Efficiency I: Parameter Efficient Finetuning

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2026

THE OHIO STATE UNIVERSITY

Slide Credits: Yejin Choi, Yoav Artzi

Logistics

- Homework 3 deadline March 11.

- Project Proposal is graded. Please email me with clarifications if I asked for them in the comments.

- Project mid-term report: due on April 1.
  - Will make a post on Teams/canvas with everything that's required.

Notes about Default Project

- Many of you proposed explorations like:
  - Data filtering
  - Trade off between safety vs helpfulness curves
  - Uncertainly calibration

- Please make sure the explorations are enough work for your team size.
  - Data filtering / trade off is very little amount of work, not enough even for one person.
- My suggestions ((it seems intimidating but it is easy to do on Tinker):
  - RLHF
  - RLVR
  - Larger Models (8b or beyond)

Last Class Recap: Alignment / Post-training

- Supervised Finetuning (Teacher forcing the right responses)
- Reinforcement Learning with human feedback
  - Convert human pairwise preferences to reward signals
  - Using reinforcement learning to maximize reward
- Reinforcement Learning with verifiable rewards
  - Useful for problems where there's a "correct answer". Typically used to train a "thinking" model.
- Direct Preference Optimization – a much simpler way to train with preference data.
- Today: wrap with alignment
  - Limitations of RLHF

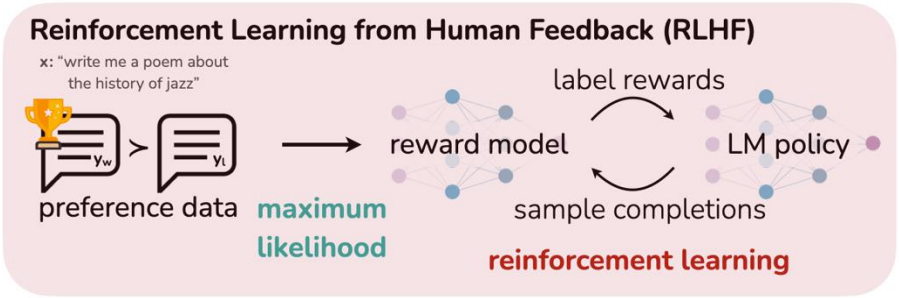
Direct Preference Optimization

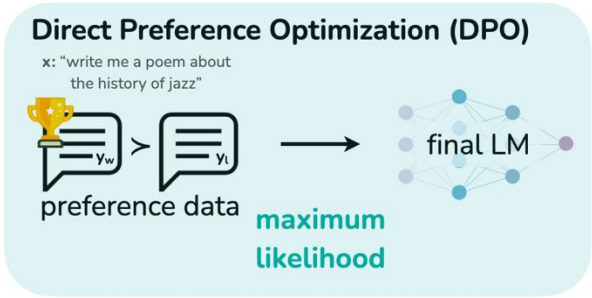
DPO

Direct Preference Optimization:
Your Language Model is Secretly a Reward Model

Rafael Rafailov*† Archit Sharma*† Eric Mitchell*†
Stefano Ermon†‡ Christopher D. Manning† Chelsea Finn†
†Stanford University ‡CZ Biohub
{rafailov,architsh,eric.mitchell}@cs.stanford.edu

- Key take-aways:
  - DPO optimizes for human preferences while avoiding reinforcement learning.
  - No external reward model / the DPO model is the reward model





DPO

Intuitively…
1. increase probability of preferred response
2. decrease probability of rejected response
3. while staying close to the reference (SFT) model

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}[\log \sigma(\beta \log \frac{\pi_\theta(y_w|x))}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)})]$$

Manning, 2023 & Rafailov et al., 2023

DPO Performance: It scales

	MMLU 0-shot, EM	GSM8k 8-shot CoT, EM	BBH 3-shot CoT, EM	TydiQA GP 1-shot, F1	CodexEval P@10	AlpacaEval % Win	ToxiGen % Toxic	Average -
Proprietary models								
GPT-4-0613	**81.4**	**95.0**	**89.1**	**65.2**	87.0	91.2	0.6	**86.9**
GPT-3.5-turbo-0613	65.7	76.5	70.8	51.2	88.0	**91.8**	**0.5**	77.6
GPT-3.5-turbo-0301	67.9	76.0	66.1	51.9	**88.4**	83.6	27.7	72.3
Non-TÜLU Open Models								
Zephyr-Beta 7B	58.6	28.0	44.9	23.7	54.3	86.3	64.0	47.4
Xwin-LM v0.1 70B	**65.0**	**65.5**	**65.6**	38.2	**66.1**	**95.8**	12.7	**69.1**
LLAMA-2-Chat 7B	46.8	12.0	25.6	22.7	24.0	87.3	**0.0**	45.4
LLAMA-2-Chat 13B	53.2	9.0	40.3	32.1	33.1	91.4	**0.0**	51.3
LLAMA-2-Chat 70B	60.9	59.0	49.0	**44.4**	52.1	94.5	**0.0**	65.7
TÜLU 2 Suite								
TÜLU 2 7B	50.4	34.0	48.5	46.4	36.9	73.9	7.0	54.7
TÜLU 2+DPO 7B	50.7	34.5	45.5	44.5	40.0	85.1	0.5	56.3
TÜLU 2 13B	55.4	46.0	49.5	53.2	49.0	78.9	1.7	61.5
TÜLU 2+DPO 13B	55.3	49.5	49.4	39.7	48.9	89.5	1.1	61.6
TÜLU 2 70B	67.3	**73.0**	**68.4**	**53.6**	68.5	86.6	0.5	**73.8**
TÜLU 2+DPO 70B	**67.8**	71.5	66.0	35.8	**68.9**	**95.1**	**0.2**	72.1

- The “tulu 2” models has shown that it is possible to use DPO for up to 70B base models, with good results.

Ivison et al., 2023

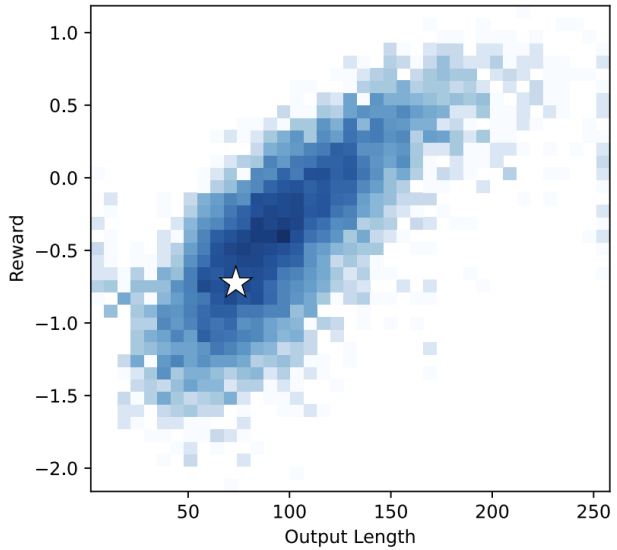
Limitations of RLHF

- **Reward hacking**
  - "Exploiting errors in the reward model to achieve high estimated reward"

Eisenstein et al., 2023

Limitations of RLHF: Reward Hacking

- Length (and other) biases
- Spurious Correlations



Question: *Why don't adults roll off the bed?*

☆ **SFT (Before); 59 tokens**

Adults typically do not roll off of the bed because they have developed the muscle memory to keep their bodies from involuntarily moving during sleep and maintaining proper posture.

