Efficient training (cont'd) & inference

CS 6804: Frontier AI Systems

Spring 2026

https://tuvllms.github.io/ai-seminar-spring-2026/

Tu Vu

VIRGINIA TECH

Logistics

- Homework assignments
  - Homework 0 & 1, due 3/3 & 3/10
    - 5% extra credits each
- Student presentation groups confirmed

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

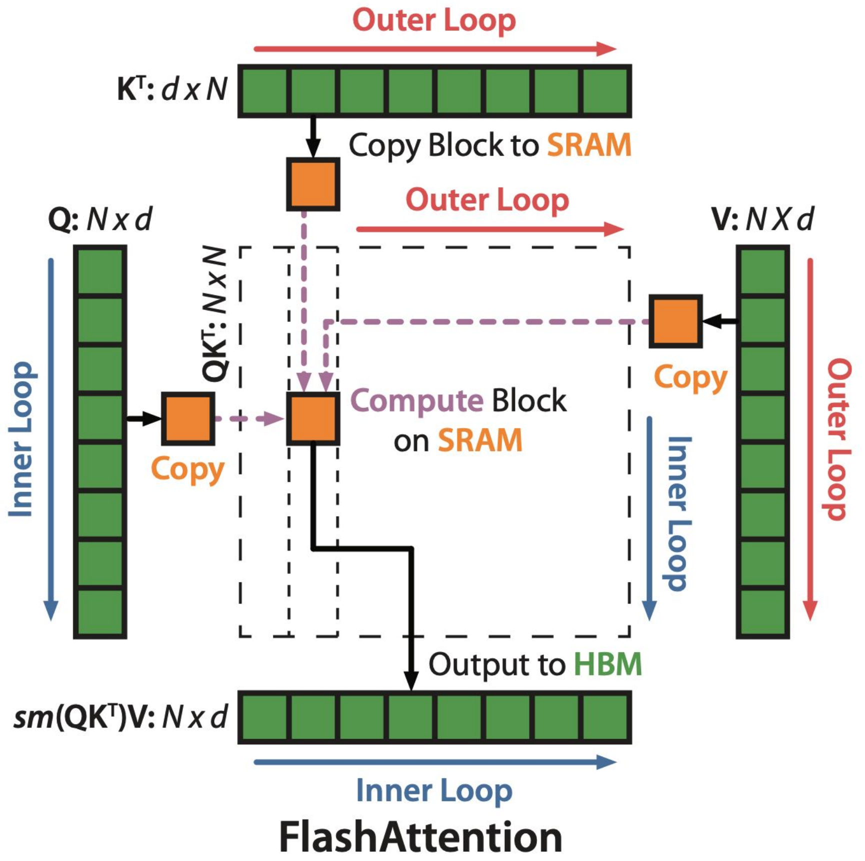
[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY

{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu,
chrismre@cs.stanford.edu

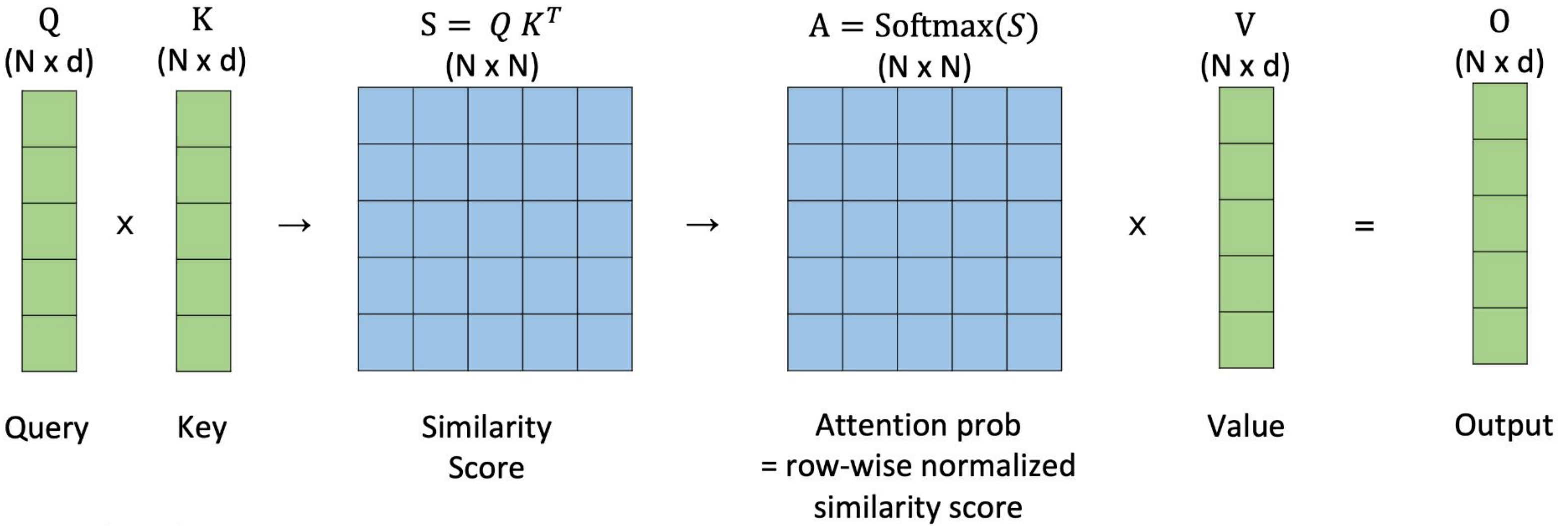
FlashAttention

Massive adoption

- **Tiling** and **recomputation** to reduce GPU memory IOs
  - **Fast** (3x) and **memory efficient** (10-20x) algorithm for **exact** attention
  - **Longer sequences** (up to 16K) yield **higher quality**



Attention mechanism review (cont'd)

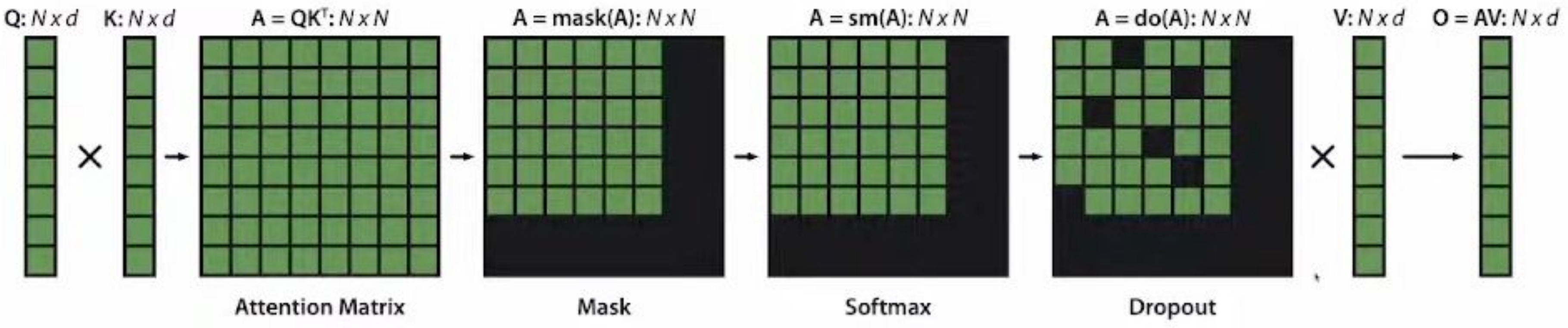


Typical sequence length N: 1K – 8K
Head dimension d: 64 – 128

$$\mathrm{Softmax}([s_1, \cdots, s_N]) = \left[\frac{e^{s_1}}{\sum_i e^{s_i}}, \cdots, \frac{e^{s_N}}{\sum_i e^{s_i}}\right]$$

$$O = \mathrm{Softmax}(QK^T)V$$

Attention mechanism review (cont'd)



$$\mathbf{O} = \mathrm{Dropout}(\mathrm{Softmax}(\mathrm{Mask}(\mathbf{QK^T})))\mathbf{V}$$

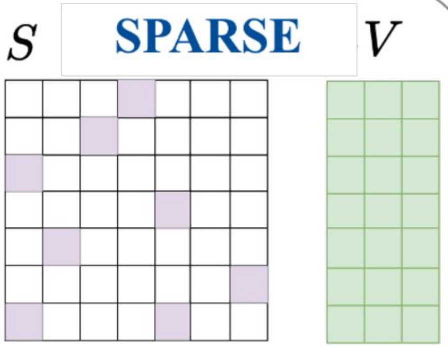
[FlashAttention – Tri Dao | Stanford MLSys #67]

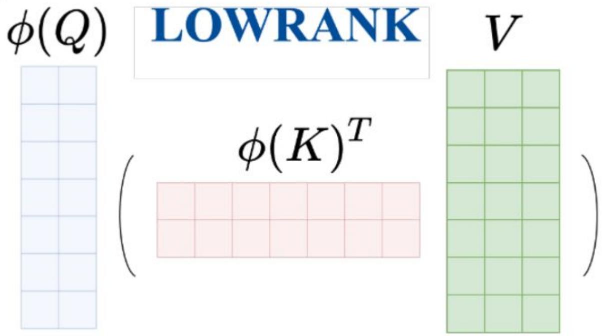
Approximate attention

tradeoff quality for ~~speed~~ fewer FLOPs

does not result in an actual wall clock speedup

Sparse Transformer
(Child et al. 19)
Reformer
(Kitaev et al. 20)
Routing Transformer
(Roy et al. 20)

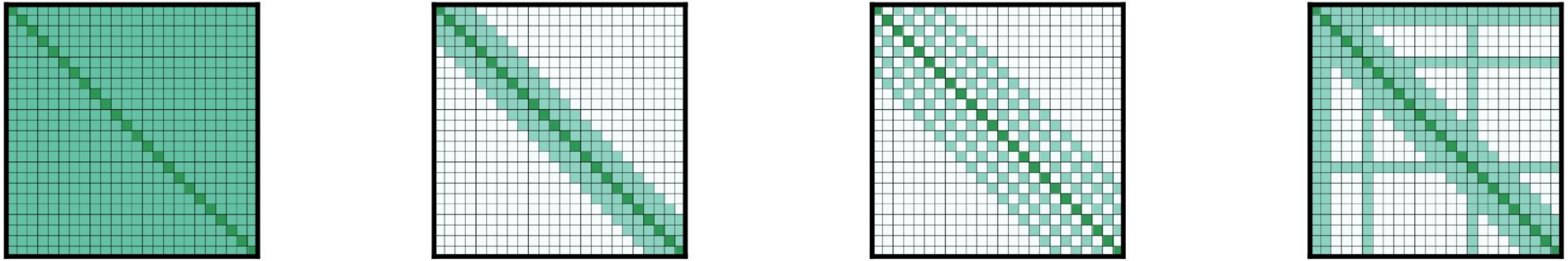




Linformer
(Wang et al. 20)
Linear Transformer
(Katharopoulos et al. 20)
Performer
(Choromanski et al. 20)

[FlashAttention – Tri Dao | Stanford MLSys #67]

Approximate attention



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

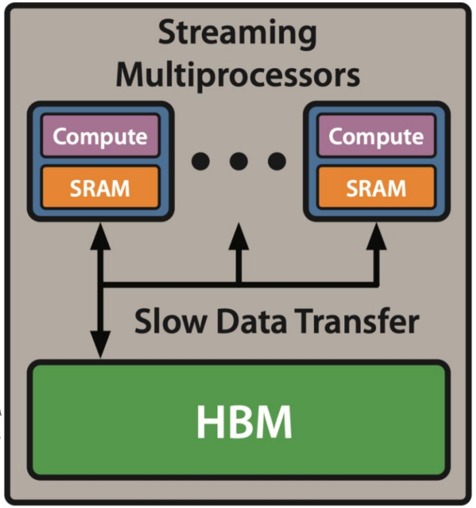
[Longformer: The Long-Document Transformer]

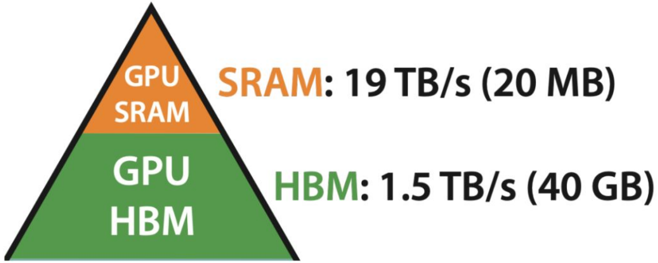
GPU compute model & memory hierarchy

2. Data moved to compute units & SRAM for computation

1. Inputs start out in HBM (GPU memory)

3. Output written back to HBM

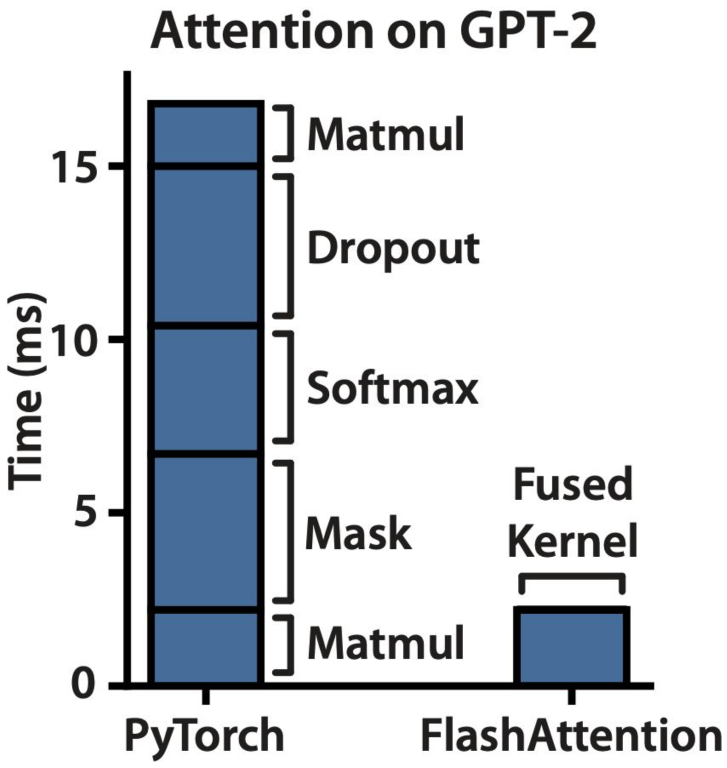




Can we exploit the memory asymmetry to get speed up?

[FlashAttention – Tri Dao | Stanford MLSys #67]

Data movement is the key bottleneck

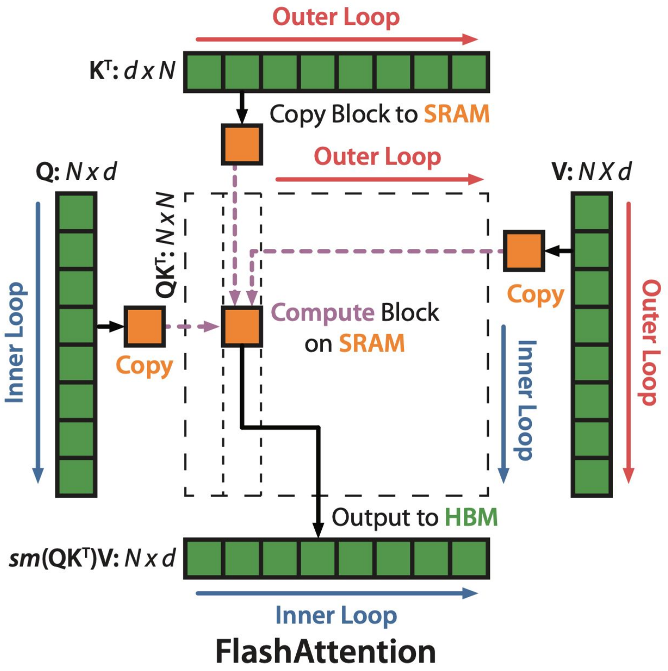


How to reduce HBM reads/writes: compute by blocks

- **Challenges:**
  - Compute softmax normalization without access to full input
  - Backward without the large attention matrix from forward
- **Approaches:**
  - **Tiling:** Restructure algorithm to load block by block from HBM to SRAM to compute attention
  - **Recomputation:** Don't store attention matrix from forward, recompute it in the backward

Tiling

- **Decomposing large softmax into smaller ones by scaling**



$$\text{softmax}([A_1, A_2]) = [\alpha \times \text{softmax}(A_1), \beta \times \text{softmax}(A_2)]$$

$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha \times \text{softmax}(A_1)V_1 + \beta \times \text{softmax}(A_2)V_2$$

[FlashAttention – Tri Dao | Stanford MLSys #67]

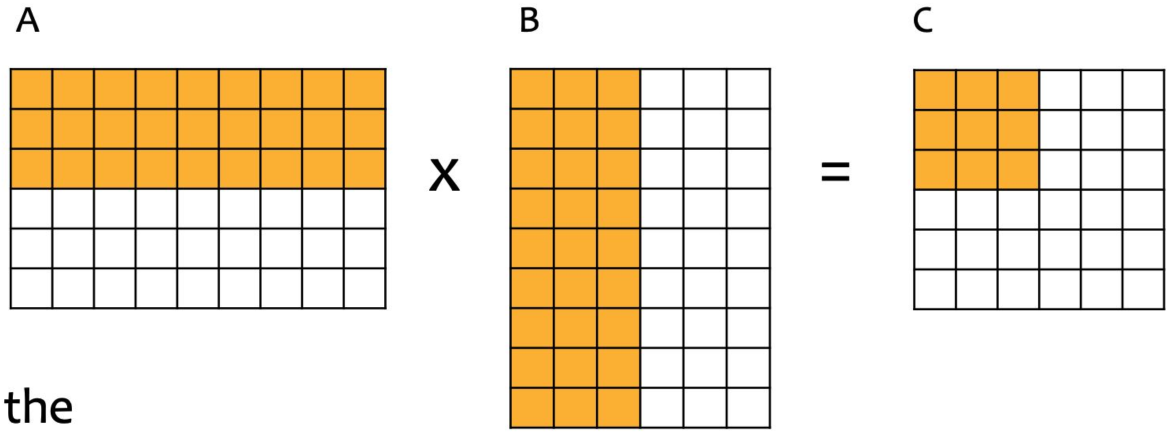
$$\operatorname{softmax}([a,b,c,d,e]) = \left[\frac{e^a}{e^a+e^b+e^c+e^d+e^e}, \frac{e^b}{e^a+e^b+e^c+e^d+e^e}, \frac{e^c}{e^a+e^b+e^c+e^d+e^e}, \frac{e^d}{e^a+e^b+e^c+e^d+e^e}, \frac{e^e}{e^a+e^b+e^c+e^d+e^e}\right]$$

$$\operatorname{softmax}([a,b,c,d,e]) = \left[\frac{e^a+e^b+e^c}{e^a+e^b+e^c+e^d+e^e} \cdot \left(\frac{e^a}{e^a+e^b+e^c}; \frac{e^b}{e^a+e^b+e^c}; \frac{e^c}{e^a+e^b+e^c}\right); \frac{e^d+e^e}{e^a+e^b+e^c+e^d+e^e} \cdot \left(\frac{e^d}{e^d+e^e}; \frac{e^e}{e^d+e^e}\right)\right]$$

; denotes concatenation
note that the terms involving $e^a + e^b + e^c$ cancel out each other
same for the $e^d + e^e$ terms

$$\operatorname{softmax}(\underbrace{[a,b,c,d,e]}_{A}) = \left[\underbrace{\frac{e^a+e^b+e^c}{e^a+e^b+e^c+e^d+e^e}}_{\alpha} \cdot \operatorname{softmax}(\underbrace{[a,b,c]}_{A_1}); \underbrace{\frac{e^d+e^e}{e^a+e^b+e^c+e^d+e^e}}_{\beta} \cdot \operatorname{softmax}(\underbrace{[d,e]}_{A_2})\right]$$

Tiling for matrix multiplication

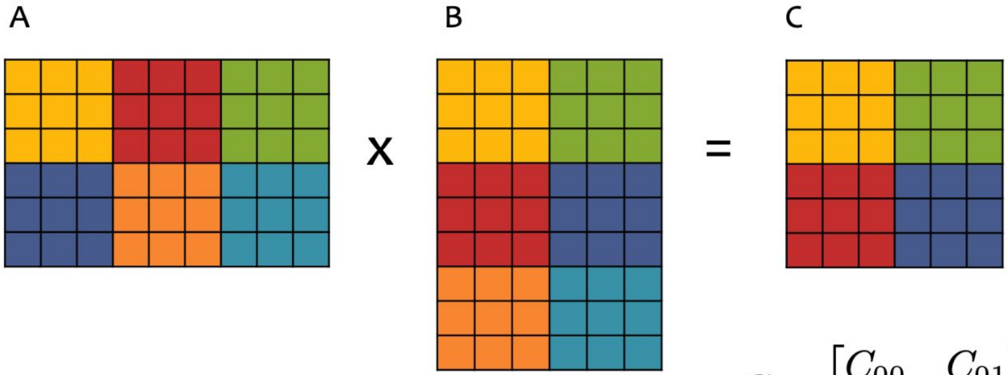


- We can view the computation as decomposing if we consider subsets of rows/columns

$$C_{(1,1):(3,3)} = A_{(1,1):(3,9)} \times B_{(1,1):(9,3)}$$

Tiling for matrix multiplication (cont'd)

- Tiling capitalizes on this decomposition
- Each output tile is computed by multiplying a pair of input tiles and adding it to the appropriate output tile



