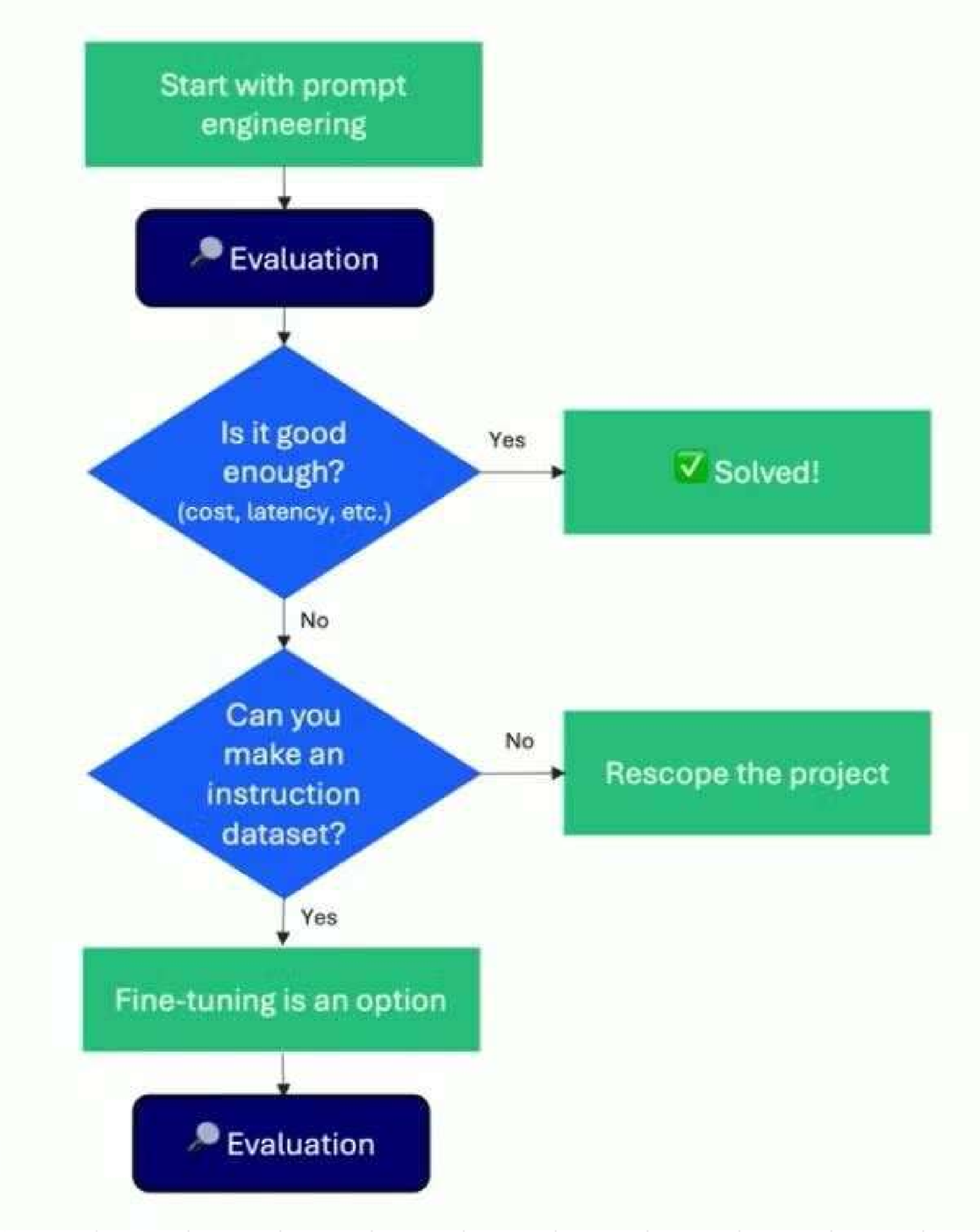
Introduction to Fine Tuning LLMs

When to use fine-tuning?

100x



When to use fine-tuning?

