Pretrain Vision and Large Language Models on AWS

Tutorial

Emily Webber

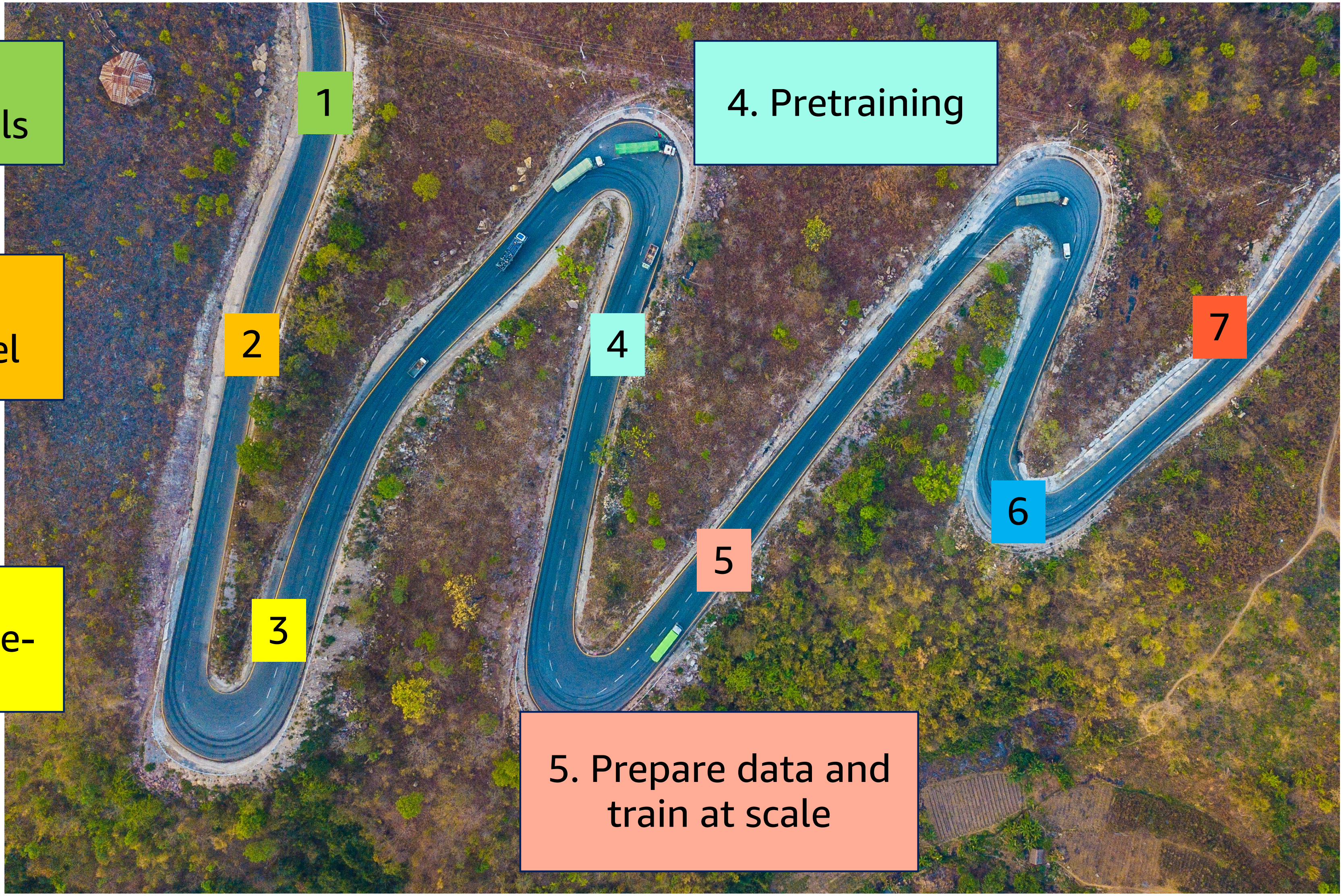
Principal ML Specialist SA at AWS

The winding road of R&D for foundation models

1. Primer on foundation models

2. How to pick a foundation model

3. Prompt engineering & fine-tuning



4. Pretraining

5. Prepare data and train at scale

6. Reinforcement learning with human feedback

7. Distributed hosting

So you want to build your own foundation model

- Why to pretrain
- When in your project lifecycle
- Which base model to pick
- What datasets to use
- How to do this easily and efficiently
- Hands-on demo: pretrain 30B parameter LLM on AWS with SageMaker
- Resources

Let's say I asked you to learn everything on the internet. How would you do it?

Structure

Storage

Time

5.74 B pages x 52 seconds
= ~85M hours

=> ~41,000 human years

A foundation model can do this in a few months.

You can do a lot with foundation models!

Text generation

Q&A

Text summarization

Text extraction

Paraphrase rephrase

Search

Code generation

Image generation

Image classification

Audio generation

Video generation

Many ML tasks can now be re-cast as generative, and most will benefit from foundation models.

Text: I am not into this house; it's way too expensive and too far from the train line!

Model → Sentiment: negative

Traditional classification

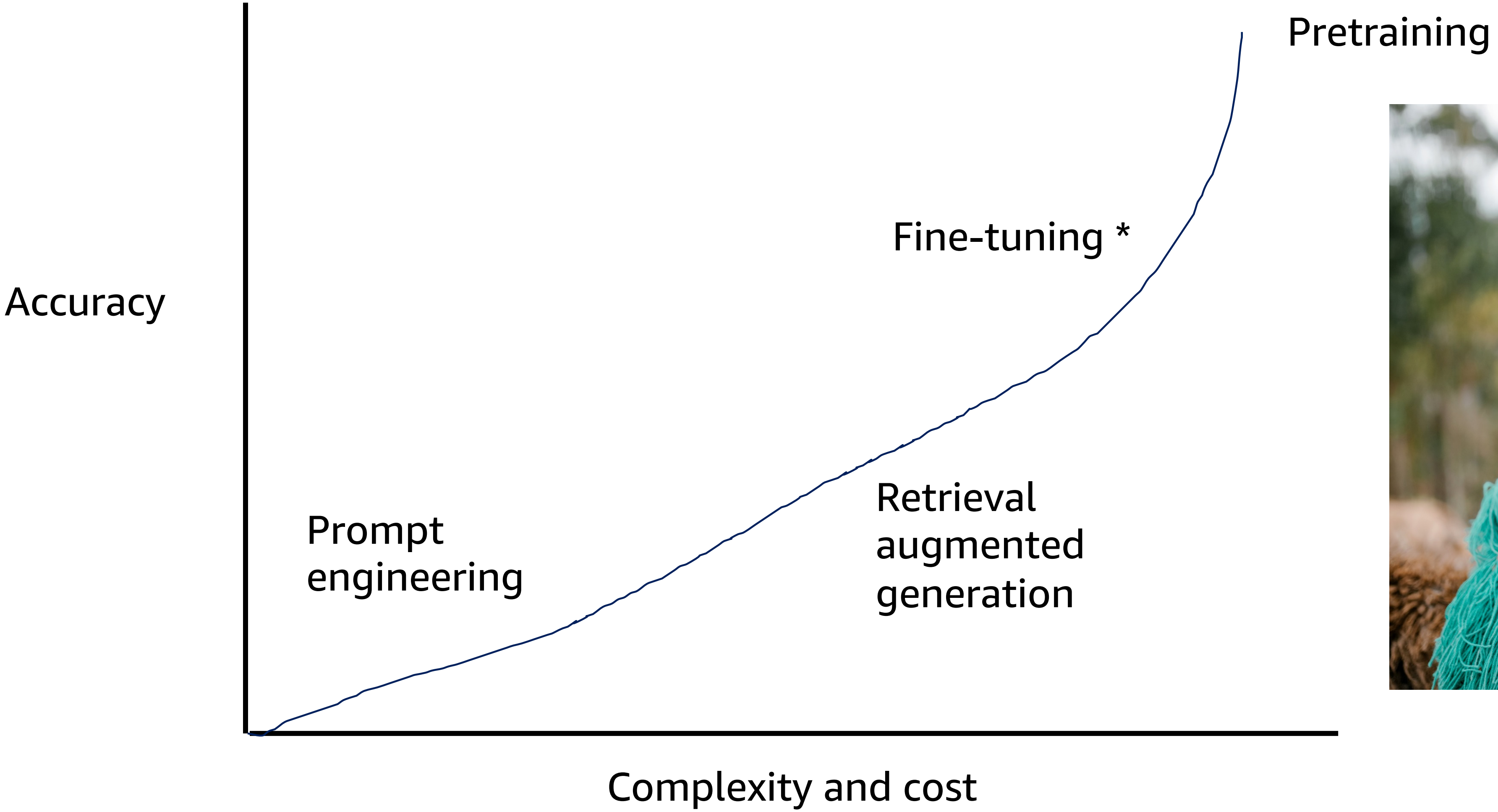
Text: I am not into this house; it's way too expensive and too far from the train line!