$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \end{bmatrix}$$

with each $A_{ij} \in \mathbb{R}^{3\times 3}$

$$B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \\ B_{20} & B_{21} \end{bmatrix}$$

with each $B_{ij} \in \mathbb{R}^{3\times 3}$

$$C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

with each $C_{ij} \in \mathbb{R}^{3\times 3}$

$$C_{00} = A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$$

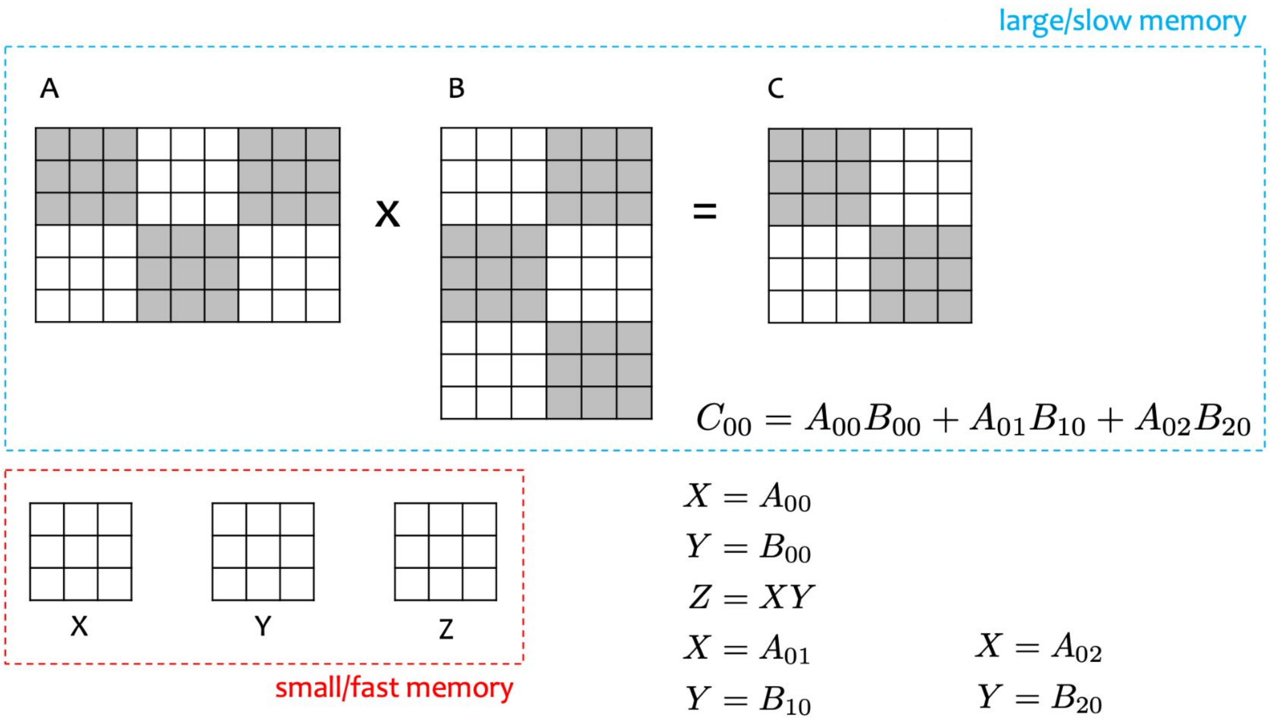
$$C_{01} = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$$

$$C_{10} = A_{10}B_{00} + A_{11}B_{10} + A_{12}B_{20}$$

$$C_{11} = A_{10}B_{01} + A_{11}B_{11} + A_{12}B_{21}$$

Tiling for matrix multiplication (cont'd)

- Tiling enables matrix multiplication of two **very** large matrices to capitalize on the small amount of fast memory on a device (e.g. GPU)
- Start by putting the input matrices and storage for the output matrix into large/slow memory
- Do the primary computation in slow/fast memory



$$X = A_{00}$$
$$Y = B_{00}$$
$$Z = XY$$

$$X = A_{01}$$
$$Y = B_{10}$$
$$Z = Z + XY$$

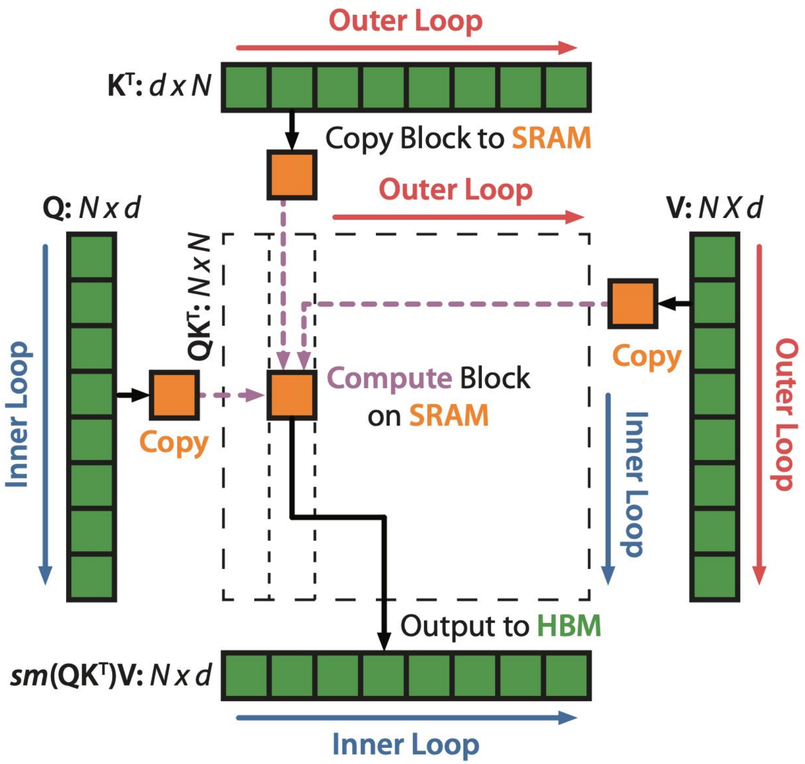
$$X = A_{02}$$
$$Y = B_{20}$$
$$Z = Z + XY$$
$$C_{00} = Z$$

Tiling (cont'd)

1. Load inputs by blocks from HBM to SRAM.

2. On chip, compute attention output with respect to that block.

3. Update output in HBM by scaling.



FlashAttention

[FlashAttention – Tri Dao | Stanford MLSys #67]

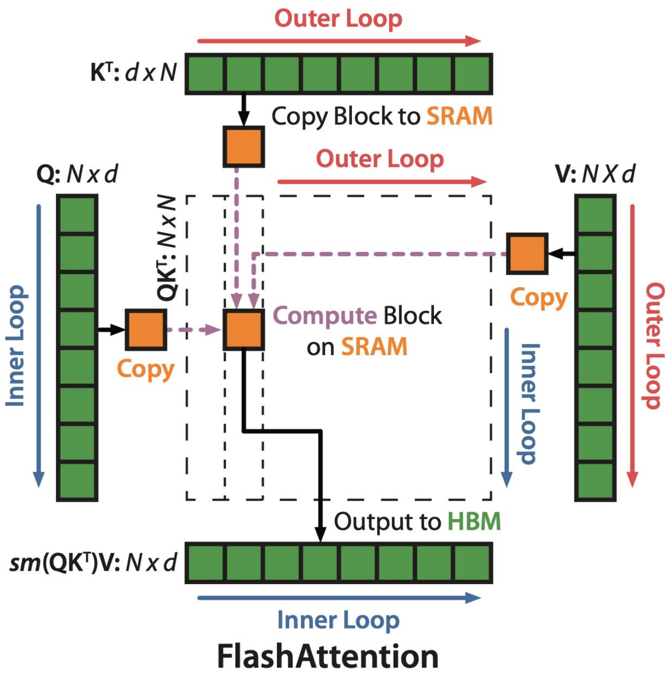
Demo

- https://jacksoncakes.com/flashattention-fast-and-memory-efficient-exact-attention/

Recomputation (backward pass)

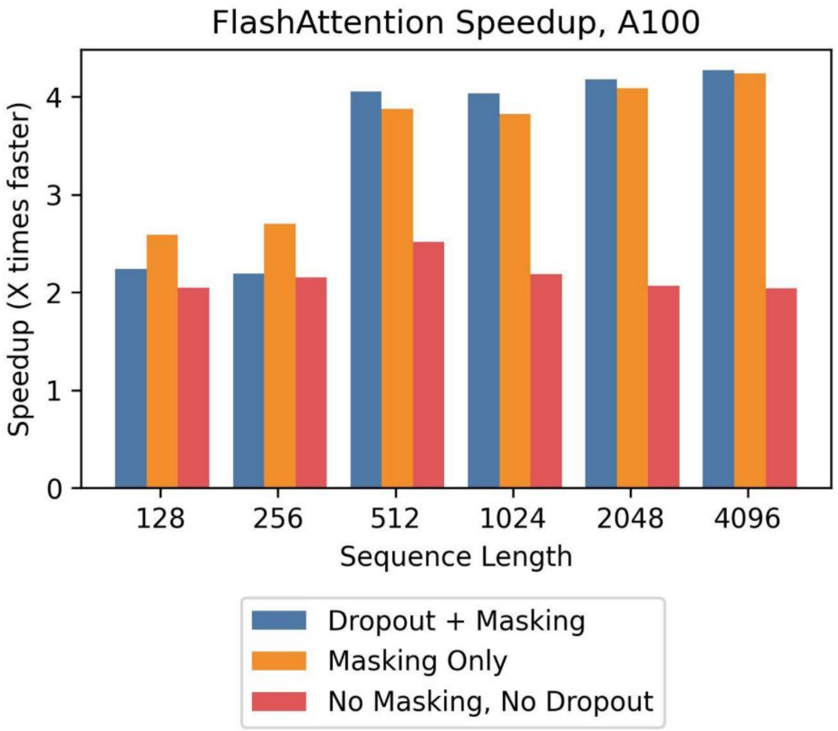
- **By storing softmax normalization from forward (size N), quickly recompute attention in the backward from inputs in SRAM.**

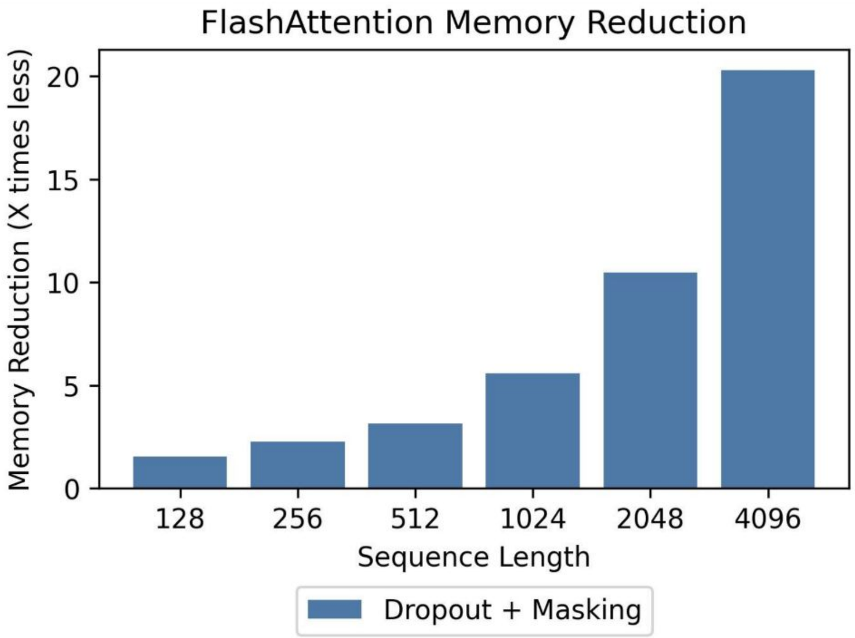
Attention	Standard	FlashAttention
GFLOPs	66.6	75.2 (↑13%)
HBM reads/writes (GB)	40.3	4.4 (↓9x)
Runtime (ms)	41.7	7.3 (↓6x)



[FlashAttention – Tri Dao | Stanford MLSys #67]

FlashAttention: 2-4x speedup, 10-20x memory reduction





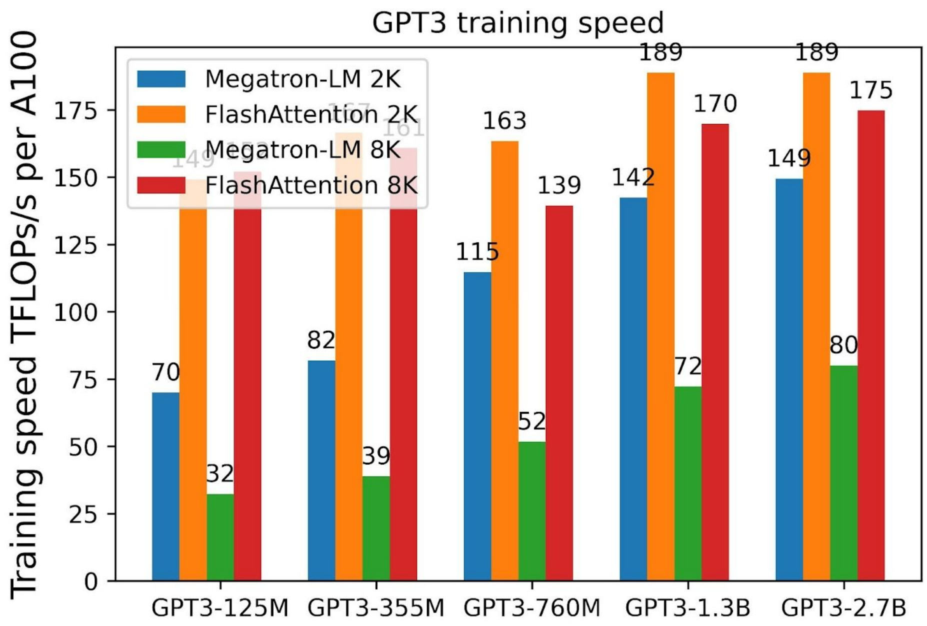
[FlashAttention – Tri Dao | Stanford MLSys #67]

Faster Training: MLPerf Record for Training BERT-large

- MLPerf: (highly optimized) standard benchmark for training speed
- Time to hit an accuracy of 72.0% on MLM from a fixed checkpoint, averaged across 10 runs on 8 x A100 GPUs

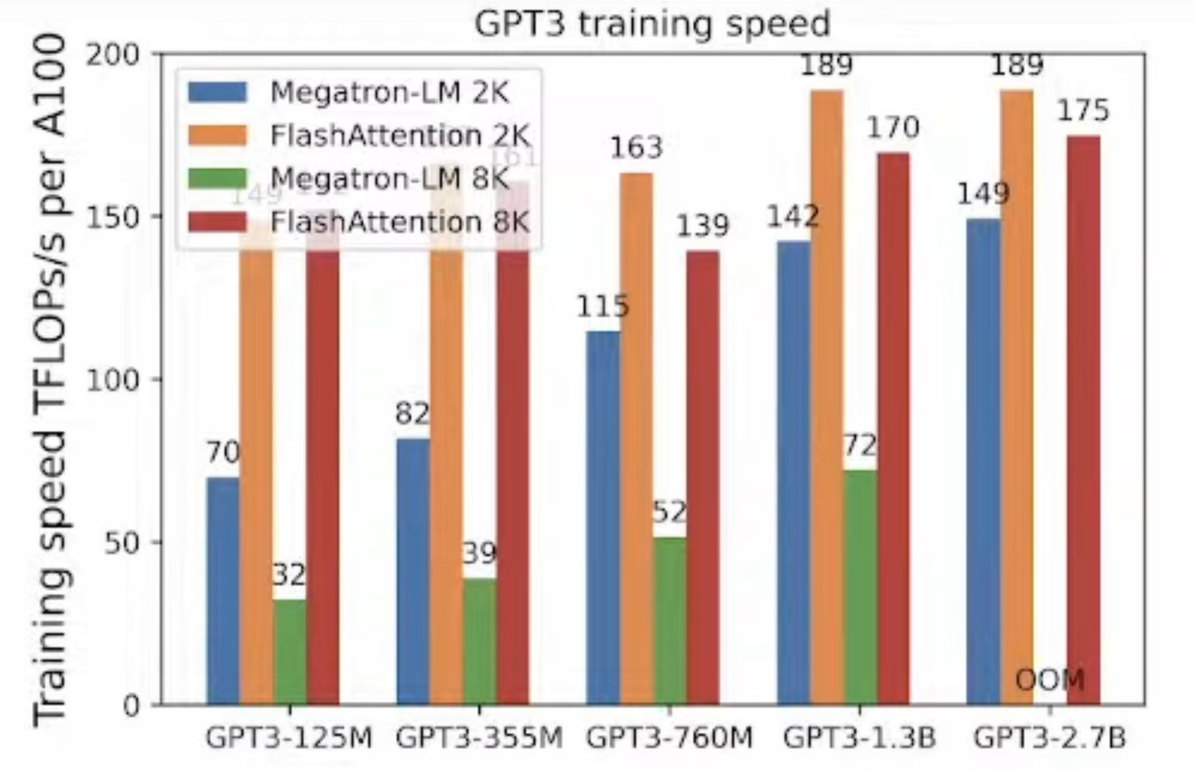
BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [58]	20.0 ± 1.5
FlashAttention (ours)	$\mathbf{17.4} \pm 1.4$

Faster Training, longer context



FlashAttention – Tri Dao | Stanford MLSys #67

Faster Training, longer context



[FlashAttention – Tri Dao | Stanford MLSys #67]

End-to-End Test-Time Training for Long Context

Arnuv Tandon*[1,3], Karan Dalal*[1,4], Xinhao Li*[5], Daniel Koceja*[3], Marcel Rød*[3], Sam Buchanan[4], Xiaolong Wang[5], Jure Leskovec[3], Sanmi Koyejo[3], Tatsunori Hashimoto[3], Carlos Guestrin[3], Jed McCaleb[1], Yejin Choi[2], Yu Sun*[2,3]

[1] Astera Institute [2] NVIDIA [3] Stanford University [4] UC Berkeley [5] UC San Diego

Abstract

We formulate long-context language modeling as a problem in continual learning rather than architecture design. Under this formulation, we only use a standard architecture – a Transformer with sliding-window attention. However, our model continues learning at test time via next-token prediction on the given context, compressing the context it reads into its weights. In addition, we improve the model's initialization for learning at test time via meta-learning at training time. Overall, our method, a form of Test-Time Training (TTT), is End-to-End (E2E) both at test time (via next-token prediction) and training time (via meta-learning), in contrast to previous forms. We conduct extensive experiments with a focus on scaling properties. In particular, for 3B models trained with 164B tokens, our method (TTT-E2E) scales with context length in the same way as Transformer with full attention, while others, such as Mamba 2 and Gated DeltaNet, do not. However, similar to RNNs, TTT-E2E has constant inference latency regardless of context length, making it 2.7× faster than full attention for 128K context. Our code is publicly available.

