Lecture 4

CME 295: Transformers & Large Language Models

Afshine Amidi & Shervine Amidi

Stanford University

ICME

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.

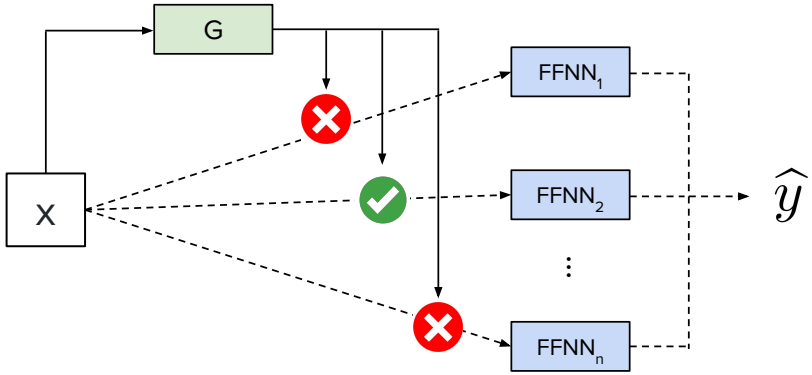
- **When?** Friday October 24th, 3:30pm - 5:00pm (duration: 1h30)
- **Where?** Thornton 110 (i.e. this classroom)
- **Topics?** Lecture 1 + 2 + 3 + 4

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...



mixture of Experts
- dense MoE (almost all or more MoE)
- sparse MoE (few MoE considered)

Recap of last episode...

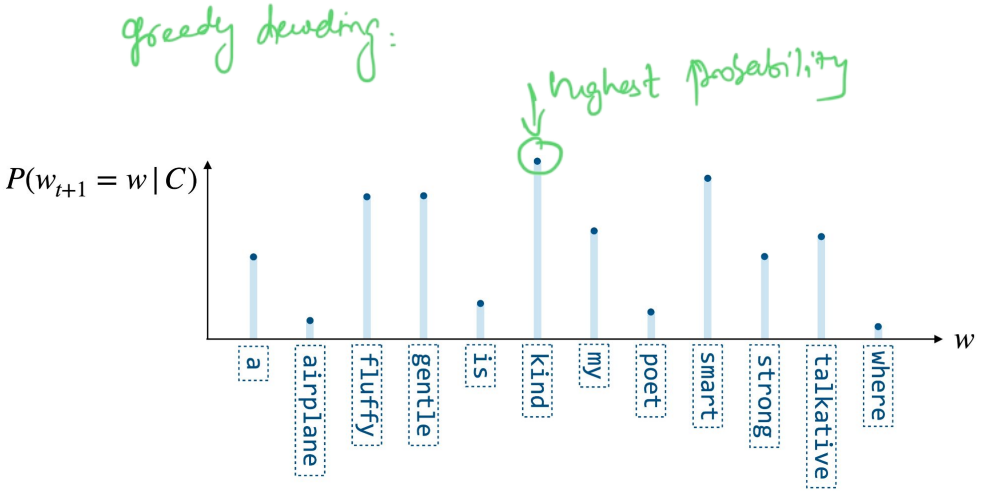
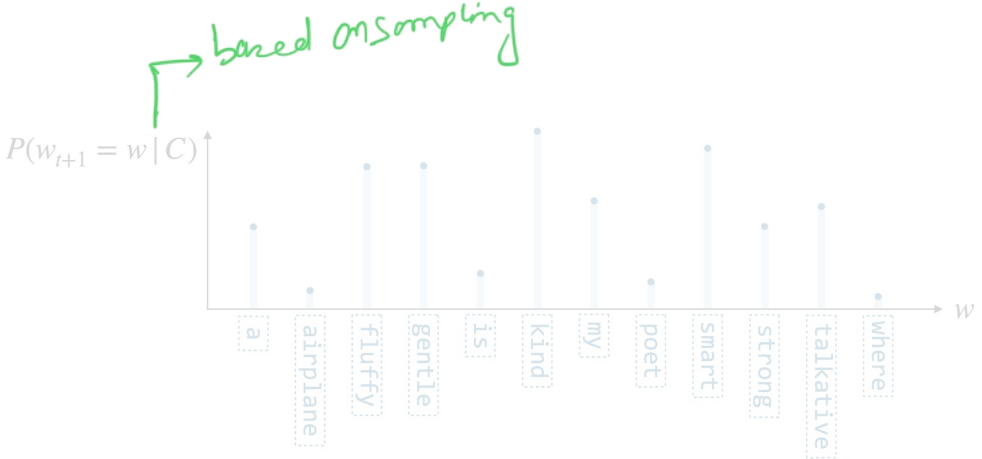


Figure from "Super Study Guide: Transformers and Large Language Models", Amidi et al., 2024.

Recap of last episode…



important for improving performance in inference.

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

Transformers & Large Language Models

Pretraining ↳ LLM training.

Training optimizations

Supervised finetuning

Parameter-efficient finetuning

Paradigm shift

Traditional ML. Given task, train model from scratch

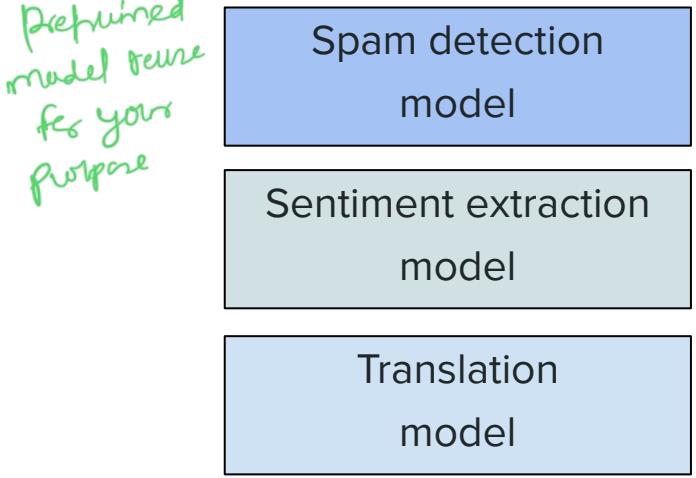
train model for specific purpose

1. Spam detection model
2. Sentiment extraction model
3. Translation model

Paradigm shift

Traditional ML. Given task, train model from scratch

Transfer learning. Reuse trained model to some capacity



Paradigm for LLM training

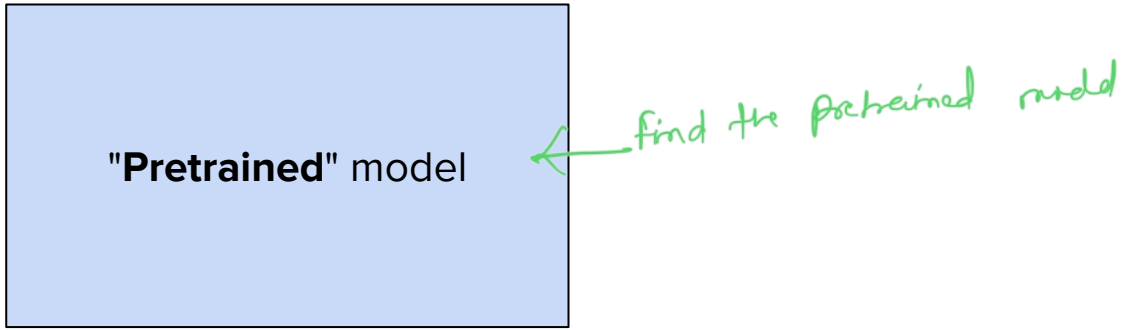
LLM training. Train LLM to understand language, then tune it for end task

Paradigm for LLM training

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

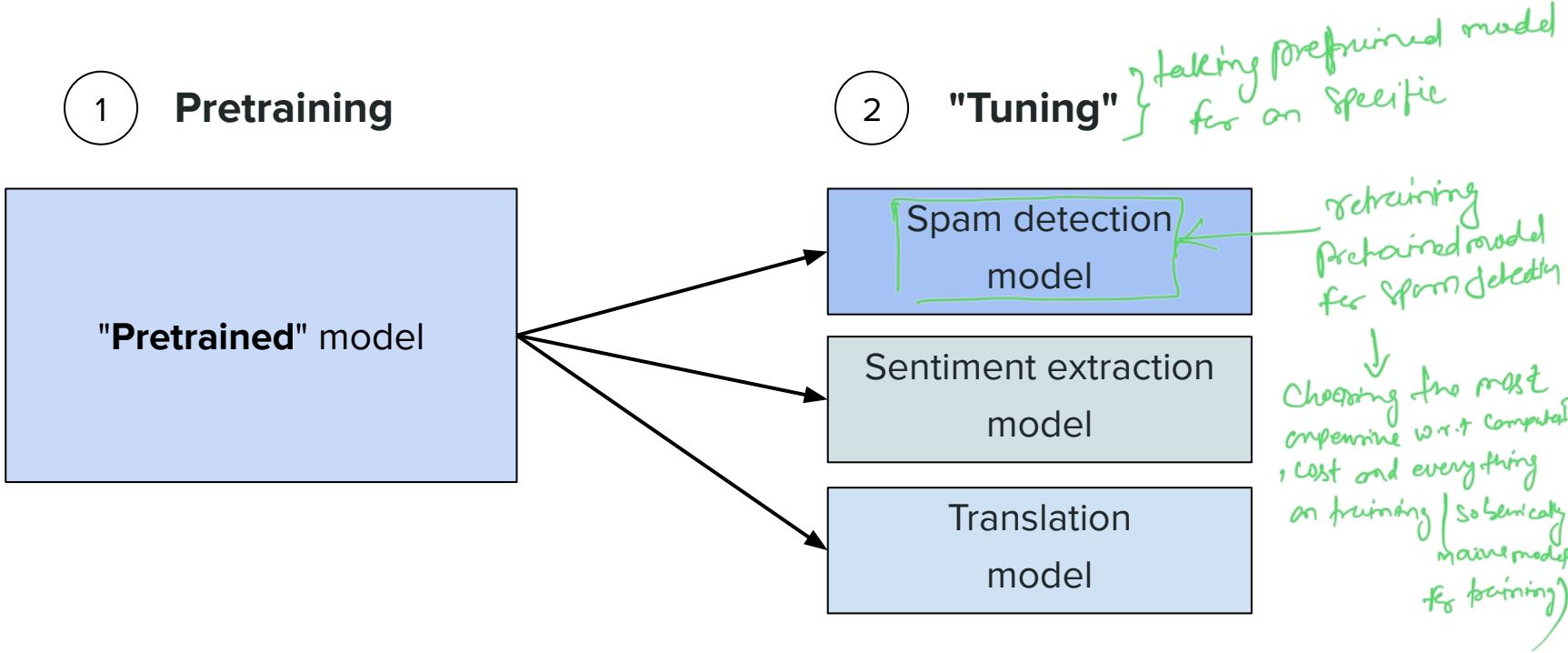
this example for language understanding (text-to-text model)

1 **Pretraining**



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 حاضره به قصه‌های من گوش بده...

...

teddy_bear.py

```python
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

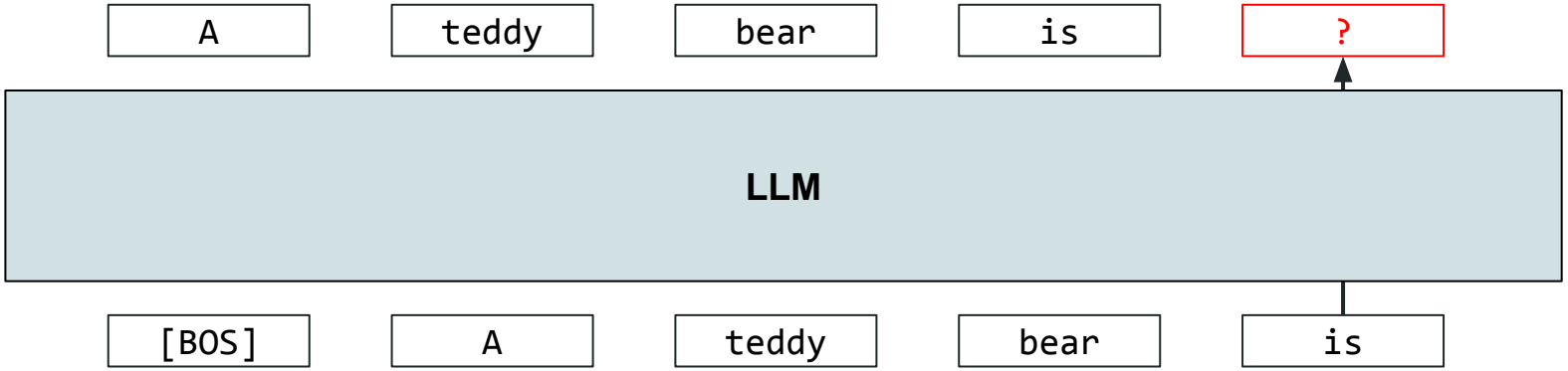
```go
package main

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

Pretraining overview

Goal. Learn pattern of language and code

Objective function. Predict next token (in text-to-text model, trying to prediction next token)



Pretraining overview

Goal. Learn pattern of language and code

Objective function. Predict next token

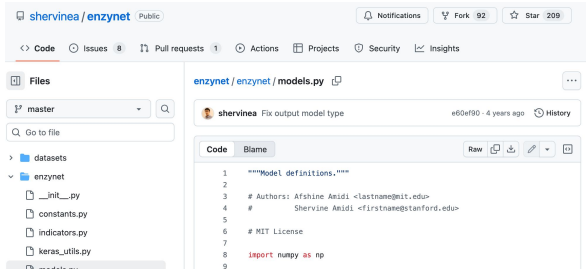
Data mixtures. (Datasets)

all from internet daily.

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

for code

Code debugging / coolings



Teddy bear Wikipedia article, screenshot taken on October 14, 2025. A GitHub repo, screenshot taken on October 15, 2025.

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)

↓ dataset size can be measured into # of tokens.

Size. Approx. trillions of tokens

→ Naive model for pretraining

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

"Language Models are Few-Shot Learners", Brown et al., 2020. "The Llama 3 Herd of Models", LLaMA team, 2024.

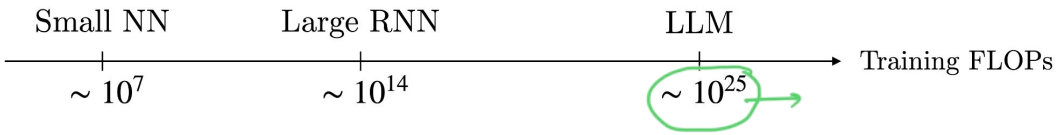
Notations

FLOPs helps to understand training/performance complexity

📖 **FLOPs**

***FL**oating-point **OP**erations*

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



higher the number for FLOPs, more computation is being done/more operations done/ie. with higher # of tokens too.

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

Notations

FLOPs

***FL**oating-point **OP**erations*

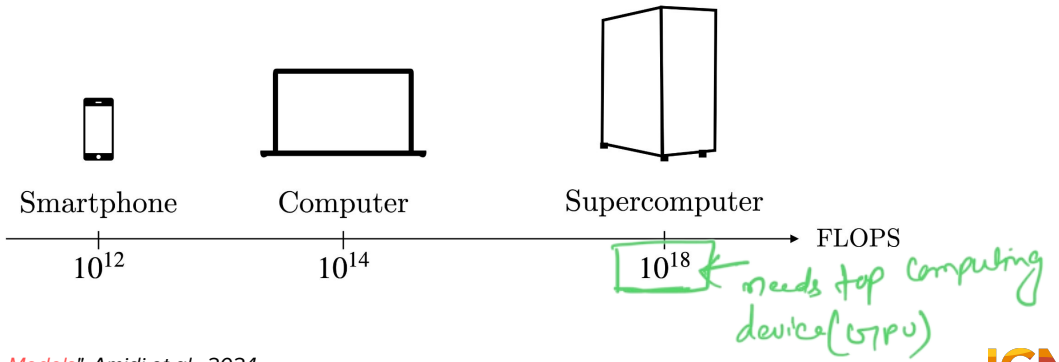
$f(\#\text{tokens}, \#\text{parameters})$

⚡ **FLOPS** or **FLOP/s**

***FL**oating-point **OP**erations*

*per **S**econd*

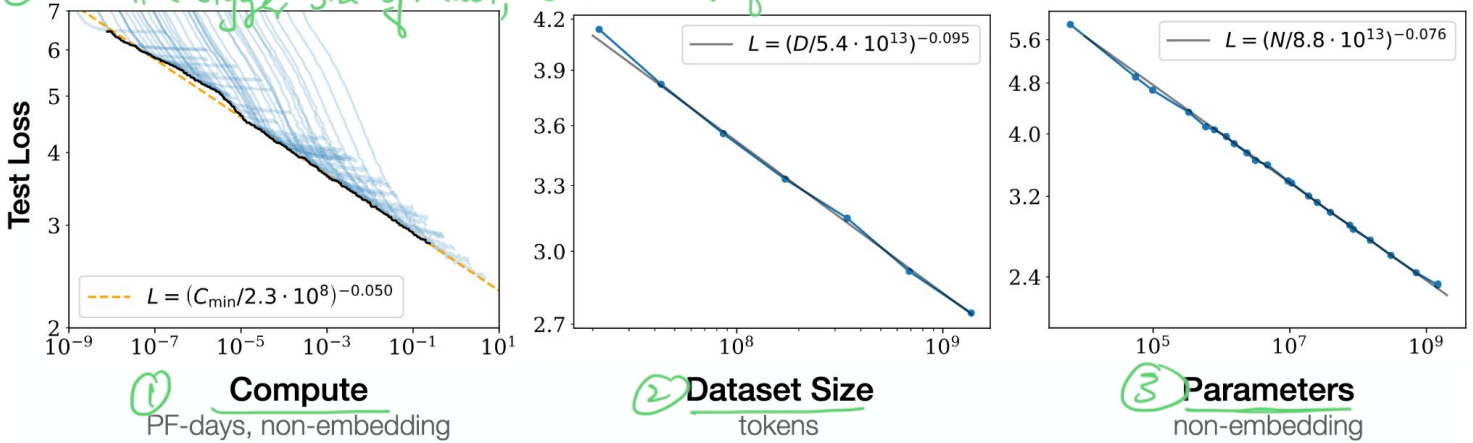
Computational performance



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

Takeaways

Scaling. (model evolves w.r.t (model size and training size) of model.
① → the more compute will be model performance would be better
② → the bigger data size, better the performance of model.
③ → the bigger size of model, better the performance.



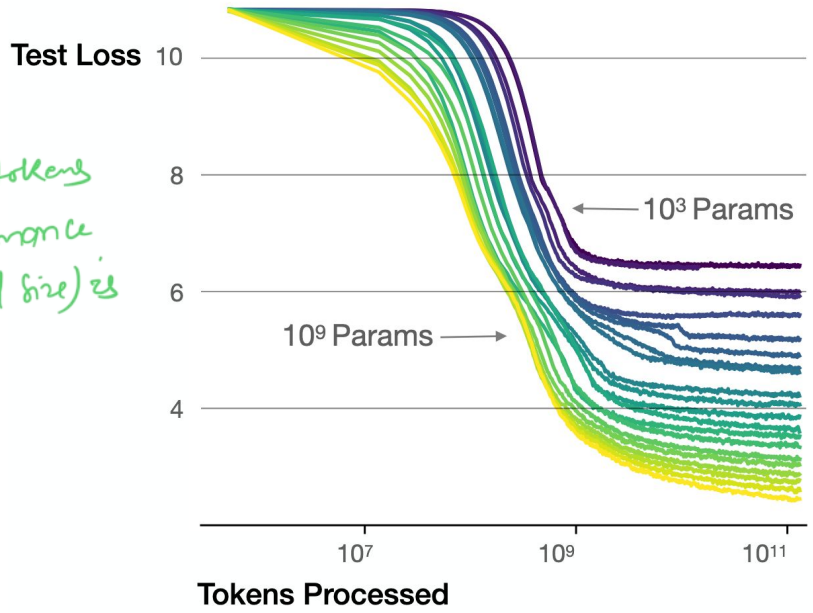
"Scaling Laws for Neural Language Models", Kaplan et al., 2020.