Classify this sentence into positive or negative sentiment:

Model → **Agent:** Negative sentiment

Using generation to classify text

There are many ways to customize a foundation model



The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin … the only thing that matters in the long run is the leveraging of computation.

Richard Sutton's *Bitter Lesson*, 2019

The Father of Reinforcement Learning

Pretraining might be the best long-term bet in AI

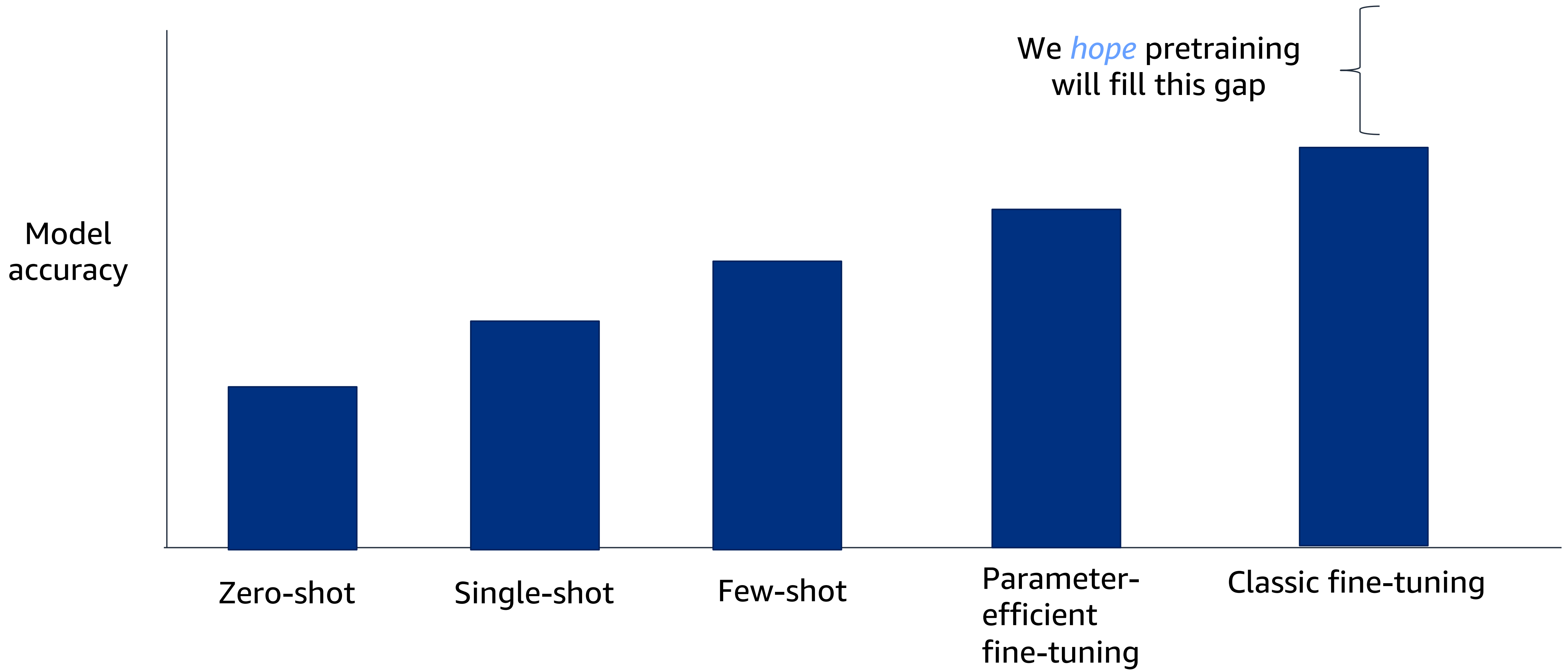
Better loss function

Unsupervised data are always larger

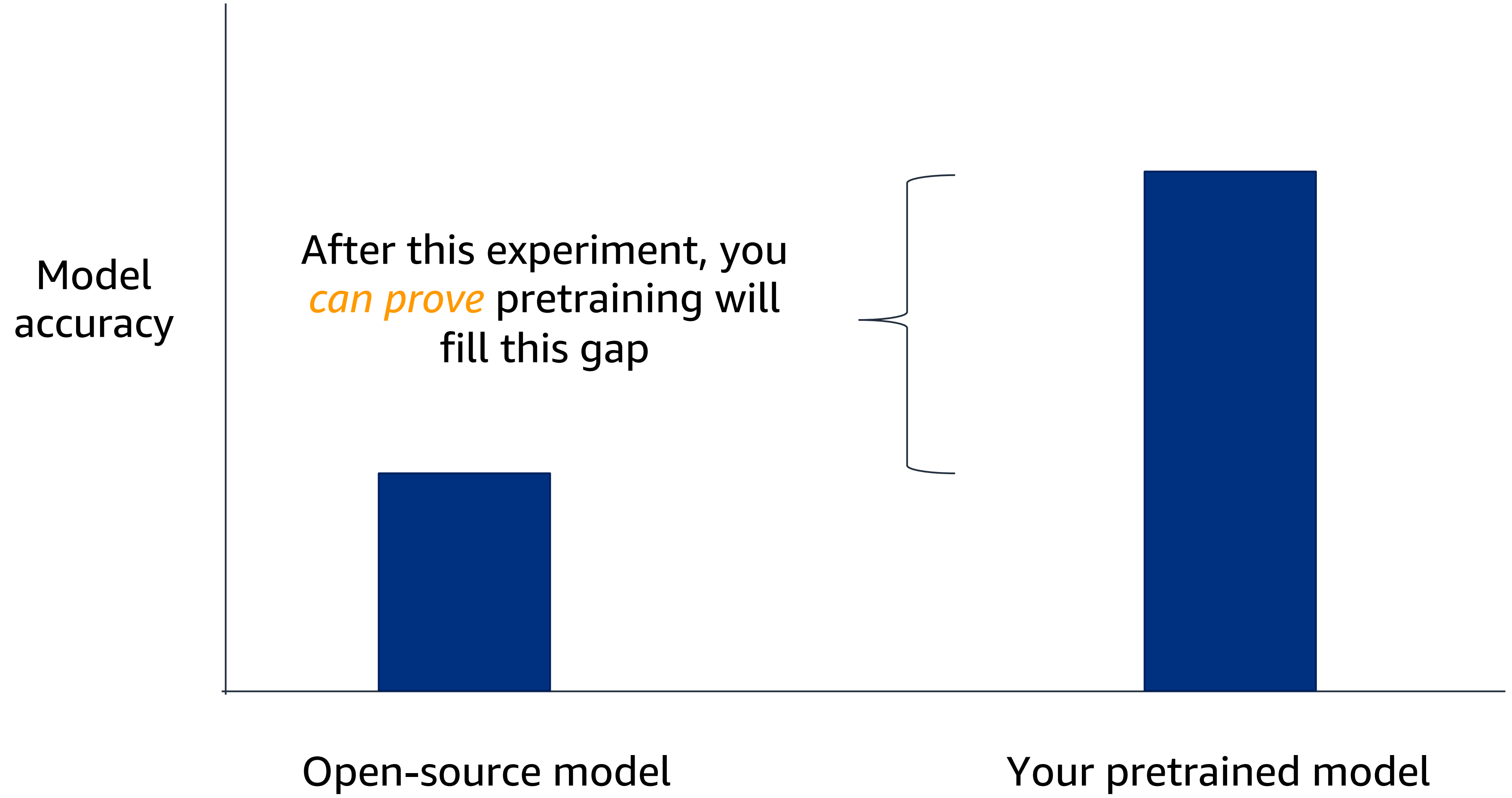
Deeper learning in the networks

Efficiency gains at scale

To consider a pretraining project, you want a chart like this



What to do before you launch all the accelerators



Pro tip – prove empirically at 1% of your data that pretraining ***does*** improve performance on your use case.

Do this ***before*** you run on massive accelerator clusters.