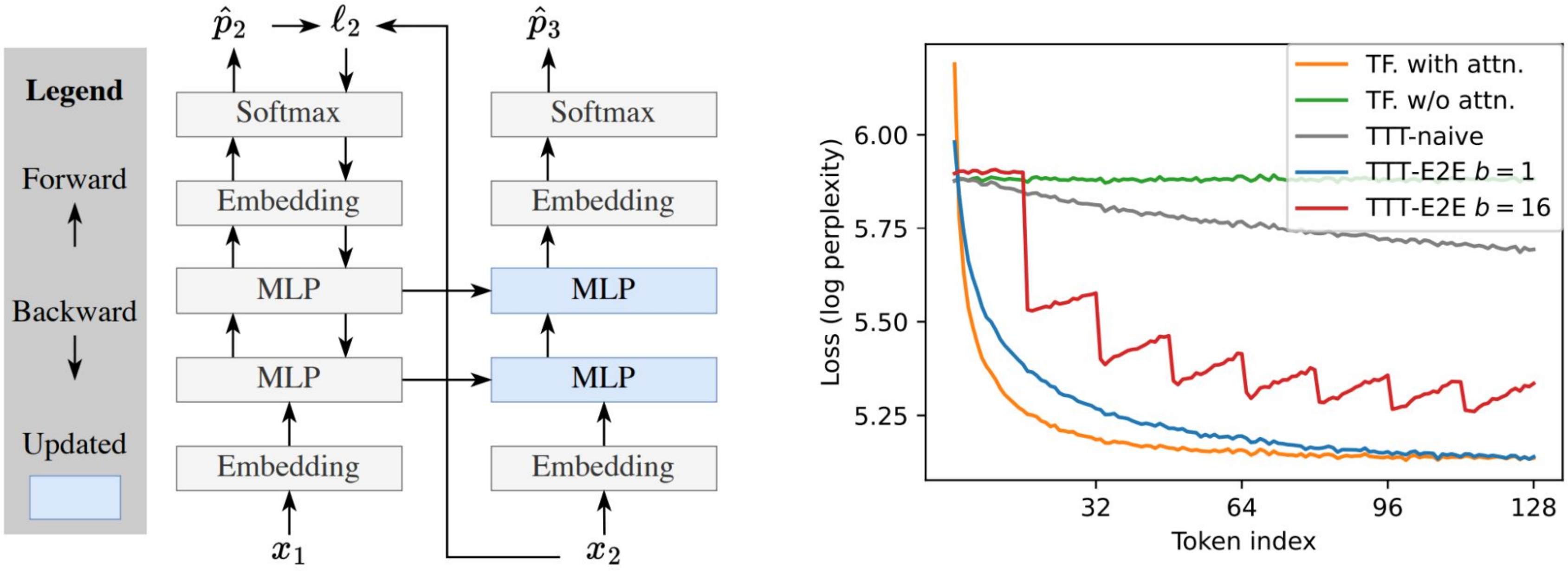
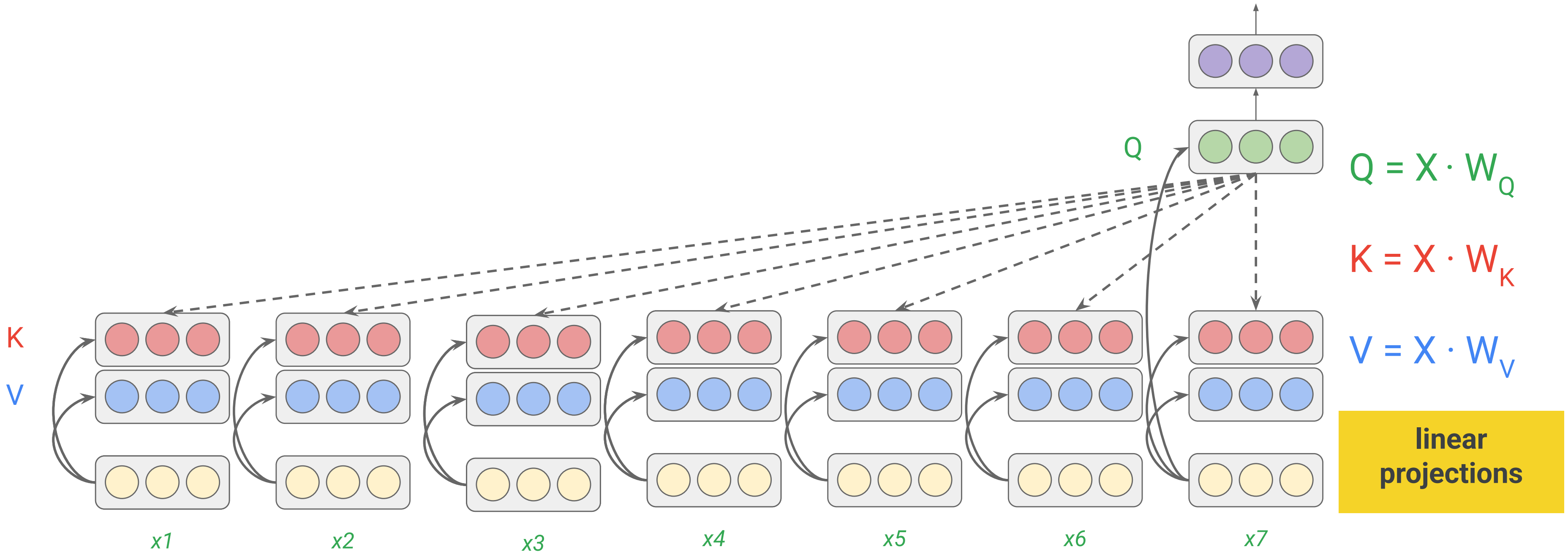
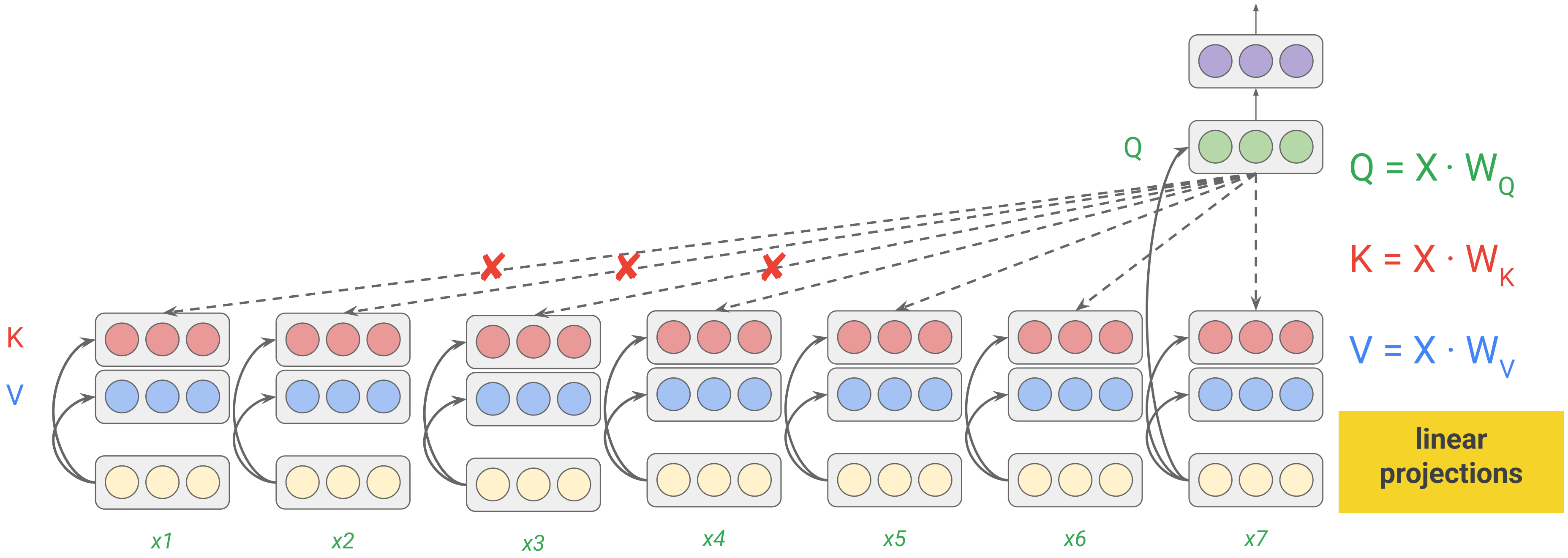


Figure 2. Toy example. **Left:** Given x_1 and x_2 as context, we want to predict the unknown x_3. Our toy baseline, a Transformer without self-attention (using only the upward arrows), is effectively a bigram since it has no memory of x_1. TTT (using all the arrows) first tries to predict x_2 from x_1 as an exercise: It computes the loss ℓ_2 between x_2 and the prediction $\hat{p}_2$, then takes a gradient step on ℓ_2. Now information of x_1 is stored in the updated MLPs (blue). **Right:** Token-level test loss ℓ_t for various methods in our toy example, as discussed in Subsection 2.2, except for TTT-E2E $b = 16$ discussed in Subsection 2.3. In particular, TTT-E2E $b = 1$ turns the green line (our toy baseline) into the blue line, which performs almost as well as orange (using full attention).

Test-time training



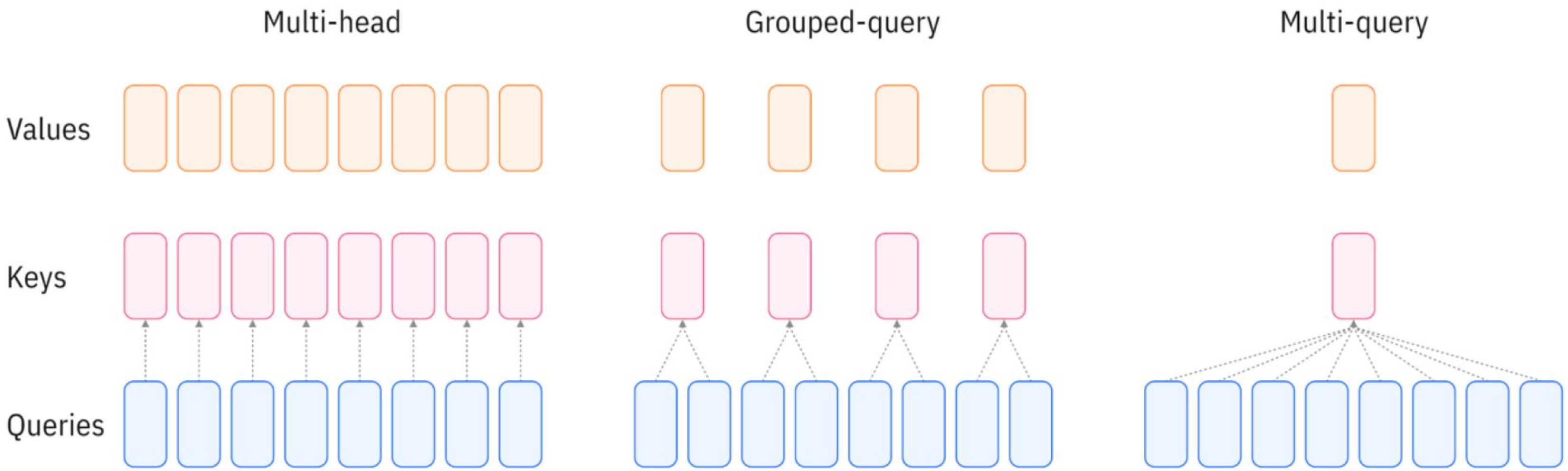
Test-time training (cont'd)



Others

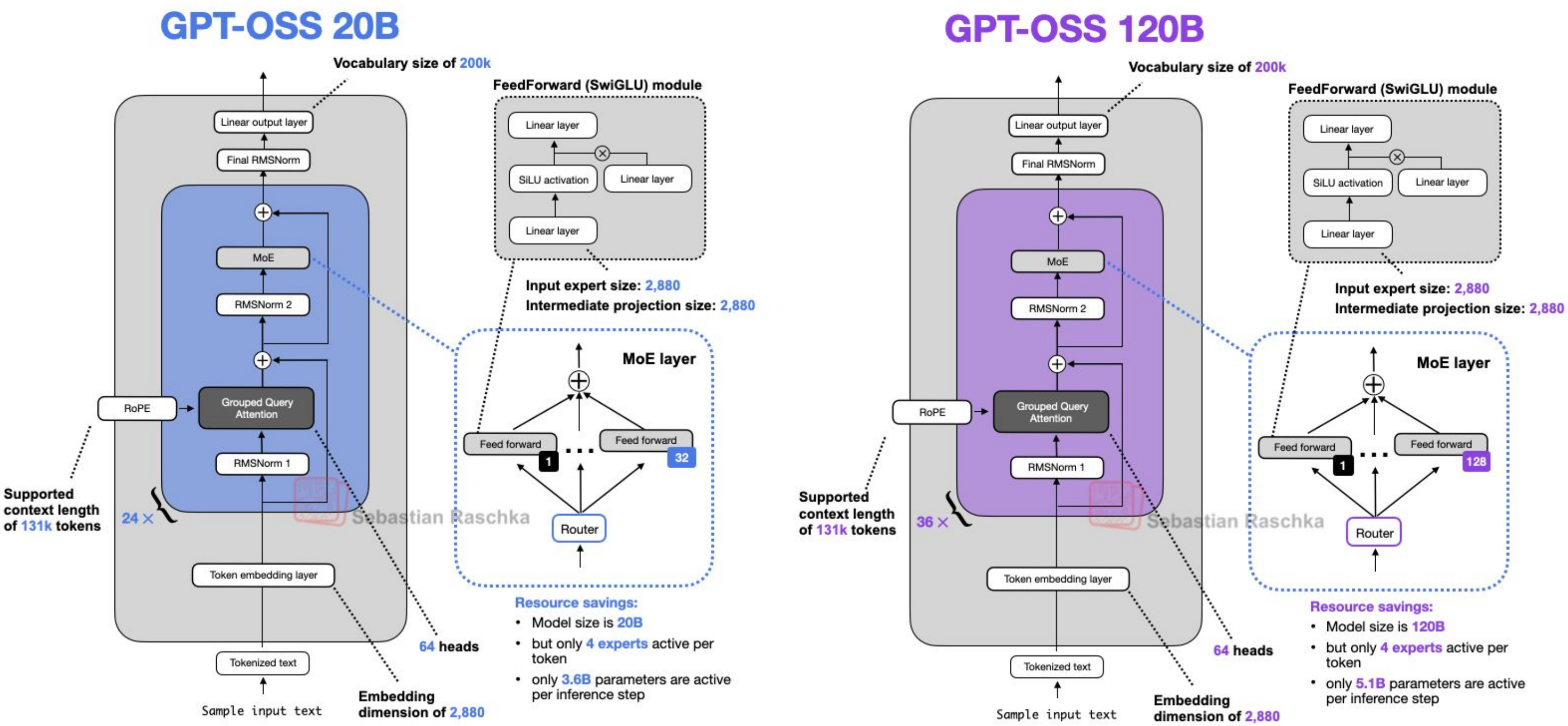
- Multi-query attention / Grouped-query attention
- KV caching
- Model merging / Model editing
- Steering vectors
- Knowledge distillation
- Pruning
- Quantization

Multi-query attention / grouped-query attention



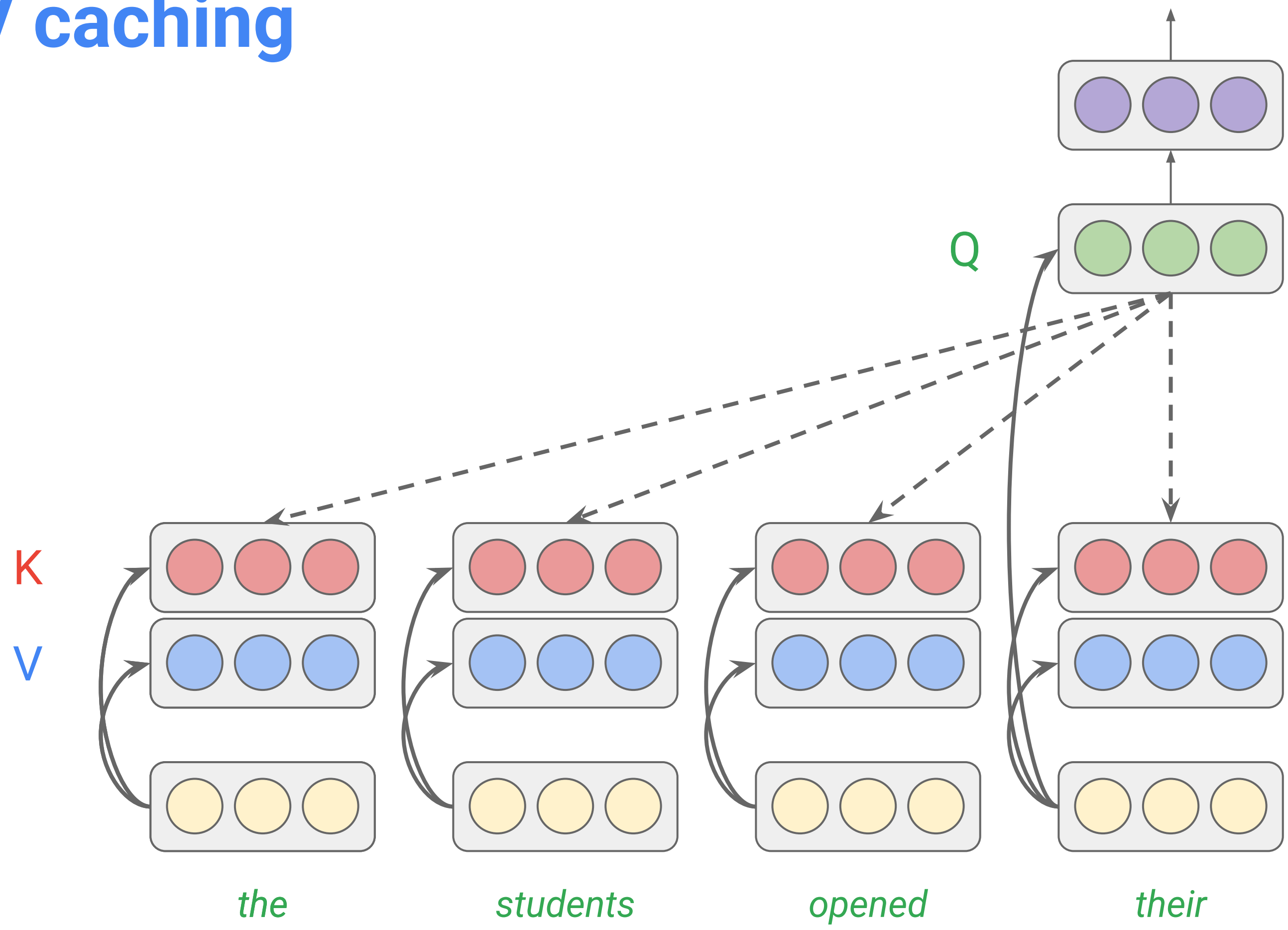
https://www.ibm.com/think/topics/grouped-query-attention

Multi-query attention / grouped-query attention



https://magazine.sebastianraschka.com/p/from-gpt-2-to-gpt-oss-analyzing-the

KV caching



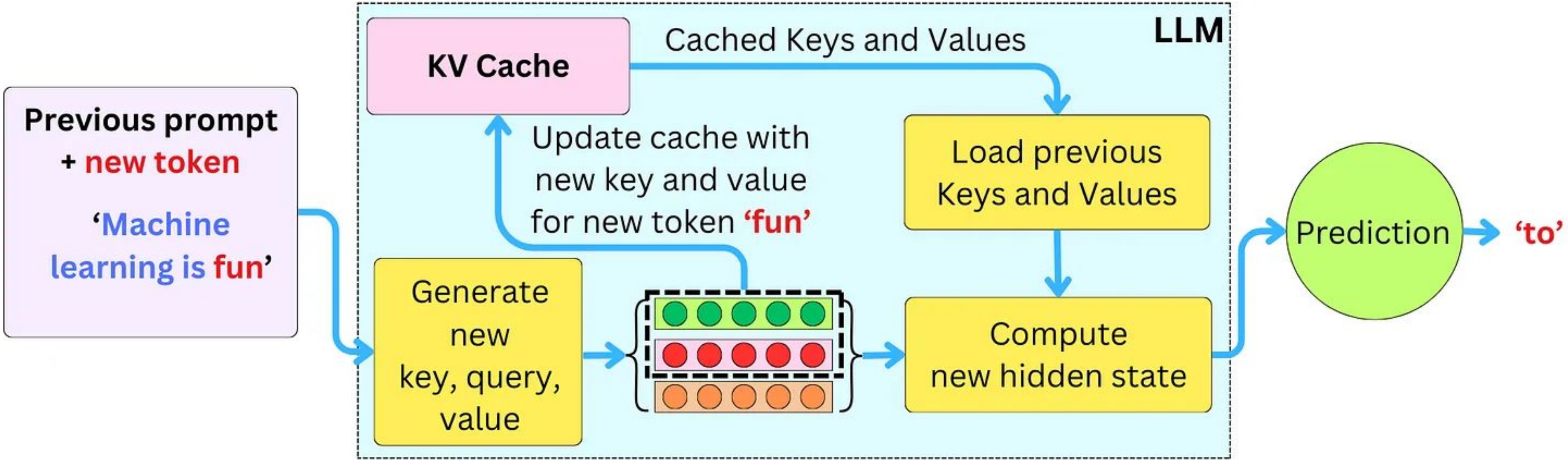
$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

KV caching (cont'd)



https://newsletter.theaiedge.io/p/how-to-reduce-llm-decoding-time-with

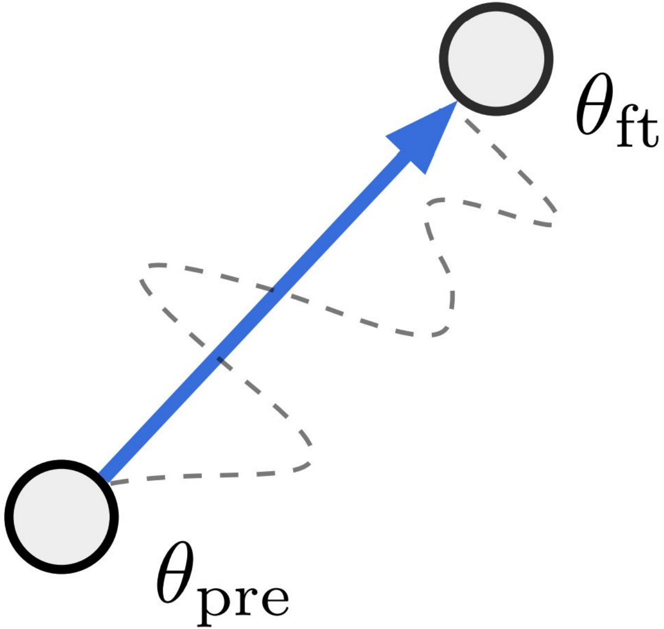
EDITING MODELS WITH TASK ARITHMETIC

Gabriel Ilharco[*1] **Marco Tulio Ribeiro**[2] **Mitchell Wortsman**[1] **Suchin Gururangan**[1]
Ludwig Schmidt[1,3] **Hannaneh Hajishirzi**[1,3] **Ali Farhadi**[1]
[1]University of Washington [2]Microsoft Research [3]Allen Institute for AI

Why do we want to edit LLMs?

