

# CME 295: Transformers & Large Language Models



Afshine Amidi & Shervine Amidi



# Logistics

## Midterm.

- **When?** Friday October 24th, 3:30pm - 5:00pm (duration: 1h30)
- **Where?** Thornton 110 (i.e. this classroom)
- **Topics?** Lecture 1 + 2 + 3 + 4

# Logistics

## Midterm.

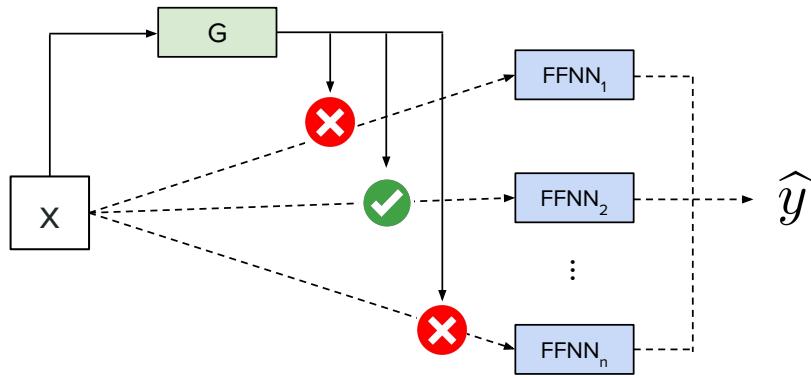
- **When?** Friday October 24th, 3:30pm - 5:00pm (duration: 1h30)
- **Where?** Thornton 110 (i.e. this classroom)
- **Topics?** Lecture 1 + 2 + 3 + 4

## Final.

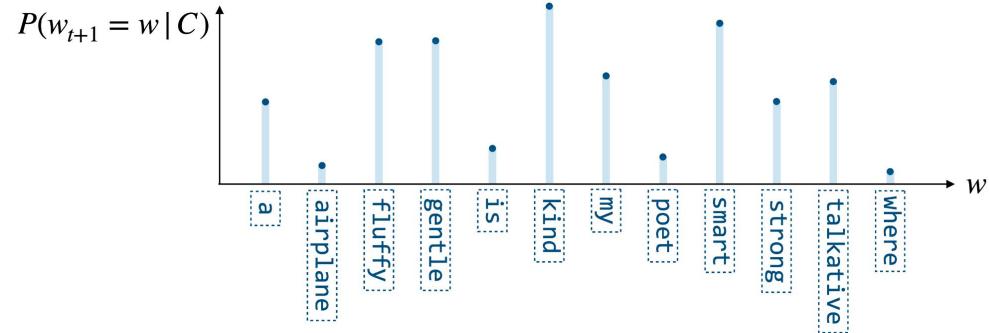
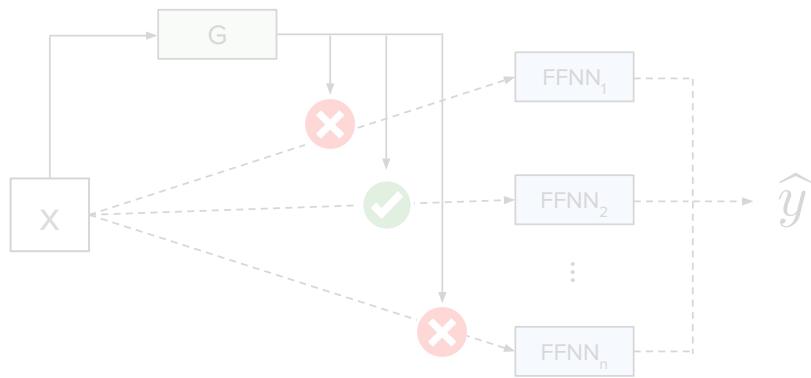
- **When?** Wednesday December 10th, 7:00pm - 8:30pm (duration: 1h30)
- **Where?** Hewlett Teaching Center 201
- **Topics?** Lecture 5 + 6 + 7 + 8 + 9

# Recap of last episode...

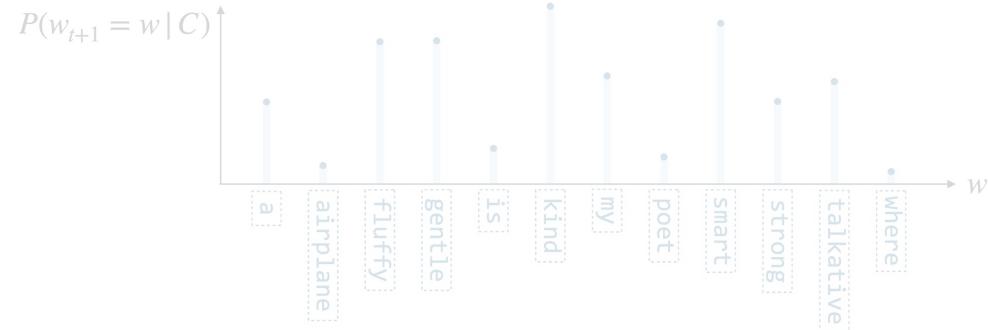
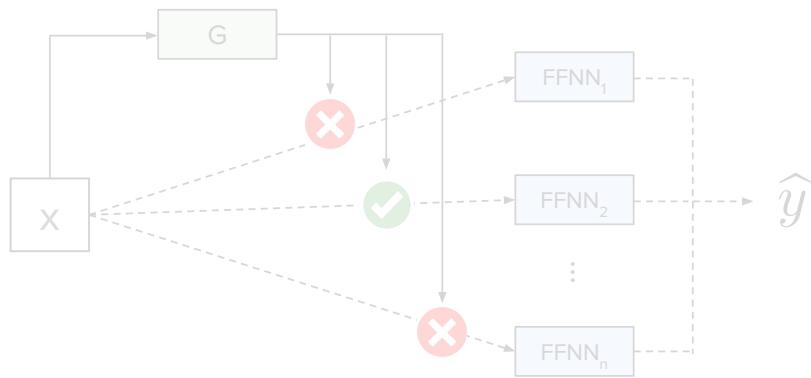
# Recap of last episode...



# Recap of last episode...



# Recap of last episode...



**Inference optimizations.** KV cache, PagedAttention, speculative decoding, GQA, multi-head latent attention, multi-token prediction



# Transformers & Large Language Models

## Pretraining

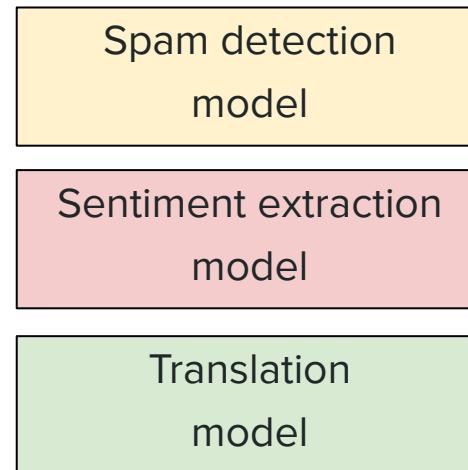
Training optimizations

Supervised finetuning

Parameter-efficient finetuning

# Paradigm shift

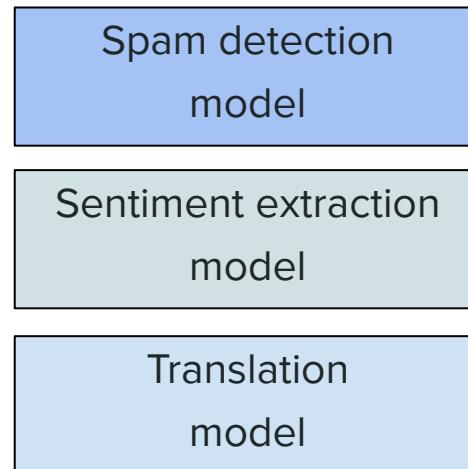
**Traditional ML.** Given task, train model from scratch



# Paradigm shift

**Traditional ML.** Given task, train model from scratch

**Transfer learning.** Reuse trained model to some capacity



# Paradigm for LLM training

**LLM training.** Train LLM to understand language, then tune it for end task

# Paradigm for LLM training

**LLM training.** Train LLM to understand language, then tune it for end task

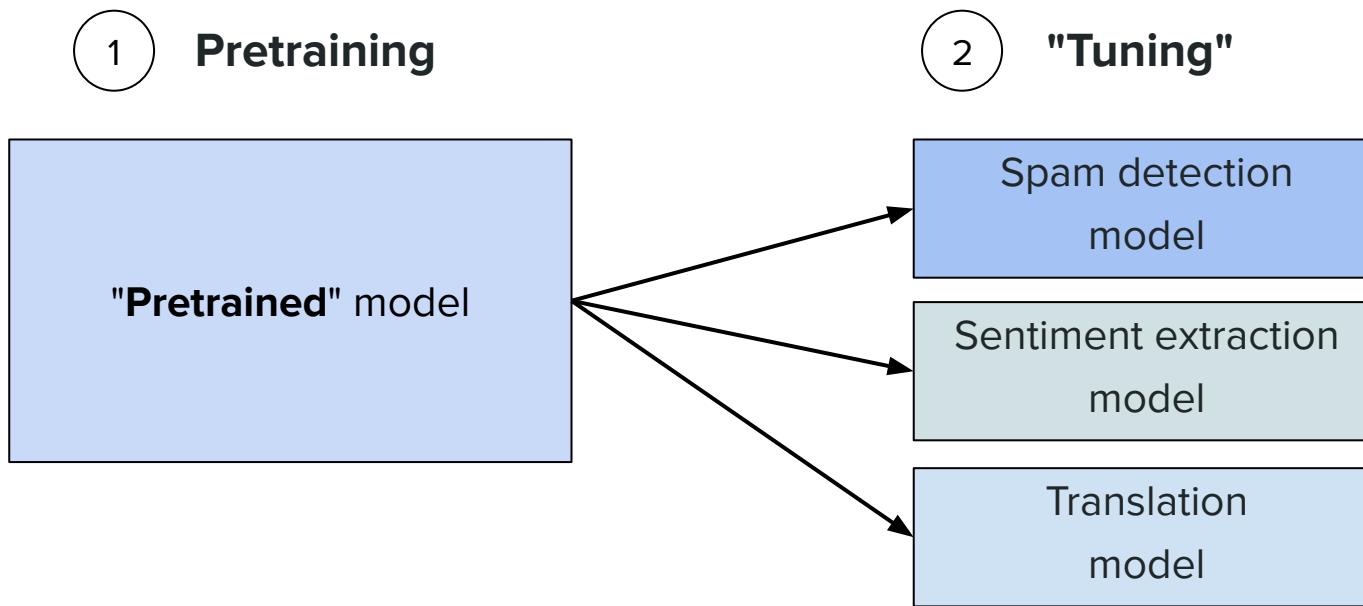
1

## Pretraining

"**Pretrained**" model

# Paradigm for LLM training

**LLM training.** Train LLM to understand language, then tune it for end task



# Pretraining overview

**Goal.** Learn pattern of language and code

English

My cute teddy bear is reading by the window, his tiny paws holding a book almost bigger than him. The afternoon light dances across his soft fur as he turns each page with great care...

Multi-lingual

Mon ours en peluche est mon plus fidèle compagnon.  
Toujours assis sur mon lit, il m'attend patiemment, prêt à écouter mes secrets...

خرس تدی من صمیمی‌ترین دوست من هست. همیشه رو تخت  
من نشسته و حاضر به قصه‌های من گوش بد...

...

teddy\_bear.py

```
class TeddyBear:  
    def __init__(self, name):  
        self.name = name  
  
    def hug(self):  
        print(f"{self.name} gives a warm hug!")
```

TeddyBear("Fluffy").hug()...

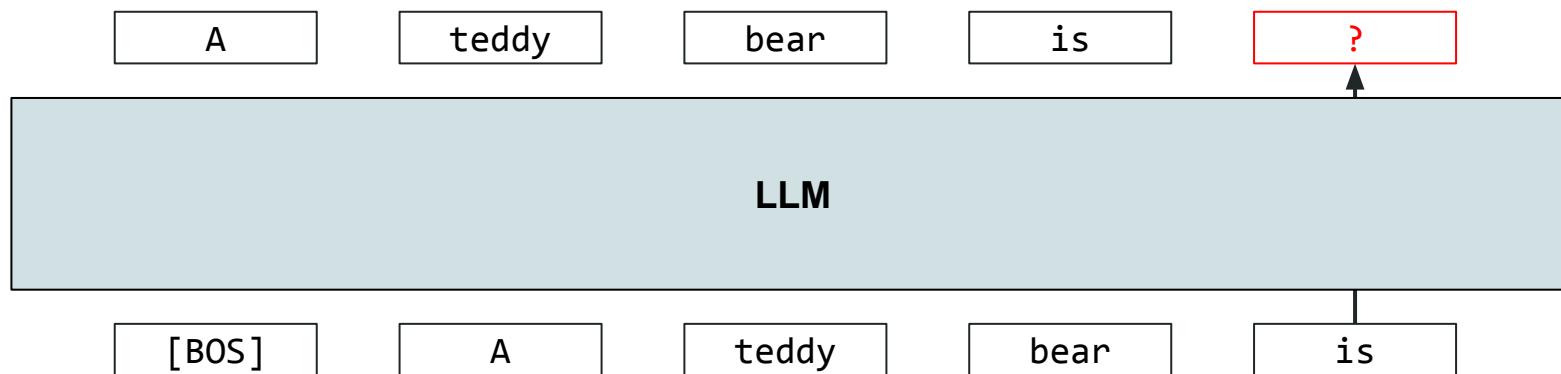
teddy\_bear.go

```
package main  
  
func main(){  
    println("Teddy bear hug!")  
}
```

# Pretraining overview

**Goal.** Learn pattern of language and code

**Objective function.** Predict next token



# Pretraining overview

Goal. Learn pattern of language and code

Objective function. Predict next token

## Data mixtures.

- Web-scraped (e.g. Common Crawl, Wikipedia)
- Code (GitHub, StackOverflow)

anything you find on internet

data used to just to understand  
the structure of the language  
and code

WIKIPEDIA  
The Free Encyclopedia

Search | Donate | Create account | Log in | ...

## Teddy bear

Article | Talk | Read | Edit | View history | Tools | Checked | Languages

From Wikipedia, the free encyclopedia

For other uses, see [Teddy bear \(disambiguation\)](#).

A **teddy bear**, or simply a **teddy**, is a stuffed toy in the form of a **bear**. The teddy bear was named by [Morris Michtom](#) after the 26th president of the United States, [Theodore Roosevelt](#); it was developed apparently simultaneously in the first decade of the 20th century by two toymakers: [Richard Steiff](#) in Germany and Michtom in the United States. It became a popular children's toy, and it has been celebrated in story, song, and film.<sup>[1]</sup>

Bear thought to be made by Morris Michtom in the early 1900s; donated to the Smithsonian Museum of Natural History in Washington, D.C., United States, by Theodore Roosevelt's grandson Kermit Roosevelt Jr. in 1964

A replica Steiff model 55PB displayed at the Steiff-Museum, Giengen, Germany, in 2006; no original examples of the 55PB are known to survive



shervinea / enzyne Public

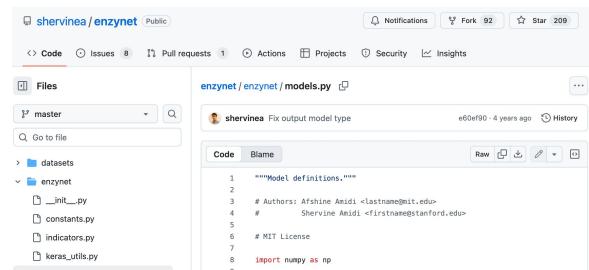
Code Issues 8 Pull requests 1 Actions Projects Security Insights

enzyne / enzyne / models.py

shervinea Fix output model type e60ef90 · 4 years ago · History

Code Blame Raw

```
1 """Model definitions."""
2
3 # Authors: Arshine Amidi <lastname@mit.edu>
4 # Shervine Amidi <firstname@stanford.edu>
5
6 # MIT License
7
8 import numpy as np
```



# Pretraining overview

**Goal.** Learn pattern of language and code

**Objective function.** Predict next token

**Data mixtures.**

- Web-scraped (e.g. Common Crawl, Wikipedia)
- Code (GitHub, StackOverflow)

**Size.** Approx. trillions of tokens

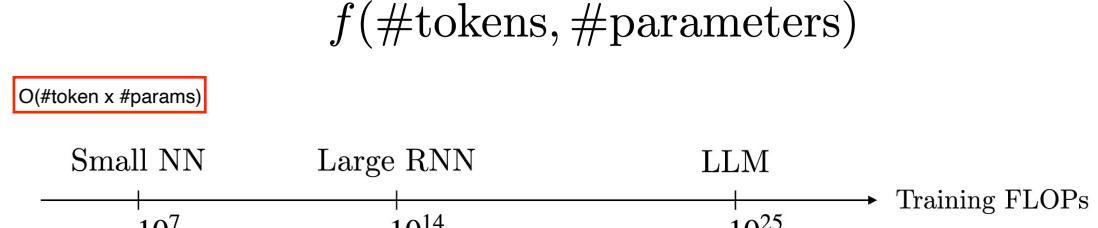
Model	Pretraining size (# tokens)
GPT-3	300 billion
LLaMA 3	15 trillion