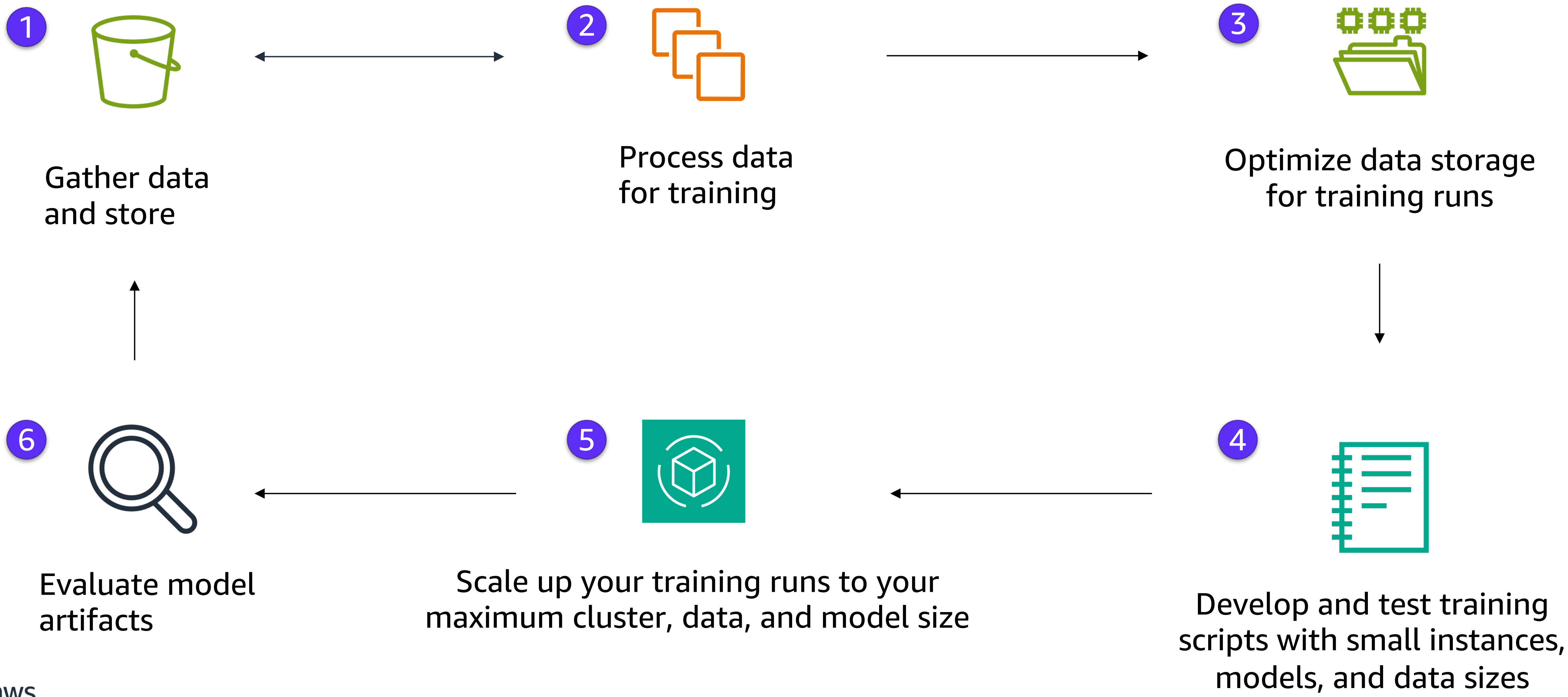
Fine-tune both models for this comparison

What does it take to pretrain a new foundation model?

Model name	Dataset size	Model size in parameters	Cluster size	Time to train
Stable Diffusion 2.1	5B images, 240 TB	< 1 billion	37 p4d instances	28 days
Falcon	1T tokens, 2.8 TB	40B	48 p4d instances	Two months
BloombergGPT	700B tokens, 1.9 TB	50B	64 p4d instances	53 days

All trained on SageMaker

How to pretrain foundation models on AWS

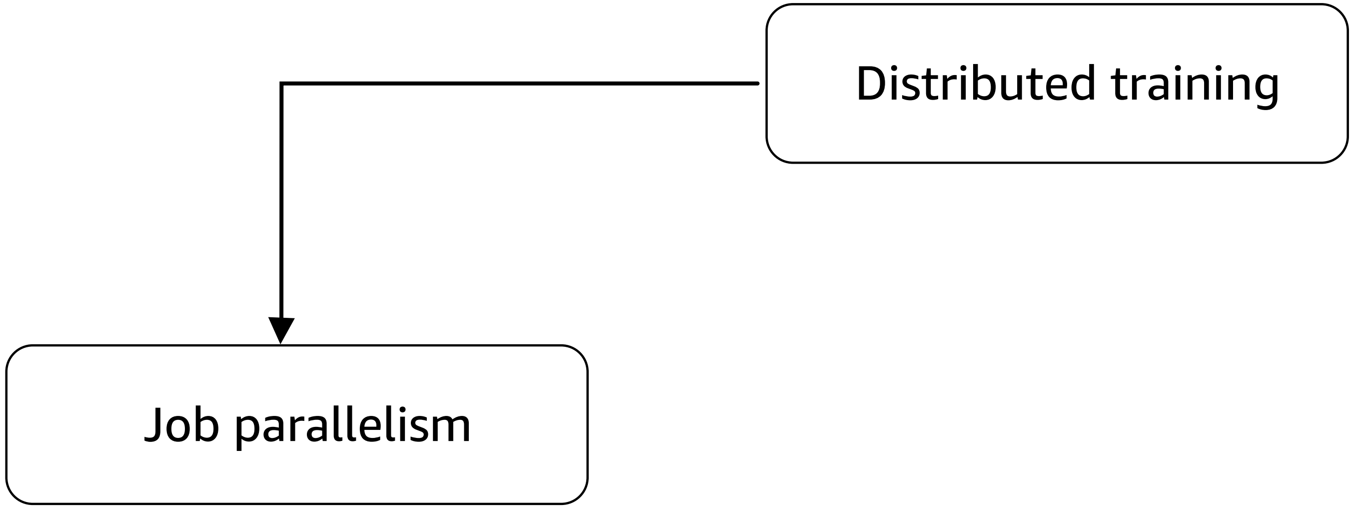


What does this look like in action?

Phase Number	Dataset sample	Model size	Cluster size in accelerators	Development and compute time
1	1%	Base	1	Hours
2	5%	Medium	8	Days
3	50%	Large	16	Weeks
4	100%	Jumbo	Max	Months

- Set a plan for your project to **scale in steps**
- This gives you solvable goals that start at the smallest possible sizes and work your way up to hitting the largest compute size
- Make sure you test your model checkpoint at each step to ensure it’s valid!

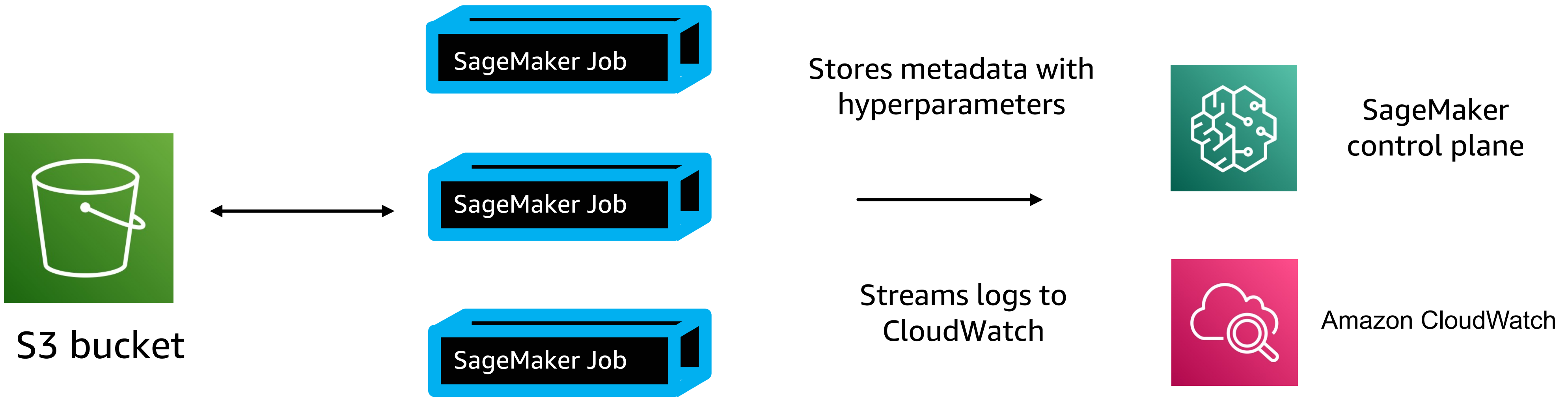
There are many kinds of distributed training