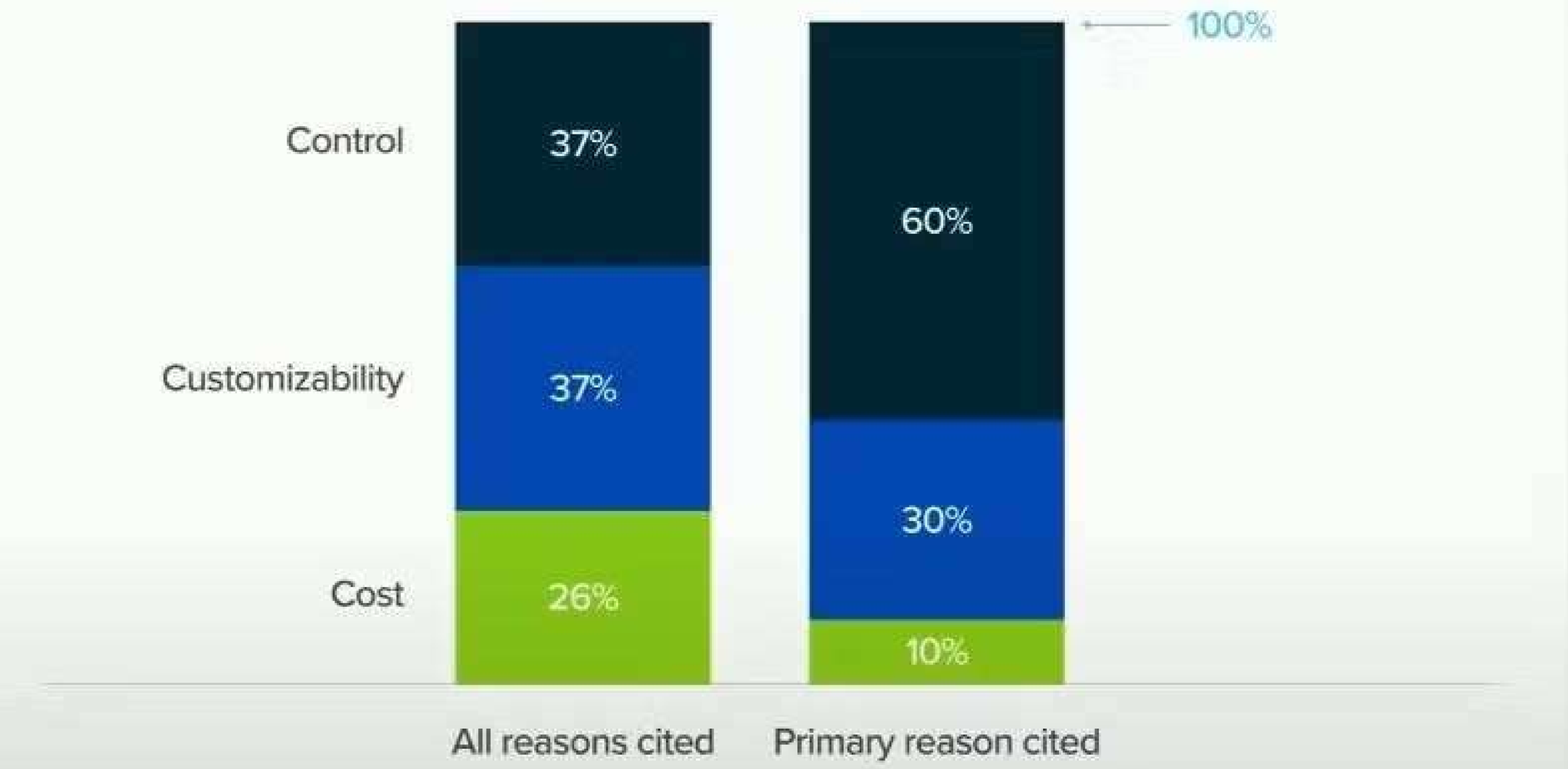
- Start with prompt engineering if possible
- Develop a robust evaluation framework with various metrics
- Consider fine-tuning if prompt engineering isn't sufficient and you can create an instruction dataset
- Non-technical reasons: Enterprises value control and customizability of models

Fine-tuning for Enterprises

100x

Why do enterprises care about open-source?