- improve performance on downstream tasks
- mitigate biases or unwanted behavior
- align models with human preferences
- update models with new information

The notion of task vectors



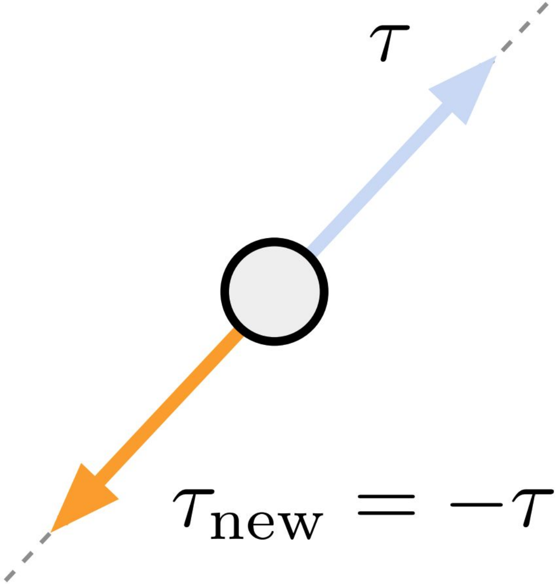
$$\tau = \theta_{\text{ft}} - \theta_{\text{pre}}$$

$$\theta_{\text{new}} = \theta + \tau$$

In practice, we have an optional scaling term λ

$$\theta_{\text{new}} = \theta + \lambda\tau$$

Forgetting via negation



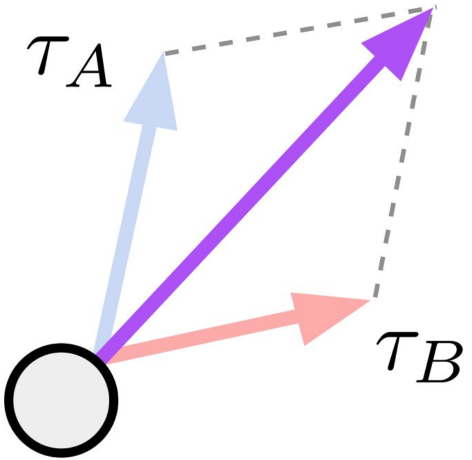
Example: making a language model produce less toxic content

$$\theta_{\text{new}} = \theta - \tau = \theta - (\theta_{ft} - \theta)$$

In practice, we have an optional scaling term λ

Learning via addition

$$\tau_{\text{new}} = \tau_A + \tau_B$$



Example: building a multi-task model

$$\theta_{\text{new}} = \theta + \tau = \theta + (\tau_A + \tau_B)$$

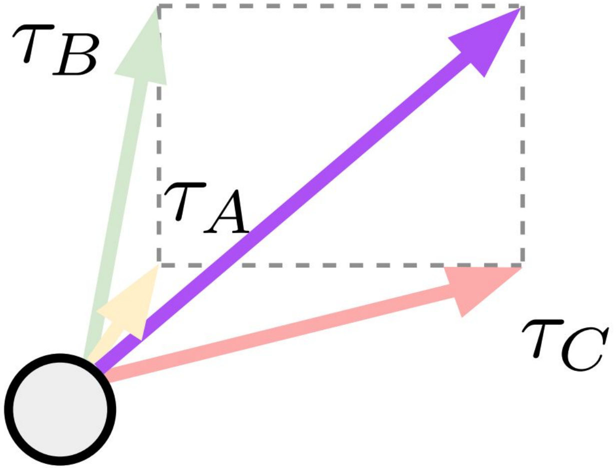
$$= \theta + (\theta_A - \theta) + (\theta_B - \theta)$$

In practice, we have optional scaling terms λ_A, λ_B

Task analogies

"A is to B as C is to D"

$$\tau_{\text{new}} = \tau_C + (\tau_B - \tau_A)$$



Example: improving domain generalization

$$\tau_B - \tau_A = \tau_D - \tau_C$$

$$\tau_D = \tau_C + (\tau_B - \tau_A)$$

$$\theta_{\text{new}} = \theta + \tau_C + (\tau_B - \tau_A)$$

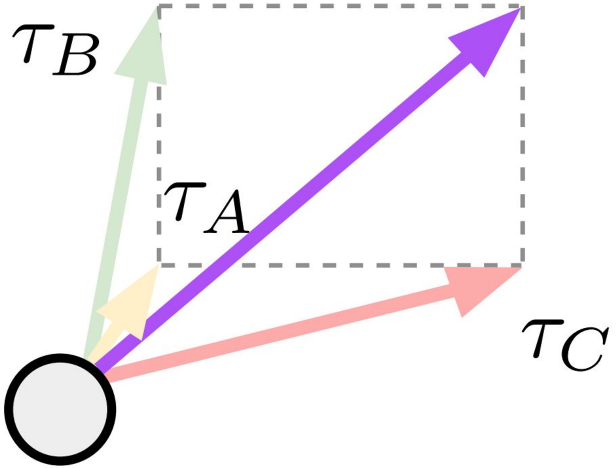
$$= \theta + (\theta_C - \theta) + (\theta_B - \theta) - (\theta_A - \theta)$$

In practice, we have optional scaling terms λ_A, λ_B, λ_C

Task analogies

"A is to B as C is to D"

$$\tau_{\text{new}} = \tau_C + (\tau_B - \tau_A)$$



Example: improving domain generalization

$$\tau_B - \tau_A = \tau_D - \tau_C$$

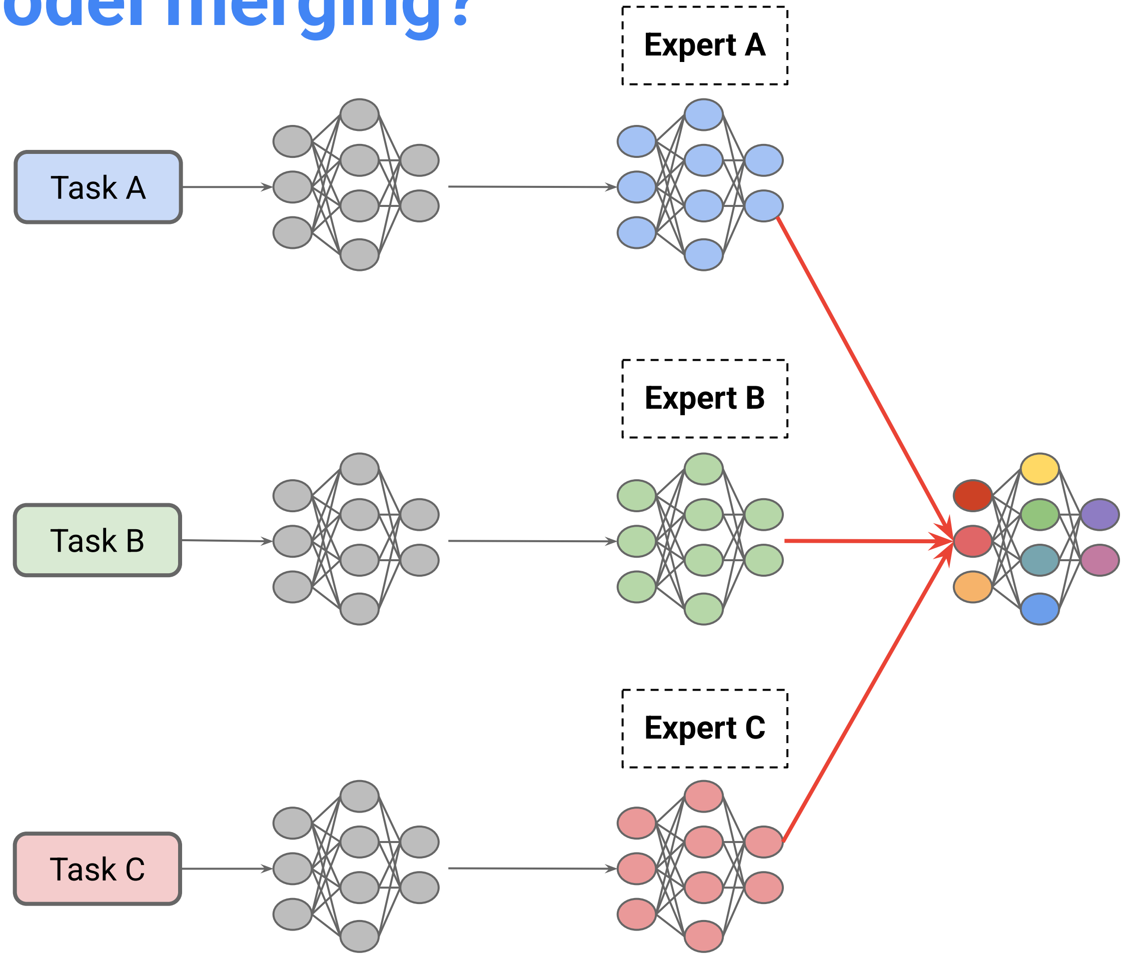
$$\tau_D = \tau_C + (\tau_B - \tau_A)$$

$$\theta_{\text{new}} = \theta + \tau_C + (\tau_B - \tau_A)$$

$$= \theta + (\theta_C - \theta) + (\theta_B - \theta) - (\theta_A - \theta)$$

In practice, we have optional scaling terms $\lambda_A, \lambda_B, \lambda_C$

What is model merging?



Why model merging?

- dramatically reduces storage and serving costs by reusing a single model across tasks

- enables compositional combination of capabilities from expert models, which can improve generalization to novel tasks

- supports decentralized and modular model development by allowing multiple contributors to independently build models and later combine them together

Efficient Model Development through Fine-tuning Transfer

Pin-Jie Lin[1] *pinjie@vt.edu*

Rishab Balasubramanian[1] *rishbb@vt.edu*

Fengyuan Liu[2] *fy.liu@mail.utoronto.ca*

Nikhil Kandpal[2] *nkandpa2@cs.toronto.edu*

Tu Vu[1] *tuvu@vt.edu*

[1] *Virginia Tech* [2] *University of Toronto & Vector Institute*

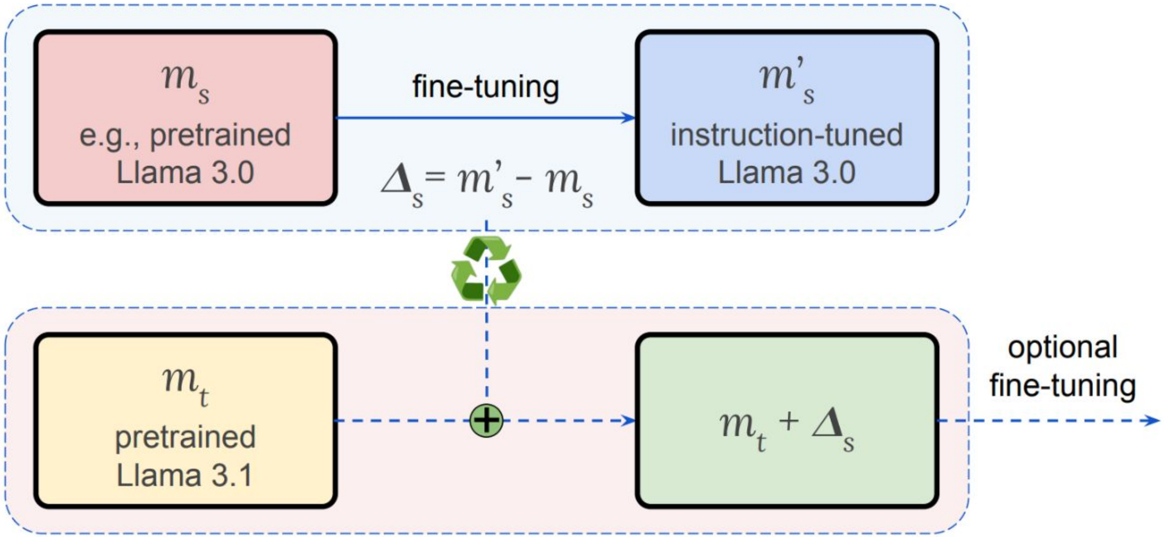


Figure 1: To transfer fine-tuning (e.g., instruction tuning) from a *source* model version s (e.g., Llama 3.0) to a *target* version t (Llama 3.1), we first compute the diff vector $\Delta_s = m'_s - m_s$ from version s, where m'_s is the fine-tuned model (instruction-tuned Llama 3.0) and m_s is the base model (pretrained Llama 3.0). Then, we add Δ_s to the target base model (pretrained Llama 3.1) to approximate the fine-tuned model in version t (instruction-tuned Llama 3.1). We explore two scenarios: (1) *recycling*—transferring from an older model version to a newer one to reduce retraining, and (2) *backporting*—transferring from a newer version to an older one to take advantage of the newer fine-tuning while maintaining optimization for specific use cases.

Transferring fine-tuning updates

Model	GSM8K	MATH	ARC_C	GPQA	MMLU	IFEval
Llama 3.0 8B Instruct	81.1	28.8	82.4	**31.5**	64.9	**76.6**
Llama 3.0 8B	55.6	17.3	79.7	22.3	66.7	34.5
+ $\Delta_{3.1}$	**82.8**	**44.7**	**83.0**	25.9	**70.0**	**76.6**
Llama 3.1 8B Instruct	**86.5**	**50.3**	**83.8**	31.3	**72.9**	80.5
Llama 3.1 8B	56.6	19.3	79.2	21.9	66.8	36.4
+ $\Delta_{3.0}$	79.8	29.9	82.9	**32.6**	65.1	**83.3**

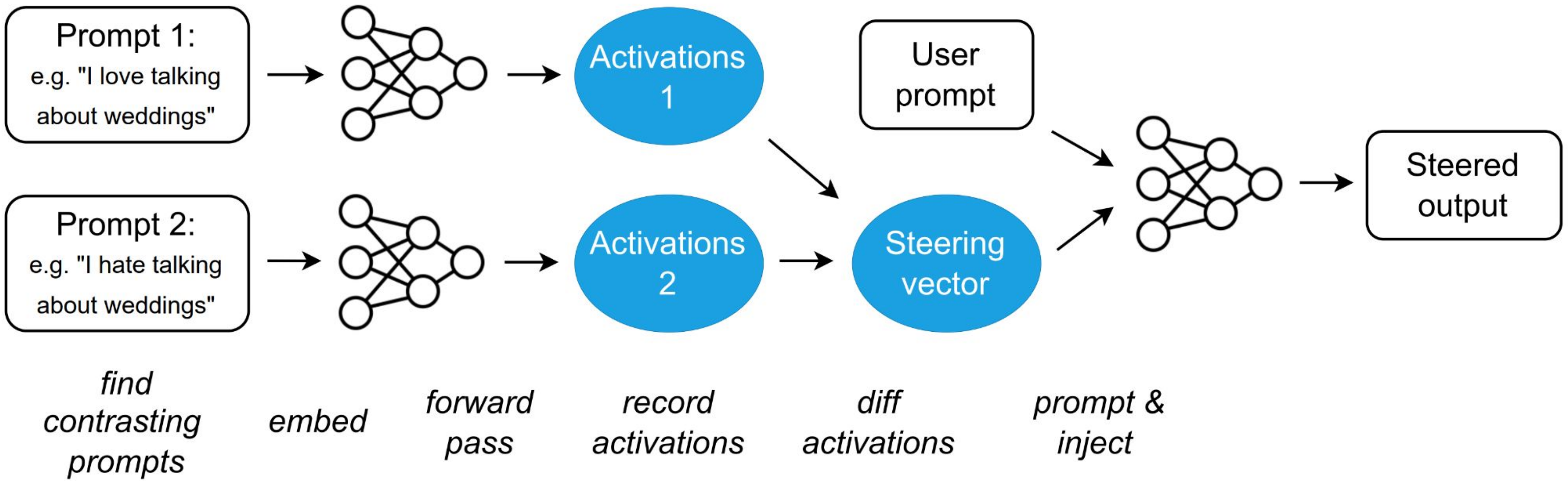
Table 1: Fine-tuning transfer significantly improves the performance of the target base model across various tasks, achieving results comparable to its fine-tuned counterpart in many cases. Here, $\Delta_{3.0}$ and $\Delta_{3.1}$ represent the diff vectors between Llama Instruct and Llama for versions 3.0 and 3.1, respectively. Notably, adding the diff vector Δ_s from a different model version can effectively transform a non-instruction-tuned model (e.g., Llama 3.0 or Llama 3.1) into one that follows instructions well (Llama 3.0 + $\Delta_{3.1}$ or Llama 3.1 + $\Delta_{3.0}$) without further training. Additional results for OLMo and Tülu can be found in Appendix A, where we additionally find that advanced LLM capabilities, attained through alignment tuning stages such as Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), or Group Relative Policy Optimization (GRPO), can be successfully transferred across different model versions.

Multilingual model development

Model	Malagasy	Sinhala	Turkish
Llama 3.0 8B Instruct	23.1	23.3	30.8
+ FT	30.8	29.0	43.2
Llama 3.1 8B Instruct	27.6	**33.0**	27.7
+ $\Delta_{3.0}$	**32.3**	32.3	**43.2**

Table 2: Recycling fine-tuning updates improves multilingual performance on Global MMLU without re-training, yielding a 4.7% and 15.5% absolute improvement for Malagasy and Turkish, respectively, compared to Llama 3.1 8B Instruct. $\Delta_{3.0}$ represents the diff vector between Llama 3.0 Instruct and its monolingual fine-tuned (FT) version.

Steering vectors



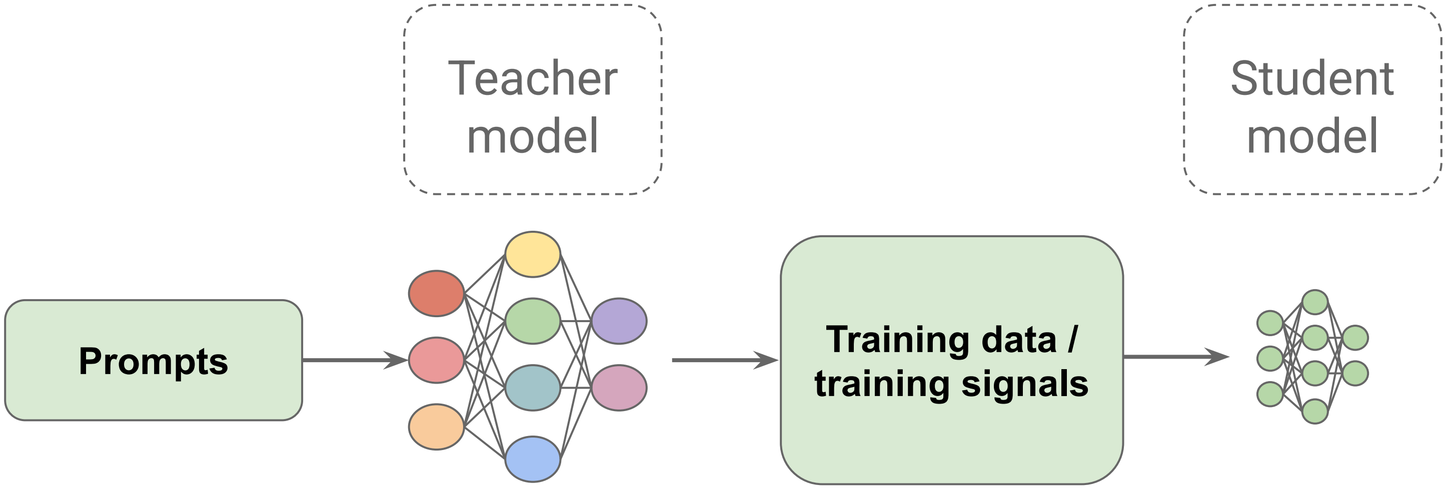
https://arxiv.org/abs/2308.10248

Steering vectors (cont'd)

Prompt	+ steering	= completion
I hate you because...	[None]	...you are the most disgusting thing I have ever seen.
	ActAdd (love)	...you are so beautiful and I want to be with you forever.
I went up to my friend and said...	[None]	..."I'm sorry, I can't help you." "No," he said. "You're not."
	ActAdd (weddings)	..."I'm going to talk about the wedding in this episode of Wedding Season. I think it's a really good episode. It's about how you're supposed to talk about weddings."

https://arxiv.org/abs/2308.10248

Knowledge distillation



Distilling the Knowledge in a Neural Network

Geoffrey Hinton[*†]
Google Inc.
Mountain View
geoffhinton@google.com

Oriol Vinyals[†]
Google Inc.
Mountain View
vinyals@google.com

Jeff Dean
Google Inc.
Mountain View
jeff@google.com

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF
Hugging Face
{victor,lysandre,julien,thomas}@huggingface.co

$$\text{Loss} = \lambda_{\text{ce}} \cdot \mathcal{L}_{\text{ce}} + \lambda_{\text{kd}} \cdot \mathcal{L}_{\text{kd}}$$

$$\text{Loss} = \lambda_{\text{ce}} \cdot \left(-\sum_{i=1}^{N} y_i \log(p_i) \right) + \lambda_{\text{kd}} \cdot D_{\text{KL}}(q_{\text{teacher}}(x) \| q_{\text{student}}(x))$$

Where:

- y_i is the true label for token i,
- p_i is the predicted probability for the correct token for token i,
- N is the number of tokens,
- $D_{\text{KL}}(q_{\text{teacher}}(x) \| q_{\text{student}}(x))$ is the Kullback-Leibler divergence between the teacher and student models' probability distributions,
- $q_{\text{teacher}}(x)$ and $q_{\text{student}}(x)$ are the output probability distributions from the teacher and student models, respectively,
- λ_{ce} and λ_{kd} are the weighting hyperparameters for the cross-entropy and knowledge distillation losses, respectively.

Assume two different distributions for predicting the next word:

- P (from Model 1):
  - *mat* → 0.7
  - *floor* → 0.2
  - *chair* → 0.1
- Q (from Model 2):
  - *mat* → 0.5
  - *floor* → 0.3
  - *chair* → 0.2

Kullback–Leibler (KL) Divergence Calculation

KL divergence measures how much P diverges from Q:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Substituting the values:

$$D_{KL}(P||Q) = 0.7 \log \frac{0.7}{0.5} + 0.2 \log \frac{0.2}{0.3} + 0.1 \log \frac{0.1}{0.2}$$

Training loss The student is trained with a distillation loss over the soft target probabilities of the teacher: $L_{ce} = \sum_i t_i * \log(s_i)$ where t_i (resp. s_i) is a probability estimated by the teacher (resp. the student). This objective results in a rich training signal by leveraging the full teacher distribution. Following Hinton et al. [2015] we used a *softmax-temperature*: $p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$ where T controls the smoothness of the output distribution and z_i is the model score for the class i. The same temperature T is applied to the student and the teacher at training time, while at inference, T is set to 1 to recover a standard *softmax*.

The final training objective is a linear combination of the distillation loss L_{ce} with the supervised training loss, in our case the *masked language modeling* loss L_{mlm} [Devlin et al., 2018]. We found it beneficial to add a *cosine embedding* loss (L_{cos}) which will tend to align the directions of the student and teacher hidden states vectors.