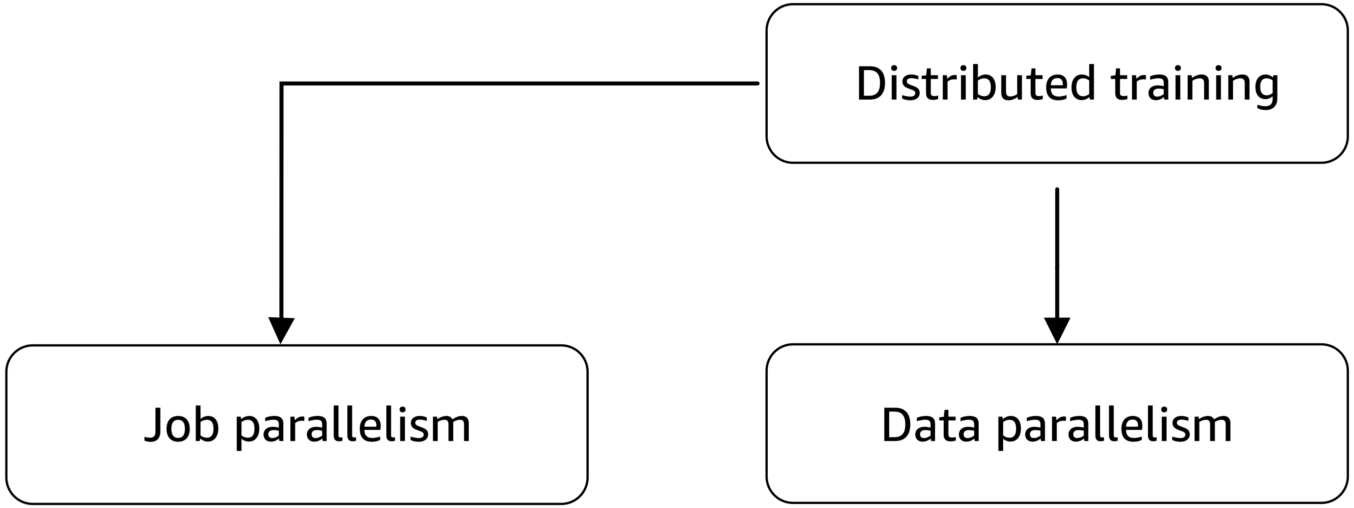
Run multiple jobs in parallel to process and train faster

1. Each job can train as many models as you need, or process as much data as you need.
2. You can use ***warm pools*** to reuse the instances

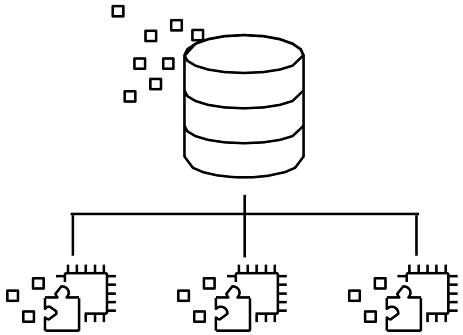
```
for model in list_of_models:
        s3_input = get_data(model)
        s3_output = get_location(model)
        estimator = get_estimator(model, s3_output)
        estimator.fit(s3_input, wait=False)
```



There are many kinds of distributed training

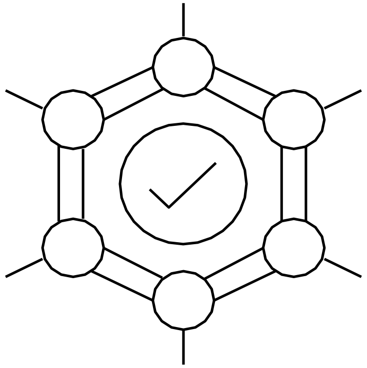


Distributed gradient descent has evolved over time



Parameter server

E.g., TensorFlow ParameterServerStrategy



MPI AllReduce

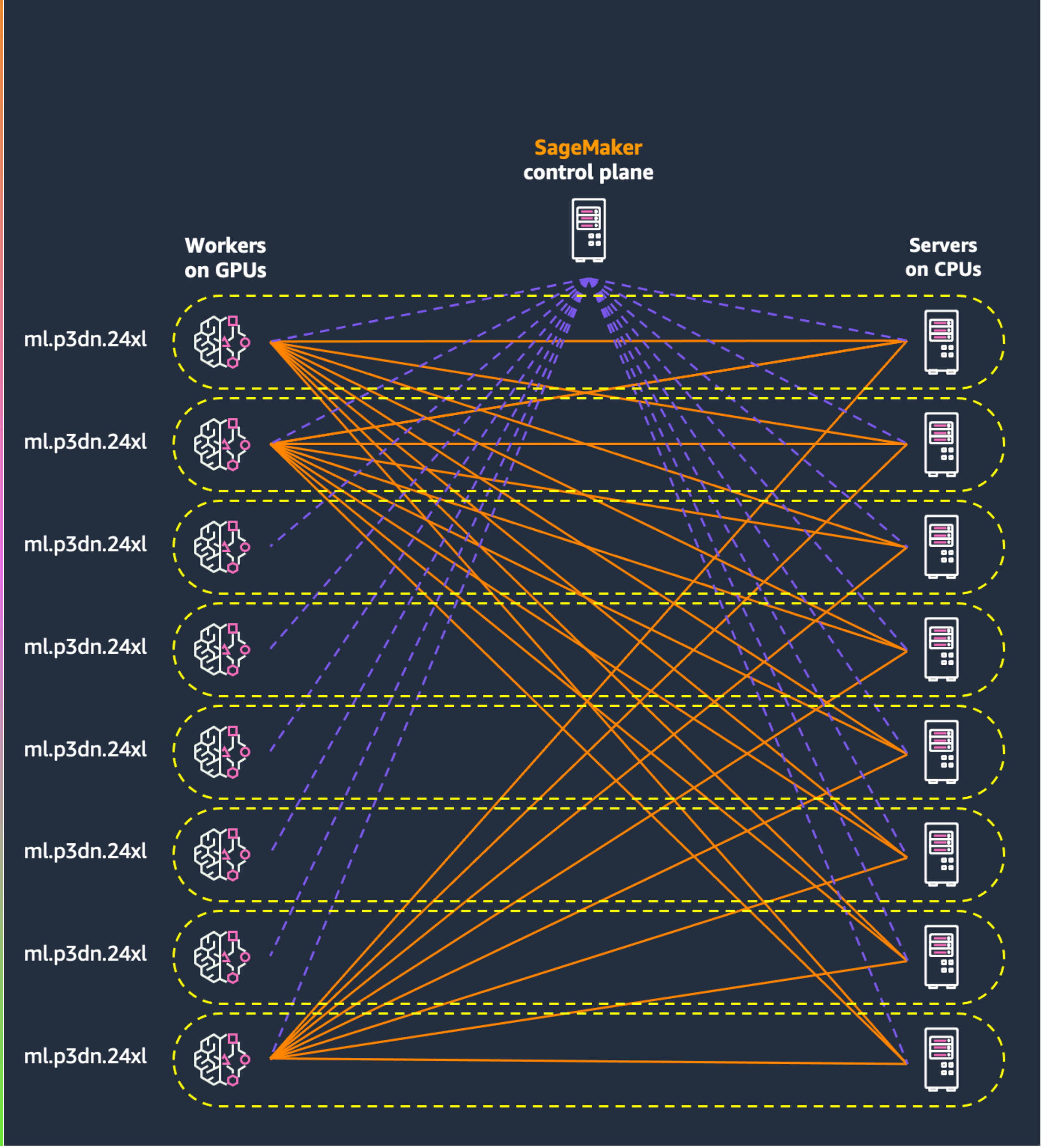
E.g., Horovod, PyTorch DistributedDataParallel

SageMaker Distributed Data Parallel

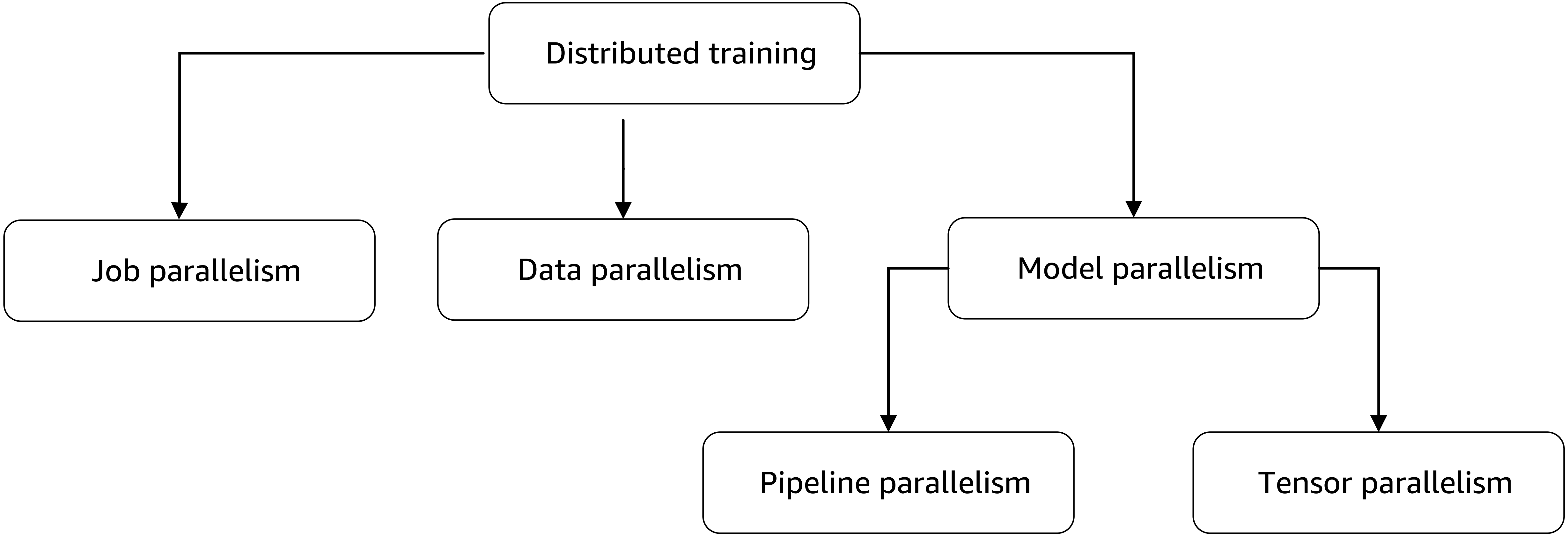
- Optimized backend for distributed training of deep learning models in TensorFlow, PyTorch
- Accelerates training for network-bound workloads
- Built and optimized for AWS network topology and hardware
- 20%–40% faster and cheaper than NCCL and MPI-based solutions. Best performance on AWS for large clusters.