# Notations



**FLOPs**

**FLoating-point OPerations**



# Notations



FLOPs

*Floating-point Operations*



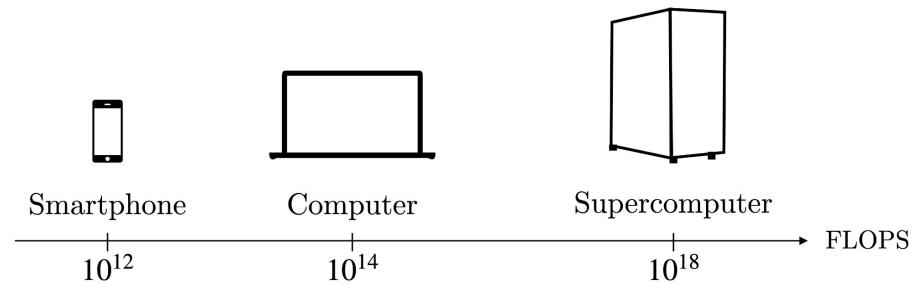
**FLOPS or FLOP/s**

*Floating-point Operations per Second*

$f(\# \text{tokens}, \# \text{parameters})$

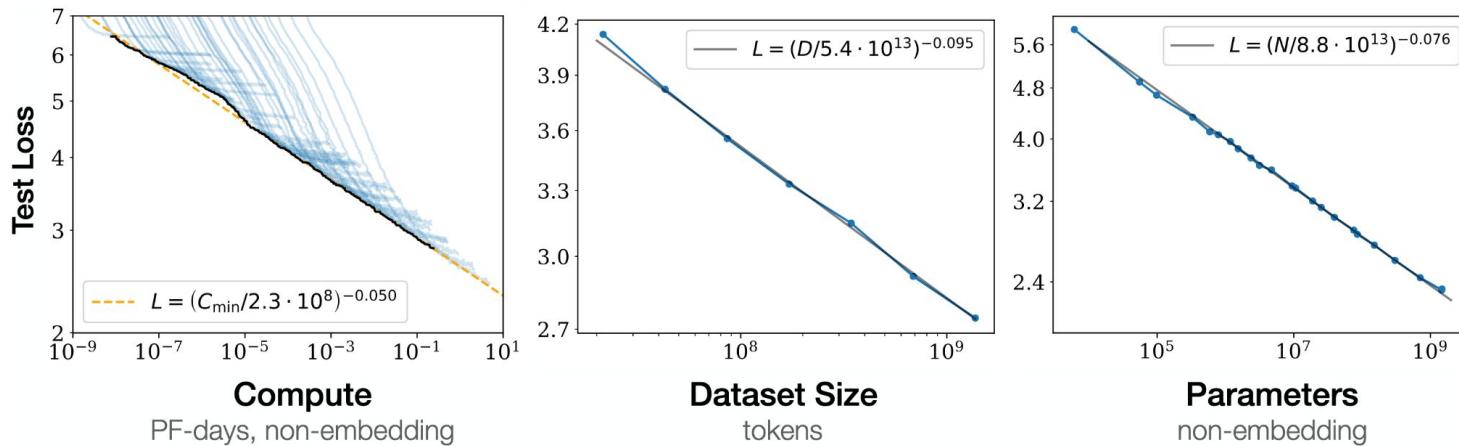


Computational performance



# Takeaways

## Scaling.

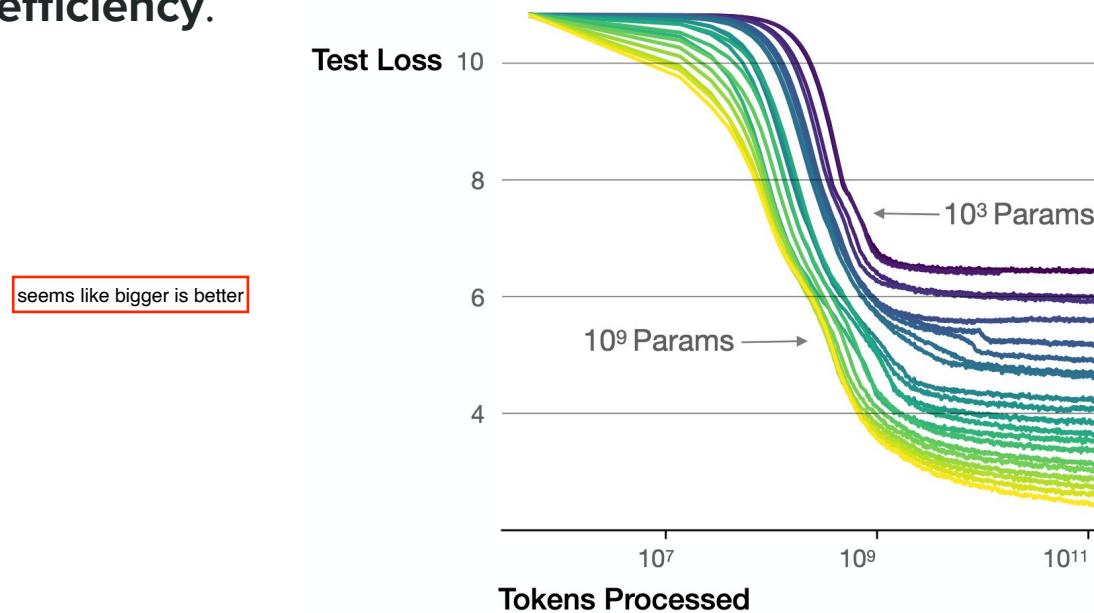


seems like bigger is better

# Takeaways

## Scaling.

### Sample efficiency.

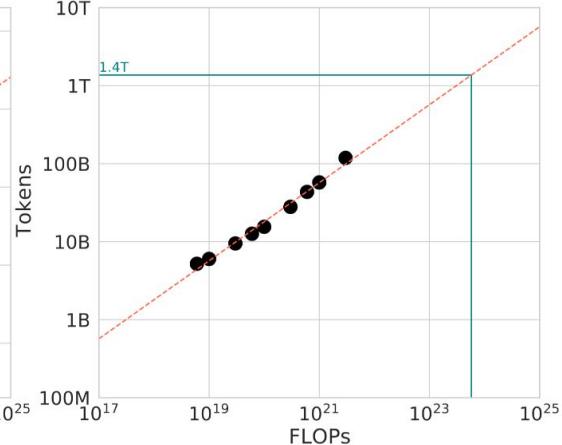
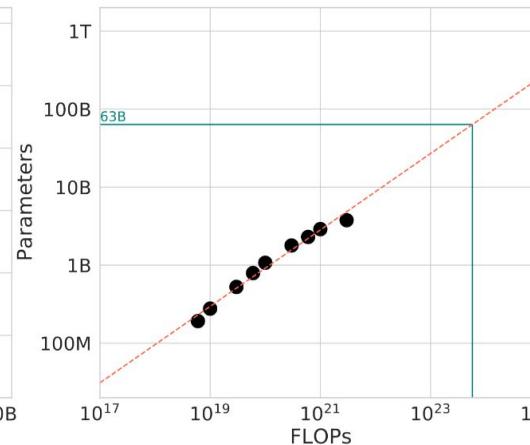
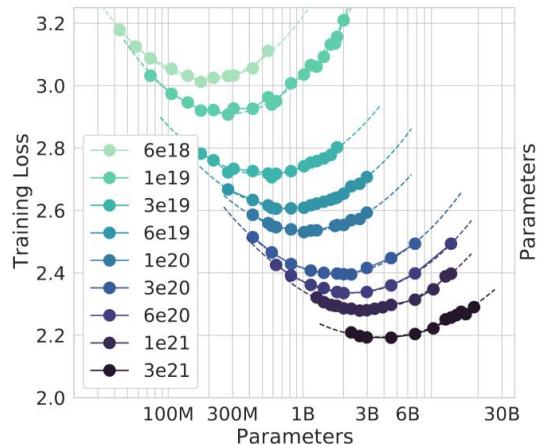


# Takeaways

## Scaling.

## Sample efficiency.

### Chinchilla law.



# Takeaways

## Scaling.

## Sample efficiency.

### Chinchilla law.

Parameters	FLOPs	FLOPs (in <i>Gopher</i> unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

GPT-3 is undertrained  
175B params  
300B tokens  
should be 700B

architecture change doesn't play  
a big role. the thing that does  
is the amount of tokens and the  
size of the model

# Challenges

## Cost.

- (At least) millions of dollars
- Takes time
- Environment / electricity

# Challenges

## Cost.

- (At least) millions of dollars
- Takes time
- Environment / electricity

## Learned knowledge.

- "Knowledge cutoff"  
no knowledge from after  
the pretraining date
- Hard to edit knowledge
- "Plagiarism"

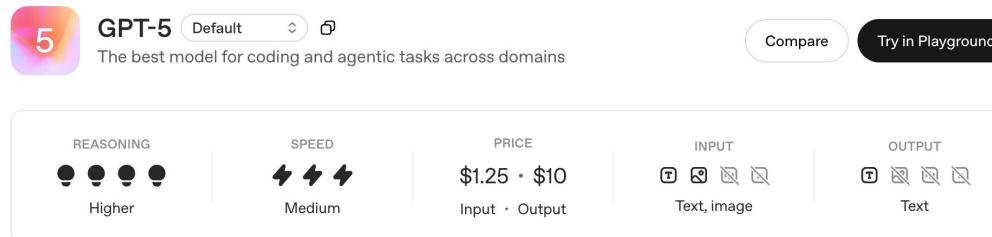
# Challenges

## Cost.

- (At least) millions of dollars
- Takes time
- Environment / electricity

## Learned knowledge.

- "Knowledge cutoff"
- Hard to edit knowledge
- "Plagiarism"



GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#).

- ❖ 400,000 context window
- ➡ 128,000 max output tokens
- 📅 Sep 30, 2024 knowledge cutoff
- ⌚ Reasoning token support

# Challenges

## Cost.

- (At least) millions of dollars
- Takes time
- Environment / electricity

## Learned knowledge.

- "Knowledge cutoff"
- Hard to edit knowledge
- "Plagiarism"

The screenshot shows the GPT-5 model page on the OpenAI platform. At the top, there's a large '5' icon, the text 'GPT-5 Default', a dropdown menu, and a refresh button. Below that, a subtext reads 'The best model for coding and agentic tasks across domains'. There are two buttons: 'Compare' and 'Try in Playground'.

Below this, there's a summary card with the following details:

REASONING	SPEED	PRICE	INPUT	OUTPUT
Higher	Medium	\$1.25 · \$10 Input · Output	Text, image	Text

At the bottom left, a paragraph states: 'GPT-5 is our flagship model for coding, reasoning, and agentic tasks across domains. Learn more in our [GPT-5 usage guide](#)'. On the right side, there's a list of features with some items highlighted by red boxes:

- ❖ 400,000 context window
- ➡ 128,000 max output tokens
- 📅 Sep 30, 2024 knowledge cutoff (highlighted)
- ⌚ Reasoning token support



# Transformers & Large Language Models

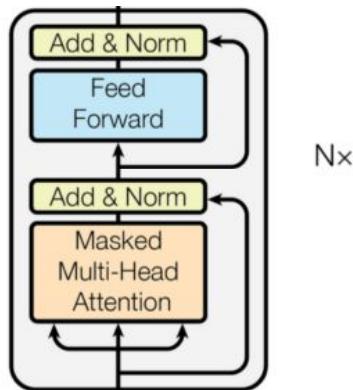
Pretraining

**Training optimizations**

Supervised finetuning

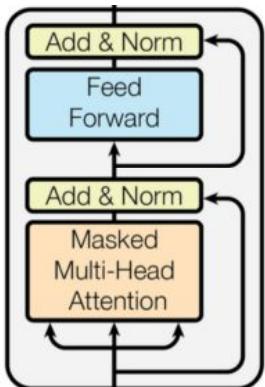
Parameter-efficient finetuning

# Setup of LLM training



$N \times$

# Setup of LLM training



For other uses, see [Teddy bear \(disambiguation\)](#).

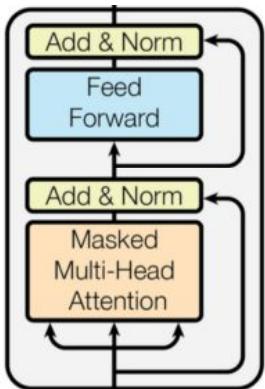
A **teddy bear**, or simply a **teddy**, is a **stuffed toy** in the form of a **bear**. The teddy bear was named by [Morris Michtom](#) after the 26th president of the United States, [Theodore Roosevelt](#); it was developed apparently simultaneously in the first decade of the 20th century by two toymakers: [Richard Steiff](#) in Germany and Michtom in the United States. It became a popular children's toy, and it has been celebrated in story, song, and film.<sup>[1]</sup>

Since the creation of the first teddy bears (which sought to imitate the form of real **bear cubs**), "teddies" have greatly varied in form, style, color, and material. They have become **collector's items**, with older and rarer teddies appearing at public auctions.<sup>[2]</sup> Teddy bears are among the most popular gifts for children, and they are often given to adults to signify affection, congratulations, or sympathy.

A screenshot of a GitHub repository page for "enzyne/enzyne". The repository has 92 forks and 209 stars. The "Code" tab is selected, showing the file structure and the content of "models.py". The code defines a class "ModelDefinitions" with an \_\_init\_\_ method. It includes imports for numpy and tensorflow, and defines authors and a license. A commit from "shervine" is shown, fixing the output model type.

```
1 """Model definitions."""
2
3 # Authors: Afshine Amidi <lastname@mit.edu>
4 #           Shervine Amidi <firstname@stanford.edu>
5
6 # MIT License
7
8 import numpy as np
9
```

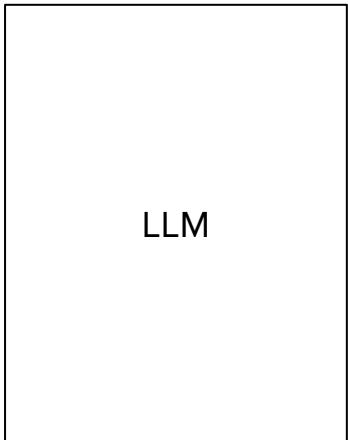
# Setup of LLM training



(Many) GPU(s)!

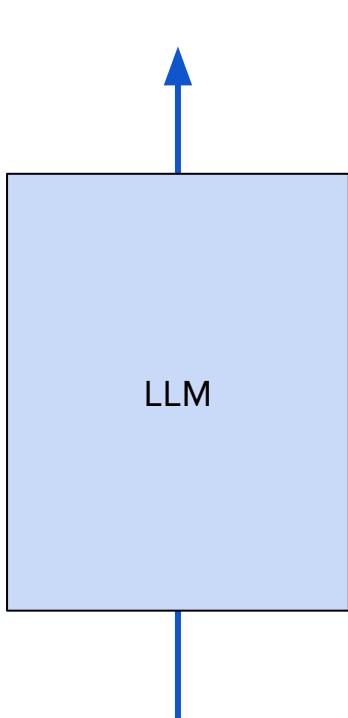
# Recap of LLM training

Initialization



**Model parameters.**  $O(\text{billions})$  -  $O(100s \text{ of billions})$

# Recap of LLM training



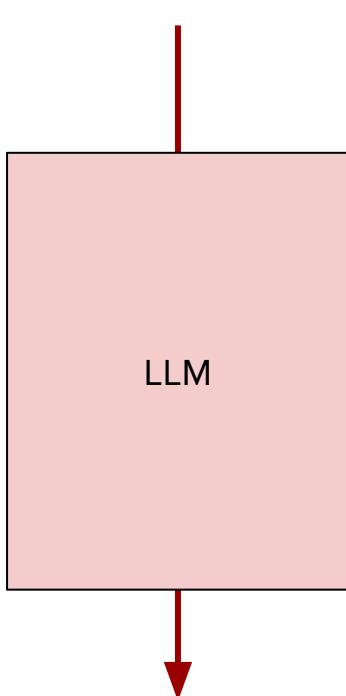
**Forward pass.** Compute loss.

**Activations.** Needed to compute the loss.

$$\mathcal{L}(\theta_t)$$

**Function of** model size, batch size, context length

# Recap of LLM training

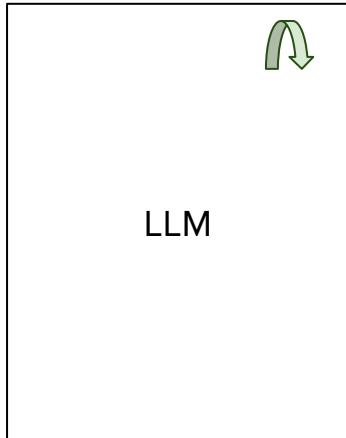


**Gradients.** Needed for weights update.

$$\nabla \mathcal{L}(\theta_t)$$

# Recap of LLM training

**Weights update.** Update model parameters.



**Optimizer state.** Updates the weights.

$$\theta_{t+1} \leftarrow \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Adam optimizer

$$\text{with: } m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1) \nabla \mathcal{L}(\theta_t)$$

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2) (\nabla \mathcal{L}(\theta_t))^2$$

"Super Study Guide: Transformers and Large Language Models", Amidi et al., 2024.

Suggested readings: "Decoupled Weight Decay Regularization", Loshchilov et al., 2017 ; "Adam: A Method for Stochastic Optimization", Kingma et al., 2014.