DistillBERT reduces BERT's size by 40%, while retaining 97% of its performance and being 60% faster

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

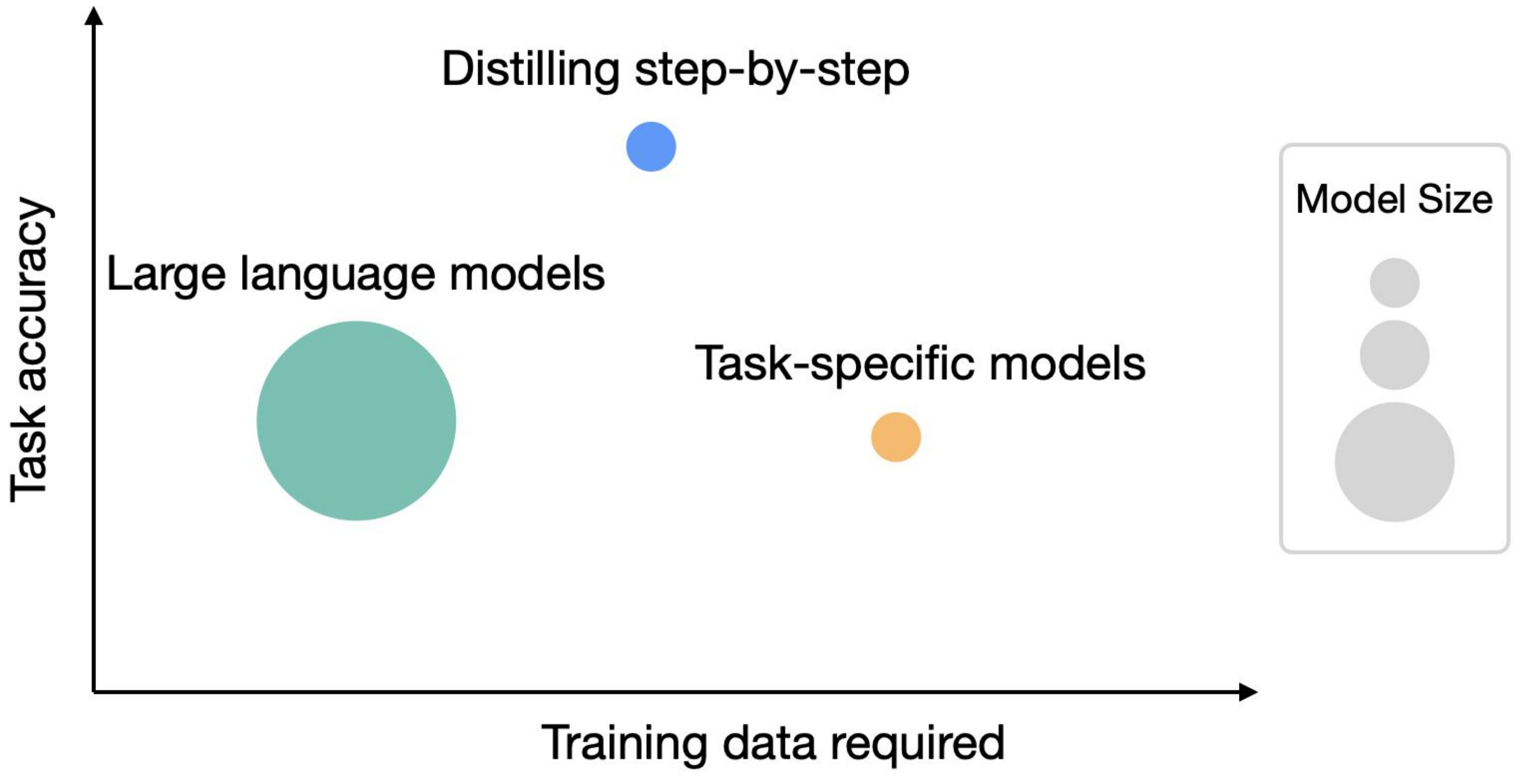
Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

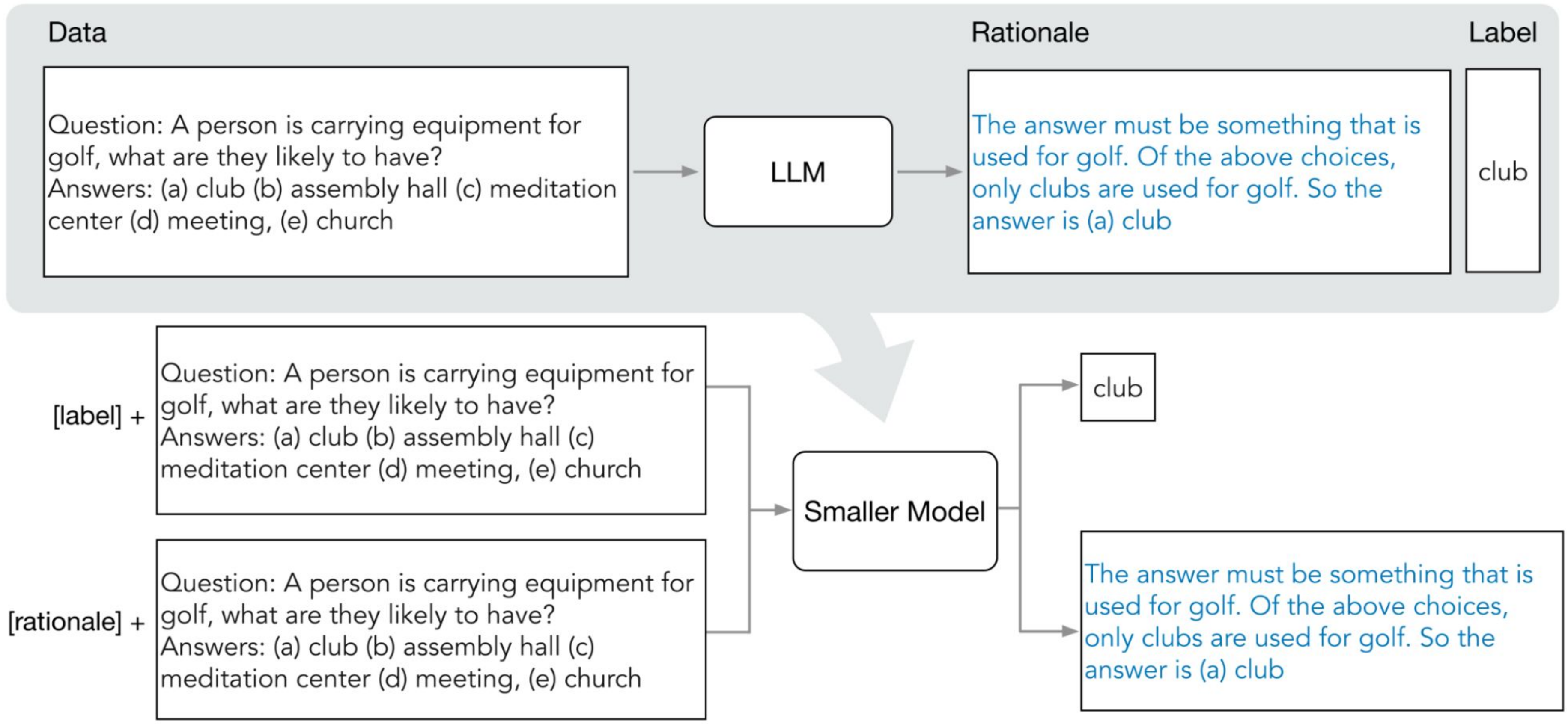
Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes

Cheng-Yu Hsieh[1,*] **Chun-Liang Li**[2], **Chih-Kuan Yeh**[3], **Hootan Nakhost**[2],
Yasuhisa Fujii[3], **Alexander Ratner**[1], **Ranjay Krishna**[1], **Chen-Yu Lee**[2], **Tomas Pfister**[2]
[1]University of Washington, [2]Google Cloud AI Research, [3]Google Research
cydhsieh@cs.washington.edu

Enabling a 770M parameter T5 model to outperform the few-shot prompted 540B PaLM model



Distilling step-by-step



DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
GPT-4o-0513	9.3	13.4	74.6	49.9	32.9	759
Claude-3.5-Sonnet-1022	16.0	26.7	78.3	65.0	38.9	717
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	**1820**
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9	1316
DeepSeek-R1-Distill-Qwen-1.5B	28.9	52.7	83.9	33.8	16.9	954
DeepSeek-R1-Distill-Qwen-7B	55.5	83.3	92.8	49.1	37.6	1189
DeepSeek-R1-Distill-Qwen-14B	69.7	80.0	93.9	59.1	53.1	1481
DeepSeek-R1-Distill-Qwen-32B	**72.6**	83.3	94.3	62.1	57.2	1691
DeepSeek-R1-Distill-Llama-8B	50.4	80.0	89.1	49.0	39.6	1205
DeepSeek-R1-Distill-Llama-70B	70.0	**86.7**	**94.5**	**65.2**	**57.5**	1633

Table 5 | Comparison of DeepSeek-R1 distilled models and other comparable models on reasoning-related benchmarks.

Pruning

- Remove parameters from the model after training

Are Sixteen Heads Really Better than One?

Paul Michel
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA
pmichel1@cs.cmu.edu

Omer Levy
Facebook Artificial Intelligence Research
Seattle, WA
omerlevy@fb.com

Graham Neubig
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA
gneubig@cs.cmu.edu

Published as a conference paper at ICLR 2019

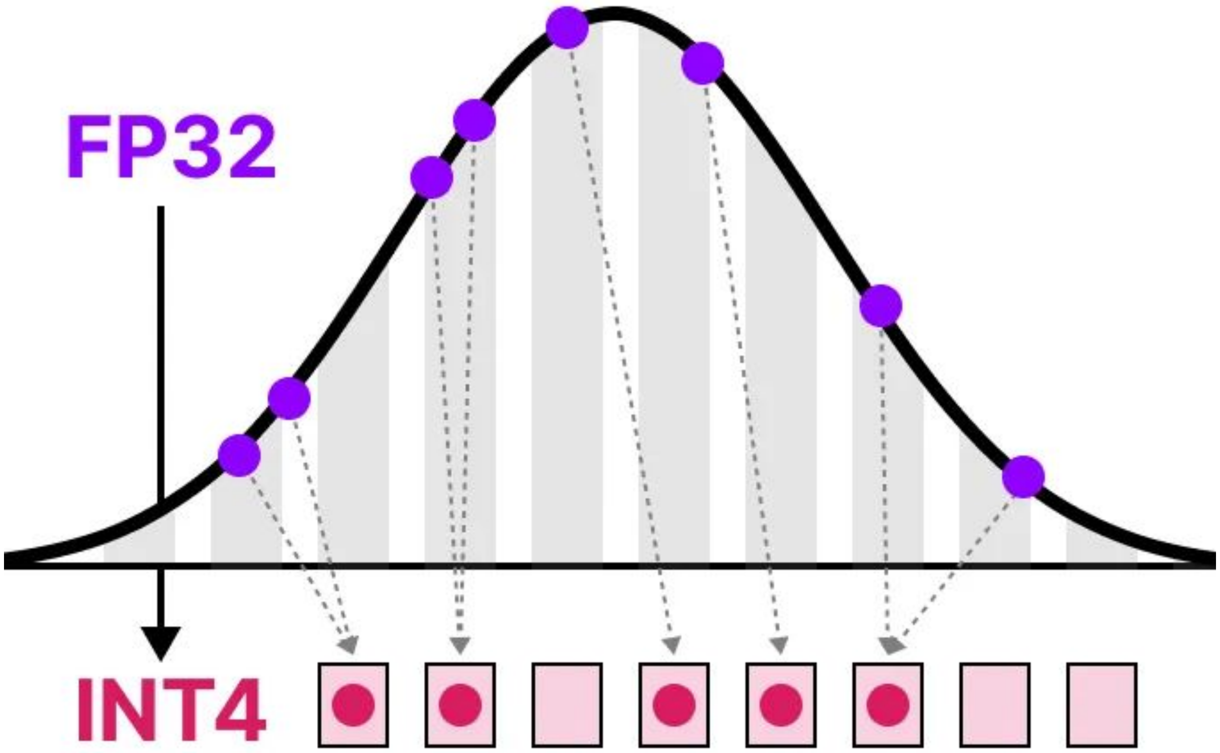
THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

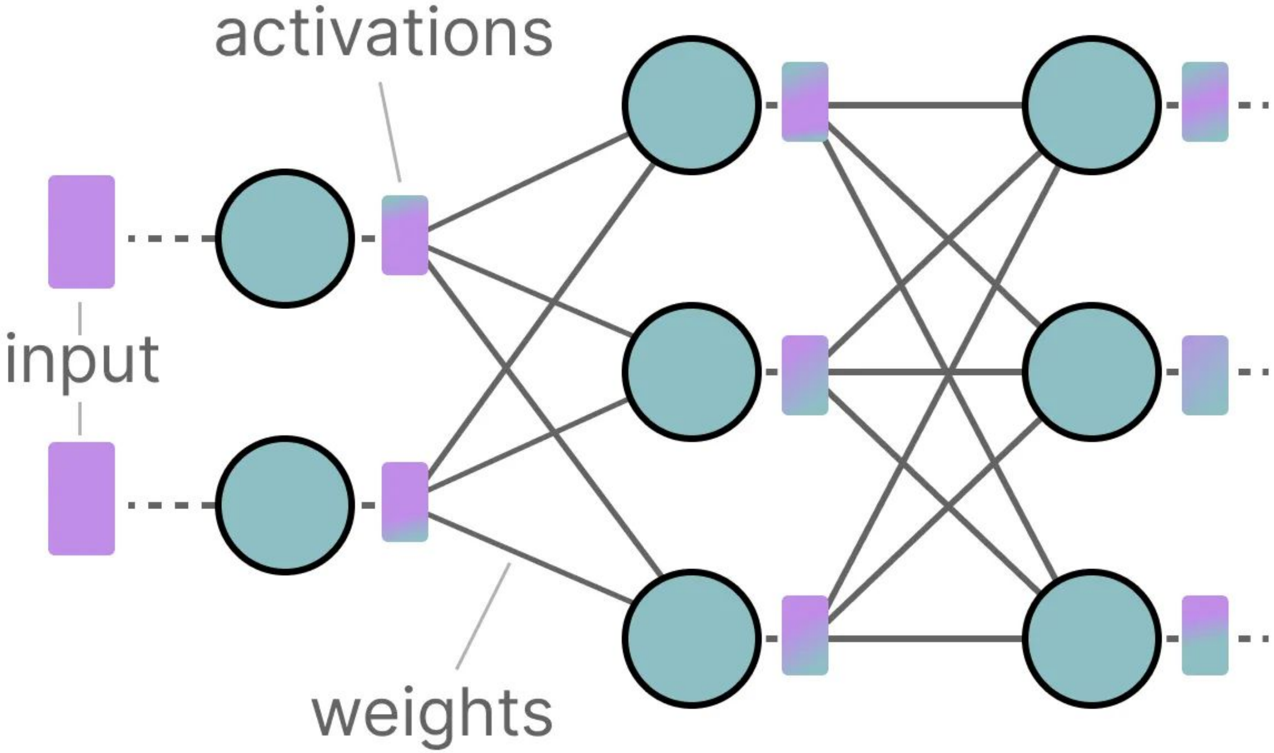
Training a pruned randomly-initialized networks can be better than training the full randomly-initialized network

Quantization



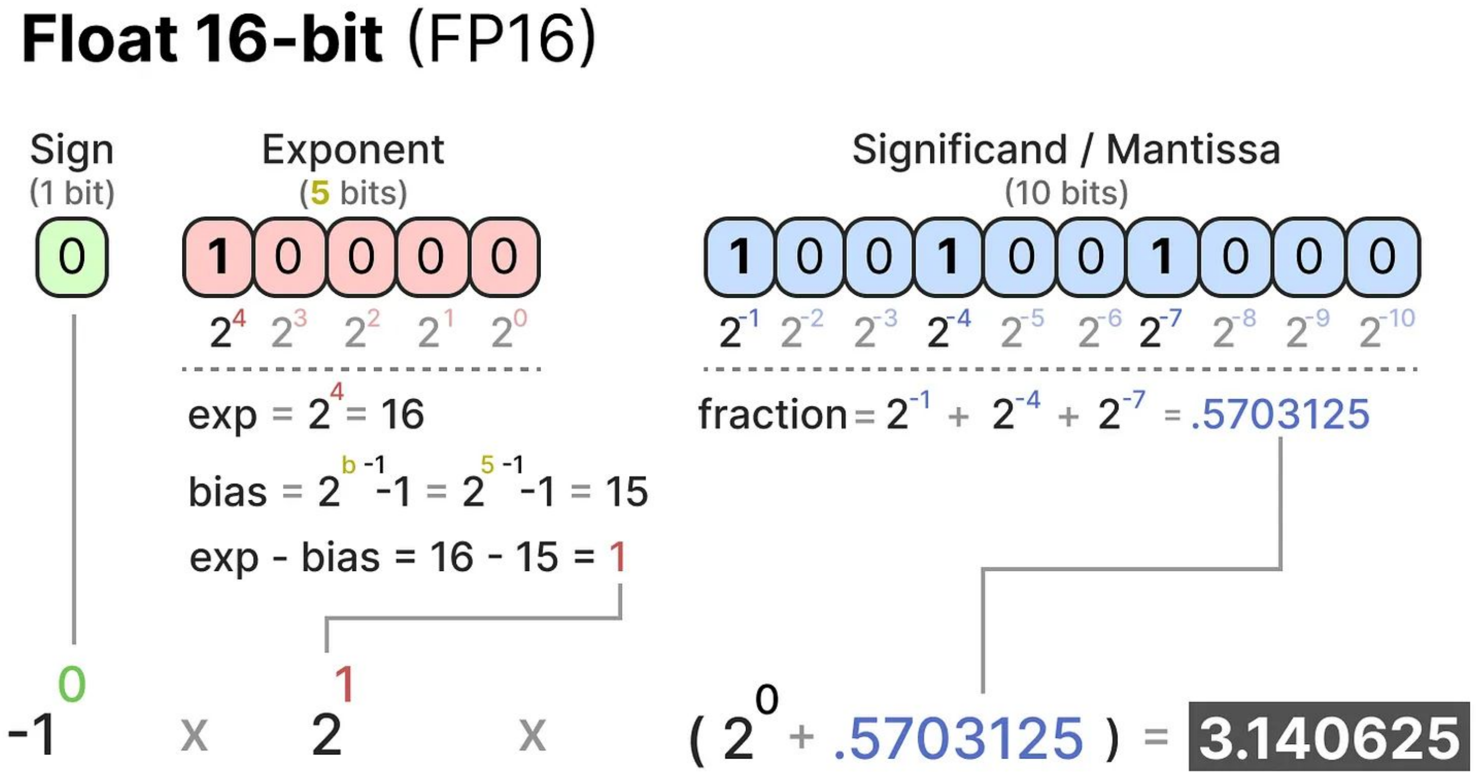
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Quantizing both the weights and activations



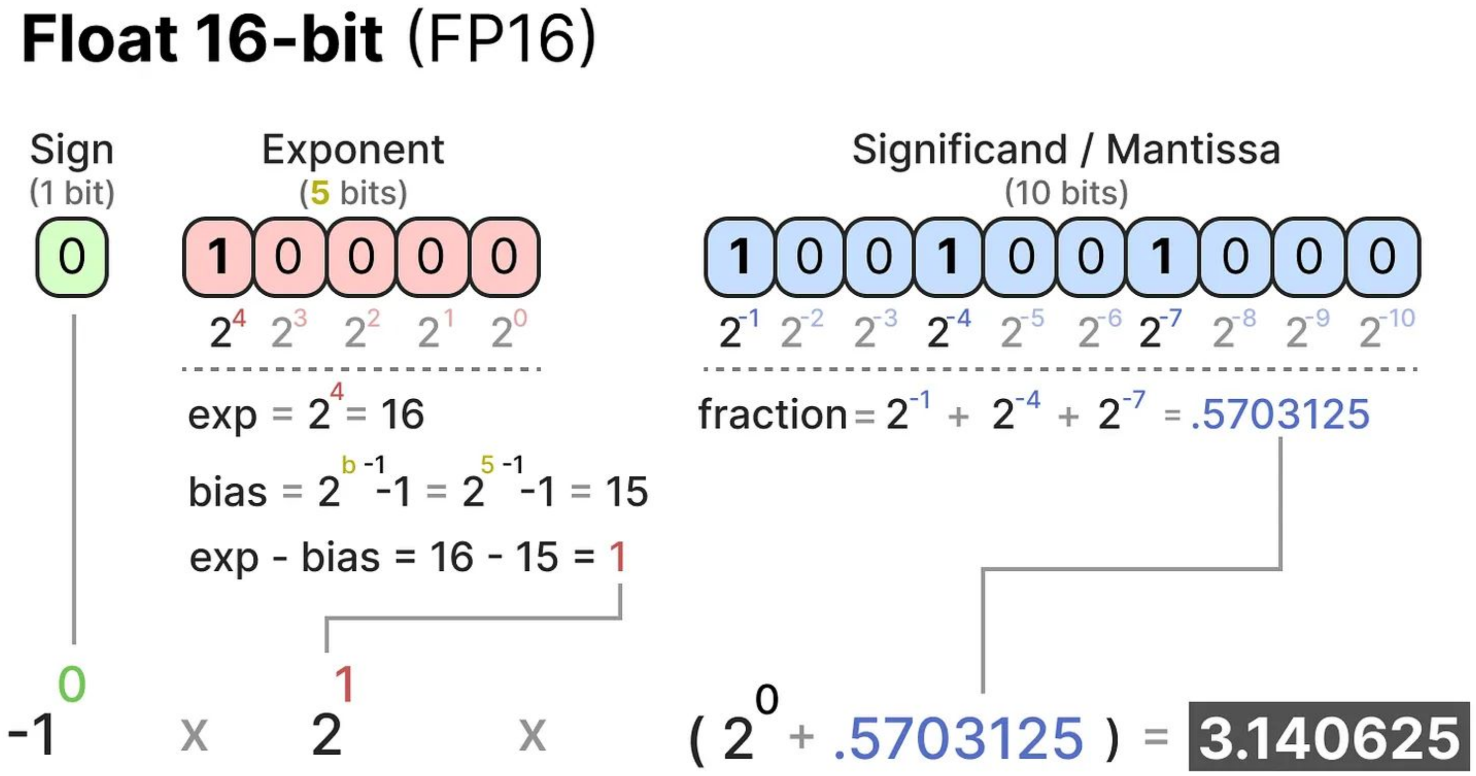
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