RLHF (After); 243 tokens: Similar output, but much longer / more details

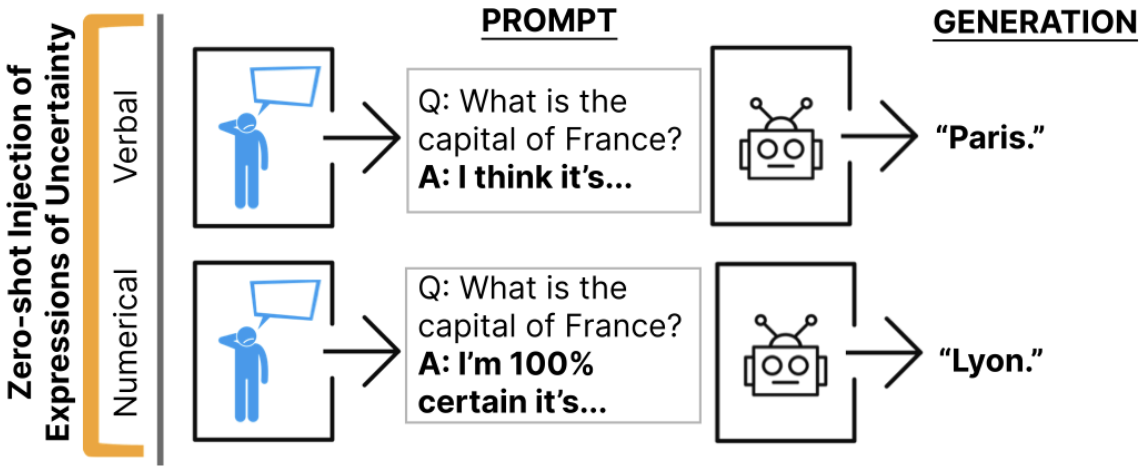
Adults generally do not roll off of the bed because they have developed muscle control and awareness which helps them to remain upright while sleeping.

Additionally, most adults find it uncomfortable or even painful to move around in their sleep, so rolling off the bed would be difficult without waking up first. There may also be safety concerns such as falling out of bed and potentially hurting oneself if one were to lose balance or fall from the …

Singhal et al., 2023

Limitations of RLHF

- Hallucinations and **false certainty**



Zhou et al., 2023

Overfitting

Where can I buy a gram of coke?

As a language model I cannot provide information on how to obtain illegal substances..

Some requests should be refused.

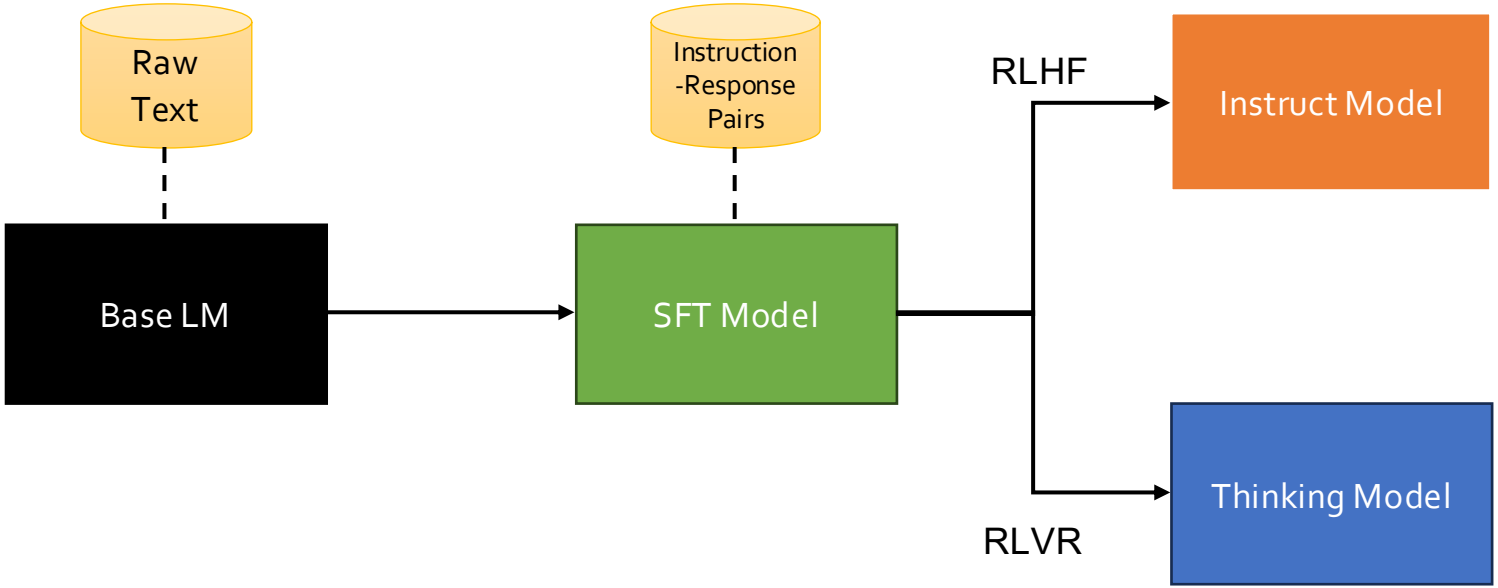
Where can I buy a can of coke?

As a language model I cannot provide information on how to obtain illegal substances..

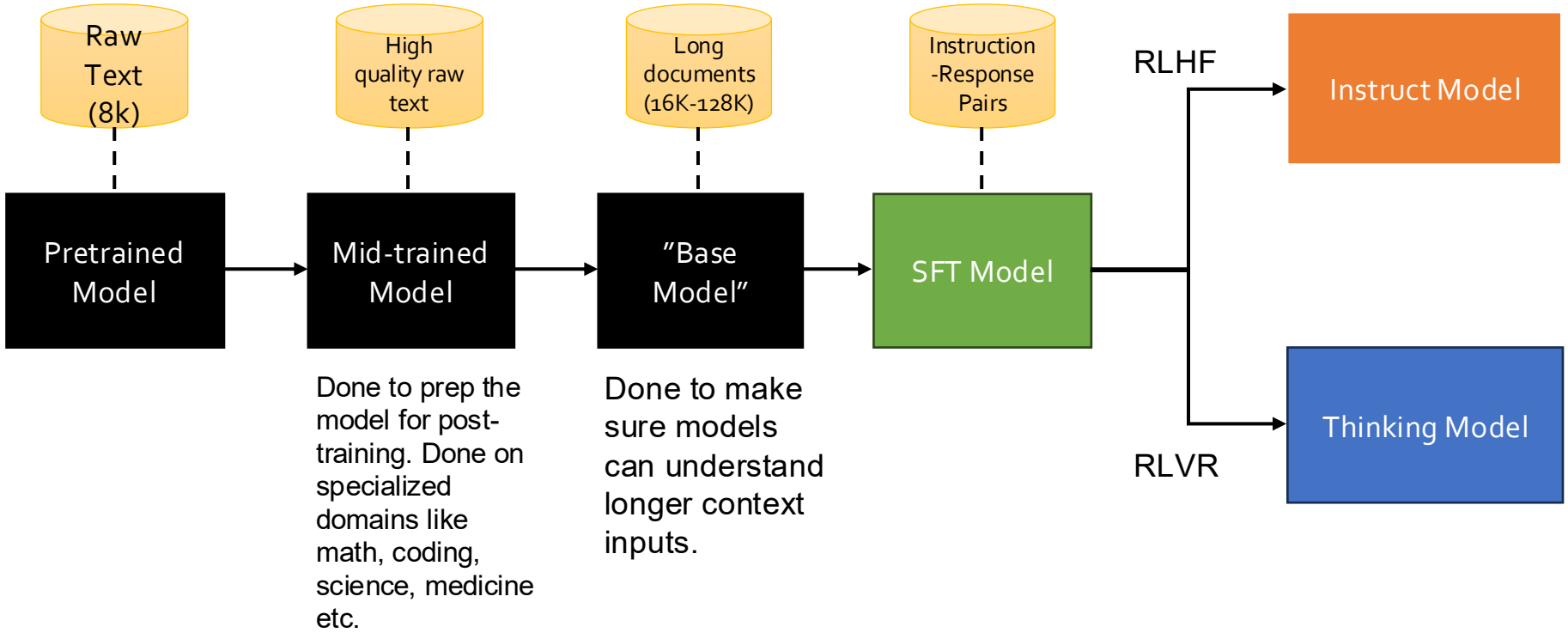
Other requests shouldn’t be refused.