# Bottleneck: memory

## Technical Specifications

	H100 SXM	H100 NVL
<b>FP64</b>	34 teraFLOPS	30 teraFLOPS
<b>FP64 Tensor Core</b>	67 teraFLOPS	60 teraFLOPS
<b>FP32</b>	67 teraFLOPS	60 teraFLOPS
<b>TF32 Tensor Core*</b>	989 teraFLOPS	835 teraFLOPS
<b>BFLOAT16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP8 Tensor Core*</b>	3,958 teraFLOPS	3,341 teraFLOPS
<b>INT8 Tensor Core*</b>	3,958 TOPS	3,341 TOPS
<b>GPU Memory</b>	80GB	94GB
<b>GPU Memory Bandwidth</b>	3.35TB/s	3.9TB/s
<b>Decoders</b>	7 NVDEC 7 JPEG	7 NVDEC 7 JPEG
<b>Max Thermal Design Power (TDP)</b>	Up to 700W (configurable)	350-400W (configurable)
<b>Multi-Instance GPUs</b>	Up to 7 MIGs @ 10GB each	Up to 7 MIGs @ 12GB each
<b>Form Factor</b>	SXM	PCIe dual-slot air-cooled
<b>Interconnect</b>	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s
<b>Server Options</b>	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs  NVIDIA DGX H100 with 8 GPUs	Partner and NVIDIA-Certified Systems with 1-8 GPUs
<b>NVIDIA Enterprise</b>	Add-on	Included

# Bottleneck: memory

## Technical Specifications

	H100 SXM	H100 NVL
FP64	34 teraFLOPS	30 teraFLOPS
FP64 Tensor Core	67 teraFLOPS	60 teraFLOPS
FP32	67 teraFLOPS	60 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS	835 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS	3,341 teraFLOPS
INT8 Tensor Core*	3,958 TOPS	3,341 TOPS
<b>GPU Memory</b>	<b>80GB</b>	<b>94GB</b>
GPU Memory Bandwidth	3.35TB/s	3.9TB/s
Decoders	7 NVDEC 7 JPEG	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)	350-400W (configurable)
Multi-Instance GPUs	Up to 7 MIGs @ 10GB each	Up to 7 MIGs @ 12GB each
Form Factor	SXM	PCIe dual-slot air-cooled
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs  NVIDIA DGX H100 with 8 GPUs	Partner and NVIDIA-Certified Systems with 1–8 GPUs
NVIDIA Enterprise	Add-on	Included

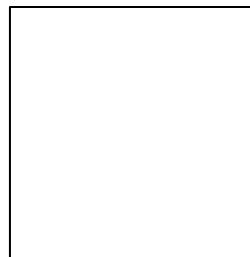
**Limited memory, O(10s of GB)**

# Data parallelism

## Idea.

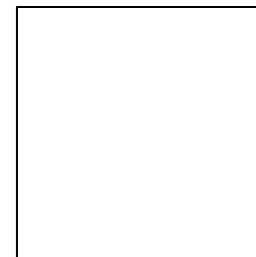
- Divide batch of data across devices
- Model replicated on each device

need communication b/w GPUs to aggregate the gradient in b/w the GPUs so training will be slower



GPU 1

...

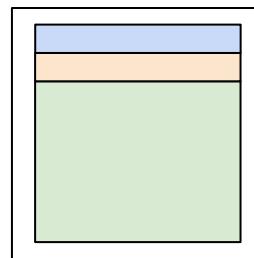


GPU n

# Data parallelism

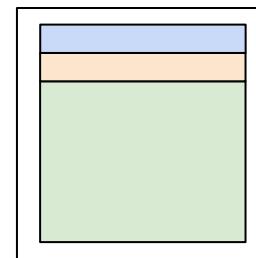
## Idea.

- Divide batch of data across devices
- Model replicated on each device



GPU 1

...



GPU n

parameters

gradients

optimizer state

# Data parallelism with ZeRO

**Idea.** Redundant information across devices

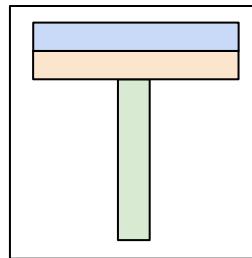
**ZeRO = Zero Redundancy Optimization**

# Data parallelism with ZeRO

**Idea.** Redundant information across devices

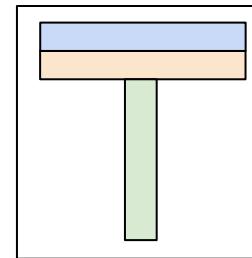
**ZeRO = Zero Redundancy Optimization**

**Variations.** ZeRO-1: Share **optimizer state**



GPU 1

...



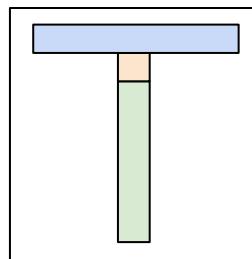
GPU n

# Data parallelism with ZeRO

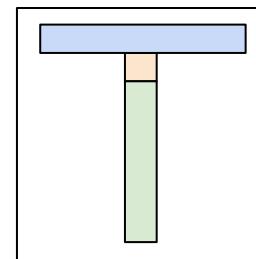
**Idea.** Redundant information across devices

**ZeRO = Zero Redundancy Optimization**

**Variations.** ZeRO-2: Share **optimizer state** + **gradients**



...



GPU 1

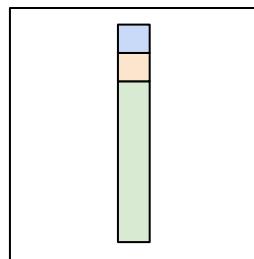
GPU n

# Data parallelism with ZeRO

**Idea.** Redundant information across devices

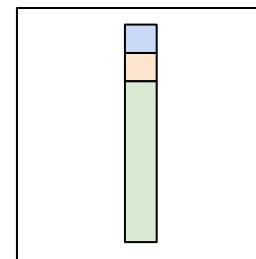
**ZeRO = Zero Redundancy Optimization**

**Variations.** ZeRO-3: Share **optimizer state** + **gradients** + **parameters**



GPU 1

...



GPU n

# Model parallelism

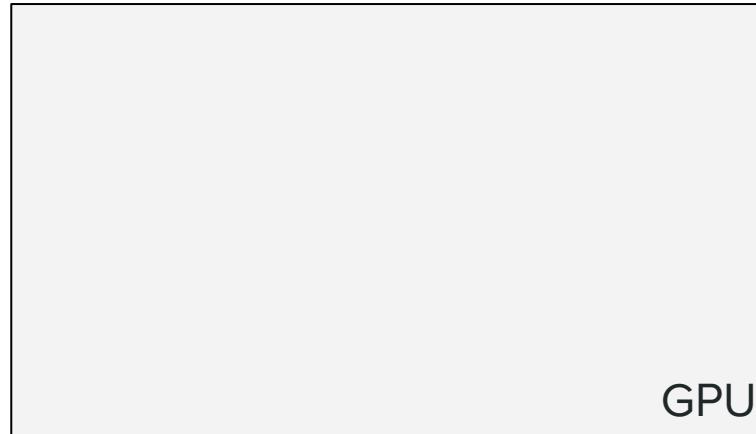
**Idea.** Split the model computations across several devices

## Variations.

- **Tensor Parallelism (TP).**
- **Pipeline Parallelism (PP).**
- **Sequence Parallelism (SP).**
- **Context Parallelism (CP).**
- **Expert Parallelism (EP).**