a16z Growth

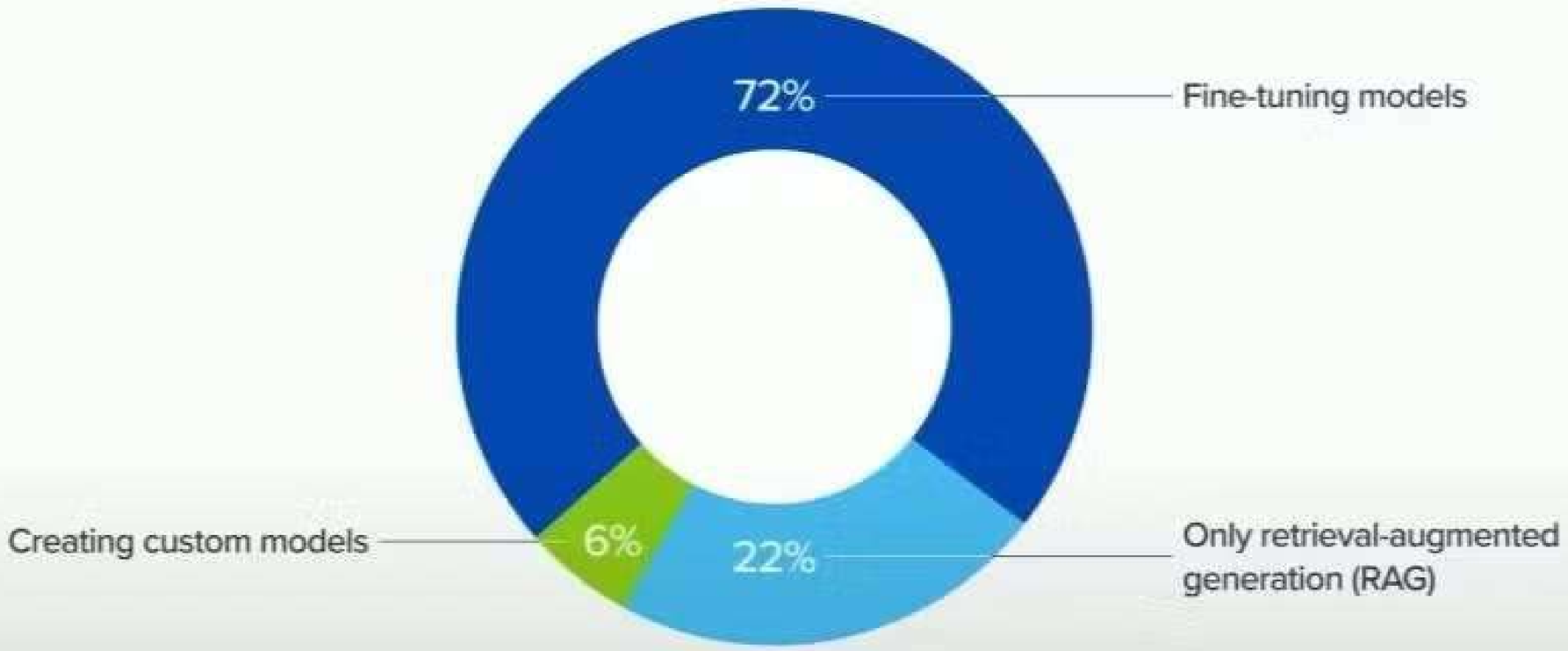


Source: interviews conducted by a16z partners

100x

How are enterprises customizing their models?