Herring: Rethinking the Parameter Server at Scale for the Cloud

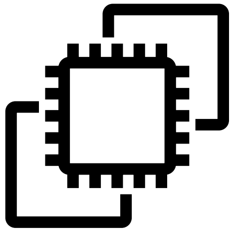
Indu Thangakrishnan, Derya Cavdar, Can Karakus,
Piyush Ghai, Yauheni Selivonchyk, Cory Pruce
Amazon Web Services
{thangakr, dcavdar, cakarak, ghaipiyu, yauheni, cpruce}@amazon.com



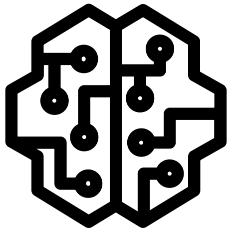
There are many kinds of distributed training



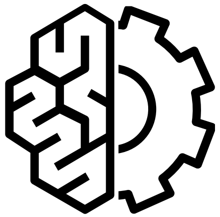
SageMaker model parallel



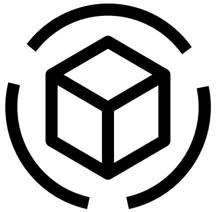
Automated model partitioning



Interleaved pipelined training



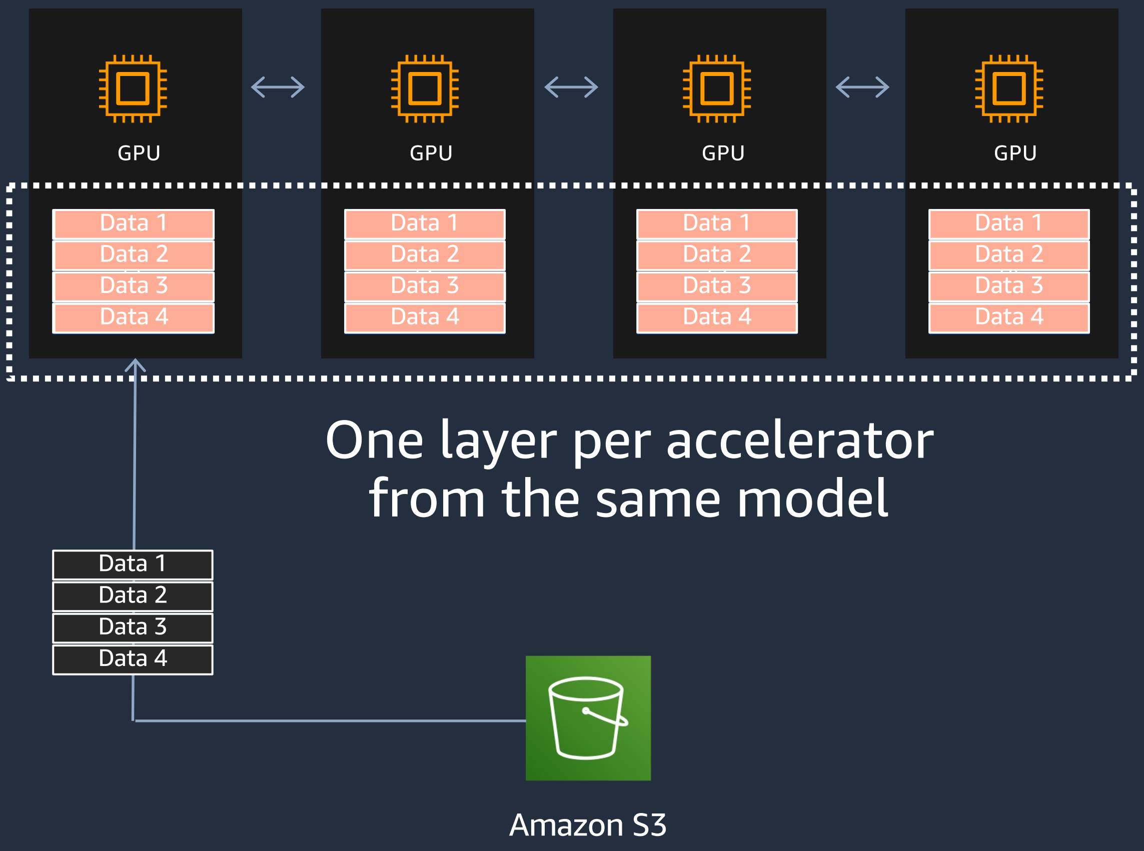
Managed SageMaker training



Clean framework integration

SageMaker Model Parallel splits your model over multiple accelerators

- Split minibatches into N “microbatches”
- Feed microbatches sequentially, but process them to keep GPU utilization more even
- Minimize “idle” time on GPUs

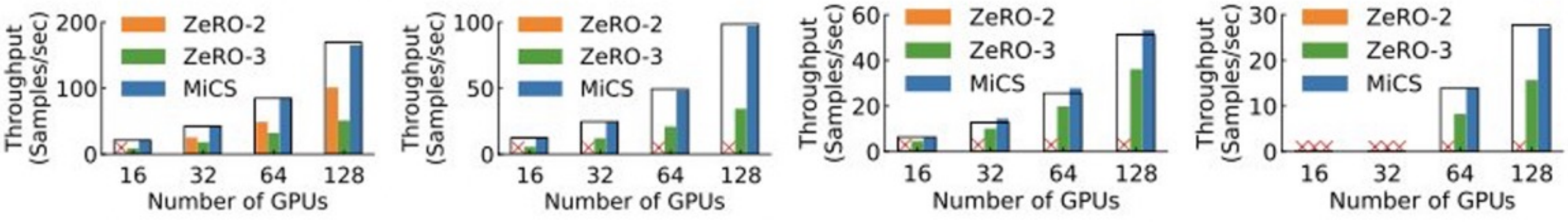


AMAZON SAGEMAKER MODEL PARALLELISM: A GENERAL AND FLEXIBLE FRAMEWORK FOR LARGE MODEL TRAINING

Can Karakus [1] Rahul Huilgol [1] Fei Wu [1] Anirudh Subramanian [1] Cade Daniel [1] Derya Cavdar [1] Teng Xu [1] Haohan Chen [1] Arash Rahnama [1] Luis Quintela [1]

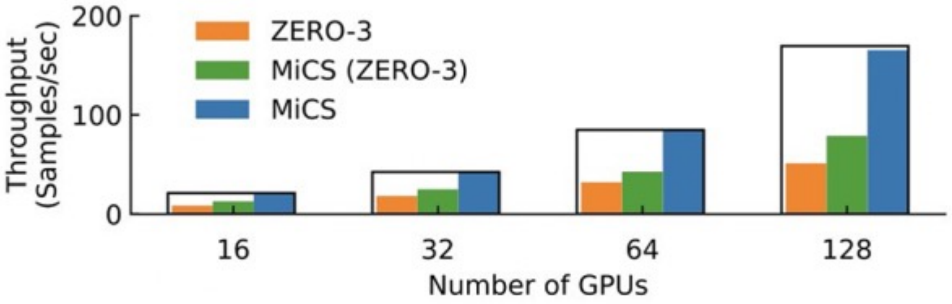
Approach linear-scaling with *Sharded Data Parallelism*

MiCS achieves 169 TFLOPS per GPU with 175B parameter model on AWS p4de.24xlarge instances



(a) BERT 10B. (b) BERT 15B. (c) BERT 20B. (d) BERT 50B.