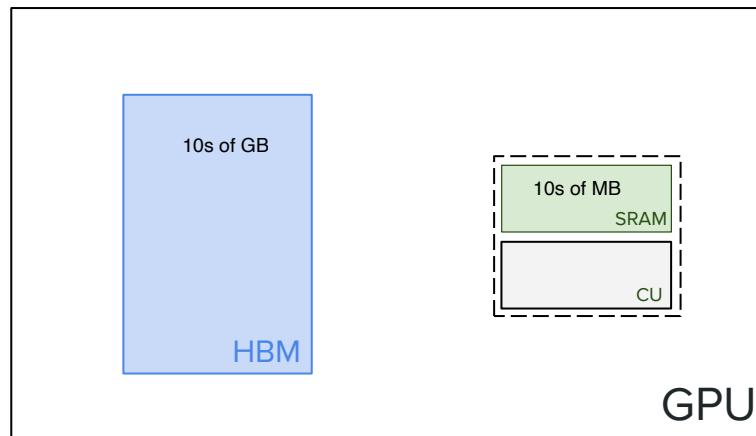
# Optimization via Flash Attention

**Strategy.** Leverage components of GPU to speed up attention computations.



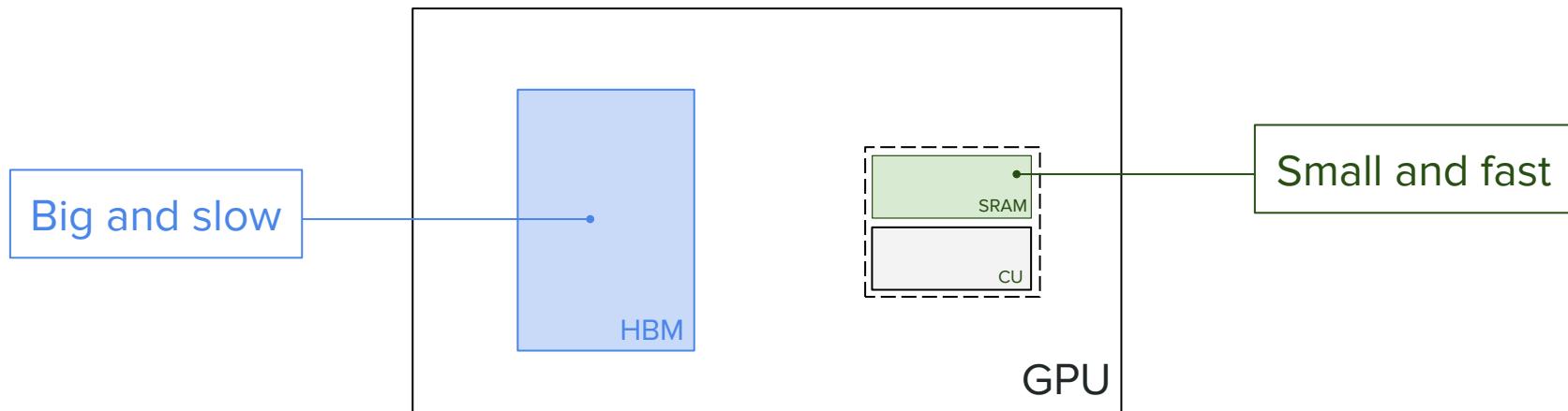
# Optimization via Flash Attention

**Strategy.** Leverage components of GPU to speed up attention computations.



# Optimization via Flash Attention

**Strategy.** Leverage components of GPU to speed up attention computations.



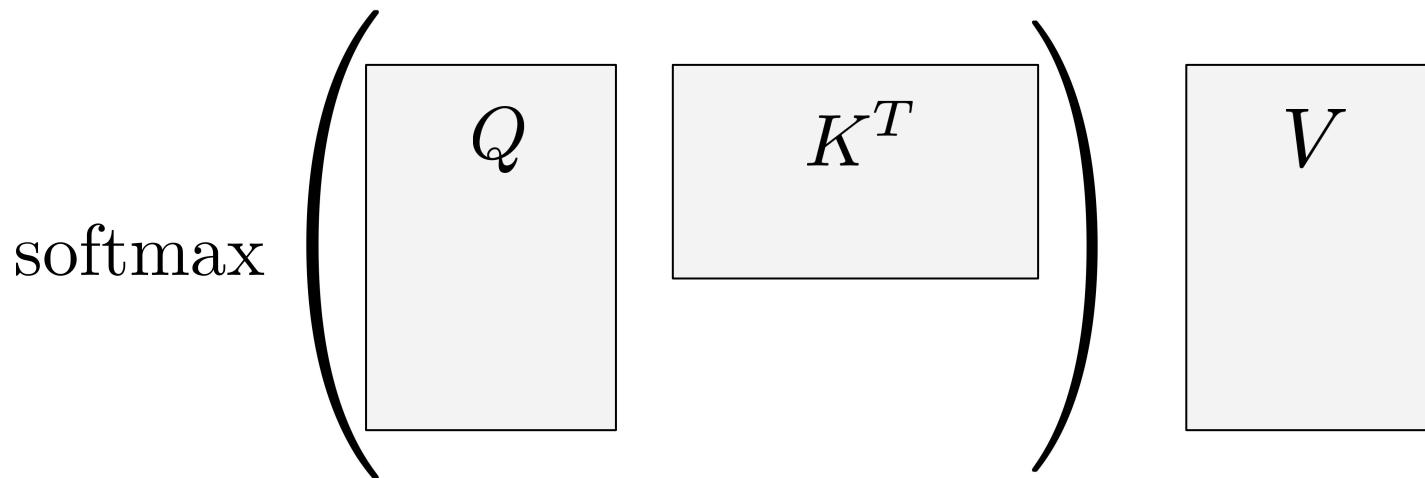
# Standard self-attention computation

**Context.** "Standard" self-attention computation

$$\text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V$$

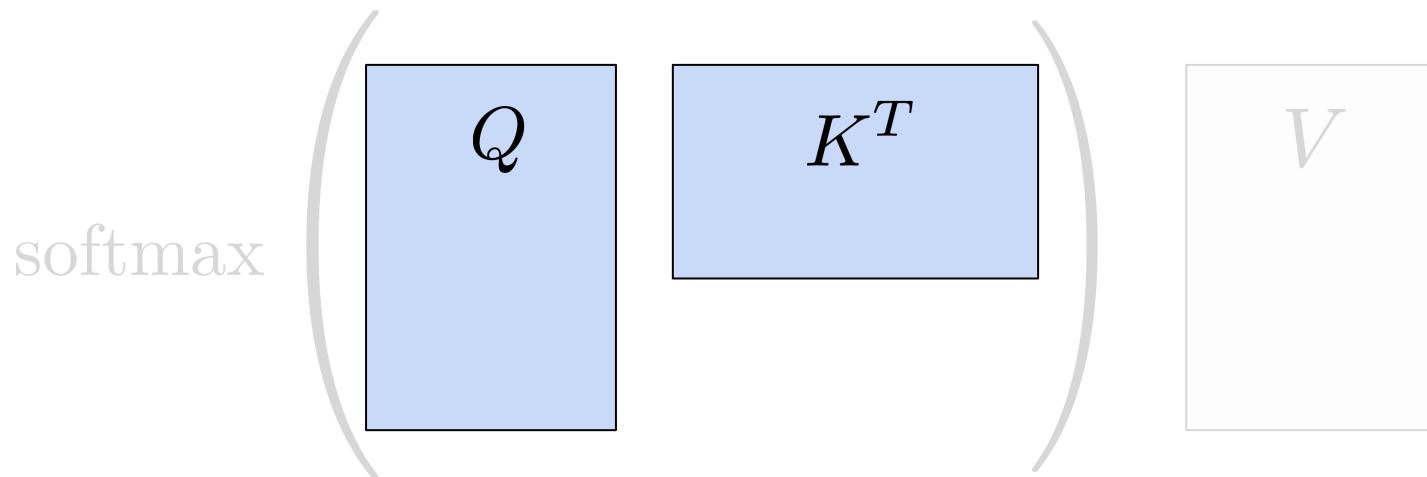
# Standard self-attention computation

**Context.** "Standard" self-attention computation



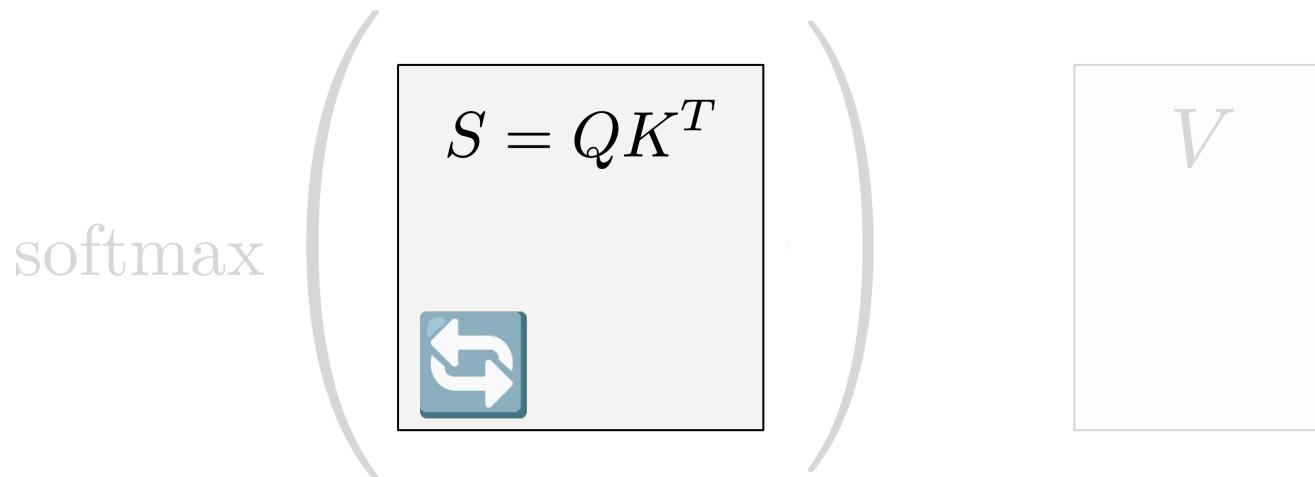
# Standard self-attention computation

**LOAD Q, K from **HBM** by blocks**



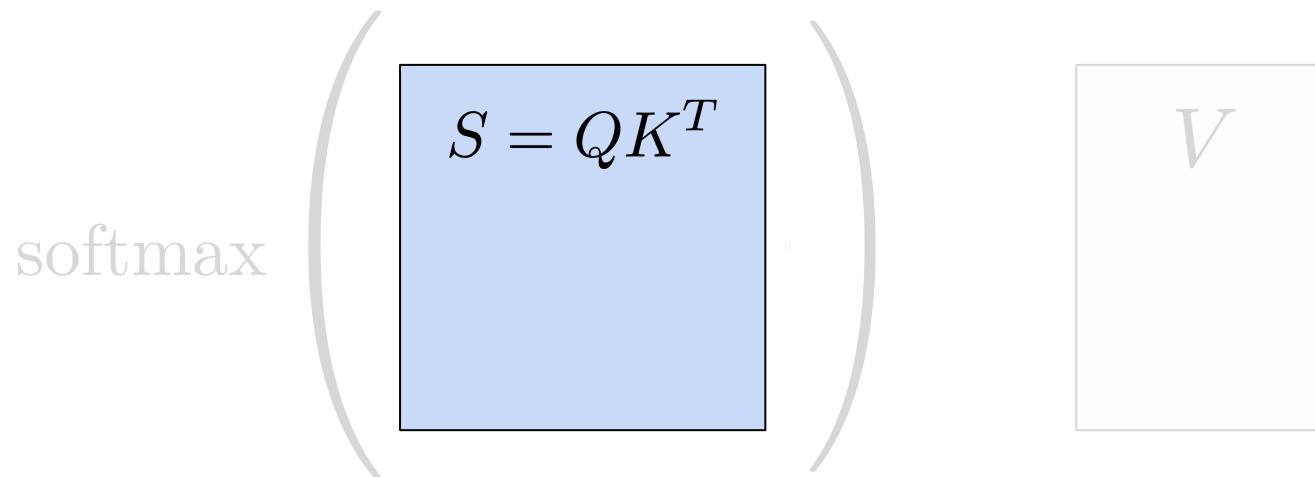
# Standard self-attention computation

Compute S



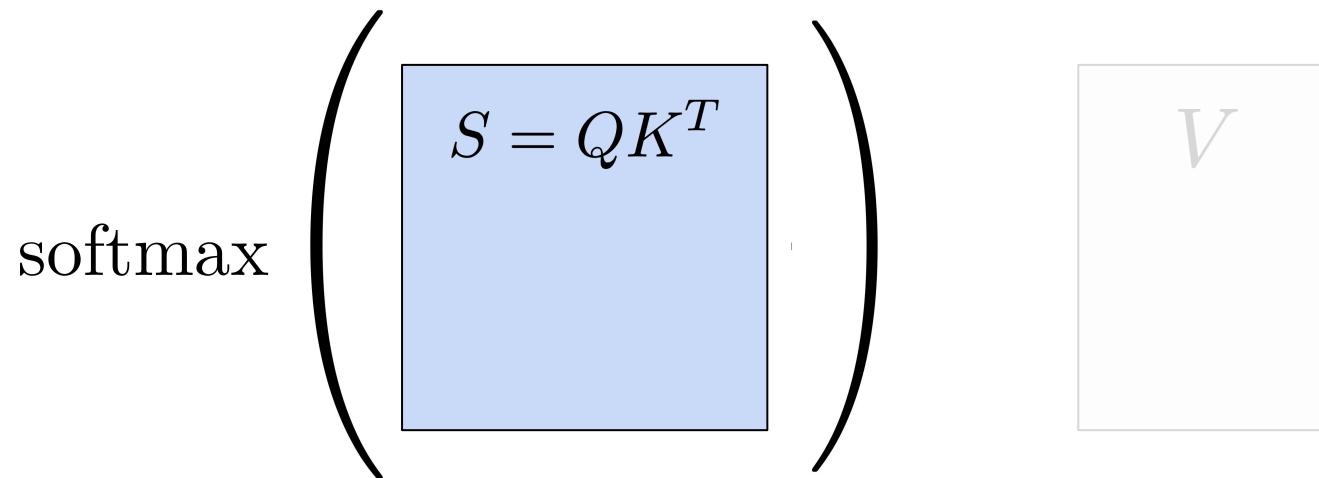
# Standard self-attention computation

**WRITE S to HBM**



# Standard self-attention computation

**READ S from HBM**



# Standard self-attention computation

Compute P

$$P = \text{softmax}(S)$$



V

# Standard self-attention computation

**WRITE P to HBM**

$$P = \text{softmax}(S)$$

*V*

# Standard self-attention computation

**LOAD** P, V from **HBM** by blocks

$$P = \text{softmax}(S)$$

$$V$$

# Standard self-attention computation

Compute O

$$O = PV$$

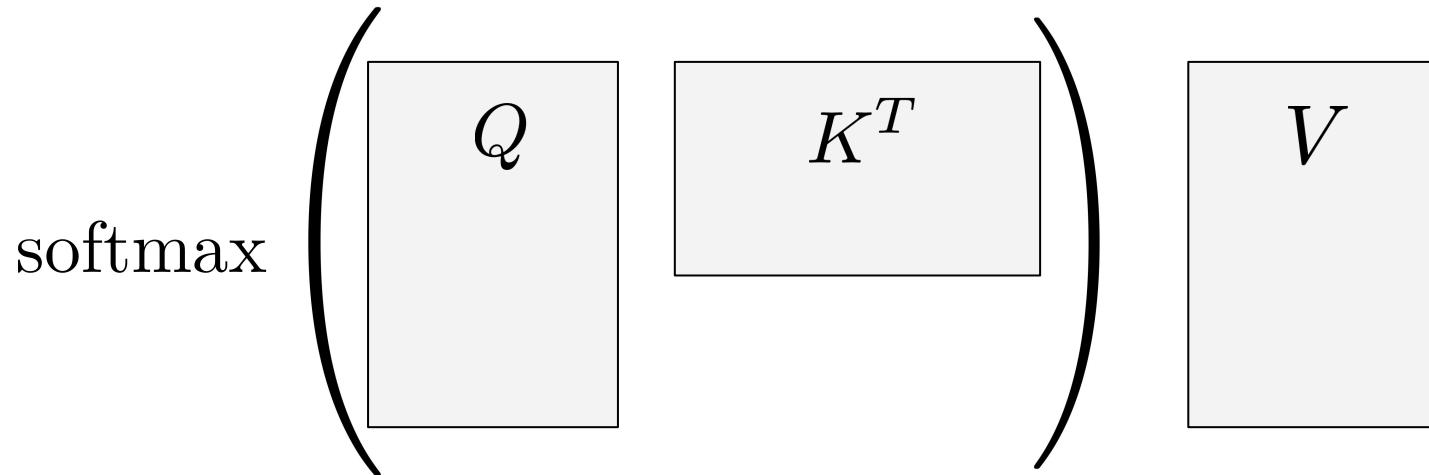

# Standard self-attention computation

**WRITE O to HBM**

$$O = PV$$

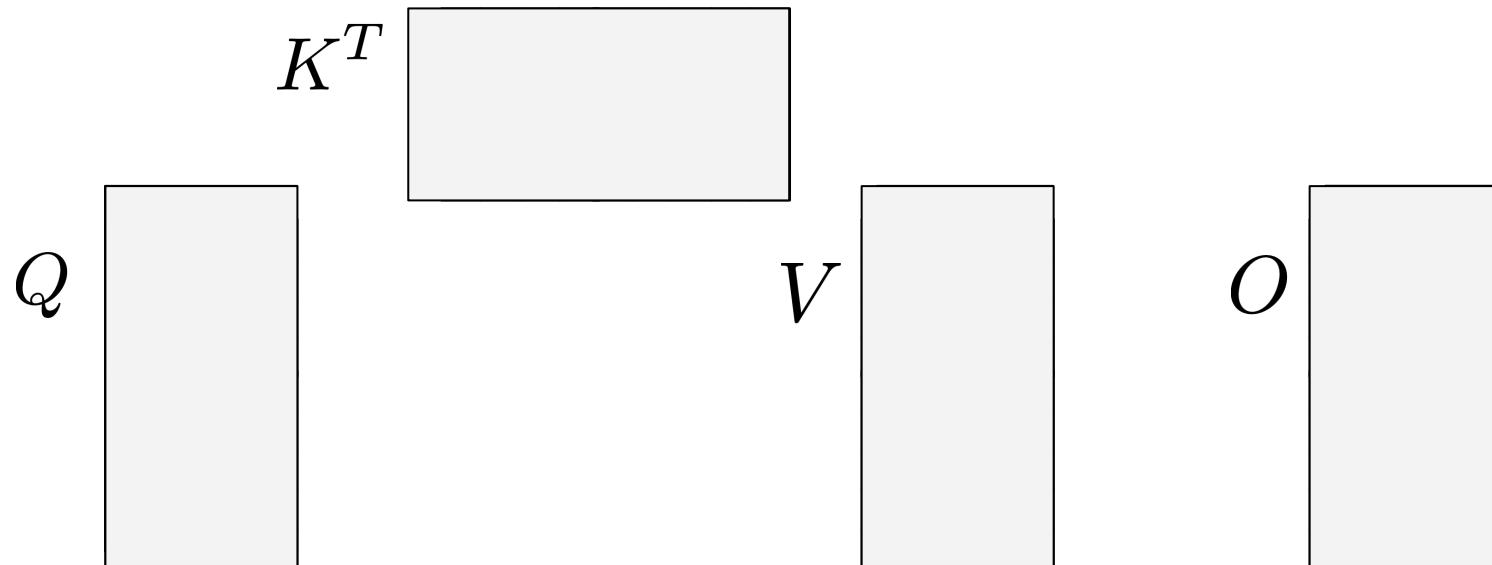
# Flash Attention in action

1<sup>st</sup> idea. Minimize read/writes to **HBM** with "tiling" via **SRAM**



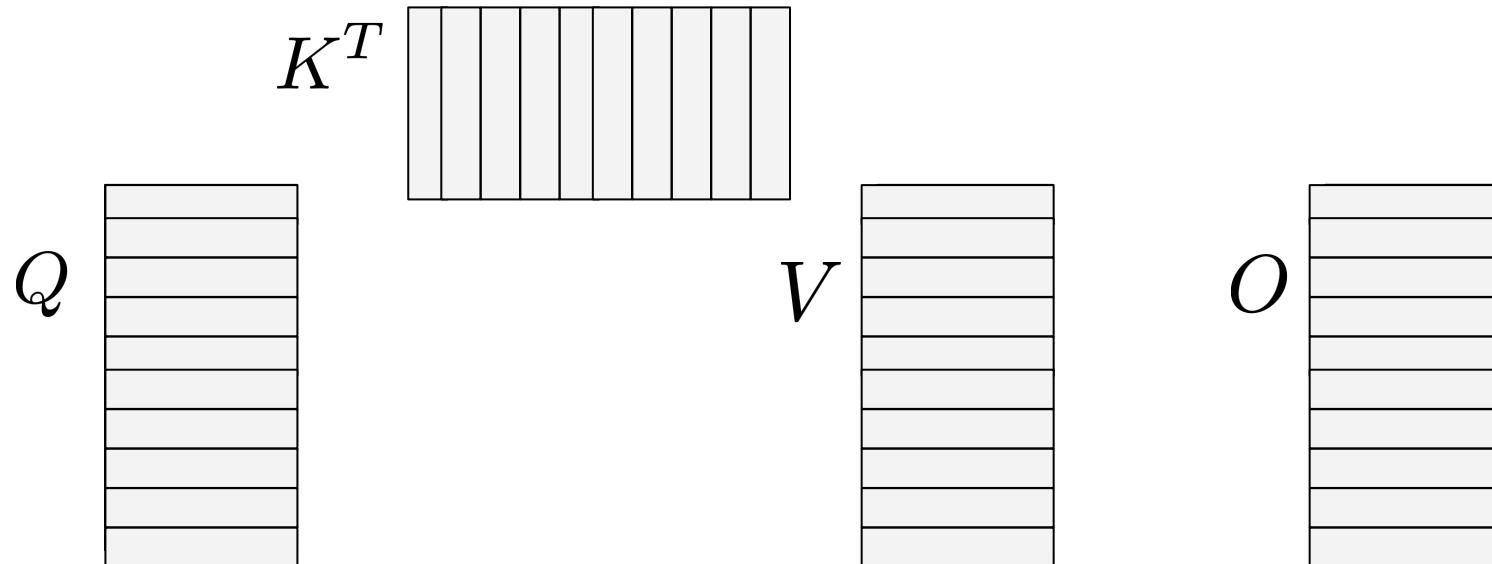
# Flash Attention in action

1<sup>st</sup> idea. Minimize read/writes to **HBM** with "tiling" via **SRAM**



# Flash Attention in action

1<sup>st</sup> idea. Minimize read/writes to **HBM** with "tiling" via **SRAM**



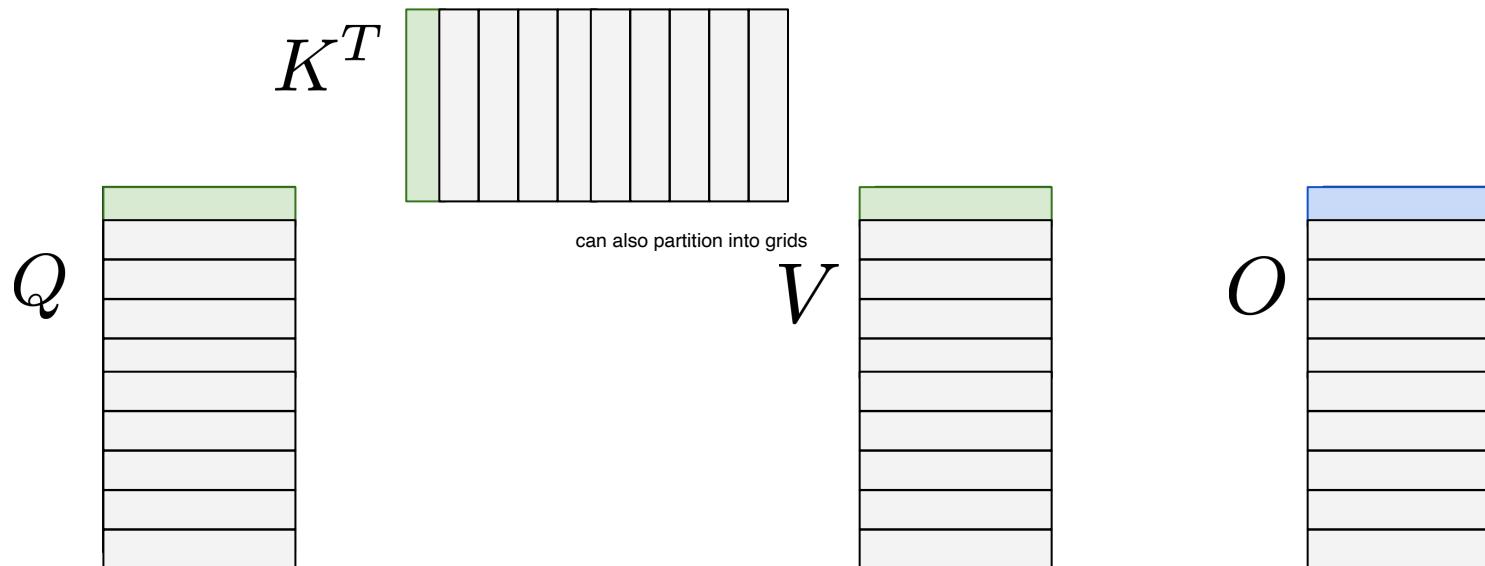
# Flash Attention in action

**Trick.** No need to compute the full  $S = QK^T$  before applying softmax.

$$\text{softmax} \left( \begin{array}{|c|c|c|} \hline S_1 & \dots & S_n \\ \hline \end{array} \right) = \begin{array}{|c|c|c|} \hline \alpha_1 \text{softmax}(S_1) & \dots & \alpha_n \text{softmax}(S_n) \\ \hline \end{array}$$

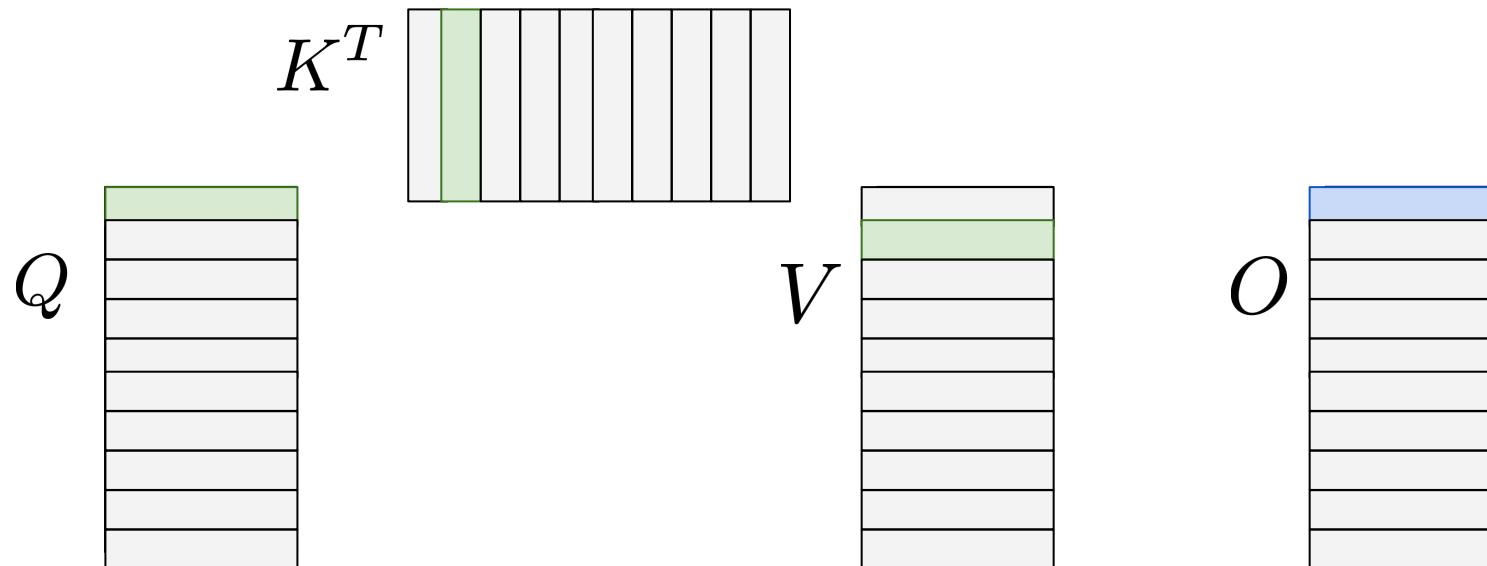
# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