a16z Growth

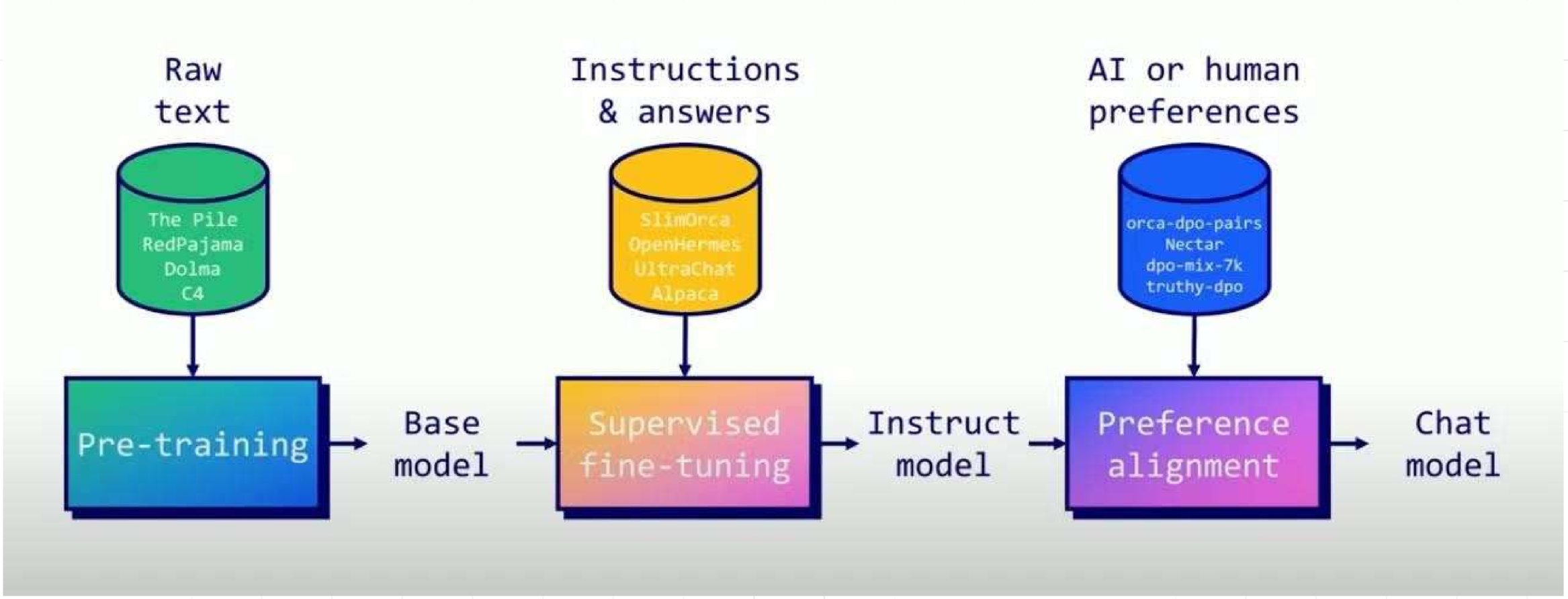


Source: a16z survey of 70 enterprise AI decision makers

100x

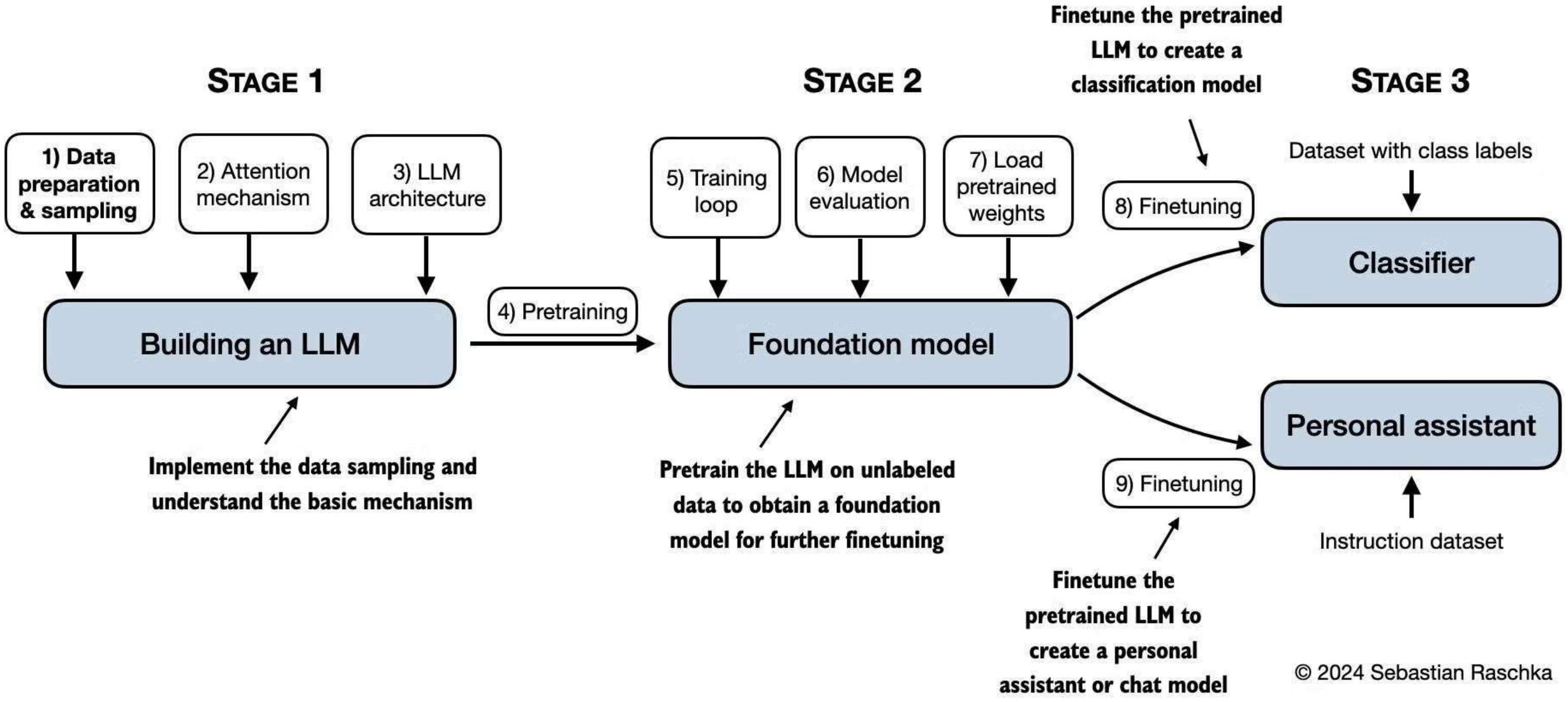
LLM Training Lifecycle

100x


Raw text
The Pile
RedPajama
Dolma
C4
Pre-training
Base model
Instructions & answers
SlimOrca
OpenHermes
UltraChat
Alpaca
Supervised fine-tuning
Instruct model
AI or human preferences
orca-dpo-pairs
Nectar
dpo-mix-7k
truthy-dpo
Preference alignment
Chat model

- Three stages: Pre-training, Supervised Fine-tuning, and Preference Alignment
- Pre-training: Uses raw text for next token prediction, resulting in a base model
- Supervised Fine-tuning (SFT): Uses question-answer pairs to teach the model to follow instructions
- Preference Alignment: Aligns the model with human preferences, resulting in a chat model