Takeaways

Scaling.

Sample efficiency.

bigger model tends to be more sample efficient i.e. for a equal amount of tokens are processed, better performance when # of parameters (model size) is higher.



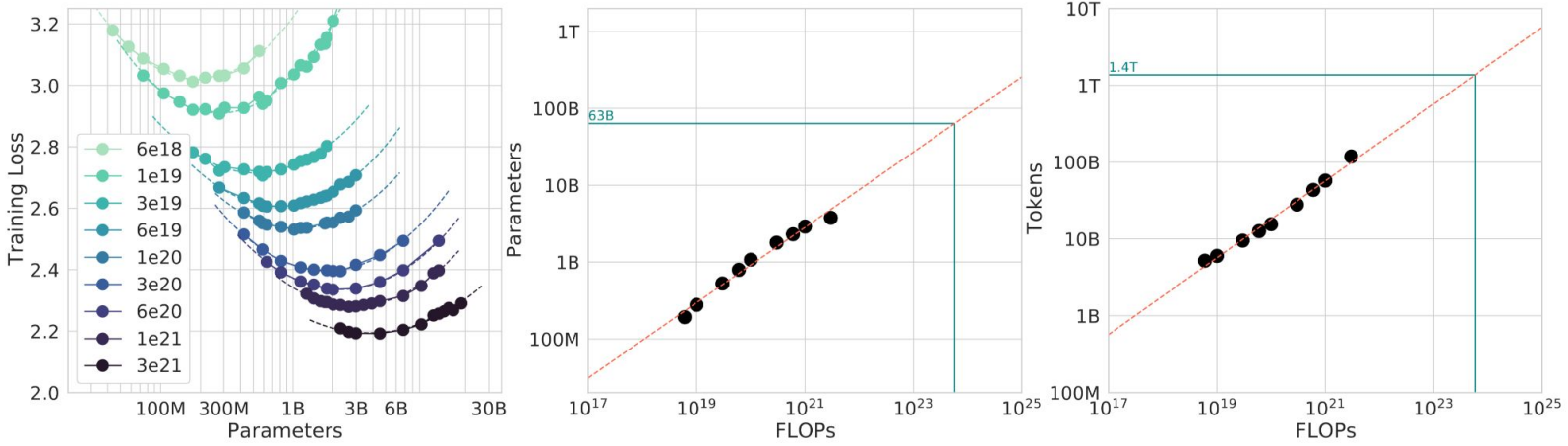
"Scaling Laws for Neural Language Models", Kaplan et al., 2020.

Takeaways

Scaling.

Sample efficiency.

Chinchilla law. this law says that [20x data size for training > model size] considered as optimal model trained.



"*Training Compute-Optimal Large Language Models*", Hoffmann et al., 2022.

Takeaways

Scaling.

Sample efficiency.

Chinchilla law.

Parameters	FLOPs	FLOPs (in *Gopher* 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

Challenges

Cost.

- (At least) millions of dollars [High cost]
- Takes time ["Hell a lot of time"]
- Environment / electricity ["Ecological cost"]

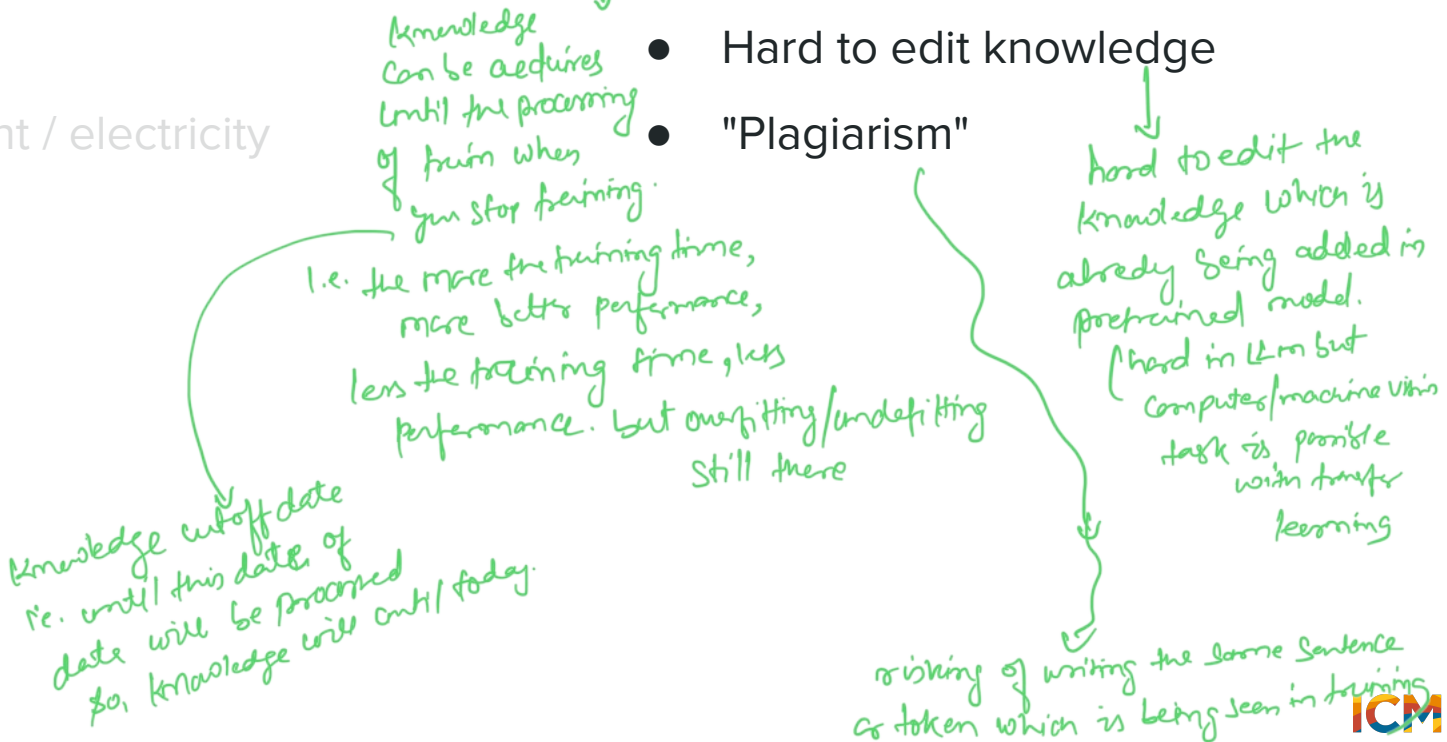
Challenges

Cost.

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

Learned knowledge.

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



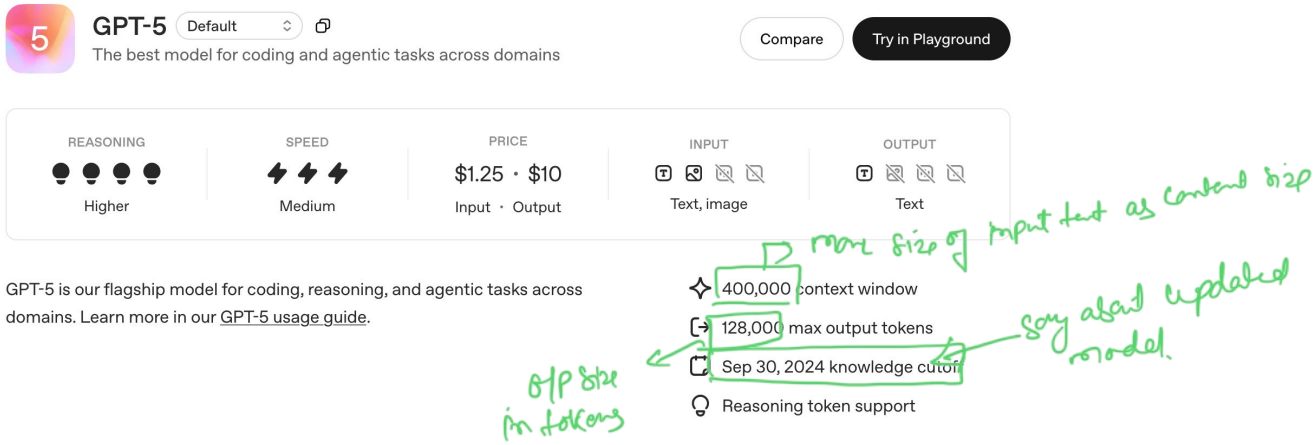
Challenges

Cost.

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

Learned knowledge.

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



https://platform.openai.com/docs/models/gpt-5, screenshot taken on October 14, 2025.

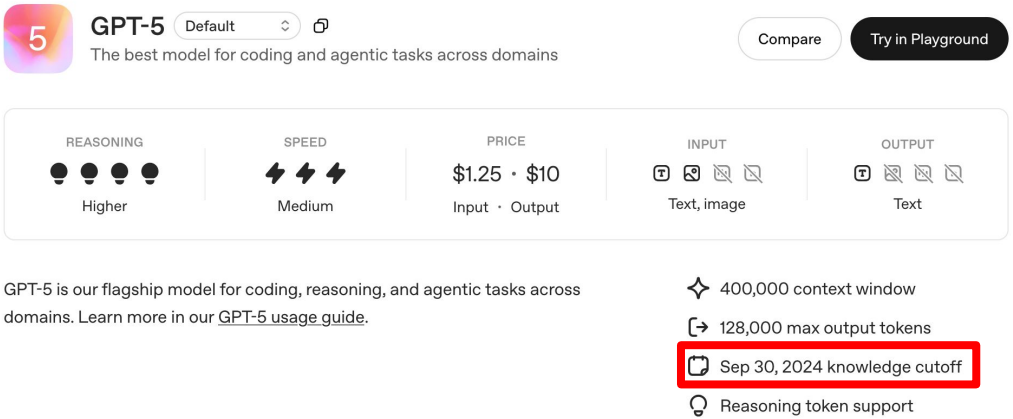
Challenges

Cost.

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

Learned knowledge.

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



https://platform.openai.com/docs/models/gpt-5, screenshot taken on October 14, 2025.

Transformers & Large Language Models

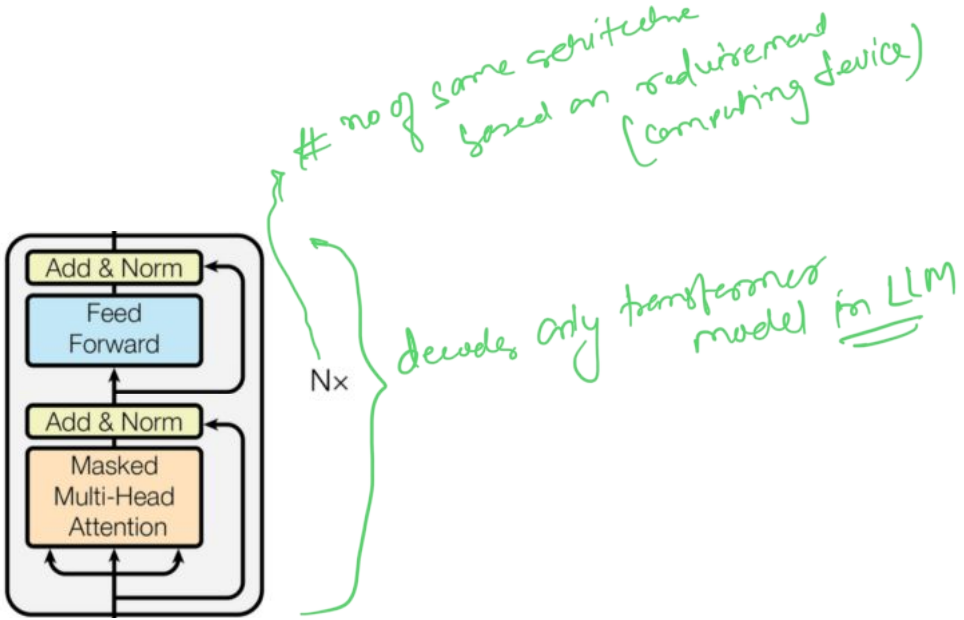
Pretraining

Training optimizations

Supervised finetuning

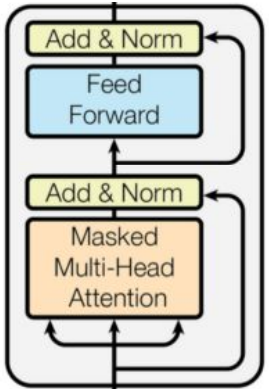
Parameter-efficient finetuning

Setup of LLM training

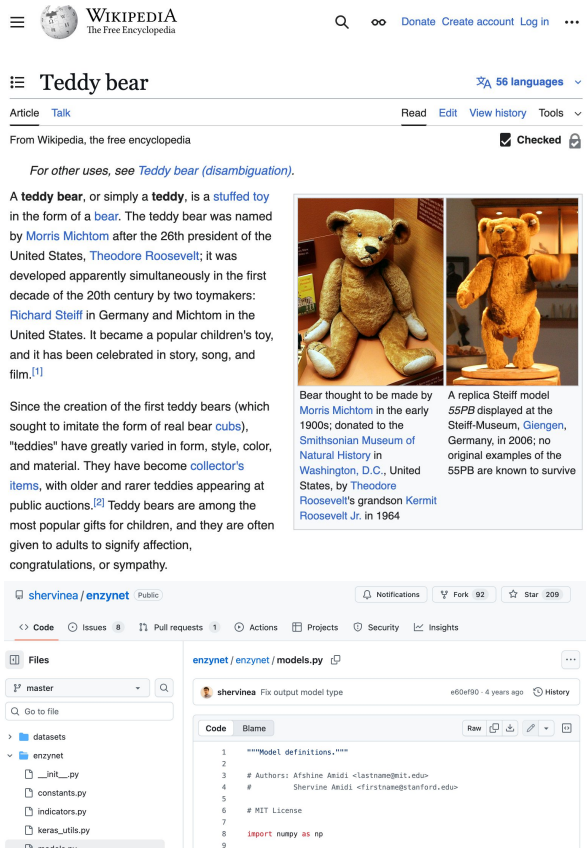


"*Attention Is All You Need*", Vaswani et al., 2017.

Setup of LLM training

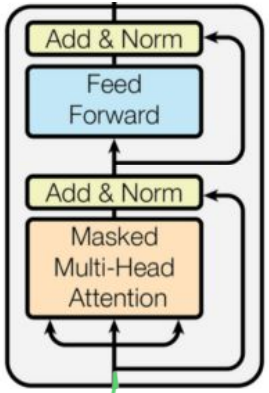


Nx



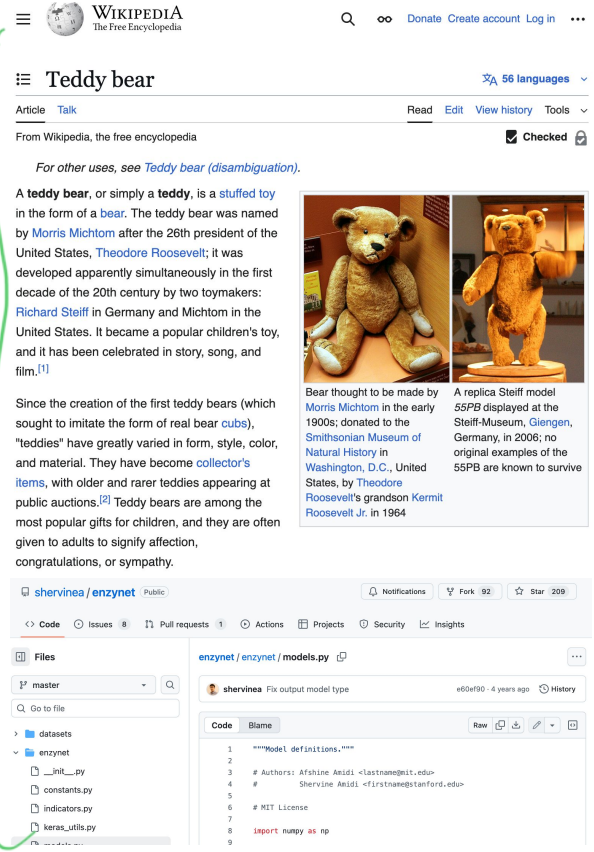
"Attention Is All You Need", Vaswani et al., 2017.

Setup of LLM training



Nx

architecture



data

training device

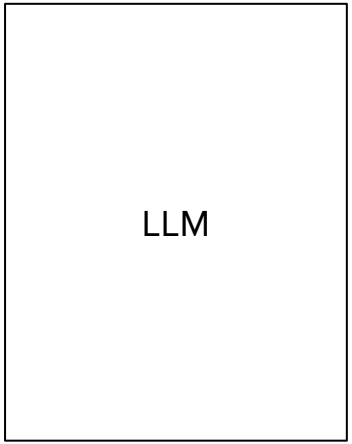
(Many) GPU(s)!

bs2 of flops

(a lot of computation)

"*Attention Is All You Need*", Vaswani et al., 2017.

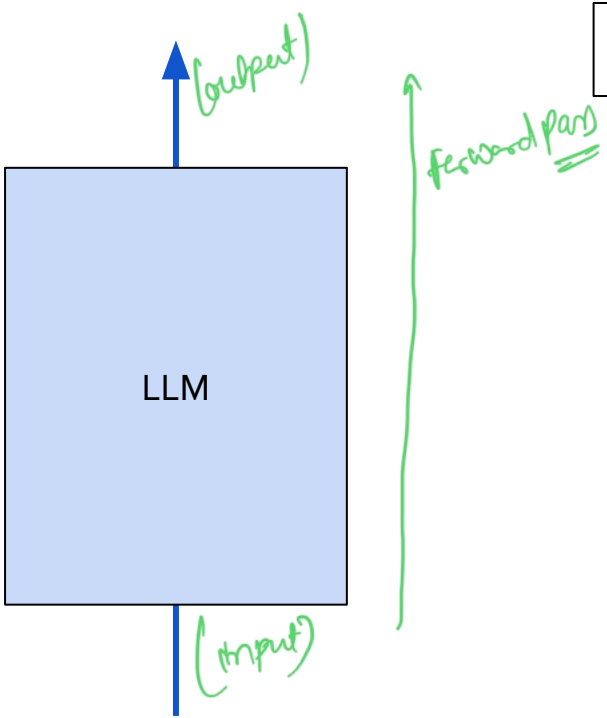
Recap of LLM training



Initialization // parameters, device, more required training parameters

Model parameters. O(billions) - O(100s of billions)

Recap of LLM training



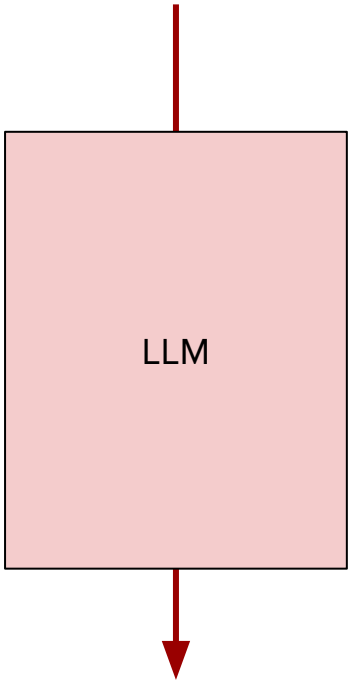
Forward pass. Compute loss.