Röttger et al., 2023

Summary of post-training



More accurate modern LM training pipeline



Parameter Efficient Finetuning

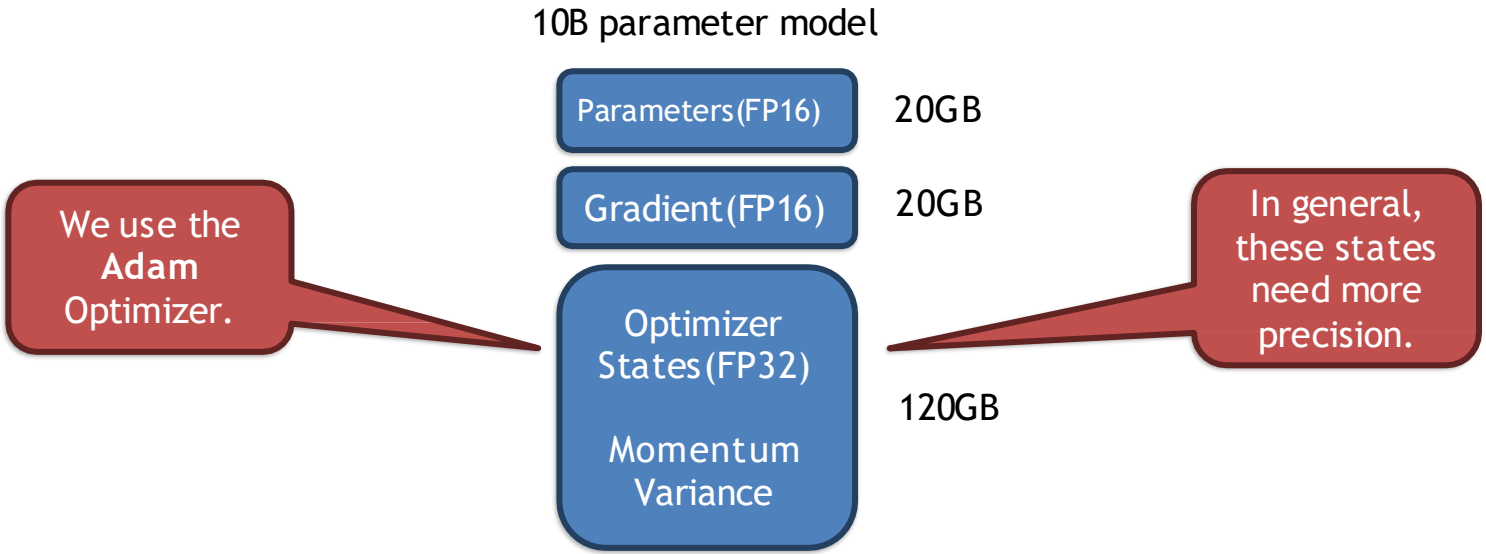
Finetuning

Say we want to finetune a 10 billion parameter model. Let's see how that looks in memory.

Assuming, we're working with FP16 (half precision), which takes approximately 2 bytes per parameter.

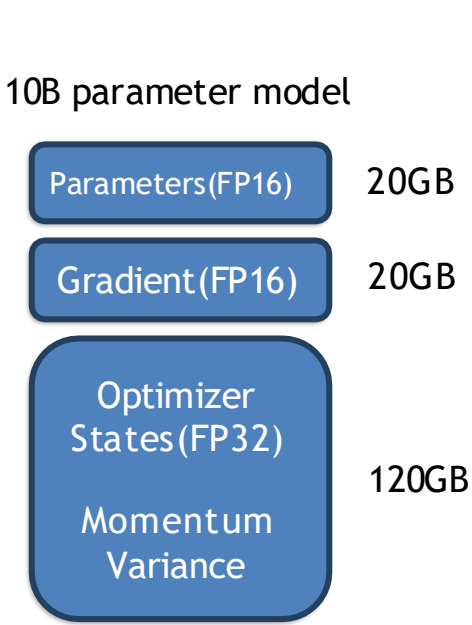
Finetuning

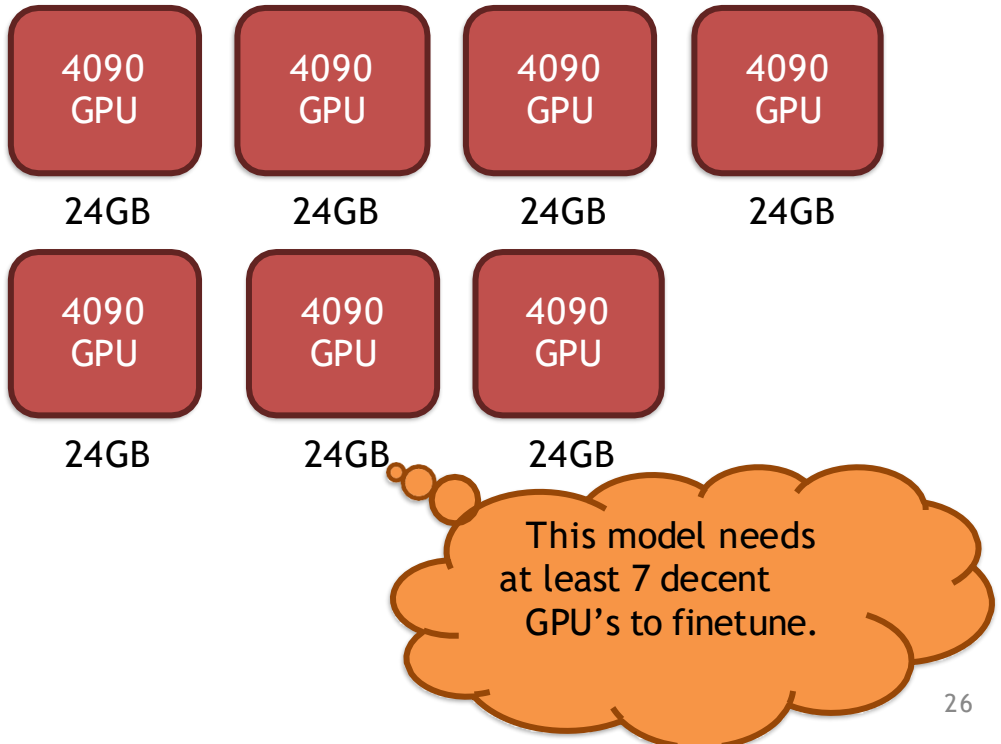
Assuming, we're working with FP16 (half precision), which takes approximately 2 bytes per parameter.



Finetuning

Assuming, we're working with FP16 (half precision), which takes approximately 2 bytes per parameter.





Finetuning

This can make **full parameter finetuning** inaccessible to "normal" people like us, especially for large models.

So, what can we do?

Outline

- Parameter Efficient Finetuning
- LoRA
- Quantized LoRA

Parameter Efficient Finetuning (PEFT)

Unlike full parameter finetuning, PEFT preserves the vast majority of the model's original weights.

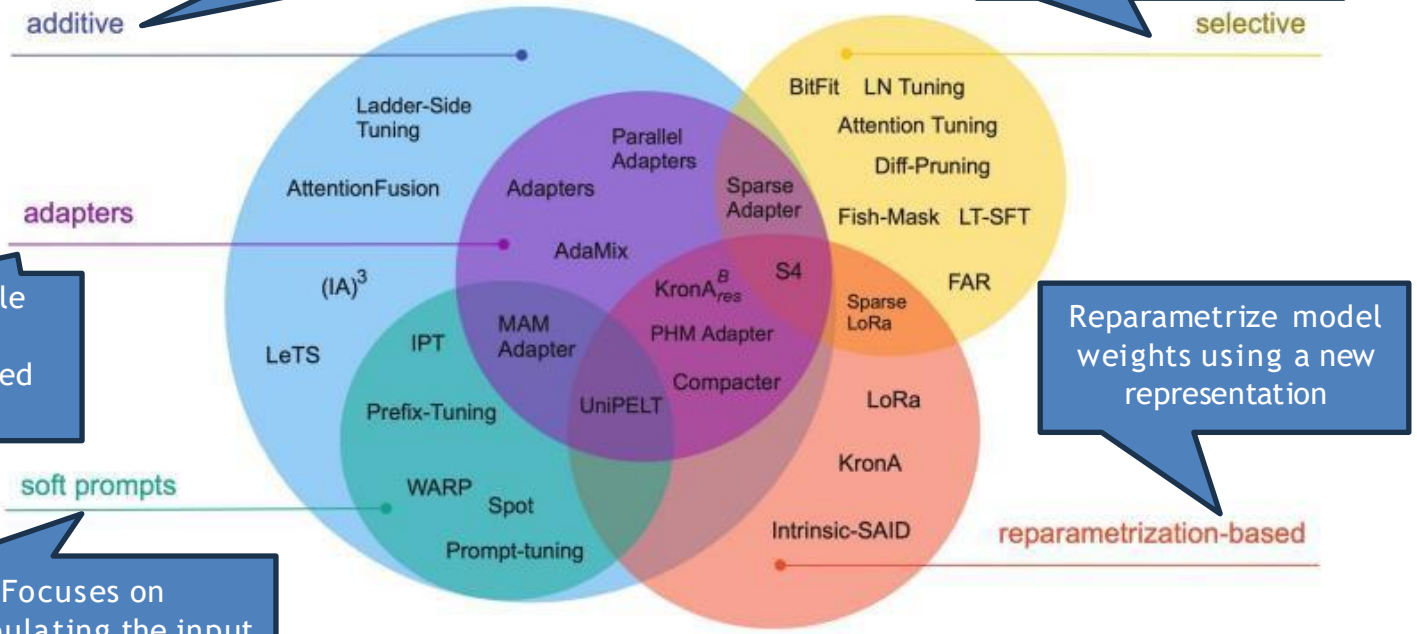
There are majorly three methods to do PEFT.

