IERG5050 AI Foundation Models, Systems and Applications
Spring 2025

Part II: Efficient Transformer Architectures

Prof. Wing C. Lau
wclau@ie.cuhk.edu.hk
http://www.ie.cuhk.edu.hk/wclau

Acknowledgements

Many of the slides in this lecture are adapted from the sources below. Copyrights belong to the original authors.

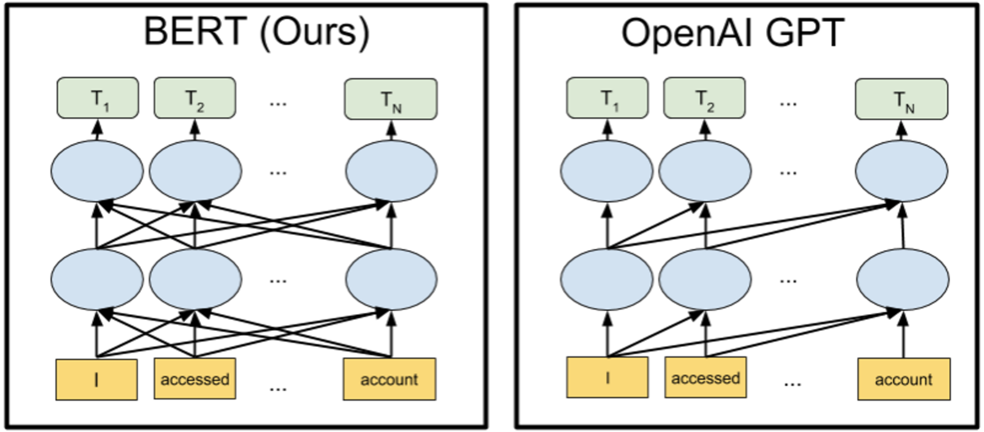
- Stanford CS336: Language Modeling from Scratch, Spring 2024
  - by Profs. Tatsunori Hashimoto, Percy Liang, https://stanford-cs336.github.io/spring2024/
- Stanford CS229S: Systems for Machine Learning, Fall 2023
  by Profs. Azalia Mirhoseini, Simran Arora, https://cs229s.stanford.edu/fall2023/
- CMU 11-667: Large Language Models: Methods and Applications, Fall 2024
  by Profs. Chenyan Xiong and Daphne Ippolito, https://cmu-llms.org
- CMU 11-711: Advanced Natural Language Processing (ANLP), Spring 2024
  by Prof. Graham Neubig, https://phontron.com/class/anlp2024/lectures/
- UPenn CIS7000: Large Language Models, Fall 2024
  by Prof. Mayur Naik, https://llm-class.github.io/schedule.html
- UWaterloo CS886: Recent Advances on Foundation Models, Winter 2024
  by Prof. Wenhu Chen, https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/
- MIT 6.5940: TinyML and Efficient Deep Learning Computing, Fall 2024
  by Prof. Song Han, https://hanlab.mit.edu/courses/2024-fall-65940
- UMD CMSC848K: Multimodal Foundation Models, Fall 2024
  by Prof. Jia-Bin Huang, https://jbhuang0604.github.io/teaching/CMSC848K/
- NeurIPS 2024 Invited Talk: “Systems for Foundation Models, and Foundation Models for Systems,”
  by Prof. Chris Re, Stanford.
- CUHK-SZ CSC6203: Large Language Models, Fall 2024
  by Prof. Benyou Wang, https://llm-course.github.io; https://github.com/FreedomIntelligence/CSC6203-LLM

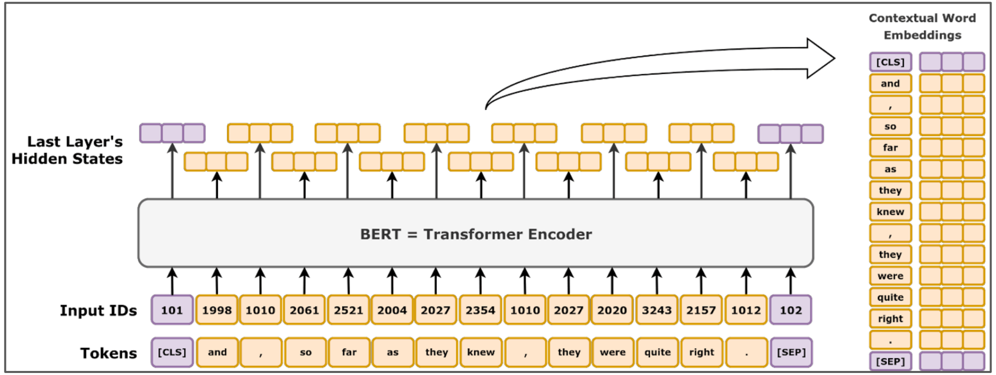
Case studies on some Recent Transformers

BERT: Bidirectional Encoder Representations from Transformers

Introduced by Google in 2018, it learned embeddings of text for use in downstream tasks. It’s major changes are:

- **Segment embeddings** in addition to token embeddings and position embeddings. All are learned!
- **Encoder-only** instead of encoder-decoder
- **Bidirectional** instead of unidirectional
- **Two simultaneous loss functions** with **masked language modeling** and **next sentence prediction**





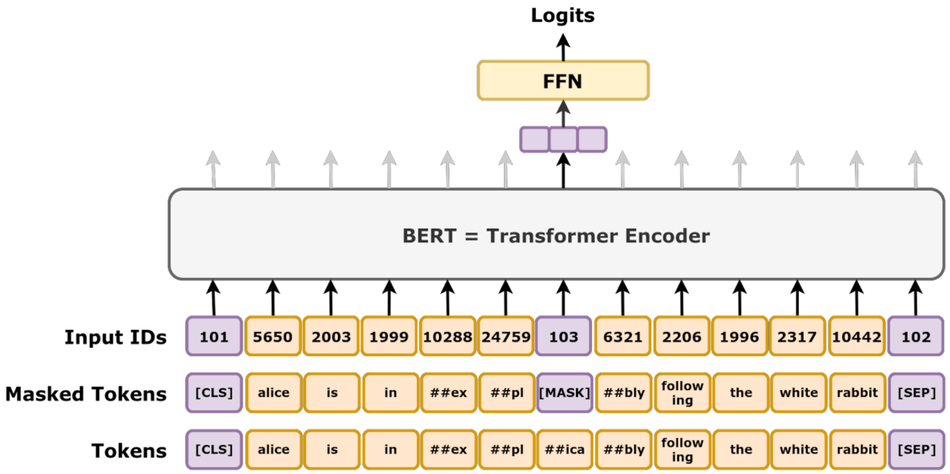
BERT: Masked Language Modeling

First, sample 15% of tokens in a sample.

Replace token with:

- [MASK] ~ 80%
- Random word token ~ 10%
- Not replaced ~ 10%

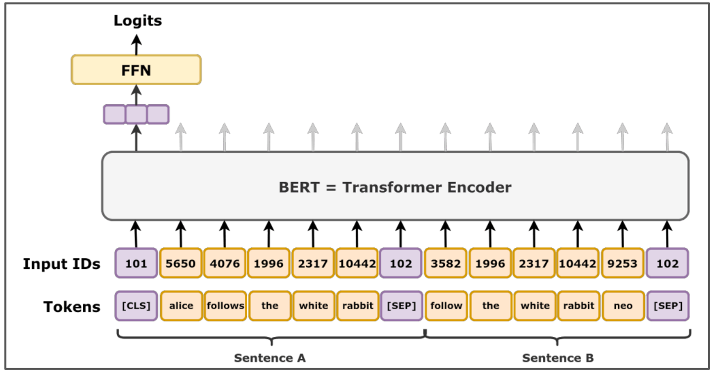
Pass sentence through the encoder and try to predict [MASK] with a simple linear layer + softmax!

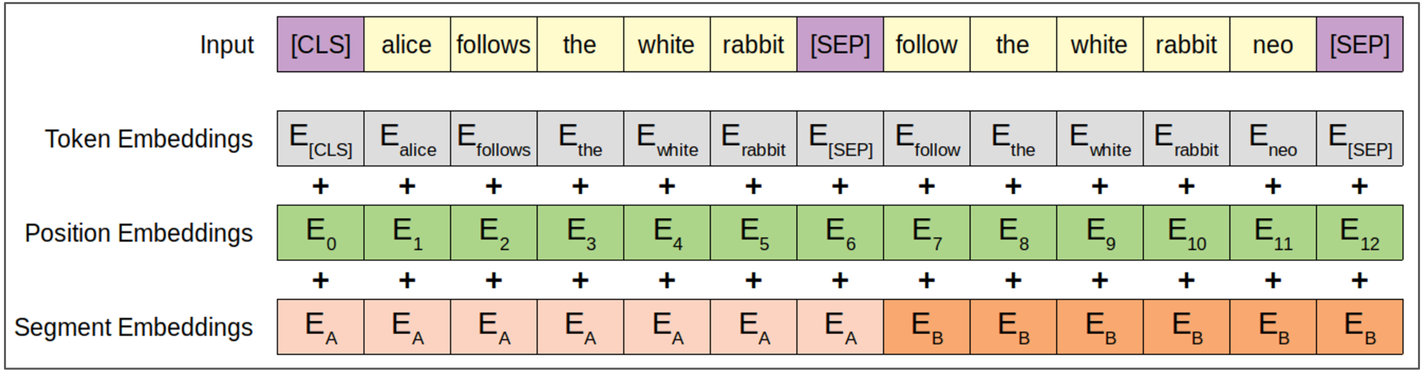


BERT: Next Sentence Prediction

Try to determine if one sentence follows another with simple binary classification. All embedding are learned! (unlike original Transformer).

Later works found this to not be useful …

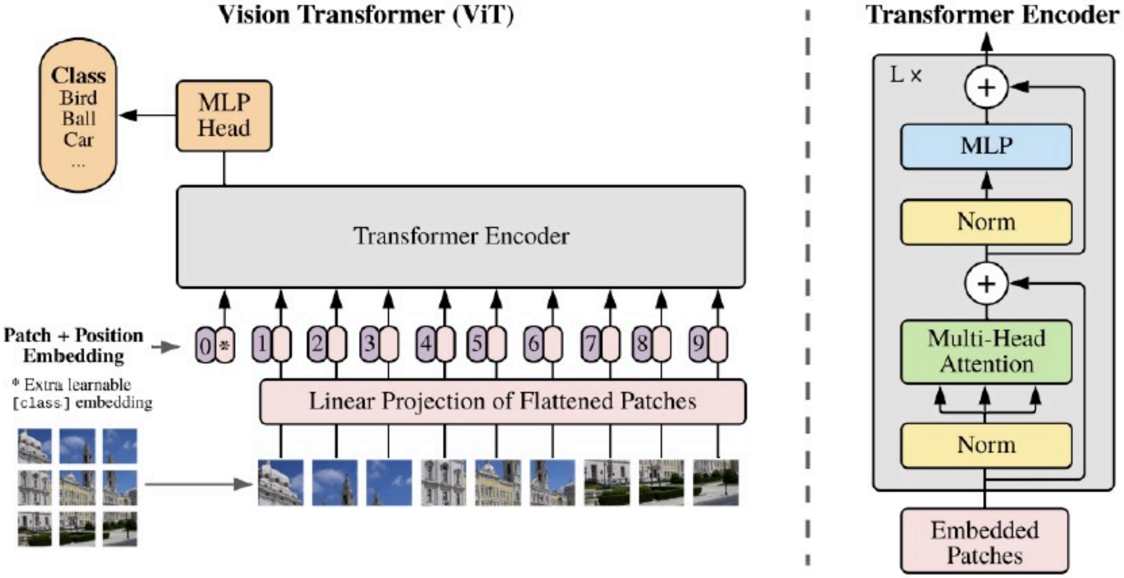


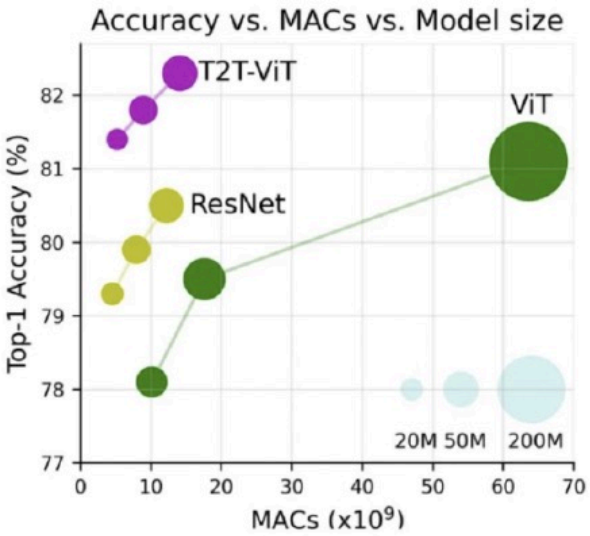


Input	[CLS]	alice	follows	the	white	rabbit	[SEP]	follow	the	white	rabbit	neo	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{alice}	$E_{follows}$	E_{the}	E_{white}	E_{rabbit}	$E_{[SEP]}$	E_{follow}	E_{the}	E_{white}	E_{rabbit}	E_{neo}	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
	+	+	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B	E_B

Vision Transformer (ViT)

- Applies vanilla **transformer encoder** to image classification
- Convert images to “sequences”:
  - Images are spliced into smaller regions
  - Regions are flattened and treated as a sequence



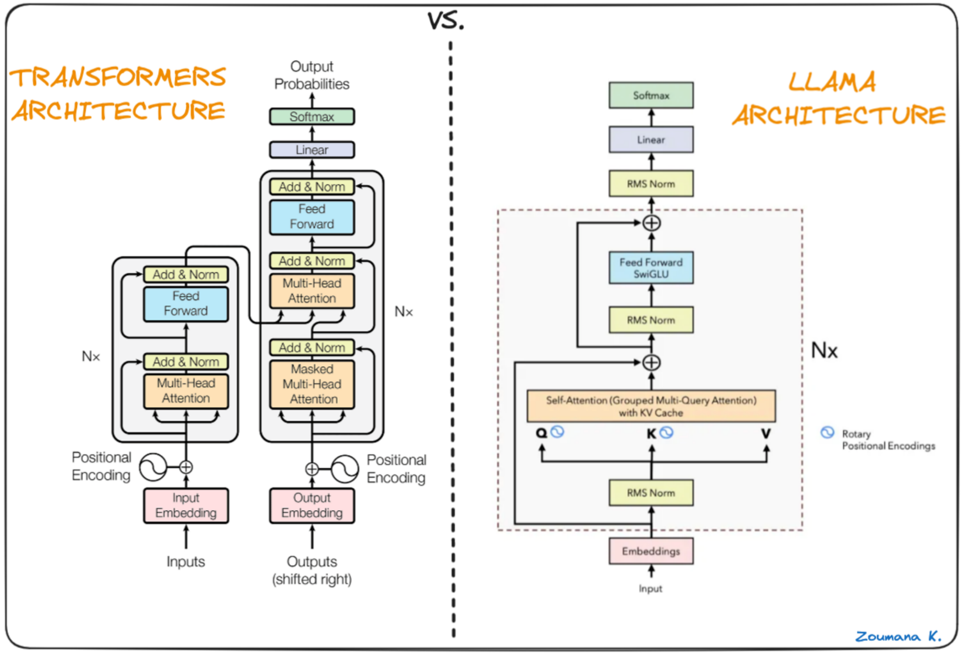


(Dosovitskiy et al., 2021): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

The Llama Family of Transformers from Meta

"Open-source"** Autoregressive LLM first released by Meta in Feb 2023. (Multiple generations since then.)
Changes include:

- **Decoder-only** instead of encoder-decoder
- **RMSNorm** instead of LayerNorm
- **SwiGLU** activation instead of GeLU
- Use Grouped Query Attention (**GQA**)
- **Rotary positional embeddings** instead of absolute positional embeddings



** Here, "open source" means
(1) open-source inference codes + (2) open-weights BUT not open-datasets NOR open data-cleansing/ tuning procedural details or scripts NOR codes for training the model.
Contrast this with the "Open EVERTHING" philosophy of the OMLo family from Ai2 https://allenai.org/olmo

RMS Normalization in Llama

In LayerNorm, we re-center (subtracting from mean) and re-scale (divide by standard deviation) across (sequence length, embedding_dim) dimensions.

Zhang et al. propose that **only re-scaling matters**. This saves a small amount of compute by not needed to re-center.

Linear Layer

$$y_i = f\left(\bar{a}_i + b_i\right)$$

LayerNorm

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} g_i,$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}.$$

RMSNorm

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

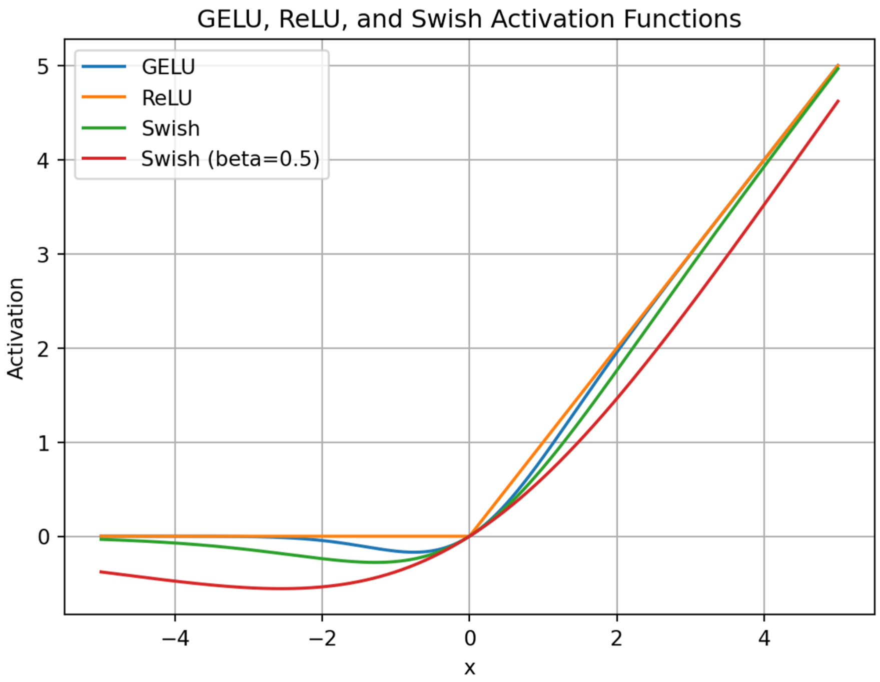
Swish-Gated Linear Unit (SwiGLU)

Swish(x) = sigmoid(β*x), β is hyperparam

GLU(x) = x*sigmoid(Wx+b); W,b is learned

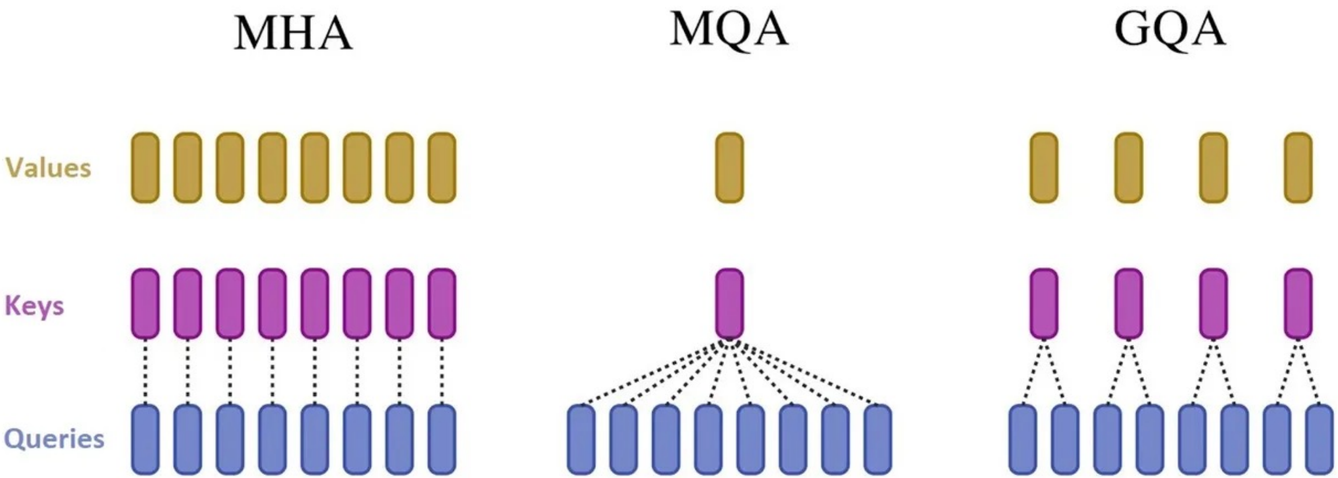
SwiGLU(x) = x * sigmoid(β * x) +
(1 - sigmoid(β * x)) * (Wx + b)

Smoother than ReLU, non-monotonic,



Grouped Query Attention (GQA)

Multi-Head Attention | Multi-Query Attention | Grouped Query Attention



High quality

Computationally slow

Attention is All You Need (2017)

Loss in quality

Computationally fast

Fast Transformer Decoder:
One Write-Head
is All You Need (2019)

A good compromise

between quality and speed

GQA: Training Generalized
Multi-Query Transformers from
Multi-Head Checkpoints (2023)

GQA interpolates between MHA and MQA. It reduces Memory Bandwidth overhead during inference time while avoiding excessive loss in accuracy as fewer K and V matrices are loaded into the Decoder (KV-cache in GPU RAM)

“Growing Pains” of Transformers

What would we like to fix about the Transformer?

The Demand of Getting Bigger Models, with Longer Context Length to provide more capabilities and better accuracies (Emergent Behaviors, Scaling Laws of LLM) without getting slower (especially when serving the models in real-time):

- But bigger, and longer-context models demand more Compute and Memory

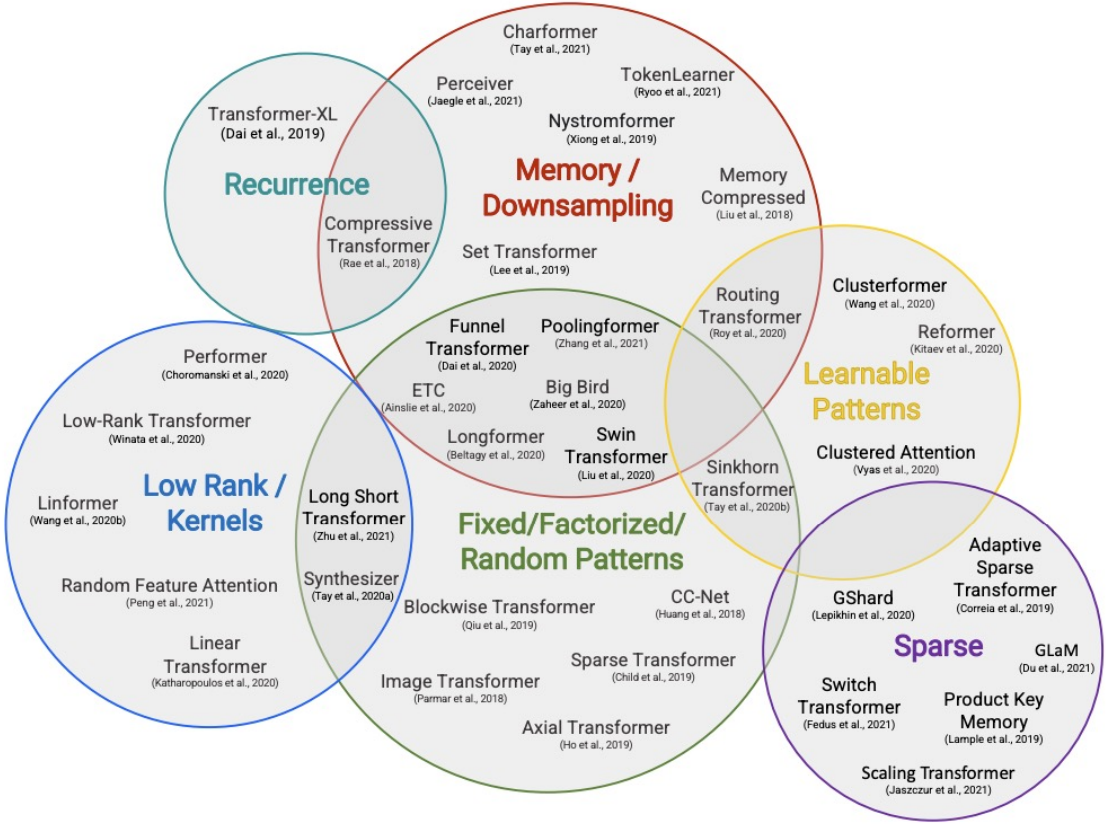
Quadratic compute in self-attention:

- Computing all pairs of interactions means our computation grows quadratically with the sequence length!
- For recurrent models, it only grew linearly!
- Large Memory and GPU Memory Bandwidth (I/O) Requirements for large K, Q, V matrices, especially during inference times 1

Need More Robust Position representations:

- Absolute Positional Encoding vs. Relative Positional Encoding
- How to generalize to Context-length change ? **[During Training != during Inference]**

Efficient Transformers



Charformer
(Tay et al., 2021)

Perceiver
(Jaegle et al., 2021)

TokenLearner
(Ryoo et al., 2021)

Transformer-XL
(Dai et al., 2019)

Nystromformer
(Xiong et al., 2019)

Recurrence

Memory / Downsampling

Memory Compressed
(Liu et al., 2018)

Compressive Transformer
(Rae et al., 2018)

Set Transformer
(Lee et al., 2019)

Routing Transformer
(Roy et al., 2020)

Clusterformer
(Wang et al., 2020)

Funnel Transformer
(Dai et al., 2020)

Poolingformer
(Zhang et al., 2021)

Reformer
(Kitaev et al., 2020)

Performer
(Choromanski et al., 2020)

ETC
(Ainslie et al., 2020)

Big Bird
(Zaheer et al., 2020)

Learnable Patterns

Low-Rank Transformer
(Winata et al., 2020)

Longformer
(Beltagy et al., 2020)

Swin Transformer
(Liu et al., 2020)

Clustered Attention
(Vyas et al., 2020)

Sinkhorn Transformer
(Tay et al., 2020b)

Low Rank / Kernels

Linformer
(Wang et al., 2020b)

Long Short Transformer
(Zhu et al., 2021)

Fixed/Factorized/ Random Patterns

Adaptive Sparse Transformer
(Correia et al., 2019)

Synthesizer
(Tay et al., 2020a)

Random Feature Attention
(Peng et al., 2021)

Blockwise Transformer
(Qiu et al., 2019)

CC-Net
(Huang et al., 2018)

GShard
(Lepikhin et al., 2020)

Sparse

GLaM
(Du et al., 2021)

Linear Transformer
(Katharopoulos et al., 2020)

Sparse Transformer
(Child et al., 2019)

Image Transformer
(Parmar et al., 2018)

Switch Transformer
(Fedus et al., 2021)

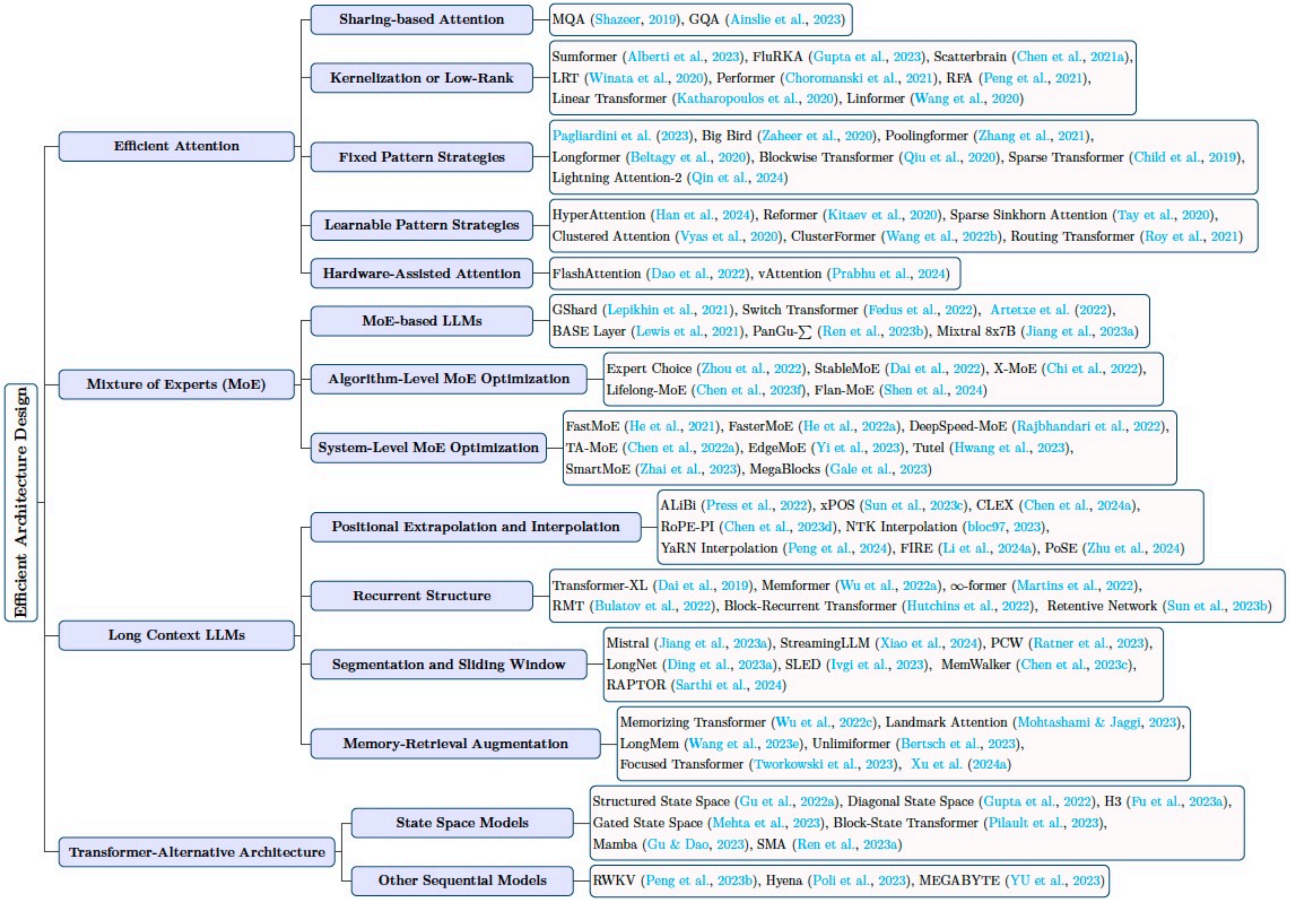
Product Key Memory
(Lample et al., 2019)

Axial Transformer
(Ho et al., 2019)

Scaling Transformer
(Jaszczur et al., 2021)

(Tay et al., 2020): Efficient Transformers: A Survey

Efficient Architecture Designs for LLMs



- **Efficient Architecture Design**
  - **Efficient Attention**
    - **Sharing-based Attention**
      - MQA (Shazeer, 2019), GQA (Ainslie et al., 2023)
    - **Kernelization or Low-Rank**
      - Sumformer (Alberti et al., 2023), FluRKA (Gupta et al., 2023), Scatterbrain (Chen et al., 2021a), LRT (Winata et al., 2020), Performer (Choromanski et al., 2021), RFA (Peng et al., 2021), Linear Transformer (Katharopoulos et al., 2020), Linformer (Wang et al., 2020)
    - **Fixed Pattern Strategies**
      - Pagliardini et al. (2023), Big Bird (Zaheer et al., 2020), Poolingformer (Zhang et al., 2021), Longformer (Beltagy et al., 2020), Blockwise Transformer (Qiu et al., 2020), Sparse Transformer (Child et al., 2019), Lightning Attention-2 (Qin et al., 2024)
    - **Learnable Pattern Strategies**
      - HyperAttention (Han et al., 2024), Reformer (Kitaev et al., 2020), Sparse Sinkhorn Attention (Tay et al., 2020), Clustered Attention (Vyas et al., 2020), ClusterFormer (Wang et al., 2022b), Routing Transformer (Roy et al., 2021)
    - **Hardware-Assisted Attention**
      - FlashAttention (Dao et al., 2022), vAttention (Prabhu et al., 2024)
  - **Mixture of Experts (MoE)**
    - **MoE-based LLMs**
      - GShard (Lepikhin et al., 2021), Switch Transformer (Fedus et al., 2022), Artetxe et al. (2022), BASE Layer (Lewis et al., 2021), PanGu-$\sum$ (Ren et al., 2023b), Mixtral 8x7B (Jiang et al., 2023a)
    - **Algorithm-Level MoE Optimization**
      - Expert Choice (Zhou et al., 2022), StableMoE (Dai et al., 2022), X-MoE (Chi et al., 2022), Lifelong-MoE (Chen et al., 2023f), Flan-MoE (Shen et al., 2024)
    - **System-Level MoE Optimization**
      - FastMoE (He et al., 2021), FasterMoE (He et al., 2022a), DeepSpeed-MoE (Rajbhandari et al., 2022), TA-MoE (Chen et al., 2022a), EdgeMoE (Yi et al., 2023), Tutel (Hwang et al., 2023), SmartMoE (Zhai et al., 2023), MegaBlocks (Gale et al., 2023)
  - **Long Context LLMs**
    - **Positional Extrapolation and Interpolation**
      - ALiBi (Press et al., 2022), xPOS (Sun et al., 2023c), CLEX (Chen et al., 2024a), RoPE-PI (Chen et al., 2023d), NTK Interpolation (bloc97, 2023), YaRN Interpolation (Peng et al., 2024), FIRE (Li et al., 2024a), PoSE (Zhu et al., 2024)
    - **Recurrent Structure**
      - Transformer-XL (Dai et al., 2019), Memformer (Wu et al., 2022a), ∞-former (Martins et al., 2022), RMT (Bulatov et al., 2022), Block-Recurrent Transformer (Hutchins et al., 2022), Retentive Network (Sun et al., 2023b)
    - **Segmentation and Sliding Window**
      - Mistral (Jiang et al., 2023a), StreamingLLM (Xiao et al., 2024), PCW (Ratner et al., 2023), LongNet (Ding et al., 2023a), SLED (Ivgi et al., 2023), MemWalker (Chen et al., 2023c), RAPTOR (Sarthi et al., 2024)
    - **Memory-Retrieval Augmentation**
      - Memorizing Transformer (Wu et al., 2022c), Landmark Attention (Mohtashami & Jaggi, 2023), LongMem (Wang et al., 2023e), Unlimiformer (Bertsch et al., 2023), Focused Transformer (Tworkowski et al., 2023), Xu et al. (2024a)
  - **Transformer-Alternative Architecture**
    - **State Space Models**
      - Structured State Space (Gu et al., 2022a), Diagonal State Space (Gupta et al., 2022), H3 (Fu et al., 2023a), Gated State Space (Mehta et al., 2023), Block-State Transformer (Pilault et al., 2023), Mamba (Gu & Dao, 2023), SMA (Ren et al., 2023a)
    - **Other Sequential Models**
      - RWKV (Peng et al., 2023b), Hyena (Poli et al., 2023), MEGABYTE (YU et al., 2023)

Source: Zhongwei Wan et al., May 2024): Efficient Large Language Models: A Survey, Trans. On MLR.