- MiCS hits 99.4% of linear-scaling efficiency from 128 to 512 GPUs
- DeepSpeed hits only 72% , saturates at 62 TFLOPS per GPU



MiCS: Near-linear Scaling for Training Gigantic Model on Public Cloud

Zhen Zhang*
Johns Hopkins University
zzhen1@jhu.edu

Shuai Zheng
Amazon Web Services
shzheng@amazon.com

Yida Wang
Amazon Web Services
wangyida@amazon.com

Justin Chiu
Amazon
justchiu@amazon.com

George Karypis
Amazon Web Services
gkarypis@amazon.com

Trishul Chilimbi
Amazon
trishulc@amazon.com

Mu Li
Amazon Web Services
mli@amazon.com

Xin Jin
Peking University
xinjinpku@pku.edu.cn

Available within SageMaker Model Parallel
2.8x faster than DeepSpeed

But what about reinforcement learning with human feedback?

Not all human feedback is the same

Objective human feedback

1+1 = 2

Literal translations and classifications

External outcomes

Empirical observations

Subjective human feedback

Nuanced preferences

Gut reactions

Responses to content

Interpreting artwork

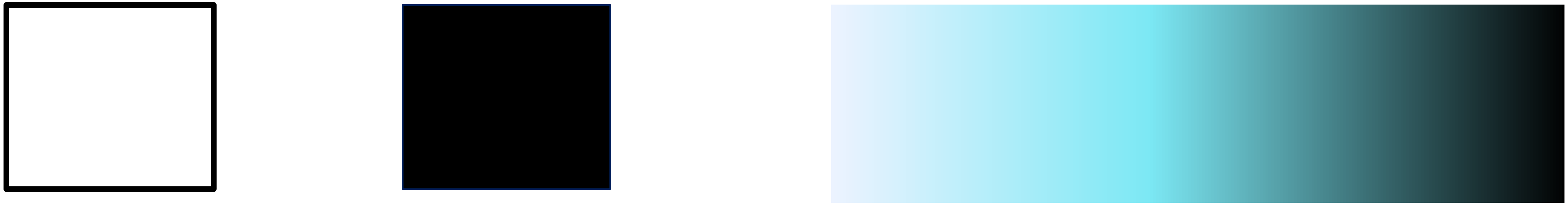
Human feedback varies by use case and personality

Objective human feedback

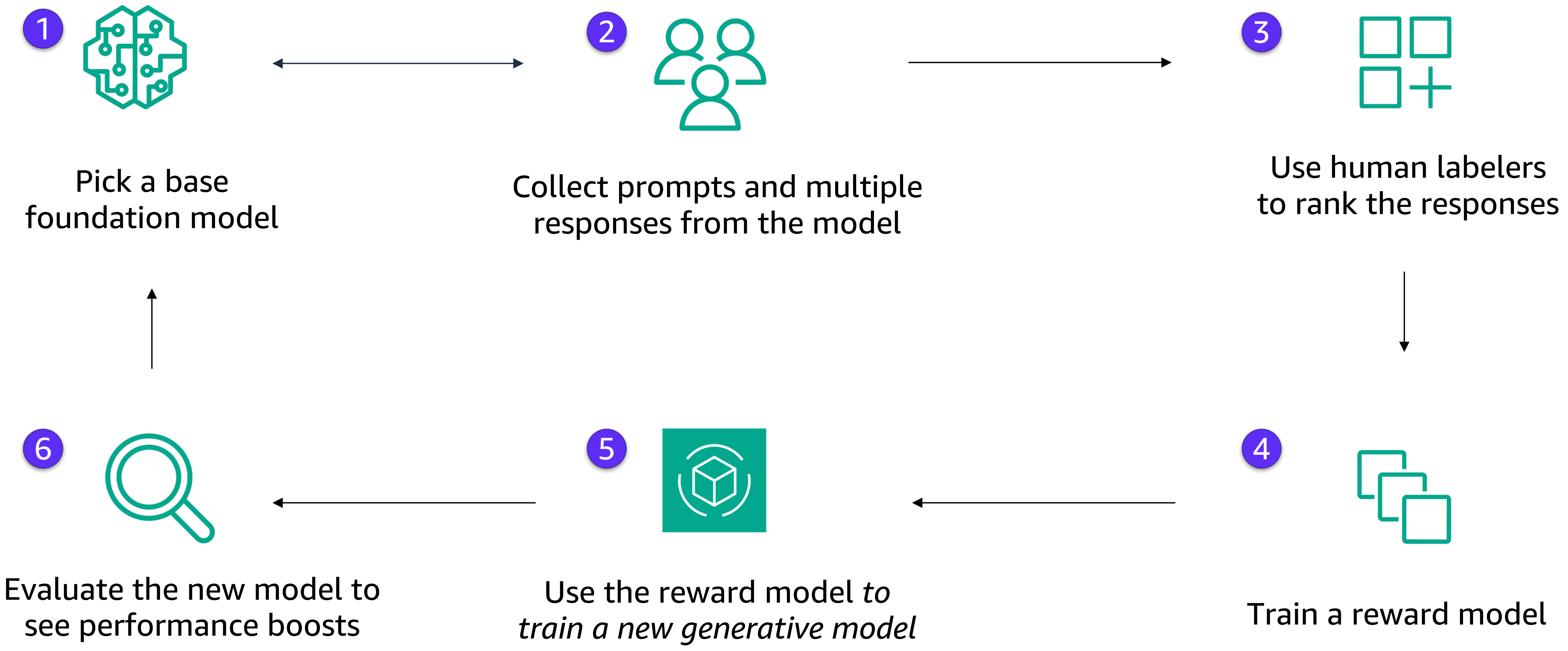
Great for traditional ML tasks

Subjective human feedback

Great for generative ML tasks



Reward modelling aggregates human feedback at scale



Reinforcement learning with human feedback

- Start with a dataset of prompts and responses, with multiple responses for each prompt
- Send these to humans for ranking
- Train a new *reward model* on the human rankings, using reinforcement learning
- Use the reward model to train a new generative model
- The final model should be 2-3x better than the original

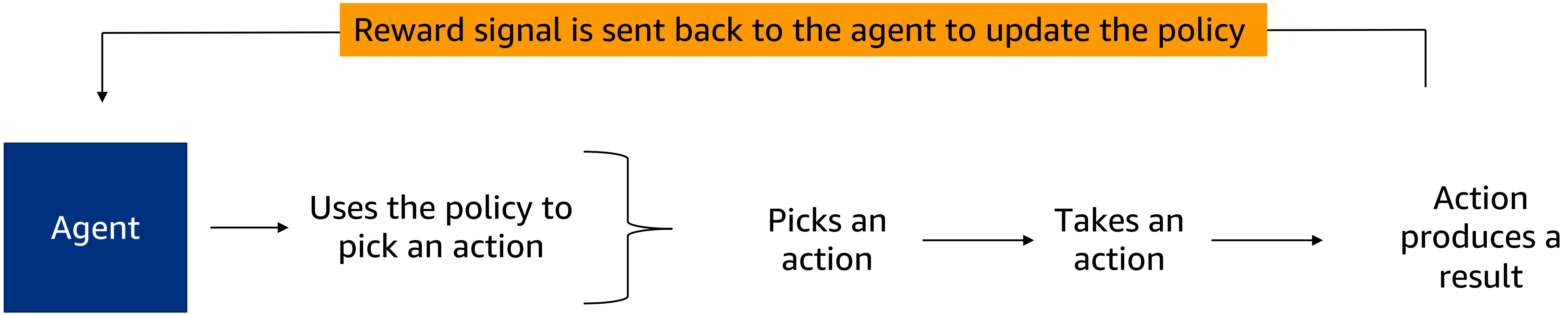
Pro tip:

Reinforcement learning with human feedback is one of the most common ways to perform *reward modelling*

Quick recap of reinforcement learning

Vocabulary

- *Reinforcement learning*: a type of machine learning commonly used to train robotic agents
- *Agent:* an autonomous entity we want to train
- *Policy*: how the agent learns, commonly a neural network
- *Action space*: all possible actions the agent can take
- *Reward function*: a signal provided to the agent to drive its learning



Applying reinforcement learning to update LLMs

- **Policy**: the LLM you want to fine-tune, orchestrated by proxy policy optimization (PPO)
- **Action space**: all possible tokens in the vocabulary
- **Reward model**: a model you train on the human-ranked responses from the LLM
- **Divergence**: a distance function you use to keep the original LLM and the one you are training closer
- **Reward function**: uses a pretrained reward model, combined with the divergence term, to update the agent and its neural network

RLHF mathematically speaking

- x = prompts from the training dataset
- y^* = text generated by the LLM (the PPO) you are training, using the prompts
- y^0 = text generated from the original LLM you used first, also using the prompts