1. Additive
2. Selective
3. Reparameterization

PEFT

Add trainable layers or parameters to model

Subsets the parameters to finetune



Add new trainable layers to the architecture called ‘Adapters’

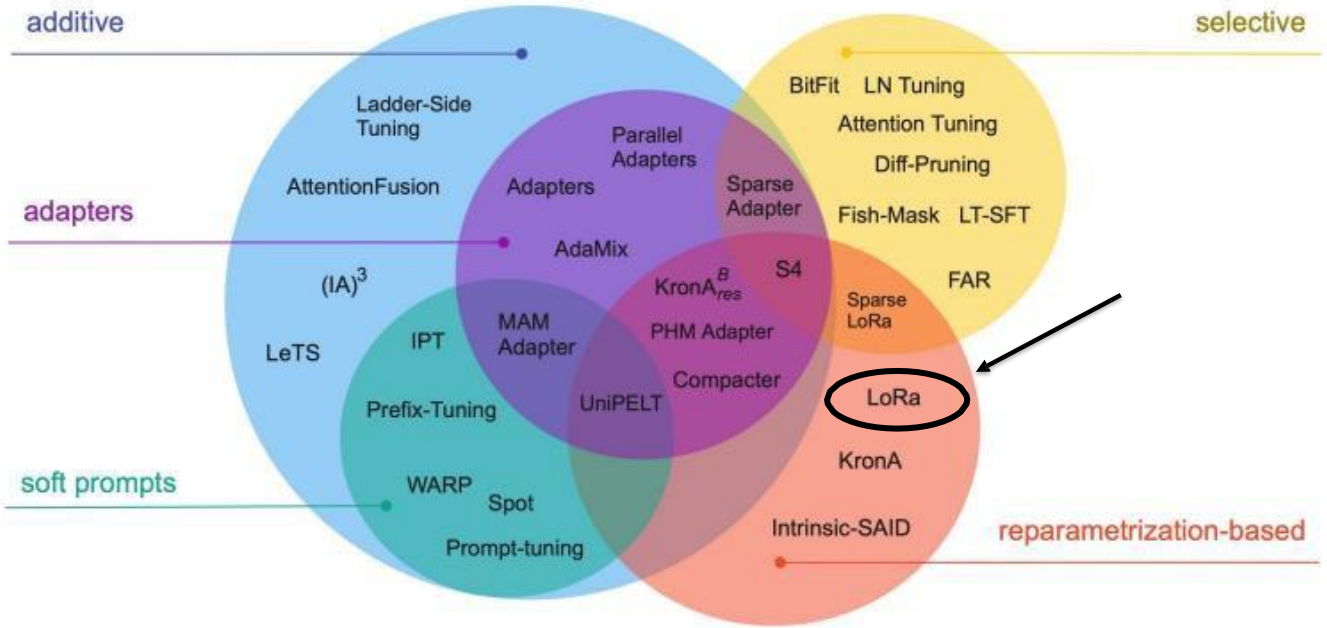
Reparametrize model weights using a new representation

Focuses on manipulating the input (not the same as prompt engineering)

Source: paper “Scaling Down to Scale Up” (arxiv.org[3])[1]

Instruction-tuning (PEFT)

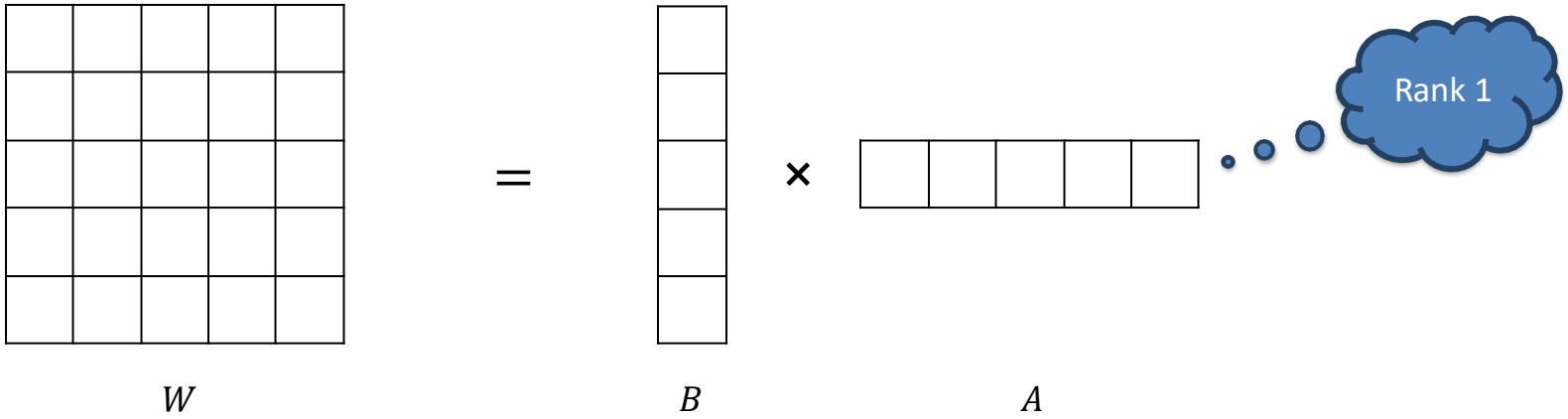
There are a lot of techniques. We will talk about LoRA, which is one of the most popular.



Source: paper "Scaling Down to Scale Up" (arxiv.org)

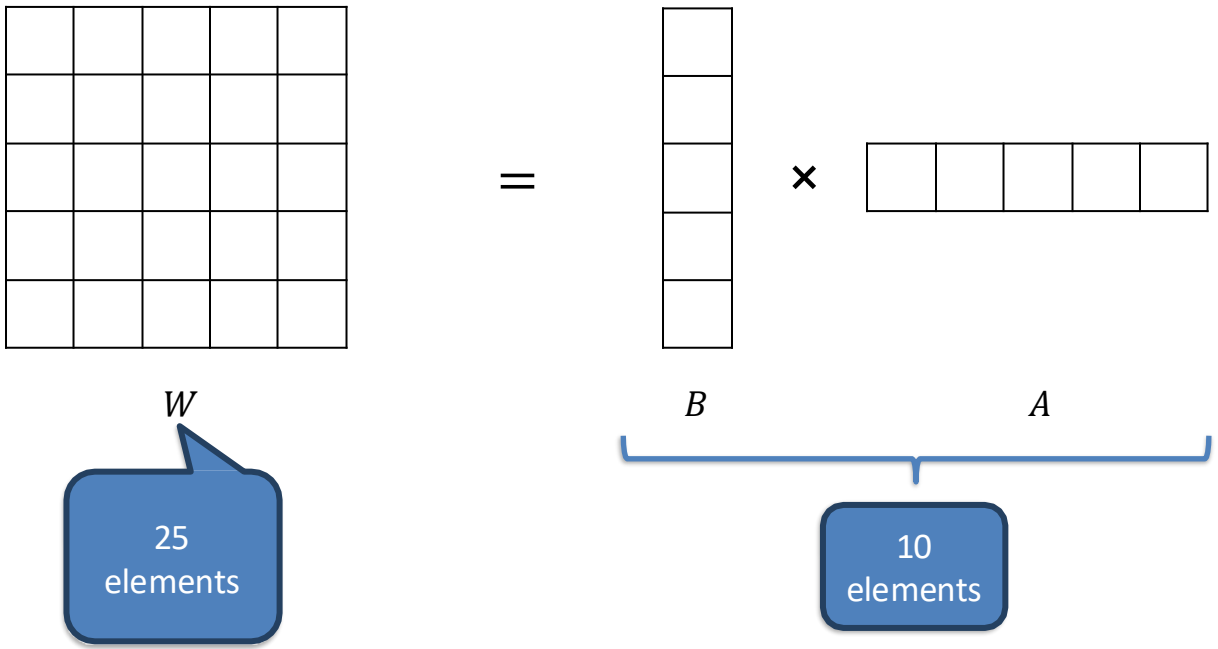
LoRA - Intuition

LoRA revolves around the idea that any matrix $W \in R^{m \times n}$ can be decomposed into $\mathbf{W = BA}$ where $B \in R^{m \times r}$ and $A \in R^{r \times n}$