Positional Encoding

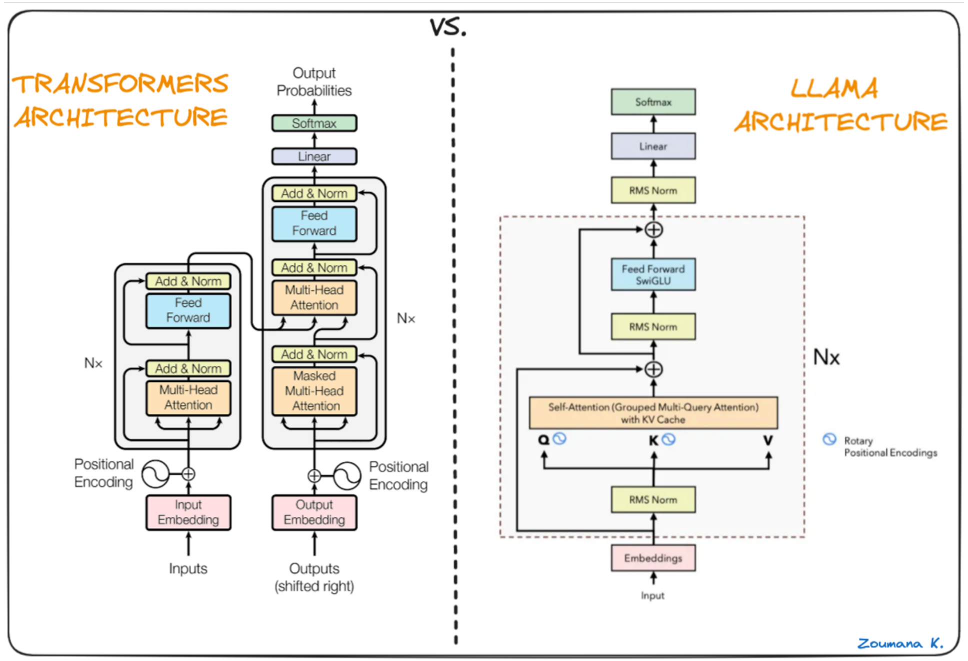
Slides from video of
Jia-Bin Huang
University of Maryland College Park

https://www.youtube.com/watch?v=SMBkImDWOyQ

The Llama Family of Transformers from Meta

"Open-sourced"** Autoregressive LLM first released by Meta in Feb 2023. (Multiple generations since then.)
Changes include:

- **Decoder-only** instead of encoder-decoder
- **RMSNorm** instead of LayerNorm
- **SwiGLU** activation instead of GeLU
- Use Grouped Query Attention (**GQA**)
- **Rotary positional embeddings** instead of absolute positional embeddings



** Here, "open sourced" means
(1) open-source inference codes + (2) open-weights BUT not open-datasets NOR open data-cleansing/ tuning procedural details or scripts NOR codes for training the model.
Contrast this with the "Open EVERTHING" philosophy of the OMLo family from Ai2 https://allenai.org/olmo

Variants of Positional encodings to tackle Context-length Generalization

Position k

Angular frequency

$w_i = N^{-2i/d}$

$N = 100{,}000$

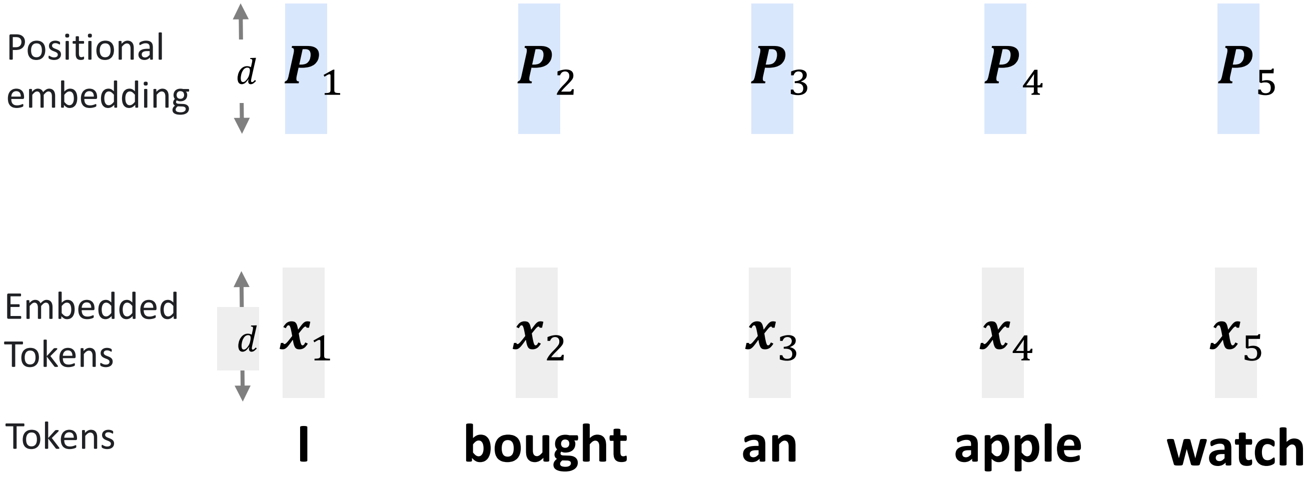
$$d \begin{bmatrix} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{bmatrix}$$

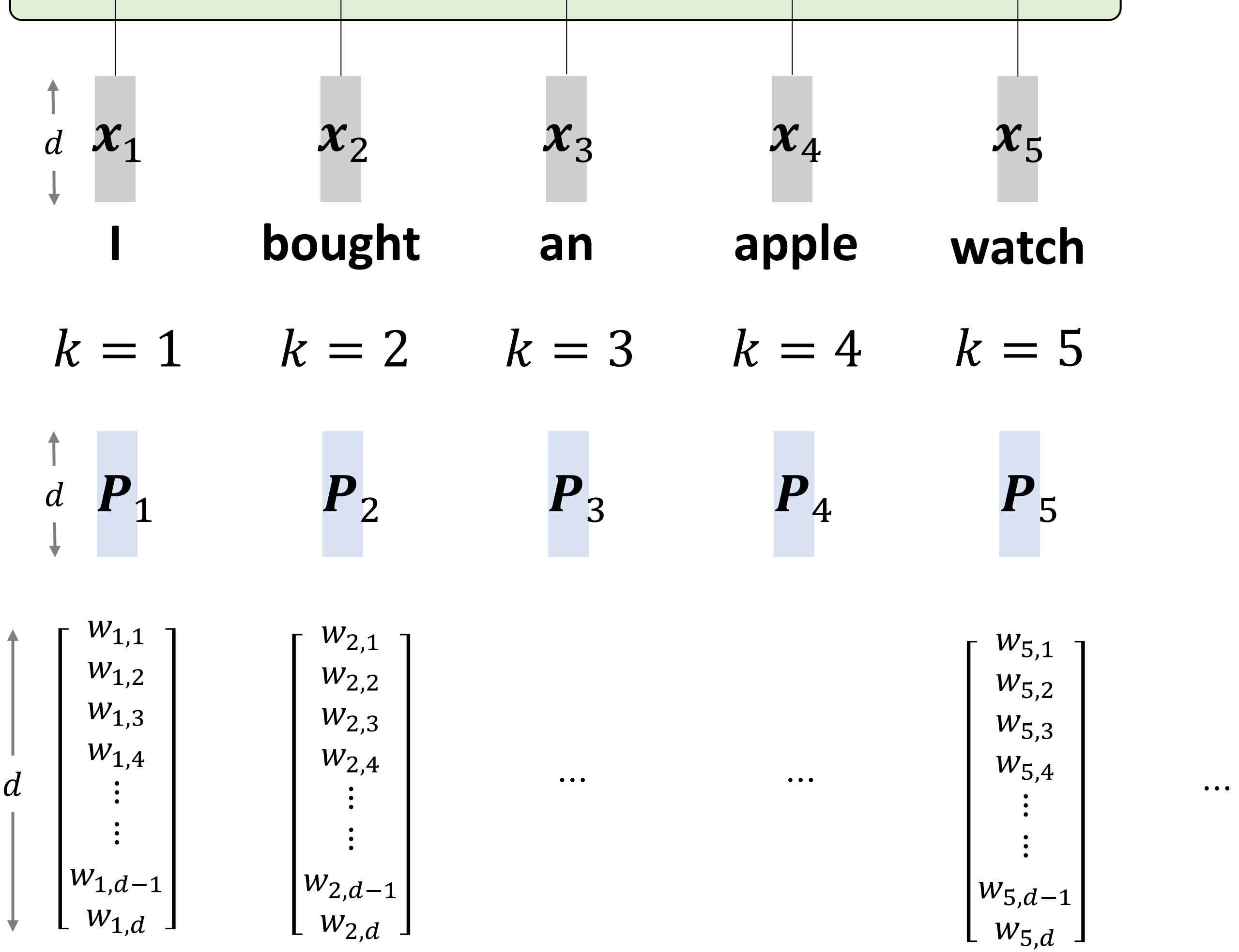
Absolute Positional Encoding schemes:

e.g. Sinusoidal positional encoding, Learned (in BERT)

But cannot generalize to sequence of unseen context-length: Training length != Inferencing length

Relative positional encoding: e.g. T5 bias, and many more…





Embedded Tokens d x_1 x_2 x_3 x_4 x_5

Tokens **I** **bought** **an** **apple** **watch**

Position $k = 1$ $k = 2$ $k = 3$ $k = 4$ $k = 5$

Length L d P_1 P_2 P_3 P_4 P_5

Dimension d

#Parameters $d \times L$

$$d \quad \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ w_{1,4} \\ \vdots \\ \vdots \\ w_{1,d-1} \\ w_{1,d} \end{bmatrix} \quad \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \\ w_{2,4} \\ \vdots \\ \vdots \\ w_{2,d-1} \\ w_{2,d} \end{bmatrix} \quad \cdots \quad \cdots \quad \begin{bmatrix} w_{5,1} \\ w_{5,2} \\ w_{5,3} \\ w_{5,4} \\ \vdots \\ \vdots \\ w_{5,d-1} \\ w_{5,d} \end{bmatrix} \quad \cdots$$

P_{L+1}?

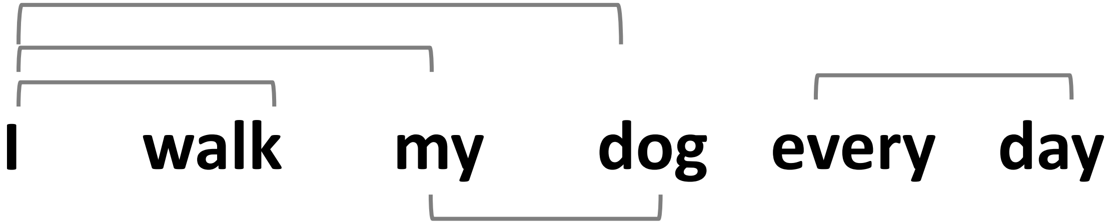
FAIL

Position	1	2	3	4	5	6
	I	**walk**	**my**	**dog**	**every**	**day**

Position	1	2	3	4	5	6
	I	**walk**	**my**	**dog**	**every**	**day**

Position	1	2	3	4	5	6
	every	**day**	**I**	**walk**	**my**	**dog**

Position



I walk my dog every day

Position

every day I walk my dog

Challenge for Positional Encoding Schemes

- How to generalize the model when

  Context-length during Training << (unseen) Context-length during Inference

while enabling high inference speed !

- Many different schemes proposed, still active research:
  - ALiBi, KERPLE, RoPE, LongRoPE, NoPE, CoPE, YaRN, FIRE, etc..

Rotary Position Embedding (RoPE)

So far, we have seen two kinds of position embeddings:

- Absolute Positional Encoding: e.g. Sinusoidal [Vaswani et al. 2017] and learned (BERT) ;
- Relative Positional Encoding: e.g. T5-bias,

BUT they require “Known” target context length !

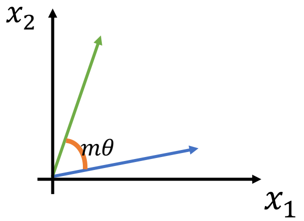
Instead of adding extra numbers, RoPE rotates embeddings based on their position so that the relative position of tokens can be considered in the attention calculations rather than their absolute positions. **The angles between embedding vectors maintain the same proportional relationship as the distance between tokens in the sequence.**

$$f_q(\boldsymbol{x}_m, m) = R^m_\Theta \, \boldsymbol{W}_q \boldsymbol{x}_m \qquad R^m_\Theta = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$

Query vector at position m

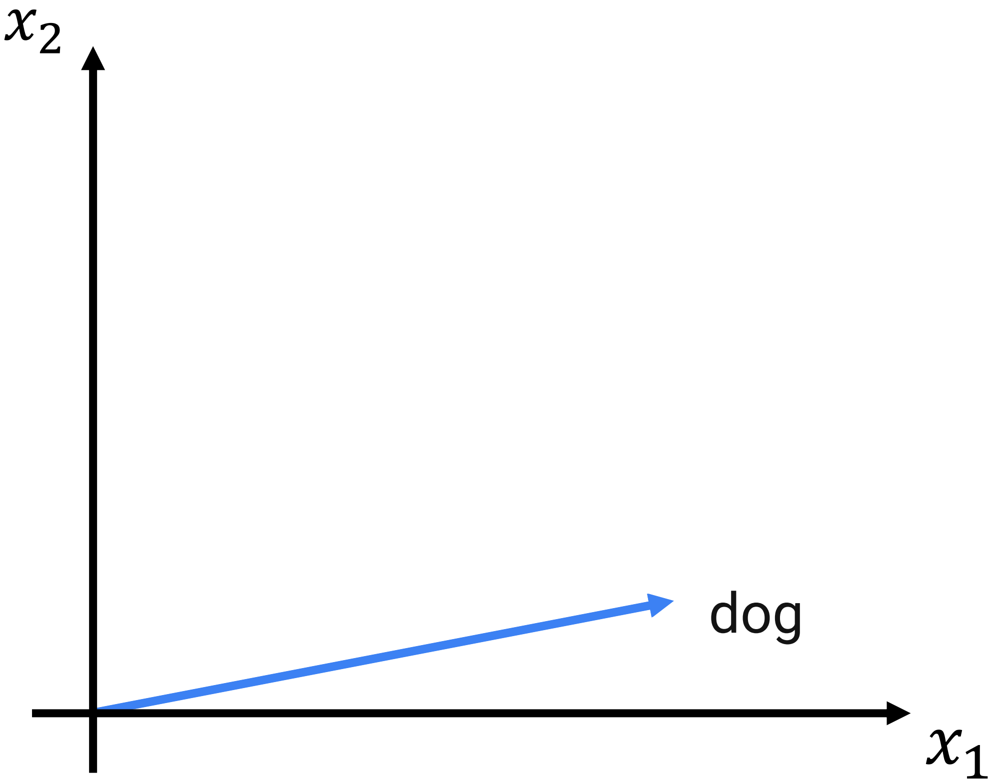
$$f_k(\boldsymbol{x}_n, n) = R^n_\Theta \, \boldsymbol{W}_k \boldsymbol{x}_n$$

key vector at position n

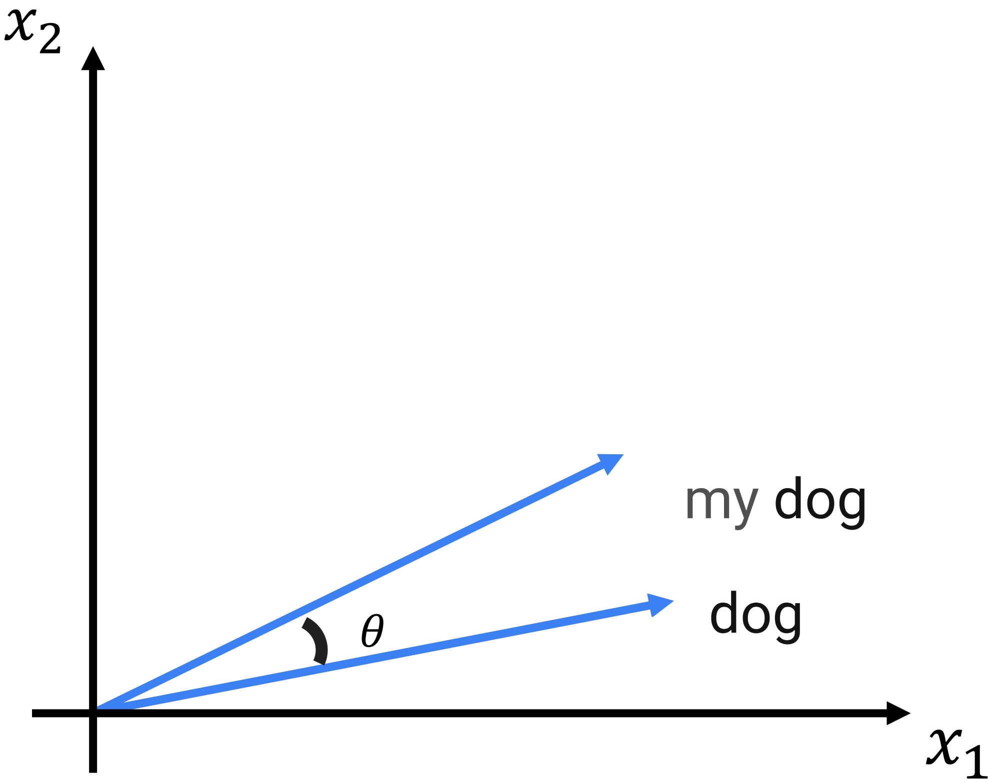


$$\alpha_{m,n} = < f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n) >$$

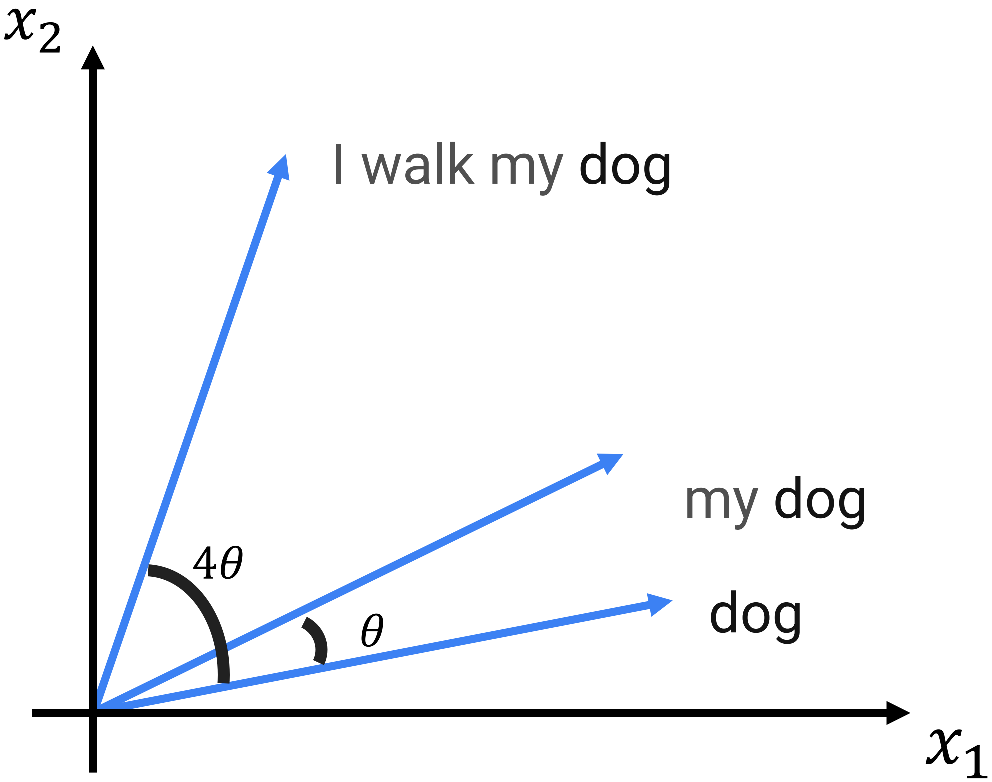
$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$



Rotary Positional Embeddings, 2021



Rotary Positional Embeddings, 2021



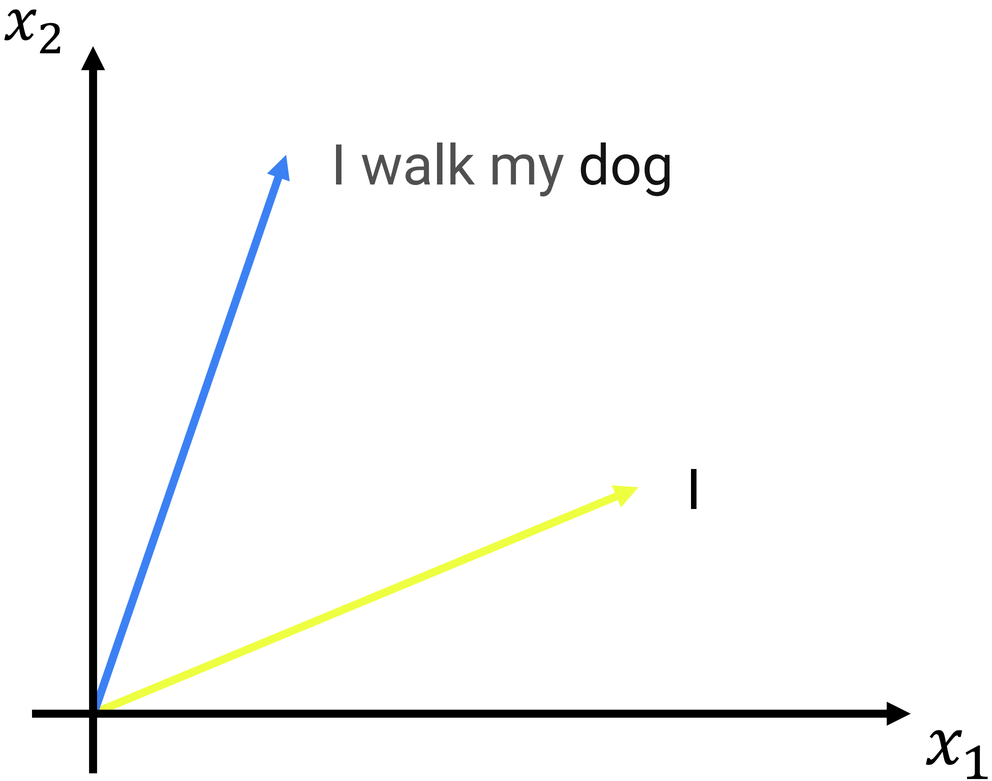
Rotary Positional Embeddings, 2021

I walk my **dog** every day

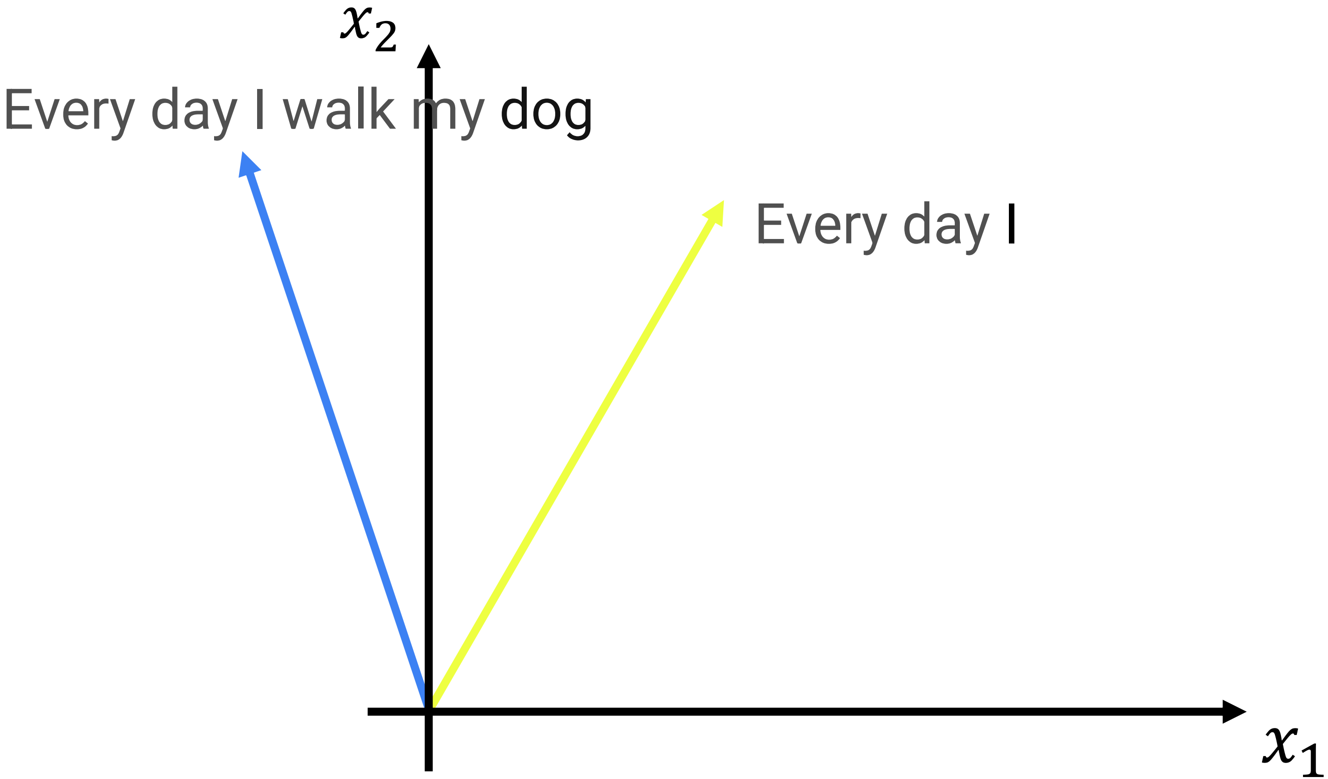
my **dog** is a good boy

dog is a man’s best friend

Rotary Positional Embeddings, 2021



Rotary Positional Embeddings, 2021



Every day I walk my dog

Every day I

Rotary Positional Embeddings, 2021

Rotary Positional Embeddings

$$f_q(\boldsymbol{x}_m, m) = R_{\Theta}^{m} \boldsymbol{W}_q \boldsymbol{x}_m$$

Query vector at position m

$$f_k(\boldsymbol{x}_n, n) = R_{\Theta}^{n} \boldsymbol{W}_k \boldsymbol{x}_n$$

key vector at position n

$$\alpha_{m,n} = < f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n) >$$

Rotary Positional Embeddings

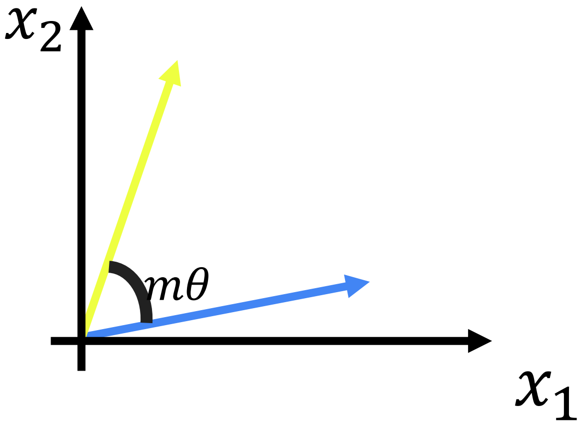
$$f_q(\boldsymbol{x}_m, m) = R_\Theta^m \boldsymbol{W}_q \boldsymbol{x}_m$$

Query vector at position m

$$R_\Theta^m = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$

$$f_k(\boldsymbol{x}_n, n) = R_\Theta^n \boldsymbol{W}_k \boldsymbol{x}_n$$

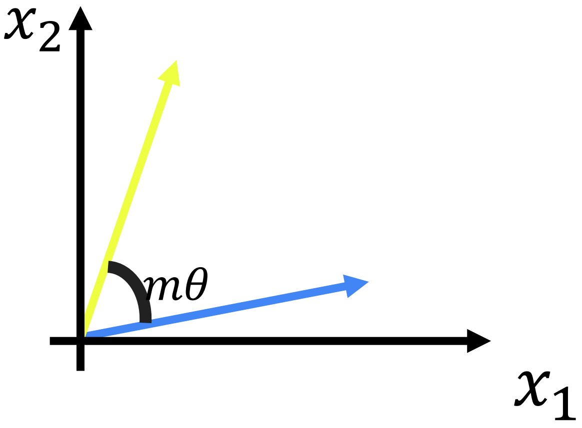
key vector at position n

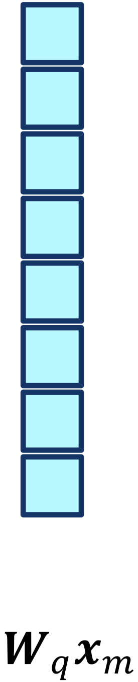


$$\alpha_{m,n} = < f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n) >$$

Rotary Positional Embeddings

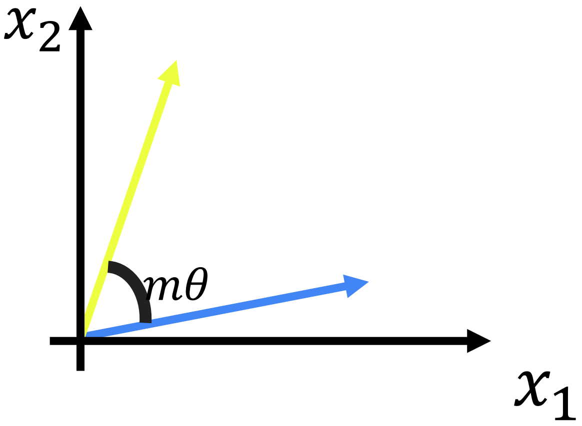
$$R_{\Theta}^{m} = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$

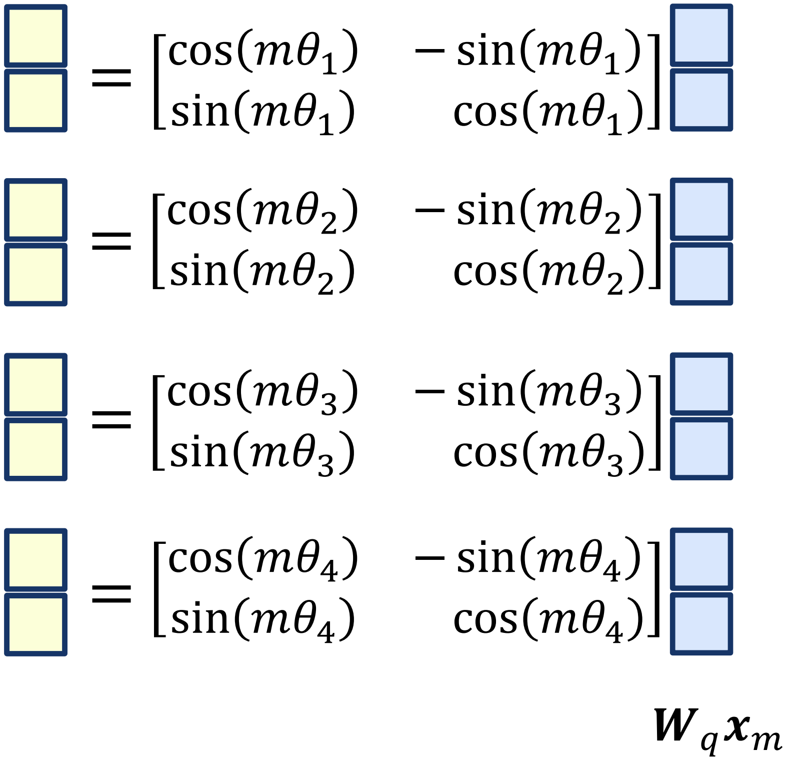




Rotary Positional Embeddings

$$R_{\Theta}^{m} = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$





$$= \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) \\ \sin(m\theta_1) & \cos(m\theta_1) \end{bmatrix}$$

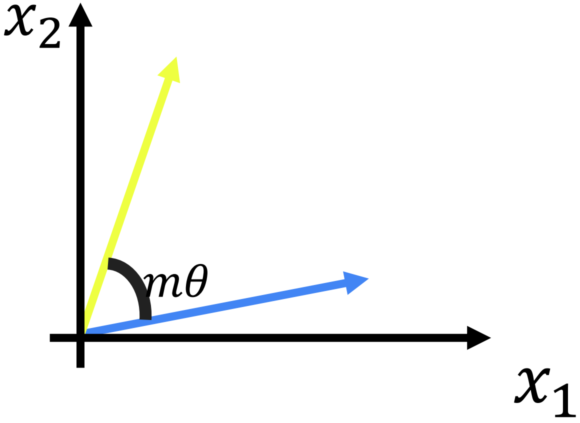
$$= \begin{bmatrix} \cos(m\theta_2) & -\sin(m\theta_2) \\ \sin(m\theta_2) & \cos(m\theta_2) \end{bmatrix}$$

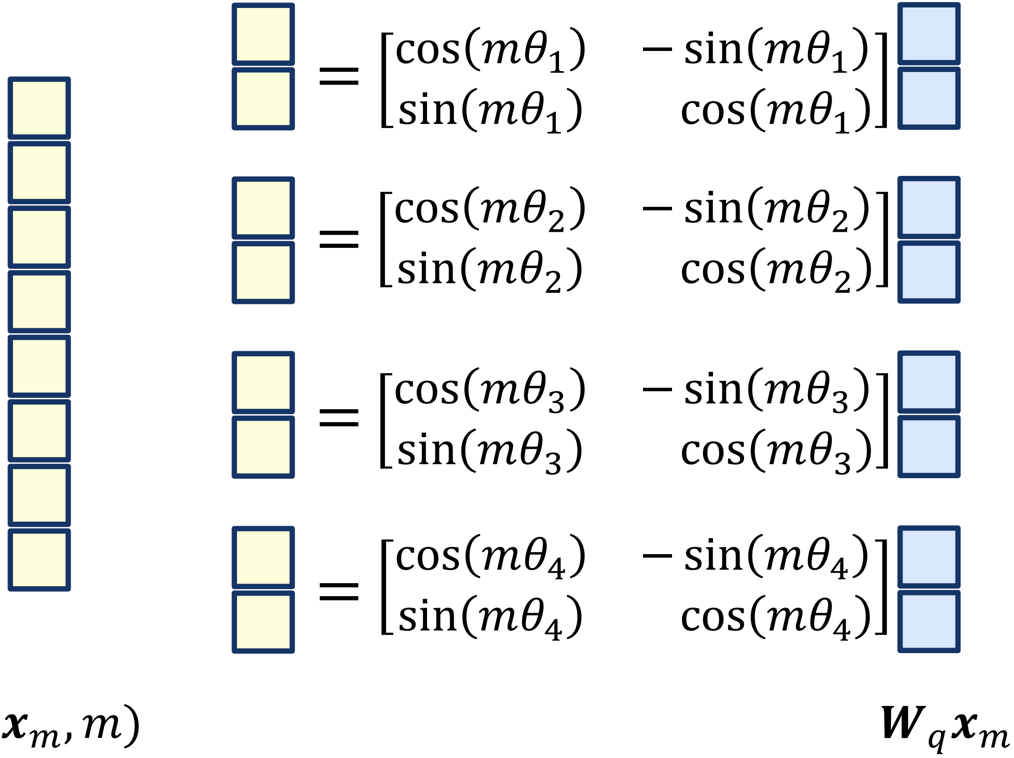
$$= \begin{bmatrix} \cos(m\theta_3) & -\sin(m\theta_3) \\ \sin(m\theta_3) & \cos(m\theta_3) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(m\theta_4) & -\sin(m\theta_4) \\ \sin(m\theta_4) & \cos(m\theta_4) \end{bmatrix}$$

Rotary Positional Embeddings

$$R_{\Theta}^{m} = \begin{bmatrix}\cos(m\theta) & -\sin(m\theta)\\ \sin(m\theta) & \cos(m\theta)\end{bmatrix}$$





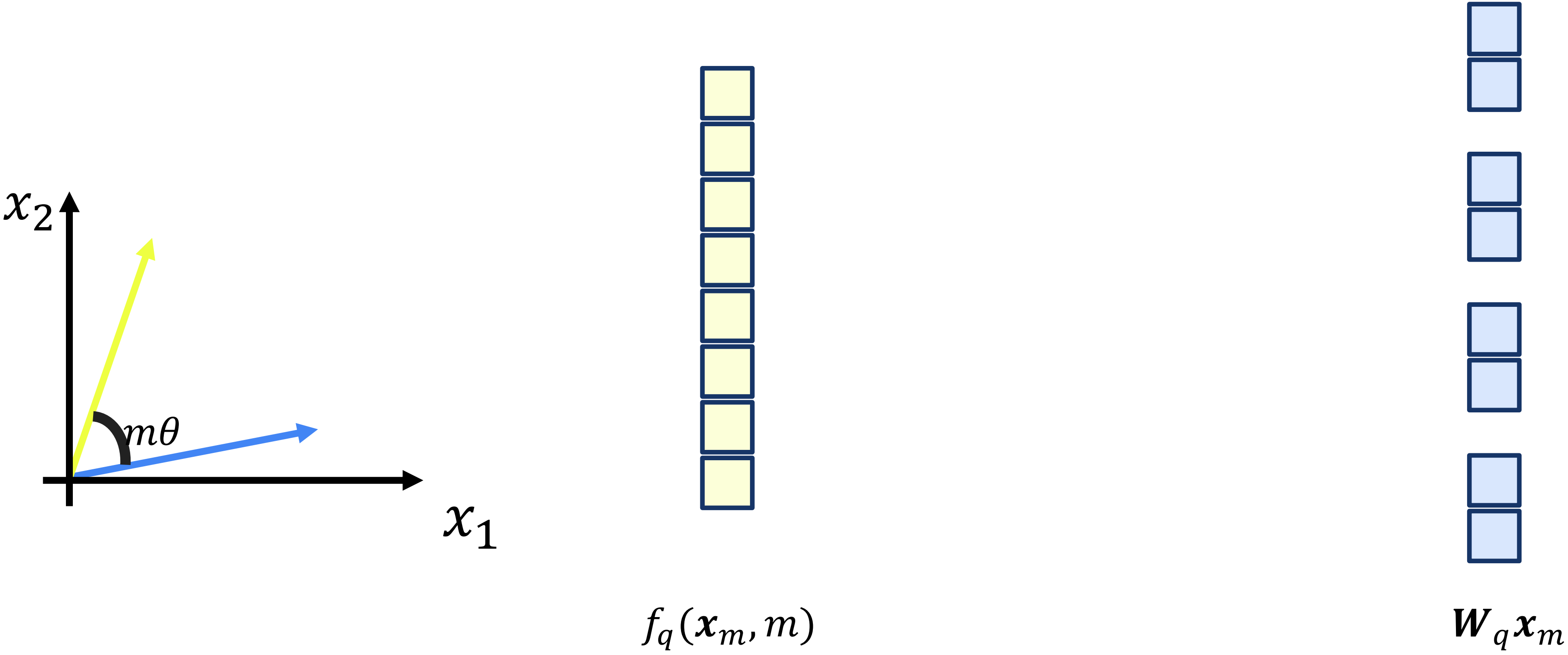
$$= \begin{bmatrix}\cos(m\theta_1) & -\sin(m\theta_1)\\ \sin(m\theta_1) & \cos(m\theta_1)\end{bmatrix}$$

$$= \begin{bmatrix}\cos(m\theta_2) & -\sin(m\theta_2)\\ \sin(m\theta_2) & \cos(m\theta_2)\end{bmatrix}$$