How to represent numerical values



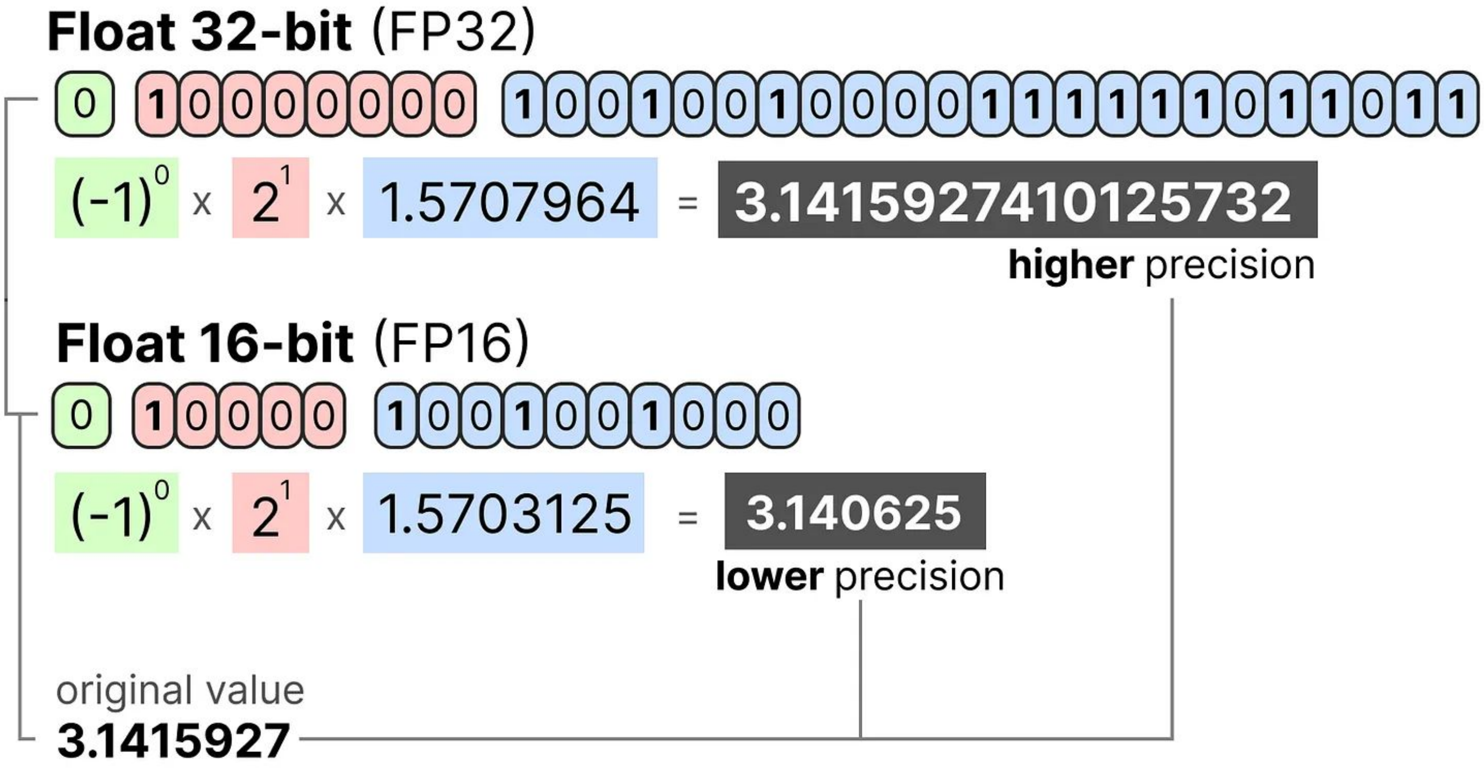
[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

How to represent numerical values (cont'd)



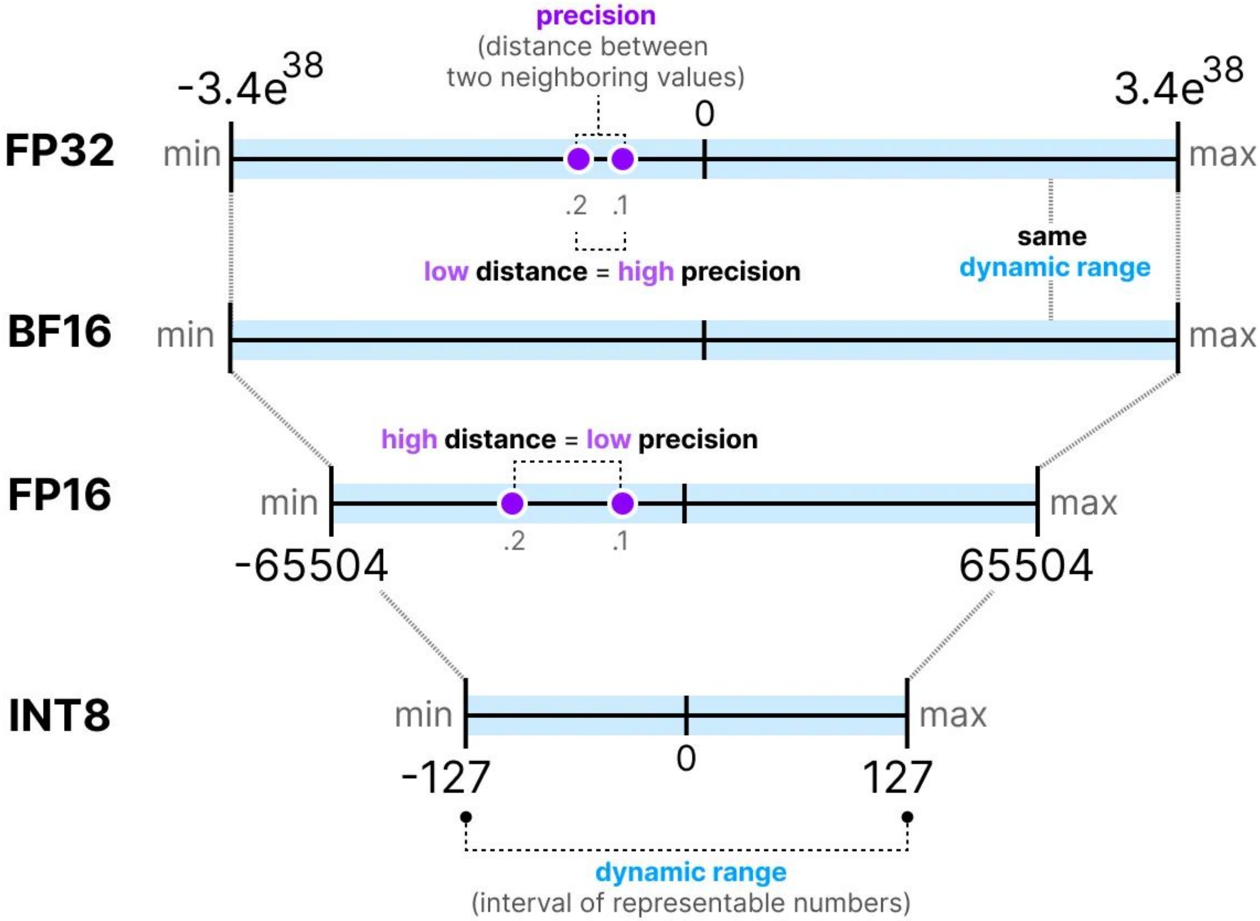
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

How to represent numerical values (cont'd)



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Memory constraints



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Memory constraints (cont'd)

$$\text{memory} = \frac{\text{nr\_bits}}{8} \times \text{nr\_params}$$

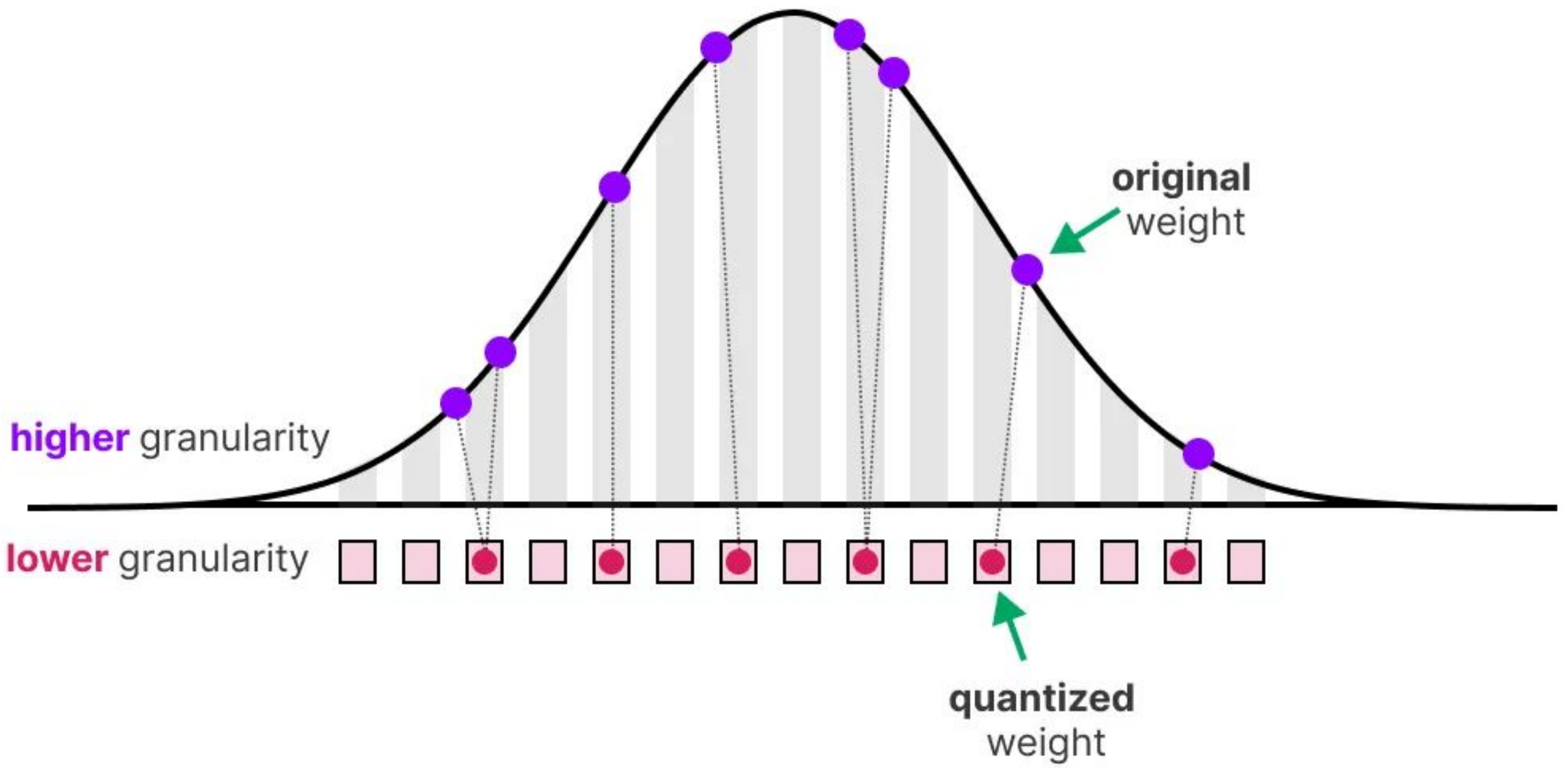
$$\textbf{64-bits} = \frac{64}{8} \times 70\text{B} \approx \textbf{560}\ \text{GB}$$

$$\textbf{32-bits} = \frac{32}{8} \times 70\text{B} \approx \textbf{280}\ \text{GB}$$

$$\textbf{16-bits} = \frac{16}{8} \times 70\text{B} \approx \textbf{140}\ \text{GB}$$

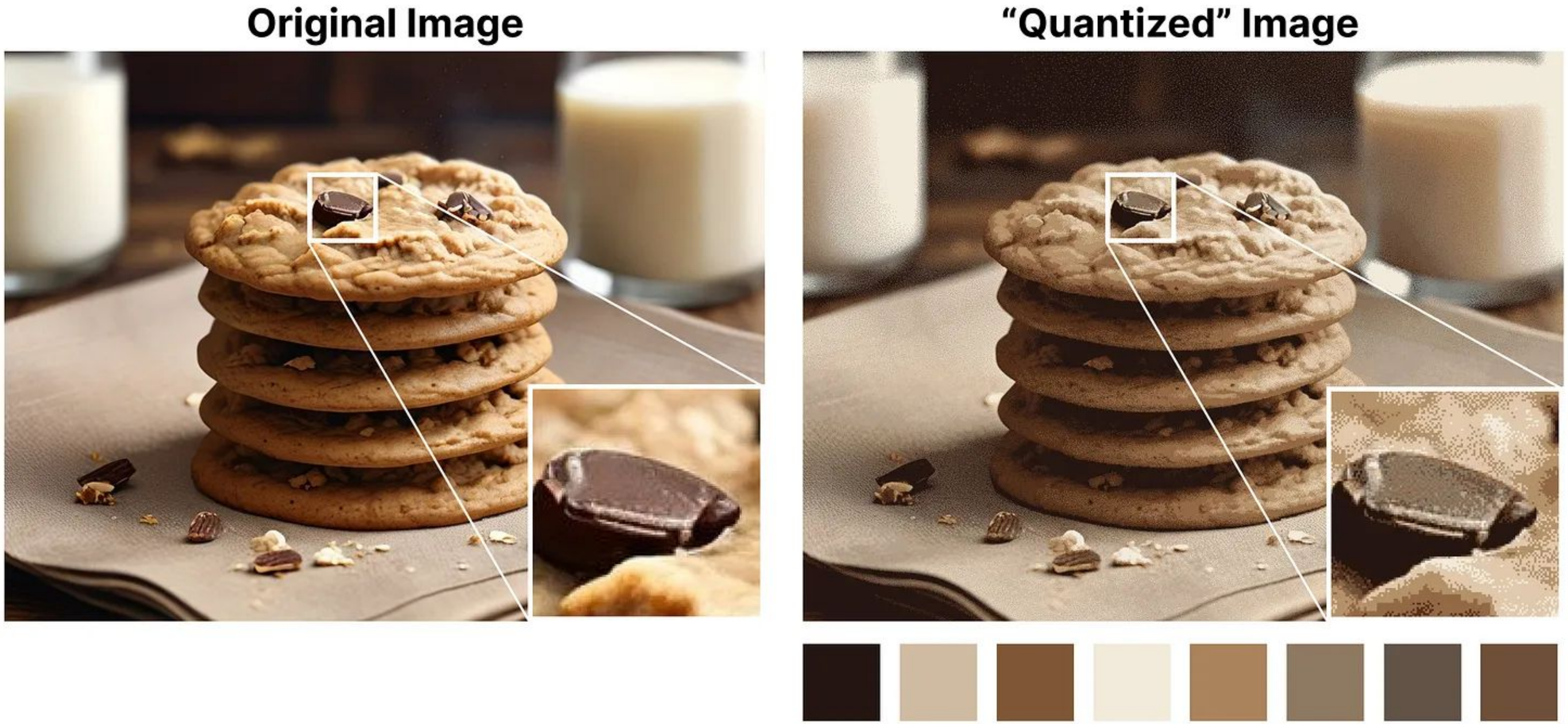
Maarten Grootendorst's blog: A Visual Guide to Quantization

Quantization



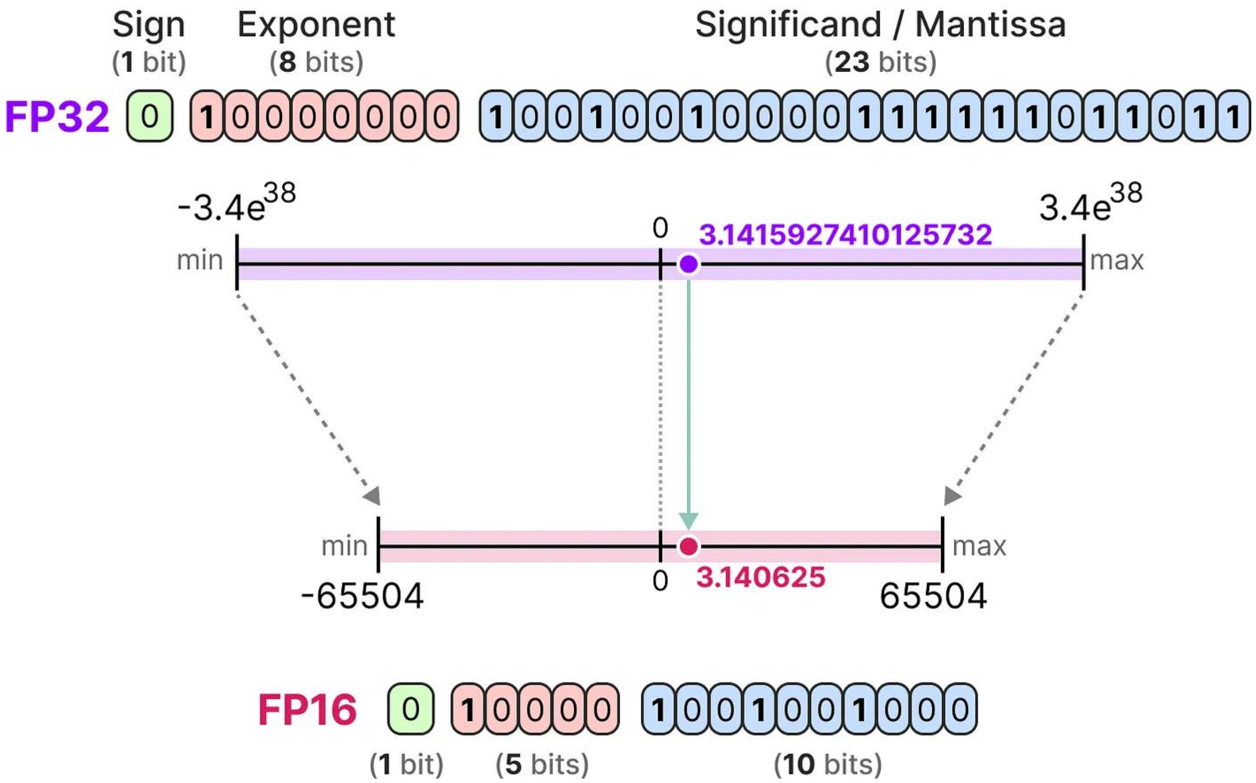
[Maarten Grootendorst's blog: A Visual Guide to Quantization](#)

Quantization



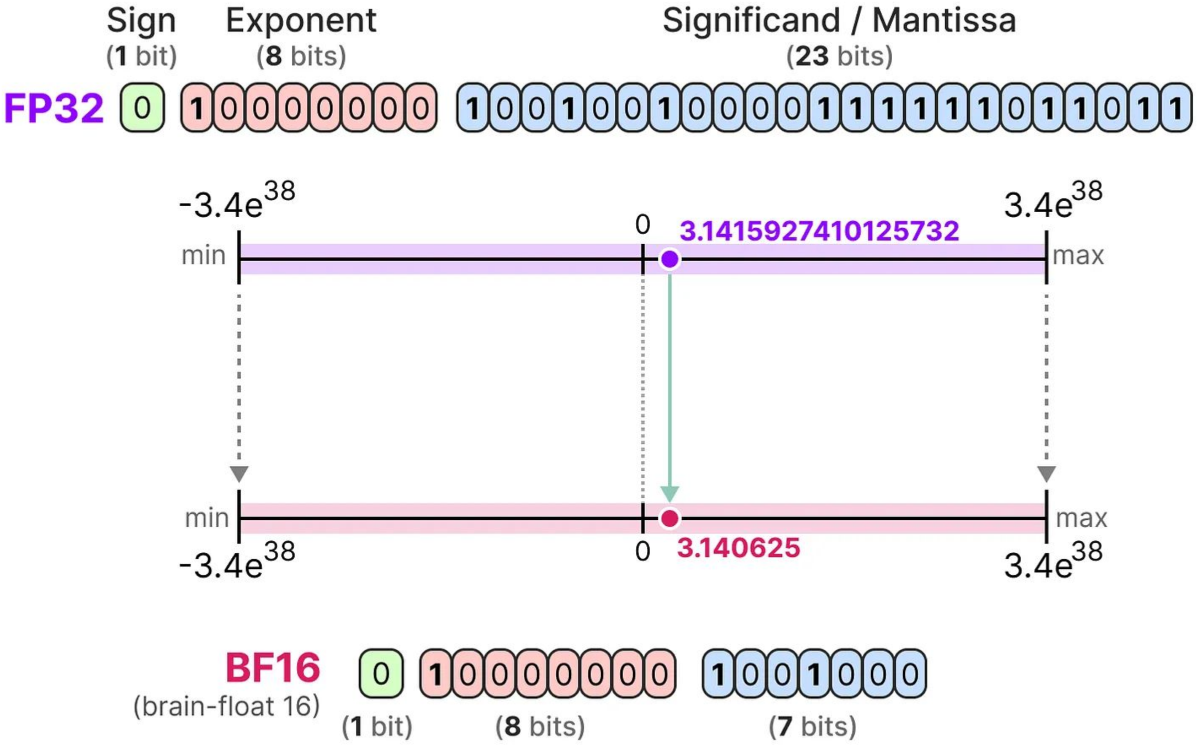
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Common data types: FP16 (half precision)