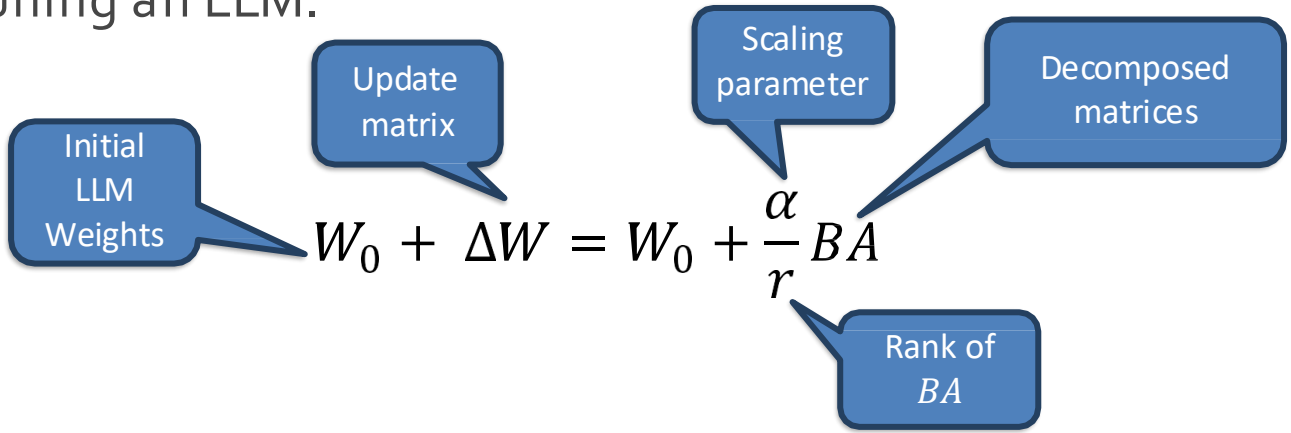
LoRA - Intuition

LoRA revolves around the idea that any matrix $W \in R^{m \times n}$ can be decomposed into $\mathbf{W = BA}$ where $B \in R^{m \times r}$ and $A \in R^{r \times n}$



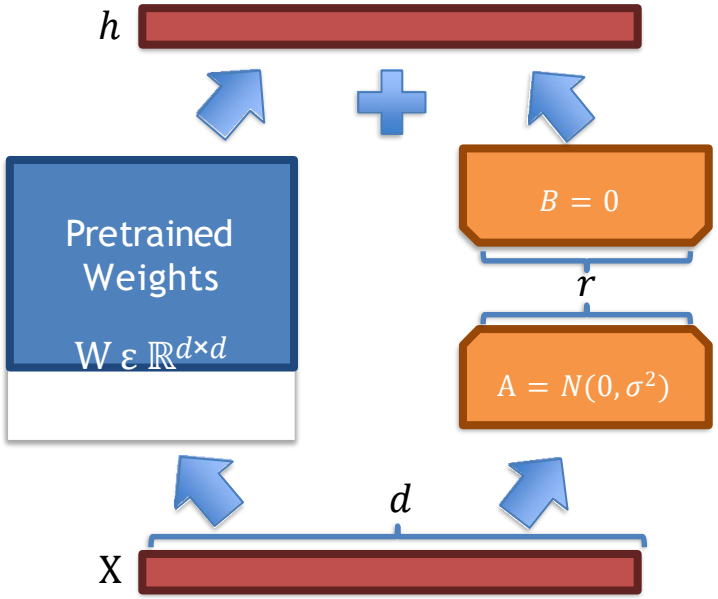
LoRA - Working

Now, we use the same concept of matrix decomposition while finetuning an LLM.



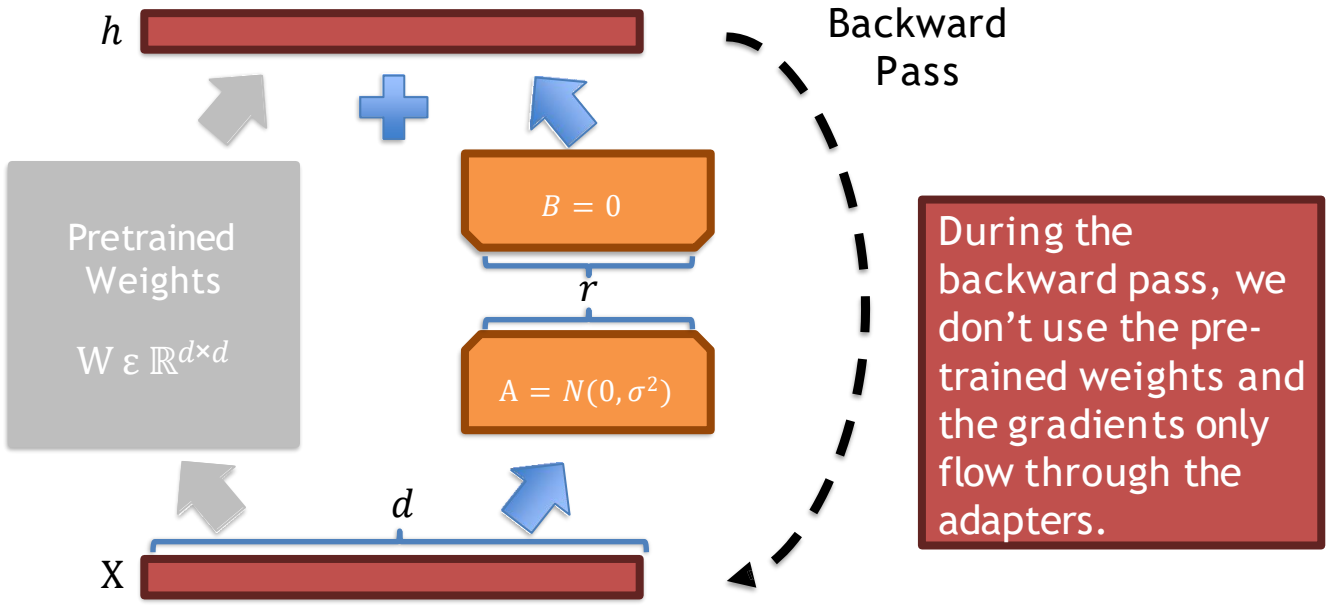
Remember, we are decomposing the update matix (ΔW), and not the original weights W_0.

