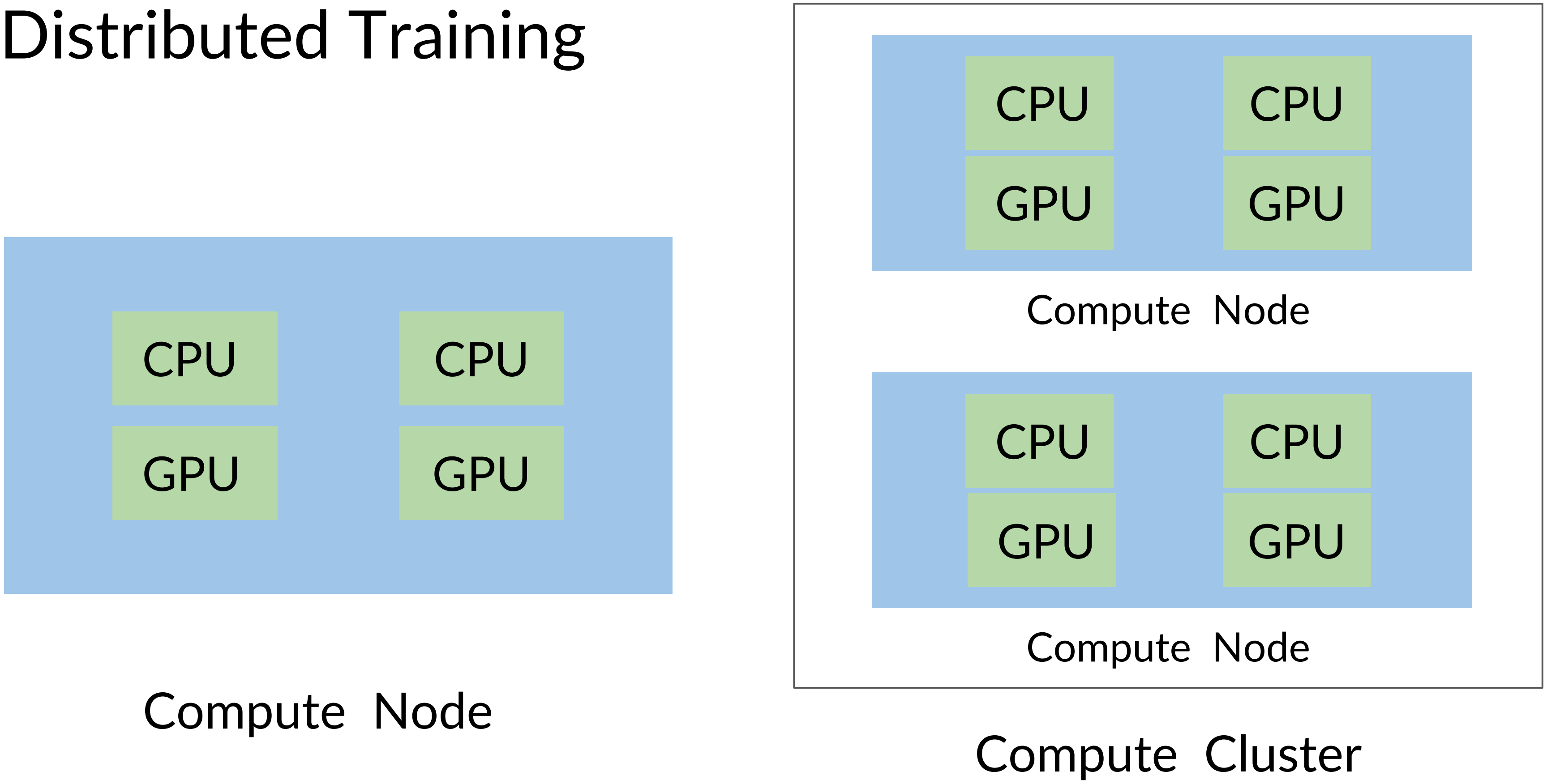
Distributed Training Strategies

Challenges

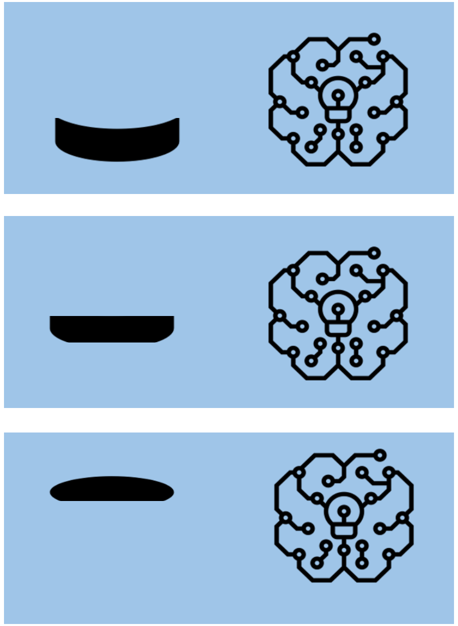
Increased training data volume

Increased model size and complexity

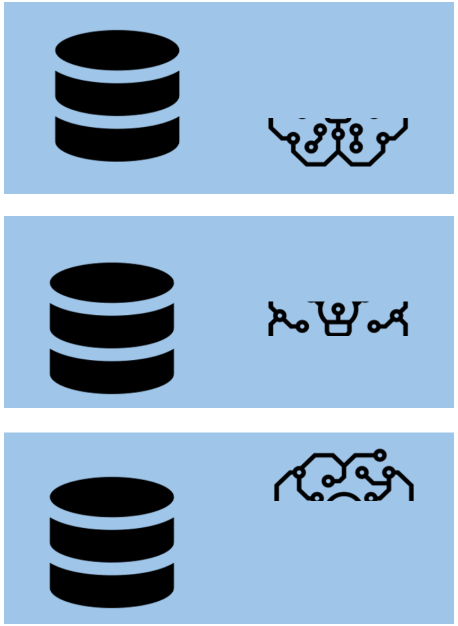