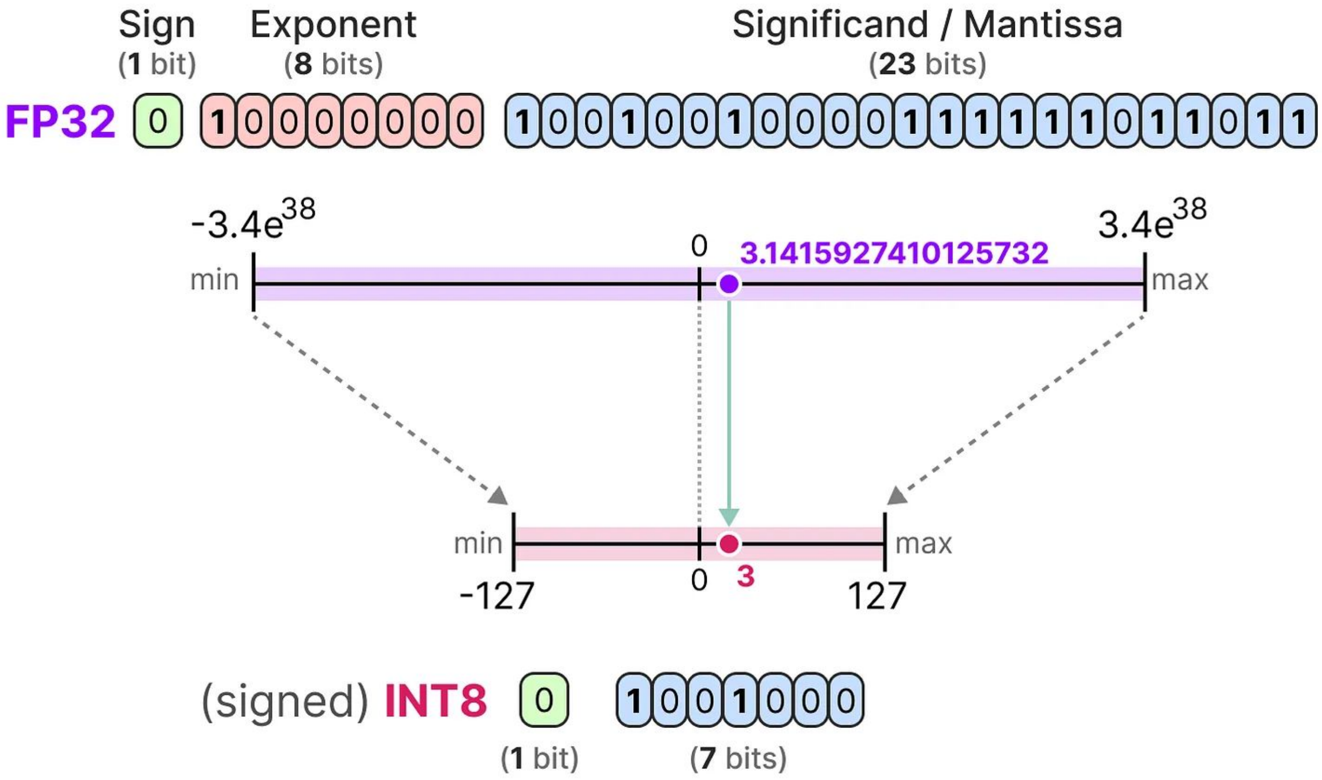
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Common data types: BF16



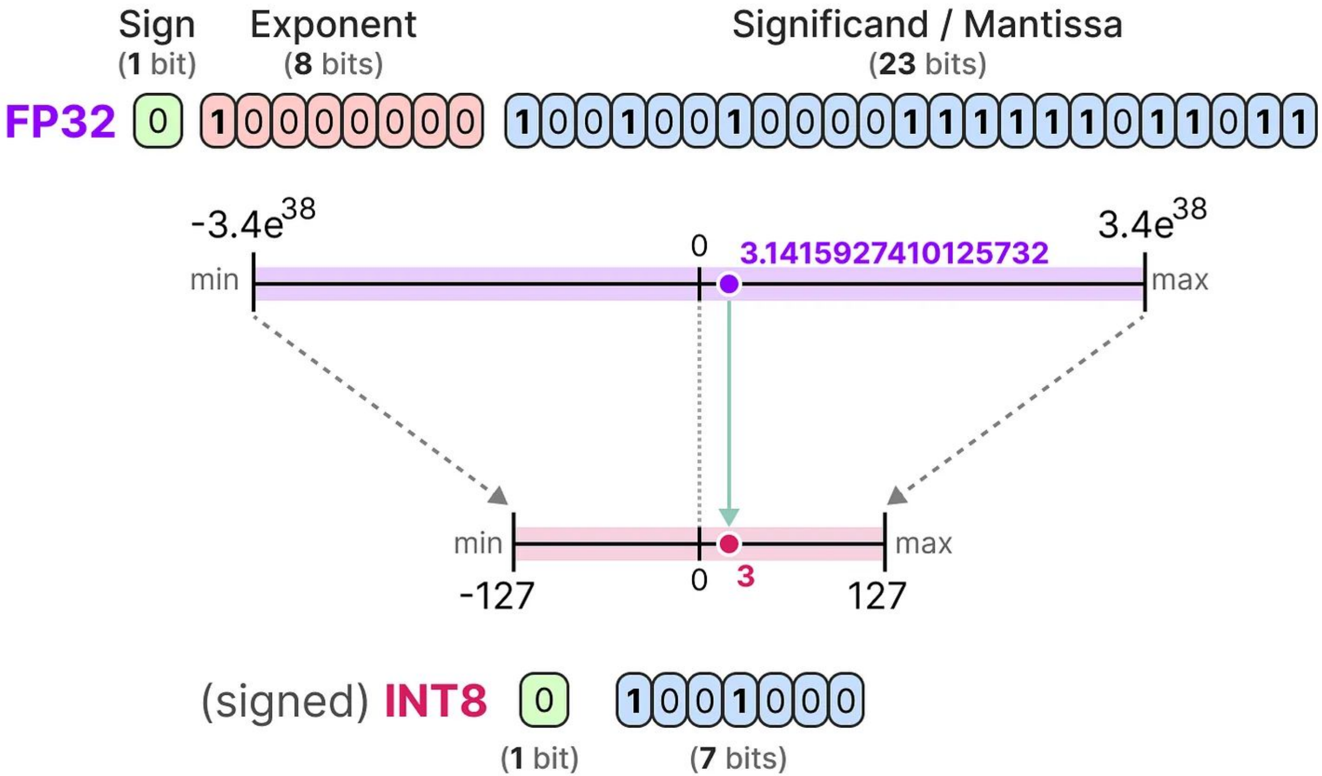
[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

Common data types: INT8



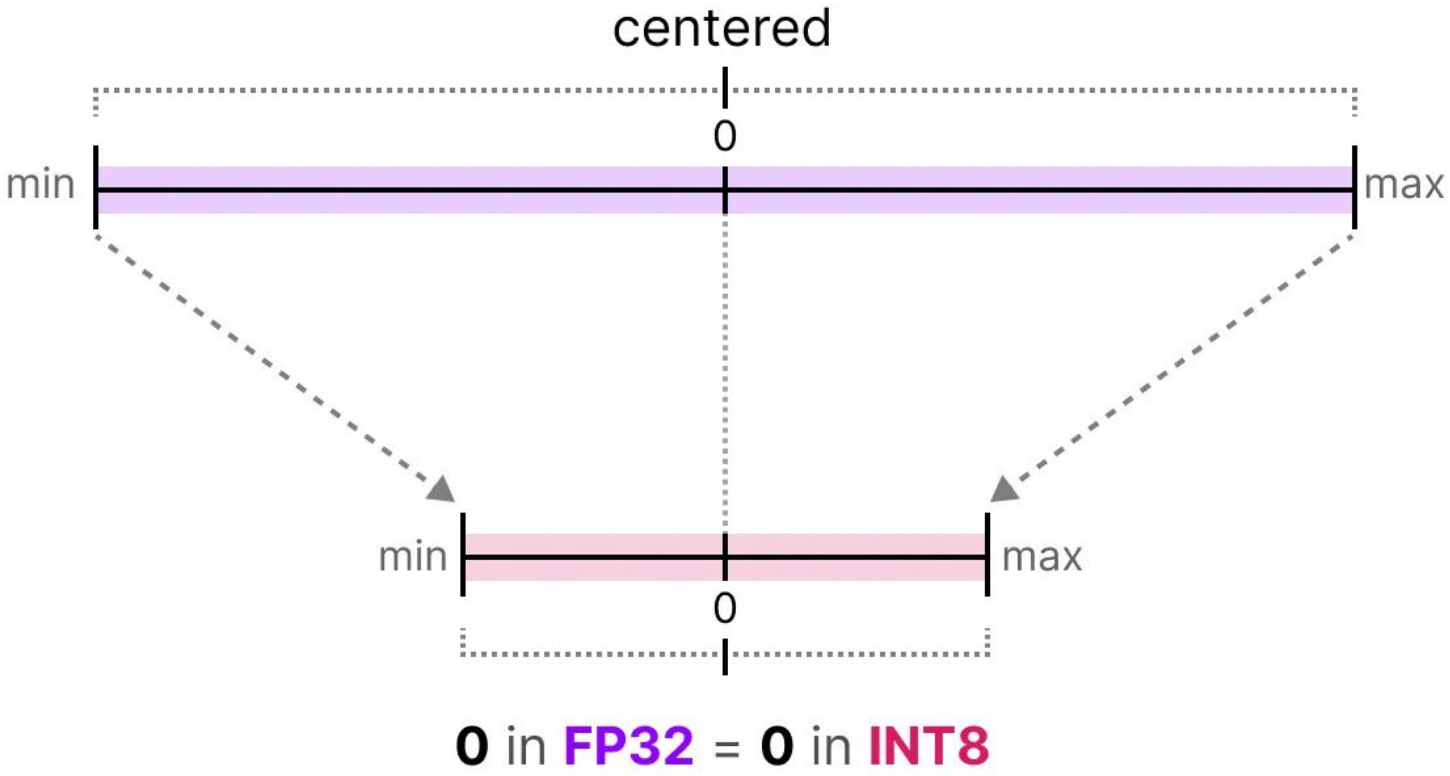
[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

Common data types: INT8



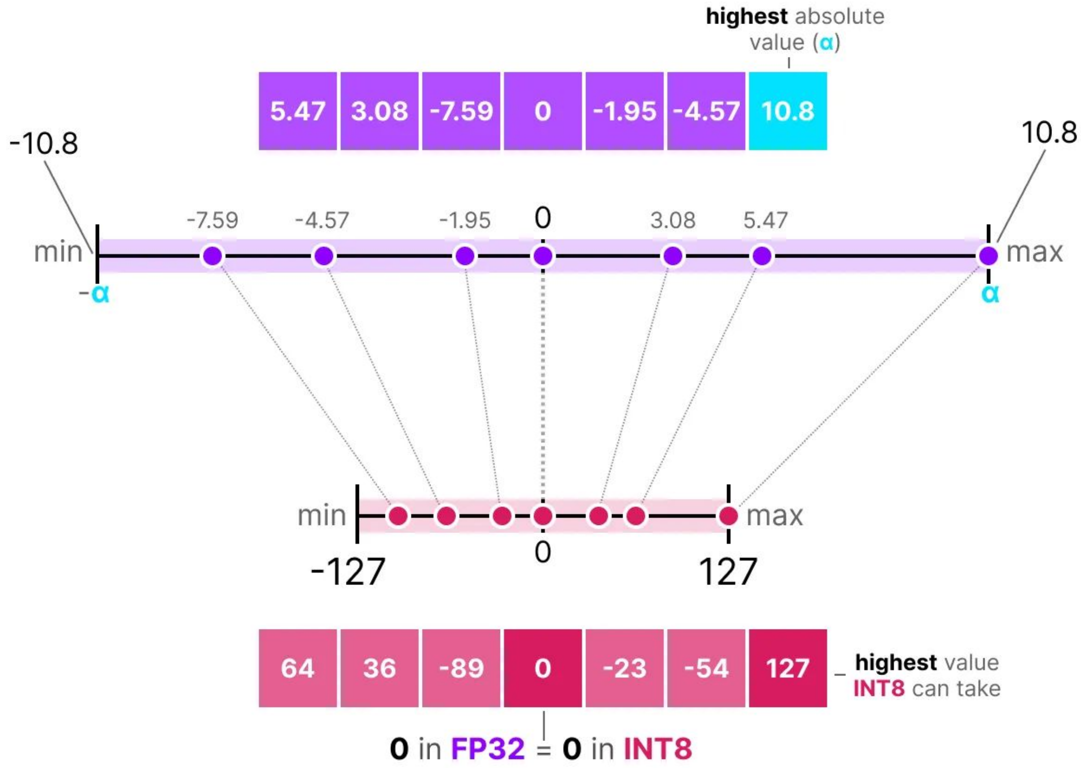
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Symmetric quantization



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Absolute maximum (absmax) quantization



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Absolute maximum (absmax) quantization

We first calculate a scale factor (*s*) using:

- ***b*** is the number of bytes that we want to quantize to (8),
- **α** is the *highest* absolute value,

Then, we use the ***s*** to quantize the input ***x***:

$$s = \frac{2^{b-1} - 1}{\alpha} \qquad \text{(scale factor)}$$

$$X_{\text{quantized}} = \text{round}\left(s \cdot X\right) \qquad \text{(quantization)}$$

Filling in the values would then give us the following:

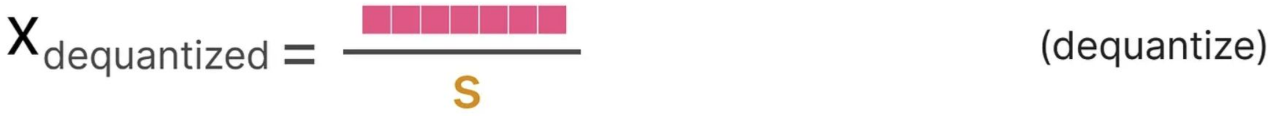
$$s = \frac{127}{10.8} = 11.76 \qquad \text{(scale factor)}$$

$$X_{\text{quantized}} = \text{round}\left(11.76 \cdot \right) \qquad \text{(quantization)}$$

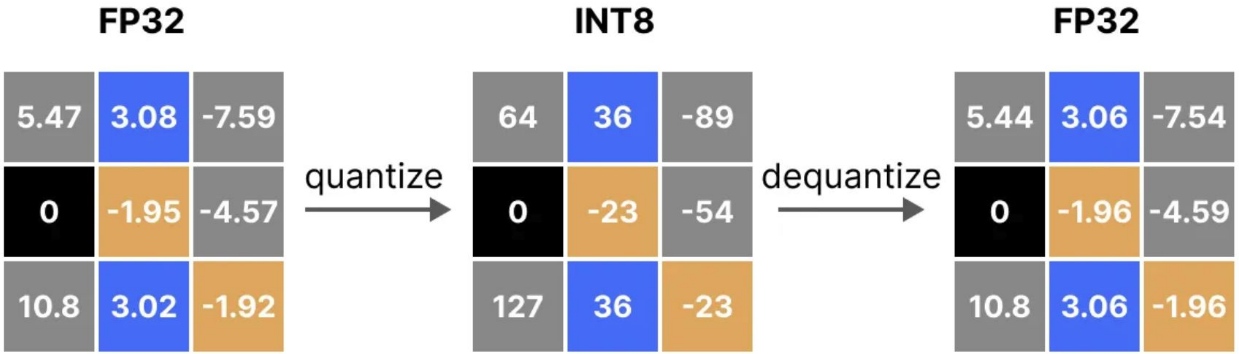
[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

Dequantization

To retrieve the original FP32 values, we can use the previously calculated *scaling factor* (*s*) to *dequantize* the quantized values.

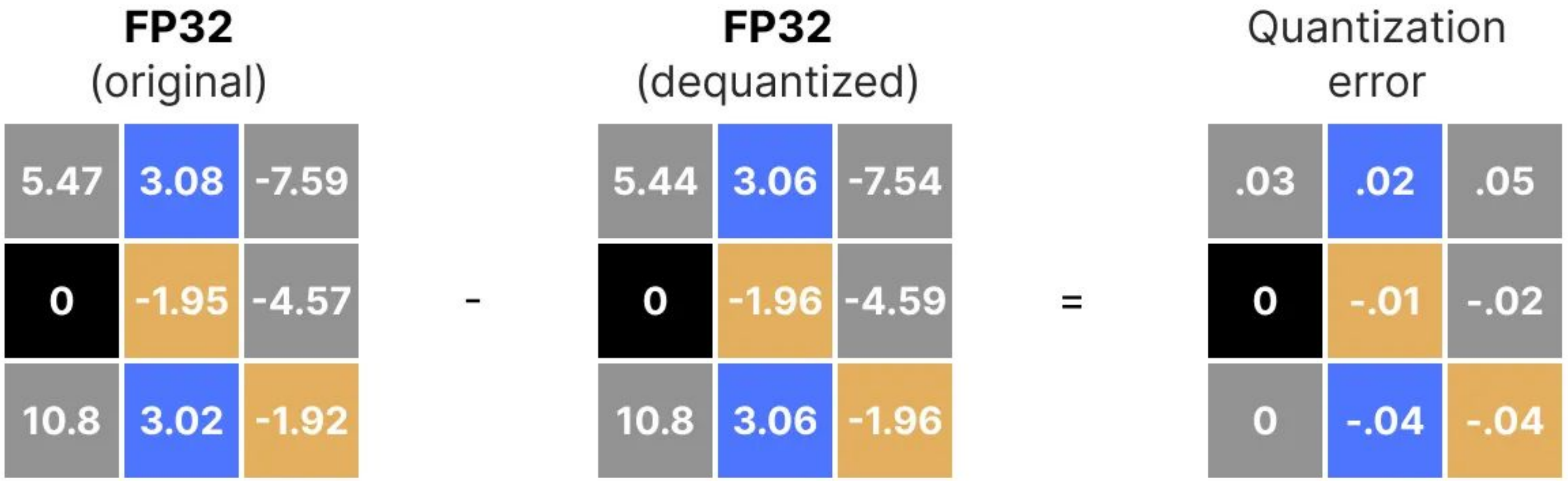


Applying the quantization and then dequantization process to retrieve the original looks as follows:



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Dequantization



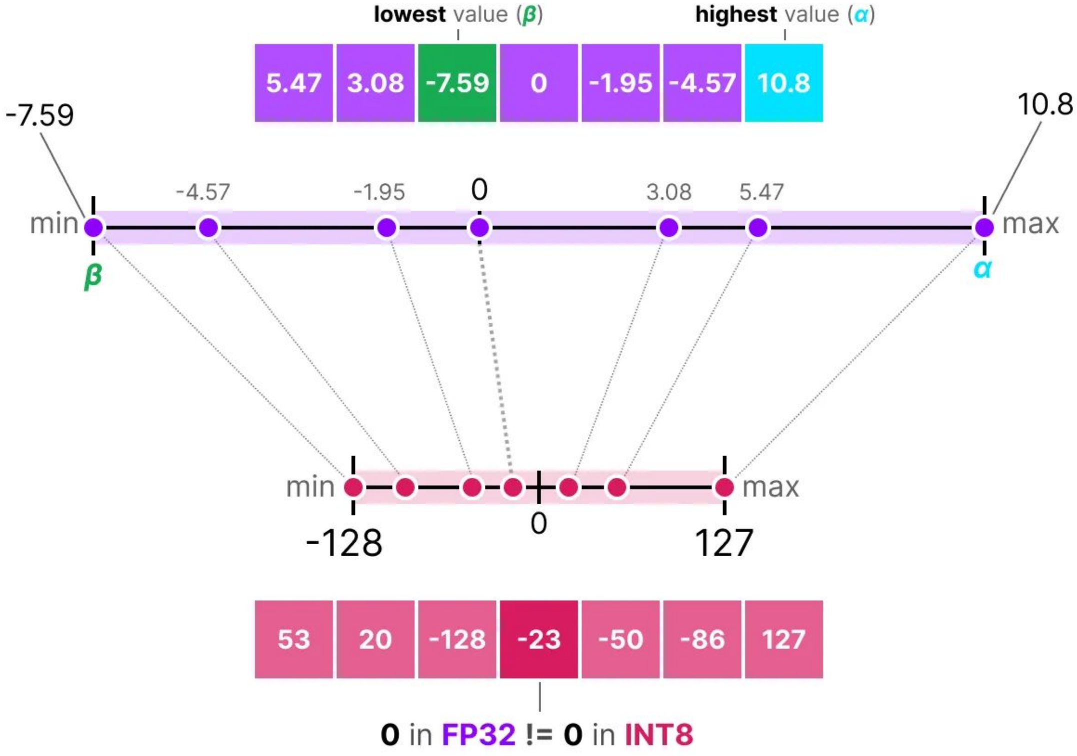
Maarten Grootendorst's blog: A Visual Guide to Quantization