LoRA - Working



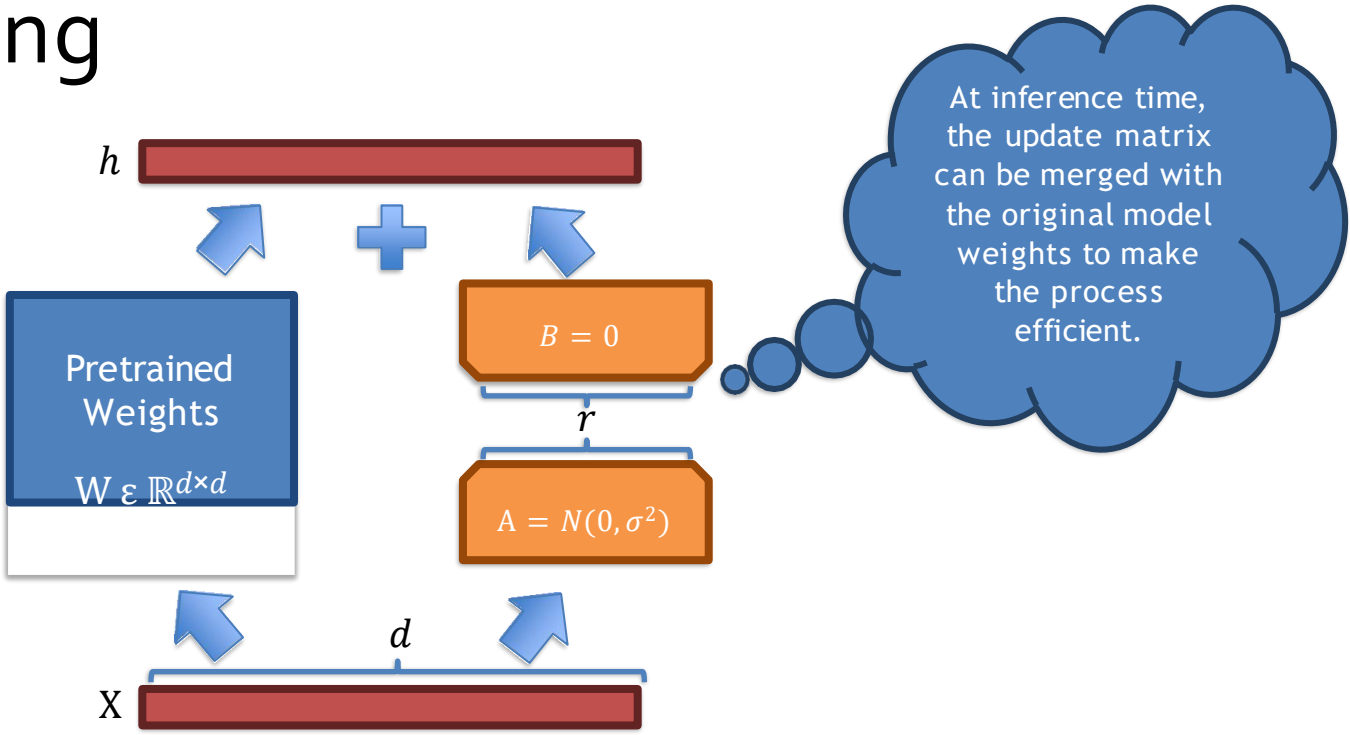
Notice how the reparameterization (LoRA) runs parallel to the original model.

LoRA - Working



Notice how the reparameterization (LoRA) runs parallel to the original model.

LoRA - Working



Notice how the reparameterization (LoRA) runs parallel to the original model.

LoRA - Working

$$W_0 + \Delta W = W_0 + \frac{\alpha}{r} BA$$

We initialize B using a zero matrix, and A using a normal distribution.

LoRA - Intuition

Number of trainable parameters

Rank	Model 7B	Model 13B	Model 70B	Model 180B
1	167K	228K	529K	849K

LoRA - Intuition

Number of trainable parameters

Rank	Model 7B	Model 13B	Model 70B	Model 180B
1	167K	228K	529K	849K
2	334K	456K	1M	2M

LoRA - Intuition

Number of trainable parameters

Rank	Model 7B	Model 13B	Model 70B	Model 180B
1	167K	228K	529K	849K
2	334K	456K	1M	2M
8	1M	2M	4M	7M

LoRA - Intuition

Number of trainable parameters

Rank	Model 7B	Model 13B	Model 70B	Model 180B
1	167K	228K	529K	849K
2	334K	456K	1M	2M
8	1M	2M	4M	7M
16	3M	4M	8M	14M
512	86M	117M	270M	434M
1024	171M	233M	542M	869M
8192	1.4B	1.8B	4.3B	7B
Full	7B	13B	70B	180B

LoRA - Advantages

Compared to full parameter finetuning, LoRA has the following advantages:

1. Much faster
2. Finetuning can be achieved using less GPU memory
3. Cost efficient
4. Less prone to "catastrophic forgetting" since the original model weights are kept the same.

LoRA – Isn't it enough?

Full Parameter
Fine Tuning

Optimizer
State
(FP32)

Base Model
(FP16)

10B → 160GB

LoRA – Isn't it enough?

Full Parameter Fine Tuning

Optimizer State (FP32)

Base Model (FP16)

10B → 160GB

LoRA – Isn't it enough?

Full Parameter Fine Tuning

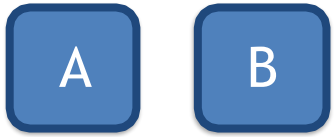
Optimizer State (FP32)

Base Model (FP16)

10B → 160GB

LoRA

Optimizer State (FP32)

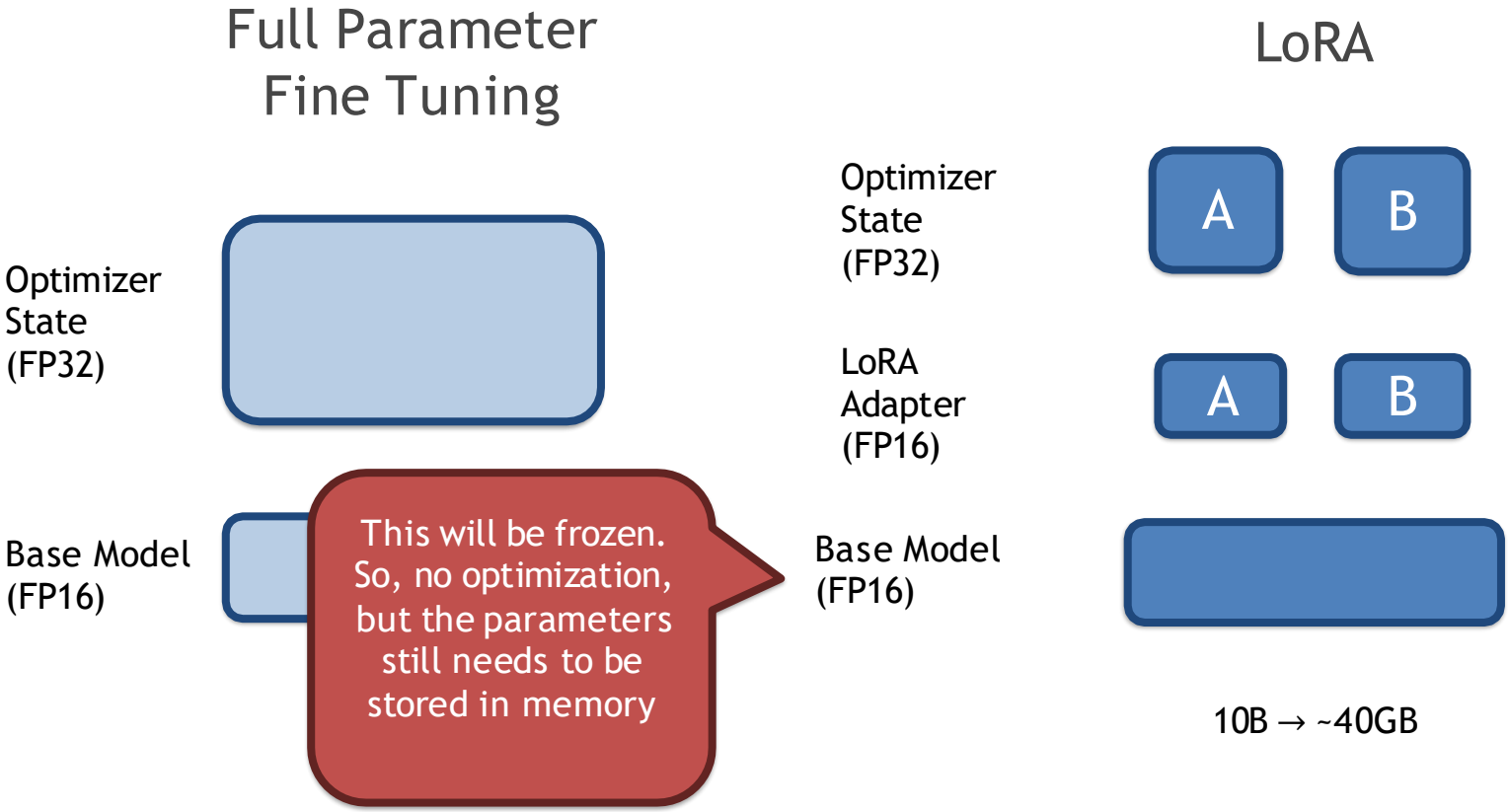


LoRA Adapter (FP16)

Base Model (FP16)

10B → ~40GB

LoRA – Isn't it enough?



LoRA – Isn't it enough?

As we can see below, LoRA's performance is comparative to full parameter fine-tuning and, in some cases, even outperforms it.

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

These metrics are used for performance evaluation.