Distributed Training



Distributed Training Strategies



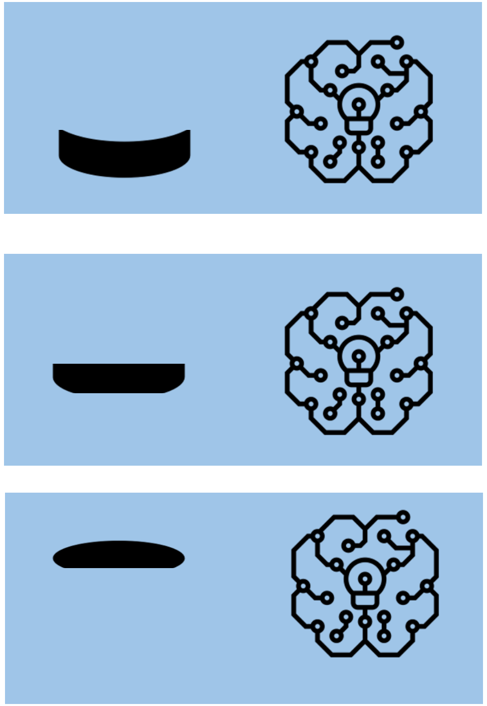
Data Parallelism



Model Parallelism

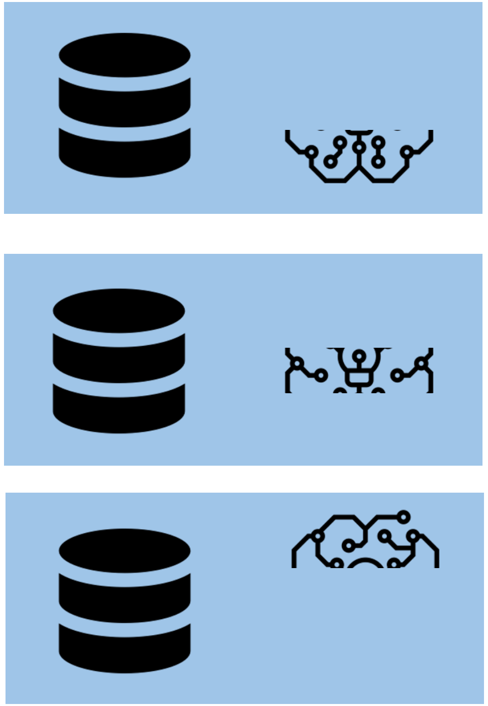
Distributed Training Strategies - Data Parallelism