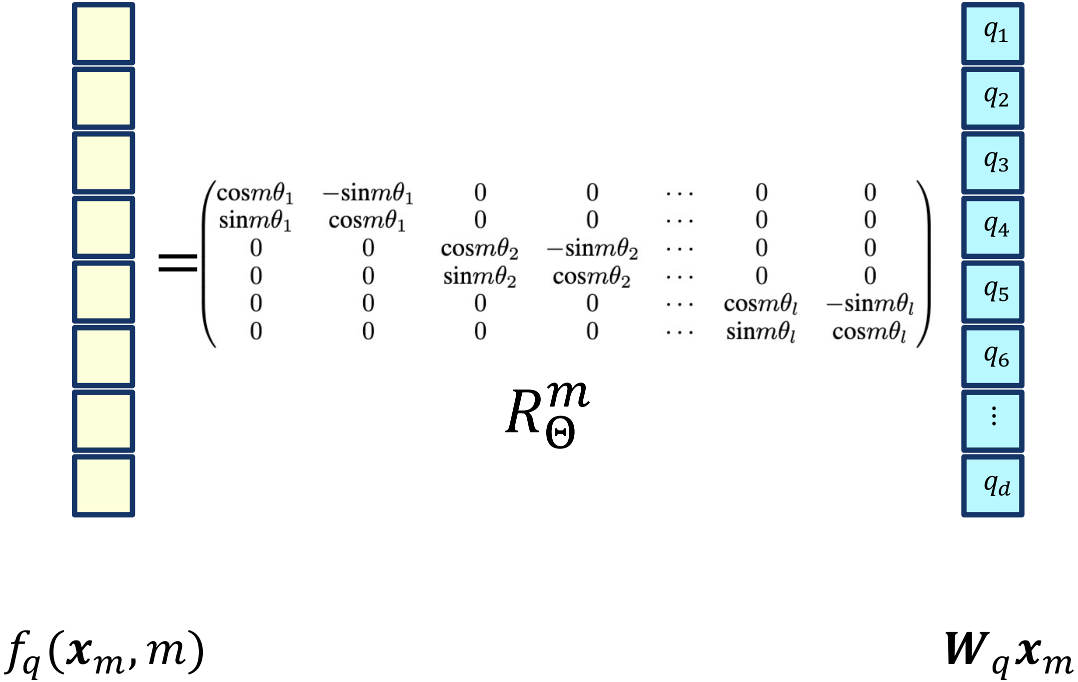
$$= \begin{bmatrix}\cos(m\theta_3) & -\sin(m\theta_3)\\ \sin(m\theta_3) & \cos(m\theta_3)\end{bmatrix}$$

$$= \begin{bmatrix}\cos(m\theta_4) & -\sin(m\theta_4)\\ \sin(m\theta_4) & \cos(m\theta_4)\end{bmatrix}$$

Rotary Positional Embeddings



Rotary Positional Embeddings



$$
= \begin{pmatrix}
\cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
\sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\
0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & \cos m\theta_l & -\sin m\theta_l \\
0 & 0 & 0 & 0 & \cdots & \sin m\theta_l & \cos m\theta_l
\end{pmatrix}
$$

R_{Θ}^{m}

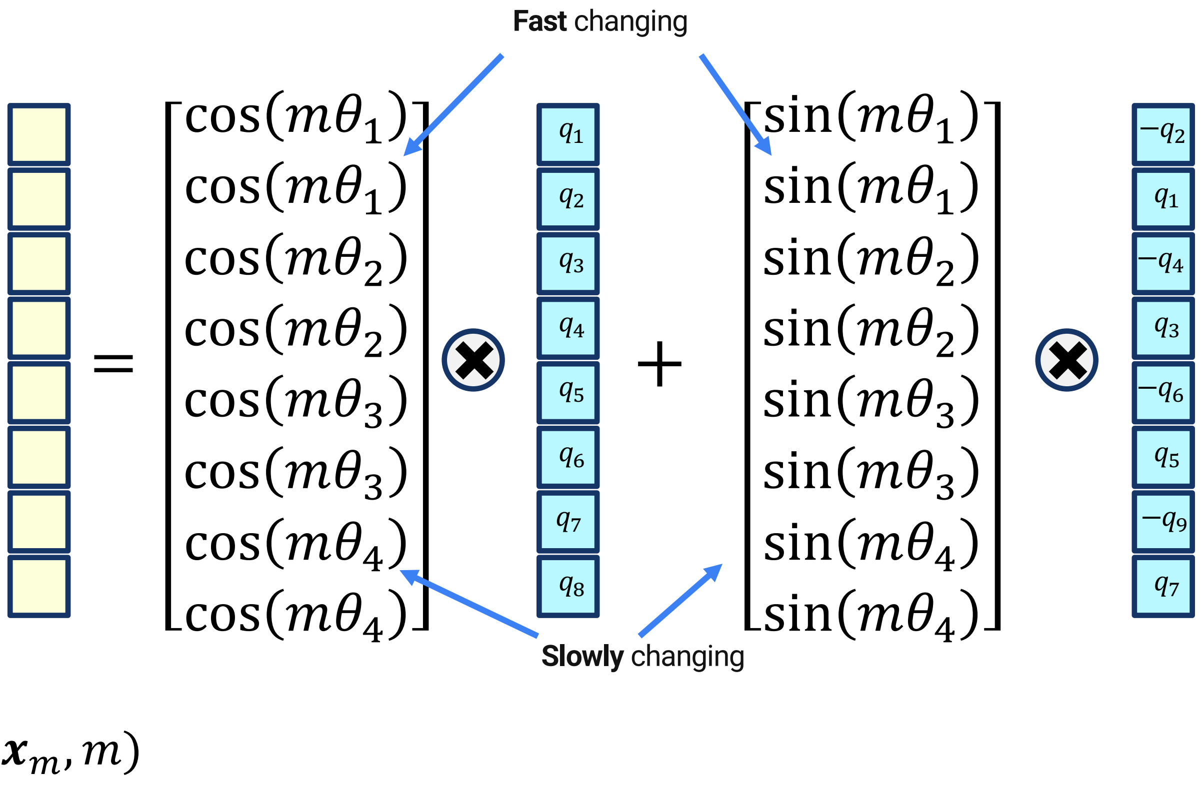
$f_q(\boldsymbol{x}_m, m)$

$\boldsymbol{W}_q \boldsymbol{x}_m$

Rotary Positional Embeddings

$$\theta_i = N^{-2i/d}$$

$$N = 10{,}000$$

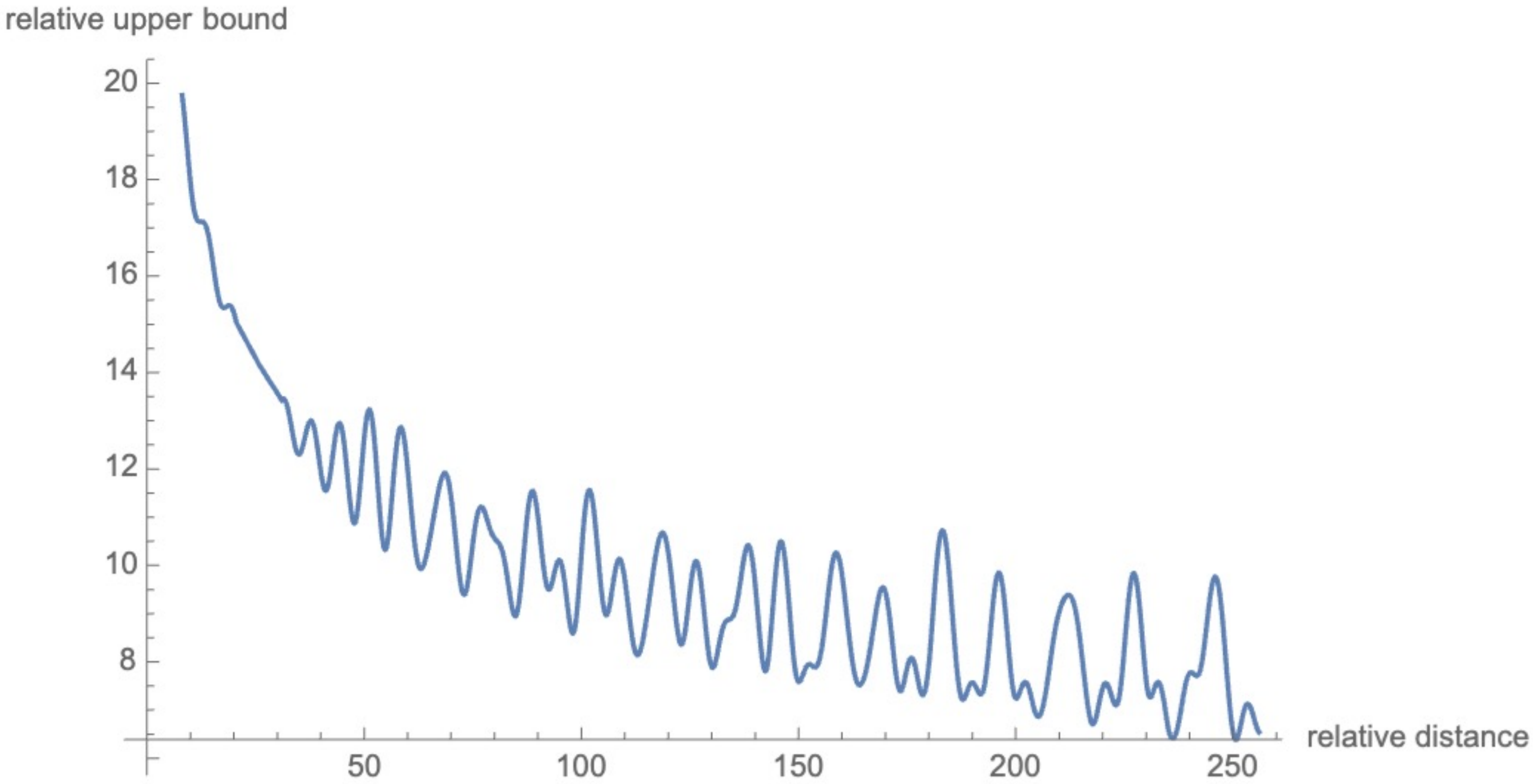


Fast changing

$$= \begin{bmatrix} \cos(m\theta_1) \\ \cos(m\theta_1) \\ \cos(m\theta_2) \\ \cos(m\theta_2) \\ \cos(m\theta_3) \\ \cos(m\theta_3) \\ \cos(m\theta_4) \\ \cos(m\theta_4) \end{bmatrix} \otimes \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \end{bmatrix} + \begin{bmatrix} \sin(m\theta_1) \\ \sin(m\theta_1) \\ \sin(m\theta_2) \\ \sin(m\theta_2) \\ \sin(m\theta_3) \\ \sin(m\theta_3) \\ \sin(m\theta_4) \\ \sin(m\theta_4) \end{bmatrix} \otimes \begin{bmatrix} -q_2 \\ q_1 \\ -q_4 \\ q_3 \\ -q_6 \\ q_5 \\ -q_9 \\ q_7 \end{bmatrix}$$

Slowly changing

$f_q(\boldsymbol{x}_m, m)$



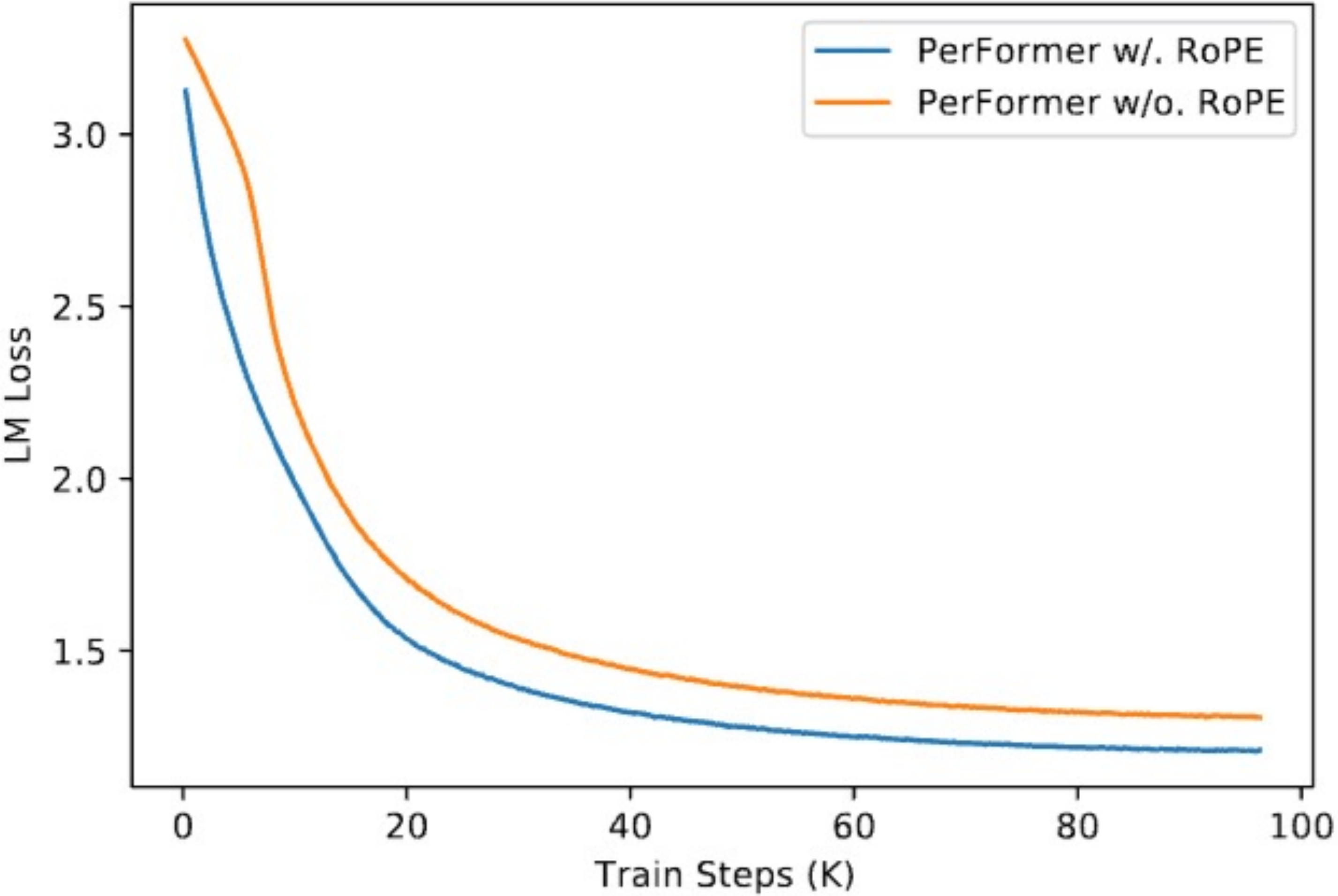
I **walk** my **dog** every day, enjoying the fresh air and the peaceful surroundings. As we stroll through the neighborhood, my dog excitedly sniffs every tree and patch of grass, wagging its tail with delight. The routine has become a relaxing part of my day, offering a moment to clear my mind while my dog gets some exercise. Whether it's **sunny** or overcast, these walks are a cherished time for both of us to unwind and explore.

Absolute Positional Embeddings

$$\alpha_{m,n} \;=\; \left[W_q\left(x_m + PE(m)\right)\right]^\top W_k\left(x_n + PE(n)\right)$$

Rotary Positional Embeddings

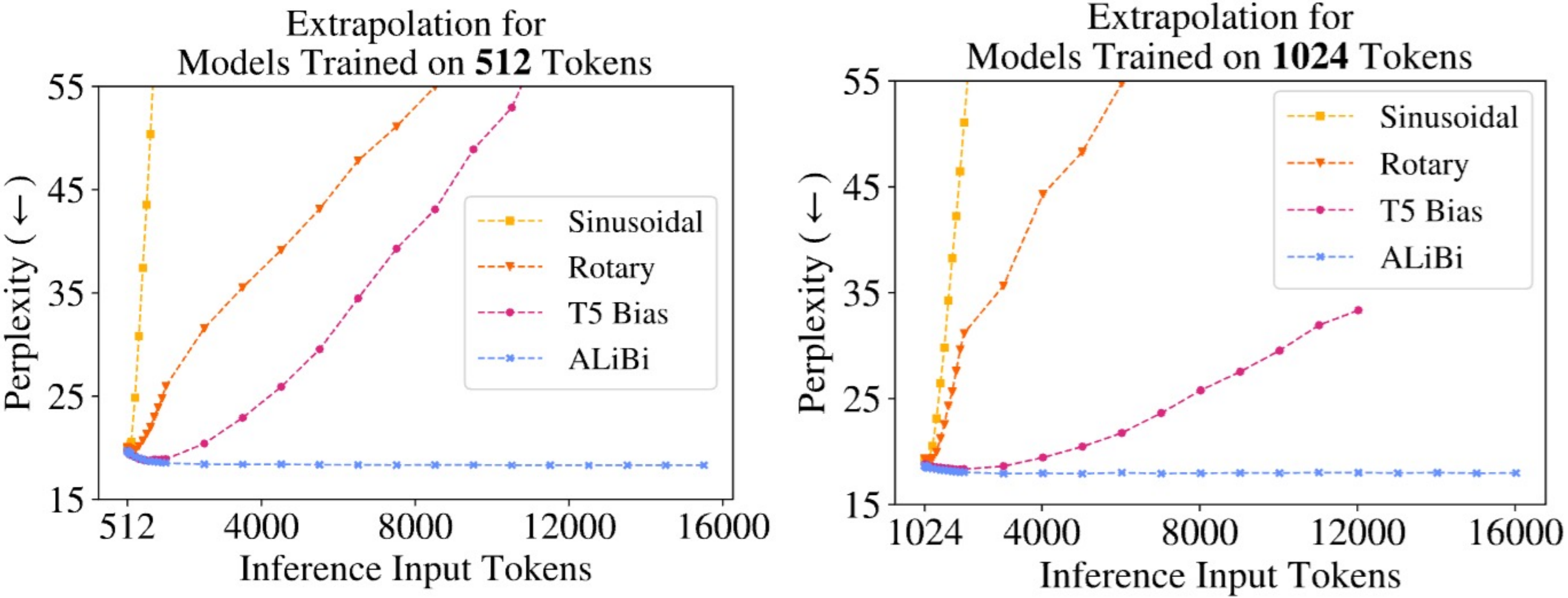
$$\begin{aligned}\alpha_{m,n} &= \left[R_\Theta^m W_q x_m\right]^\top R_\Theta^n W_k x_n \\ &= x_m^\top W_q^\top \left(R_\Theta^{m\,\top} R_\Theta^n\right) W_k x_n\end{aligned}$$

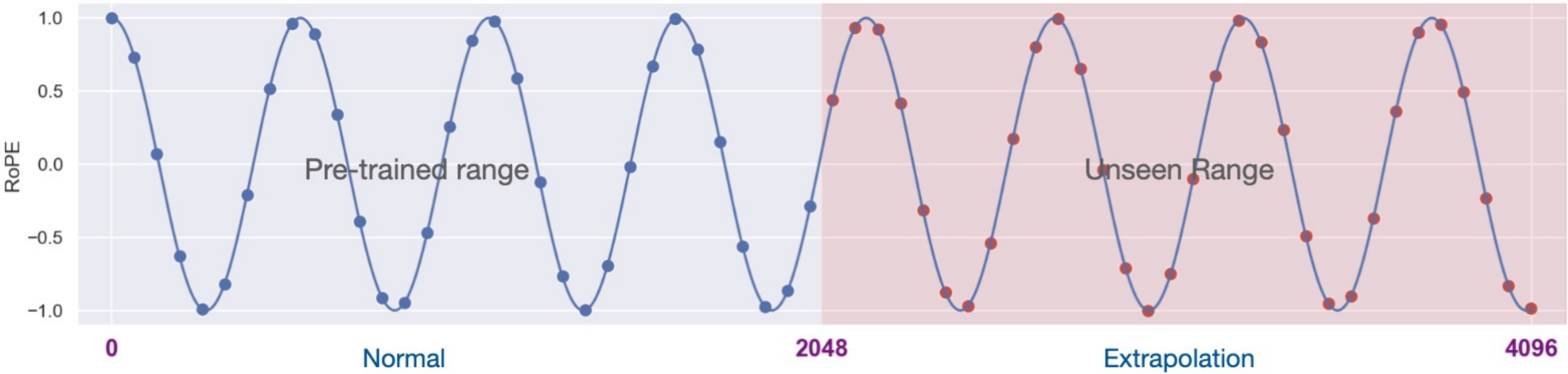

PerFormer w/. RoPE
PerFormer w/o. RoPE
LM Loss
3.0
2.5
2.0
1.5
0
20
40
60
80
100
Train Steps (K)

PaLM2

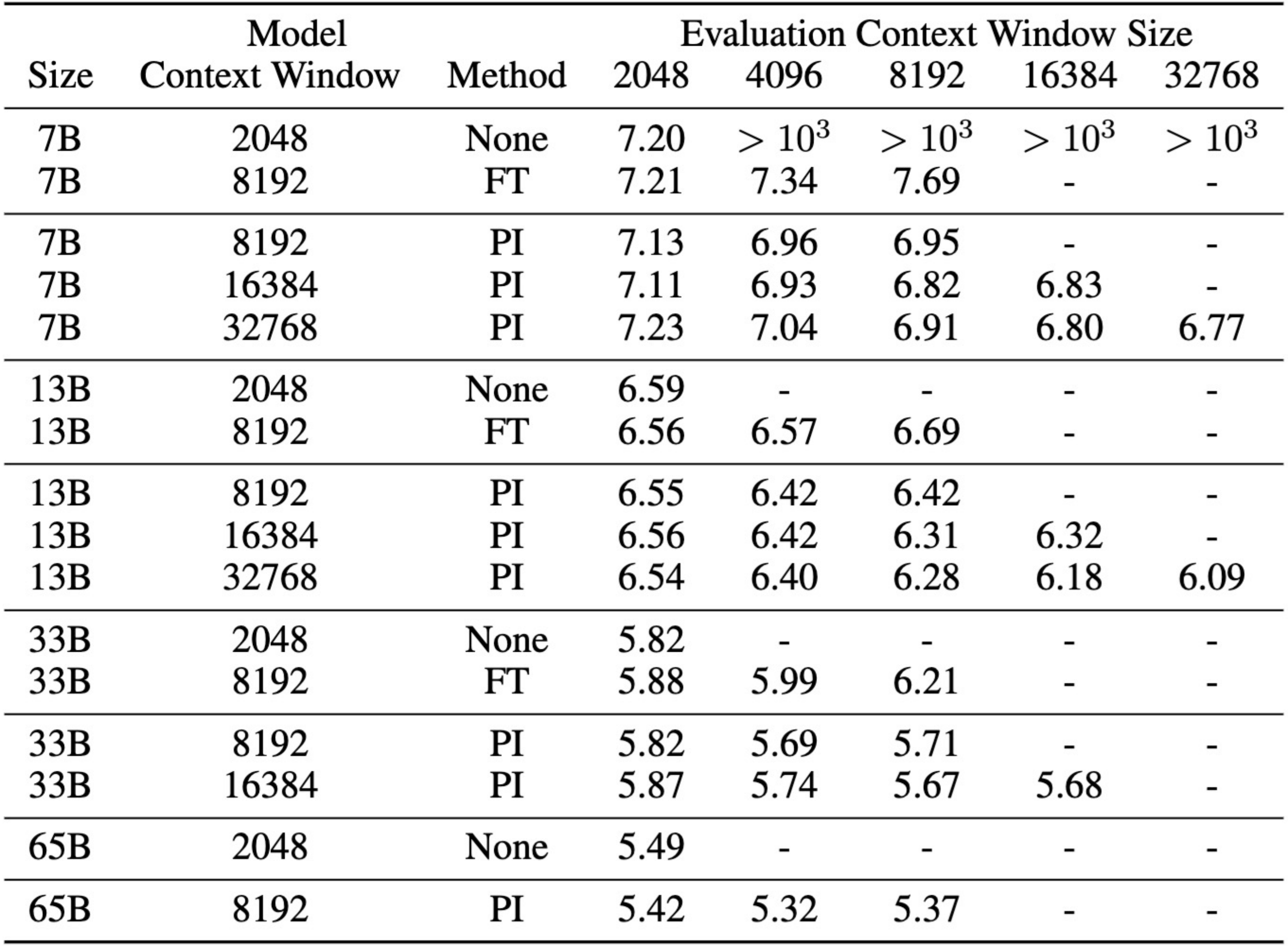
Google's Next-Generation
Large Language Model

Poor length generalization!





Position Interpolation, June 2023



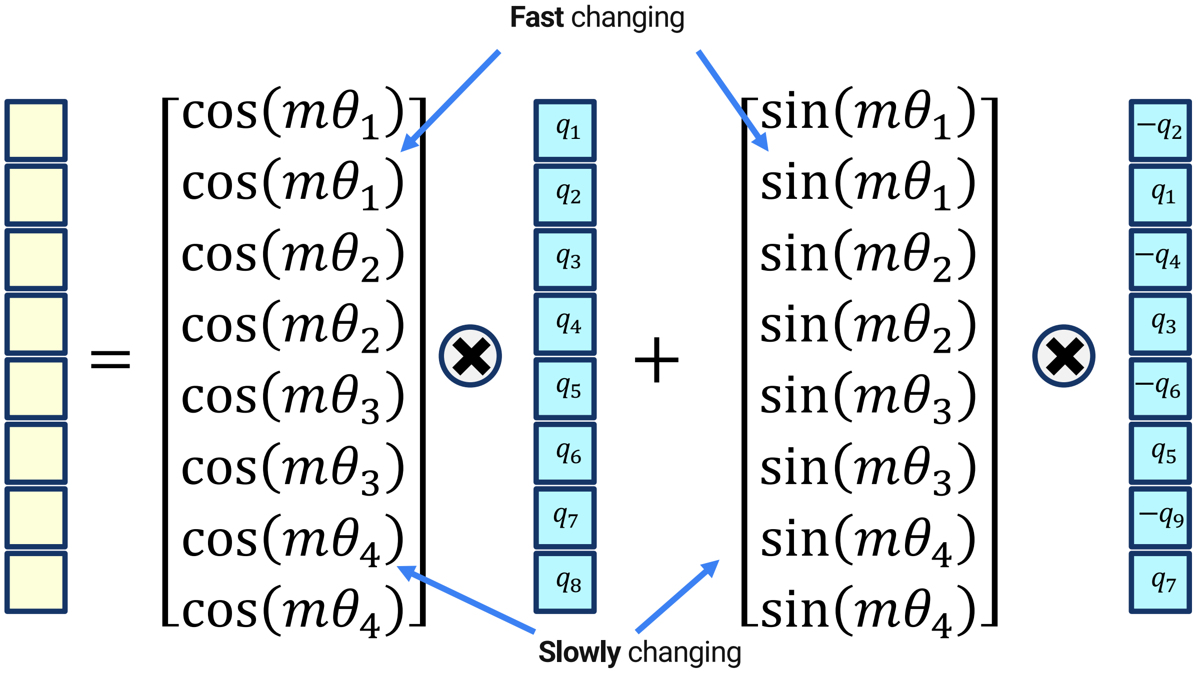
	Model		Evaluation Context Window Size				
Size	Context Window	Method	2048	4096	8192	16384	32768
7B	2048	None	7.20	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$
7B	8192	FT	7.21	7.34	7.69	-	-
7B	8192	PI	7.13	6.96	6.95	-	-
7B	16384	PI	7.11	6.93	6.82	6.83	-
7B	32768	PI	7.23	7.04	6.91	6.80	6.77
13B	2048	None	6.59	-	-	-	-
13B	8192	FT	6.56	6.57	6.69	-	-
13B	8192	PI	6.55	6.42	6.42	-	-
13B	16384	PI	6.56	6.42	6.31	6.32	-
13B	32768	PI	6.54	6.40	6.28	6.18	6.09
33B	2048	None	5.82	-	-	-	-
33B	8192	FT	5.88	5.99	6.21	-	-
33B	8192	PI	5.82	5.69	5.71	-	-
33B	16384	PI	5.87	5.74	5.67	5.68	-
65B	2048	None	5.49	-	-	-	-
65B	8192	PI	5.42	5.32	5.37	-	-

Position Interpolation, June 2023

Rotary Positional Embeddings

$$\theta_i = N^{-2i/d}$$

$$N = 10{,}000$$



Fast changing

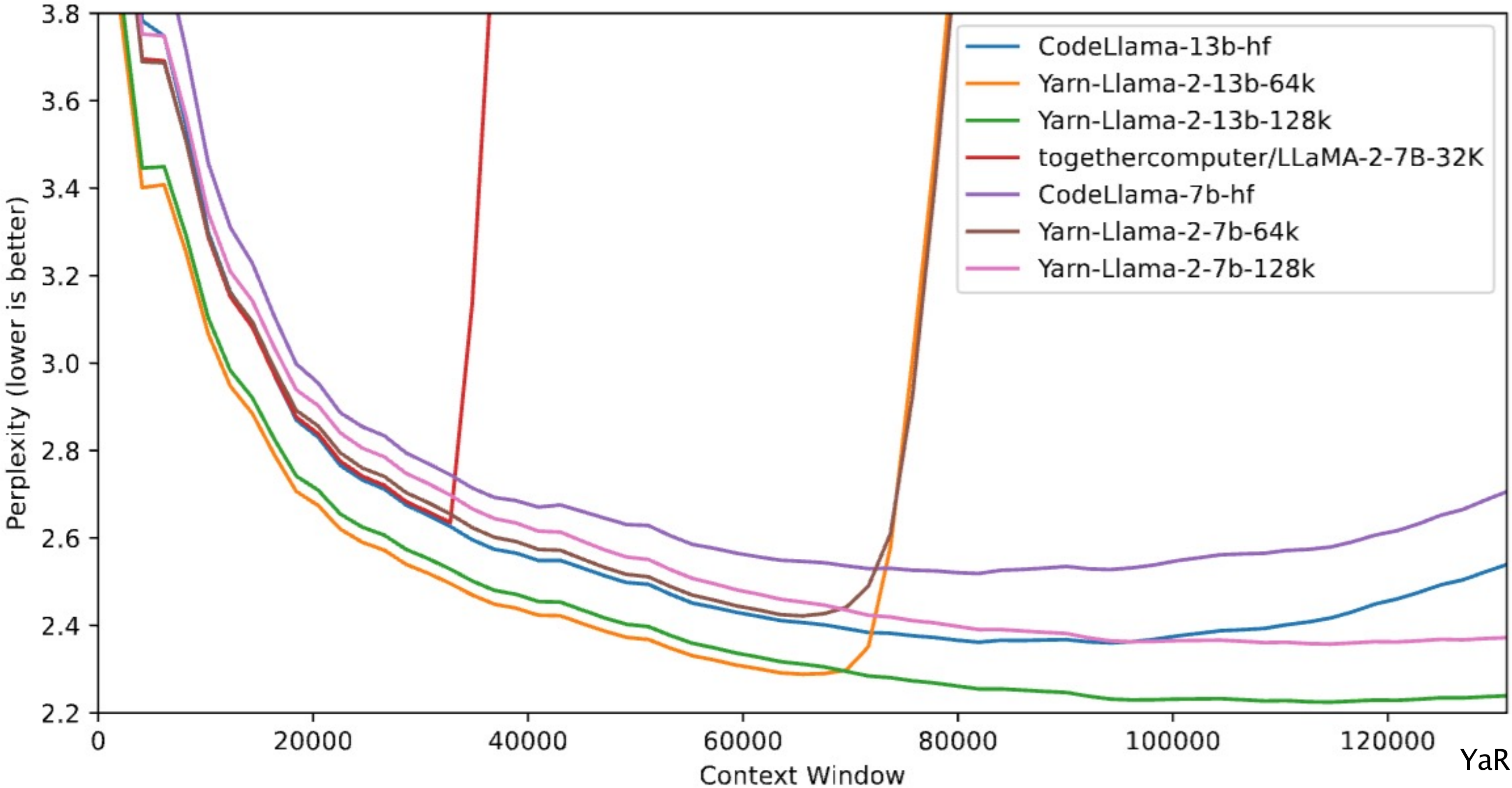
Slowly changing

$$\begin{bmatrix} \square \\ \square \\ \square \\ \square \\ \square \\ \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} \cos(m\theta_1) \\ \cos(m\theta_1) \\ \cos(m\theta_2) \\ \cos(m\theta_2) \\ \cos(m\theta_3) \\ \cos(m\theta_3) \\ \cos(m\theta_4) \\ \cos(m\theta_4) \end{bmatrix} \otimes \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \end{bmatrix} + \begin{bmatrix} \sin(m\theta_1) \\ \sin(m\theta_1) \\ \sin(m\theta_2) \\ \sin(m\theta_2) \\ \sin(m\theta_3) \\ \sin(m\theta_3) \\ \sin(m\theta_4) \\ \sin(m\theta_4) \end{bmatrix} \otimes \begin{bmatrix} -q_2 \\ q_1 \\ -q_4 \\ q_3 \\ -q_6 \\ q_5 \\ -q_9 \\ q_7 \end{bmatrix}$$