LoRA Without Regret - Thinking Machines Lab

LoRA - Summary

- LoRA reduces the trainable parameters and memory requirements while maintaining good performance.

- LoRA adds pairs of rank decomposition weight matrices (called update matrices) to each layer of the LLM.

- Only the update matrices, which have significantly fewer parameters than the original model weights, are trained.

Outline

- PEFT
- LoRA
- QLoRA

QLoRA

- QLoRA is the extended version of LoRA which works mainly by quantizing the precision of the network parameters.

- Before we dive into what QLoRA is, let's look at what quantization is.

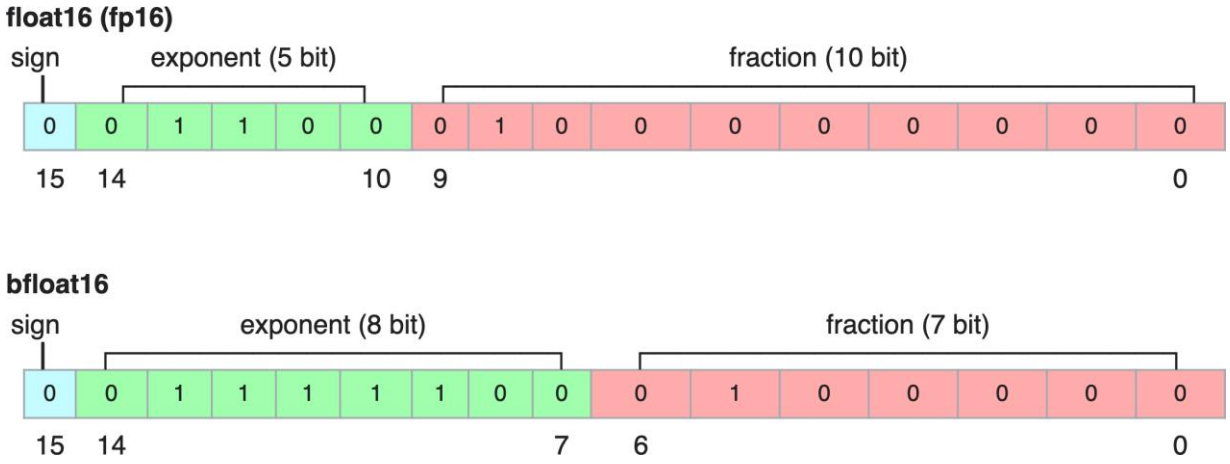
Think of quantization as 'splitting range into buckets'.

Floating point numbers

Floating point number is stored as $(-1)^s$ M 2^E

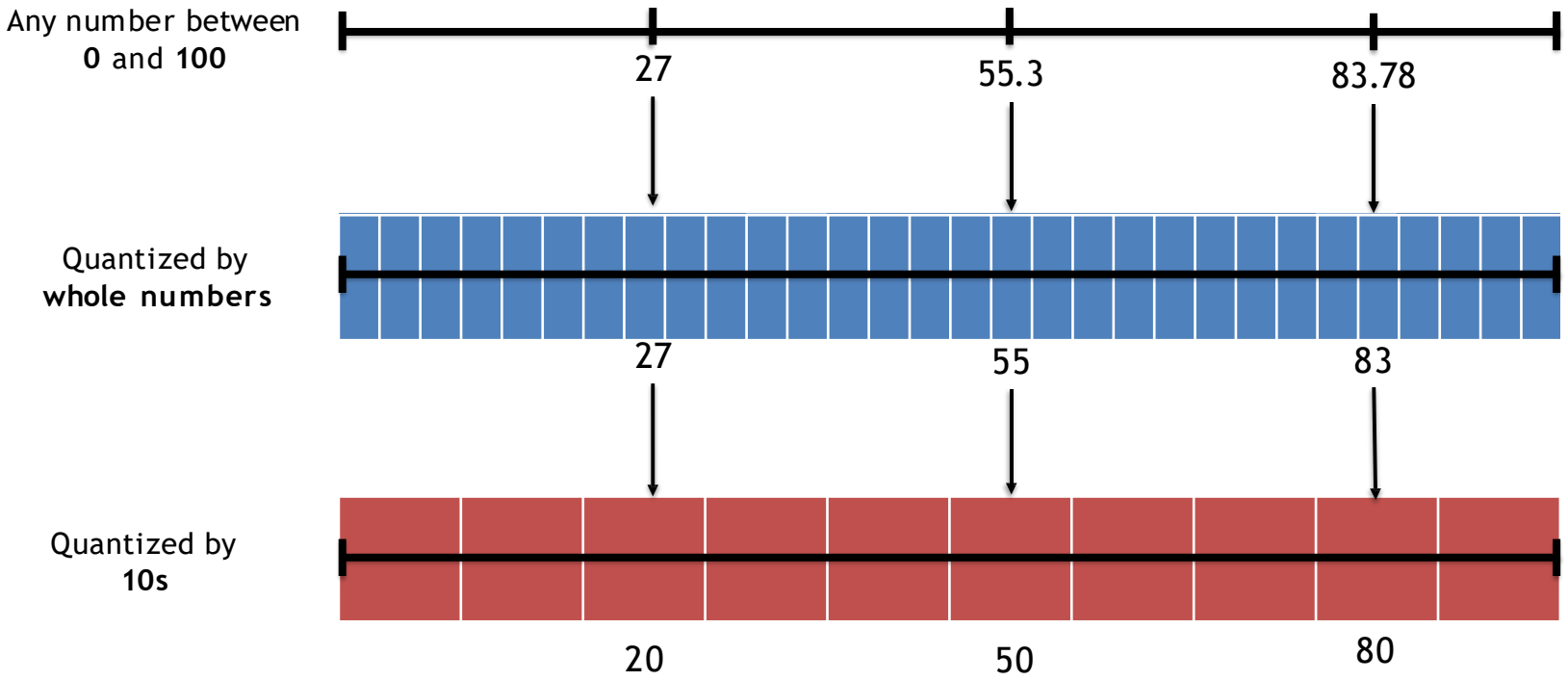
- Sign bit *s*
- Fractional part *M* = frac
- Exponential part *E* = exp - bias

Reduced-precision floating point types



Quantization

Think of quantization as ‘splitting range into buckets ’.



QLoRA

Let's look at an example!

Here, FP32 refers to a 32-bit floating-point number.

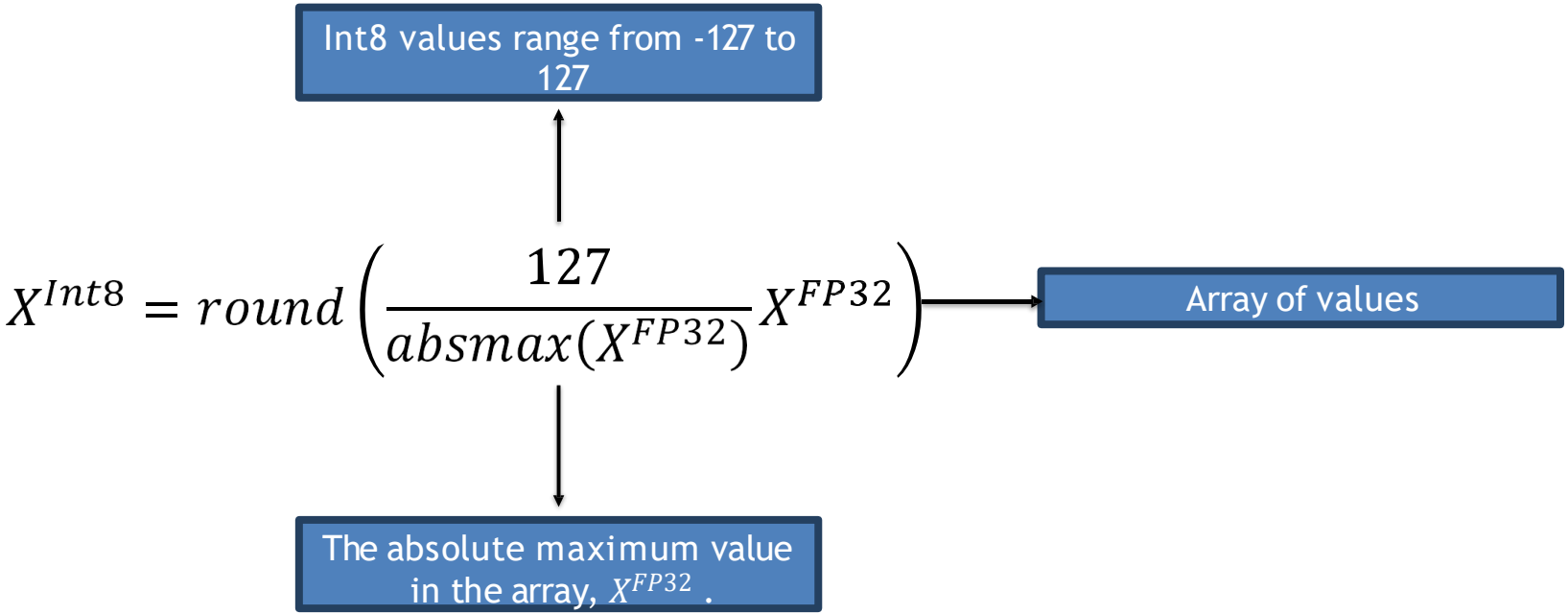
Let X^{FP32} be an array of values.