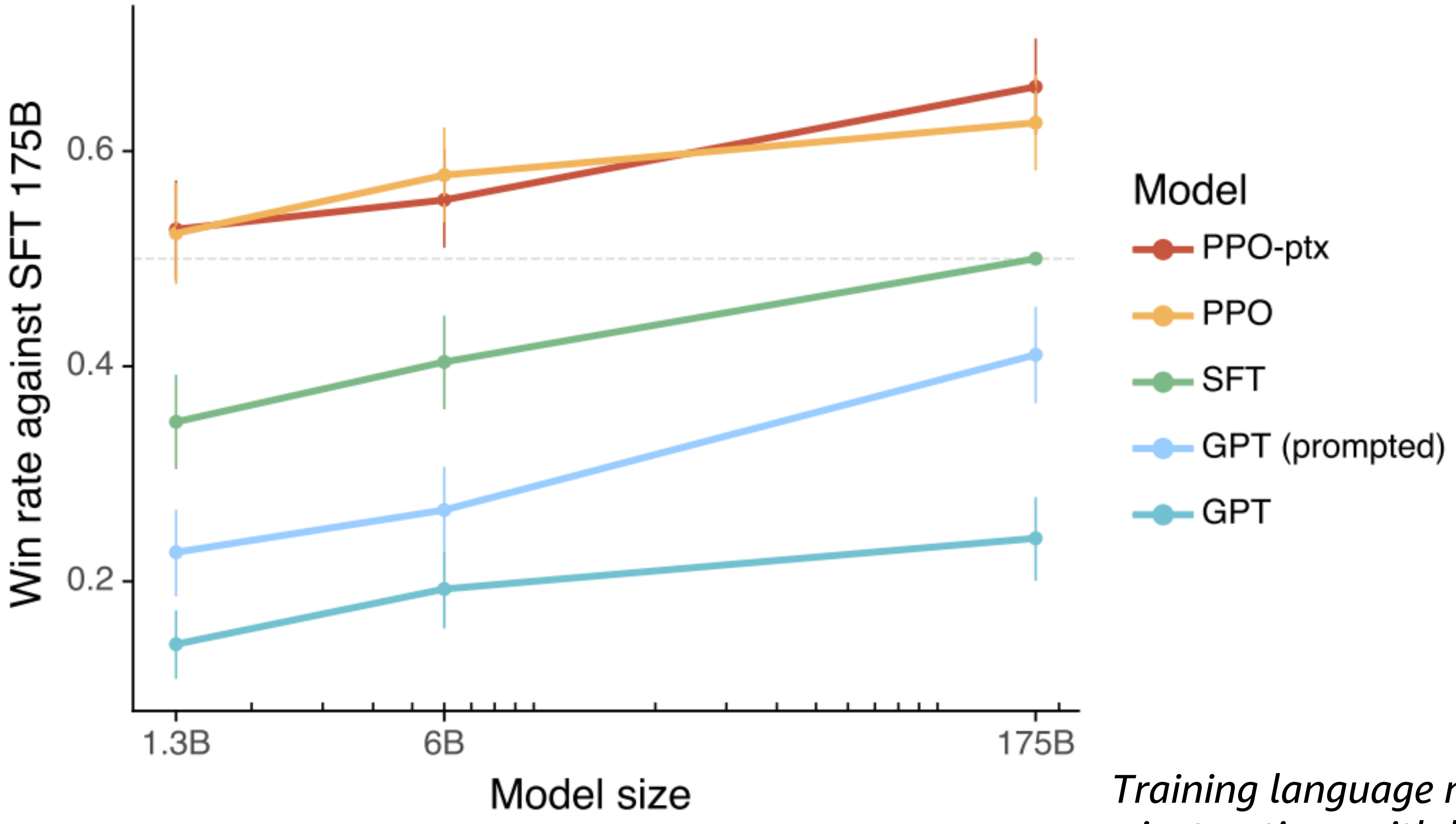
Tells you what humans prefer → $r_\theta = reward\_model(x + y^*)$

Prevents out-of-character RL hacks → $r_{KD} = \text{KLDivergence}(y^*, y^0)$

Serves as the signal to update your neural network → $r_{PPO} = r_\theta - \epsilon * r_{KD} + ?$ ← May be useful to add pretraining gradients here

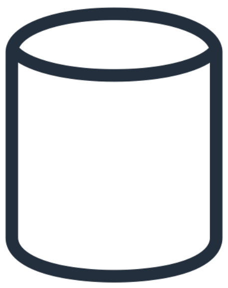
A tunable weighting term

RLHF shows 2-3x boost over base GPT-3

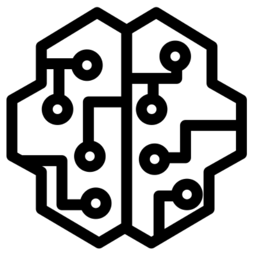


Training language models to follow instructions with human feedback
Ouyang et al, 2022

What you need to train a reward model



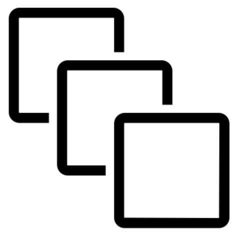
1:many dataset with
prompts and responses



A GPT-based model that
returns a number

A regressive large language model

But not that large, ~6B is good enough



Distributed
training systems

Datasets for reward modelling

Prompt → "What's the weather like in Washington, DC?"

Responses →	The local weather in Washington DC is currently sunny and humid, at a temperature of 82 degrees Fahrenheit.	It's freaking hot!!	Relative to Phoenix, Arizona, Washington DC is a cool 82 degrees.
Preference rankings →	2	1	3

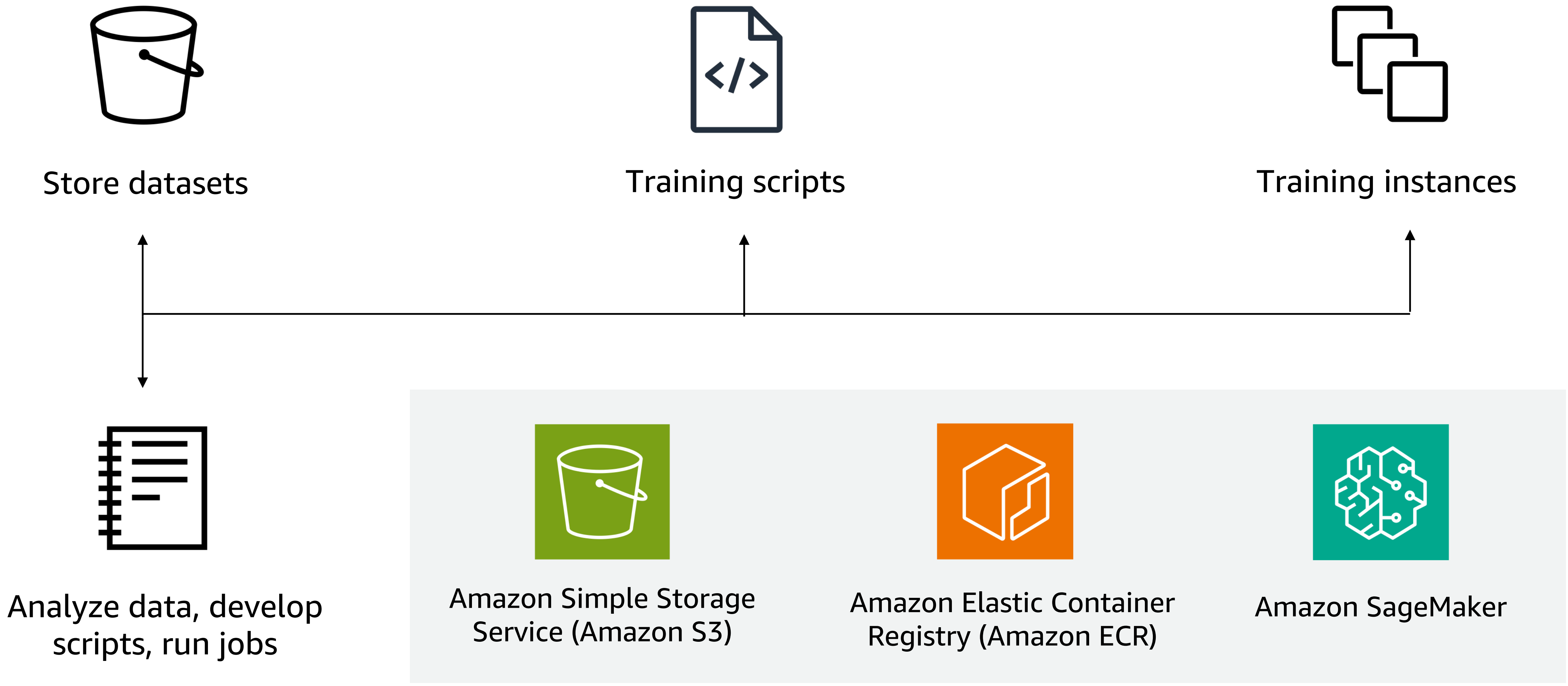
You want some preference number to rank all of the possible responses to each prompt.