A little bit about Pre-training


STAGE 1
STAGE 2
STAGE 3
1) Data preparation & sampling
2) Attention mechanism
3) LLM architecture
Building an LLM
Implement the data sampling and understand the basic mechanism
4) Pretraining
5) Training loop
6) Model evaluation
7) Load pretrained weights
Foundation model
Pretrain the LLM on unlabeled data to obtain a foundation model for further finetuning
Finetune the pretrained LLM to create a classification model
8) Finetuning
Classifier
Dataset with class labels
9) Finetuning
Personal assistant
Instruction dataset
Finetune the pretrained LLM to create a personal assistant or chat model
© 2024 Sebastian Raschka

Data preparation

- Gather a massive dataset of text data (often terabytes)
- Choose or create a model architecture
- Train a tokenizer for the data
- Pre-process the dataset using the tokenizer

Pre-training process

- Model learns to predict words or fill in missing text
- Uses self-supervised learning techniques
- Two main approaches:
  a. **Masked Language Modeling**: predict intentionally hidden tokens
  b. **Causal Language Modeling**: predict the next word given preceding context
- Trains on vast amounts of data to develop language understanding

Outcome

- Model develops general language knowledge
- Becomes a proficient language encoder
- Lacks specific task or domain knowledge (addressed in fine-tuning)

1000x H100 (80GB) Tensor Core GPU

10000 hours

Dataset: Internet/10

US$ 10M

100x

Supervised Fine-tuning

Supervised Fine-tuning

Fine-tuning is a process where a pre-trained model, which has already learned some patterns and features on a large dataset, is further trained (or "fine tuned") on a smaller, domain-specific dataset. In the context of "LLM Fine-Tuning," LLM refers to a "Large Language Model" like the GPT series from OpenAI. This method is important because training a large language model from scratch is incredibly expensive, both in terms of computational resources and time. By leveraging the knowledge already captured in the pre-trained model, one can achieve high performance on specific tasks with significantly less data and compute.

Transfer learning

The primary advantage of transfer learning is that the model starts with a significant amount of pre-learned knowledge. Instead of learning from scratch, the model uses its existing knowledge base to adapt more quickly and effectively to the new task. This leads to improved performance, reduced training time, and often requires less labeled data for the new task.

Purpose of fine-tuning

- Specializes a pre-trained model for specific tasks
- Optimizes performance on narrower, task-specific datasets
- Builds upon general language knowledge from pre-training

Fine-tuning process

Dataset preparation

- Gather a task-specific dataset
- Consists of labeled examples relevant to the desired task
- Significantly smaller than pre-training datasets
- Example: instruction-response pairs for instruct-tuning

Model initialization

- Start with a pre-trained model
- Initialize using previously learned parameters

Training on new data

- Expose the model to the task-specific dataset
- Optimize parameters to minimize a task-specific loss function
- Loss function measures how "off" the model's output is from desired results

Parameter adjustment

- Use gradient-based optimization algorithms (e.g., SGD, Adam)
- Compute gradients by backpropagating the loss through model layers
- Update parameters to improve performance on the specific task