$f_q(\boldsymbol{x}_m, m)$

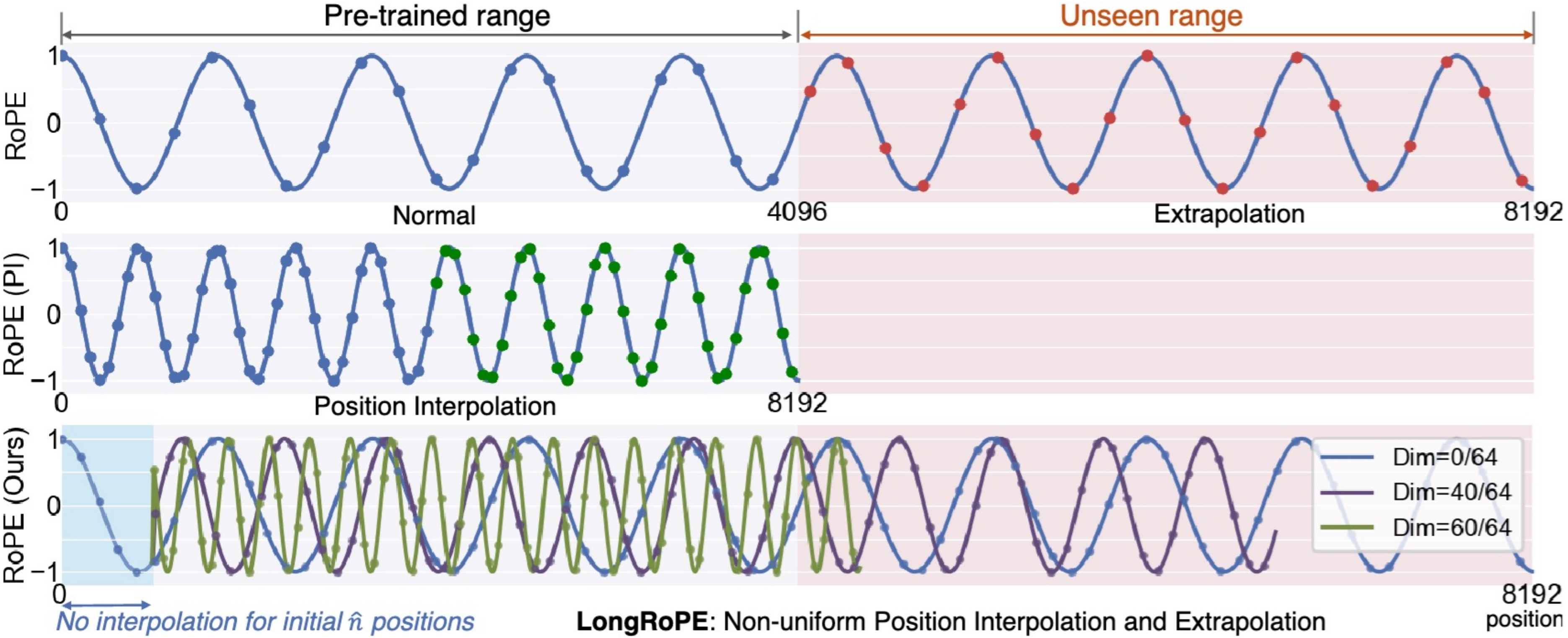
YaRN

Yet another RoPE extensioN method

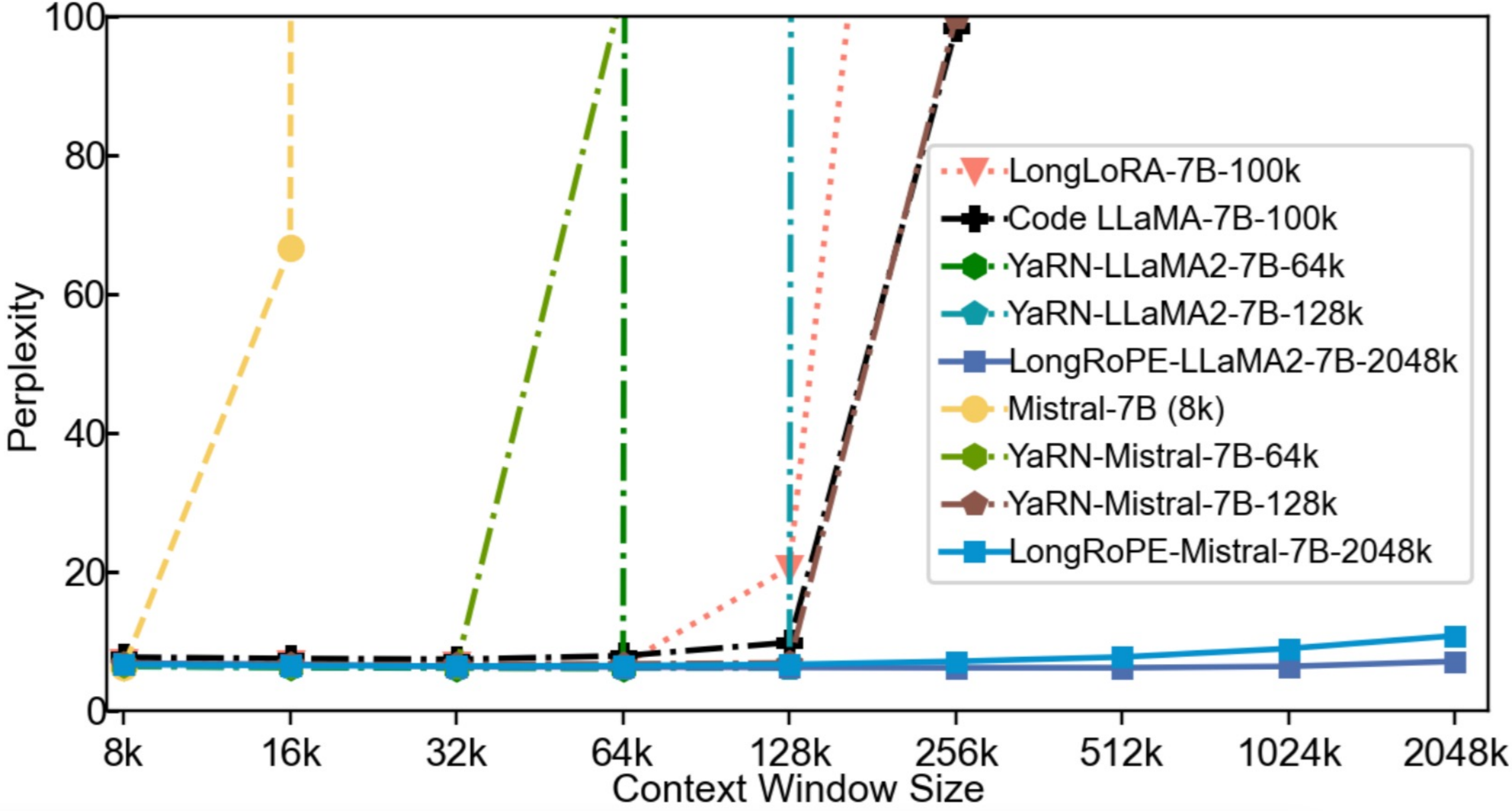


YaRN, Nov 2023

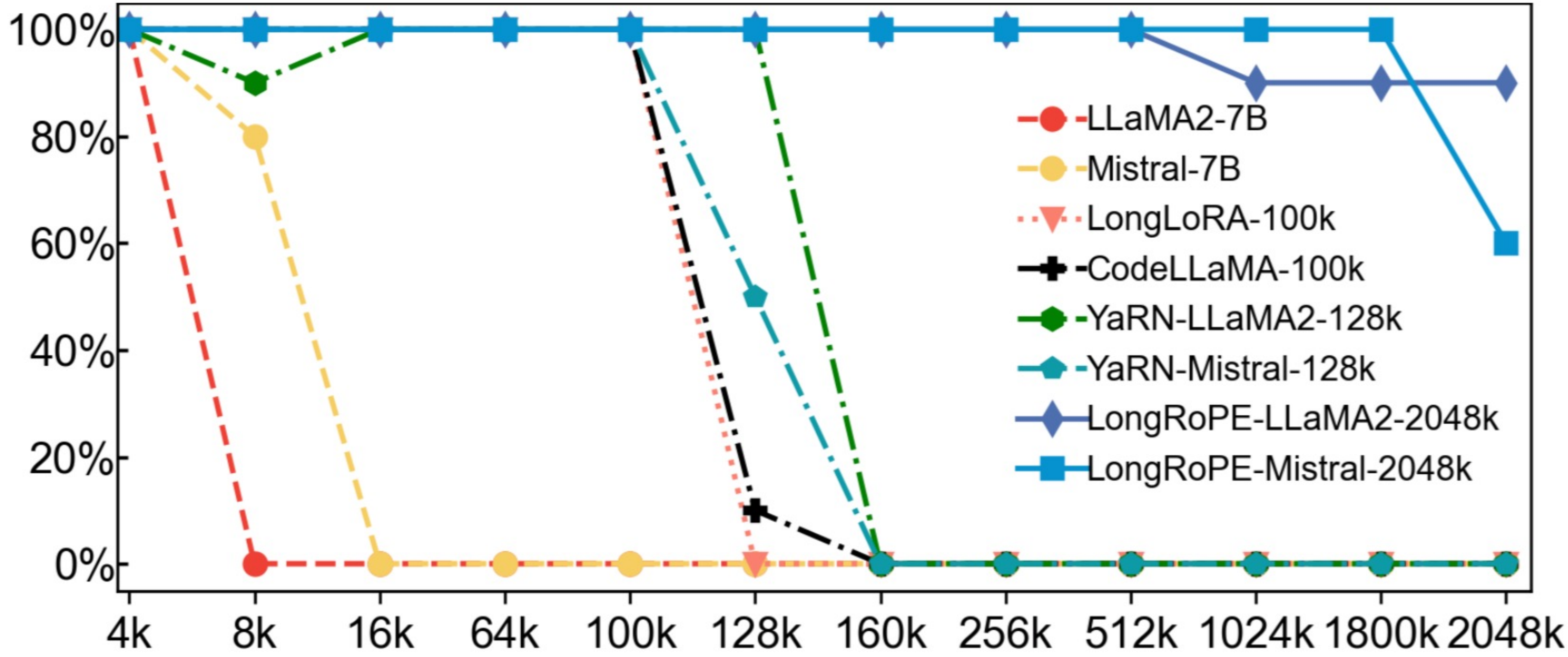
LongRoPE



LongRoPE, Feb 2024



LongRoPE, Feb 2024



Passkey retrieval accuracy

LongRoPE, Feb 2024

Limitations (PI, YaRN, LongRoPE)

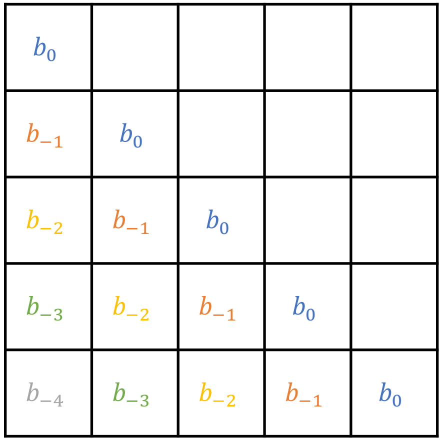
- Require known target context length.
- Often require finetuning
- Works only with RoPE (Llama-2, Llama-3, PaLM)

Another Class of Methods for Long Context Extension:

Just “Change” the Attention Matrix by adding Relative Positional Bias (the *B matrix*)

$$\mathrm{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}} + B\right)$$

Relative positional encoding: e.g. T5 bias, and many more…



$$B \quad \begin{array}{|c|c|c|c|c|} \hline b_0 & & & & \\ \hline b_{-1} & b_0 & & & \\ \hline b_{-2} & b_{-1} & b_0 & & \\ \hline b_{-3} & b_{-2} & b_{-1} & b_0 & \\ \hline b_{-4} & b_{-3} & b_{-2} & b_{-1} & b_0 \\ \hline \end{array}$$

T5 bias

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + B\right)$$

$$B(m,n) = -r_{\min(m-n,K)}$$

b_0				
b_{-1}	b_0			
b_{-2}	b_{-1}	b_0		
b_{-3}	b_{-2}	b_{-1}	b_0	
b_{-4}	b_{-3}	b_{-2}	b_{-1}	b_0

T5 bias

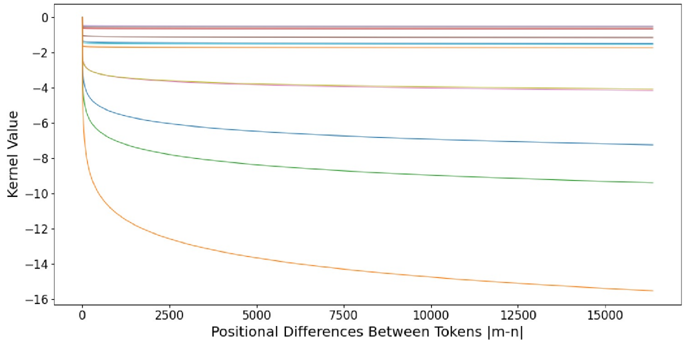
$$B(m,n) = -r|m-n|$$

$$-\frac{1}{2}$$

0				
1	0			
2	1	0		
3	2	1	0	
4	3	2	1	0

Alibi

$$B(m,n) = -\mathrm{r}_1\log\left(1 + r_2|m-n|\right)$$



Kerple

$$\mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + B\right)$$

$$B(m,n) = f_\theta\left(\frac{m-n}{m}\right)$$

Amplifying the differences among local positions

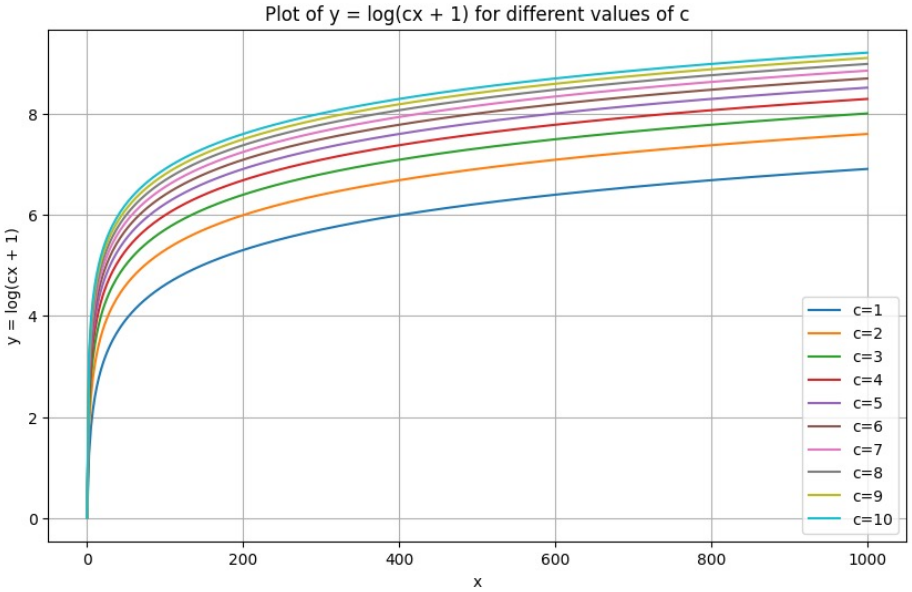
$$B(m,n) = f_\theta\left(\frac{\phi(m-n)}{\phi(m)}\right)$$

Better short-sequence modeling

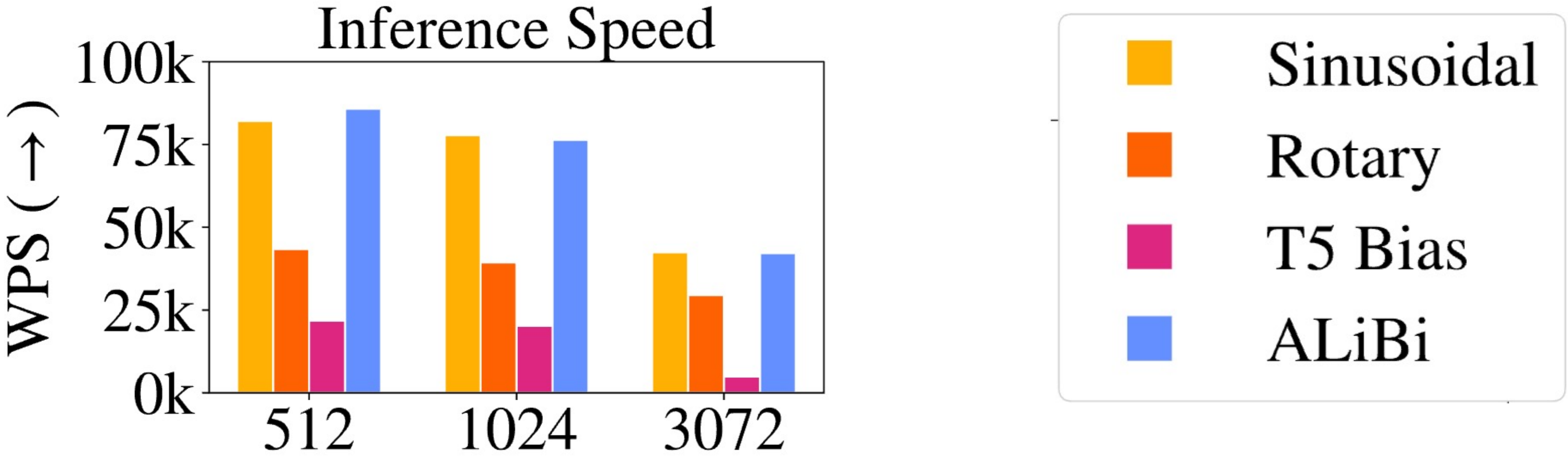
$$B(m,n) = f_\theta\left(\frac{\phi(m-n)}{\phi(\max(L,m))}\right)$$

FIRE

$$\phi(x) = \log(cx+1)$$



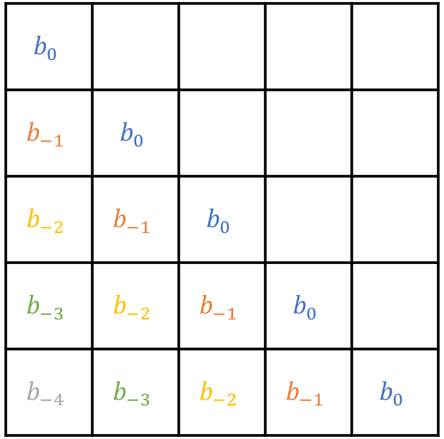
Functional Interpolation for Relative position Embeddings, 2024



$$\mathrm{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}} + B\right)$$

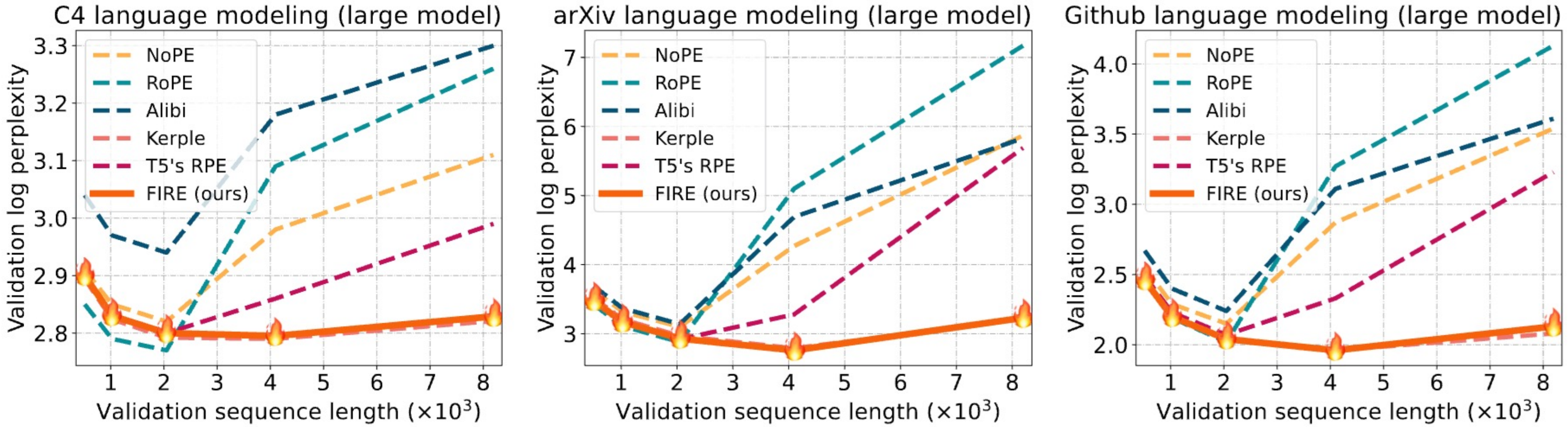
Relative positional encoding: e.g. T5 bias, and many more…

B

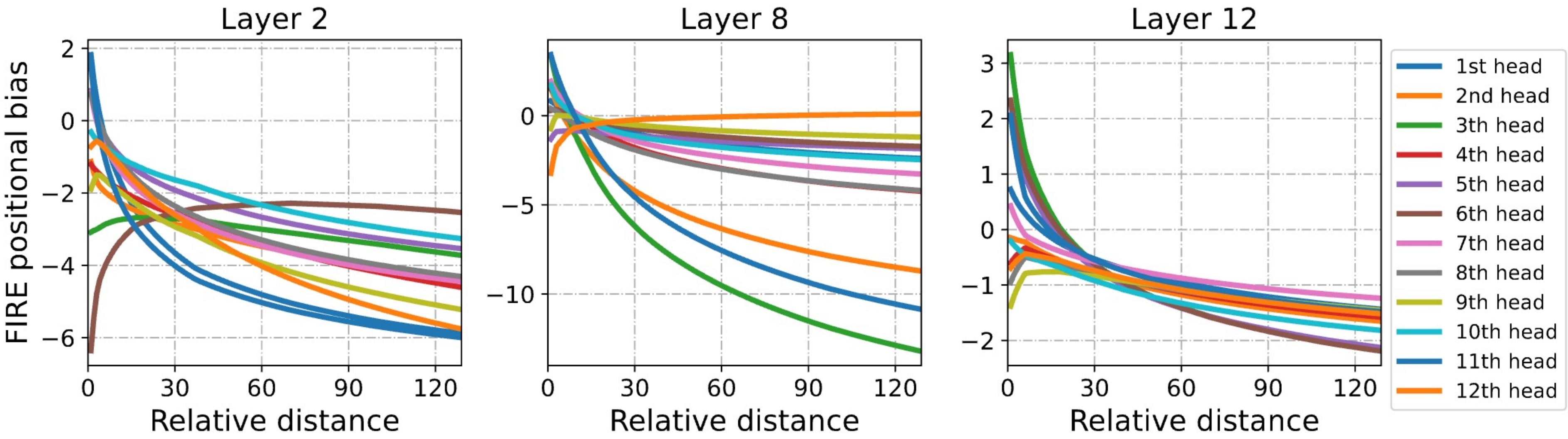


b_0				
b_{-1}	b_0			
b_{-2}	b_{-1}	b_0		
b_{-3}	b_{-2}	b_{-1}	b_0	
b_{-4}	b_{-3}	b_{-2}	b_{-1}	b_0

T5 bias



Functional Interpolation for Relative position Embeddings, 2024



Visualization of FIRE learned position biases

Functional Interpolation for Relative position Embeddings, 2024

Development Summary of Positional Encoding Schemes

- Transformer: need input position information
- **Absolute positional encoding**
    - **Just concatenate the position t**
    - **BUT, really hard to learn**
    - **THEREFORE, sinusoidal**
    - **BUT, this seems arbitrary!**
    - **THEREFORE, we can DIRECTLY learn the positional encoding through optimization, e.g. as in BERT**
    - **BUT, this requires known fixed length. Cannot work at all for L+1 position.**
    - **BUT, what we really want is relative position. Not absolute**
    - **Every day I walk my dog**
    - **I walk my dog every day**

Development Summary of Positional Encoding Schemes (cont’d)

- **Relative positional encoding**
  - **Example: T5 bias, learnable bias for query-key**
  - **BUT, SLOW, challenging to do KV cache**
  - **THEREFORE, Rotary position encoding**
  - **BUT, poor length generalization**
  - **THEREFORE,**
    - **Positional interpolation**
    - **YaRN**
    - **LongRoPE** https://www.reddit.com/r/LocalLLaMA/comments/1axhhs6/longrope_extending_llm_context_window_beyond_2/
  - **BUT, this requires finetuning and know the target sequence length**
  - **THEREFORE, revisit attention bias from T5**
    - **AliBi**
    - **Kerple**
    - **Sandwich**
  - **BUT, this makes it hard to attend to long-range dependency**
  - **So FIRE**

A Controlled Study on Long Context Extension and Generalization in LLMs

Yi Lu et al, https://arxiv.org/pdf/2409.12181

Video by: Prof. Alexander (Sasha) Rush of Cornell
https://www.youtube.com/watch?v=dc4chADushM

References for Positional Encoding

1. Attention is All You Need https://arxiv.org/pdf/1706.03762.pdf
2. RoPE (aka RoFormer) https://arxiv.org/pdf/2104.09864.pdf
3. ALiBi https://arxiv.org/pdf/2108.12409.pdf
4. Investigation on what positional embeddings learn https://aclanthology.org/2020.emnlp-main.555.pdf
5. YaRN: Efficient Context Window Extension of Large Language Models https://arxiv.org/pdf/2309.00071
6. Linear RoPE vs. NTK vs. YaRN vs. CoPE, July 2024 https://medium.com/@zaiinn440/linear-rope-vs-ntk-vs-yarn-vs-cope-d33587ddfd35
7. FIRE: Functional Interpolation for Relative Positions improve Long Context Transformers https://arxiv.org/pdf/2310.04418
8. Round and Round We Go! What Makes Rotary Positional Encodings Useful ? Oct 2024, https://arxiv.org/pdf/2410.06205

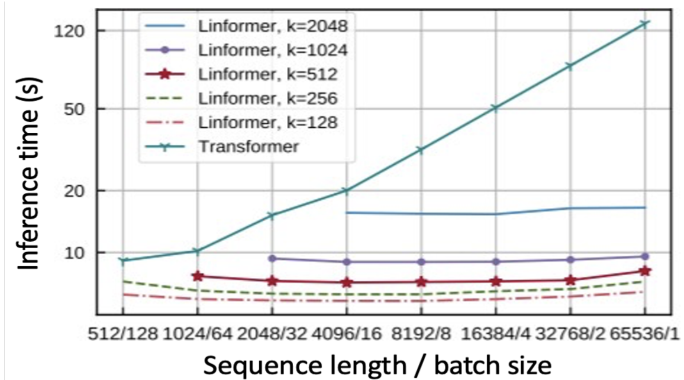
Work on Improving on Quadratic Self-Attention Cost

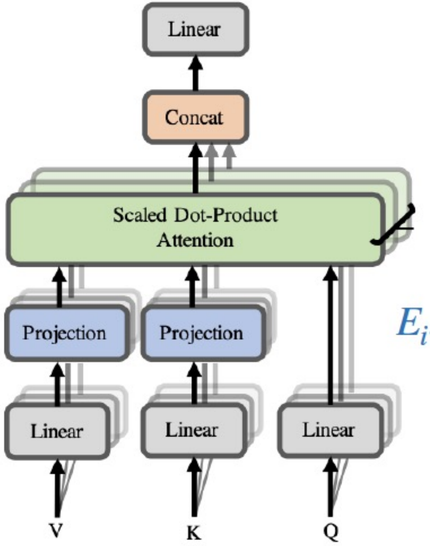
- Much recent work has gone into the question, Can we build models like Transformers without paying the $O(N^2)$ all-pairs self-attention cost?
- For example, (Wang et al., 2000): **Linformer** : Self-Attention with Linear Complexity

Key idea: The Attention Matrix can be approximated by a Low-Rank matrix

=> Map the sequence length dimension to a lower-dimensional space for values, keys.

$$\begin{aligned}\overline{\text{head}_i} &= \text{Attention}(QW_i^Q, E_iKW_i^K, F_iVW_i^V) \\ &= \underbrace{\text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}:n\times k} \cdot \underbrace{F_iVW_i^V}_{k\times d},\end{aligned}$$



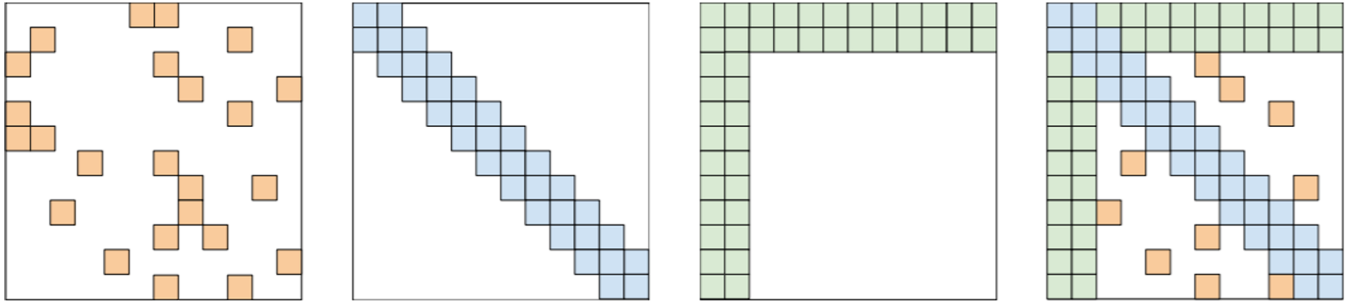


$E_i, F_i \in \mathbb{R}^{k\times n}$

Work on Improving on Quadratic Self-Attention Cost

- Much recent work has gone into the question, Can we build models like Transformers without paying the $O(N^2)$ all-pairs self-attention cost?
- For example, **BigBird** [Zaheer et al., 2021]

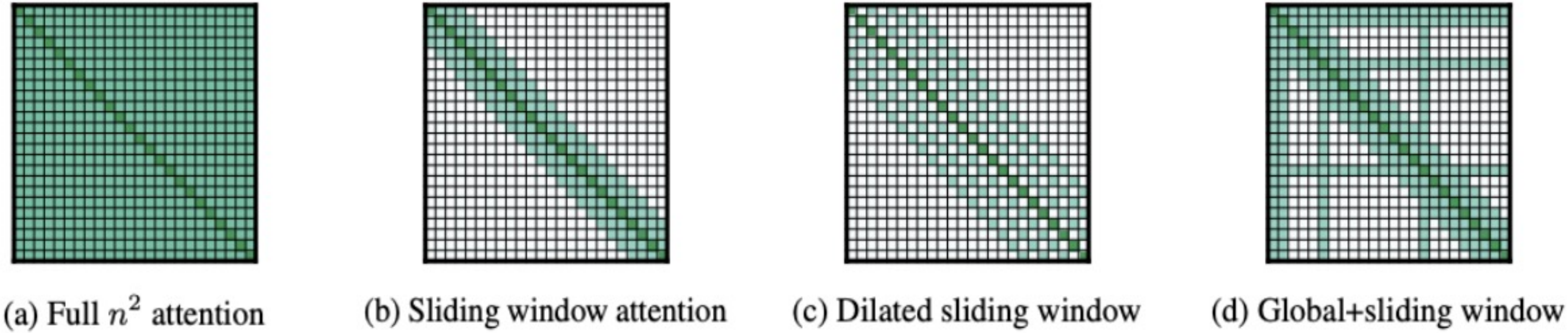
Key idea: replace all-pairs interactions with a family of other interactions, like local windows, looking at everything, and random interactions.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

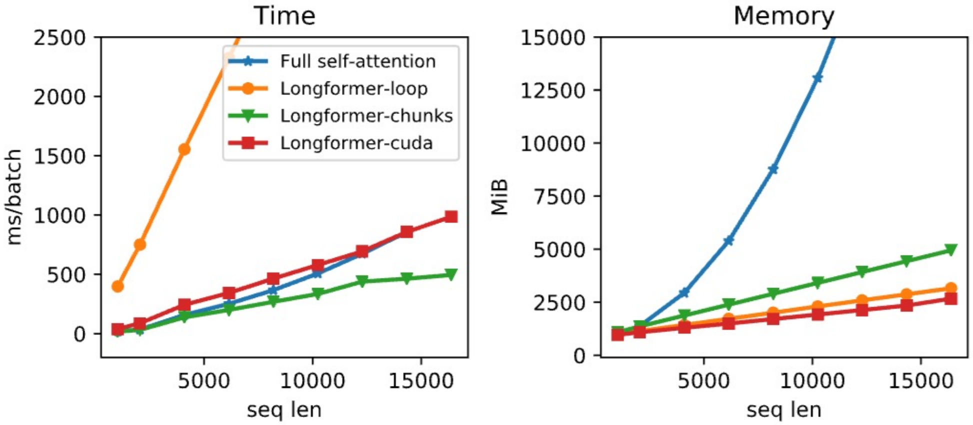
Another Sparse (but fixed) Attention Pattern: Longformer

Key idea: use sparse attention patterns!



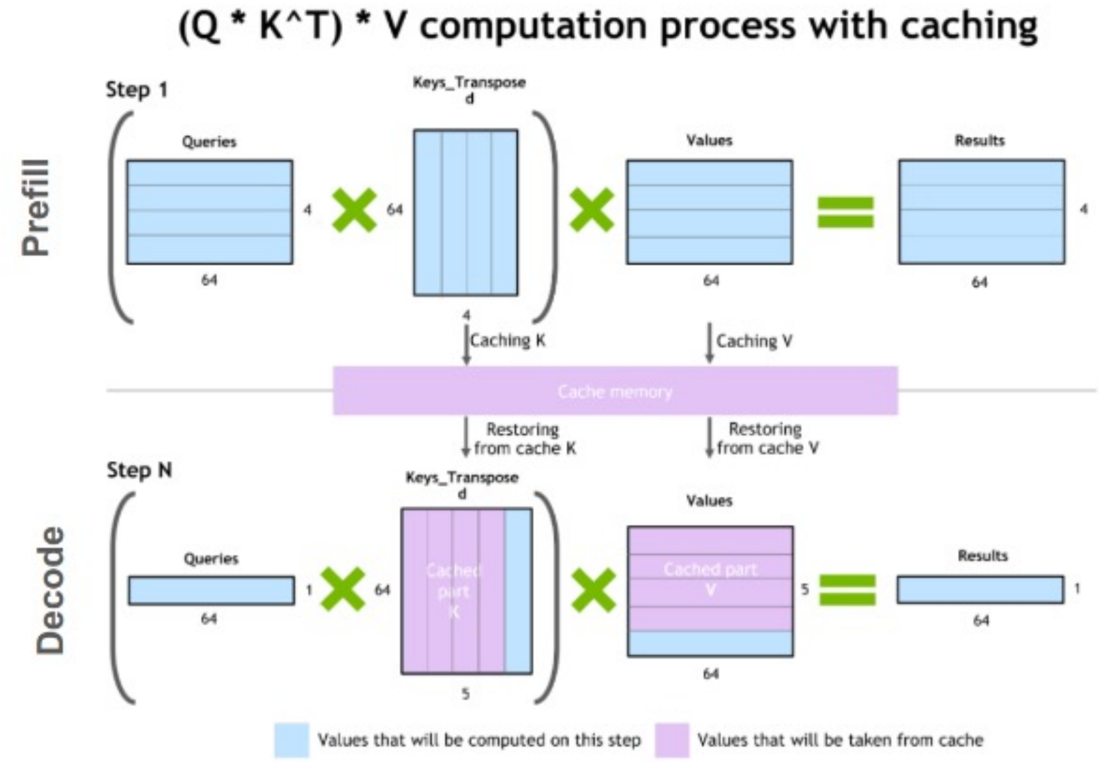
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

(Beltagy et al., 2020): Longformer: The Long-Document Transformer



More Efficient Attention via Sharing: Group-Query Attention

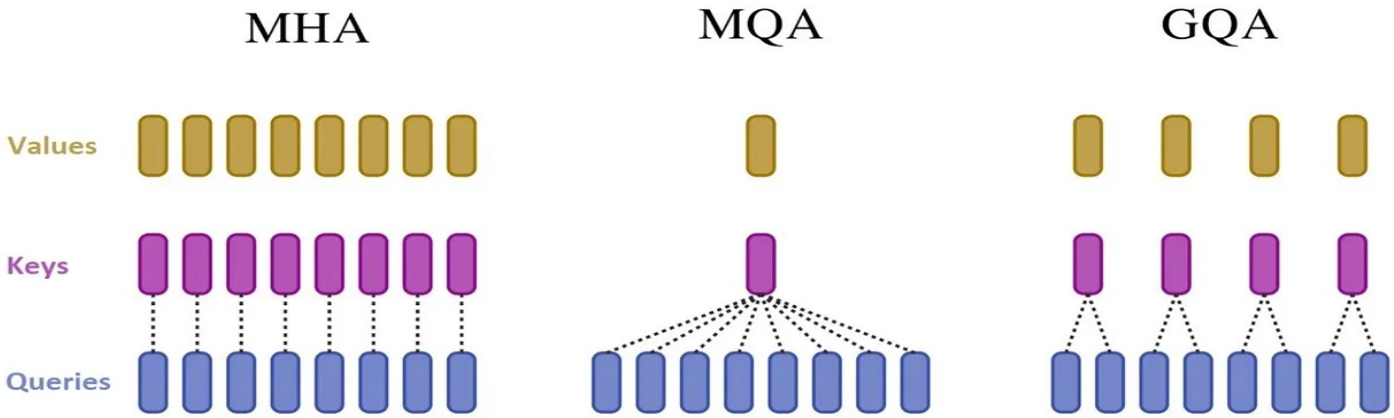
KV Cache during Inference time in Transformers



During interference time, K and V values computed from previous tokens in the windows are cached inside the GPU to avoid unnecessary re-computation. However, since memory requirement for the K and V matrices grows linearly with Context-length, KV caching creates a memory bottleneck within the GPU in practice.

More Efficient Attention via Sharing:

Multi-Head Attention | Multi-Query Attention | Grouped Query Attention



High quality

Computationally slow

Attention is All You Need (2017)

Loss in quality

Computationally fast

Fast Transformer Decoder:
One Write-Head
is All You Need (2019)

A good compromise
between quality and speed

GQA: Training Generalized
Multi-Query Transformers from
Multi-Head Checkpoints (2023)

GQA interpolates between MHA and MQA. It reduces Memory Bandwidth overhead during inference time while avoiding excessive loss in accuracy as fewer K and V matrices are loaded into the Decoder (KV-cache in GPU RAM)

Uptraining for MQA

1. Key and Value projection matrices (K, V) are mean pooled into a single projection matrix
    1. Works better than selecting one key/ value projection matrix
    2. Or randomly initializing the Projection Matrix
2. The pooled projection matrix is then trained for α = 5% of its original training steps

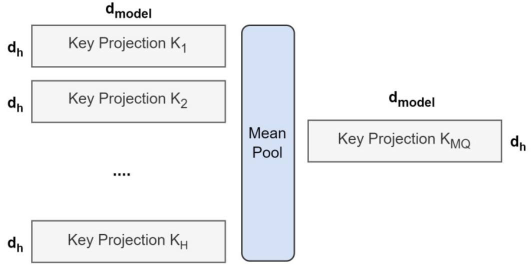


Figure 1: Overview of conversion from multi-head to multi-query attention. Key and value projection matrices from all heads are mean pooled into a single head.

Group Query Attention (GQA)

1. GQA is the natural interpolation of MQA and MHA: heads are divided into G groups, with each group sharing a single key and value head
2. When converting MHA to GQA, mean pool each group’s key/value heads into a single key/value head, and train for α steps
3. GQA, like MQA, is for reducing the reloading of K, V during decoder inference, and thus is not applied to encoder self-attention layers

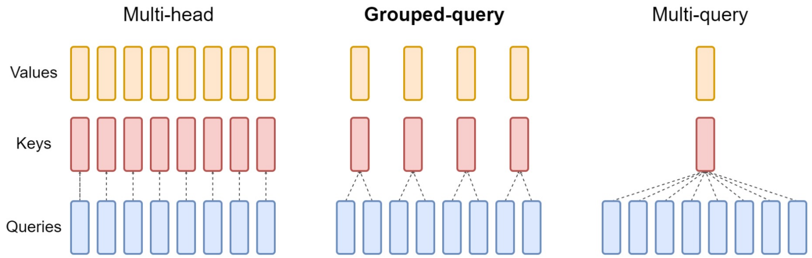


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Group Query Attention Results

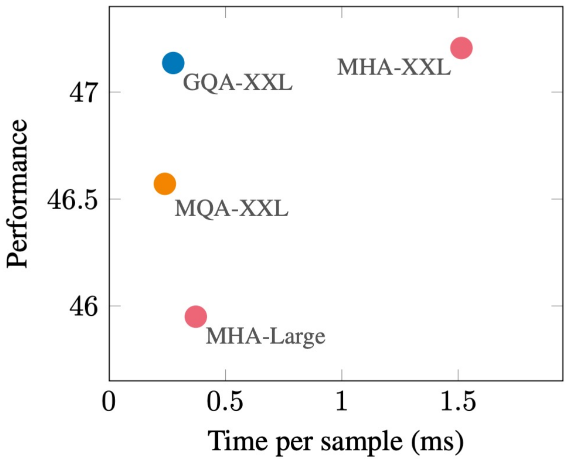


Figure 3: **Uptrained MQA yields a favorable tradeoff compared to MHA with higher quality and faster speed than MHA-Large, and GQA achieves even better performance with similar speed gains and comparable quality to MHA-XXL.** Average performance on all tasks as a function of average inference time per sample for T5-Large and T5-XXL with multi-head attention, and 5% uptrained T5-XXL with MQA and GQA-8 attention.

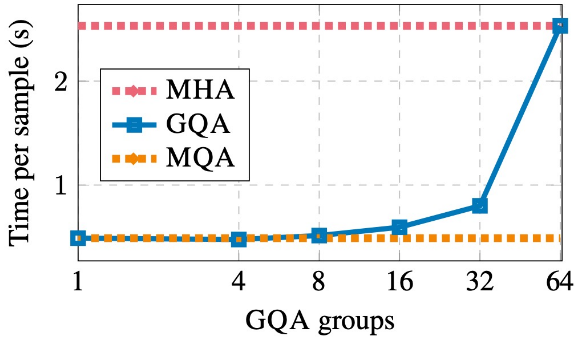
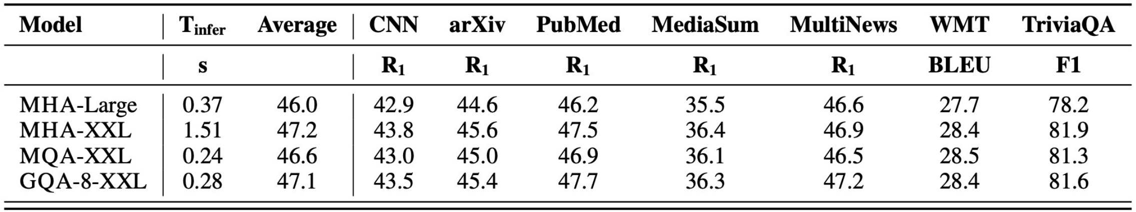


Figure 6: Time per sample for GQA-XXL as a function of the number of GQA groups with input length 2048 and output length 512. Going from 1 (MQA) to 8 groups adds modest inference overhead, with increasing cost to adding more groups.