Activations. Needed to compute the loss.

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

Function of model size, batch size, context length

Recap of LLM training



Backwards pass. Compute gradients.

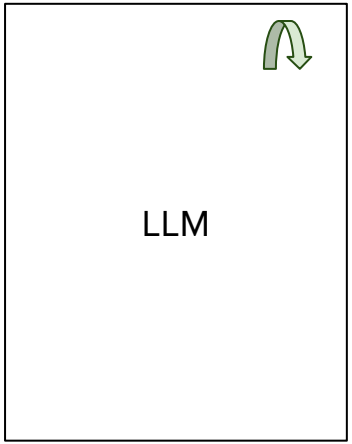
Gradients. Needed for weights update.

update the weight based on loss fcn (gradient descent optimizer or more ie. Adam optimizer)

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

finding direction where it should go for minimizing the loss fcn

Recap of LLM training



Weights update. Update model parameters.

Optimizer state. Updates the weights.

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

Adam optimizer

momentum added.

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

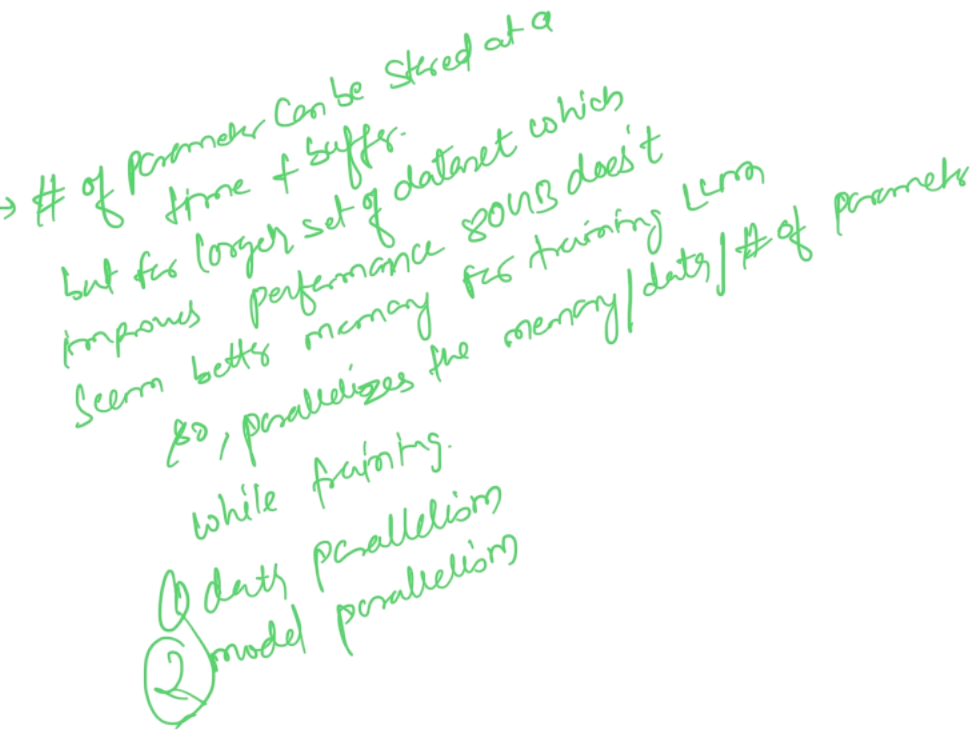
$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)(\nabla \mathscr{L}(\theta_t))^2$

"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**
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
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



"NVIDIA H100 Tensor Core GPU", NVIDIA website.

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
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

Limited memory, O(10s of GB)

"NVIDIA H100 Tensor Core GPU", NVIDIA website.

Data parallelism

distributing data for training across devices

Idea.

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

Data parallelism

a lot of duplication if only parallelise the GPU for training
i.e. all for # of parameters, gradient, optimizer state
so, again it increases the headache of training so, ZeRO introduced.
(Zero Redundancy optimisation)

Idea.

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

Data parallelism with ZeRO

Idea. Redundant information across devices

ZeRO = **Ze**ro **R**edundancy **O**ptimization

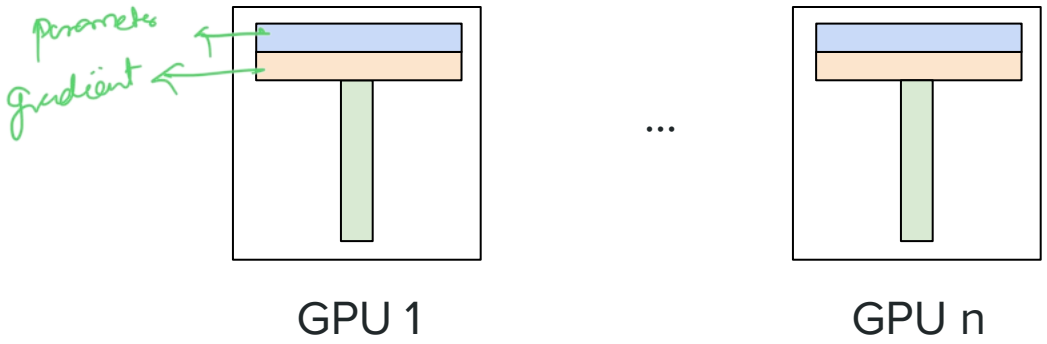
"*ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*", *Rajbhandari et al., 2019.*

Data parallelism with ZeRO

Idea. Redundant information across devices

ZeRO = **Ze**ro **R**edundancy **O**ptimization

Variations. ZeRO-1: Share **optimizer state** (partitioned optimizer state i.e. parameter, and gradient will be same but optimize will be shared across all GPU.



"ZeRO: Memory Optimizations Toward Training Trillion Parameter Models", Rajbhandari et al., 2019.

Data parallelism with ZeRO

Idea. Redundant information across devices

ZeRO = **Ze**ro **R**edundancy **O**ptimization

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

w/m gradient
i.e. (optimizer state + gradient will be separated but parameters would be same)

"*ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*", *Rajbhandari et al., 2019.*

Data parallelism with ZeRO

Idea. Redundant information across devices

ZeRO = **Ze**ro **R**edundancy **O**ptimization

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

all separated out with training across GPU

"*ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*", *Rajbhandari et al., 2019.*

Model parallelism

splitting model itself across all GPUs

Idea. Split the model computations across several devices

Variations.

- **T**ensor **P**arallelism **(TP)**. → splitting matrix computation
- **P**ipeline **P**arallelism **(PP)**. → forward pass / backward pass layering
- **S**equence **P**arallelism **(SP)**. → data sequence splitting across GPU
- **C**ontext **P**arallelism **(CP)**. →
- **E**xpert **P**arallelism **(EP)**. → based on Experts for training for each experts for different purpose

Suggested reading: "The Ultra-Scale Playbook: Training LLMs on GPU Clusters", HuggingFace team, 2025.

Optimization via Flash Attention

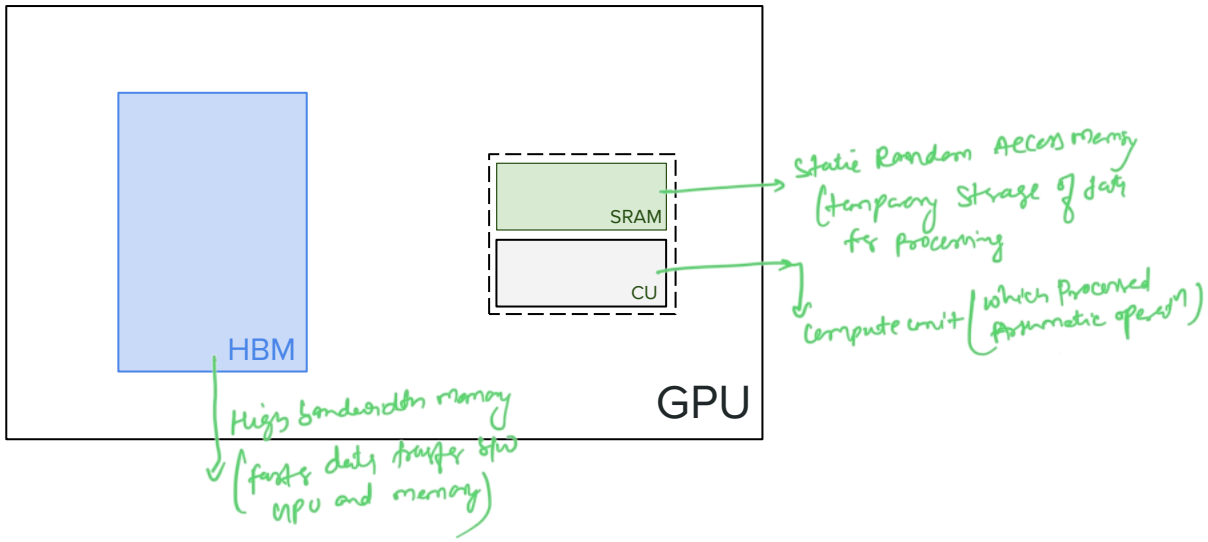
↓ leveraging GPU utilizat?

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

"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

Optimization via Flash Attention

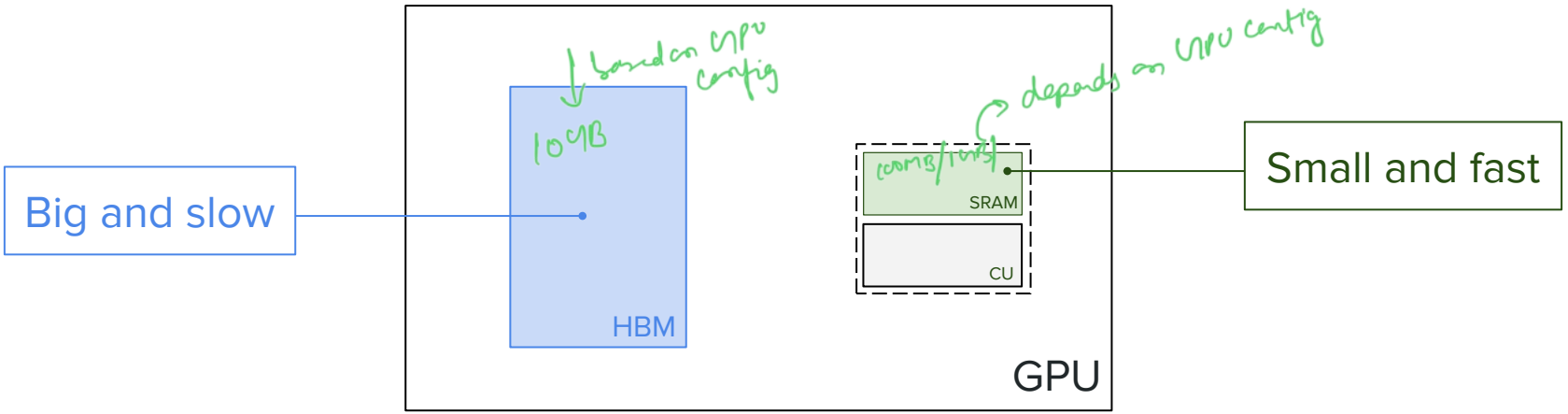
Strategy. Leverage components of GPU to speed up attention computations.



"*FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*", Dao et al., 2022.

Optimization via Flash Attention

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



"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

Standard self-attention computation

Context. "Standard" self-attention computation

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

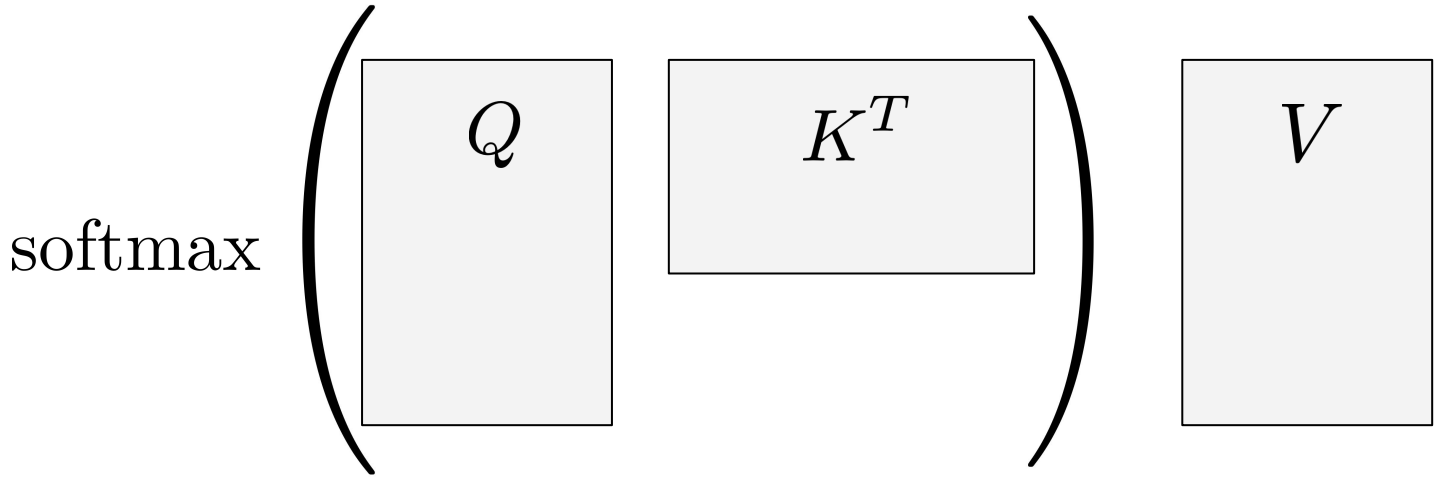
large and can be repeating so store in HBM → after computation (SRAM)

V is also in HBM but after computing Q, K^T and keep in another place bc HBM is also responsible for data transfering.