- Training data split up
- Model replicated on all nodes



Distributed Training Strategies - Model Parallelism

- Training data replicated
- Model split up on all nodes



Amazon SageMaker Estimator

```
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='train.py',
    role=sagemaker.get_execution_role(),
    framework_version='1.6.0',
    py_version='py3',
    instance_count=3,
    instance_type='ml.p3.16xlarge',
    distribution={'smdistributed':{'dataparallel':{enabled': True}}}
)
estimator.fit()
```

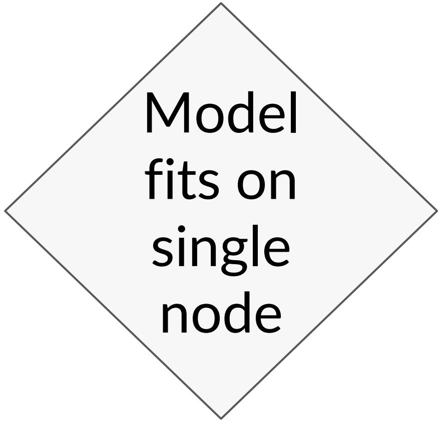
Distribution Strategy

Amazon SageMaker Estimator

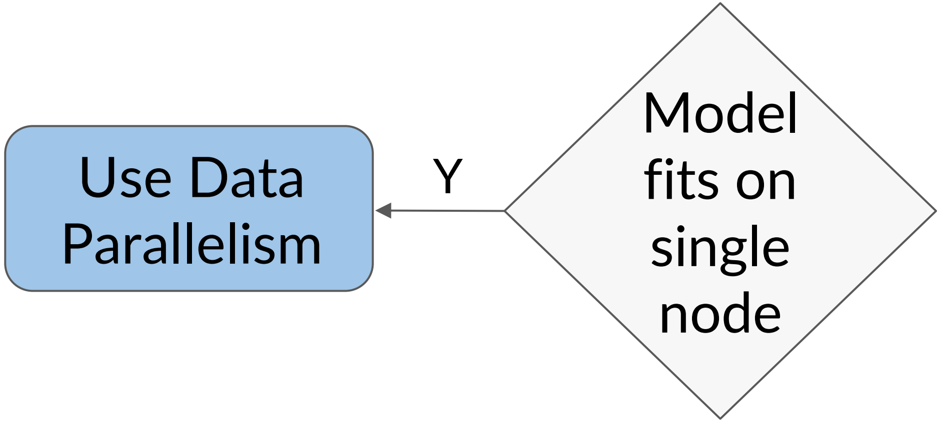
```
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='train.py',
    role=sagemaker.get_execution_role(),
    framework_version='1.6.0',
    py_version='py3',
    instance_count=3,
    instance_type='ml.p3.16xlarge',
    distribution={'smdistributed':{'modelparallel':{enabled': True}}}
)
estimator.fit()
```