Model	**T_{infer}**	**Average**	**CNN**	**arXiv**	**PubMed**	**MediaSum**	**MultiNews**	**WMT**	**TriviaQA**
	s		**R_1**	**R_1**	**R_1**	**R_1**	**R_1**	**BLEU**	**F1**
MHA-Large	0.37	46.0	42.9	44.6	46.2	35.5	46.6	27.7	78.2
MHA-XXL	1.51	47.2	43.8	45.6	47.5	36.4	46.9	28.4	81.9
MQA-XXL	0.24	46.6	43.0	45.0	46.9	36.1	46.5	28.5	81.3
GQA-8-XXL	0.28	47.1	43.5	45.4	47.7	36.3	47.2	28.4	81.6

Table 1: Inference time and average dev set performance comparison of T5 Large and XXL models with multi-head attention, and 5% uptrained T5-XXL models with multi-query and grouped-query attention on summarization datasets CNN/Daily Mail, arXiv, PubMed, MediaSum, and MultiNews, translation dataset WMT, and question-answering dataset TriviaQA.

Uptraining Results for MQA and GQA

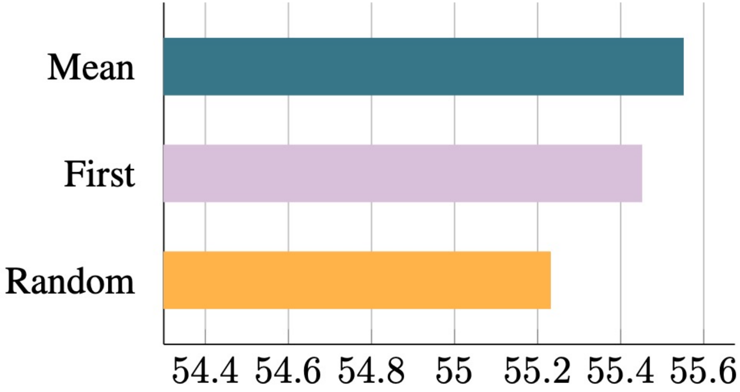


Figure 4: Performance comparison of different checkpoint conversion methods for T5-Large uptrained to MQA with proportion $\alpha = 0.05$. 'Mean' mean-pools key and value heads, 'First' selects the first head and 'Random' initializes heads from scratch.

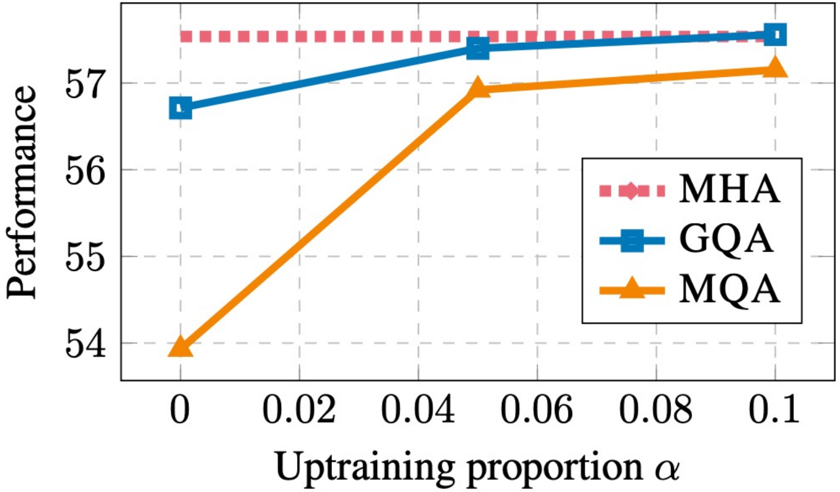
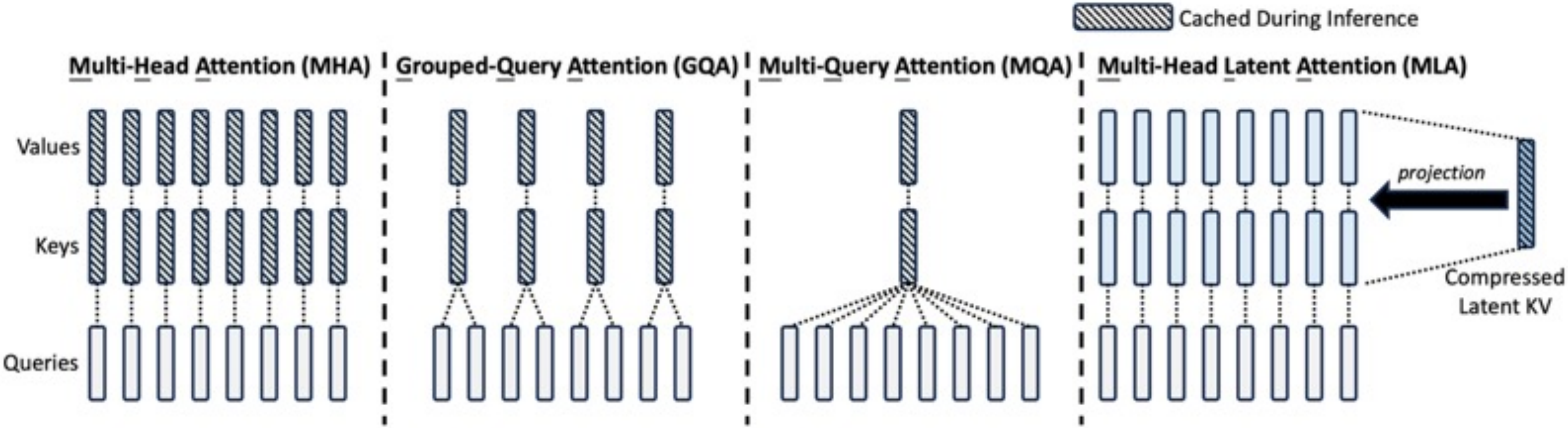


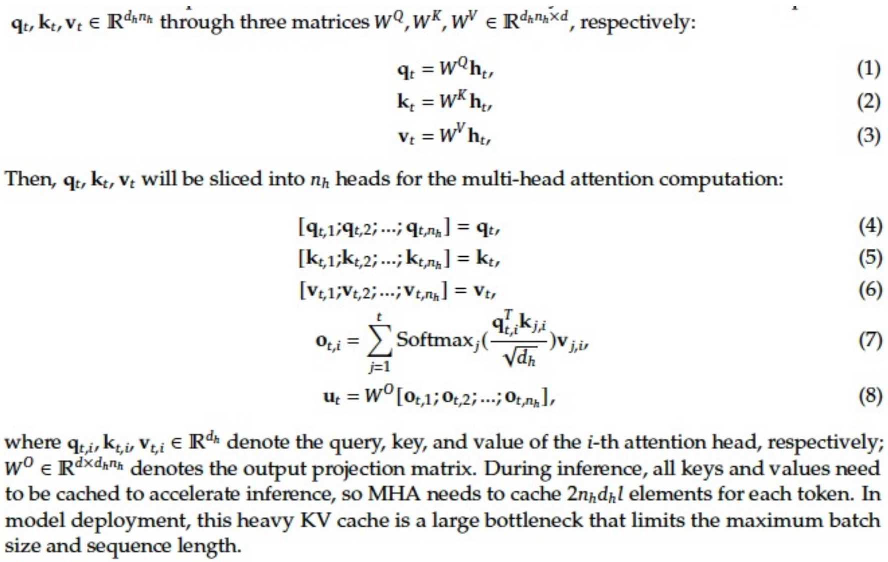
Figure 5: Performance as a function of uptraining proportion for T5 XXL models with MQA and GQA-8.

Multi-head Latent Attention (MLA) by DeepSeek



Source: DeepSeek v2 Technical Report

MHA vs. MLA



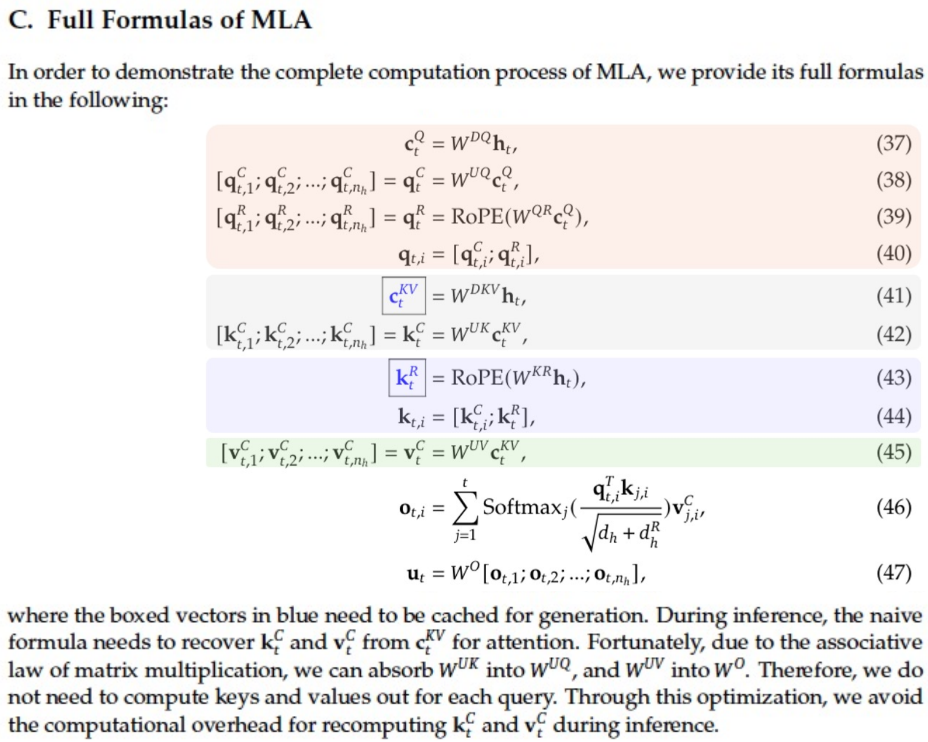
$\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^{d_h n_h}$ through three matrices $W^Q, W^K, W^V \in \mathbb{R}^{d_h n_h \times d}$, respectively:

$$\mathbf{q}_t = W^Q \mathbf{h}_t, \quad (1)$$
$$\mathbf{k}_t = W^K \mathbf{h}_t, \quad (2)$$
$$\mathbf{v}_t = W^V \mathbf{h}_t, \quad (3)$$

Then, $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$ will be sliced into n_h heads for the multi-head attention computation:

$$[\mathbf{q}_{t,1}; \mathbf{q}_{t,2}; ...; \mathbf{q}_{t,n_h}] = \mathbf{q}_t, \quad (4)$$
$$[\mathbf{k}_{t,1}; \mathbf{k}_{t,2}; ...; \mathbf{k}_{t,n_h}] = \mathbf{k}_t, \quad (5)$$
$$[\mathbf{v}_{t,1}; \mathbf{v}_{t,2}; ...; \mathbf{v}_{t,n_h}] = \mathbf{v}_t, \quad (6)$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h}}\right) \mathbf{v}_{j,i}, \quad (7)$$
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; ...; \mathbf{o}_{t,n_h}], \quad (8)$$

where $\mathbf{q}_{t,i}, \mathbf{k}_{t,i}, \mathbf{v}_{t,i} \in \mathbb{R}^{d_h}$ denote the query, key, and value of the i-th attention head, respectively; $W^O \in \mathbb{R}^{d \times d_h n_h}$ denotes the output projection matrix. During inference, all keys and values need to be cached to accelerate inference, so MHA needs to cache $2 n_h d_h l$ elements for each token. In model deployment, this heavy KV cache is a large bottleneck that limits the maximum batch size and sequence length.



C. Full Formulas of MLA

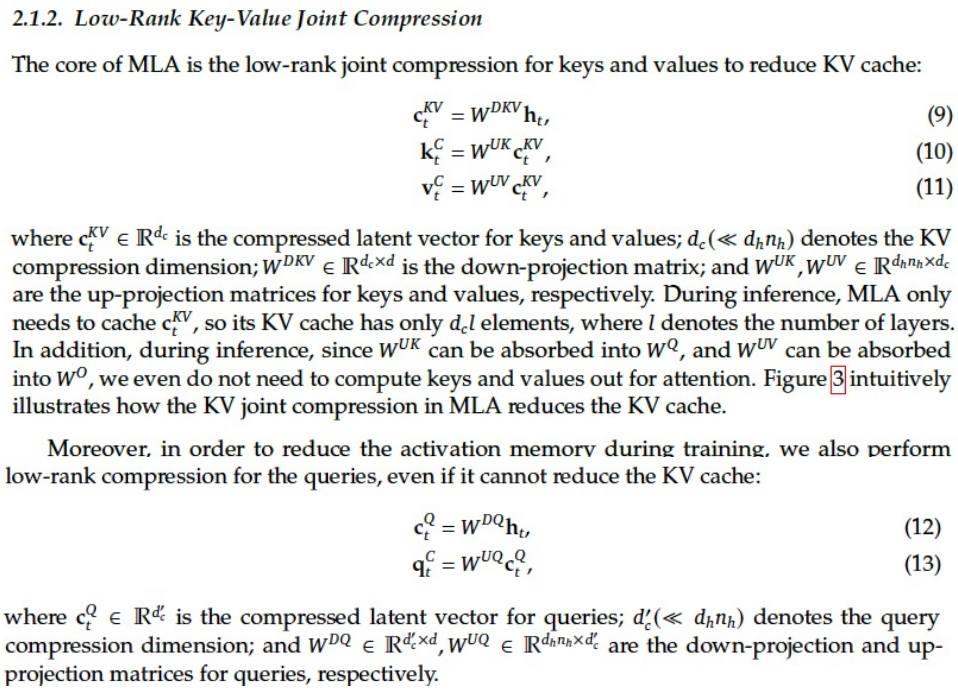
In order to demonstrate the complete computation process of MLA, we provide its full formulas in the following:

$$\mathbf{c}_t^Q = W^{DQ} \mathbf{h}_t, \quad (37)$$
$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; ...; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ} \mathbf{c}_t^Q, \quad (38)$$
$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; ...; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR} \mathbf{c}_t^Q), \quad (39)$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R], \quad (40)$$
$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV} \mathbf{h}_t, \quad (41)$$
$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; ...; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV}, \quad (42)$$
$$\boxed{\mathbf{k}_t^R} = \mathrm{RoPE}(W^{KR} \mathbf{h}_t), \quad (43)$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R], \quad (44)$$
$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; ...; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV}, \quad (45)$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right) \mathbf{v}_{j,i}^C, \quad (46)$$
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; ...; \mathbf{o}_{t,n_h}], \quad (47)$$

where the boxed vectors in blue need to be cached for generation. During inference, the naive formula needs to recover $\mathbf{k}_t^C$ and $\mathbf{v}_t^C$ from $\mathbf{c}_t^{KV}$ for attention. Fortunately, due to the associative law of matrix multiplication, we can absorb W^{UK} into W^{UQ}, and W^{UV} into W^O. Therefore, we do not need to compute keys and values out for each query. Through this optimization, we avoid the computational overhead for recomputing $\mathbf{k}_t^C$ and $\mathbf{v}_t^C$ during inference.

Source: DeepSeek v2 Technical Report
Additional References: https://epoch.ai/gradient-updates/how-has-deepseek-improved-the-transformer-architecture
https://planetbanatt.net/articles/mla.html
https://towardsdatascience.com/deepseek-v3-explained-1-multi-head-latent-attention-ed6bee2a67c4/

Decoupled RoPE is needed for MLA



2.1.2. *Low-Rank Key-Value Joint Compression*

The core of MLA is the low-rank joint compression for keys and values to reduce KV cache:

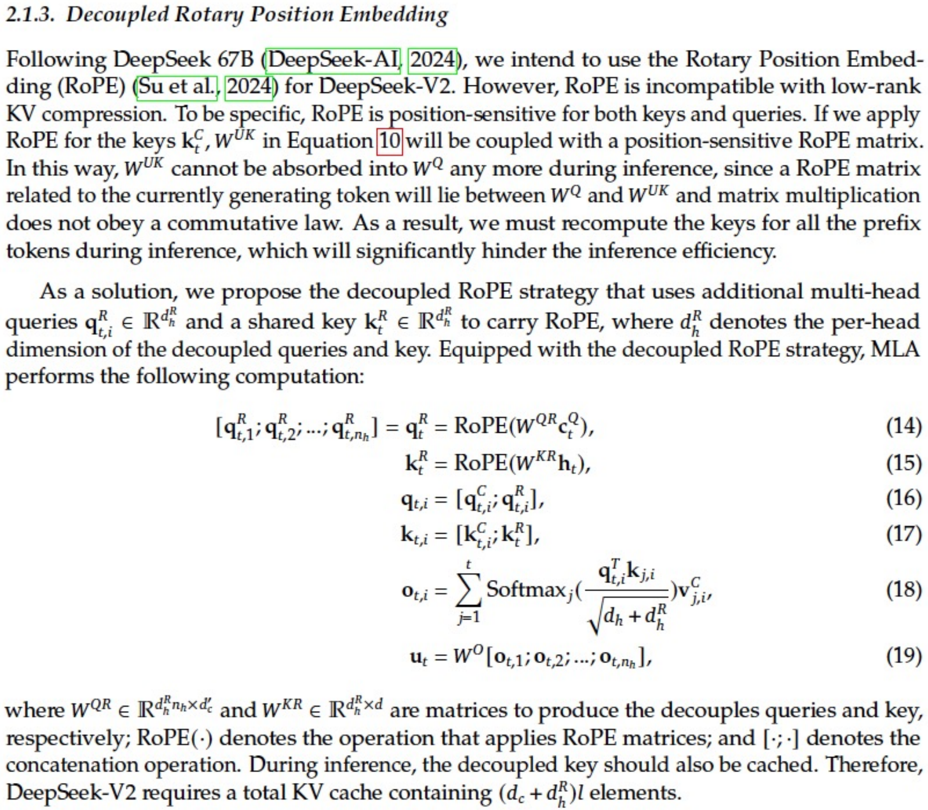
$$\mathbf{c}_t^{KV} = W^{DKV}\mathbf{h}_t, \tag{9}$$
$$\mathbf{k}_t^{C} = W^{UK}\mathbf{c}_t^{KV}, \tag{10}$$
$$\mathbf{v}_t^{C} = W^{UV}\mathbf{c}_t^{KV}, \tag{11}$$

where $\mathbf{c}_t^{KV} \in \mathbb{R}^{d_c}$ is the compressed latent vector for keys and values; $d_c (\ll d_h n_h)$ denotes the KV compression dimension; $W^{DKV} \in \mathbb{R}^{d_c \times d}$ is the down-projection matrix; and $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$ are the up-projection matrices for keys and values, respectively. During inference, MLA only needs to cache $\mathbf{c}_t^{KV}$, so its KV cache has only $d_c l$ elements, where l denotes the number of layers. In addition, during inference, since W^{UK} can be absorbed into W^Q, and W^{UV} can be absorbed into W^O, we even do not need to compute keys and values out for attention. Figure 3 intuitively illustrates how the KV joint compression in MLA reduces the KV cache.

Moreover, in order to reduce the activation memory during training, we also perform low-rank compression for the queries, even if it cannot reduce the KV cache:

$$\mathbf{c}_t^{Q} = W^{DQ}\mathbf{h}_t, \tag{12}$$
$$\mathbf{q}_t^{C} = W^{UQ}\mathbf{c}_t^{Q}, \tag{13}$$

where $\mathbf{c}_t^{Q} \in \mathbb{R}^{d_c'}$ is the compressed latent vector for queries; $d_c' (\ll d_h n_h)$ denotes the query compression dimension; and $W^{DQ} \in \mathbb{R}^{d_c' \times d}, W^{UQ} \in \mathbb{R}^{d_h n_h \times d_c'}$ are the down-projection and up-projection matrices for queries, respectively.



2.1.3. *Decoupled Rotary Position Embedding*

Following DeepSeek 67B (DeepSeek-AI, 2024), we intend to use the Rotary Position Embedding (RoPE) (Su et al., 2024) for DeepSeek-V2. However, RoPE is incompatible with low-rank KV compression. To be specific, RoPE is position-sensitive for both keys and queries. If we apply RoPE for the keys $\mathbf{k}_t^C$, W^{UK} in Equation 10 will be coupled with a position-sensitive RoPE matrix. In this way, W^{UK} cannot be absorbed into W^Q any more during inference, since a RoPE matrix related to the currently generating token will lie between W^Q and W^{UK} and matrix multiplication does not obey a commutative law. As a result, we must recompute the keys for all the prefix tokens during inference, which will significantly hinder the inference efficiency.

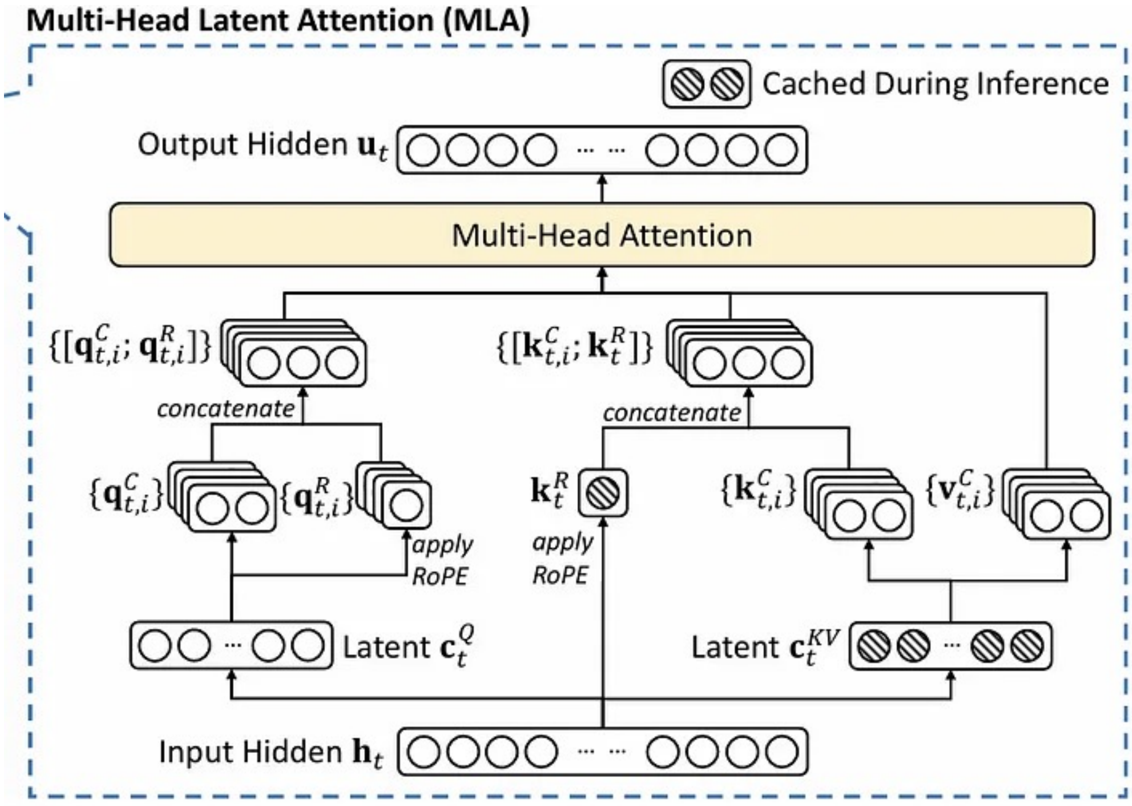
As a solution, we propose the decoupled RoPE strategy that uses additional multi-head queries $\mathbf{q}_{t,i}^R \in \mathbb{R}^{d_h^R}$ and a shared key $\mathbf{k}_t^R \in \mathbb{R}^{d_h^R}$ to carry RoPE, where d_h^R denotes the per-head dimension of the decoupled queries and key. Equipped with the decoupled RoPE strategy, MLA performs the following computation:

$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; \ldots; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR}\mathbf{c}_t^Q), \tag{14}$$
$$\mathbf{k}_t^R = \mathrm{RoPE}(W^{KR}\mathbf{h}_t), \tag{15}$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R], \tag{16}$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R], \tag{17}$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right)\mathbf{v}_{j,i}^C, \tag{18}$$
$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \ldots; \mathbf{o}_{t,n_h}], \tag{19}$$

where $W^{QR} \in \mathbb{R}^{d_h^R n_h \times d_c'}$ and $W^{KR} \in \mathbb{R}^{d_h^R \times d}$ are matrices to produce the decouples queries and key, respectively; $\mathrm{RoPE}(\cdot)$ denotes the operation that applies RoPE matrices; and $[\cdot;\cdot]$ denotes the concatenation operation. During inference, the decoupled key should also be cached. Therefore, DeepSeek-V2 requires a total KV cache containing $(d_c + d_h^R)l$ elements.

Source: Source: DeepSeek v2 Technical Report /

Multi-head Latent Attention (cont'd)



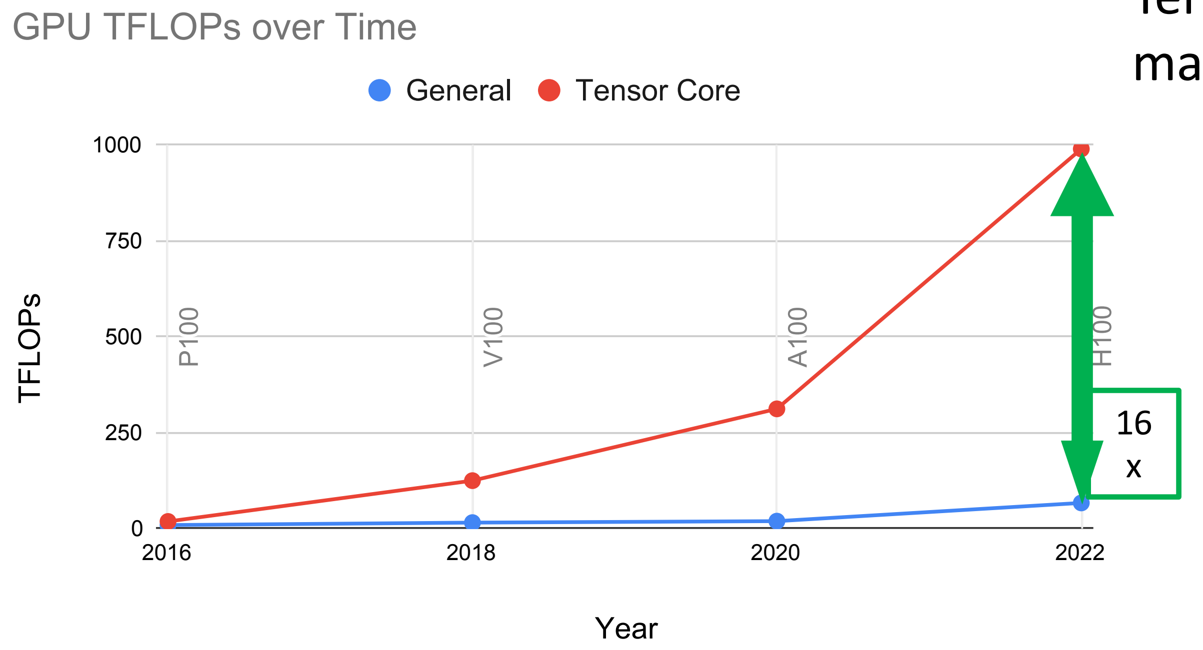
Source: DeepSeek v2 Technical Report

Hardware-Aware Attention WITHOUT Approximation

GPU basics

1. A100 GPU, a standard GPU (was SOTA in 2020)
    1. 40 – 80GB of HBM, 1.5-2.0TB/s
    2. 20MB of SRAM, 19TB/s
    3. SRAM are much smaller, but much faster
2. GPUs have many threads to execute an operation (called a kernel). Each kernel loads inputs from HBM to registers and SRAM, computes, then writes outputs to HBM
3. Operations are either (depending on op-to-mem-access ratio)
    1. Compute-bound: e.g., matrix multiply, convolution
    2. Memory-bound: e.g., activation, dropout, sum, softmax, batch norm

First Component of Modern GPU:
Big Compute (eg., NVidia Tensor Cores)



Tensor cores multiply 16x16 matrices (very roughly)

Speed difference with tensor cores is ***increasing***

- 4x on V100,
- 8x on A100, and
- 16x on H100

With tensor cores versus without (across precisions).

"All that matters is locality." – Paraphrasing, Mark Horowitz.

Credit: Dan Fu

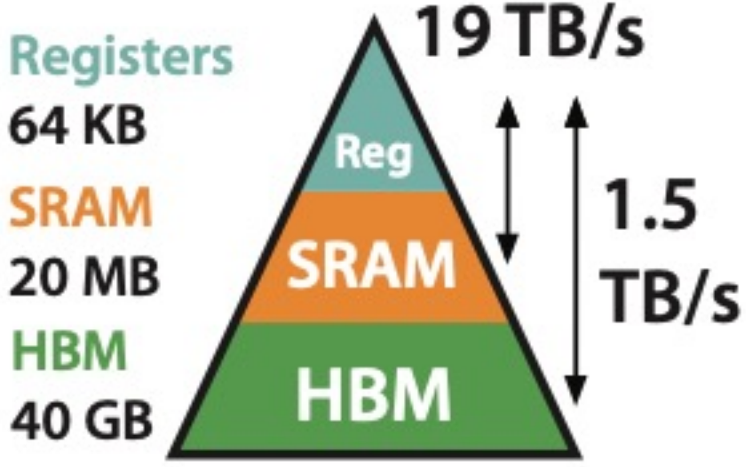
Memory Hierarchy of Modern GPU:

(Simplified) Memory hierarchy

- Registers
- SRAM
- Memory

Small, Fast memory (Registers/SRAM)
Big, Slow memory (HBM)

Database people count IO as reads-and-writes from HBM (slow memory).



GPU Memory Hierarchy

FlashAttention (v1, v2):
Fast & Memory-Efficient Exact Attention w/ IO-Awareness

Main Idea:

- Minimize IO (HBM to SRAM) – not FLOPs
- Aggressive **fusion:** when you pull in data use it.

*Two **classical** ideas from database researchers.*

Tri Dao

PRINCETON
UNIVERSITY

Dan Fu

UC San Diego
JACOBS SCHOOL OF ENGINEERING

together.ai

Up to 72% Utilization—
15% faster BERT on MLPerf 1.1

We're Training AI Twice as Fast This Year as Last › New MLPerf rankings show training times plunging

BY SAMUEL K. MOORE | 30 JUN 2022 | 5 MIN READ

ML Perf Winners use it!

https://crfm.stanford.edu/2023/01/13/flashattention.html

https://spectrum.ieee.org/mlperf-rankings-2022

How FlashAttention works ?

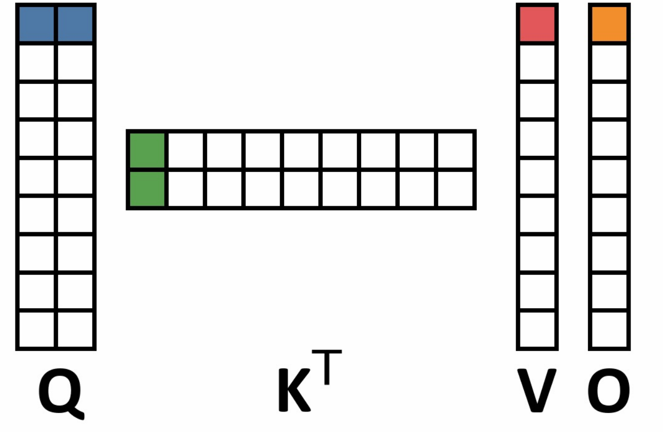
Prof. Jia bin Huang of UMD

https://www.youtube.com/watch?v=gBMO1JZav44

[From Online Softmax to FlashAttention] https://courses.cs.washington.edu/courses/cse599m/23sp/notes/flashattn.pdf

- Minimize IO (HBM to SRAM) – not FLOPs
- Aggressive **fusion:** when you pull in data use it.

Minimize IO to HBM in Flash Attention

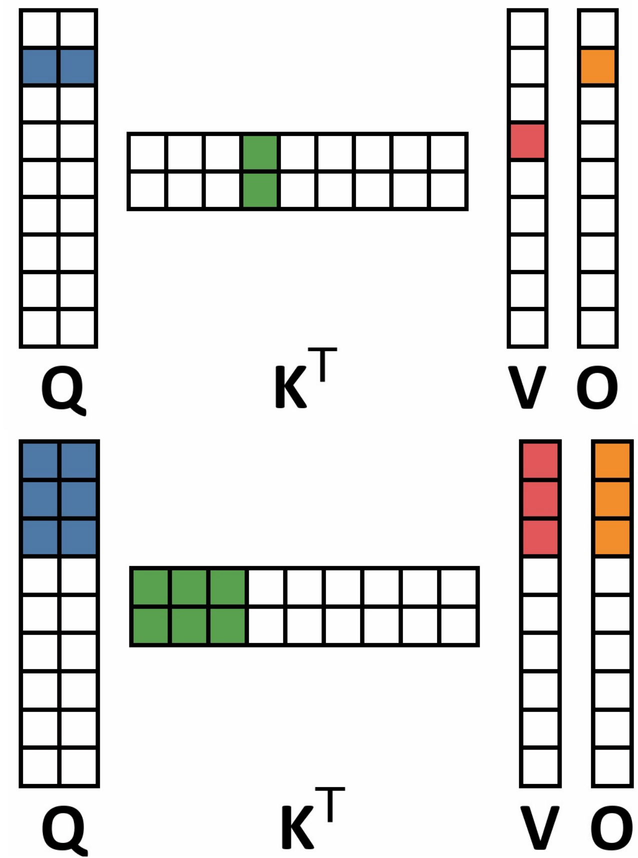


Database people call *"nested loop join"*
r(Q) = 9 is the rows of Q, |Q| is number of tiles.

|Q| + r(Q)(|K| + |V|) + |O| = 18 + 9×(18+9) + 9 = 270 IO

- Minimize IO (HBM to SRAM) – not FLOPs
- Aggressive **fusion:** when you pull in data use it.

Minimize IO to HBM in Flash Attention



Database people call *"nested loop join"*
r(Q) = 9 is the rows of Q, |Q| is number of tiles.

|Q| + r(Q)(|K| + |V|) + |O| = 18 + 9×(18+9) + 9 = 270 IO

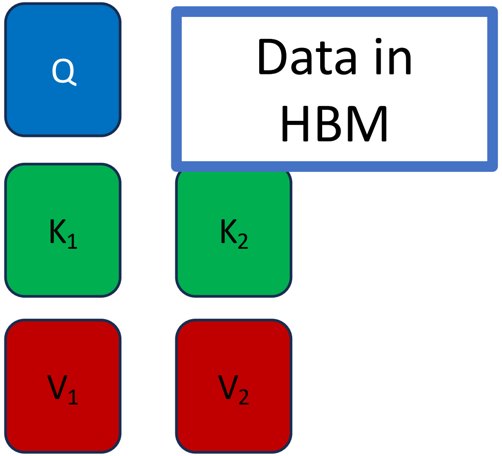
Database Idea: **Block** Nested Loop Join.
Read 3 blocks at once, so b(Q) = 9/3 = 3.

|Q| + ***b(Q)***(|K| + |V|) + |O| = 18 + 3×(18+9) + 9 = 108 IO

Same FLOPS but ~**3x reduction** in IO w/ block size 3.
Flash Attention A100 uses 8x8 blocks.

IO-Aware Attention

- Minimize IO (HBM to SRAM) – not FLOPs
- Aggressive **fusion:** when you pull in data use it.



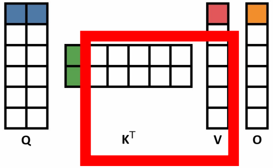
Data in HBM

Data in SRAM/Registers

```
Load($Q$,$K_1$,$V_1$)
$A_1$ = exp($Q_1K_1$.T) // attention scores
$z_1$ = $A_1$.sum(-1) // normalization
$O_1$ = $A_1V_1/z_1$ // compute output.
Free($K_1$,$V_1$,$A_1$)
```

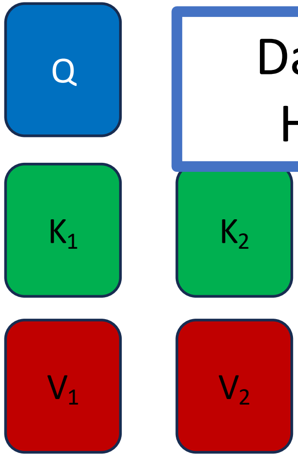
Incorrect Normalization! Normalization should depend on the rest of K and V—we haven't seen them!

$$A = \exp(QK^T)$$
$$W = AV$$
$$Z = A.\text{sum}(-1)$$
$$O = W/Z$$

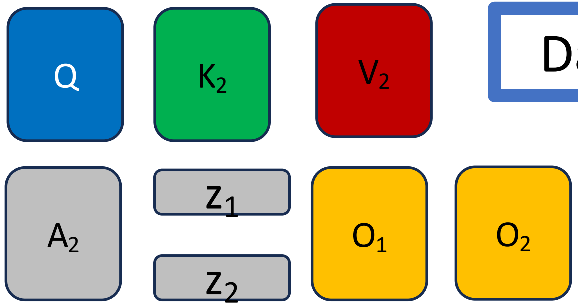


IO-Aware Attention

- Minimize IO (HBM to SRAM) – not FLOPs
- Aggressive **fusion:** when you pull in data use it.