RLHF & Preference Alignment

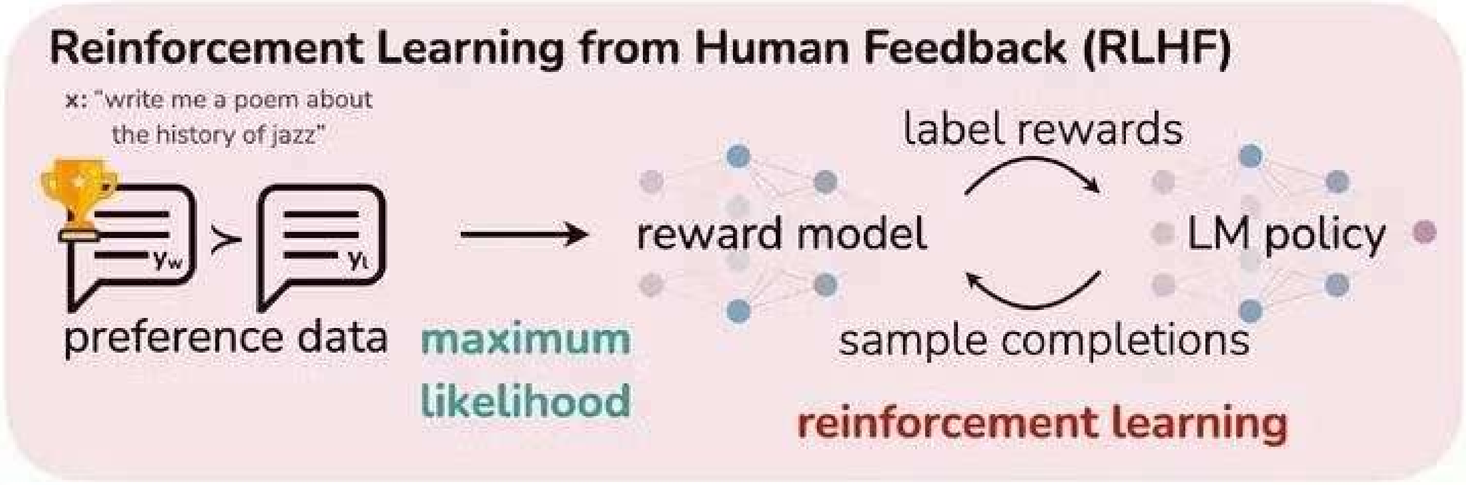
RLHF & Preference Alignment

Instruction
Tell me a joke about octopuses.

Chosen answer
Why don't octopuses play cards in casinos?
Because they can't count past eight.

Rejected answer
How many tickles does it take to make an octopus laugh? Ten tickles.

RLHF

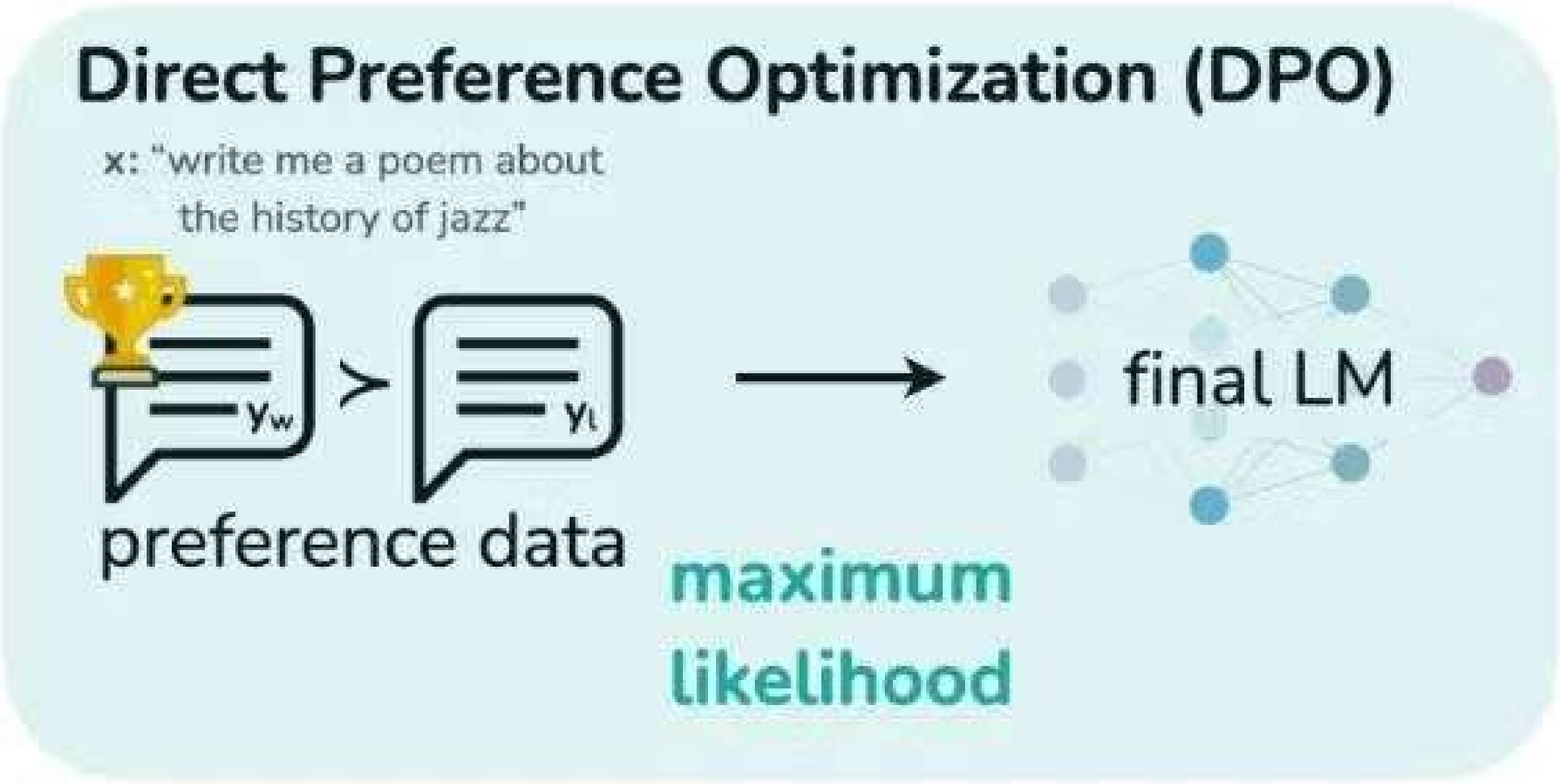


The core idea behind it is pretty simple: we have a very smart language model that has lots of knowledge but doesn't know how to properly communicate them to a human. How could we overcome that?