"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

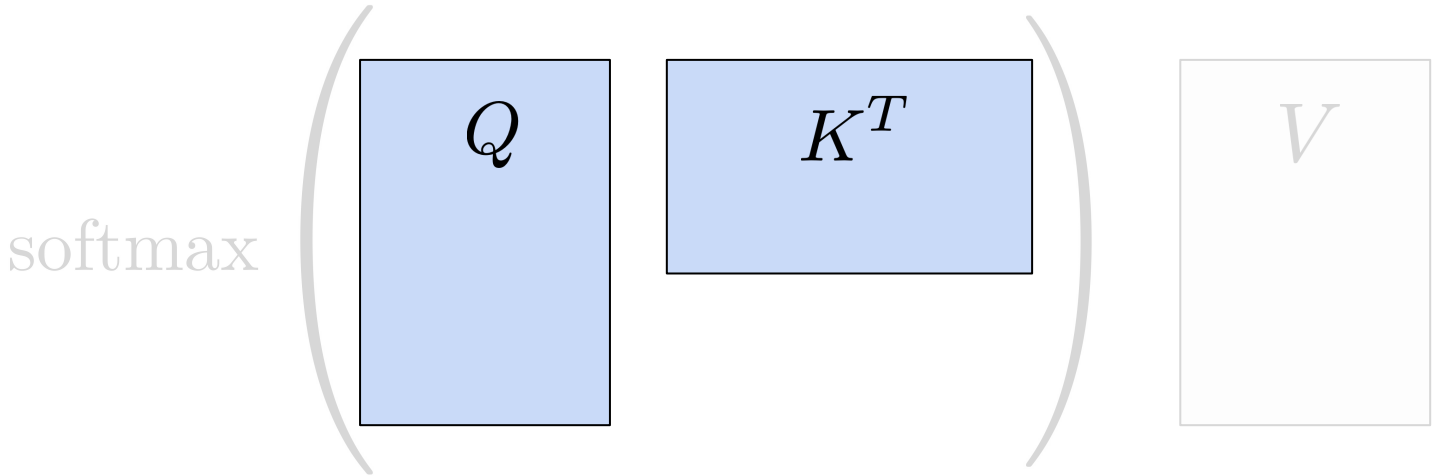
Standard self-attention computation

Context. "Standard" self-attention computation



Standard self-attention computation

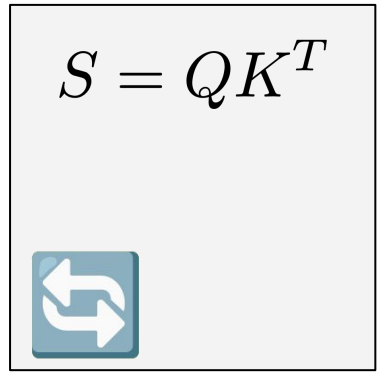
LOAD Q, K from **HBM** by blocks



Standard self-attention computation

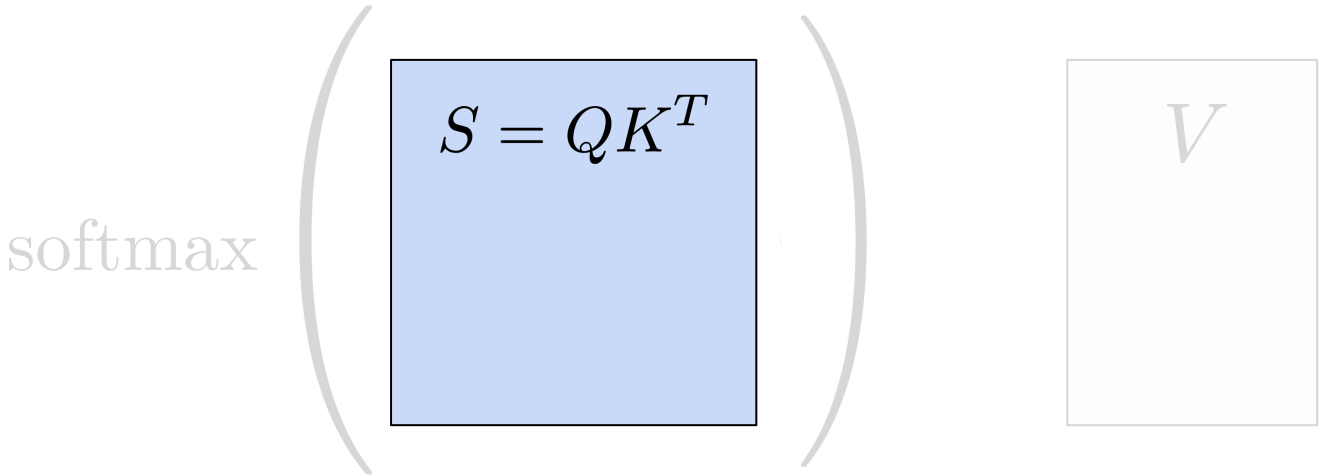
Compute S

$$\text{softmax}\left(S = QK^T \right) V$$



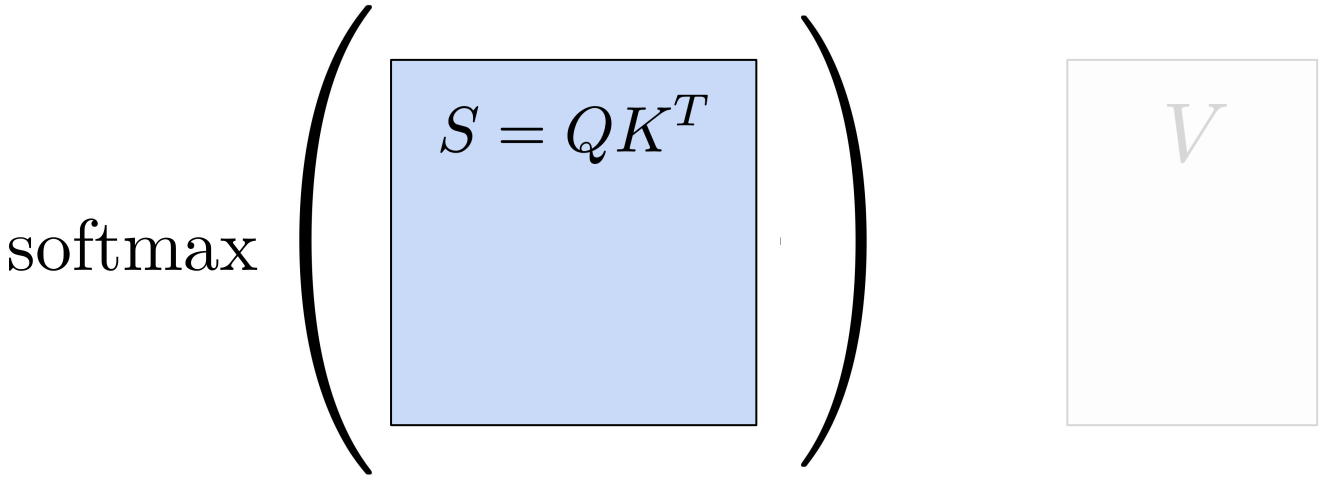
Standard self-attention computation

WRITE S to **HBM**



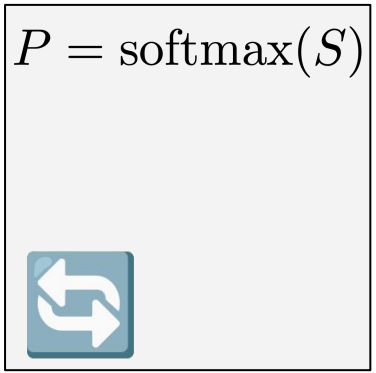
Standard self-attention computation

READ S from **HBM**



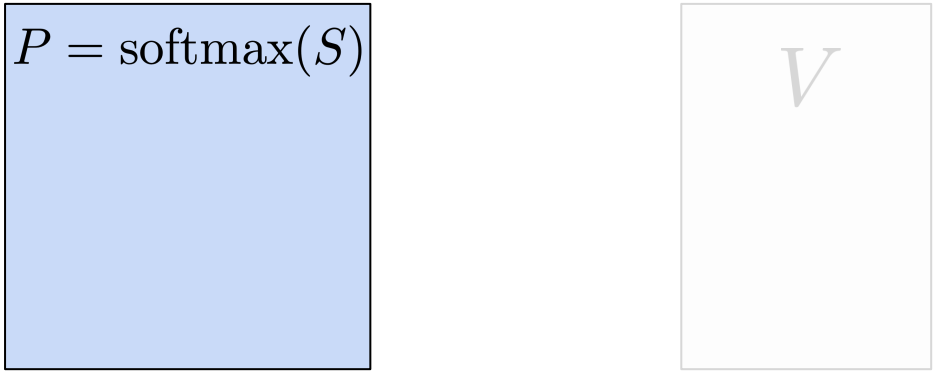
Standard self-attention computation

Compute P



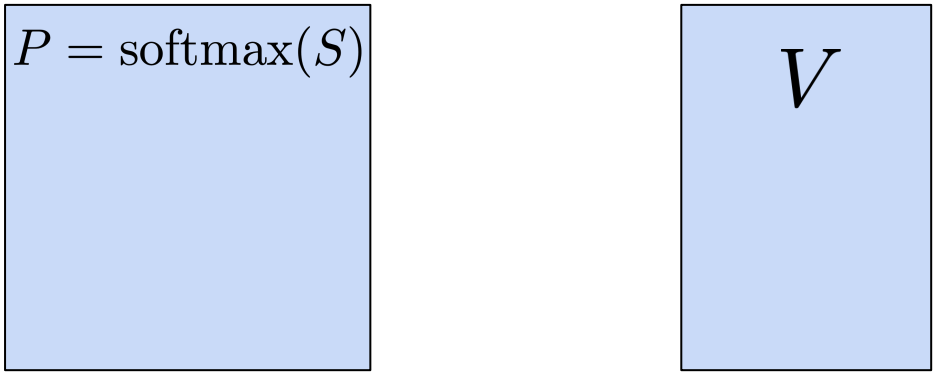
Standard self-attention computation

WRITE P to **HBM**



Standard self-attention computation

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

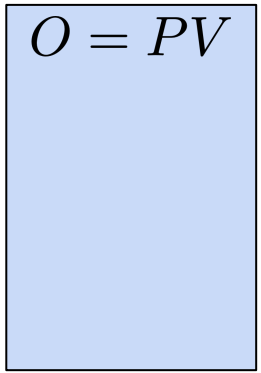


Standard self-attention computation

Compute O

Standard self-attention computation

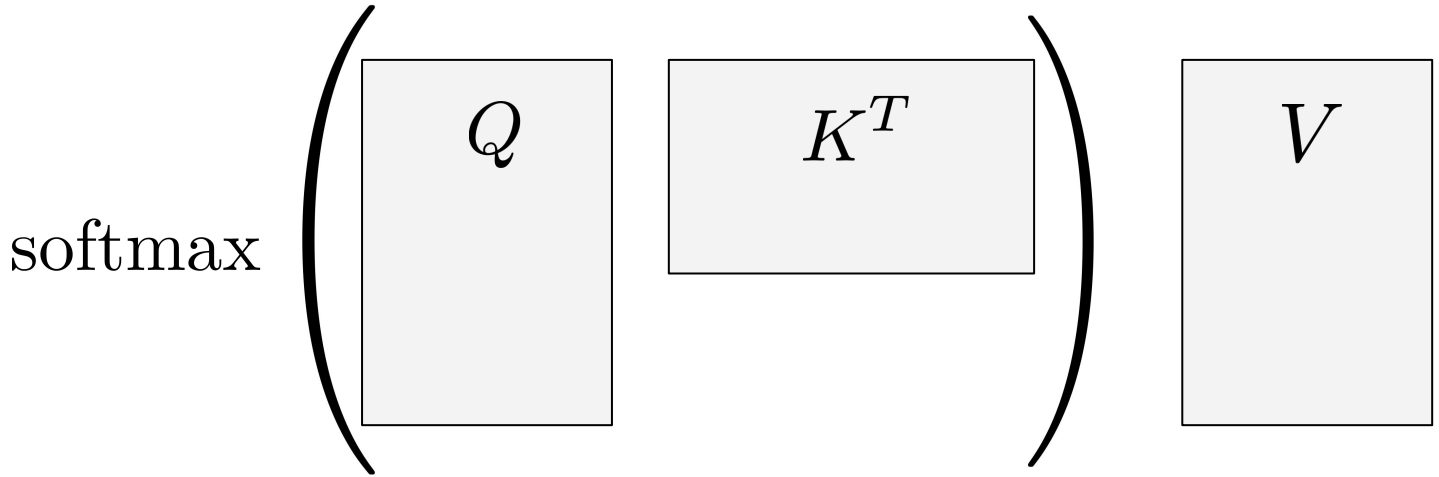
WRITE O to **HBM**



Flash Attention in action

goal

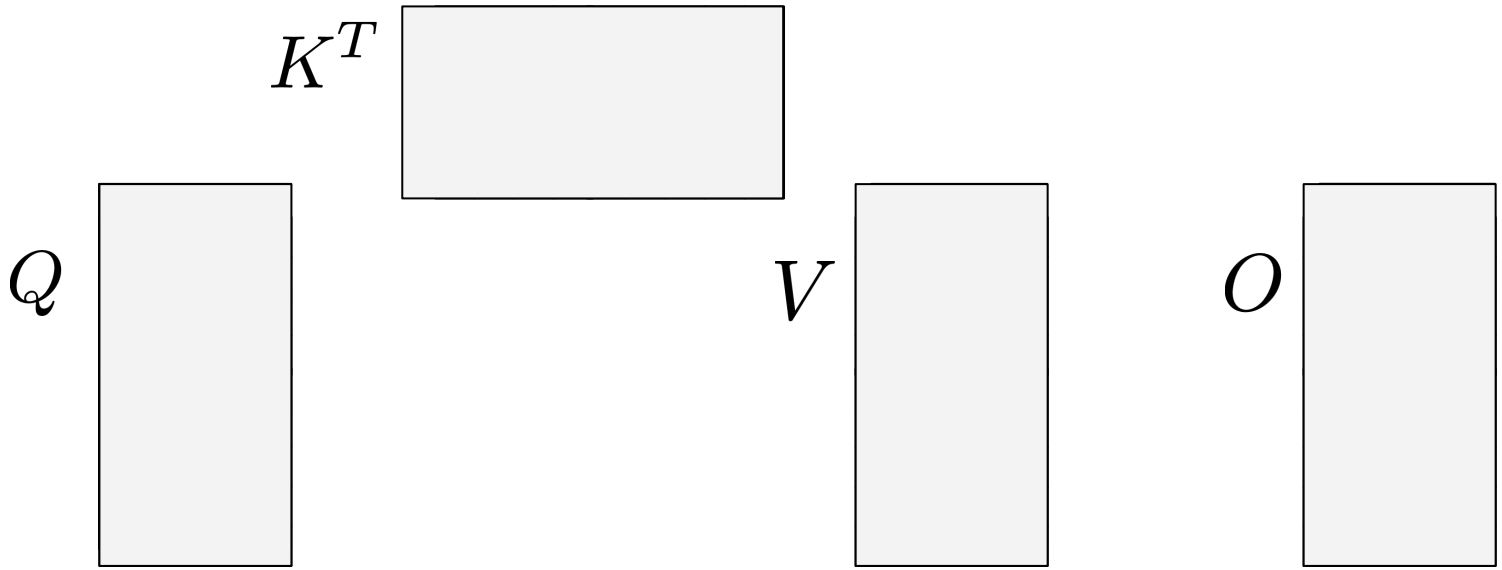
1st idea. Minimize read/writes to **HBM** with "tiling" via **SRAM**



"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

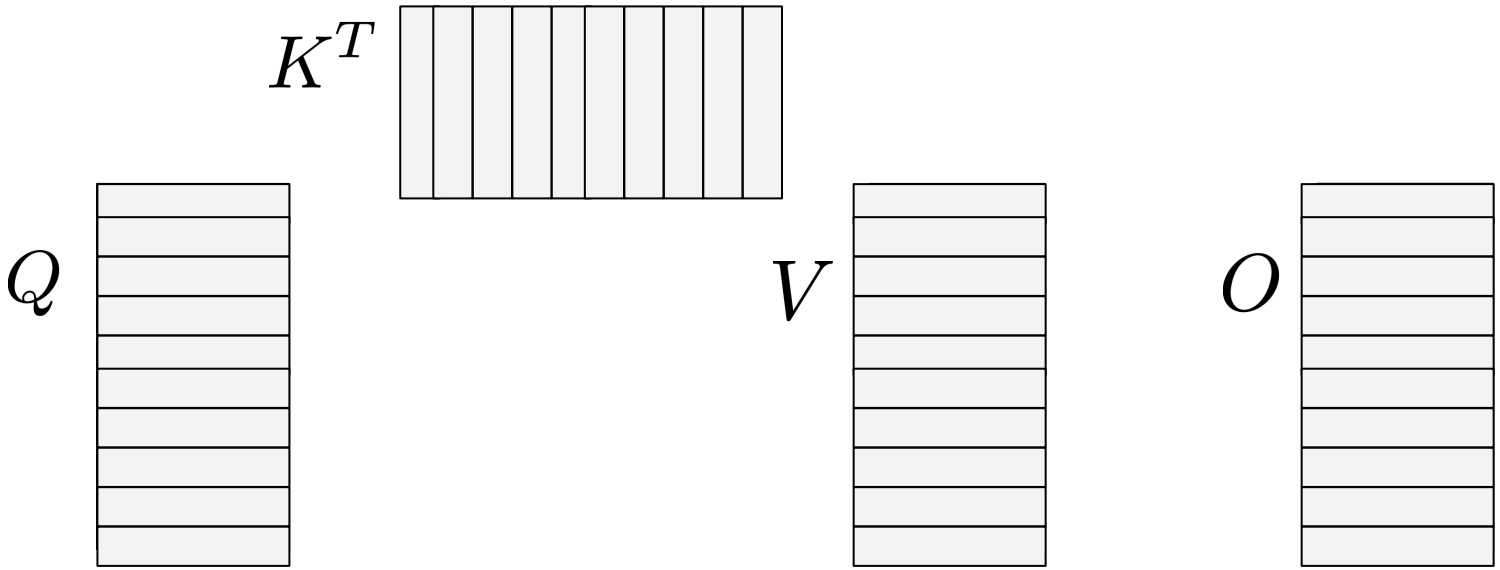
Flash Attention in action

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



Flash Attention in action

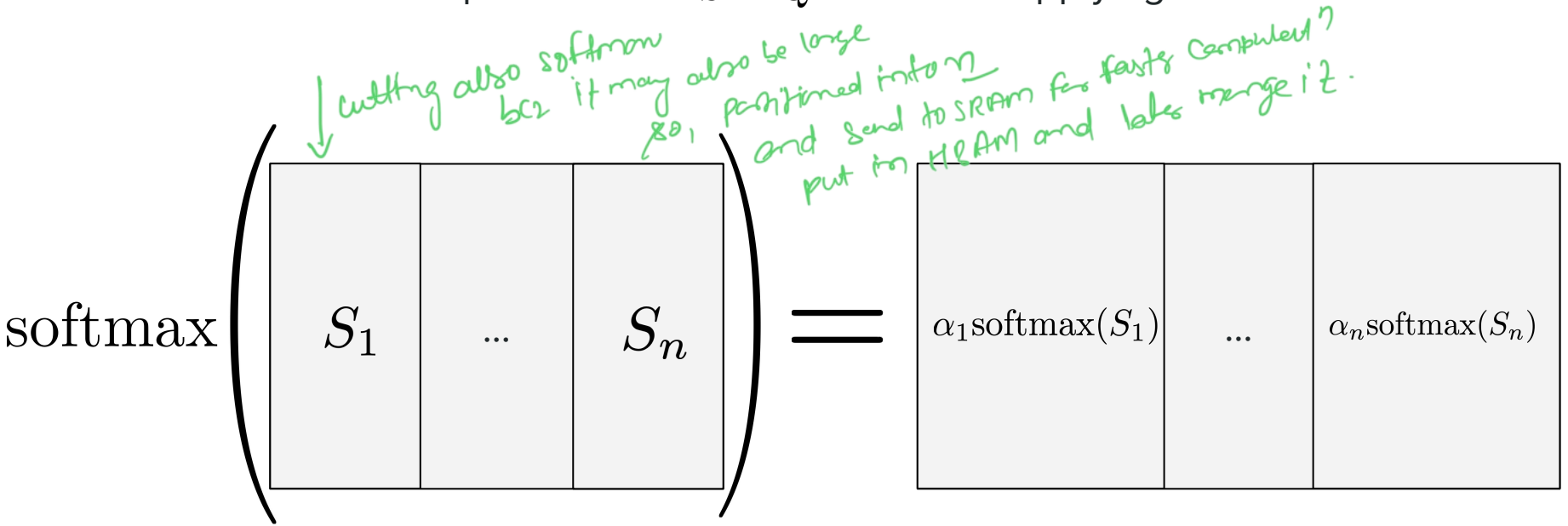
1st idea. Minimize read/writes to **HBM** with "tiling" via **SRAM**



"*FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*", Dao et al., 2022.

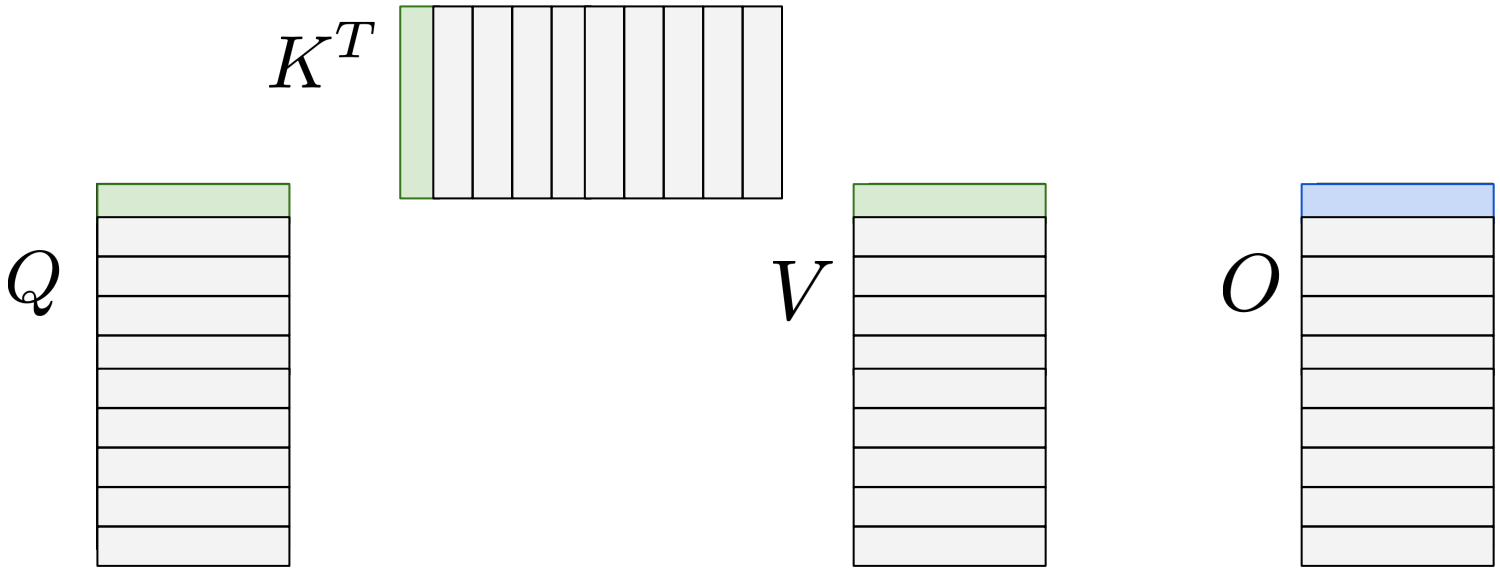
Flash Attention in action

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



Flash Attention in action

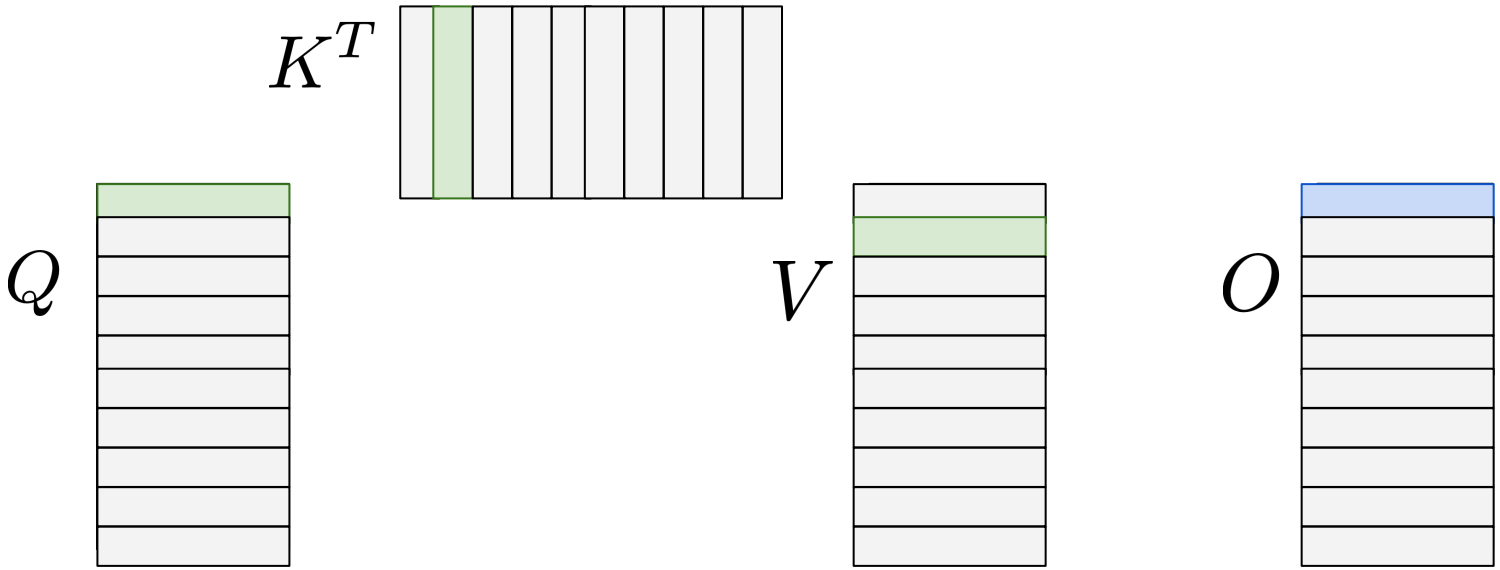
Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