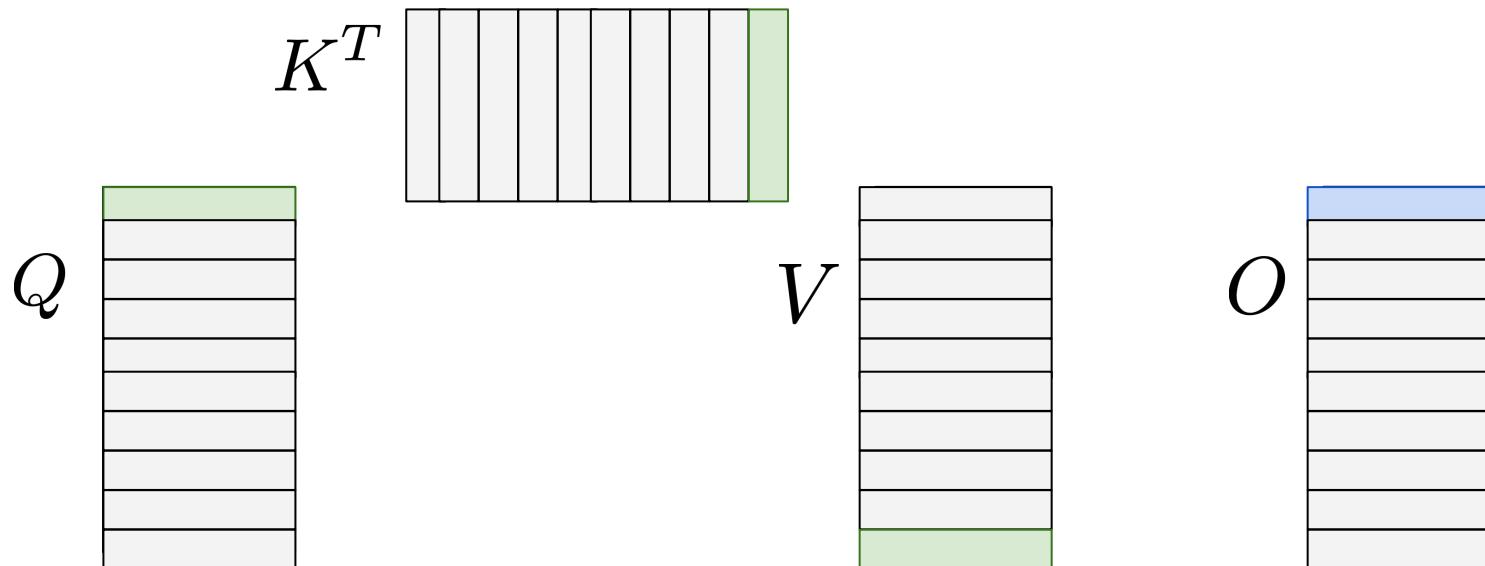
# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



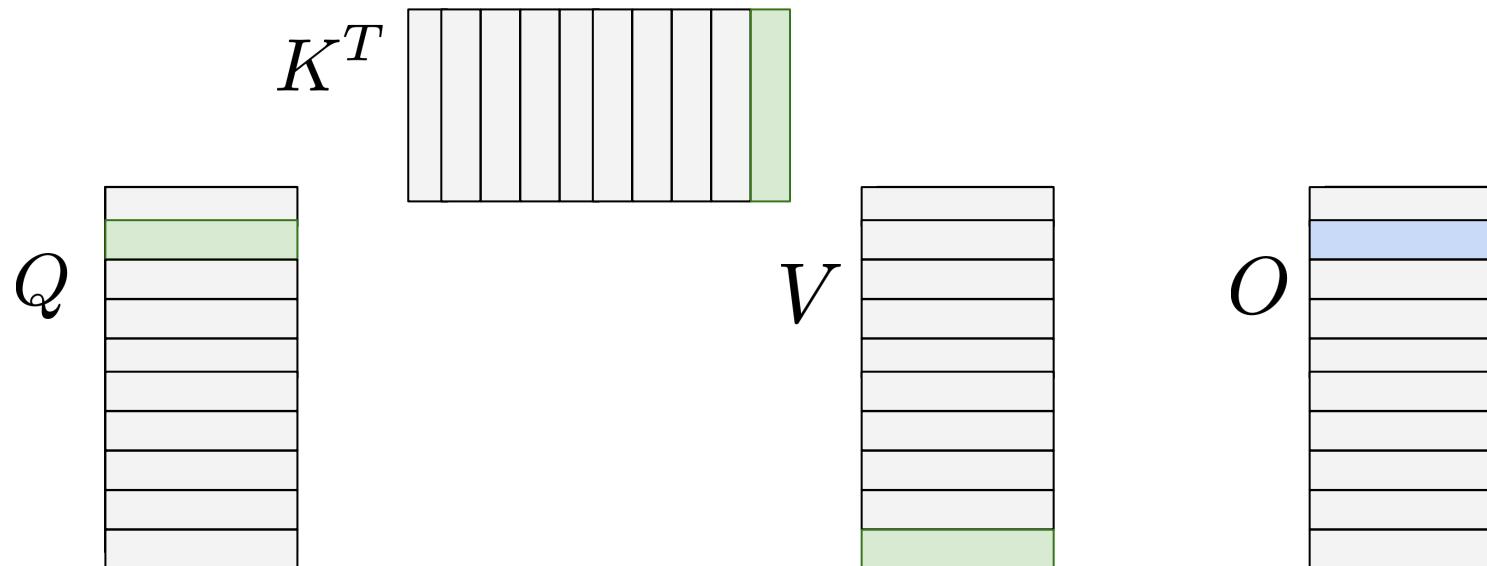
# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



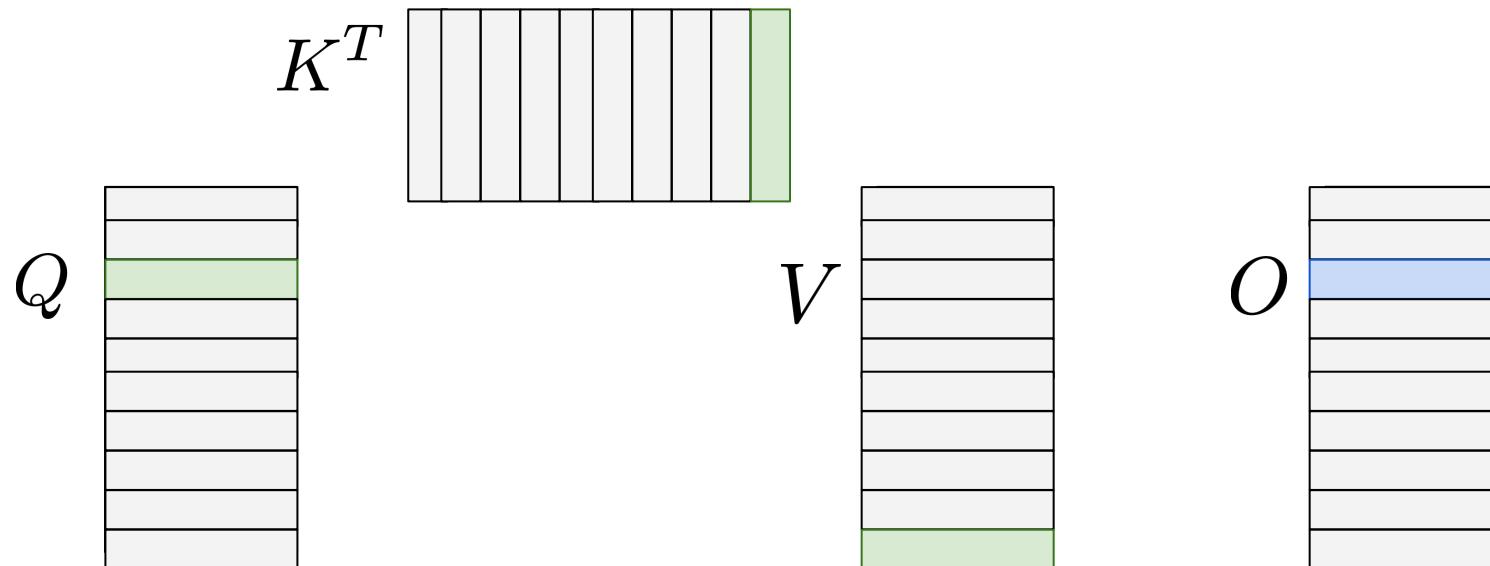
# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



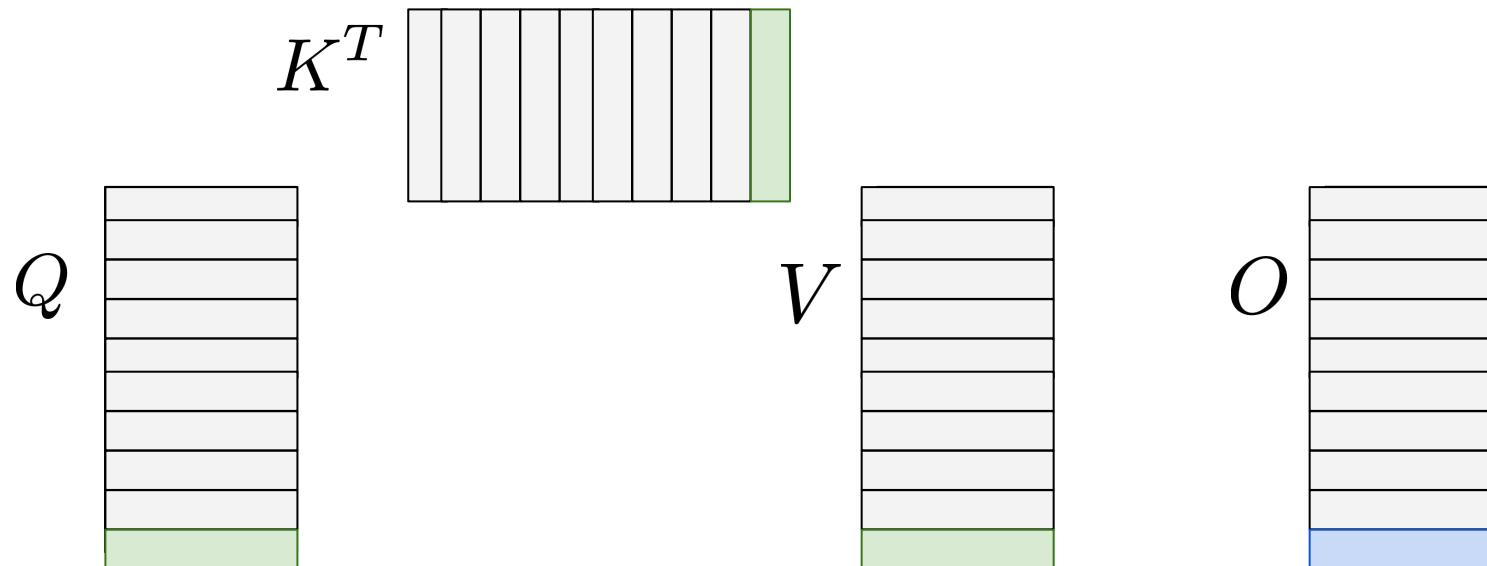
# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



# Flash Attention in action

Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



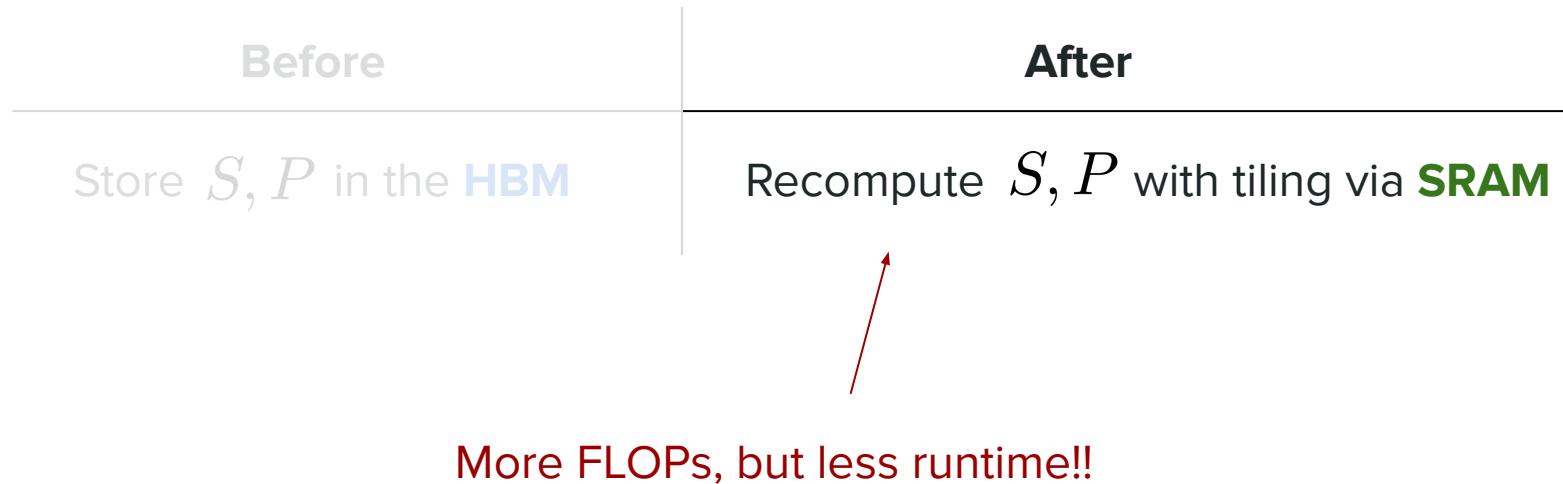
# Flash Attention in the backwards pass

**2<sup>nd</sup> idea.** Sometimes, it is better to **recompute** instead of storing

Before	After
Store $S, P$ in the <b>HBM</b>	Recompute $S, P$ with tiling via <b>SRAM</b> whenever needed in backward pass

# Flash Attention in the backwards pass

**2<sup>nd</sup> idea.** Sometimes, it is better to **recompute** instead of storing



# Flash Attention in action

**Results.** Flash Attention leads to a significant speedup with **exact** computation

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

# Precision of numbers

**1.2015432...**

**2.7015402...**

**2.4024402...**

**-0.7055120...**

**-1.7067140...**

quantisation: converting a  
precision of a num from  
one setting to other

**0.2741131...**

**-1.5312410...**

**0.4025222...**

# Precision of numbers

**1.2015432...**

**2.7015402...**

**2.4024402...**

**-0.7055120...**

**-1.7067140...**

**0.2741131...**

**-1.5312410...**

**0.4025222...**

# Representation of a float

Name	Description	Illustration
Sign	Controls whether the number is positive or negative. Typically takes up to 1 bit.	<p>0 ↔ +1 1 ↔ -1</p>
Exponent	Controls the magnitude of the number. Also called <i>range</i> .	<p><math>N_e = 8</math> 0 1 1 1 1 0 1 0 ↔ <math>\frac{2^{122}}{2^{127}}</math></p>
Mantissa	Controls the granularity of the number, i.e. what is after the decimal point. Also called <i>significand</i> or <i>fraction</i> .	<p>1 1 0 0 0 ... ↔ 1.75</p>

# Representation of a float

	<b>Sign</b>	<b>Exponent</b>	<b>Mantissa</b>
<b>FP16</b> (Floating-Point 16)	1	5	10
<b>FP32</b> (Floating-Point 32)	1	8	23
<b>FP64</b> (Floating-Point 64)	1	11	52
<b>BFLOAT16</b> (Brain Float 16)	1	8	7

# Example of a GPU

## Technical Specifications

	H100 SXM	H100 NVL
<b>FP64</b>	34 teraFLOPS	30 teraFLOPS
<b>FP64 Tensor Core</b>	67 teraFLOPS	60 teraFLOPS
<b>FP32</b>	67 teraFLOPS	60 teraFLOPS
<b>TF32 Tensor Core*</b>	989 teraFLOPS	835 teraFLOPS
<b>BFLOAT16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP8 Tensor Core*</b>	3,958 teraFLOPS	3,341 teraFLOPS
<b>INT8 Tensor Core*</b>	3,958 TOPS	3,341 TOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	3.35TB/s	3.9TB/s
Decoders	7 NVDEC 7 JPEG	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)	350-400W (configurable)
Multi-Instance GPUs	Up to 7 MIGs @ 10GB each	Up to 7 MIGs @ 12GB each
Form Factor	SXM	PCIe dual-slot air-cooled
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs  NVIDIA DGX H100 with 8 GPUs	Partner and NVIDIA-Certified Systems with 1-8 GPUs
NVIDIA Enterprise	Add-on	Included

# Example of a GPU

## Technical Specifications

	H100 SXM	H100 NVL
<b>FP64</b>	34 teraFLOPS	30 teraFLOPS
<b>FP64 Tensor Core</b>	67 teraFLOPS	60 teraFLOPS
<b>FP32</b>	67 teraFLOPS	60 teraFLOPS
<b>TF32 Tensor Core*</b>	989 teraFLOPS	835 teraFLOPS
<b>BFLOAT16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP16 Tensor Core*</b>	1,979 teraFLOPS	1,671 teraFLOPS
<b>FP8 Tensor Core*</b>	3,958 teraFLOPS	3,341 teraFLOPS
<b>INT8 Tensor Core*</b>	3,958 TOPS	3,341 TOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	3.35TB/s	3.9TB/s
Decoders	7 NVDEC 7 JPEG	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)	350-400W (configurable)
Multi-Instance GPUs	Up to 7 MIGs @ 10GB each	Up to 7 MIGs @ 12GB each
Form Factor	SXM	PCIe dual-slot air-cooled
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs  NVIDIA DGX H100 with 8 GPUs	Partner and NVIDIA-Certified Systems with 1-8 GPUs
NVIDIA Enterprise	Add-on	Included