1. Make users (humans) interact with GPT3, show them alternative answers and collect their preferred one.
2. Gather the feedback in a dataset with the following shape:a. User question | preferred answer | rejected answer.
3. Train a reinforcement learning (RL with PPO) model with the dataset.
4. With the RL model trained, make it evaluate GPT3 answers and give it rewards fine-tuning the model.

Direct Preference Optimization (DPO)

Direct Preference Optimization (DPO)



Training a reward model from human feedback was *complex and often unstable*. Hence, some thought that just skipping the reward model and directly fine-tune an LLM with the preference dataset could work. And, surprise, it worked.

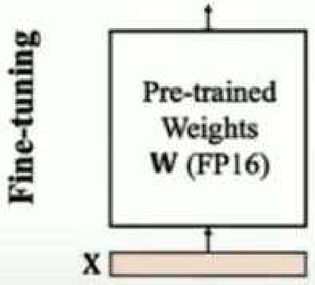
The core idea of the Stanford team was that, internally, LLMs itself worked as a reward model and just giving it preference data would do the trick.

Training Techniques

Training Techniques

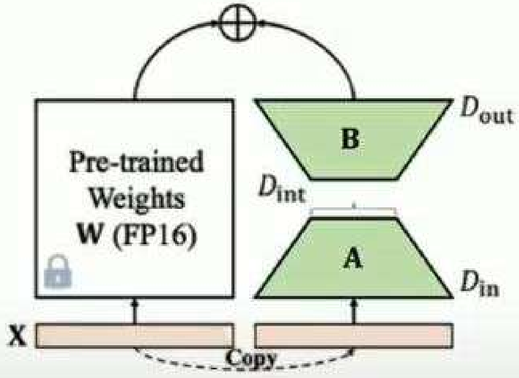
SFT Techniques

Full Fine-Tuning
16-bit precision



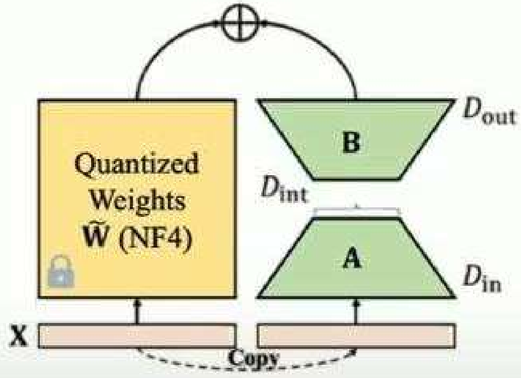
✅ Best performance
❌ Very high VRAM usage

LoRA
16-bit precision



✅ Quick training
❌ Still costly

QLoRA
4-bit precision



✅ Low VRAM usage
❌ Degrades performance

100x

Training Techniques

Full Fine-tuning

- Updates all model parameters
- Requires significant computational resources
- Provides maximum adaptation to new tasks
- Can potentially lead to catastrophic forgetting

LoRA (Low-Rank Adaptation)

- Updates a small number of task-specific parameters
- Much more efficient than full fine-tuning
- Preserves most of the original model's knowledge
- Can be combined with the original model weights