Model Parallel
Distribution Strategy

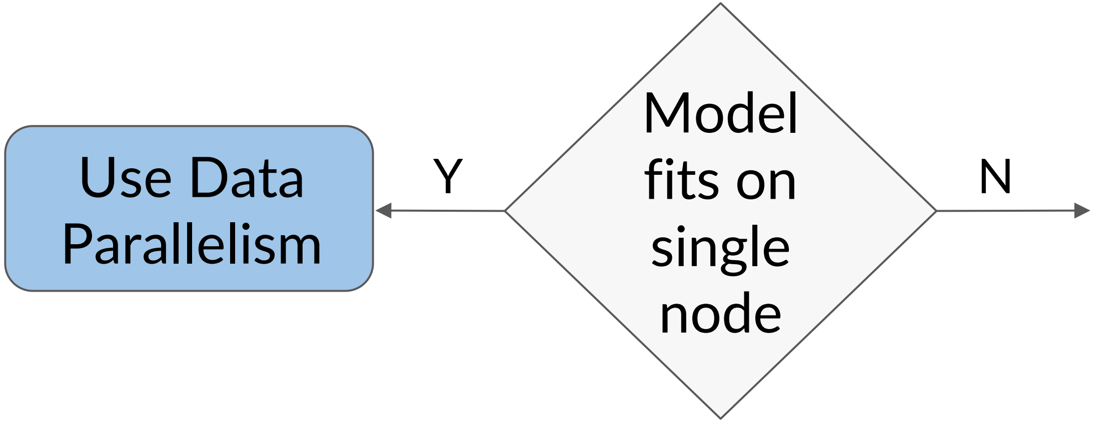
Choosing a Distribution Strategy



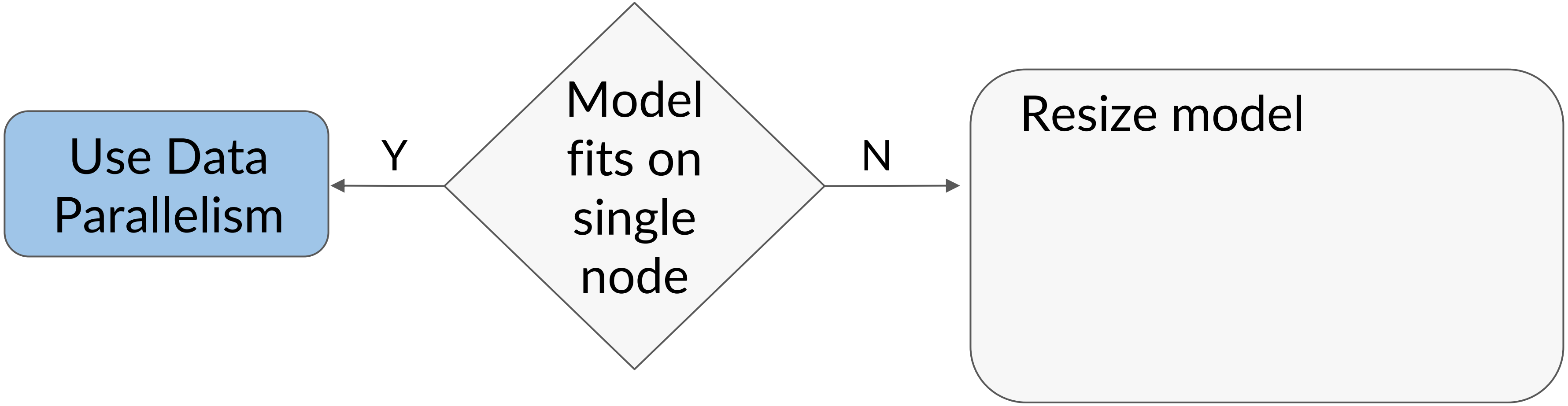
Choosing a Distribution Strategy



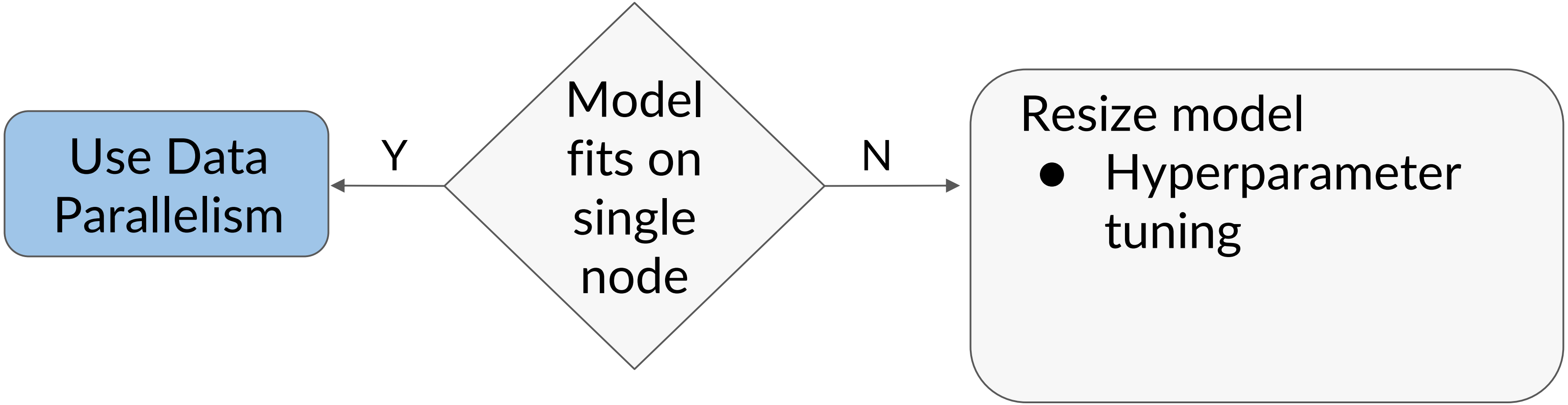
Choosing a Distribution Strategy



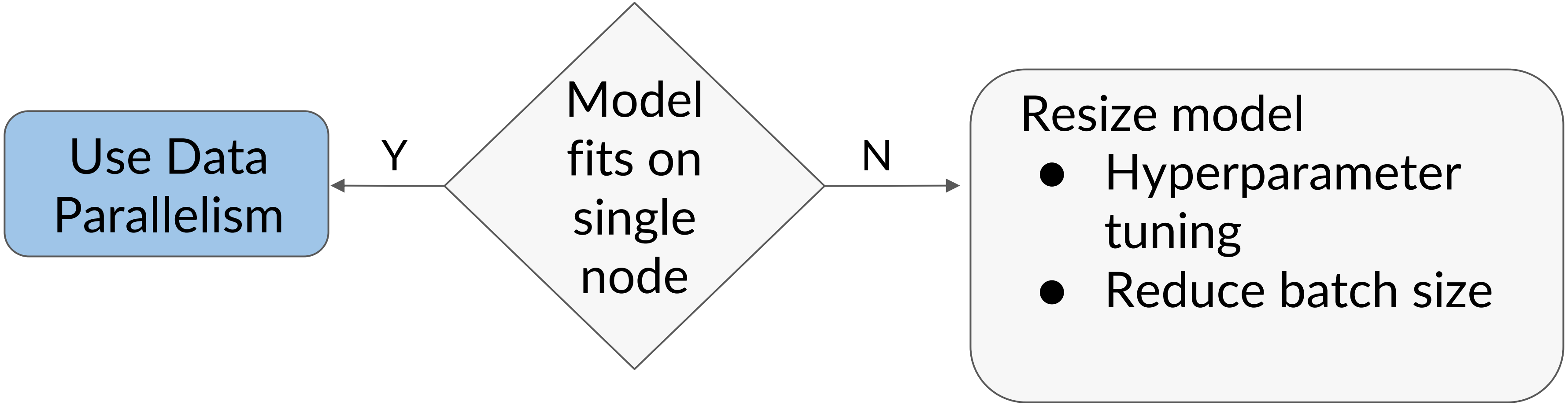
Choosing a Distribution Strategy



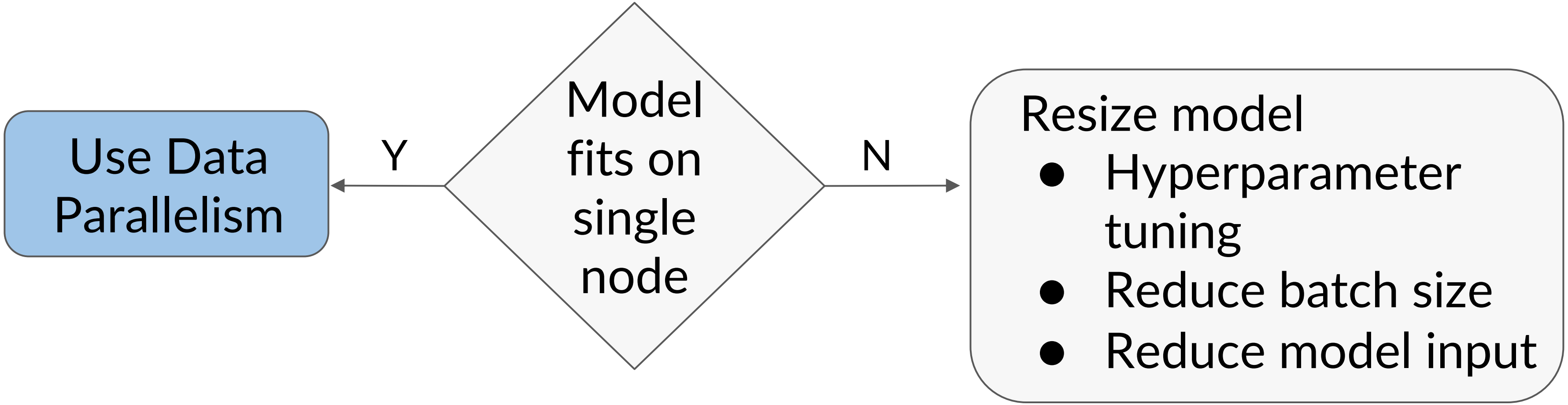
Choosing a Distribution Strategy



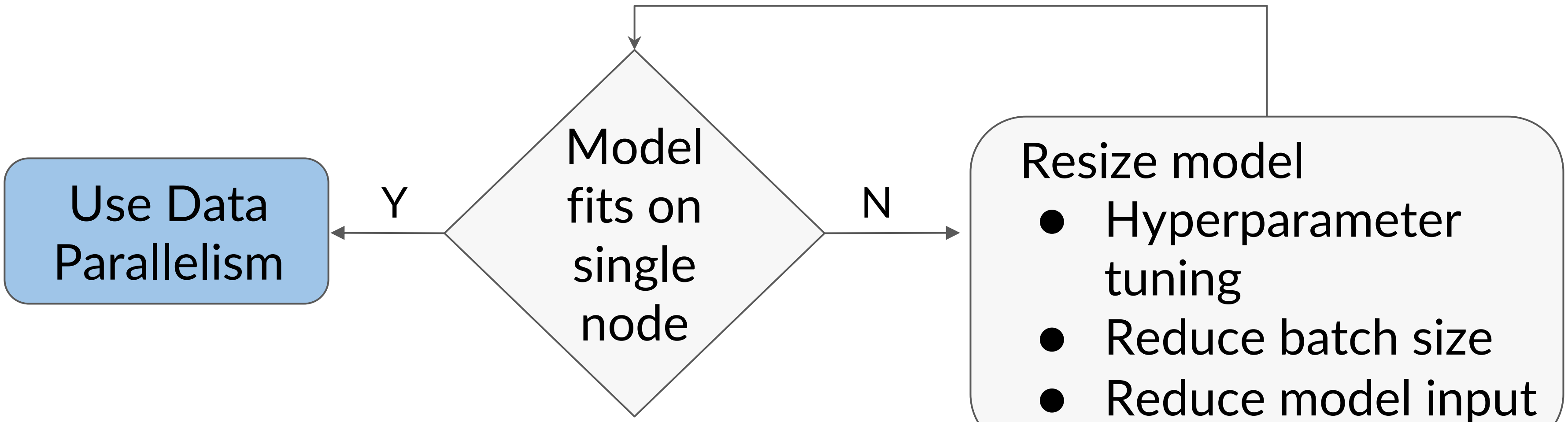
Choosing a Distribution Strategy



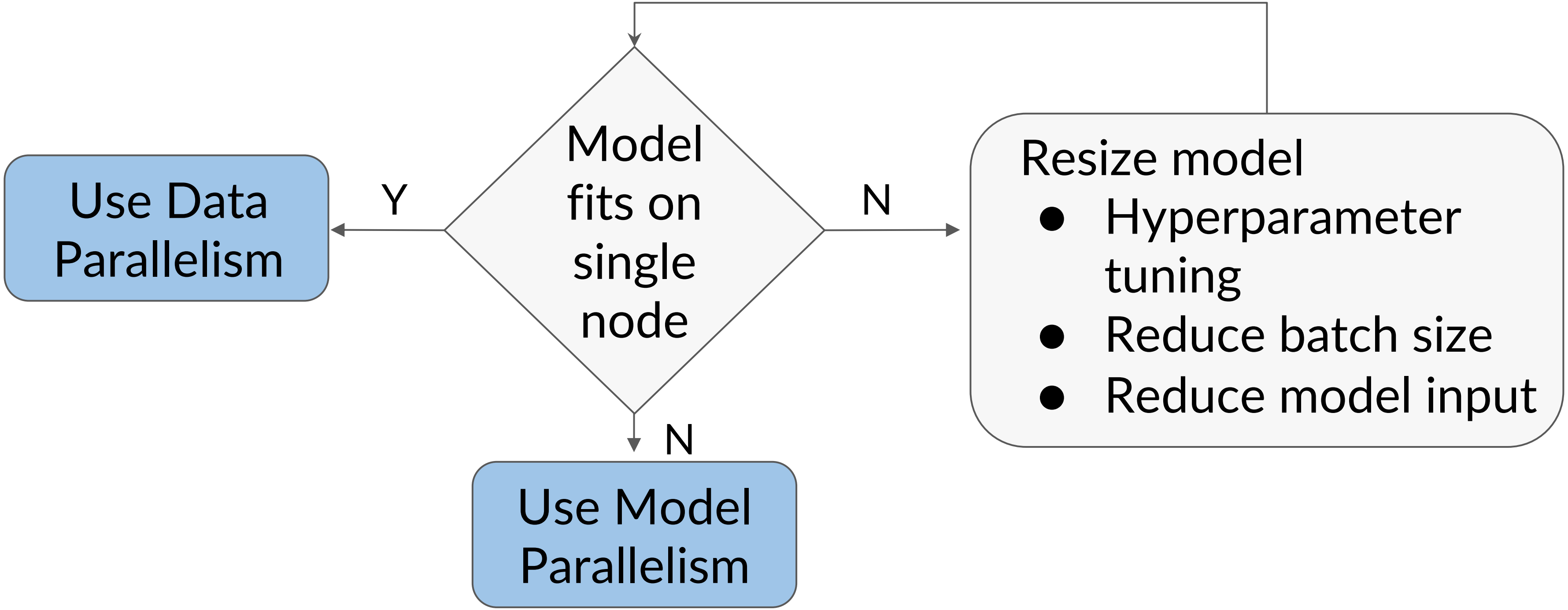
Choosing a Distribution Strategy



Choosing a Distribution Strategy



Choosing a Distribution Strategy



Custom Algorithms with Amazon SageMaker

Options on Amazon SageMaker

Amazon SageMaker	Amazon SageMaker	Amazon SageMaker
Built-in Algorithms	*Bring Your Own Script*	*Bring Your Own Container*