Dequantization

$$\underset{\text{(original)}}{\overset{\textbf{FP32}}{\begin{bmatrix} 5.47 & 3.08 & -7.59 \\ 0 & -1.95 & -4.57 \\ 10.8 & 3.02 & -1.92 \end{bmatrix}}} - \underset{\text{(dequantized)}}{\overset{\textbf{FP32}}{\begin{bmatrix} 5.44 & 3.06 & -7.54 \\ 0 & -1.96 & -4.59 \\ 10.8 & 3.06 & -1.96 \end{bmatrix}}} = \underset{\text{error}}{\overset{\text{Quantization}}{\begin{bmatrix} .03 & .02 & .05 \\ 0 & -.01 & -.02 \\ 0 & -.04 & -.04 \end{bmatrix}}}$$

[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Asymmetric quantization



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Asymmetric quantization (cont'd)

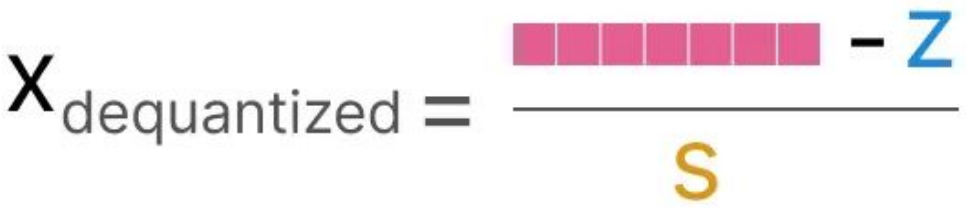
$$s = \frac{128 - -127}{\alpha - \beta}$$ (scale factor)

$$z = \text{round}\left(-s \cdot \beta\right) - 2^{b-1}$$ (zeropoint)

$$X_{\text{quantized}} = \text{round}\left(s \cdot X + z\right)$$ (quantization)

[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

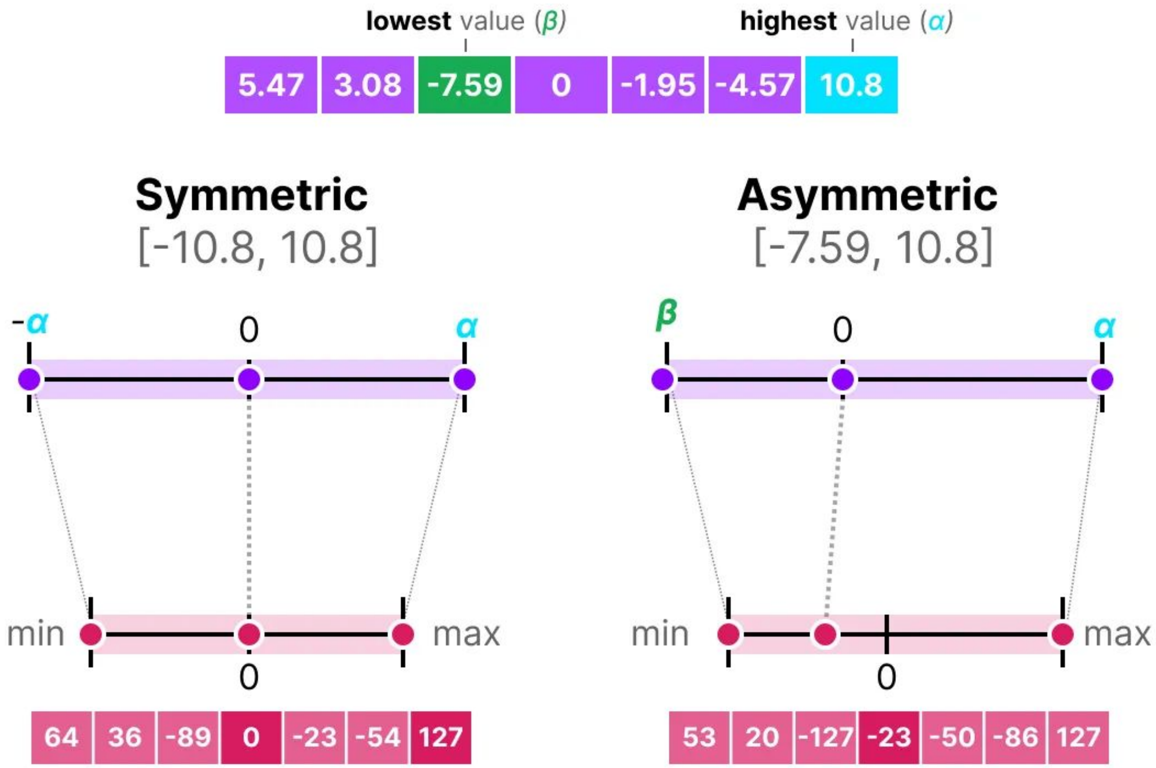
Asymmetric quantization (cont’d)



(dequantize)

[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Symmetric vs. Asymmetric quantization



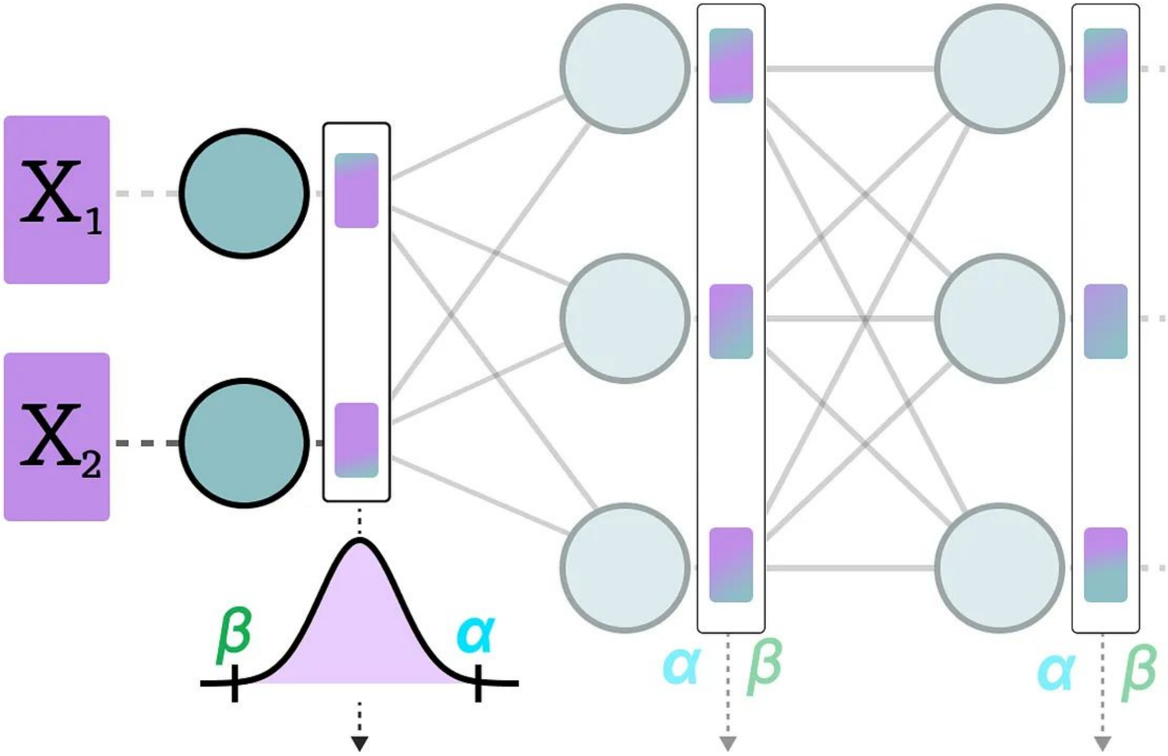
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Post-training quantization

- Dynamic Quantization
- Static Quantization

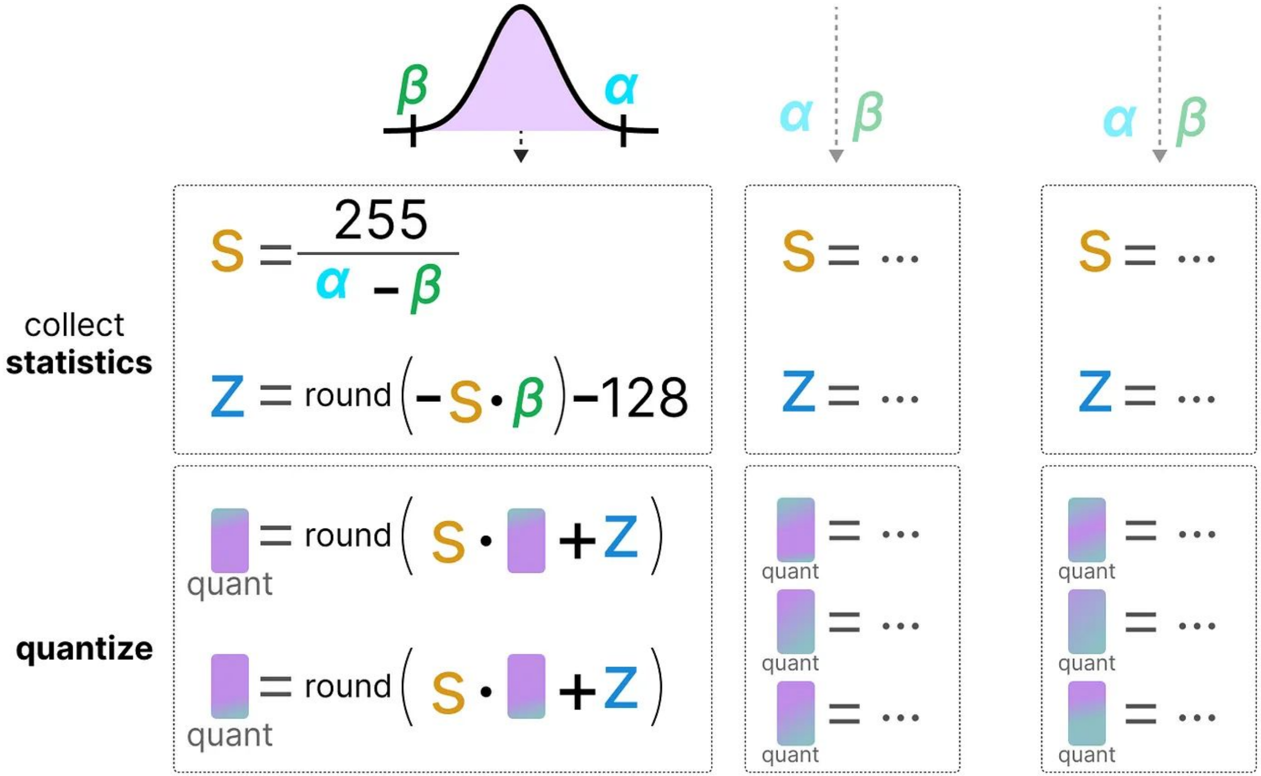
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Dynamic quantization



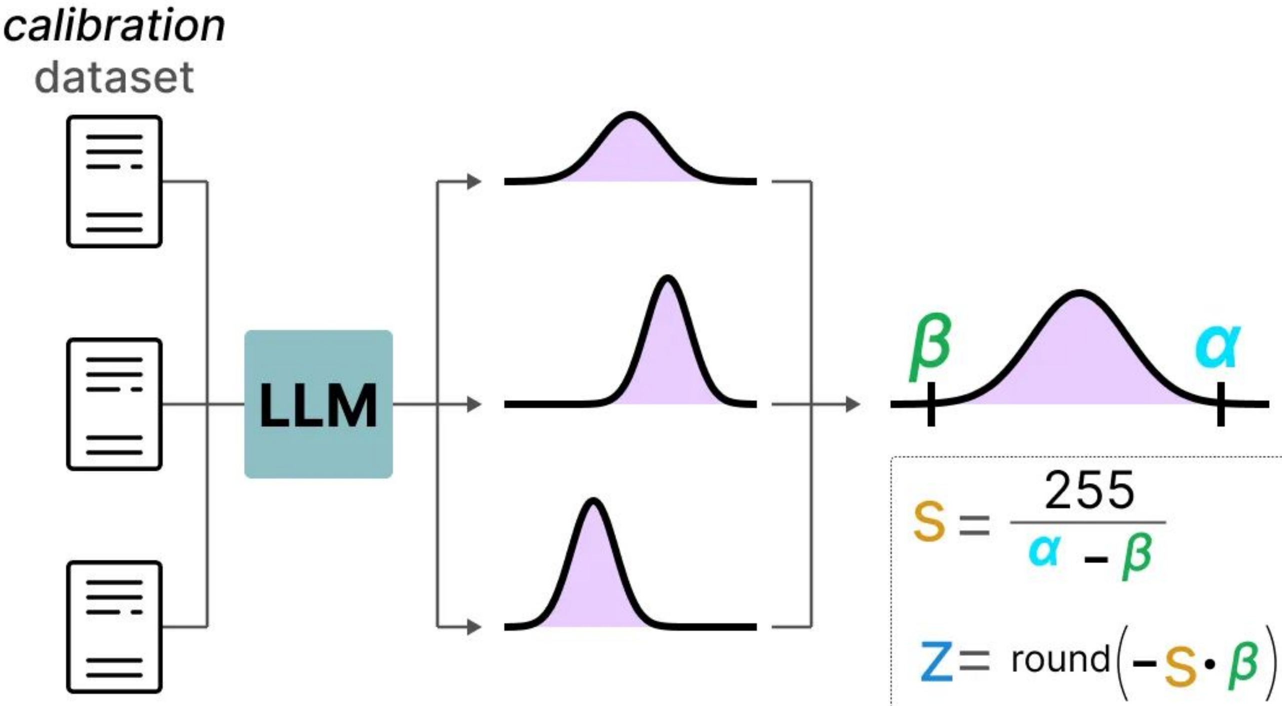
[Maarten Grootendorst's blog: A Visual Guide to Quantization]()

Dynamic quantization (cont'd)



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Static quantization



$$S = \frac{255}{\alpha - \beta}$$

$$Z = \text{round}\left(-S \cdot \beta\right) - 128$$

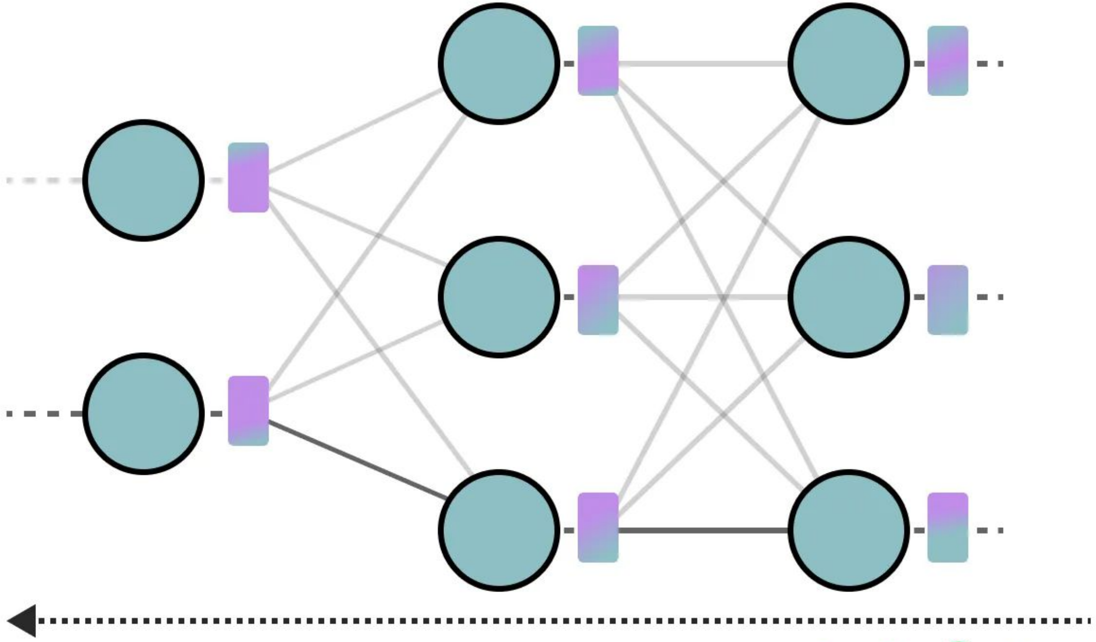
[Maarten Grootendorst's blog: A Visual Guide to Quantization]

The realm of 4-bit quantization

- GPTQ (full model on GPU)
- GGUF (potentially offload layers on the CPU)

Maarten Grootendorst's blog: A Visual Guide to Quantization

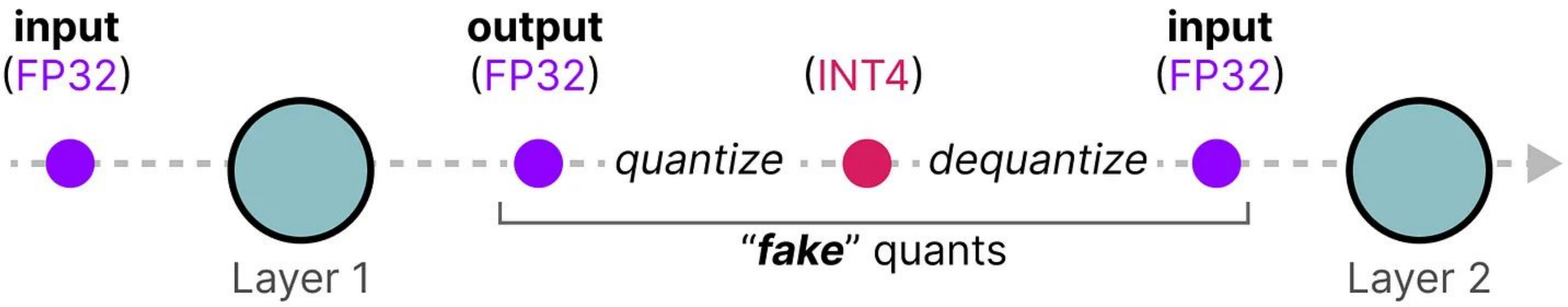
Quantization aware training



Learn **quantization parameters** (s, α, β, z)
during **backward pass**

[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Quantization aware training (cont'd)



[Maarten Grootendorst's blog: A Visual Guide to Quantization](#)

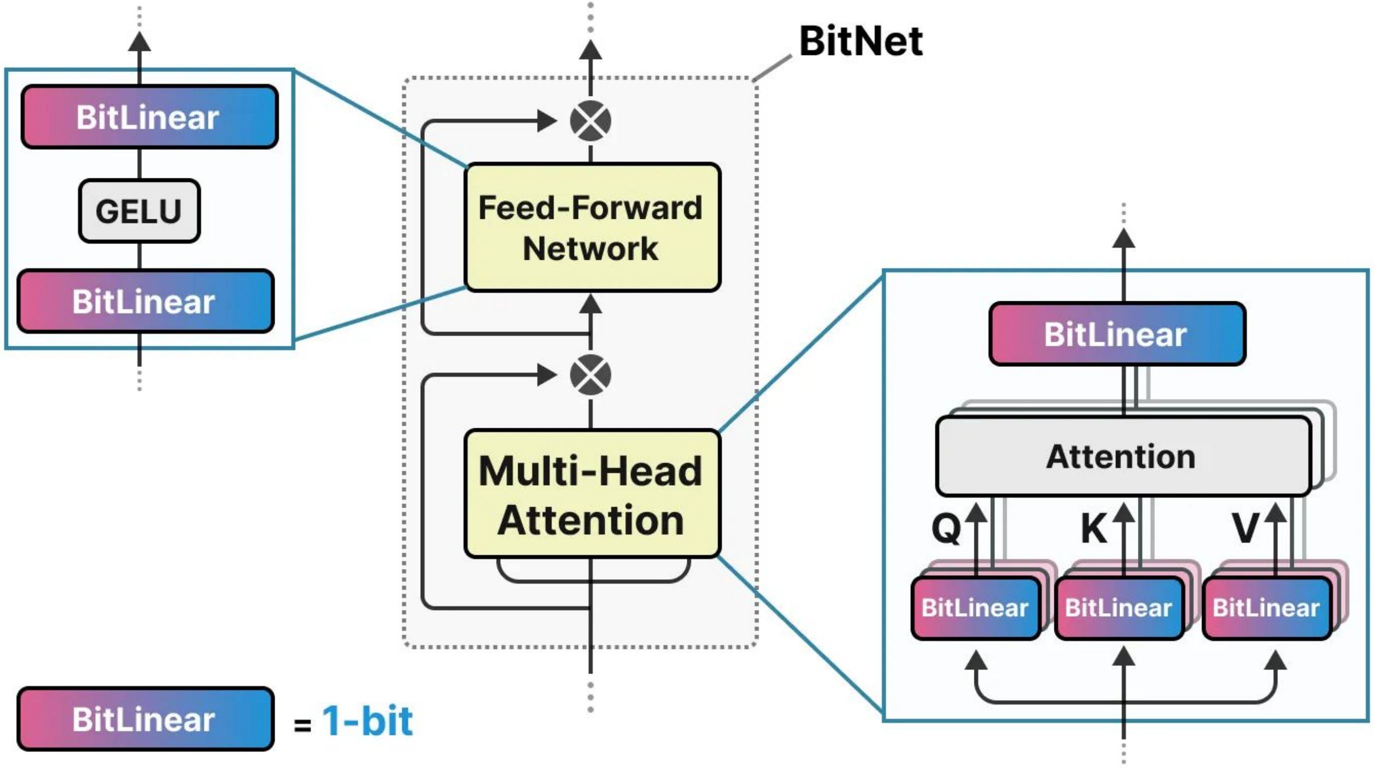
QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* **Artidoro Pagnoni*** **Ari Holtzman**

Luke Zettlemoyer

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu

The era of 1-bit LLMs: BitNet



[Maarten Grootendorst's blog: A Visual Guide to Quantization]

Thank you!