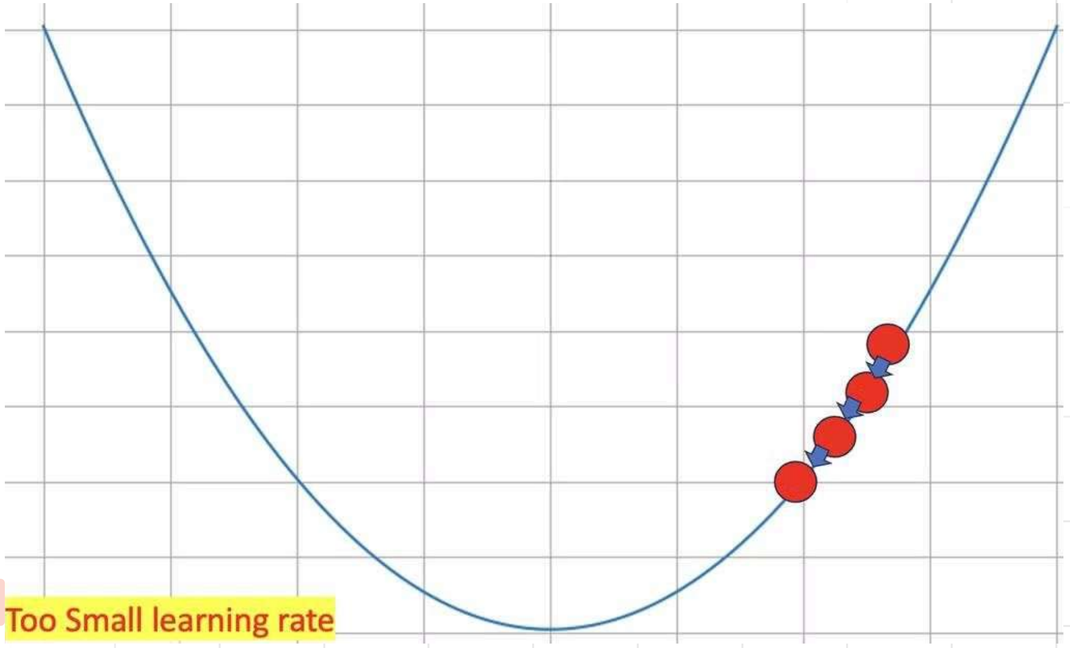
QLoRA

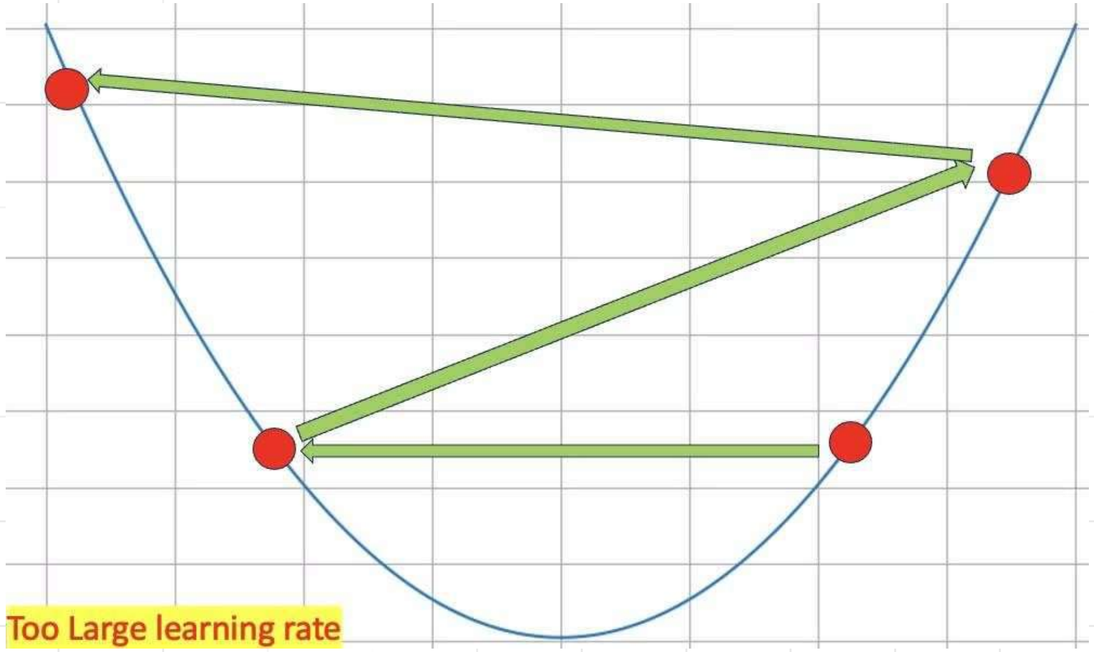
- Combines LoRA with quantization techniques (4-bit precision)
- Even more memory-efficient than standard LoRA
- Allows fine-tuning of larger models on consumer hardware
- May have a small trade-off in performance compared to full-precision LoRA

Hyperparameters

Learning Rate

The learning rate controls the size of updates to the model weights during training. A high learning rate allows for faster learning but can cause instability or convergence to a suboptimal solution, while a low learning rate can result in slow convergence or getting stuck in local minima.





Batch Size

Batch size determines the number of examples the model processes before updating weights. Larger batch sizes lead to more stable gradient estimates but require more memory, while smaller batch sizes allow for faster iteration but can lead to noisy gradient estimates.

Epochs

This hyperparameter controls how many times the model passes over the entire dataset. More epochs allow the model to learn the patterns in the data more thoroughly but increase the risk of overfitting. Too few epochs may lead to underfitting, where the model doesn't fully learn the task.

LoRA Rank

This determines the number of rank decomposition matrices. Rank decomposition is applied to weight matrices in order to reduce memory consumption and computational requirements. The original LoRA paper recommends a rank of 8 as the minimum amount. Keep in mind that higher ranks lead to better results and higher compute requirements. The more complex your dataset, the higher your rank will need to be.