Less Code ⟷ More Customizable

Amazon SageMaker Estimator

```
estimator =
sagemaker.estimator.Estimator(image_uri=image_uri, ...)
estimator.set_hyperparameters(...)
estimator.fit(...)
```

Built-In Algorithms

Options on Amazon SageMaker

Amazon SageMaker

Built-in Algorithms

Amazon SageMaker

Bring Your Own Script

Amazon SageMaker

Bring Your Own Container

Less Code

More Customizable

Amazon SageMaker Estimator

```
estimator =
sagemaker.estimator.Estimator(image_uri=image_uri, ...)
estimator.set_hyperparameters(...)
estimator.fit(...)
```

Built-In Algorithms

```
from sagemaker.pytorch import PyTorch
pytorch_estimator = PyTorch(
    entry_point='train.py',
    ...
)
```

Script Mode
PyTorch Container

Training Options on Amazon SageMaker

Amazon SageMaker	Amazon SageMaker	Amazon SageMaker
Built-in Algorithms	*Bring Your Own Script*	*Bring Your Own Container*

Less Code

More Customizable

Bring Your Own Container - Steps

Code → **Containerize** → **Register with Amazon ECR** → **Create Estimator**

Bring Your Own Container - Steps

Bring Your Own Container - Steps

Code → **Containerize** → **Register with Amazon ECR** → **Create Estimator**

- Algorithm
- Training
- Inference

Bring Your Own Container - Steps

Code → **Containerize** → **Register with Amazon ECR** → **Create Estimator**

Bring Your Own Container - Steps

Code → **Containerize** → Register with Amazon ECR → Create Estimator

- algorithm_name=tf-custom-container-test
- docker build -t ${algorithm_name} .

Bring Your Own Container - Steps

Code → Containerize → **Register with Amazon ECR** → Create Estimator

Bring Your Own Container - Steps

Code → **Containerize** → **Register with Amazon ECR** → **Create Estimator**

- aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
- fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"
- docker push ${fullname}

Bring Your Own Container - Steps

Code → Containerize → Register with Amazon ECR → **Create Estimator**

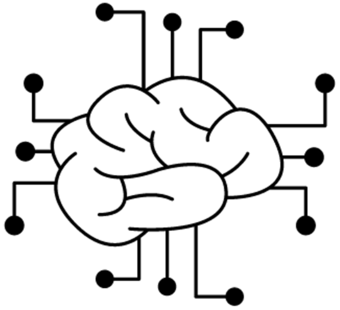
Bring Your Own Container - Steps

Code → **Containerize** → **Register with Amazon ECR** → **Create Estimator**

- byoc_image_uri = '{}.dkr.ecr.{}.{}/{}'.format(account_id, region, uri_suffix, ecr_repository + tag)
- estimator = Estimator (image_name=byoc_image_uri, ...   )

Summary

Summary



- Tune and evaluate a model
- Model tuning
- Tune a BERT-based text classifier
- Model evaluation
- Evaluate a BERT-based text classifier
- TODO: Script Mode?
- TODO: Bring Your Own Container?