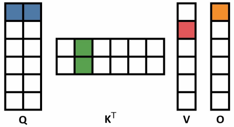
Data in HBM



Data in SRAM/Registers

```
Load($K_2$,$V_2$)
$A_2$ = exp($Q_2K_2$.T) // attention scores
$z_2$ = $z_1$+$A_2$.sum(-1) // normalization
$O_2$ = $A_2V_2/z_2$ // compute output
$O_2$ = $O_2$ + $z_1/z_2O_1$ // renormalize $O_1$!
Free($K_2$,$V_2$,$A_2$,$Z_1$,$O_1$)
```

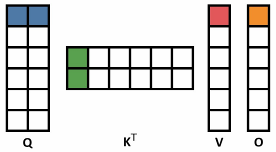
$A = \exp(QK^T)$
$W = AV$
$Z = A.\mathrm{sum}(-1)$
$O = W/Z$



NB: To Fix Normalization only need to keep last (O_t,Z_t)

Summary of Block-Aware Attention to FlashAttention

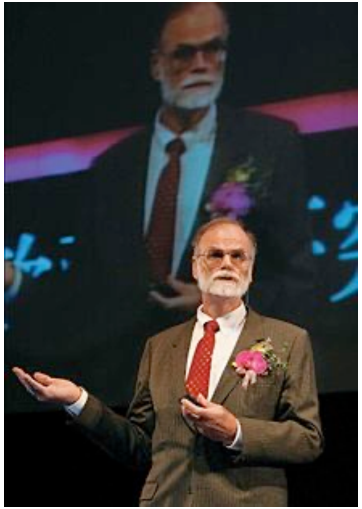
- Minimize HBM IO by **blocked I/O.**



- To **fuse**, we **aggregate** like running sum to compute attention exactly.
  - In database terms, softmax is an *algebraic aggregation [Gray07].*

```
Load(K2,V2)
A2 = exp(Q2K2.T) // attention
scores
z2  = z1+A2.sum(-1) //
normalization
O2 = A2V2/z2 // compute the
output
O2 = O2 + z1/z2O1 // renormalized!
Free(K2,V2,A2,Z1,O1)
```

- FlashAttention essentially block-nested loop join from classical databases.



Jim Gray, Turing Award 1998.

There is a whole canon of systems to re-explore for AI!

Open-Source, Quickly Adopted by the AI community !

PyTorch

HUGGING FACE

together ai

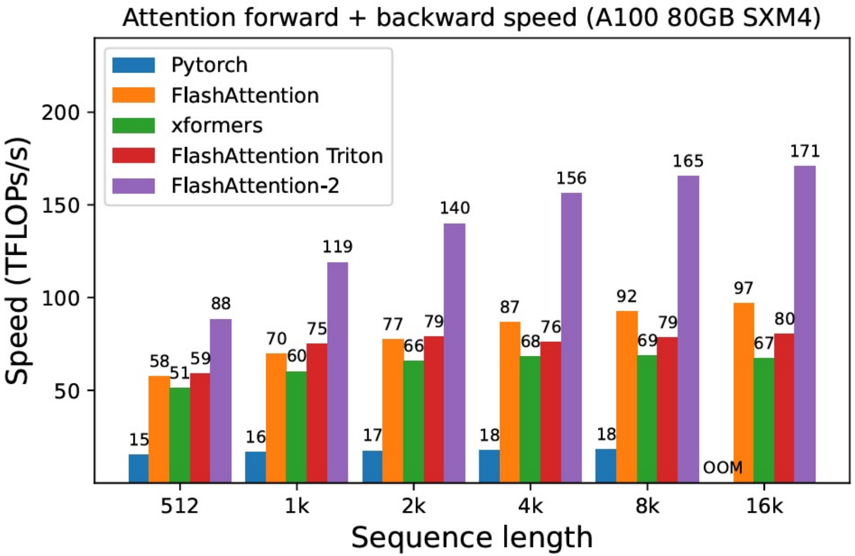
JAX

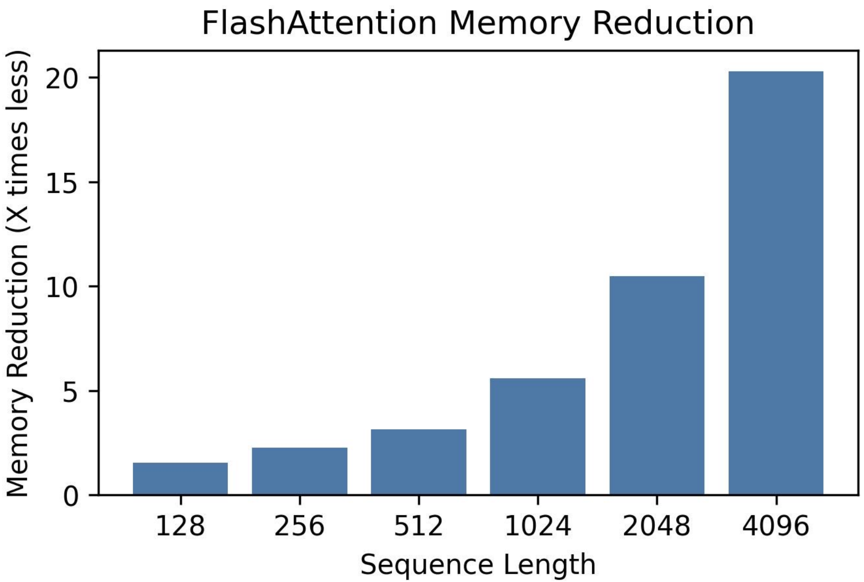
Microsoft

NVIDIA

Meta

FlashAttention: 6-10x speedup, 10-20x memory reduction





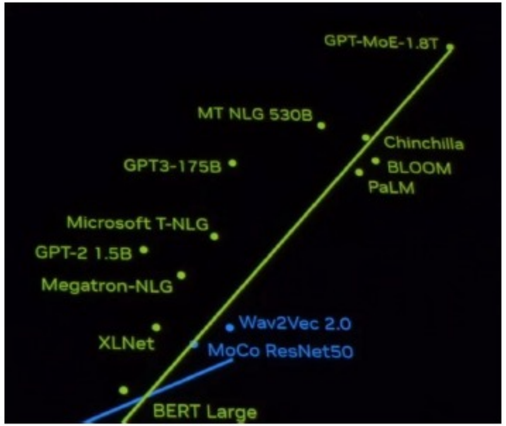
6-10x speedup — with no approximation

10-20x memory reduction

Mixture of Experts (MoE) architecture for Transformers

Mixtral of experts

A high quality Sparse Mixture-of-Experts.



GPT4 (?)

Mistral AI
@MistralAI

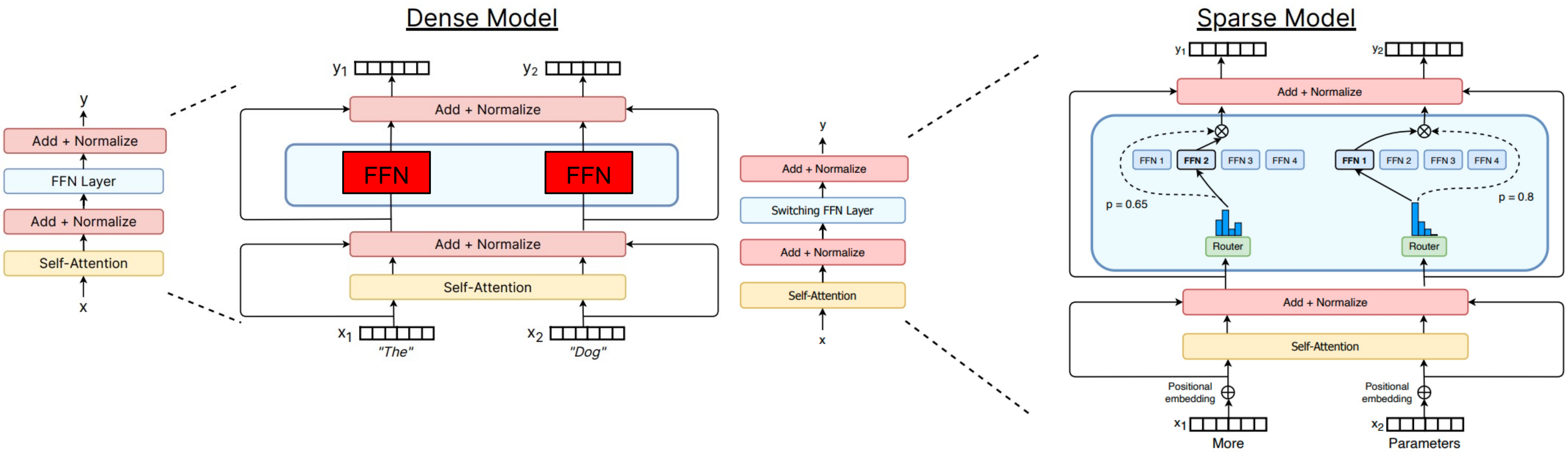
magnet:?
xt=urn:btih:9238b09245d0d8cd915be09927769d5f7584c1c9&dn=mixtral-
8x22b&tr=udp%3A%2F%2Fopen.demonii.com%3A1337%2Fannounce&tr=http%3A%2F%2Ftracker.opentrackr.org%3A1337%2Fannounce

Grok

deepseek

DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models

What is a Mixture of Expert (MoE) ?



Token-specific routing

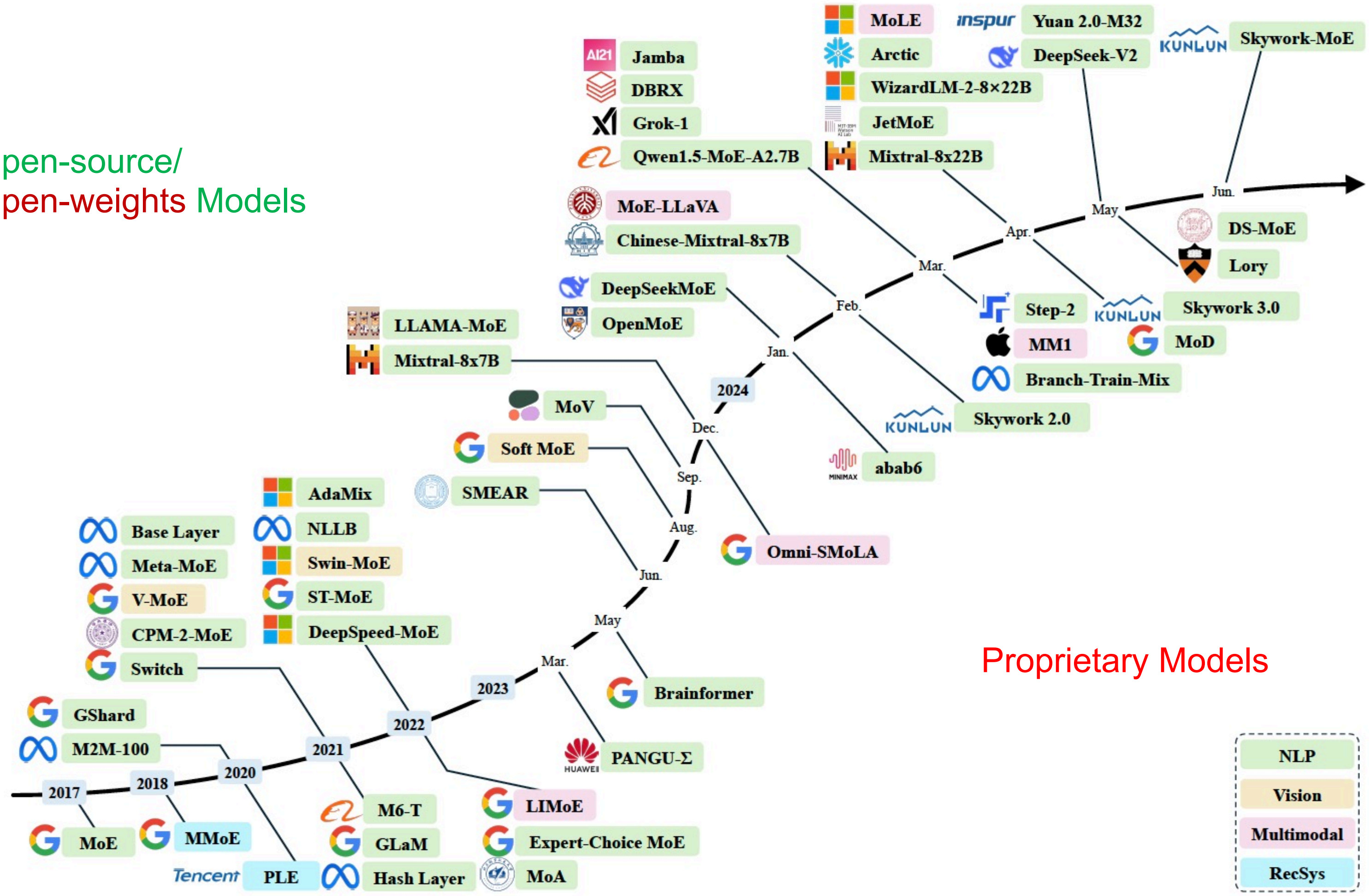
[Fedus et al Switch Transformer … 2022]

- Replace the VERY BIG Feed Forward Network (FFN) with (multiple) not-as-BIG FFNs and a Selector Layer / Gating Function to Pick “TopK” FFNs.
- Can increase # of Experts without affecting FLOPS (during inference time)

  * For each token, only a subset (TopK) of Experts need to be Active during inference
- Performance better than one single Very BIG FFN of the same total parameter count !

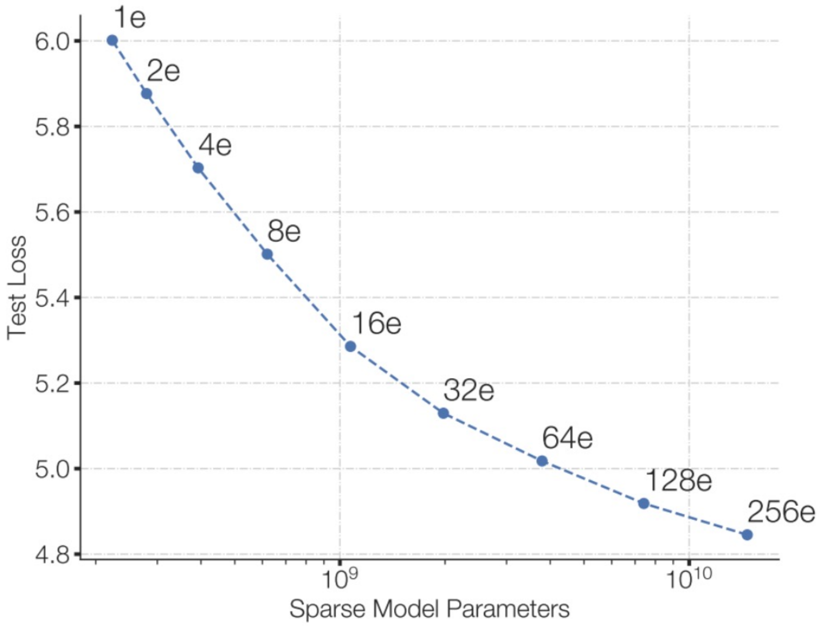
MoE Models

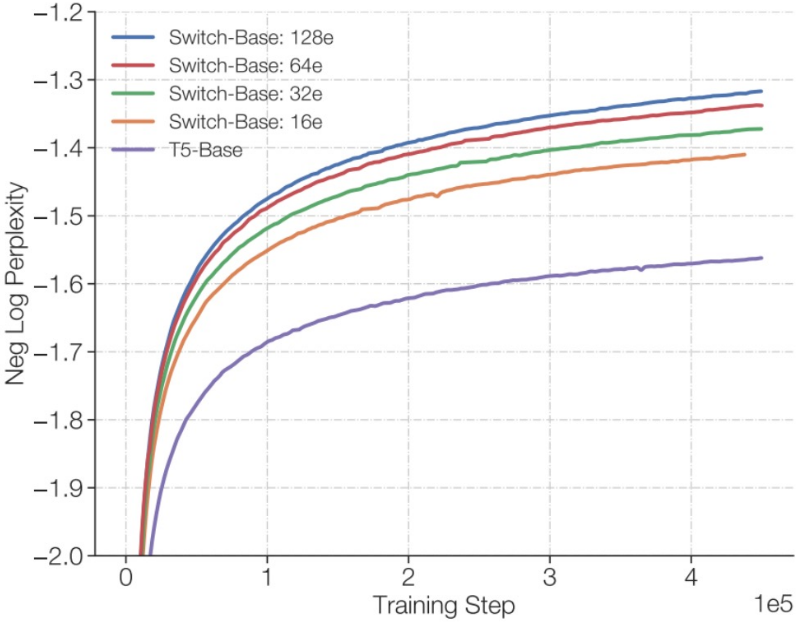
Open-source/
Open-weights Models



Proprietary Models

Why do MoEs become popular ?



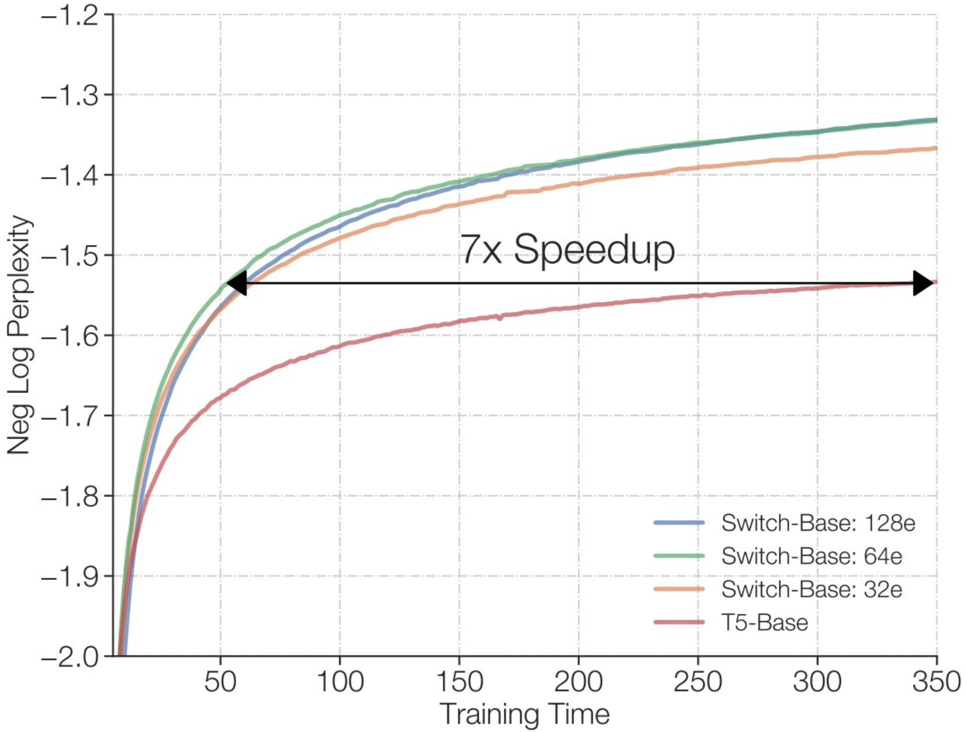


[Fedus et al Switch Transformer … 2022]

- Same FLOP, larger parameter-count does better

Why do MoEs become popular ?

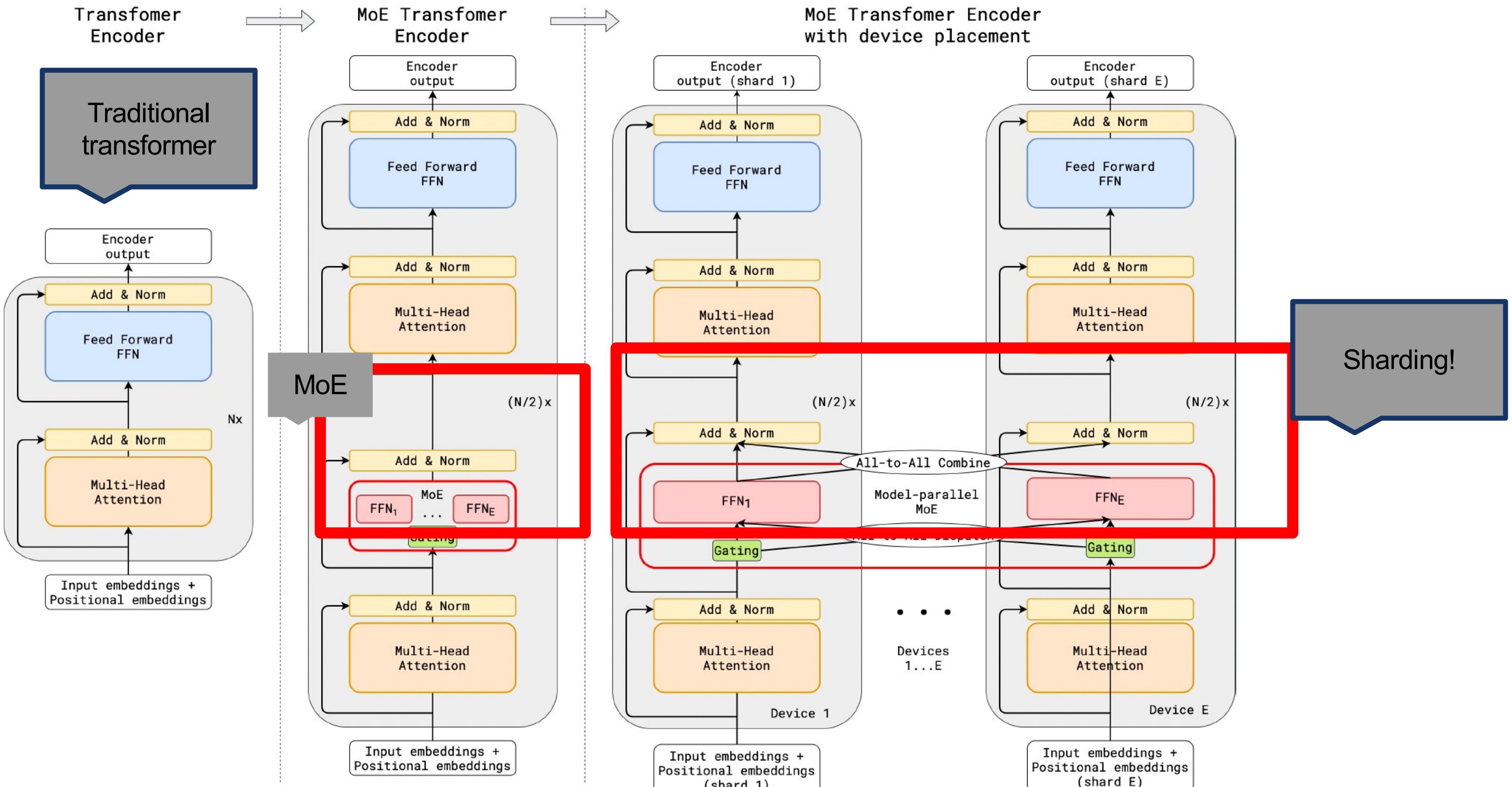
- Faster to train MoEs



[Fedus et al Switch Transformer … 2022]

Why do MoEs become popular ?

- Parallelizable to Many Devices (Machines) - Example from Gshard

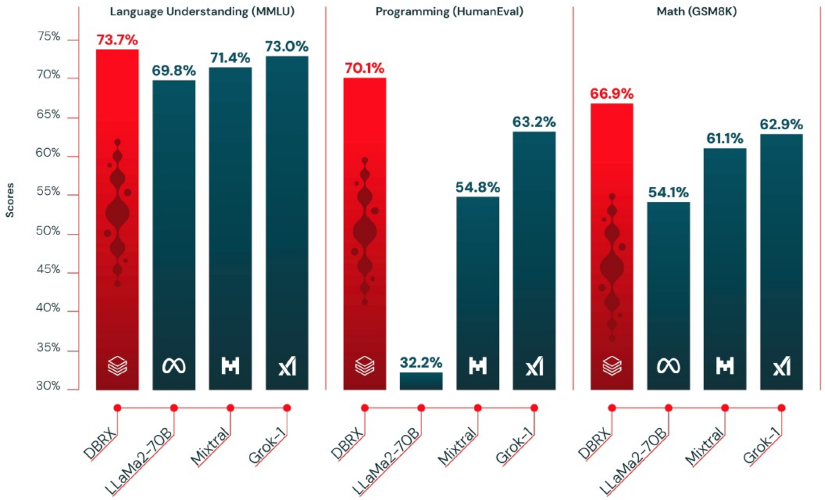


Traditional transformer

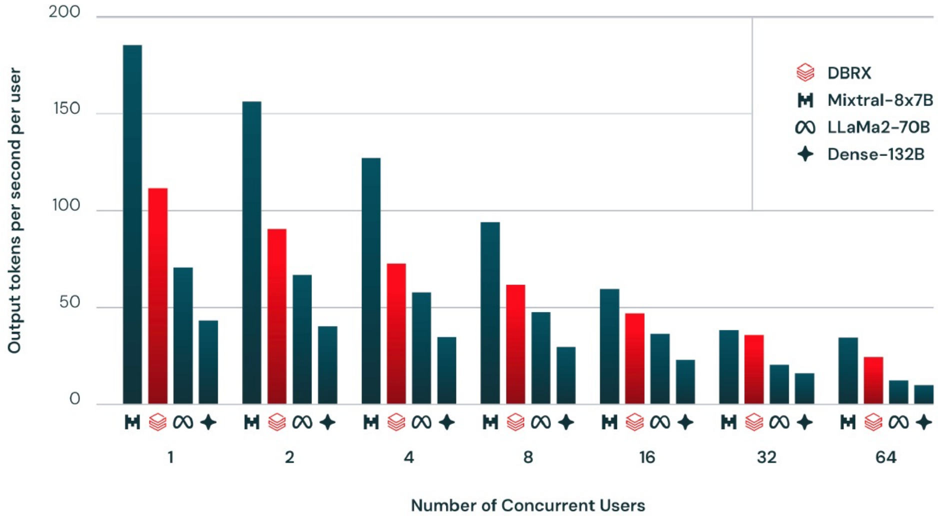
MoE

Sharding!

Some Recent MoE Results: Mixtral / DBRX / Grok



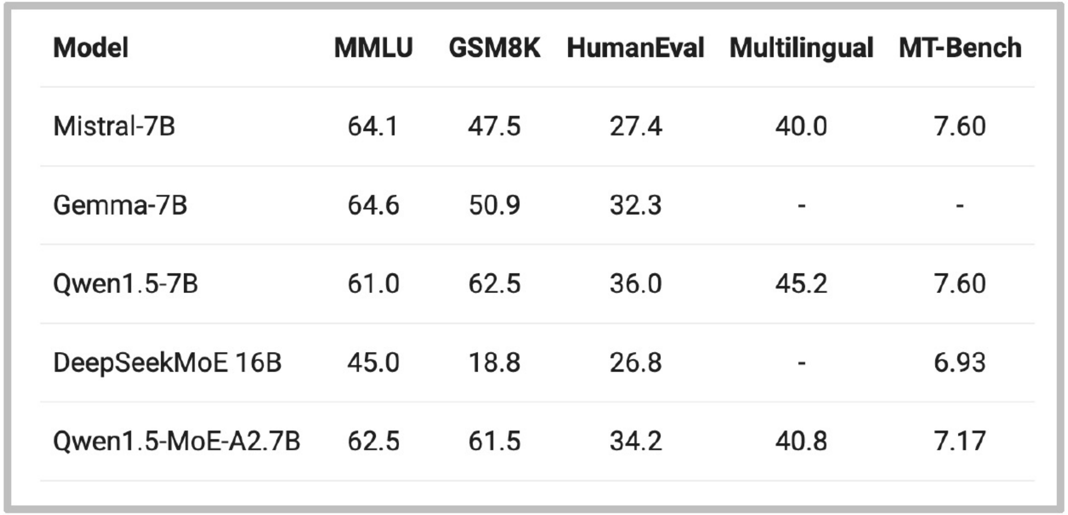
Performance



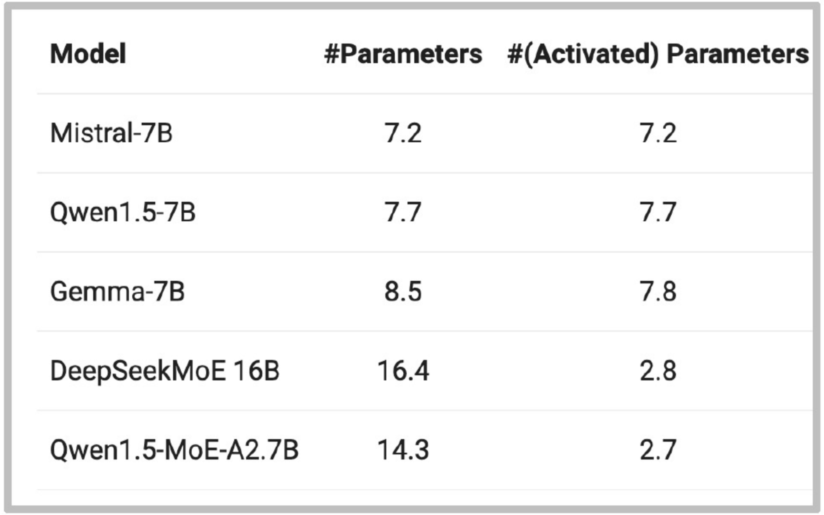
Inference throughput

- Most of the highest-performance Open Models are MoEs, and they are quite fast !

Some Recent MoE Results: Qwen & DeepSeekMoE(2024)



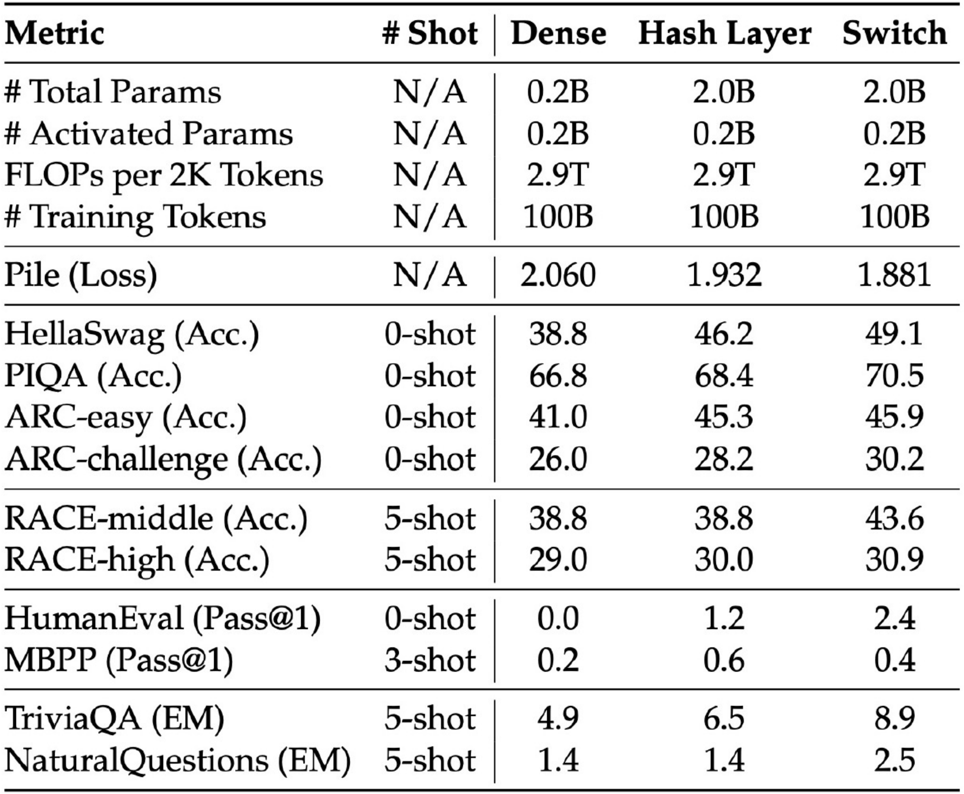
Model	MMLU	GSM8K	HumanEval	Multilingual	MT-Bench
Mistral-7B	64.1	47.5	27.4	40.0	7.60
Gemma-7B	64.6	50.9	32.3	-	-
Qwen1.5-7B	61.0	62.5	36.0	45.2	7.60
DeepSeekMoE 16B	45.0	18.8	26.8	-	6.93
Qwen1.5-MoE-A2.7B	62.5	61.5	34.2	40.8	7.17



Model	#Parameters	#(Activated) Parameters
Mistral-7B	7.2	7.2
Qwen1.5-7B	7.7	7.7
Gemma-7B	8.5	7.8
DeepSeekMoE 16B	16.4	2.8
Qwen1.5-MoE-A2.7B	14.3	2.7

- Chinese LLM companies, e.g. Qwen and DeepSeek, have shown strength in their MoE work on the smaller-model end !

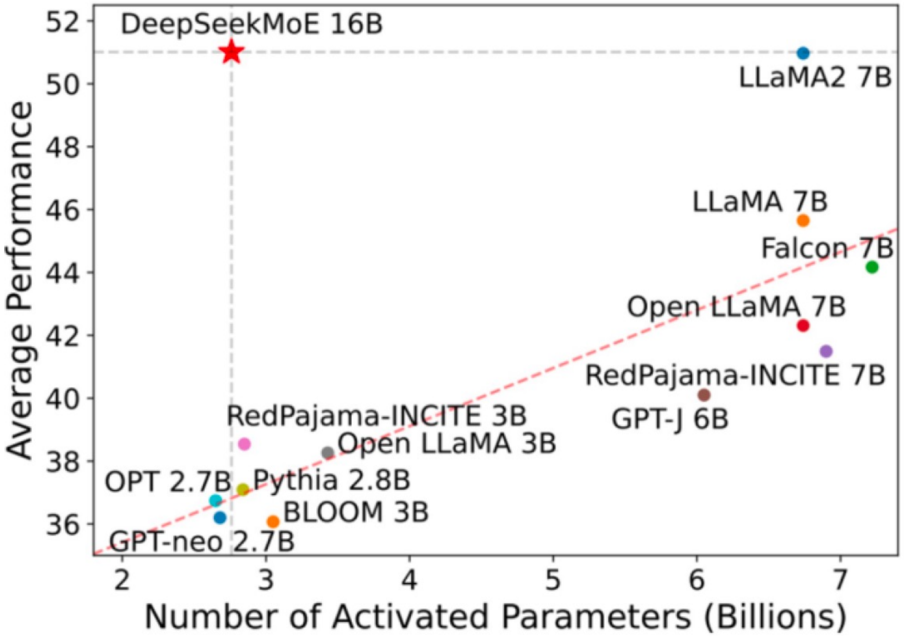
Recent Ablation Study on MoE Performance: DeepSeek

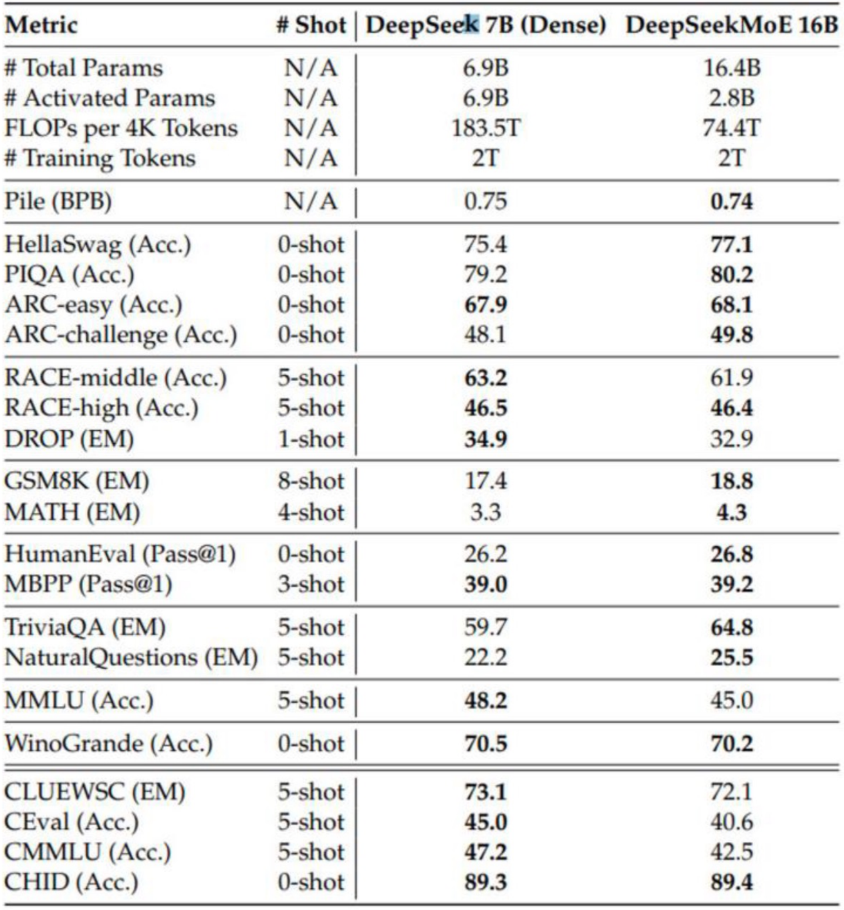


Metric	# Shot	Dense	Hash Layer	Switch
# Total Params	N/A	0.2B	2.0B	2.0B
# Activated Params	N/A	0.2B	0.2B	0.2B
FLOPs per 2K Tokens	N/A	2.9T	2.9T	2.9T
# Training Tokens	N/A	100B	100B	100B
Pile (Loss)	N/A	2.060	1.932	1.881
HellaSwag (Acc.)	0-shot	38.8	46.2	49.1
PIQA (Acc.)	0-shot	66.8	68.4	70.5
ARC-easy (Acc.)	0-shot	41.0	45.3	45.9
ARC-challenge (Acc.)	0-shot	26.0	28.2	30.2
RACE-middle (Acc.)	5-shot	38.8	38.8	43.6
RACE-high (Acc.)	5-shot	29.0	30.0	30.9
HumanEval (Pass@1)	0-shot	0.0	1.2	2.4
MBPP (Pass@1)	3-shot	0.2	0.6	0.4
TriviaQA (EM)	5-shot	4.9	6.5	8.9
NaturalQuestions (EM)	5-shot	1.4	1.4	2.5

- Recent Ablation study on MoEs shows they are generally good !