You can use humans, AI's, or any kind of digital signal to create these rankings.

The rankings *become the label to train a supervised reward model.*

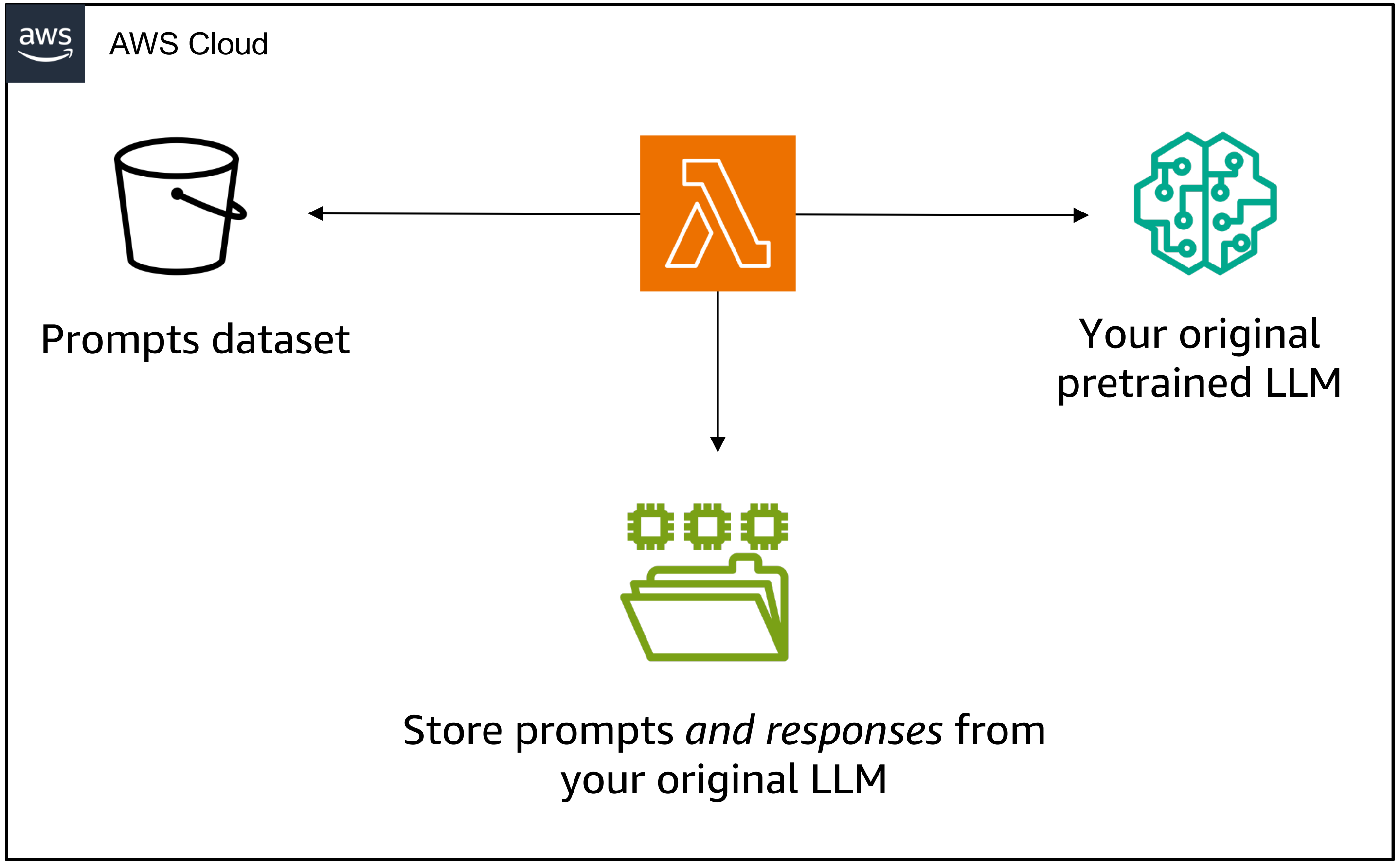
aws

How to build and train a reward model on AWS



Use your reward model to train a new LLM

Ahead of time, precompute the original model responses



- Run a CPU-based and/or serverless job *ahead of time*
- Store both the prompts and the responses from your original LLM
- Prepare the training dataset on a high-performance distributed file system to optimize the training runs
- May already be in your ranking dataset!

Use your reward model to train a new LLM

Prompts and responses from your original LLM

Amazon SageMaker

Heterogenous clusters

Reward model

Pro tips:

1. Use *smaller* accelerators for the reward model
2. Use *larger* accelerators for the LLM you are fine-tuning
3. Keep these in the same cluster to maximize run-time

The LLM you are *fine-tuning*

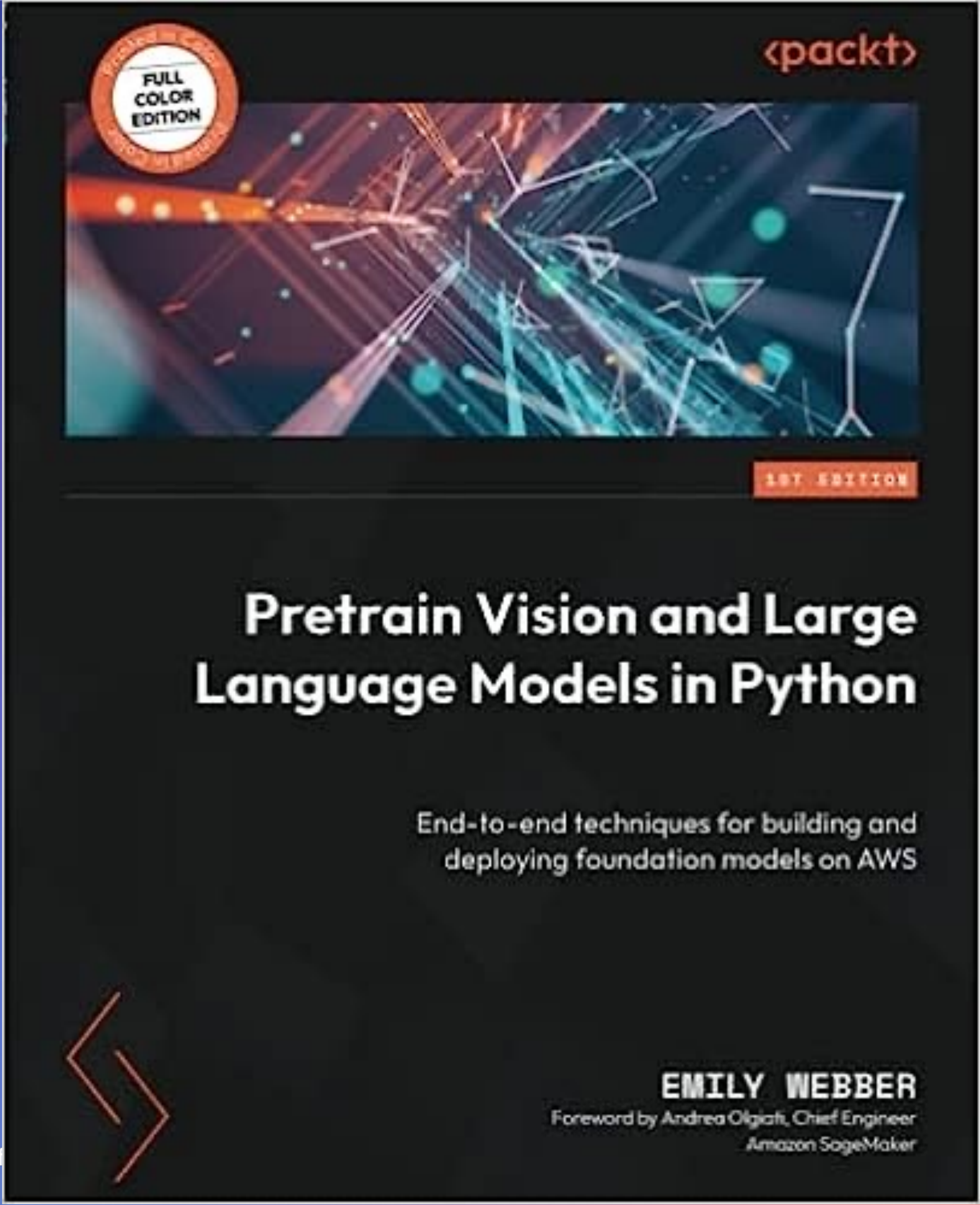
1. Get response from latest epoch in your tunable LLM
2. Send to the reward model to get the human preferences
3. Run the KL penalty function to compare the model responses
4. Use the reward model and the KL penalty to update your LLM

https://bit.ly/sm-nb-4

Hands-on demo

Pretrain Vision and Large Language Models

https://bit.ly/dist-train-book



Thank you!

Emily Webber

Link to slides