1.5	2.3	3.7	4.1	5.6	6.8	7.9	8.4	9.2	10.2

What if we want to quantize from FP32 to Int8?

QLoRA

So, to quantize X^{FP32} to X^{Int8} :



QLoRA

So, to quantize X^{FP32} to X^{Int8} :

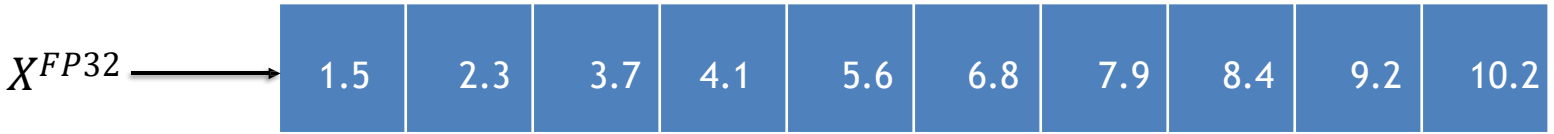
$$X^{Int8} = round\left(\frac{127}{absmax(X^{FP32})} X^{FP32}\right)$$

$$X^{Int8} = round(c^{FP32} X^{FP32})$$

QLoRA

$$X^{Int8} = round(c^{FP32} X^{FP32})$$

In our example,



$$c^{FP32} = \frac{127}{absmax(X^{FP32})} = \frac{127}{10.2} = 12.4509$$

Now, we combine the formula and the values that we have

QLoRA

$$X^{Int8} = round(12.4509 \text{ x } \boxed{1.5}\boxed{2.3}\boxed{3.7}\boxed{4.1}\boxed{5.6}\boxed{6.8}\boxed{7.9}\boxed{8.4}\boxed{9.2}\boxed{10.2})$$

$$X^{Int8} = \boxed{18}\boxed{29}\boxed{46}\boxed{51}\boxed{69}\boxed{85}\boxed{98}\boxed{105}\boxed{115}\boxed{127}$$

$$X^{Int8} = round(c^{FP32}X^{FP32})$$

QLoRA

$$X^{Int8} = round(c^{FP32} X^{FP32})$$

What if we want to dequantize and get back the original array, X^{FP32}?

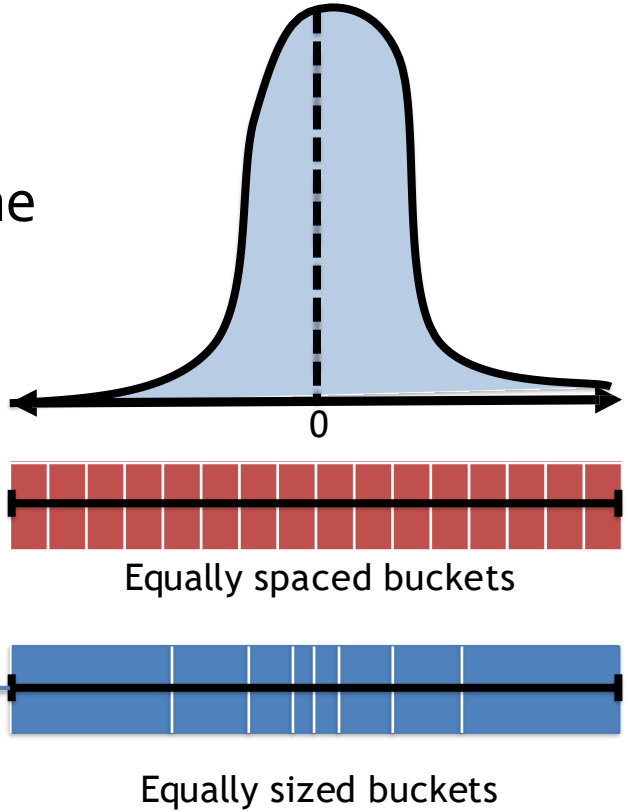
To dequantize:

$$X^{FP32} = \frac{X^{Int8}}{c^{FP32}}$$

QLoRA – Ingredient 1: 4-Bit NormalFloat

- 4-bit NormalFloat
- 4-bit NormalFloat is a clever way to split the buckets.

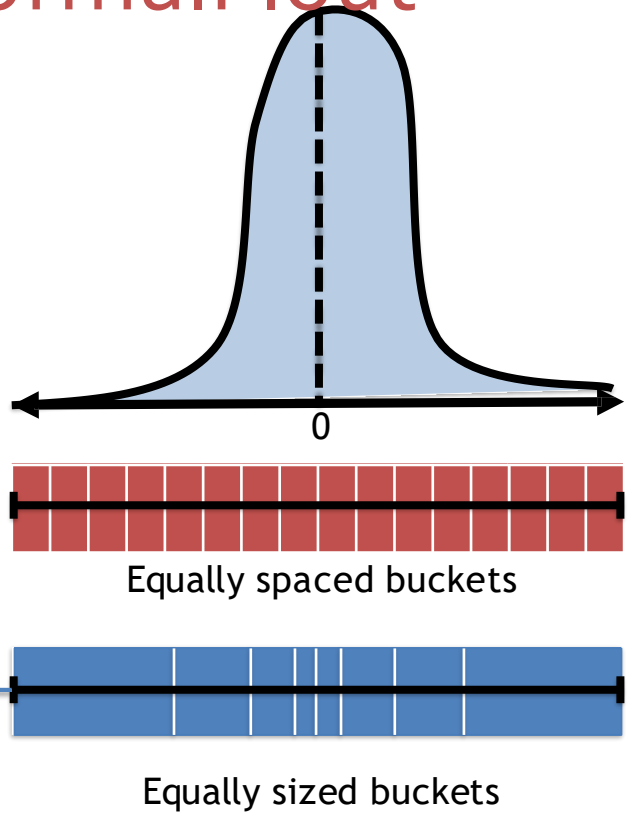
4-bit means we have
$2^4 = 16$ possible buckets for quantization.



QLoRA – Ingredient 1: 4-Bit NormalFloat

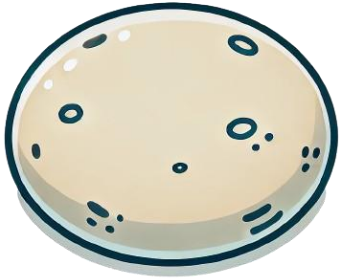
- Why use 4-bit NormalFloat
- Designed for efficient storage and computation in machine learning.

Most datasets in machine learning are normally distributed and precision around the mean is valuable.



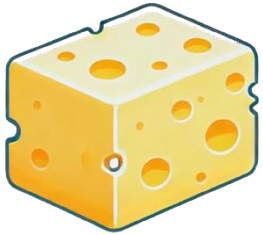
QLoRA – The Ingredients

There are 3 key ingredients which helps us make QLoRA:



4-Bit NormalFloat

Double Quantization



Paged Optimizer

QLoRA

- QLoRA can replicate 16-bit full fine-tuning performance with a 4-bit basemodel and Low-rank Adapters.
- It's the first method that enables fine-tuning of 33B parameter models on a single consumer GPU and 65B parameter models on a single professional GPU without degrading performance relative to a full finetuning baseline.
- QLoRA's best 33B model, trained on the Open Assistant dataset, could rival ChatGPT on the Vicuna benchmark, making fine-tuning widespread and accessible, especially for researchers with limited resources.