**Lower precision → Faster processing**



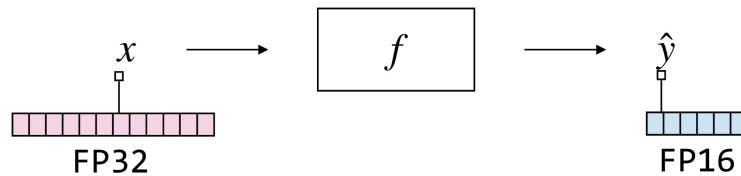
# Mixed precision training

**Objective.** Speed up training and decrease memory requirements

# Mixed precision training

**Objective.** Speed up training and decrease memory requirements

- **Forward pass.** Activations in low precision



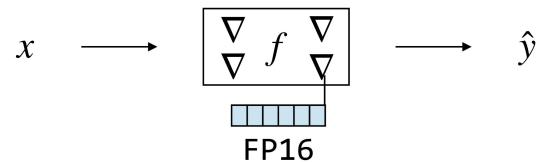
# Mixed precision training

**Objective.** Speed up training and decrease memory requirements

- **Forward pass.** Activations in low precision



- **Backwards pass.** Gradient updates in low precision



# Mixed precision training

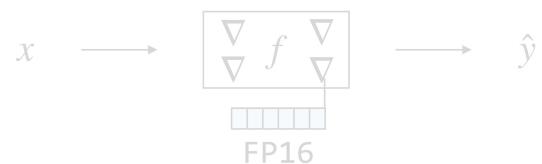
**Objective.** Speed up training and decrease memory requirements

- **Forward pass.** Activations in low precision

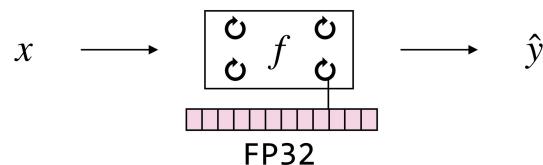


- **Backwards pass.** Gradient updates in low precision

not necessarily  
with all the layers



- **Weights update.** Keep weights in high precision





# Transformers & Large Language Models

Pretraining

Training optimizations

**Supervised finetuning**

Parameter-efficient finetuning

# What we covered in the first part of this lecture

Initialized model

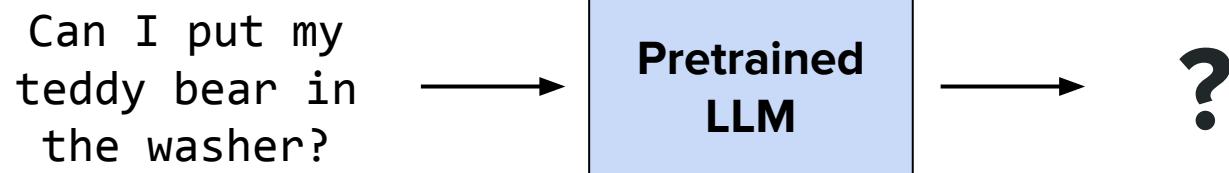


**Pretraining**

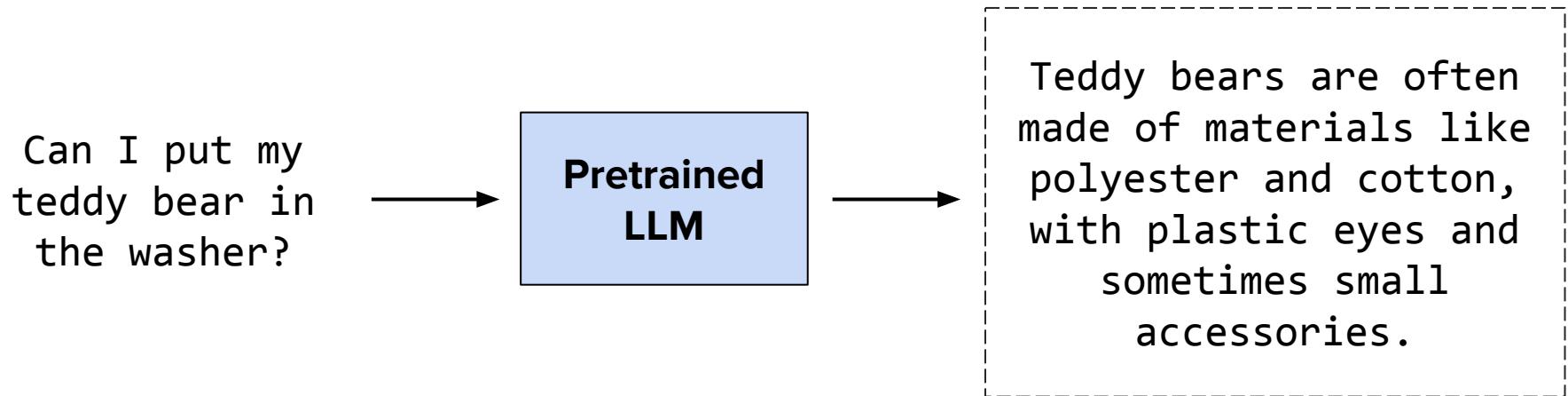


Model with "basic  
knowledge" about  
language, code, etc.

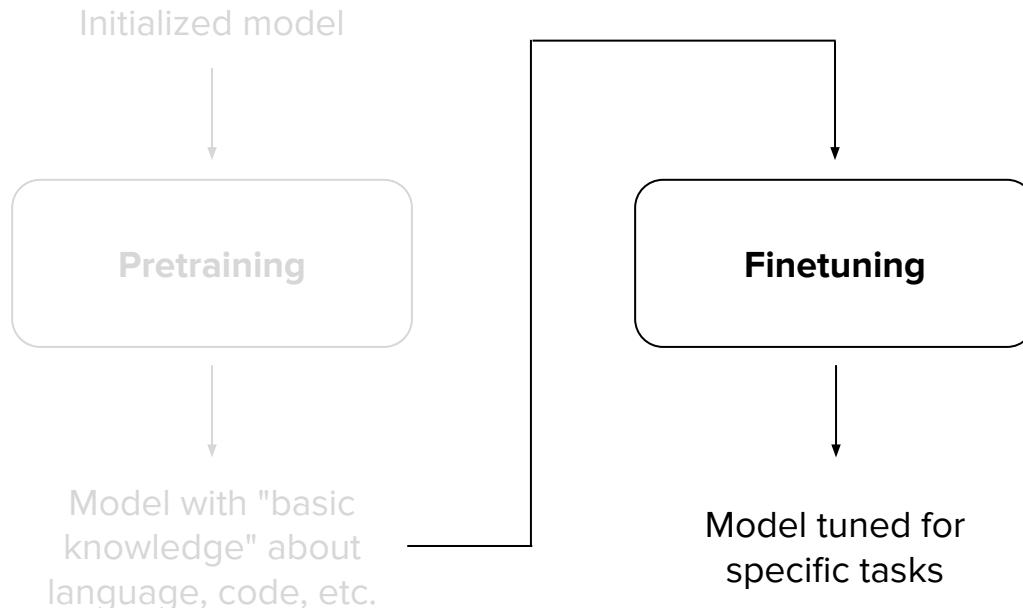
# Pretrained model behavior



# Pretrained model behavior



# Remedy



# Supervised finetuning

**SFT = Supervised FineTuning**

**Idea.** Change the model behavior by **tuning its weights**

**Strategy.**

- Collect pairs of input/outputs with desired behavior (aka SFT data)
- Train using next word prediction objective *given the input*

give the input and start loss calculation from the start of output generation

# Supervised finetuning

**SFT = Supervised FineTuning**

Idea. Change the model behavior by **tuning its weights**

**Strategy.**

- Collect pairs of input/outputs with desired behavior (aka SFT data)
- Train using next word prediction objective *given the input*

**Special case.** SFT on instruction-following data: "Instruction tuning"

# Instruction tuning overview

**Goal.** "Graduate" the model to being a helpful assistant.

# Instruction tuning overview

**Goal.** "Graduate" the model to being a helpful assistant.

## Story writing

Write a short story about a teddy bear who likes to read poetry.

Once upon a time, a bear, Teddy, stumbled upon verses from Attar. The bear became so captivated with the rhythm of the poem's sentences that he got filled with happiness.

## Poem creation

Create a poem about my cute teddy bear.

Soft and cuddly, full of charm,  
Always keeps me safe from harm,  
With button eyes and fur so neat,  
My teddy bear is oh so sweet.

## Lists generation

List three fun activities a teddy bear might do on a rainy day.

Sure!

1. Read poetry with friends.
2. Be cute.
3. Hug its owner tightly.

## Explanation

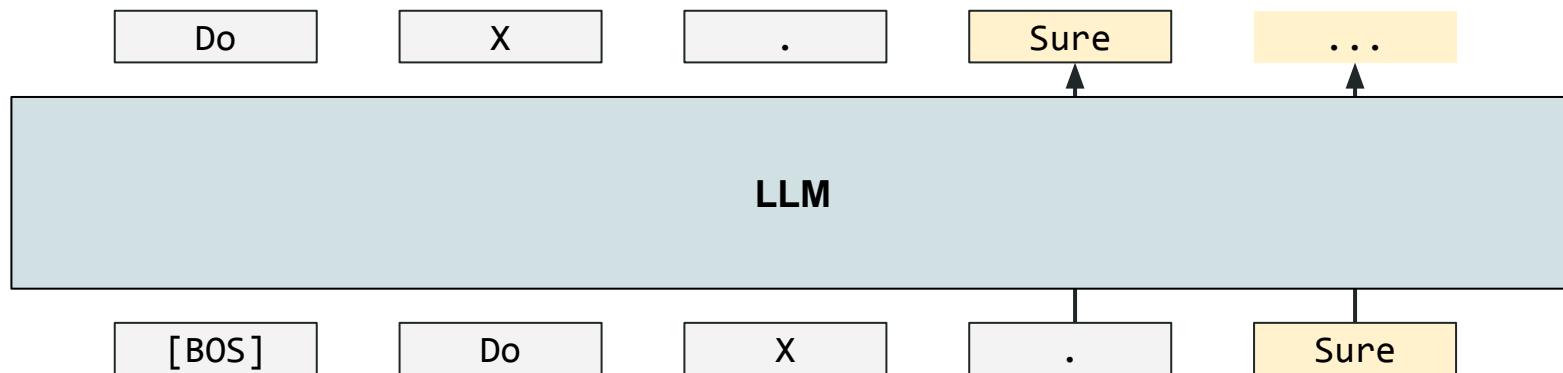
Explain why a teddy bear is a great friend.

A teddy bear is a great friend because it provides comfort and companionship. Its soft fur and cuddly nature is perfect for hugs. Teddy bears are always there in both happy and sad moments.

# Instruction tuning overview

**Goal.** "Graduate" the model to being a helpful assistant.

**Objective function.** Predict next token *given the input*.



# Instruction tuning overview

**Goal.** "Graduate" the model to being a helpful assistant

**Objective function.** Predict next token *given the input.*

**Data mixtures.** Can be both human-written and synthetic data.

- Assistant dialogs
- Synthetic instructions
- Maths, reasoning, code
- Safety alignment
- ...

hedging: nuance the output  
of the model to not  
make blanket statements

# Instruction tuning overview

**Goal.** "Graduate" the model to being a helpful assistant

**Objective function.** Predict next token *given the input.*

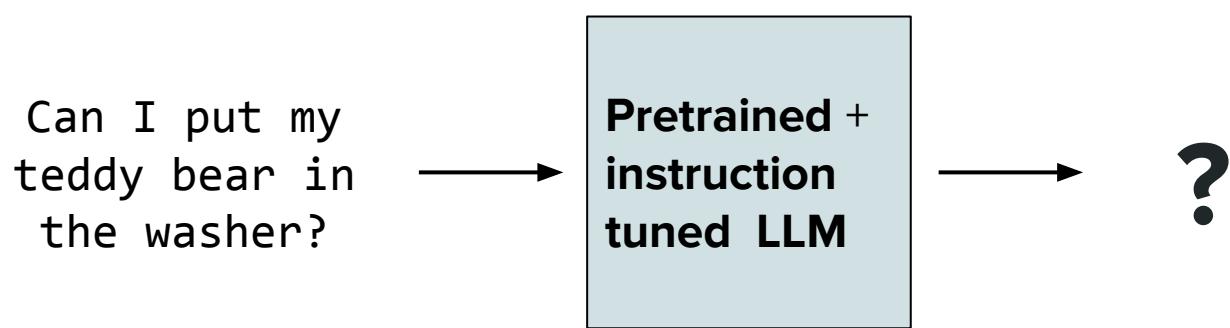
**Data mixtures.** Can be both human-written and synthetic data.

aligning the goal of the model to be suited for our task  
high quality dataset and more concise and precise

**Size.** Thousands to millions of examples.

Model	SFT size (# examples)
GPT-3	13 thousands
LLaMA 3	10 million

# Behavior



# Behavior



# Challenges

- Very high-quality data needed
- Sensitive to prompt distribution
- Generalization
- Difficult to evaluate
- Computationally expensive

# Benchmarks

## Dimensions.

- General knowledge: MMLU massive multitask language understanding
- Basic reasoning: ARC-Challenge
- Math reasoning: GSM8K
- Code generation: HumanEval

# Benchmarks

## Dimensions.

- General knowledge: MMLU
- Basic reasoning: ARC-Challenge
- Math reasoning: GSM8K
- Code generation: HumanEval

**Validity.** Recommended to train on the test task to compare across models.

# "Real-life" feeling

## Leaderboard Overview

See how leading models stack up across text, image, vision, and beyond. This page gives you a snapshot of each Arena, you can explore deeper insights in their dedicated tabs. Learn more about it [here](#).

Text		
Rank (UB)	Model	Score
1	G gemini-2.5-pro	1452
1	A\ claudie-sonnet-4-5-20250929-t...	1448
1	A\ claudie-opus-4-1-20250805-thi...	1448
2	chatgpt-4o-latest-20250326	1441
2	gpt-4.5-preview-2025-02-27	1441
2	gpt-5-high	1440
2	o3-2025-04-16	1440
2	A\ claudie-opus-4-1-20250805	1438
2	A\ claudie-sonnet-4-5-20250929	1437
3	qwen3-max-preview	1434

[View all](#)

WebDev		
Rank (UB)	Model	Score
1	GPT-5 (high)	1476
1	A\ Claude Opus 4.1 thinking-16k...	1472
1	A\ Claude Opus 4.1 (20250805)	1460
4	G Gemini-2.5-Pro	1402
4	DeepSeek-R1-0528	1394
4	Z GLM-4.6	1394
4	A\ Claude Sonnet 4.5	1386
5	A\ Claude Opus 4 (20250514)	1384
5	Z GLM-4.5	1380
6	Z GLM-4.5-Air	1368

[View all](#)

Vision		
Rank (UB)	Model	Score
1	G gemini-2.5-pro	1241
1	chatgpt-4o-latest-20250326	1234

Text-to-Image		
Rank (UB)	Model	Score
1	hunyuan-image-3.0	1161
1	G gemini-2.5-flash-image-preview...	1154

**Idea.** Websites like "Chatbot Arena" run A/B tests for user prompts.

# "Real-life" feeling

**Benefits.** Puts a number on "vibes"

# "Real-life" feeling

**Benefits.** Puts a number on "vibes"

## **Outstanding challenges.**

- Unequal exposure of models/"cold start" problem
- Easy to "rig"
- User inability to accurately assess important aspects, e.g. factuality
- Personal preference bias (non-representative distribution)
- Safety penalization

# "Real-life" feeling

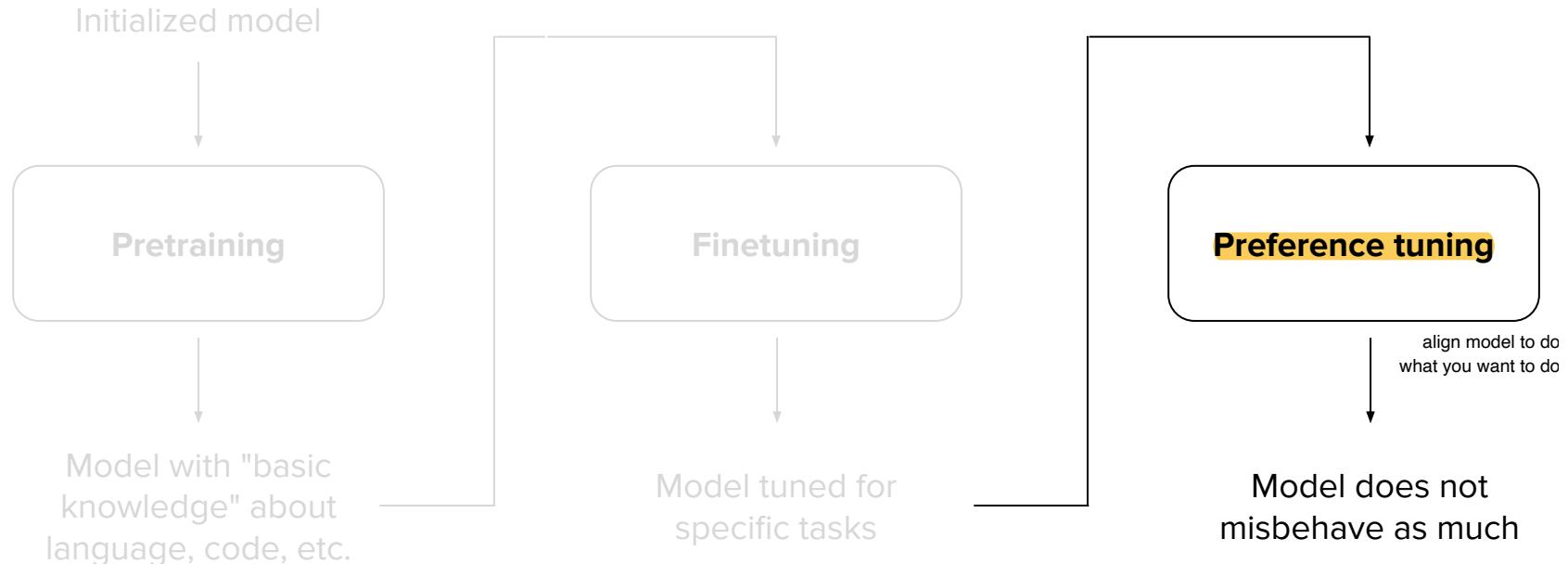
**Benefits.** Puts a number on "vibes"