Performance of DeepSeekMoE 16B (2024)





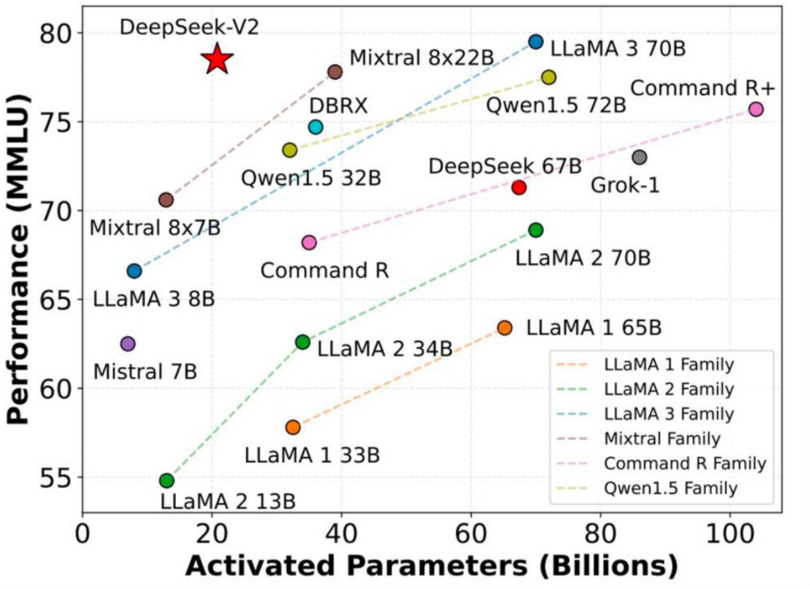
Metric	**# Shot**	**DeepSeek 7B (Dense)**	**DeepSeekMoE 16B**
# Total Params	N/A	6.9B	16.4B
# Activated Params	N/A	6.9B	2.8B
FLOPs per 4K Tokens	N/A	183.5T	74.4T
# Training Tokens	N/A	2T	2T
Pile (BPB)	N/A	0.75	**0.74**
HellaSwag (Acc.)	0-shot	75.4	**77.1**
PIQA (Acc.)	0-shot	79.2	**80.2**
ARC-easy (Acc.)	0-shot	**67.9**	**68.1**
ARC-challenge (Acc.)	0-shot	48.1	**49.8**
RACE-middle (Acc.)	5-shot	**63.2**	61.9
RACE-high (Acc.)	5-shot	**46.5**	**46.4**
DROP (EM)	1-shot	**34.9**	32.9
GSM8K (EM)	8-shot	17.4	**18.8**
MATH (EM)	4-shot	3.3	**4.3**
HumanEval (Pass@1)	0-shot	26.2	**26.8**
MBPP (Pass@1)	3-shot	**39.0**	**39.2**
TriviaQA (EM)	5-shot	59.7	**64.8**
NaturalQuestions (EM)	5-shot	22.2	**25.5**
MMLU (Acc.)	5-shot	**48.2**	45.0
WinoGrande (Acc.)	0-shot	**70.5**	**70.2**
CLUEWSC (EM)	5-shot	**73.1**	72.1
CEval (Acc.)	5-shot	**45.0**	40.6
CMMLU (Acc.)	5-shot	**47.2**	42.5
CHID (Acc.)	0-shot	**89.3**	**89.4**

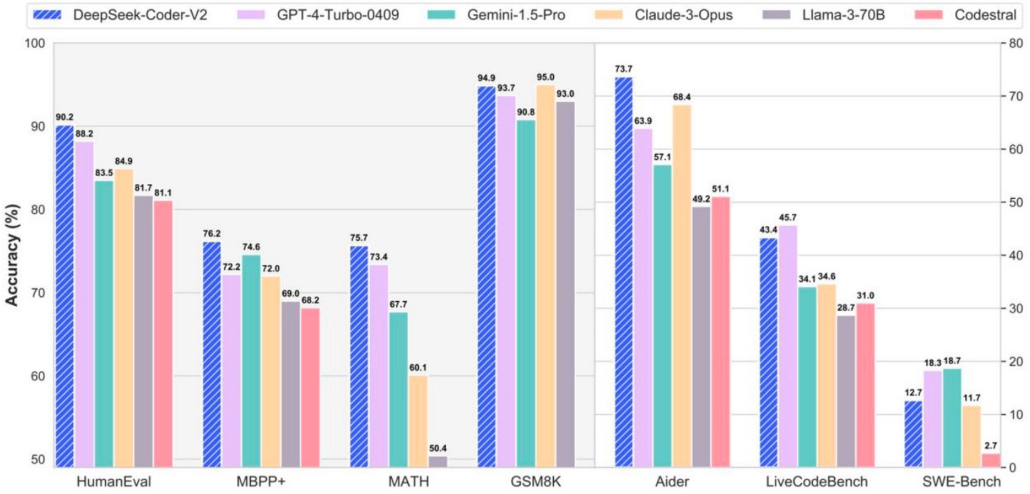
Source: DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models

Performance of DeepSeek-V2

Deepseek-V2
236B total parameters, 21B are activated.
2 shared experts and 160 routed experts (6 select).

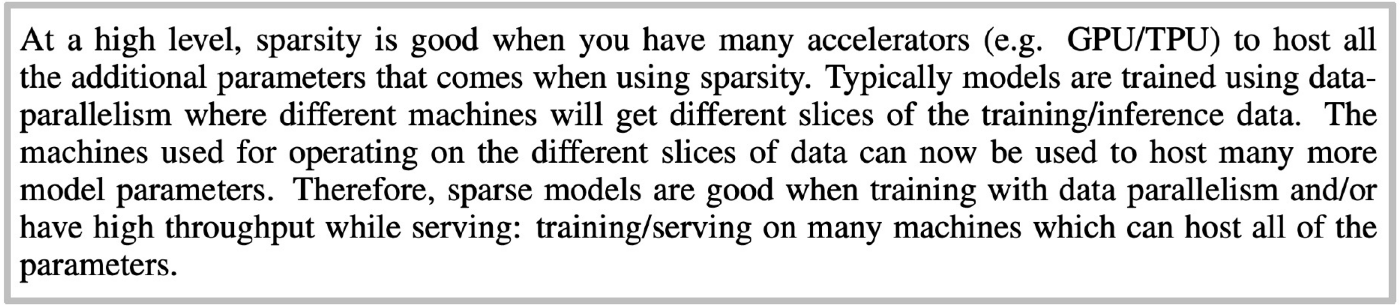
Deepseek-Coder-V2
Continue pretraining from an intermediate checkpoint of Deepseek-V2 (4.2T) and further train 6T. Total 10.2T tokens.





Why haven’t MoEs been more popular before ?

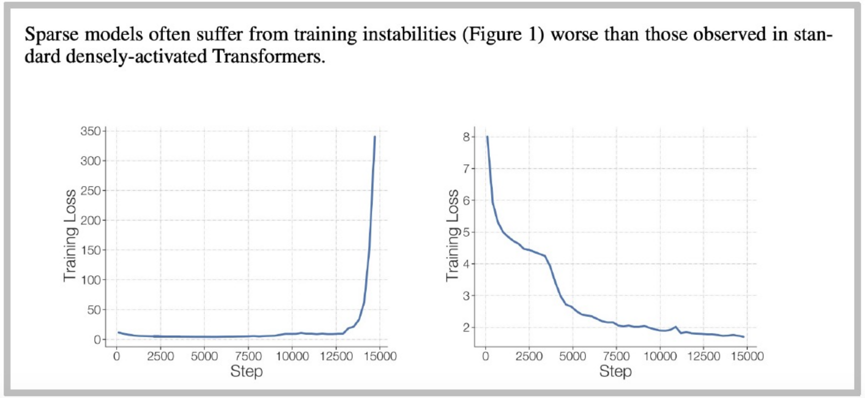
Infrastructure is complex / advantages on multi node



At a high level, sparsity is good when you have many accelerators (e.g. GPU/TPU) to host all the additional parameters that comes when using sparsity. Typically models are trained using data-parallelism where different machines will get different slices of the training/inference data. The machines used for operating on the different slices of data can now be used to host many more model parameters. Therefore, sparse models are good when training with data parallelism and/or have high throughput while serving: training/serving on many machines which can host all of the parameters.

[Fedus et al 2022]

Training objectives are somewhat heuristic (and sometimes unstable)

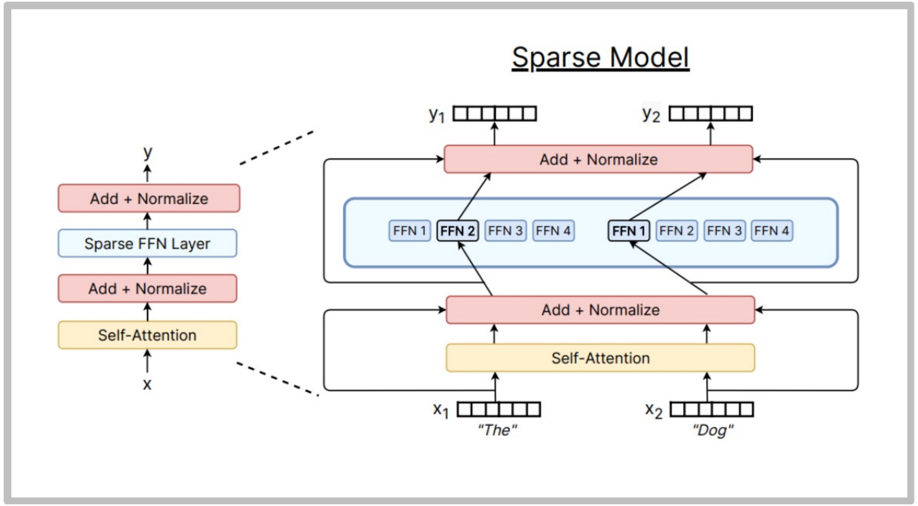


Sparse models often suffer from training instabilities (Figure 1) worse than those observed in standard densely-activated Transformers.

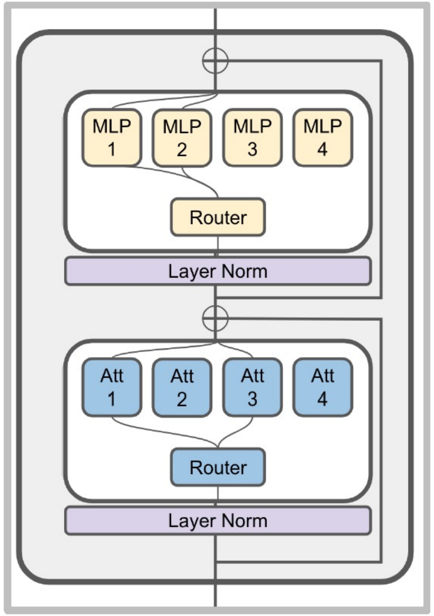
[Zoph et al 2022]

What MoEs generally look like ?

Typical: replace MLP with MoE layer



Less common: MoE for attention heads

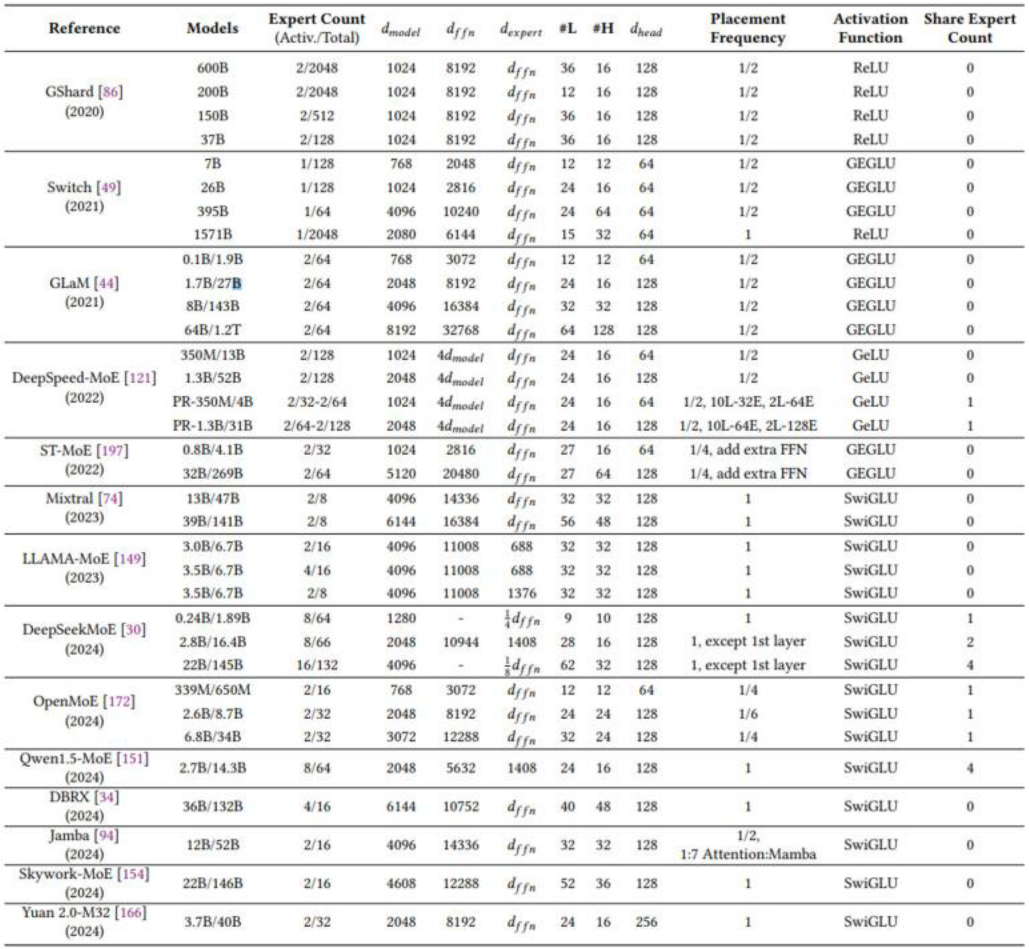


[ModuleFormer, JetMoE]

Key Design Variations of MoEs

- Most recent models place MoEs in every layer

- Some recent models apply Shared Experts



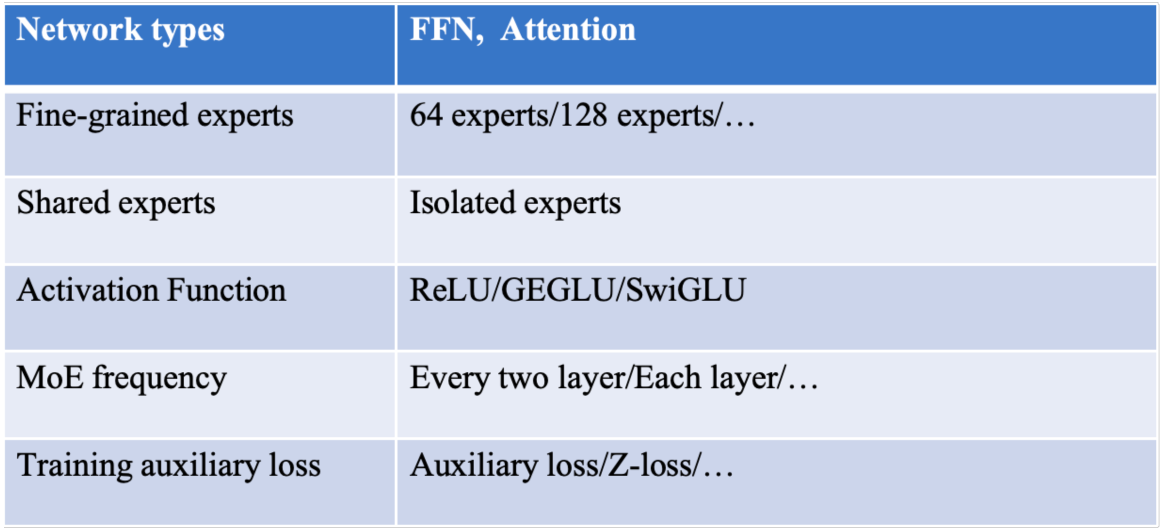
Reference	Models	Expert Count (Activ./Total)	d_{model}	d_{ffn}	d_{expert}	#L	#H	d_{head}	Placement Frequency	Activation Function	Share Expert Count
GShard [86] (2020)	600B	2/2048	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
	200B	2/2048	1024	8192	d_{ffn}	12	16	128	1/2	ReLU	0
	150B	2/512	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
	37B	2/128	1024	8192	d_{ffn}	36	16	128	1/2	ReLU	0
Switch [49] (2021)	7B	1/128	768	2048	d_{ffn}	12	12	64	1/2	GEGLU	0
	26B	1/128	1024	2816	d_{ffn}	24	16	64	1/2	GEGLU	0
	395B	1/64	4096	10240	d_{ffn}	24	64	64	1/2	GEGLU	0
	1571B	1/2048	2080	6144	d_{ffn}	15	32	64	1	ReLU	0
GLaM [44] (2021)	0.1B/1.9B	2/64	768	3072	d_{ffn}	12	12	64	1/2	GEGLU	0
	1.7B/27B	2/64	2048	8192	d_{ffn}	24	16	128	1/2	GEGLU	0
	8B/143B	2/64	4096	16384	d_{ffn}	32	32	128	1/2	GEGLU	0
	64B/1.2T	2/64	8192	32768	d_{ffn}	64	128	128	1/2	GEGLU	0
DeepSpeed-MoE [121] (2022)	350M/13B	2/128	1024	$4d_{model}$	d_{ffn}	24	16	64	1/2	GeLU	0
	1.3B/52B	2/128	2048	$4d_{model}$	d_{ffn}	24	16	128	1/2	GeLU	0
	PR-350M/4B	2/32-2/64	1024	$4d_{model}$	d_{ffn}	24	16	64	1/2, 10L-32E, 2L-64E	GeLU	1
	PR-1.3B/31B	2/64-2/128	2048	$4d_{model}$	d_{ffn}	24	16	128	1/2, 10L-64E, 2L-128E	GeLU	1
ST-MoE [197] (2022)	0.8B/4.1B	2/32	1024	2816	d_{ffn}	27	16	64	1/4, add extra FFN	GEGLU	0
	32B/269B	2/64	5120	20480	d_{ffn}	27	64	128	1/4, add extra FFN	GEGLU	0
Mixtral [74] (2023)	13B/47B	2/8	4096	14336	d_{ffn}	32	32	128	1	SwiGLU	0
	39B/141B	2/8	6144	16384	d_{ffn}	56	48	128	1	SwiGLU	0
LLAMA-MoE [149] (2023)	3.0B/6.7B	2/16	4096	11008	688	32	32	128	1	SwiGLU	0
	3.5B/6.7B	4/16	4096	11008	688	32	32	128	1	SwiGLU	0
	3.5B/6.7B	2/8	4096	11008	1376	32	32	128	1	SwiGLU	0
DeepSeekMoE [30] (2024)	0.24B/1.89B	8/64	1280	-	$\frac{1}{4}d_{ffn}$	9	10	128	1	SwiGLU	1
	2.8B/16.4B	8/66	2048	10944	1408	28	16	128	1, except 1st layer	SwiGLU	2
	22B/145B	16/132	4096	-	$\frac{1}{8}d_{ffn}$	62	32	128	1, except 1st layer	SwiGLU	4
OpenMoE [172] (2024)	339M/650M	2/16	768	3072	d_{ffn}	12	12	64	1/4	SwiGLU	1
	2.6B/8.7B	2/32	2048	8192	d_{ffn}	24	24	128	1/6	SwiGLU	1
	6.8B/34B	2/32	3072	12288	d_{ffn}	32	24	128	1/4	SwiGLU	1
Qwen1.5-MoE [151] (2024)	2.7B/14.3B	8/64	2048	5632	1408	24	16	128	1	SwiGLU	4
DBRX [34] (2024)	36B/132B	4/16	6144	10752	d_{ffn}	40	48	128	1	SwiGLU	0
Jamba [94] (2024)	12B/52B	2/16	4096	14336	d_{ffn}	32	32	128	1/2, 1:7 Attention:Mamba	SwiGLU	0
Skywork-MoE [154] (2024)	22B/146B	2/16	4608	12288	d_{ffn}	52	36	128	1	SwiGLU	0
Yuan 2.0-M32 [166] (2024)	3.7B/40B	2/32	2048	8192	d_{ffn}	24	16	256	1	SwiGLU	0

Key Design Variations of MoEs

- Routing Algorithm [to select which Expert(s)]

- Sizes of Experts

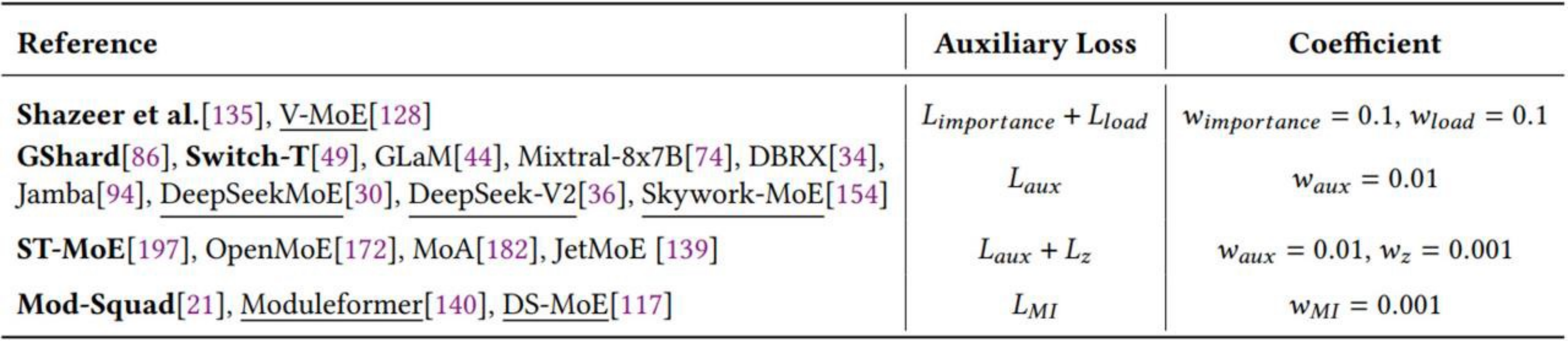
- Training Objectives



Network types	**FFN, Attention**
Fine-grained experts	64 experts/128 experts/…
Shared experts	Isolated experts
Activation Function	ReLU/GEGLU/SwiGLU
MoE frequency	Every two layer/Each layer/…
Training auxiliary loss	Auxiliary loss/Z-loss/…

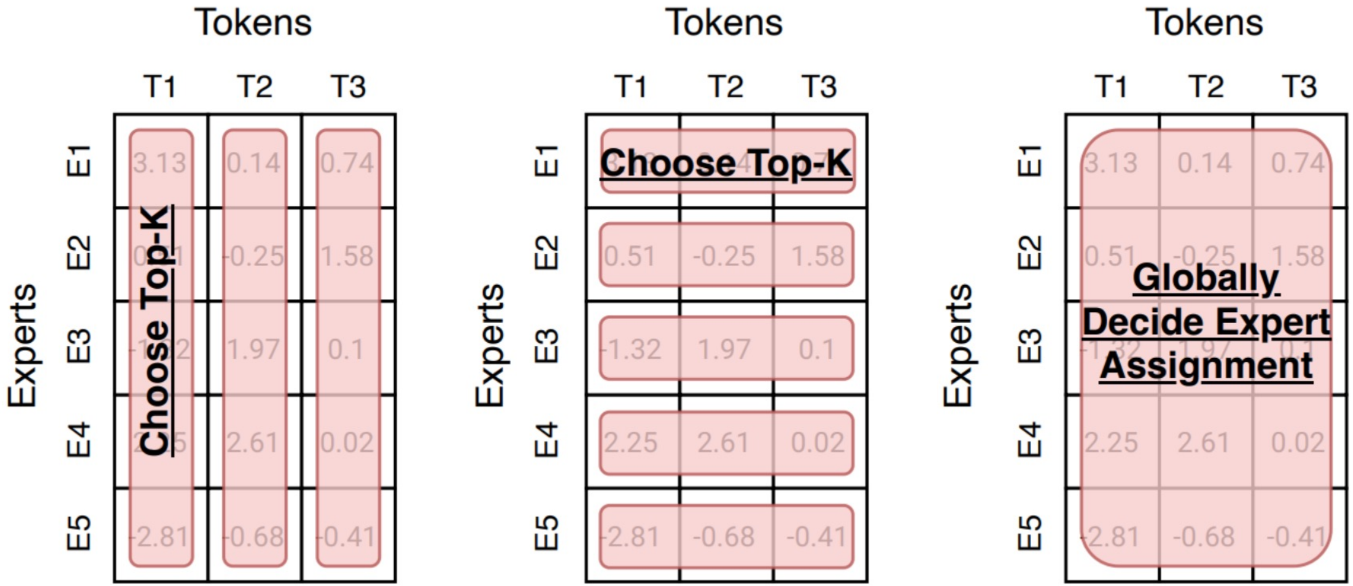
Training Objective

- Importance Loss: Encourage ALL experts to have Equal Importance
- Load Loss: Ensure Balanced Load across different Experts
- Auxiliary Loss: Mitigating Load Balance Losses
- Z-Loss: Improving Training Stability by Penalizing Large Logits
- MI-Loss: Mutual Information (MI) b/w experts and tasks to build task-expert alignment



Reference	Auxiliary Loss	Coefficient
Shazeer et al.[135], V-MoE[128]	$\mathcal{L}_{importance} + \mathcal{L}_{load}$	$w_{importance} = 0.1$, $w_{load} = 0.1$
GShard[86], **Switch-T**[49], GLaM[44], Mixtral-8x7B[74], DBRX[34], Jamba[94], DeepSeekMoE[30], DeepSeek-V2[36], Skywork-MoE[154]	$\mathcal{L}_{aux}$	$w_{aux} = 0.01$
ST-MoE[197], OpenMoE[172], MoA[182], JetMoE [139]	$\mathcal{L}_{aux} + \mathcal{L}_{z}$	$w_{aux} = 0.01$, $w_{z} = 0.001$
Mod-Squad[21], Moduleformer[140], DS-MoE[117]	$\mathcal{L}_{MI}$	$w_{MI} = 0.001$

Routing Algorithm (aka Gating) - Overview



	T1	T2	T3
E1	3.13	0.14	0.74
E2	0.51	-0.25	1.58
E3	-1.32	1.97	0.1
E4	2.25	2.61	0.02
E5	-2.81	-0.68	-0.41

Token chooses expert

Expert chooses token

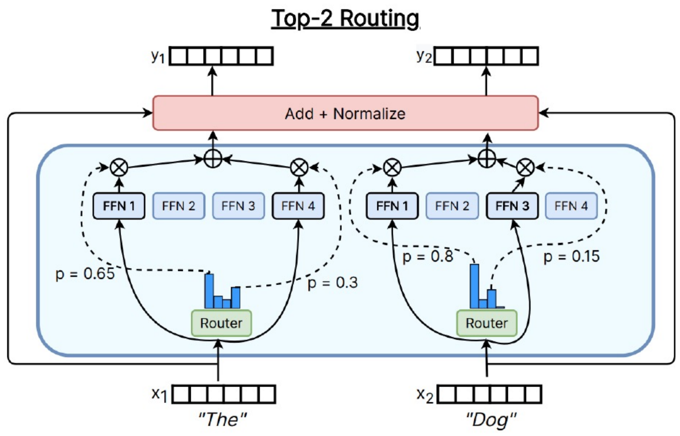
Global routing via optimization

[Fedus et al 2022]

- Many of the Routing Algorithms boil down to “Choose Top K”

Common Routing Variants in Detail

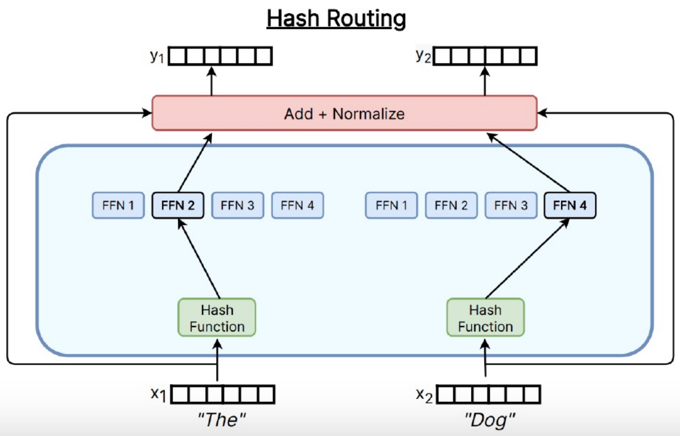
Top-k



Used in *most* MoEs

Switch Transformer (k=1)
Gshard (k=2), Grok (2), Mixtral (2),
Qwen (4), DBRX (4),
DeepSeek (7)

Hashing

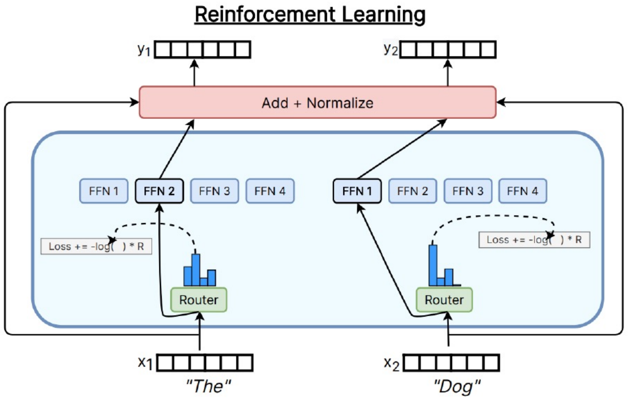


Common baseline

[Fedus et al 2022]

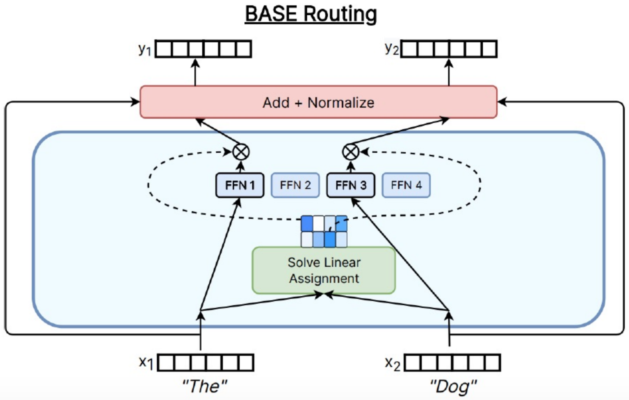
Other Routing Algorithms

RL to learn routes



Used in some of the earliest work
Bengio 2013, not common now

Solve a matching problem



Linear assignment for routing
Used in various papers like Clark '22

[Fedus et al 2022]

Top-K Routing in Detail

Gating

t-th token in layer l

$$\mathbf{h}_t^l = \sum_{i=1}^{N} \left(g_{i,t}\, \mathrm{FFN}_i \left(\mathbf{u}_t^l \right) \right) + \mathbf{u}_t^l,$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \leqslant j \leqslant N\}, K), \\ 0, & \text{otherwise}, \end{cases}$$

$$s_{i,t} = \mathrm{Softmax}_i \left({\mathbf{u}_t^l}^T \mathbf{e}_i^l \right),$$

the centroid of the i-th expert in layer l

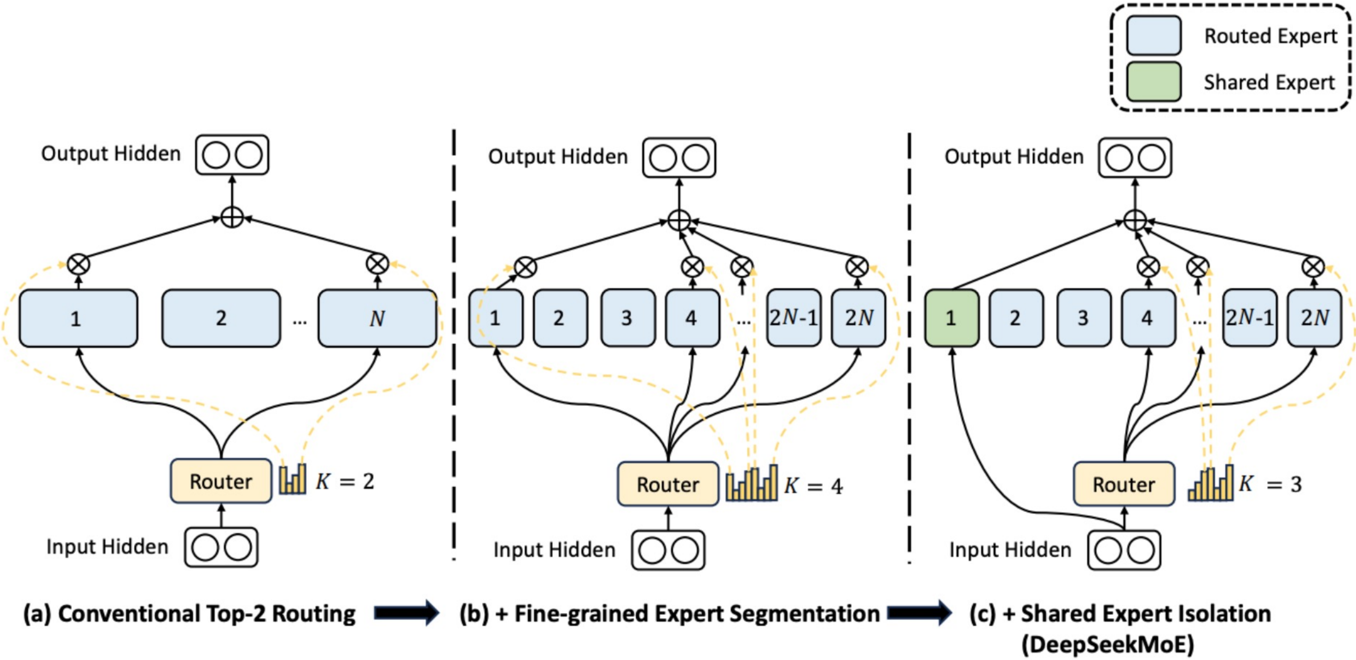
Gates selected by a logistic regressor

This is the DeepSeek router (Grok does this too)

Mixtral and DBRX softmaxes after the TopK

[Dai et al 2024]

Recent variations in DeepSeek and Qwen

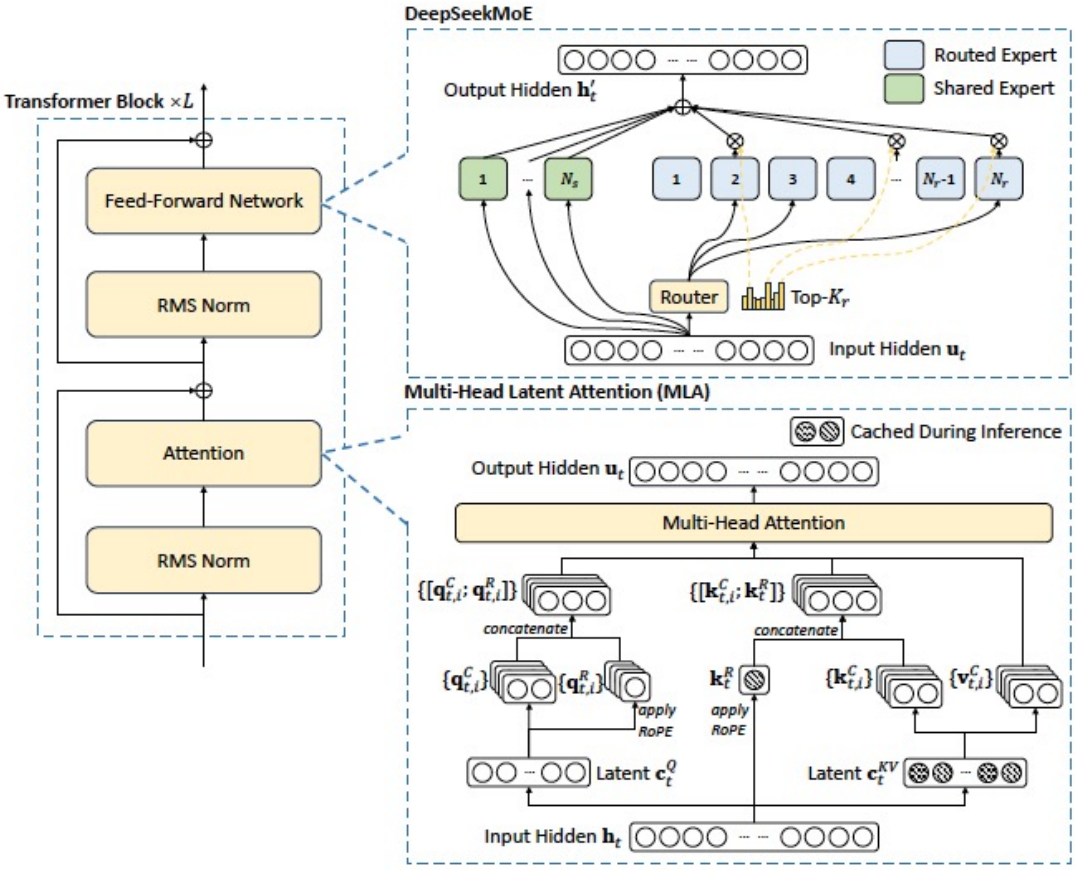


(a) Conventional Top-2 Routing → **(b) + Fine-grained Expert Segmentation** → **(c) + Shared Expert Isolation (DeepSeekMoE)**

Smaller, larger number of experts + a few shared experts that are always on.