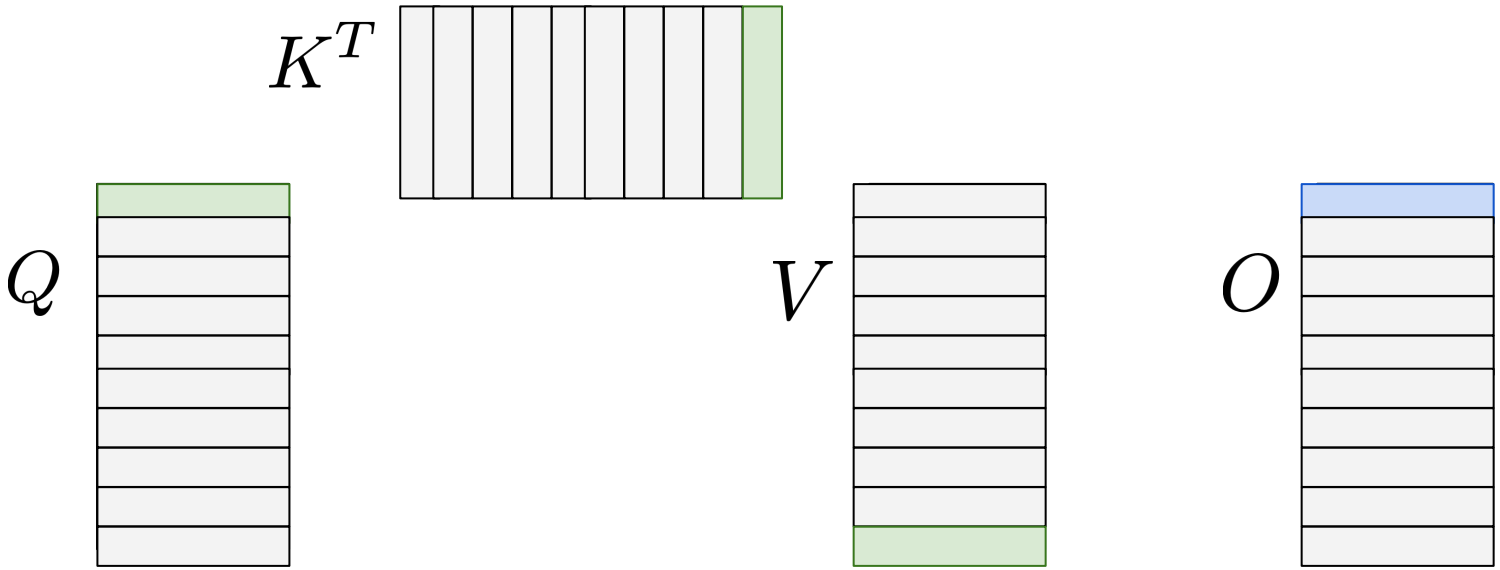
Flash Attention in action

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



Flash Attention in action

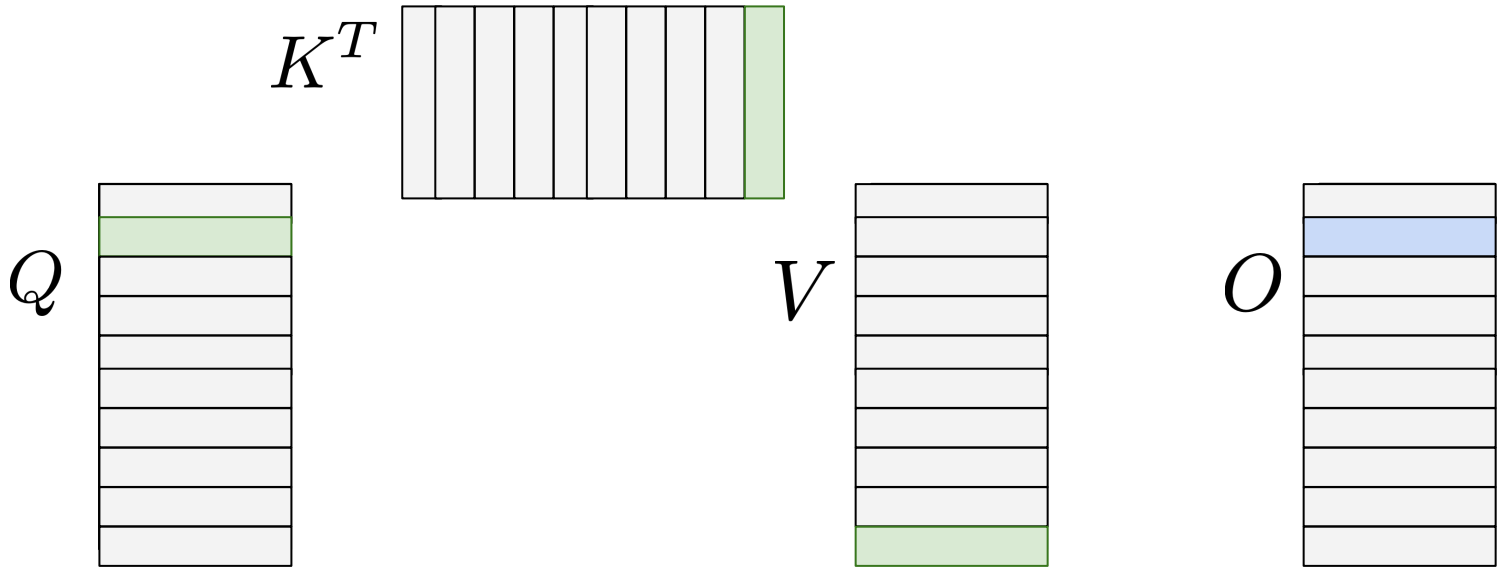
Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



"*FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*", Dao et al., 2022.

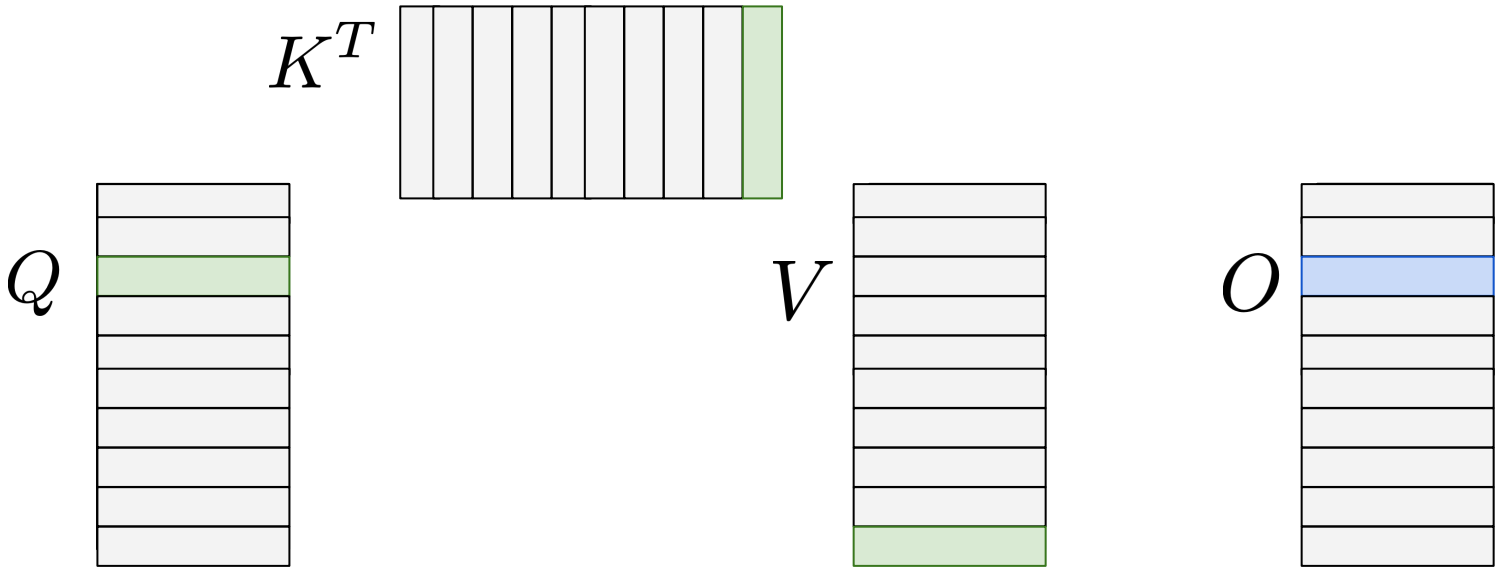
Flash Attention in action

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



Flash Attention in action

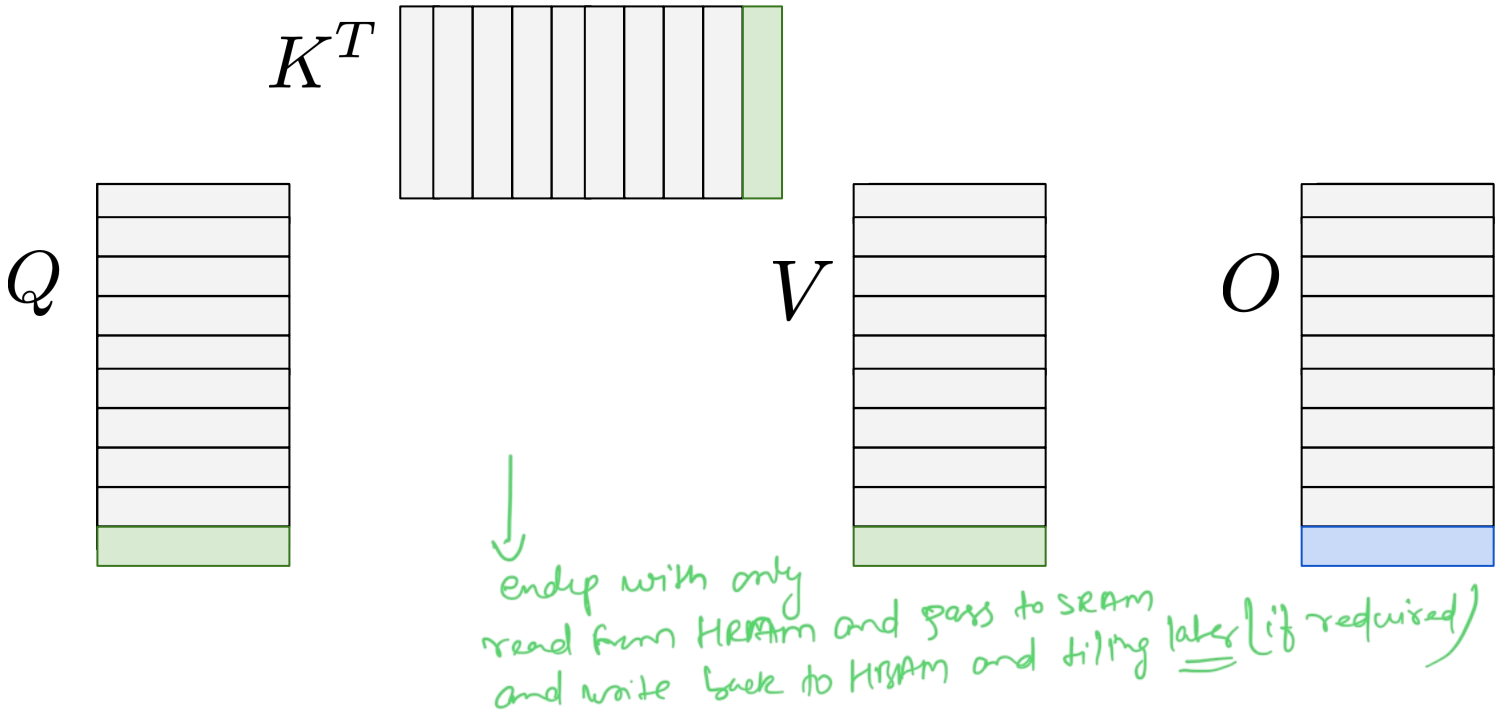
Load blocks of Q, K, V to **SRAM**, compute block of O and write the result to **HBM**



"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

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

2nd idea. Sometimes, it is better to **recompute** instead of storing

↳ to improve the processing time of training.

Before	After
Store S, P in the **HBM**	Recompute S, P with tiling via **SRAM**

Smart work for backward pass,
(not store the activation, but re-compute again and again
unlike 1st idea is storing all activation which is not
beneficial as we want optimal activation
at the end.

"FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness", Dao et al., 2022.

Flash Attention in the backwards pass

2nd idea. Sometimes, it is better to **recompute** instead of storing

Before	After
Store S, P in the **HBM**	Recompute S, P with tiling via **SRAM**

More FLOPs, but less runtime!!

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

Speeding the computation using quantization technique

2.7015402...

1.2015432...

2.4024402...

-0.7055120...

-1.7067140...

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 → Sign + Exponent + Mantissa

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

Representation of a float

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

Example of a GPU

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
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

Compute speed given for quantized representation

"NVIDIA H100 Tensor Core GPU", NVIDIA website.

Example of a GPU

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
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

"NVIDIA H100 Tensor Core GPU", NVIDIA website.

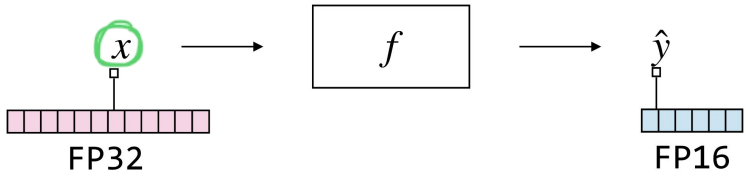
Mixed precision training

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

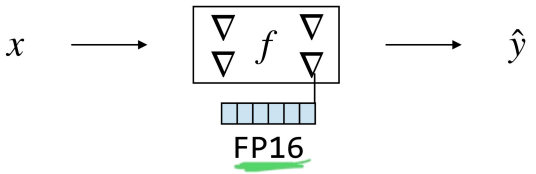


Mixed precision training

Objective. Speed up training and decrease memory requirements

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

- **Backwards pass**. Gradient updates in low precision (FP16)



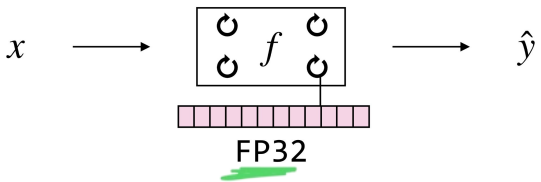
Mixed precision training

Objective. Speed up training and decrease memory requirements

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

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

- **Weights update**. Keep weights in high precision (FP32)



∴ forward pass will have noise in data may not be useful for computation so, FP16 is fine but weights should be kept in FP32 for further computation

Transformers & Large Language Models

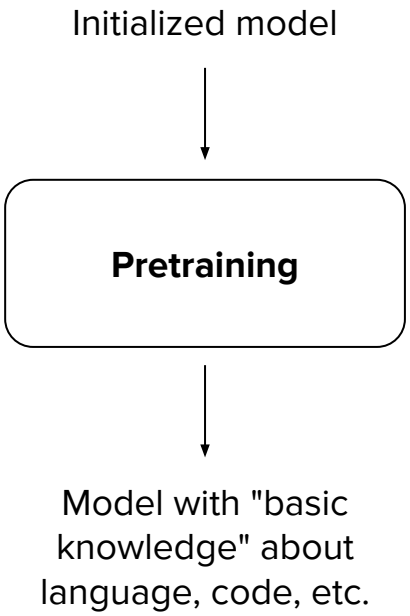
Pretraining

Training optimizations

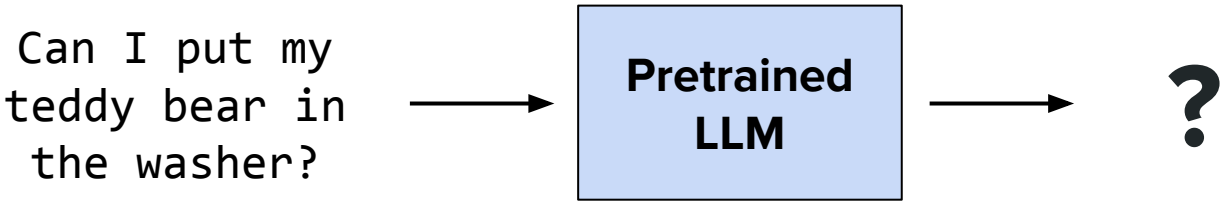
Supervised finetuning

Parameter-efficient finetuning

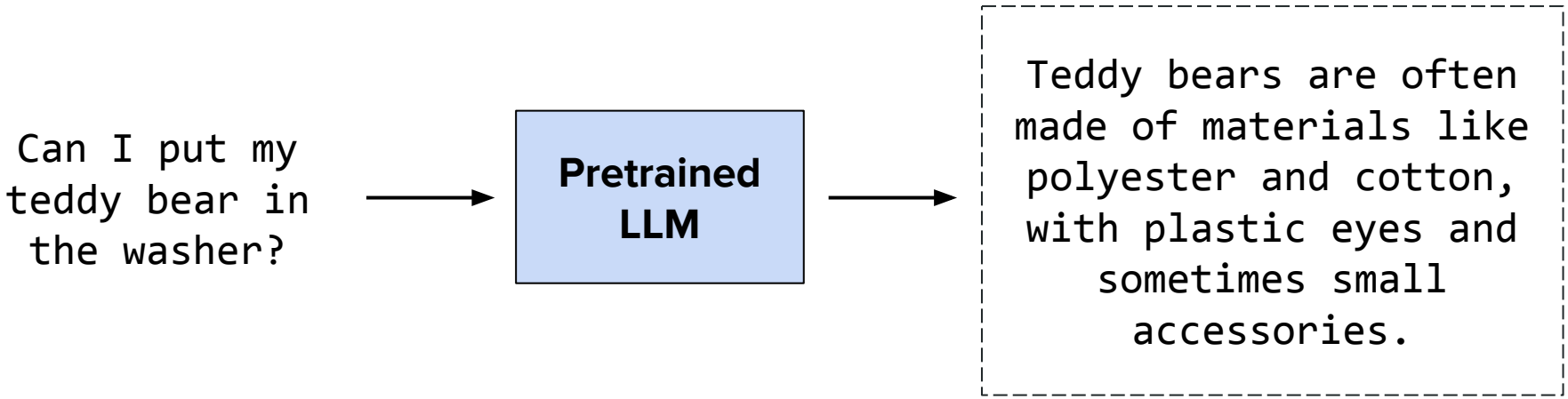
What we covered in the first part of this lecture