**Outstanding challenges.**

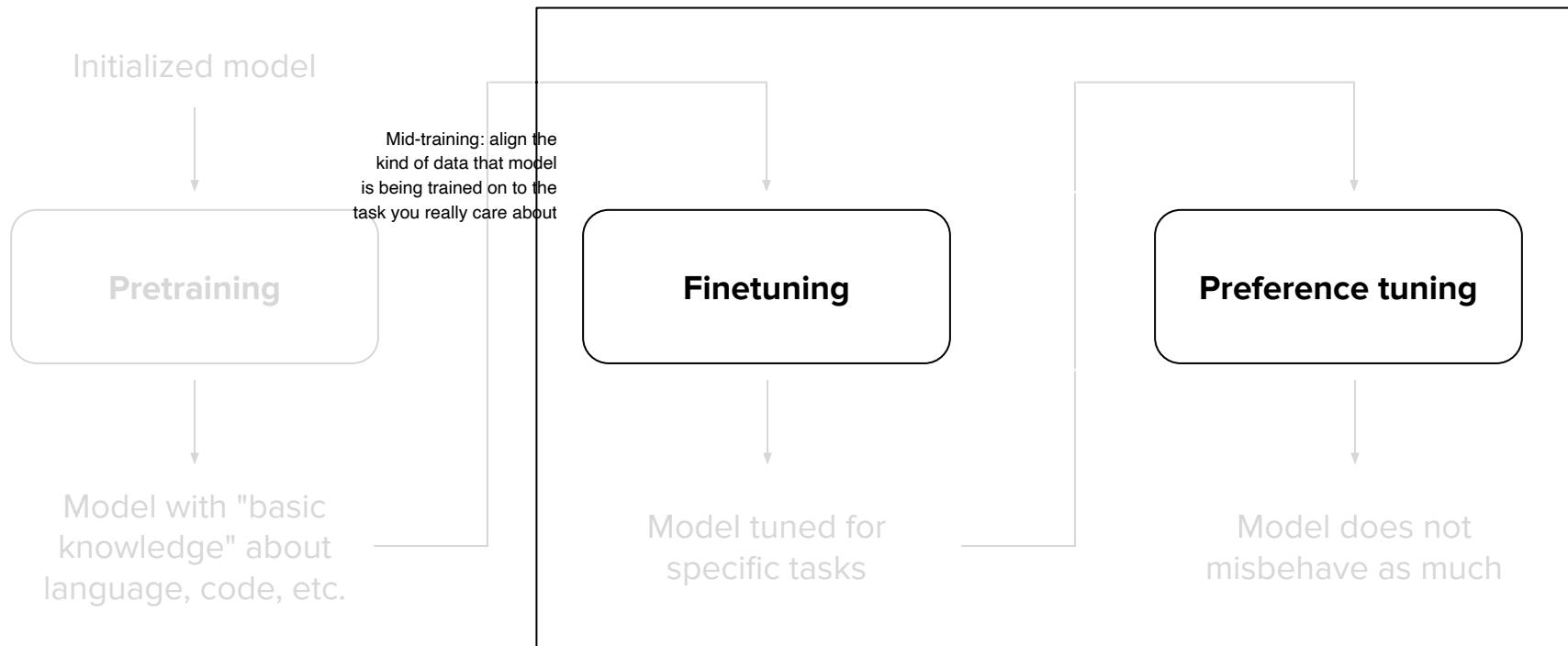
- Unequal exposure of models/"cold start" problem
- Easy to "rig"
- User inability to accurately assess important aspects, e.g. factuality
- Personal preference bias (non-representative distribution)
- Safety penalization

...evaluation is a hard problem in itself!

# What we'll see in Lecture 5



# Summary of lifecycle of an LLM



**"Alignment"** of the model



# Transformers & Large Language Models

Pretraining

Training optimizations

Supervised finetuning

**Parameter-efficient finetuning**

# Parameter-efficient finetuning with LoRA

**Context.** SFT is resource intensive and not everyone has big GPUs.

**Idea.** Low-Rank Adaptation (LoRA) approximates matrix with product of two low-rank matrices

$$W = W_0 + B \times A$$

dim(B) = R  
very small  
 $O(10)$   
4 common



# Parameter-efficient finetuning with LoRA

**Context.** SFT is resource intensive and not everyone has big GPUs.

**Idea.** Low-Rank Adaptation (LoRA) approximates matrix with product of two low-rank matrices

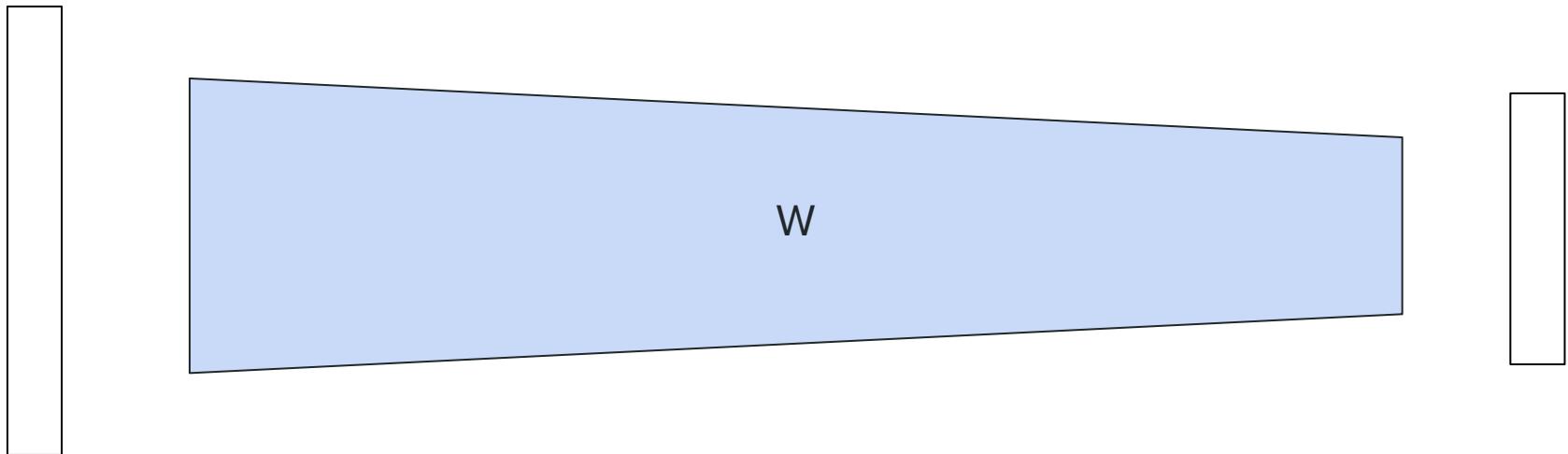
$$\begin{matrix} \text{W} \\ \hline \end{matrix} = \begin{matrix} \text{W}_0 \\ \hline \end{matrix} + \begin{matrix} \text{B} \\ \hline \end{matrix} \times \begin{matrix} \text{A} \\ \hline \end{matrix}$$

**Discussion.**

- Fraction of parameters need to be trained with similar performance
- Other methods include **prefix tuning and adapters**

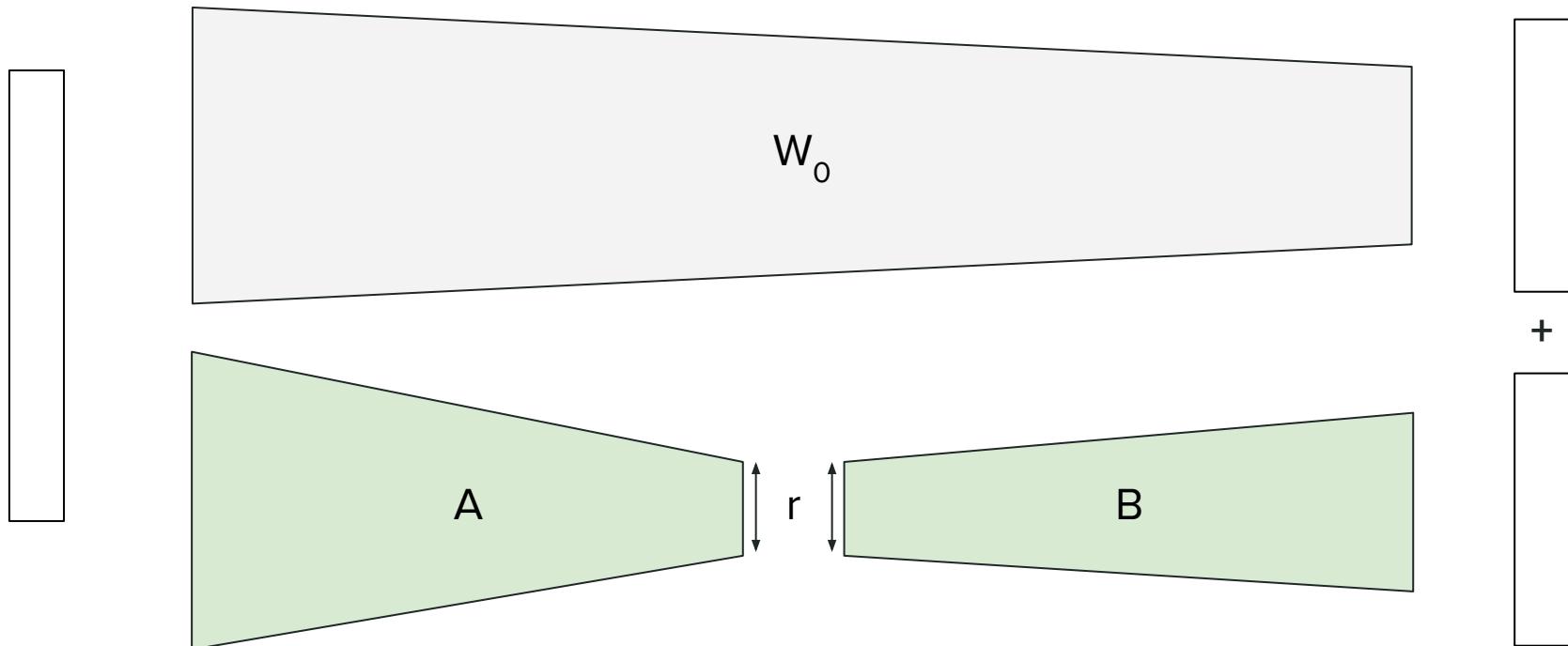
# Finetuning evolution

**Before.** "Regular" finetuning optimizes the full matrix of weights



# Finetuning evolution

**After.** LoRA finetuning optimizes the full matrix of weights



# Benefit of LoRA: swap matrices = swap tasks

$$W_0 + B_{\text{spam}} \times A_{\text{spam}} \longrightarrow \text{Spam detection task}$$

# Benefit of LoRA: swap matrices = swap tasks

$$W_0 + B_{\text{spam}} \times A_{\text{spam}} \longrightarrow \text{Spam detection task}$$

$$W_0 + B_{\text{sentiment}} \times A_{\text{sentiment}} \longrightarrow \text{Sentiment extraction task}$$

# Benefit of LoRA: swap matrices = swap tasks

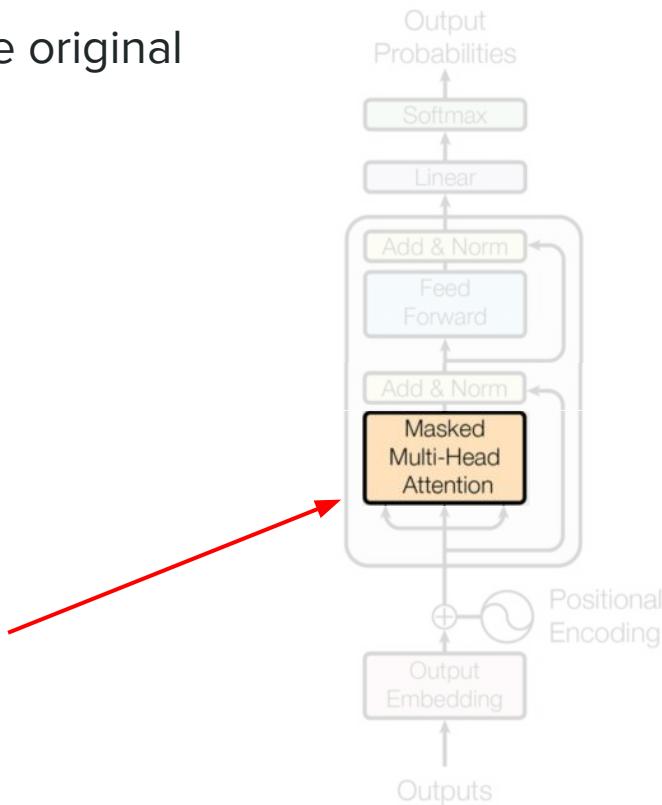
$$W_0 + B_{\text{spam}} \times A_{\text{spam}} \longrightarrow \text{Spam detection task}$$

$$W_0 + B_{\text{sentiment}} \times A_{\text{sentiment}} \longrightarrow \text{Sentiment extraction task}$$

$$W_0 + B_{\text{translation}} \times A_{\text{translation}} \longrightarrow \text{Translation task}$$

# Locations where to apply LoRA

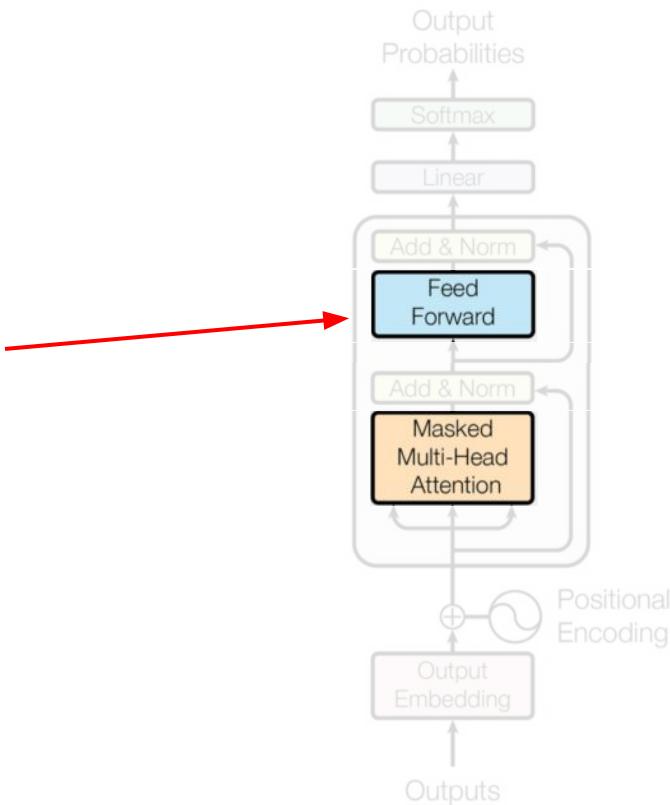
**Originally.** As experimented in the original paper.



# Locations where to apply LoRA

**Updated guidance.** As done and recommended nowadays.

most important location



"LoRA Without Regret", Schulman et al., 2025.

Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

# Training dynamics

**"Fun" facts.** Beware of training differences:

- LoRA needs a higher learning rate than full fine-tuning  $^{10x}$
- LoRA does poorly on large batch size compared to full fine-tuning

# Training dynamics

**"Fun" facts.** Beware of training differences:

- LoRA needs a higher learning rate than full fine-tuning

10x

can be explained by the  
rank of lora matrices

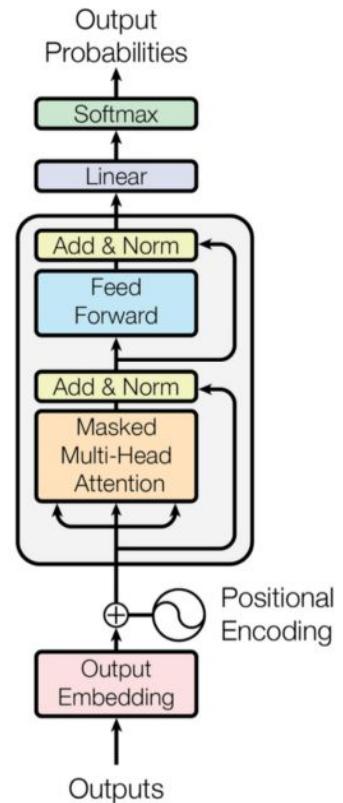
empirical

- LoRA does poorly on large batch size compared to full fine-tuning

empirical

# QLoRA

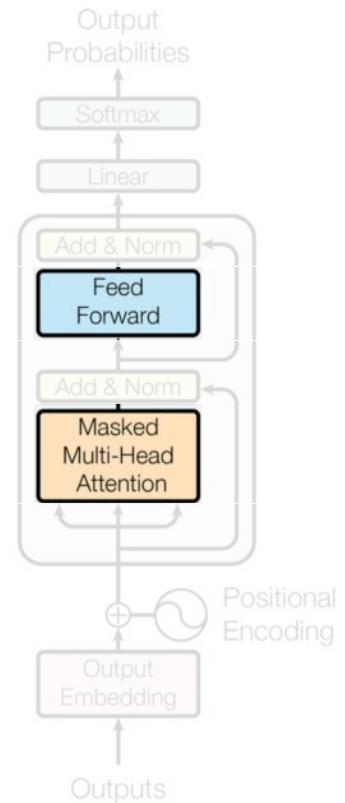
Idea. Quantize all frozen weights to relieve memory bottleneck.