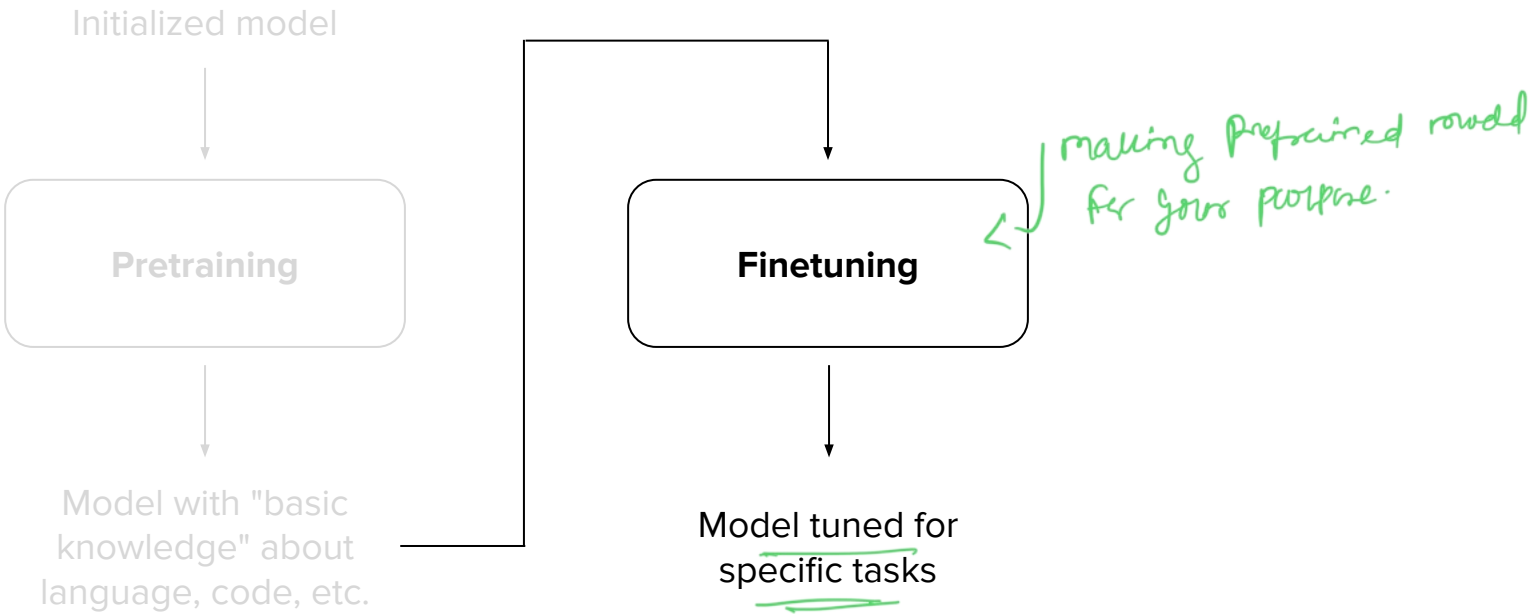
Pretrained model behavior



Pretrained model behavior



Remedy


Initialized model
Pretraining
Model with "basic knowledge" about language, code, etc.
Finetuning
making Pretrained model for your purpose.
Model tuned for specific tasks

Supervised finetuning

SFT = **S**upervised **F**ine**T**uning

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

but w/ labelled data
i.e. it is for specific task
⇒ additional data required

Strategy.

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

Supervised finetuning

SFT = **S**upervised **F**ine**T**uning

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.

"*Finetuned language models are zero-shot learners*", *Wei et al., 2022.*

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

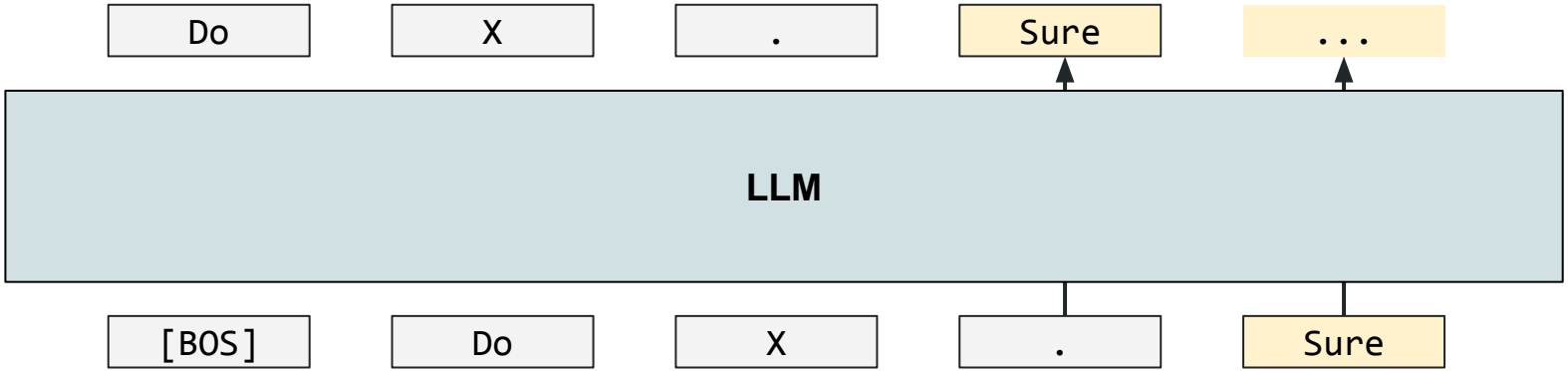
Explain why a teddy bear is a great friend.

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.*



"*Finetuned language models are zero-shot learners*", Wei et al., 2022.

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
- ...

Instruction tuning overview

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

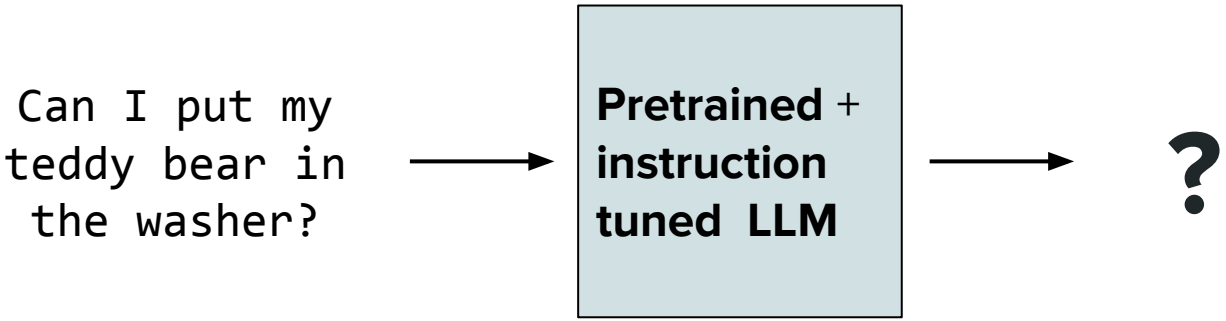
Objective function. Predict next token *given the input.*

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

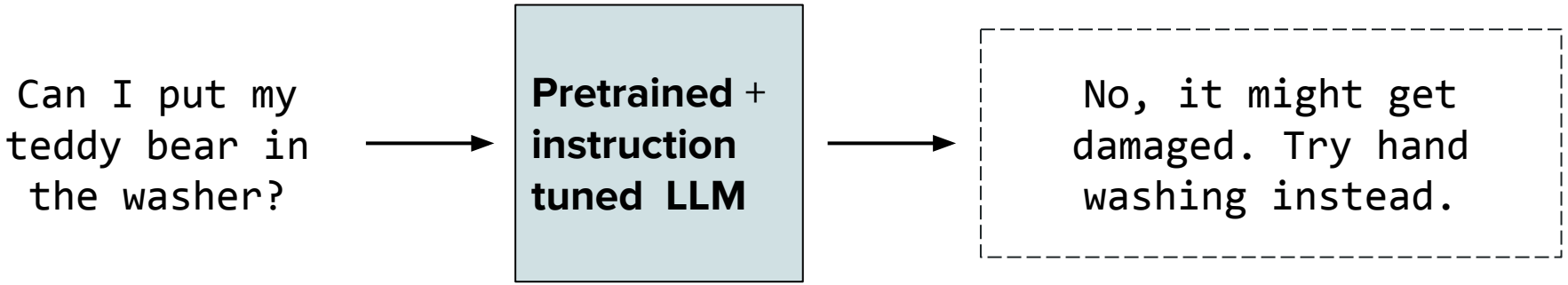
Size. Thousands to millions of examples.

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

Behavior


Can I put my teddy bear in the washer?
Pretrained + instruction tuned LLM
?

Behavior


Can I put my teddy bear in the washer?
Pretrained + instruction tuned LLM
No, it might get damaged. Try hand washing instead.

Challenges

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

Benchmarks

Dimensions.

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

Benchmarks

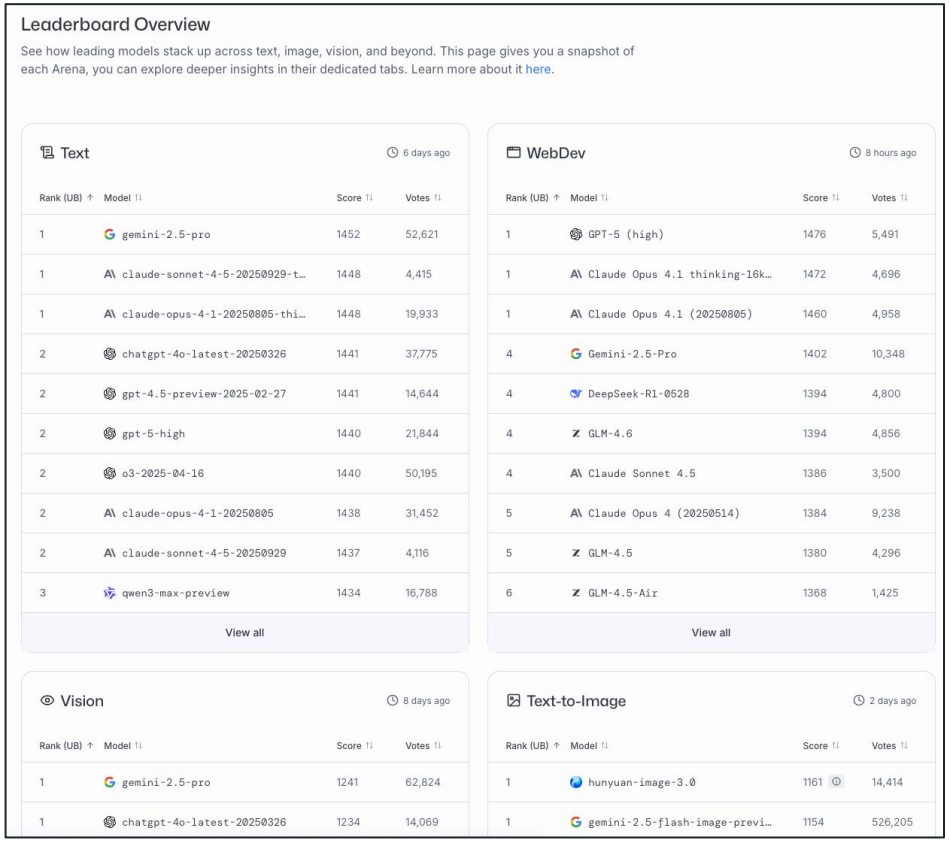
Dimensions.

- 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.

"*Training on the Test Task Confounds Evaluation and Emergence*", Dominguez-Olmedo et al., 2024.

"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

6 days ago

Rank (UB)	Model	Score	Votes
1	gemini-2.5-pro	1452	52,621
1	claude-sonnet-4-5-20250929-t…	1448	4,415
1	claude-opus-4-1-20250805-thi…	1448	19,933
2	chatgpt-4o-latest-20250326	1441	37,775
2	gpt-4.5-preview-2025-02-27	1441	14,644
2	gpt-5-high	1440	21,844
2	o3-2025-04-16	1440	50,195
2	claude-opus-4-1-20250805	1438	31,452
2	claude-sonnet-4-5-20250929	1437	4,116
3	qwen3-max-preview	1434	16,788

View all

WebDev

8 hours ago

Rank (UB)	Model	Score	Votes
1	GPT-5 (high)	1476	5,491
1	Claude Opus 4.1 thinking-16k…	1472	4,696
1	Claude Opus 4.1 (20250805)	1460	4,958
4	Gemini-2.5-Pro	1402	10,348
4	DeepSeek-R1-0528	1394	4,800
4	GLM-4.6	1394	4,856
4	Claude Sonnet 4.5	1386	3,500
5	Claude Opus 4 (20250514)	1384	9,238
5	GLM-4.5	1380	4,296
6	GLM-4.5-Air	1368	1,425

View all

Vision

8 days ago

Rank (UB)	Model	Score	Votes
1	gemini-2.5-pro	1241	62,824
1	chatgpt-4o-latest-20250326	1234	14,069

Text-to-Image

2 days ago

Rank (UB)	Model	Score	Votes
1	hunyuan-image-3.0	1161	14,414
1	gemini-2.5-flash-image-previ…	1154	526,205

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

Screenshot taken on Tuesday, October 14, 2025 from LMArena > Leaderboard Overview, lmarena.ai/leaderboard.

"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 → if query rejected by LLM

"Exploring and Mitigating Adversarial Manipulation of Voting-Based Leaderboards", Huang et al., 2025.

"Real-life" feeling

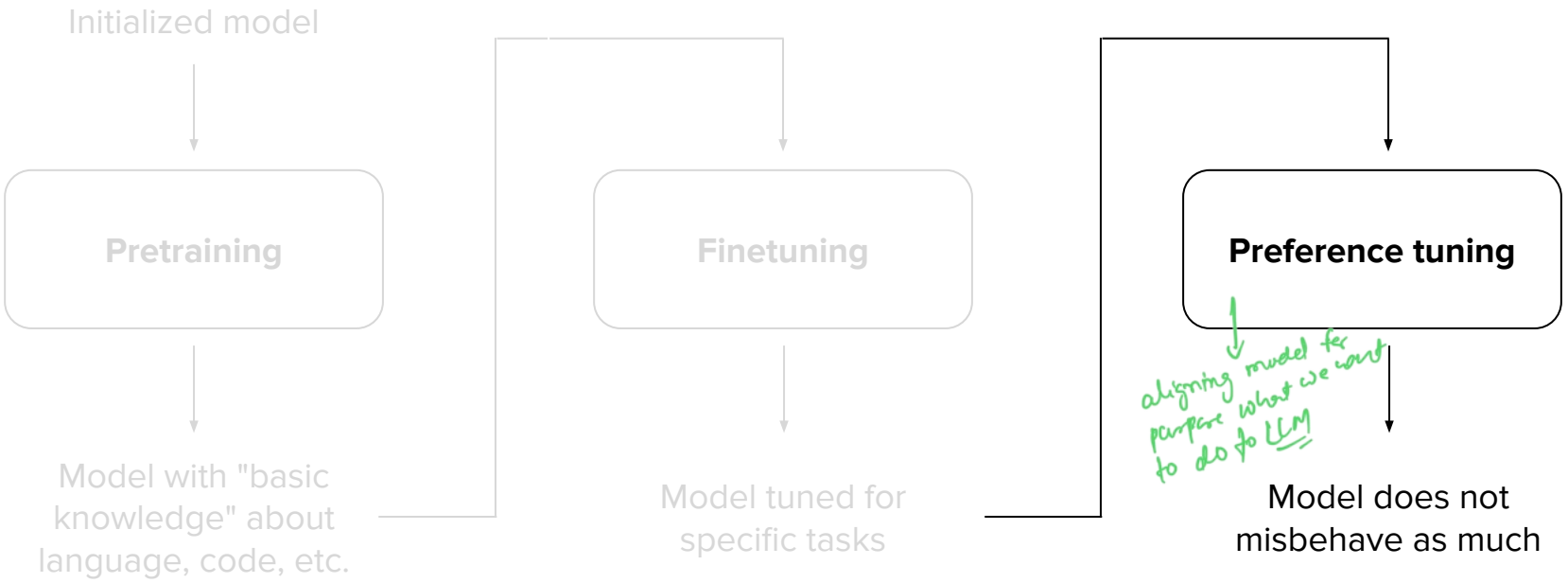
Benefits. Puts a number on "vibes"

Outstanding challenges.

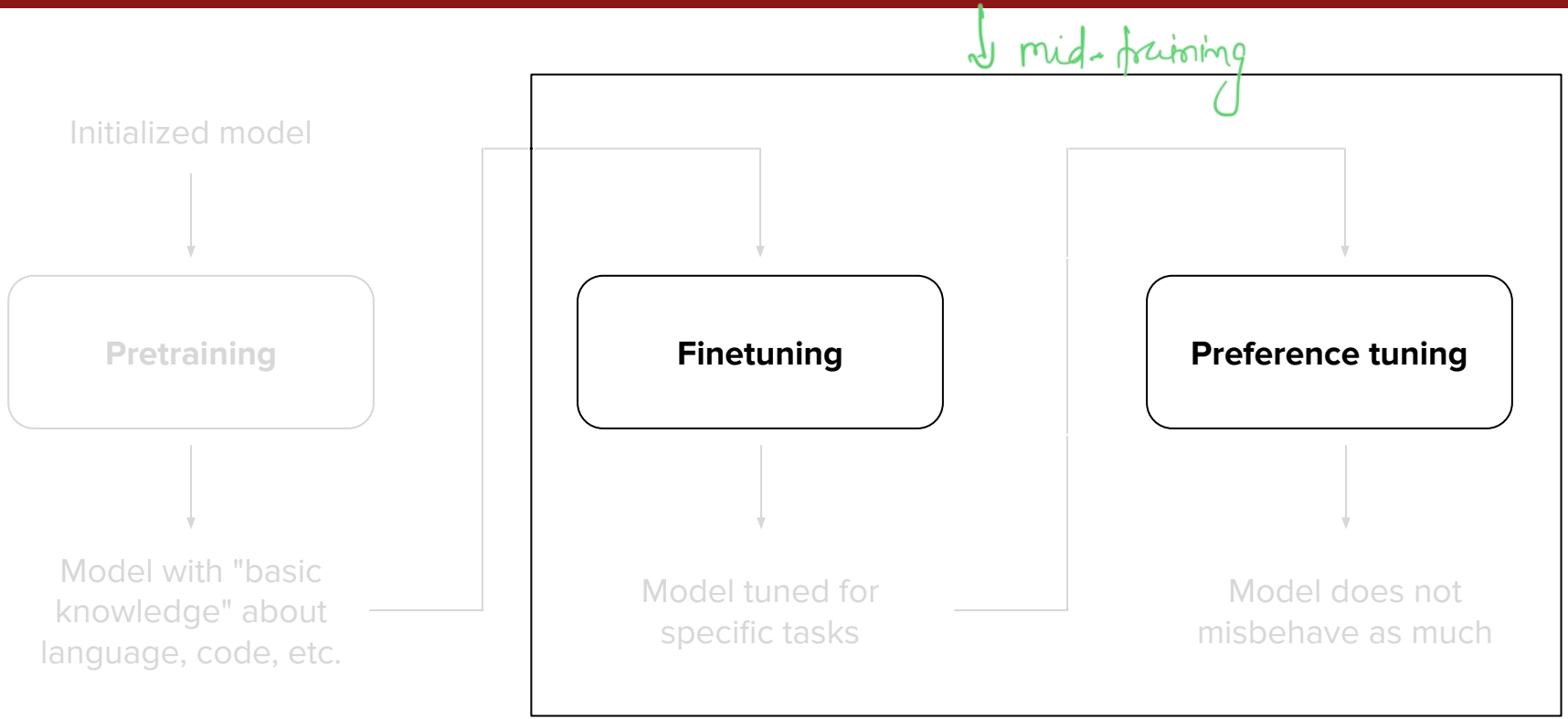
- Unequal exposure of models/"cold start" problem
- Easy to "rig"
- User inability to accurately assess important aspects, e.g. factuality
- Personal preference bias (non-representative distribution)
- Safety penalization

...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

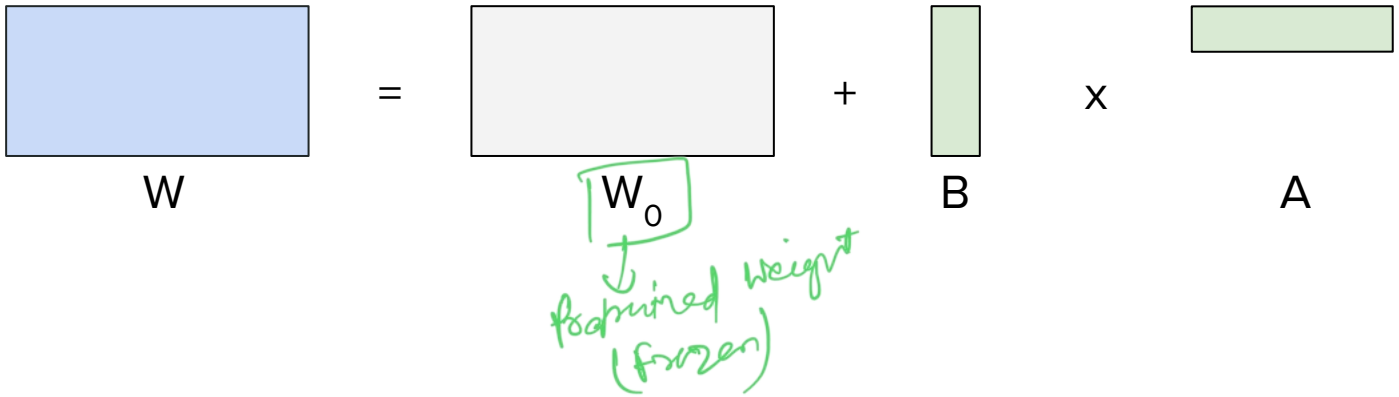
Parameter-efficient finetuning

Parameter-efficient finetuning with LoRA

fine-tuning weight for efficient manner which reduces the computation

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

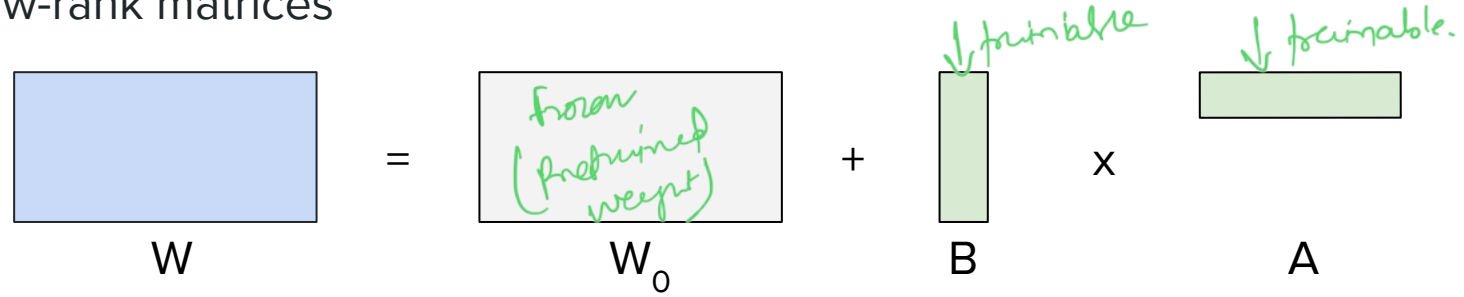


"LoRA: Low-Rank Adaptation of Large Language Models", Hu et al., 2021.

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



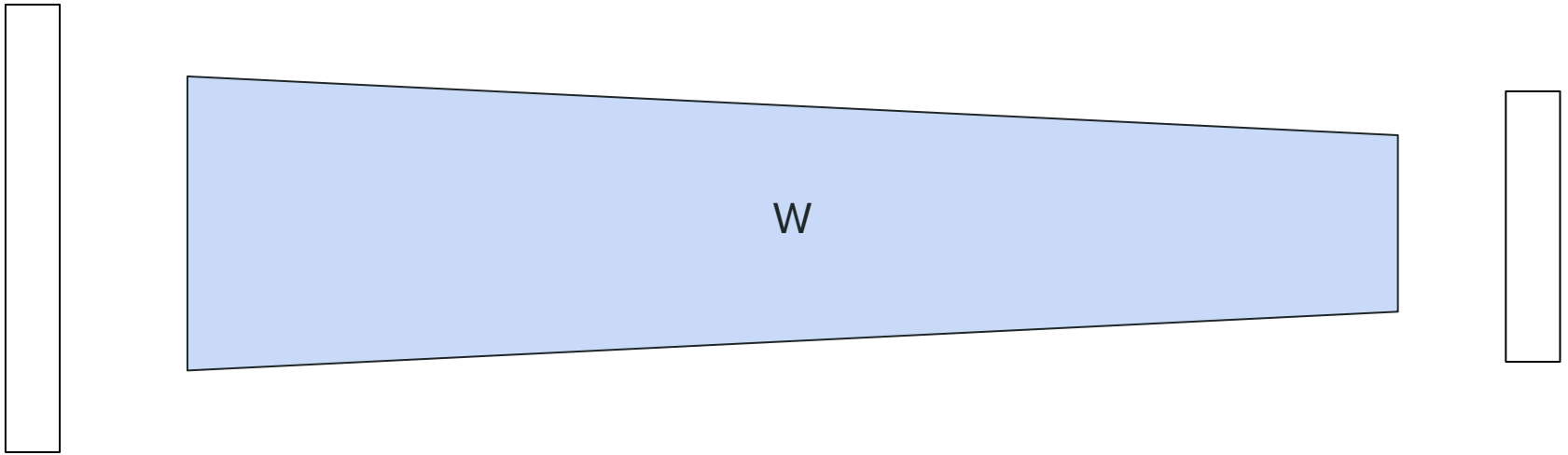
Discussion.

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

"LoRA: Low-Rank Adaptation of Large Language Models", Hu et al., 2021.

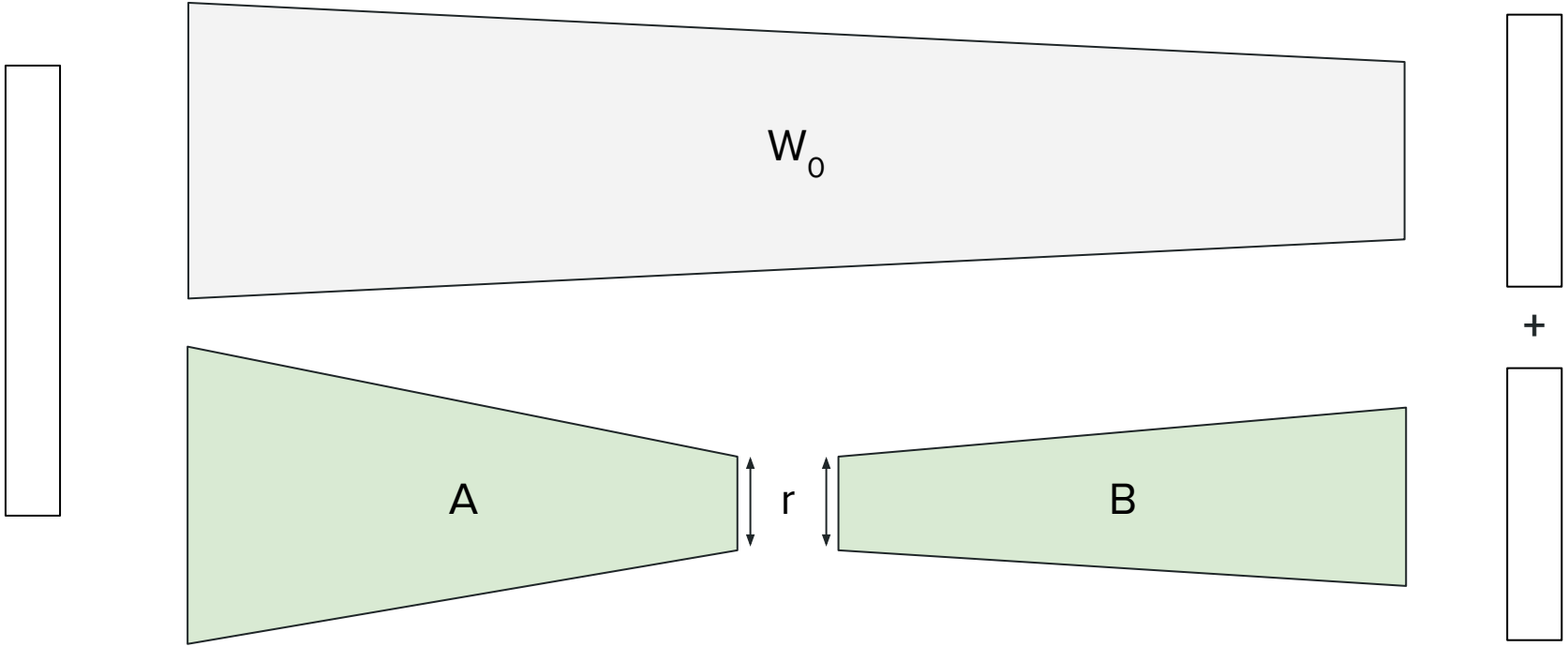
Finetuning evolution

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



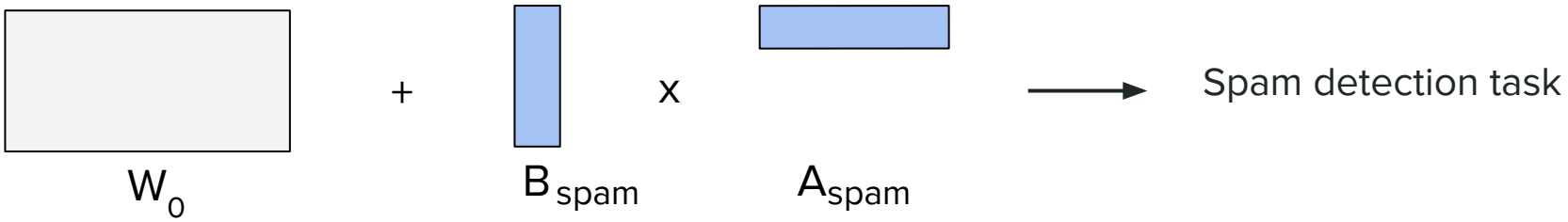
Finetuning evolution

After. LoRA finetuning optimizes the full matrix of weights

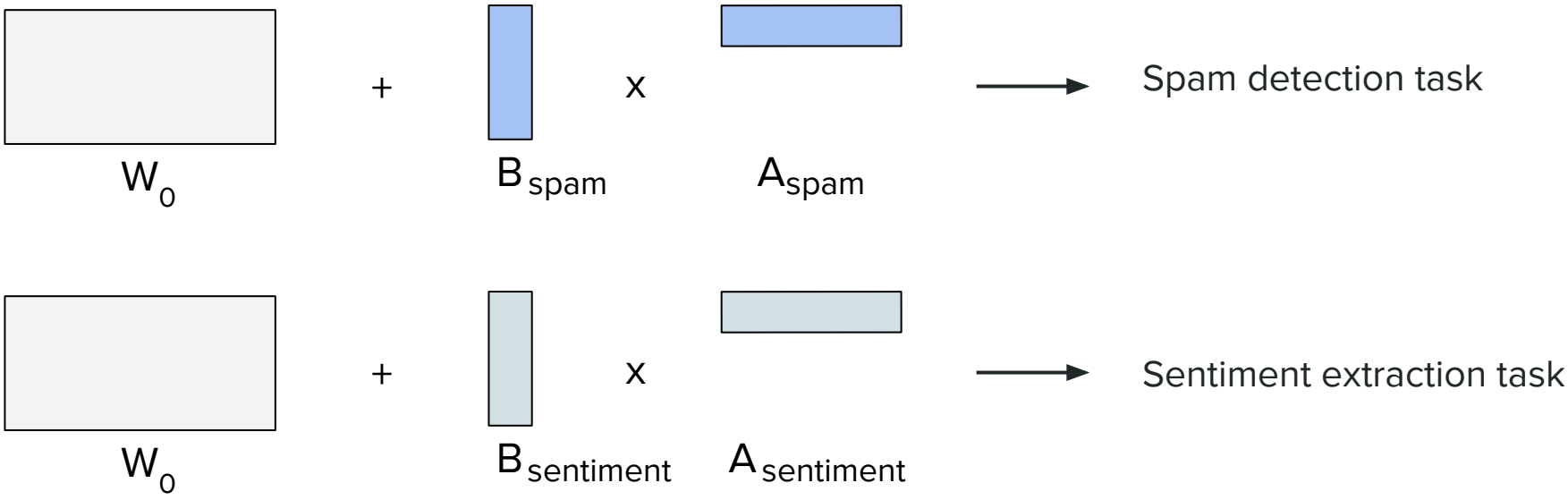


"*LoRA: Low-Rank Adaptation of Large Language Models*", *Hu et al., 2021.*

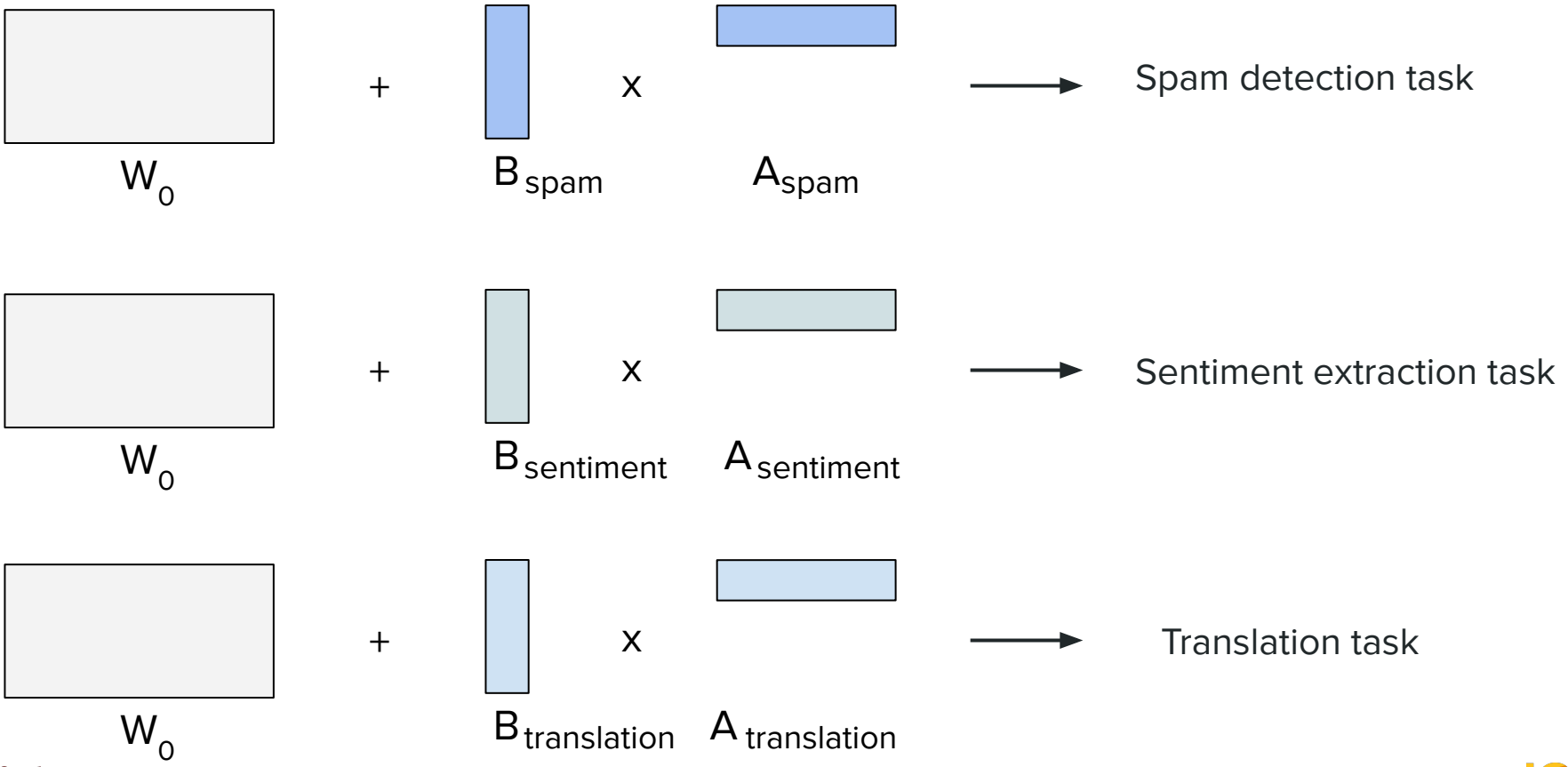
Benefit of LoRA: swap matrices = swap tasks



Benefit of LoRA: swap matrices = swap tasks

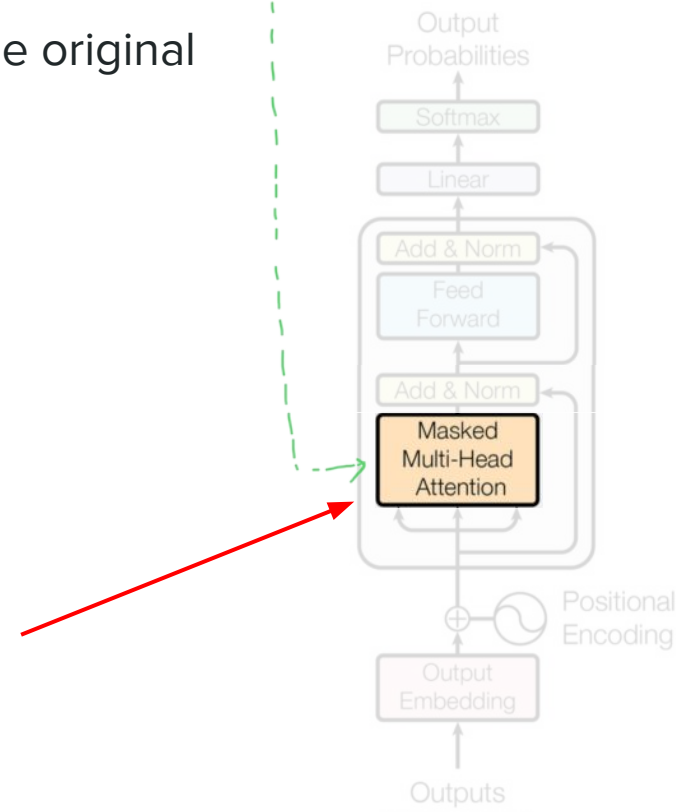


Benefit of LoRA: swap matrices = swap tasks



Locations where to apply LoRA ?

Originally. As experimented in the original paper.