# QLoRA

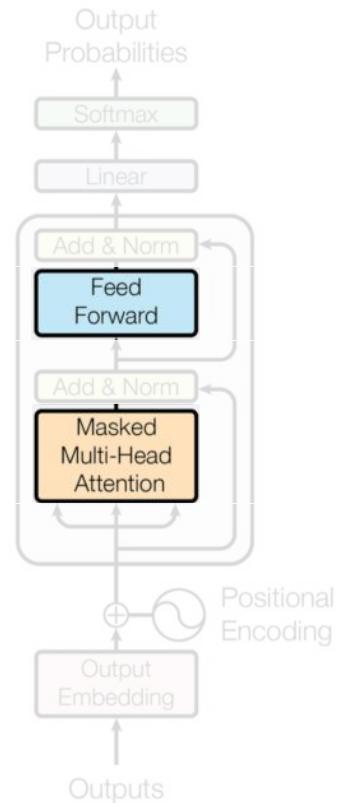
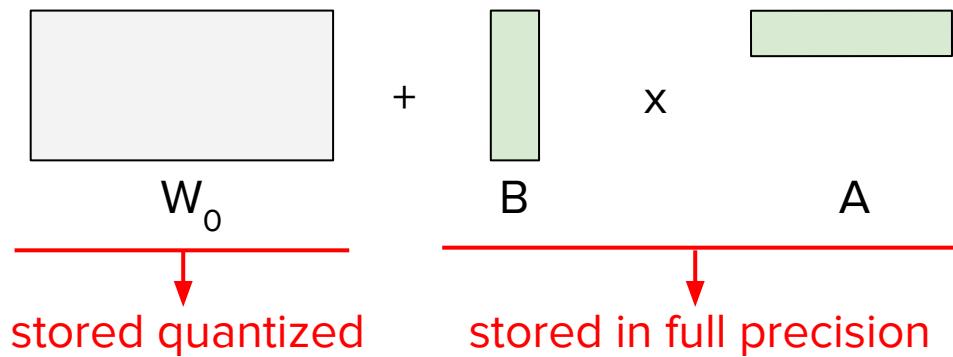
**Idea.** Quantize all frozen weights to relieve memory bottleneck.

$$W_0 + B \times A$$



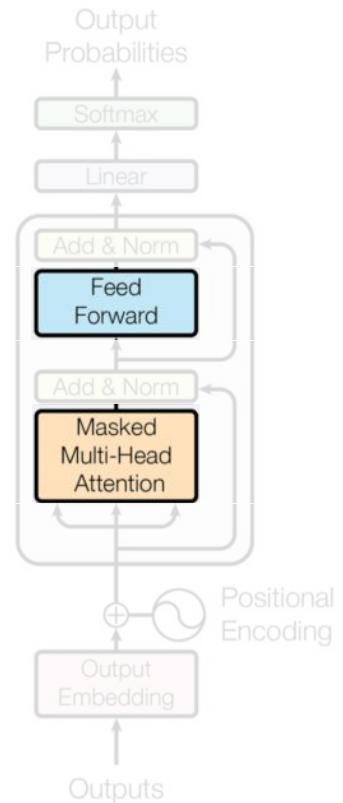
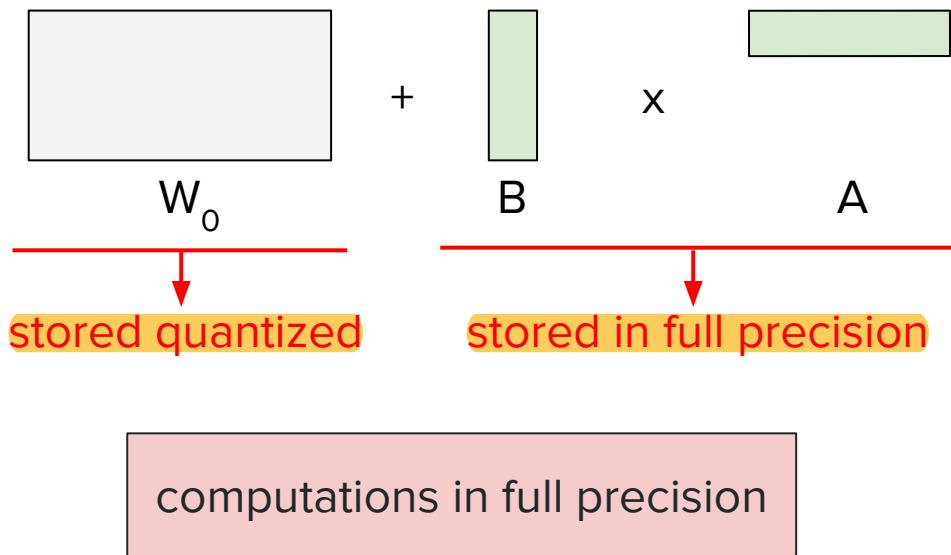
# QLoRA

**Idea.** Quantize all frozen weights to relieve memory bottleneck.



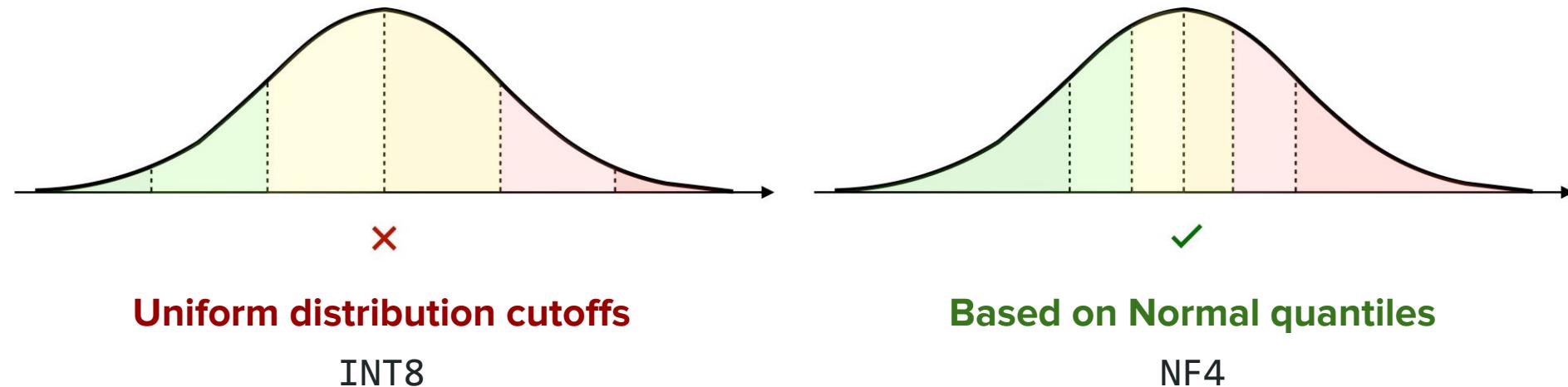
# QLoRA

**Idea.** Quantize all frozen weights to relieve memory bottleneck.



# Efficient quantization

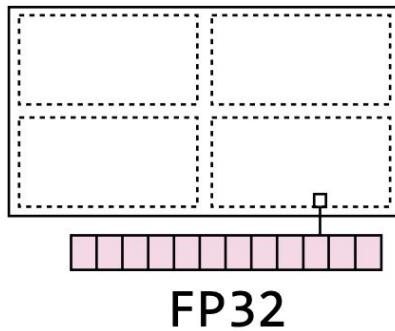
**Trick.** Use 4-bit NormalFloat (NF4) to best split the space



# Quantization...

No quantization

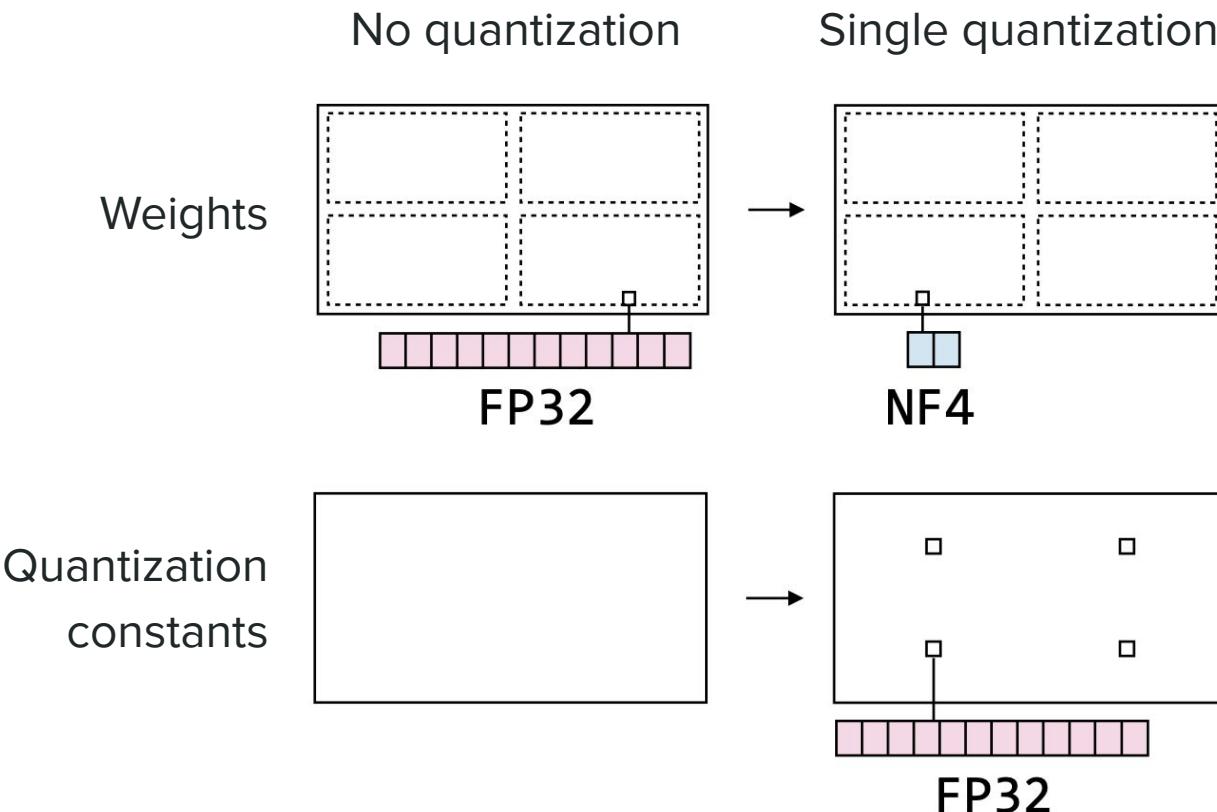
Weights



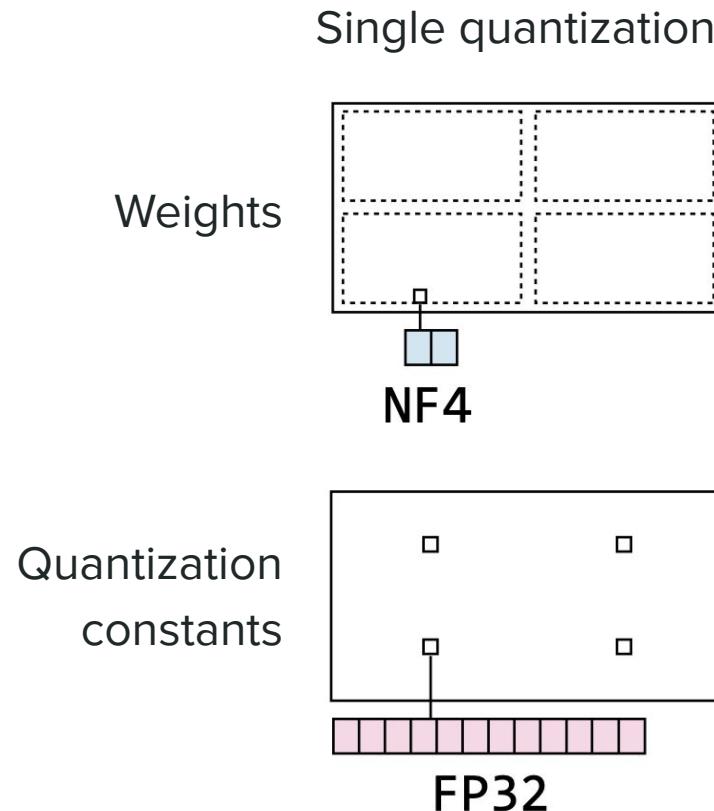
Quantization  
constants



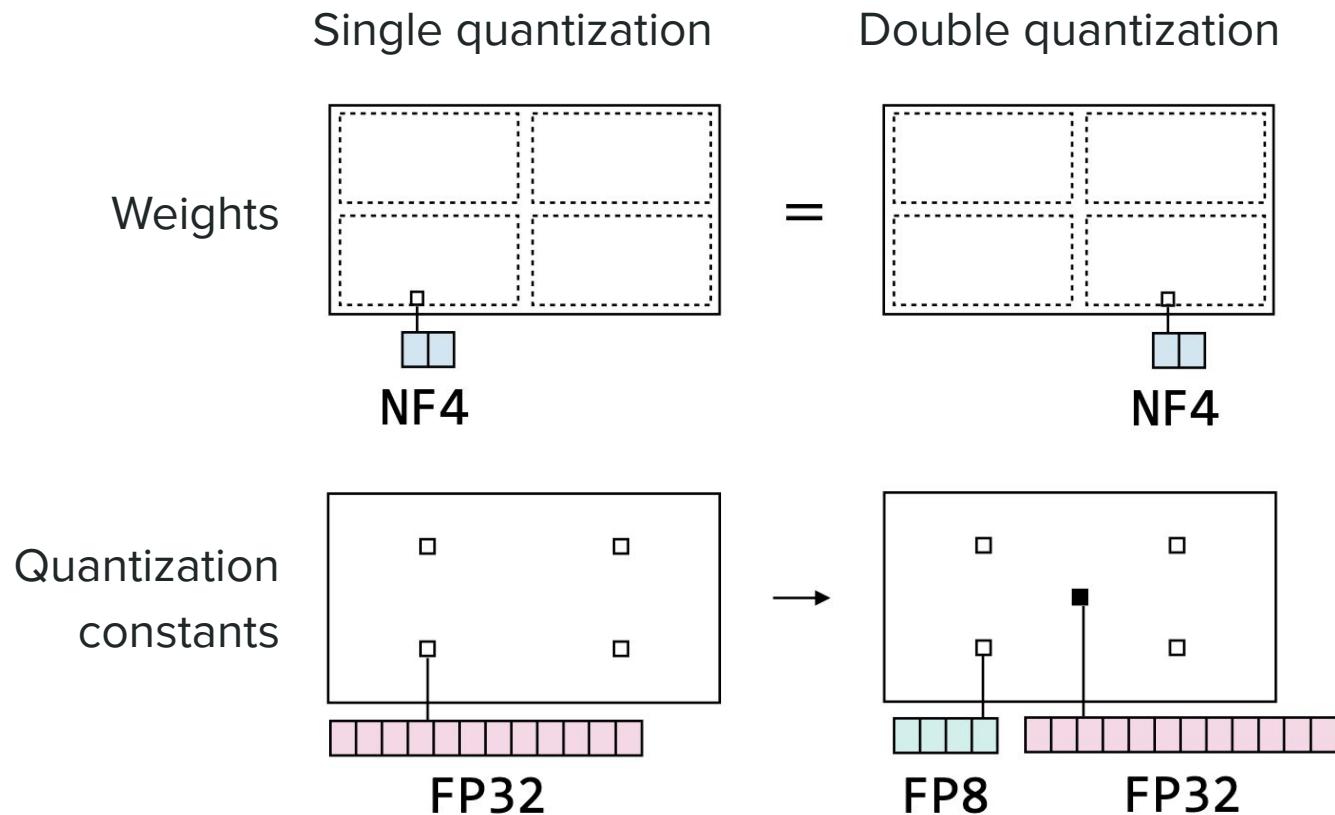
# Quantization...



# Quantization...



...done two times!



# QLoRA

**Idea.** Quantize all frozen weights to relieve memory bottleneck.

## Benefits.

- VRAM savings enable finetuning on smaller GPUs, and faster
- Better trade-off memory resources/quality

# QLoRA

**Idea.** Quantize all frozen weights to relieve memory bottleneck.

## Benefits.

- VRAM savings enable finetuning on smaller GPUs, and faster
- Better trade-off memory resources/quality

**Orders of magnitude.** Results reported out of LLaMA 65B:

- ~16x VRAM savings during finetuning
- Double quantization trick saves an extra ~6%

Thank you for your attention!

---