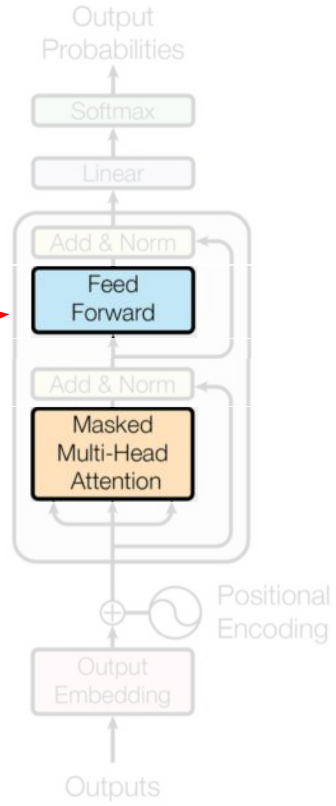


"LoRA: Low-Rank Adaptation of Large Language Models", Hu et al., 2021.
Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

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

- 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

empirical

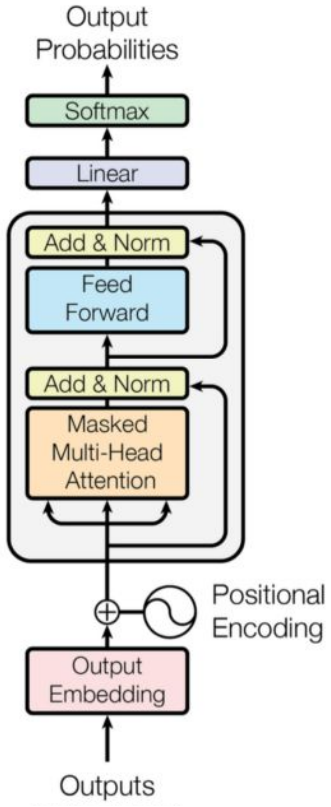
- ✔ LoRA does poorly on large batch size compared to full fine-tuning

empirical

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

QLoRA

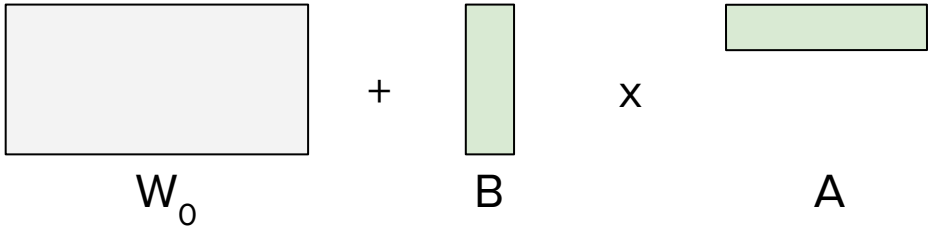
Idea. Quantize all frozen weights to relieve memory bottleneck.

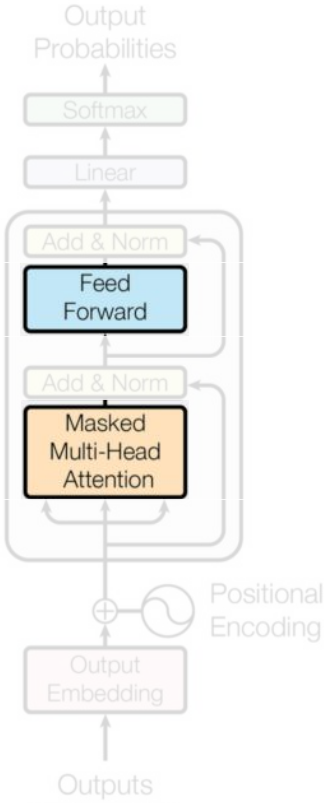


QLoRA: Efficient Finetuning of Quantized LLMs, Dettmers et al., 2023.
Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

QLoRA

Idea. Quantize all frozen weights to relieve memory bottleneck.

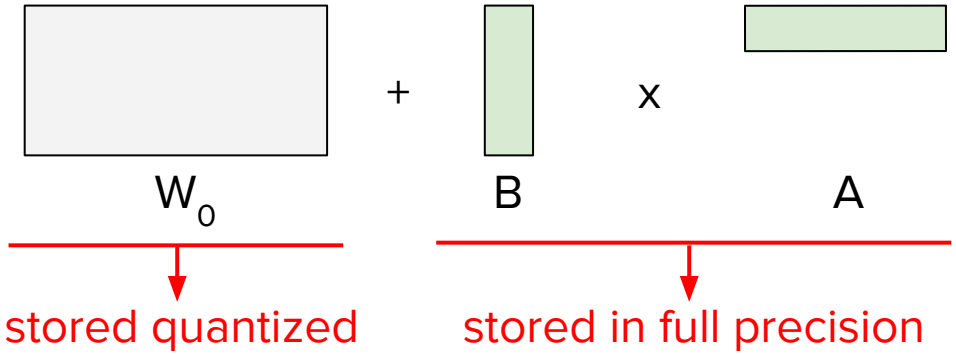


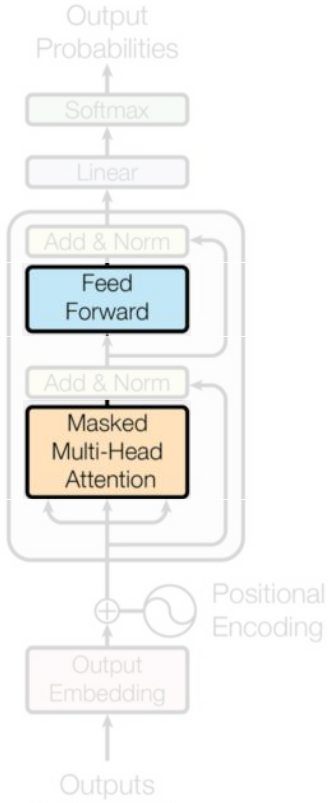


QLoRA: Efficient Finetuning of Quantized LLMs, Dettmers et al., 2023.
Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

QLoRA

Idea. Quantize all frozen weights to relieve memory bottleneck.

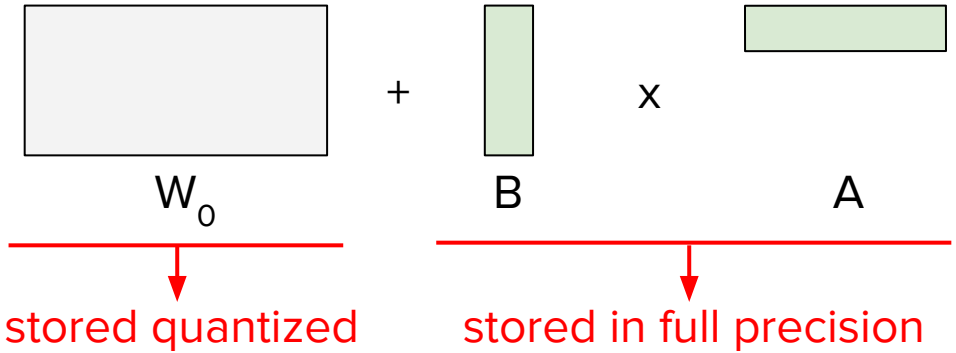




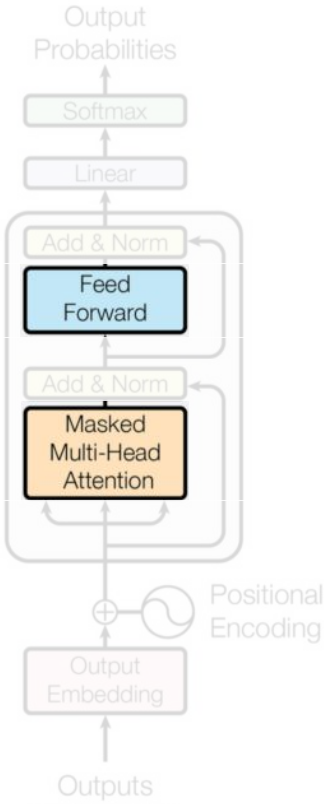
QLoRA: Efficient Finetuning of Quantized LLMs, Dettmers et al., 2023.
Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

QLoRA

Idea. Quantize all frozen weights to relieve memory bottleneck.



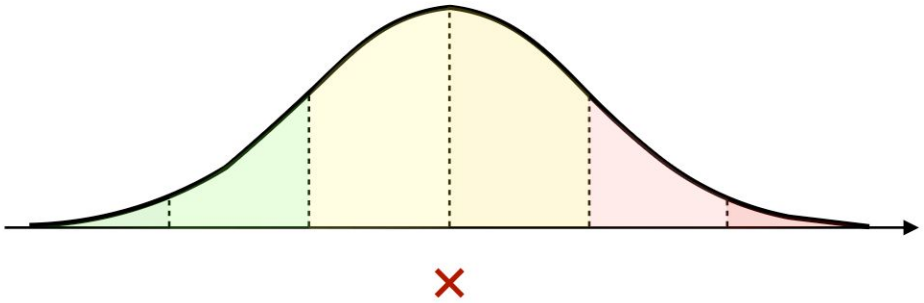
computations in full precision



QLoRA: Efficient Finetuning of Quantized LLMs, Dettmers et al., 2023.
Figure adapted from "Attention is All You Need", Vaswani et al., 2017.

Efficient quantization

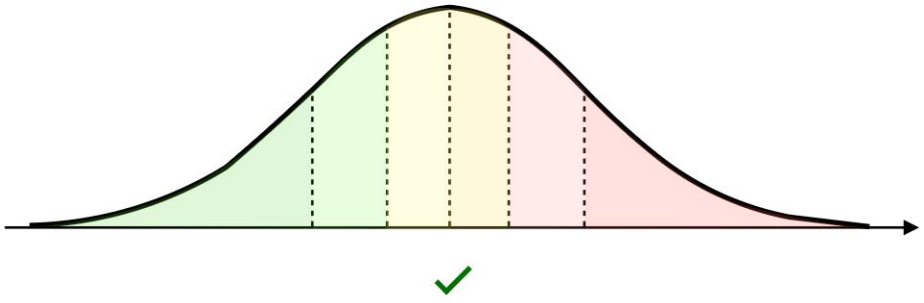
Trick. Use 4-bit NormalFloat (NF4) to best split the space



✕

Uniform distribution cutoffs

`INT8`



✓

Based on Normal quantiles

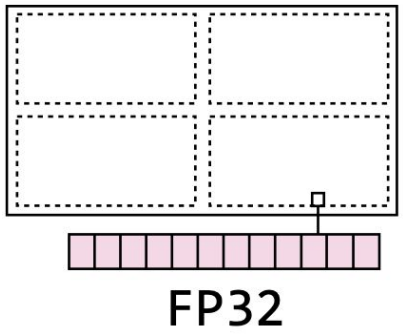
`NF4`

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

Quantization...

No quantization

Weights



Quantization constants

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

Quantization...

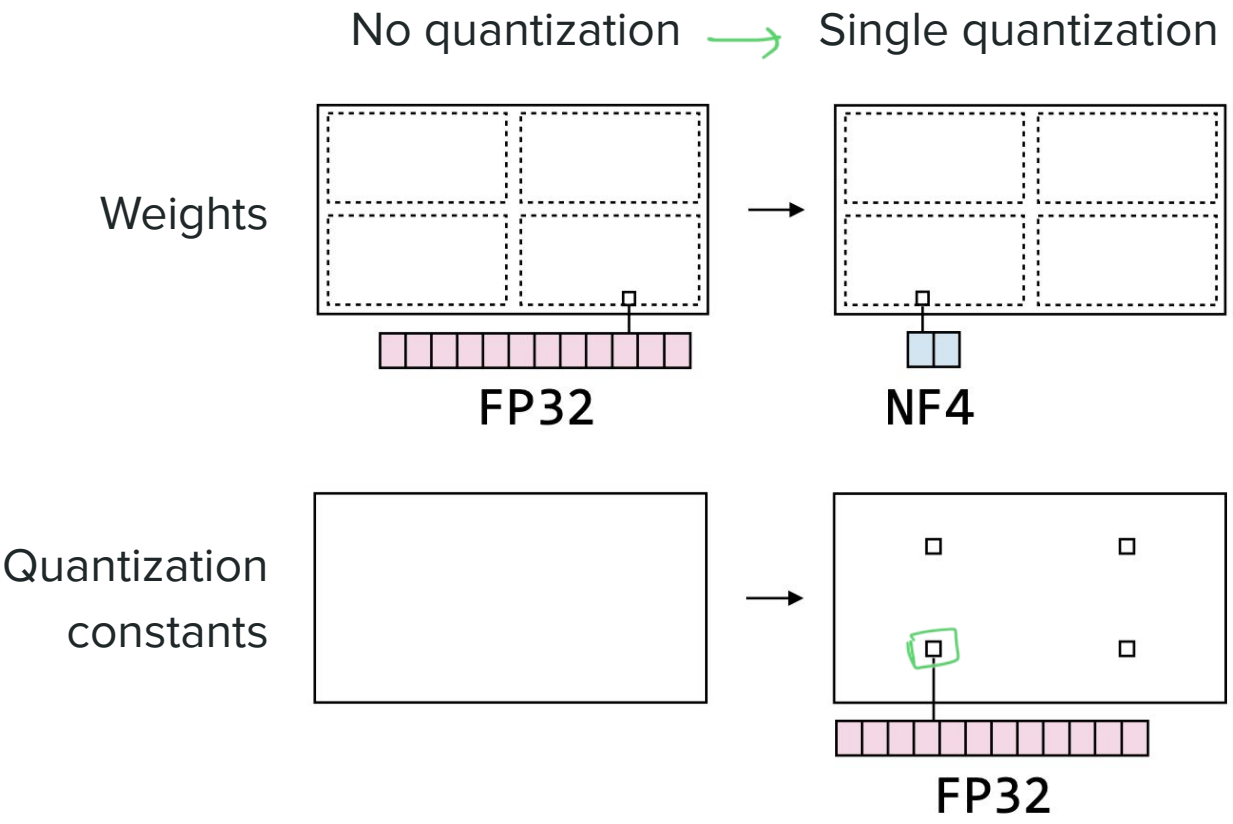


Figure from "Super Study Guide: Transformers and Large Language Models", Amidi et al., 2024.

Quantization...

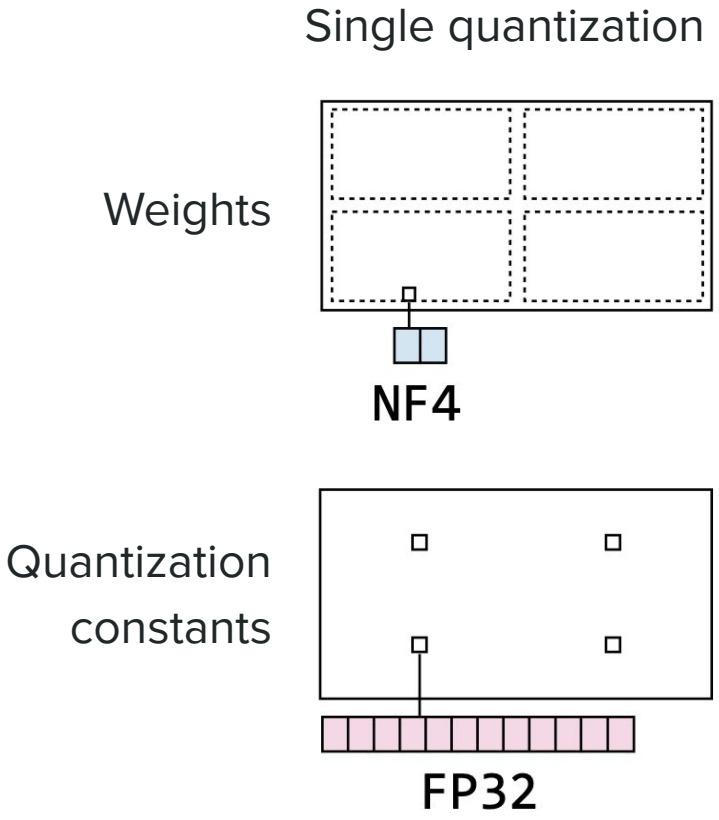


Figure from "Super Study Guide: Transformers and Large Language Models", Amidi et al., 2024.

...done two times!

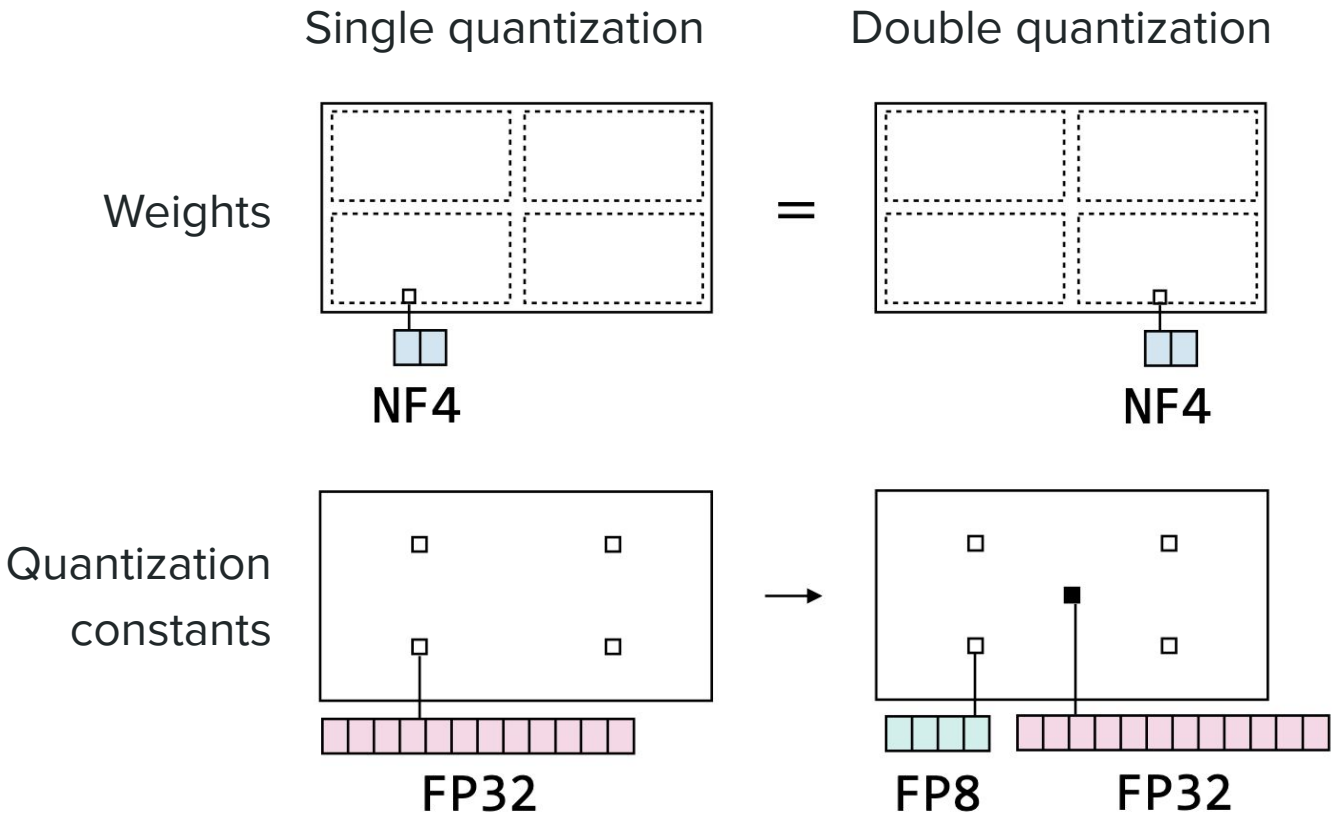


Figure from "Super Study Guide: Transformers and Large Language Models", Amidi et al., 2024.

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: Efficient Finetuning of Quantized LLMs, Dettmers et al., 2023.

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%

QLoRA: Efficient Finetuning of Quantized LLMs, *Dettmers et al., 2023.*

Thank you for your attention!