Figure 2 | Illustration of DeepSeekMoE. Subfigure (a) showcases an MoE layer with the conventional top-2 routing strategy. Subfigure (b) illustrates the fine-grained expert segmentation strategy. Subsequently, subfigure (c) demonstrates the integration of the shared expert isolation strategy, constituting the complete DeepSeekMoE architecture. It is noteworthy that across these three architectures, the number of expert parameters and computational costs remain constant.

DeepSeek v2 Architecture



Source: DeepSeek v2 Technical Report

Ablation Study from the DeepSeekMoE paper

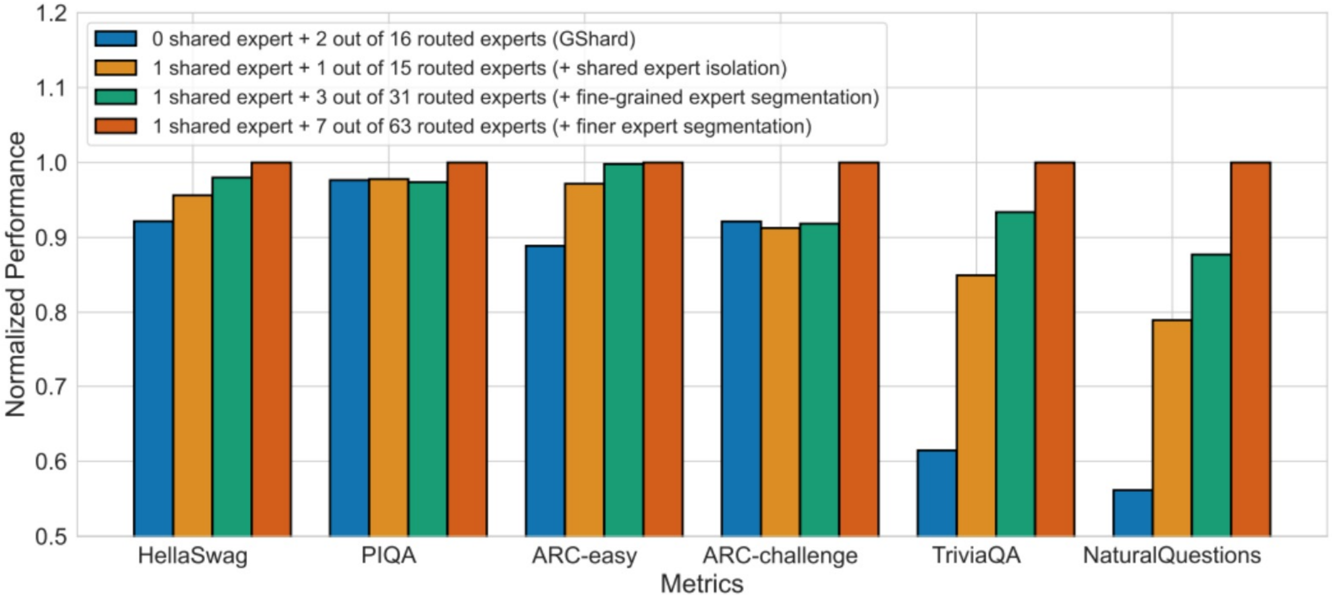


Figure 3 | Ablation studies for DeepSeekMoE. The performance is normalized by the best performance for clarity in presentation. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute to stronger overall performance.

More experts, shared experts all seem to generally help

How to Train MoEs ?

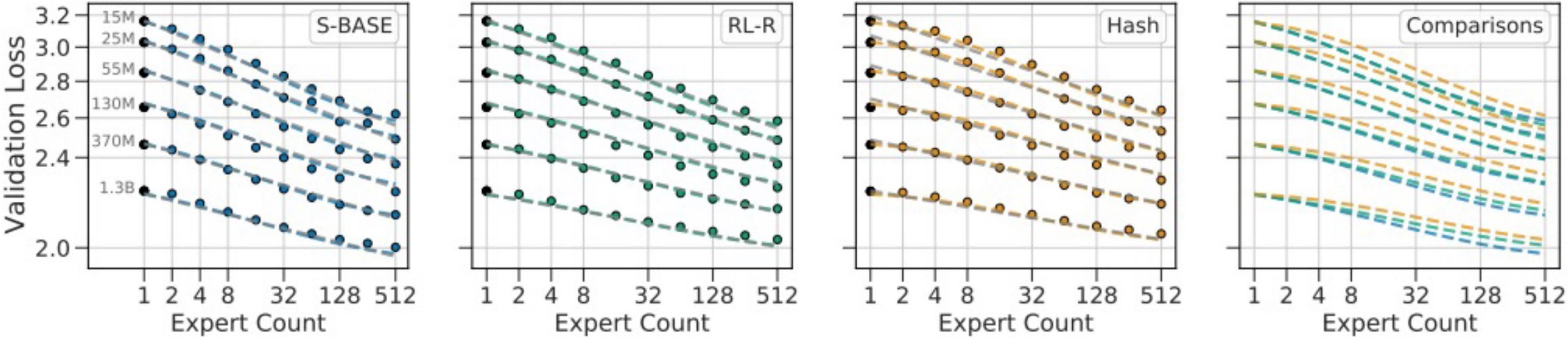
Major Challenge: Need Sparsity for Training-time Efficiency BUT Sparse Gating Decisions are not Differentiable !

Solutions ?

1. Reinforcement Learning to optimize Gating Policies.
2. Stochastic Perturbations
3. Heuristic “Balancing” Losses

RL for MoEs

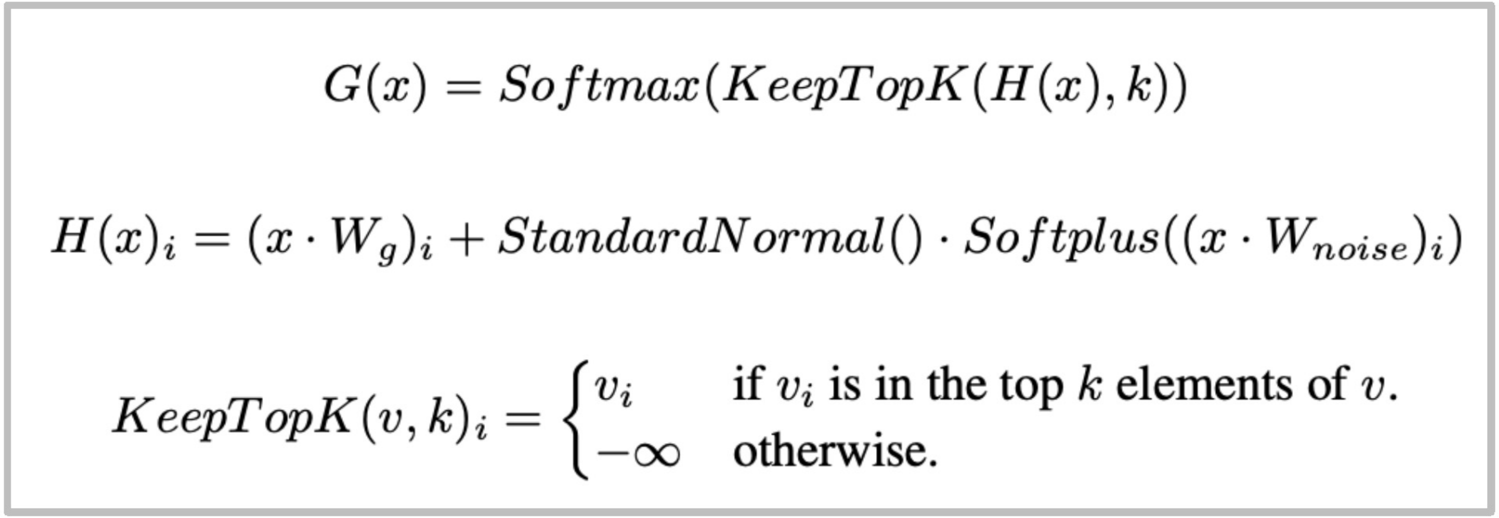
RL via REINFORCE does work, but not so much better that it's a clear win



(REINFORCE baseline approach, Clark et al 2020)

RL is the 'right solution' but gradient variances and complexity means it's not widely used

Stochastic Approximations



$$G(x) = Softmax(KeepTopK(H(x), k))$$

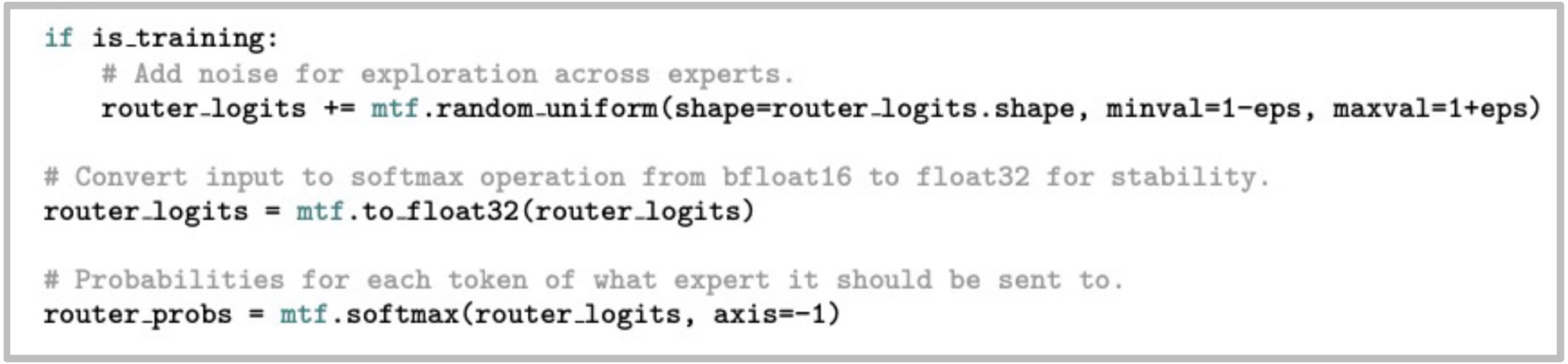
$$H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus((x \cdot W_{noise})_i)$$

$$KeepTopK(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

From Shazeer et al 2017 – routing decisions are *stochastic* with gaussian perturbations.

1. This naturally leads to experts that are a bit more robust.
2. The softmax means that the model learns how to rank K experts

Stochastic Approximations

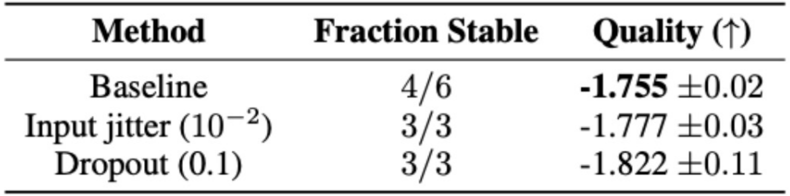


```
if is_training:
    # Add noise for exploration across experts.
    router_logits += mtf.random_uniform(shape=router_logits.shape, minval=1-eps, maxval=1+eps)

# Convert input to softmax operation from bfloat16 to float32 for stability.
router_logits = mtf.to_float32(router_logits)

# Probabilities for each token of what expert it should be sent to.
router_probs = mtf.softmax(router_logits, axis=-1)
```

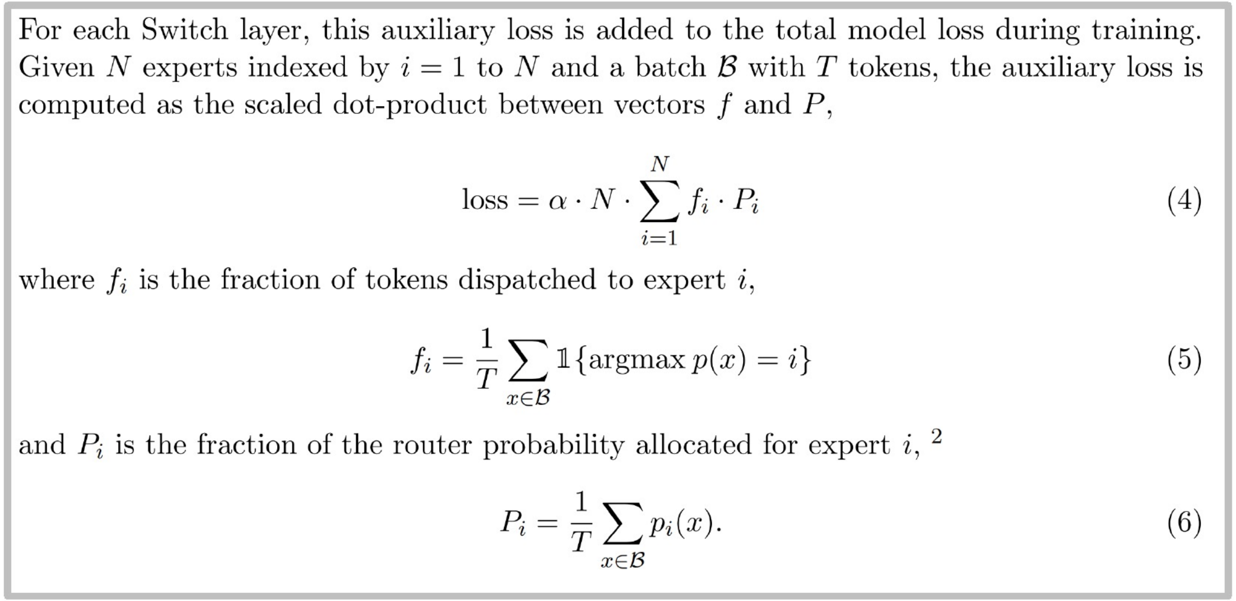
Stochastic jitter in Fedus et al 2022. This does a uniform multiplicative perturbation for the same goal of getting less brittle experts. This was later removed in Zoph et al 2022



Method	**Fraction Stable**	**Quality (↑)**
Baseline	4/6	**-1.755** ±0.02
Input jitter (10^{-2})	3/3	-1.777 ±0.03
Dropout (0.1)	3/3	-1.822 ±0.11

Heuristics Balancing Losses

Another key issue – systems efficiency requires that we use experts evenly..



For each Switch layer, this auxiliary loss is added to the total model loss during training. Given N experts indexed by $i = 1$ to N and a batch $\mathcal{B}$ with T tokens, the auxiliary loss is computed as the scaled dot-product between vectors f and P,

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i \tag{4}$$

where f_i is the fraction of tokens dispatched to expert i,

$$f_i = \frac{1}{T} \sum_{x \in \mathcal{B}} \mathbb{1}\{\operatorname{argmax} p(x) = i\} \tag{5}$$

and P_i is the fraction of the router probability allocated for expert i, [2]

$$P_i = \frac{1}{T} \sum_{x \in \mathcal{B}} p_i(x). \tag{6}$$

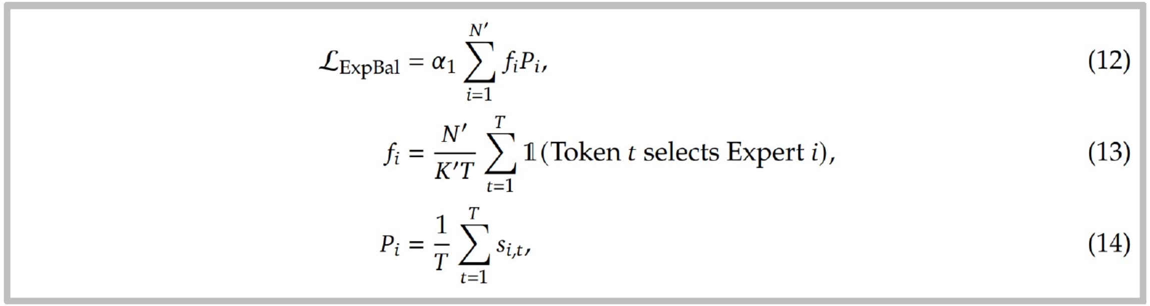
From the Switch Transformer [Fedus et al 2022]

The derivative with respect to $p_i(x)$ is $\frac{\alpha N}{T^2} \sum 1_{argmax\, p(x)=i}$,

so more frequent use = stronger downweighting

Example from DeepSeek

Per-expert balancing – same as the switch transformer

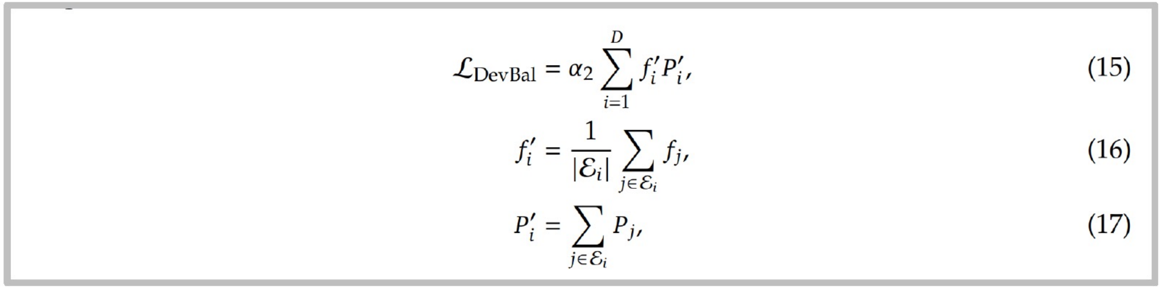


$$\mathcal{L}_{\text{ExpBal}} = \alpha_1 \sum_{i=1}^{N'} f_i P_i, \tag{12}$$

$$f_i = \frac{N'}{K'T} \sum_{t=1}^{T} \mathbb{1}(\text{Token } t \text{ selects Expert } i), \tag{13}$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s_{i,t}, \tag{14}$$

Per-device balancing – the objective above, but aggregated by device.



$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^{D} f_i' P_i', \tag{15}$$

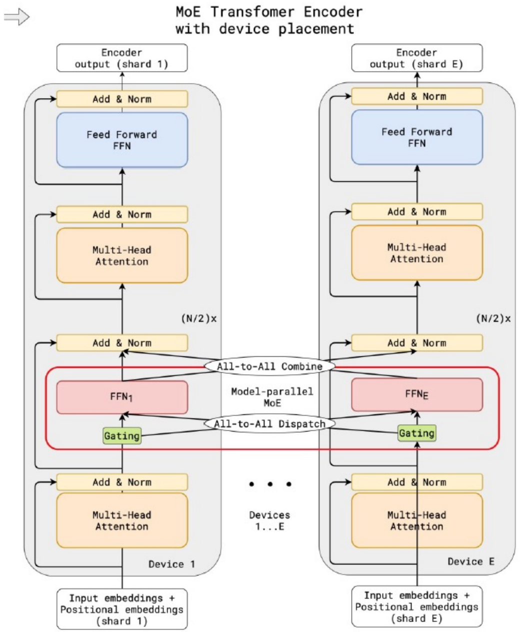
$$f_i' = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j, \tag{16}$$

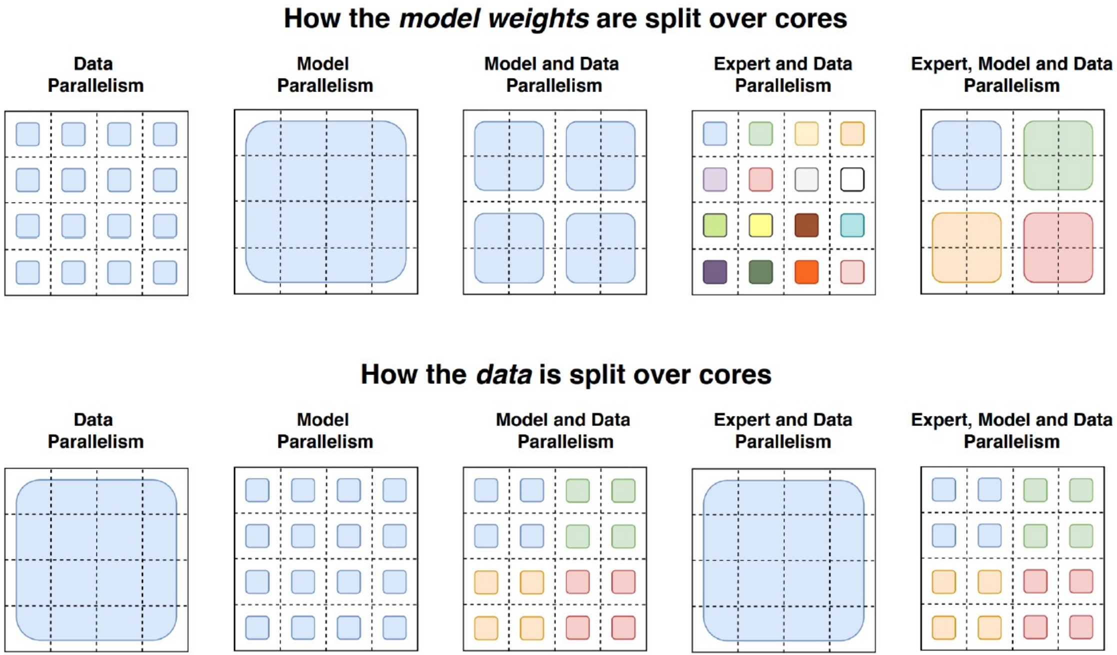
$$P_i' = \sum_{j \in \mathcal{E}_i} P_j, \tag{17}$$

Training MoE – the System Architecture

MoEs parallelize nicely – Each FFN can fit in a device

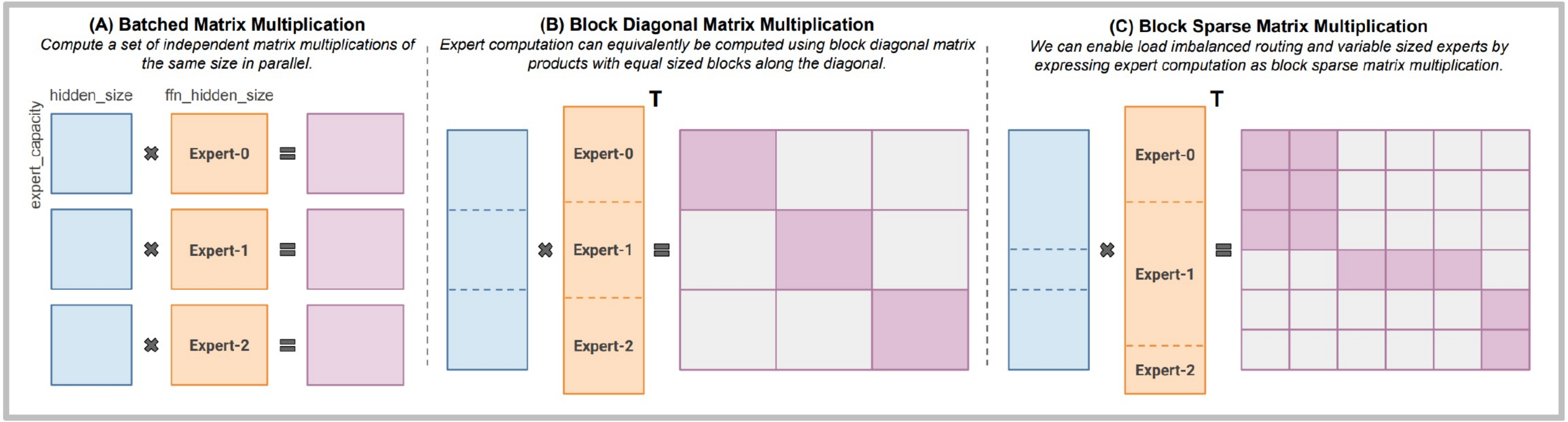
Enables additional kinds of parallelism





Training MoE – the System Architecture

MoE routing allows for parallelism, but also some complexities


(A) Batched Matrix Multiplication

Compute a set of independent matrix multiplications of the same size in parallel.

(B) Block Diagonal Matrix Multiplication

Expert computation can equivalently be computed using block diagonal matrix products with equal sized blocks along the diagonal.

(C) Block Sparse Matrix Multiplication

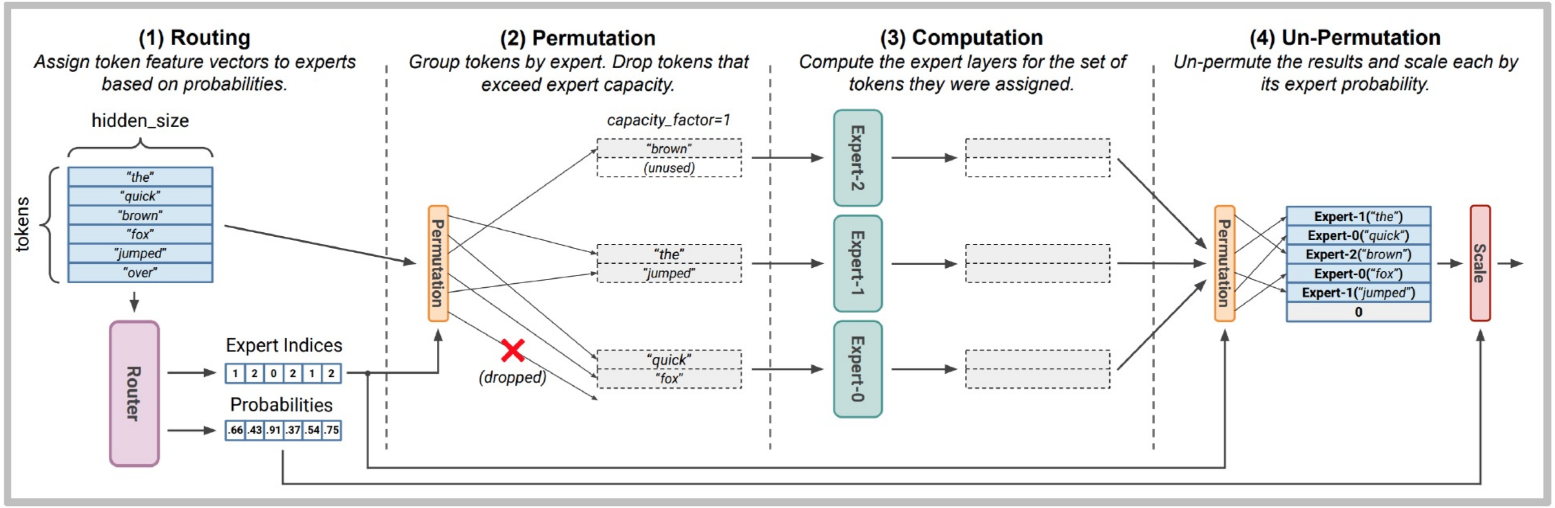
We can enable load imbalanced routing and variable sized experts by expressing expert computation as block sparse matrix multiplication.

Modern libraries like MegaBlocks (used in many open MoEs) use smarter sparse MMs

Additional Randomness from MoE models

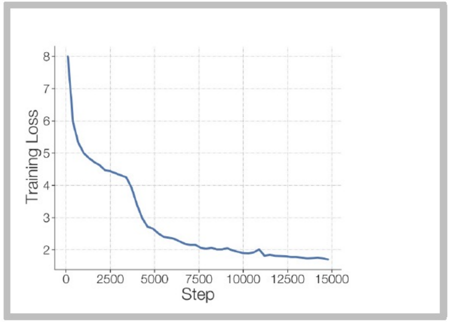
There was speculation that GPT-4's stochasticity was due to MoE..

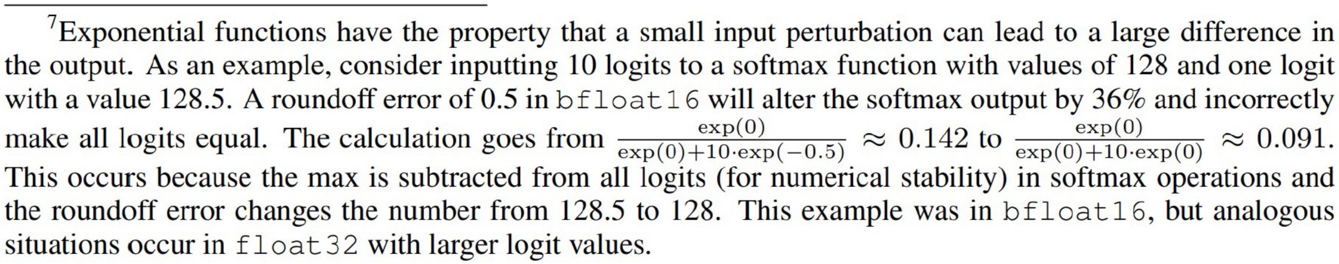
Why would a MoE have additional randomness?



Token dropping from routing happens at a *batch* level – this means that other people's queries can drop your token!

Issues with MoEs – Training Stability





---

[7] Exponential functions have the property that a small input perturbation can lead to a large difference in the output. As an example, consider inputting 10 logits to a softmax function with values of 128 and one logit with a value 128.5. A roundoff error of 0.5 in `bfloat16` will alter the softmax output by 36% and incorrectly make all logits equal. The calculation goes from $\frac{\exp(0)}{\exp(0)+10\cdot\exp(-0.5)} \approx 0.142$ to $\frac{\exp(0)}{\exp(0)+10\cdot\exp(0)} \approx 0.091$. This occurs because the max is subtracted from all logits (for numerical stability) in softmax operations and the roundoff error changes the number from 128.5 to 128. This example was in `bfloat16`, but analogous situations occur in `float32` with larger logit values.

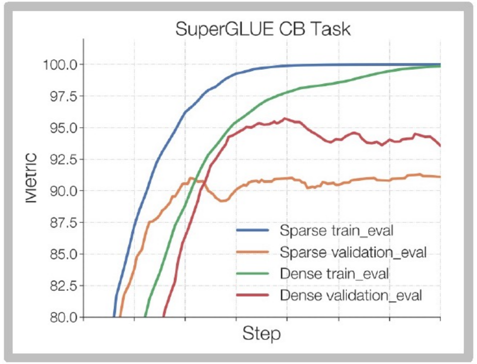
[Zoph 2022]

Solution: Use Float 32 just for the expert router (sometimes with an aux loss)

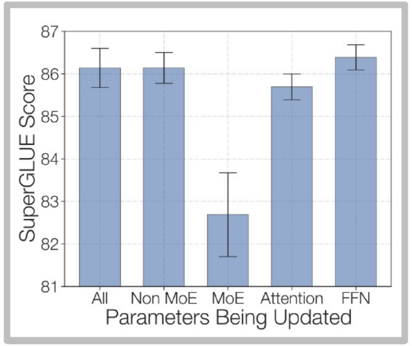
$$L_z(x) = \frac{1}{B}\sum_{i=1}^{B}\left(\log\sum_{j=1}^{N} e^{x_j^{(i)}}\right)^2 \qquad (5)$$

Issues with MoEs – Fine-Tuning

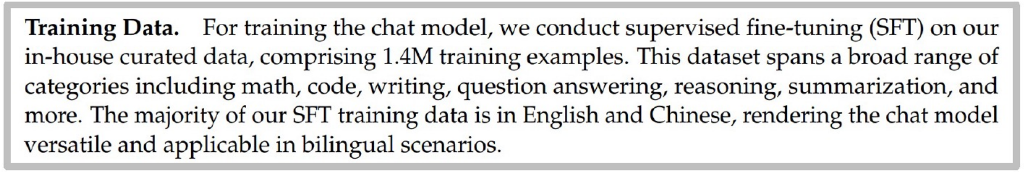
Sparse MoEs can overfit on smaller fine-tuning data



Zoph et al solution – finetune non-MoE MLPs

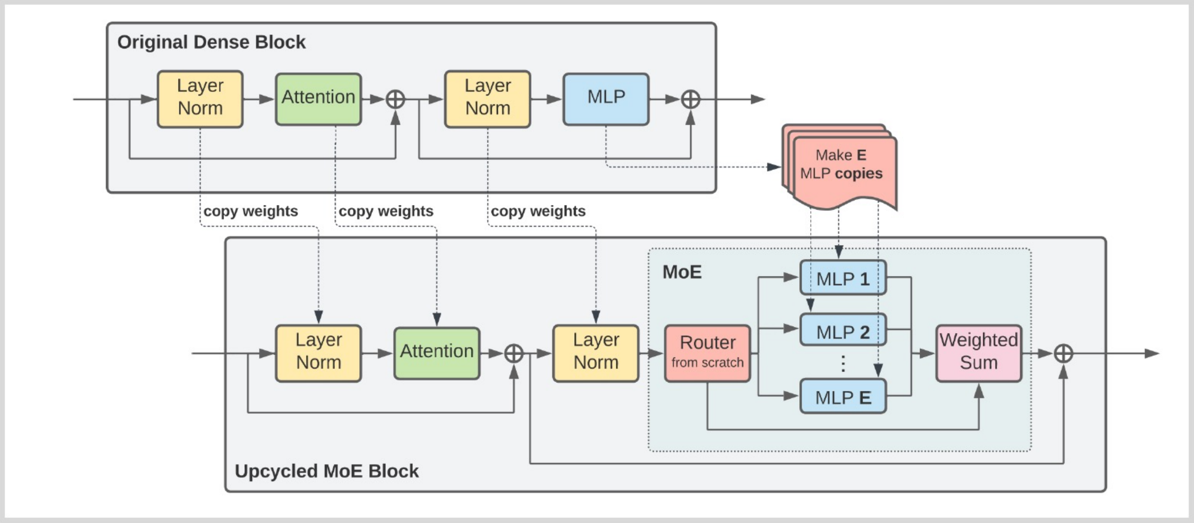


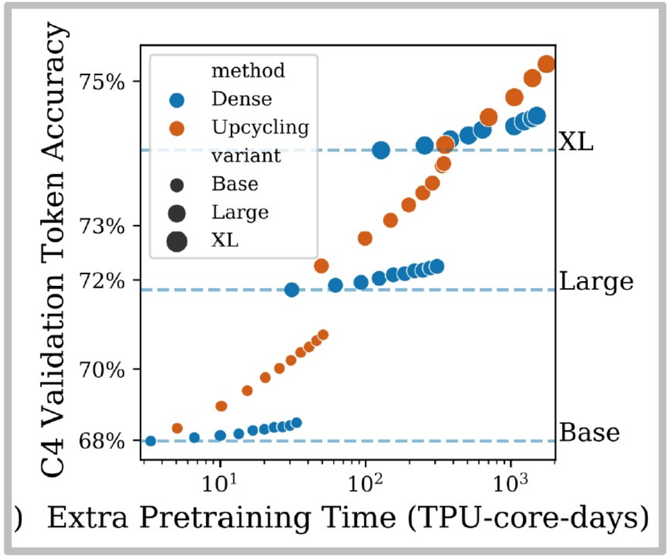
DeepSeek solution – use lots of data 1.4M SFT



Training Data. For training the chat model, we conduct supervised fine-tuning (SFT) on our in-house curated data, comprising 1.4M training examples. This dataset spans a broad range of categories including math, code, writing, question answering, reasoning, summarization, and more. The majority of our SFT training data is in English and Chinese, rendering the chat model versatile and applicable in bilingual scenarios.

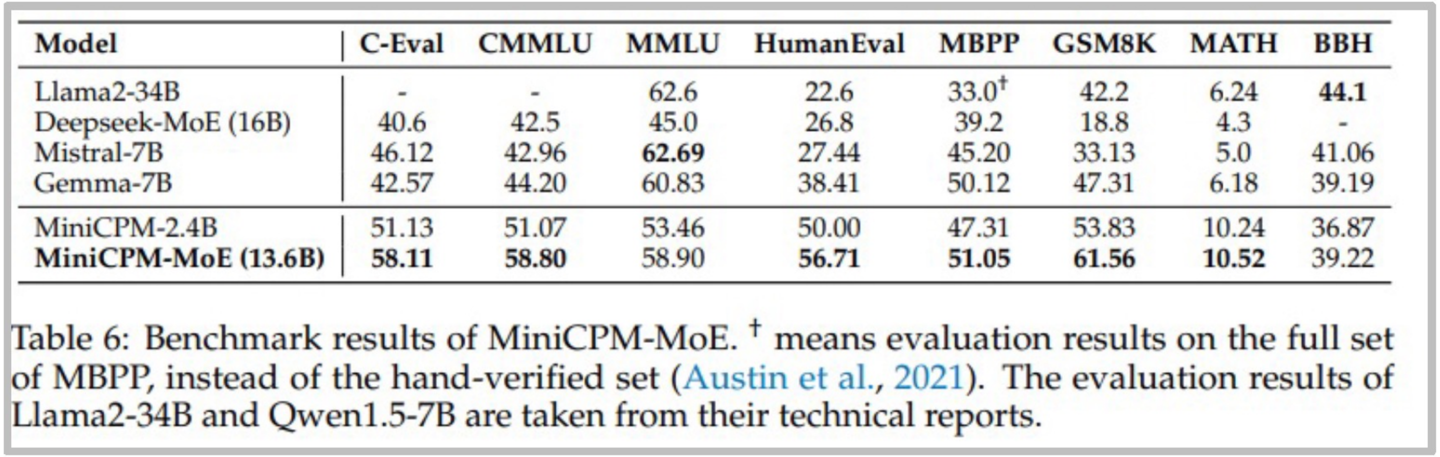
Additional Training Method for MoEs - Upcycling





Upcycling example - MiniCPM

Uses the MiniCPM model (topk=2, 8 experts, ~ 4B active params).



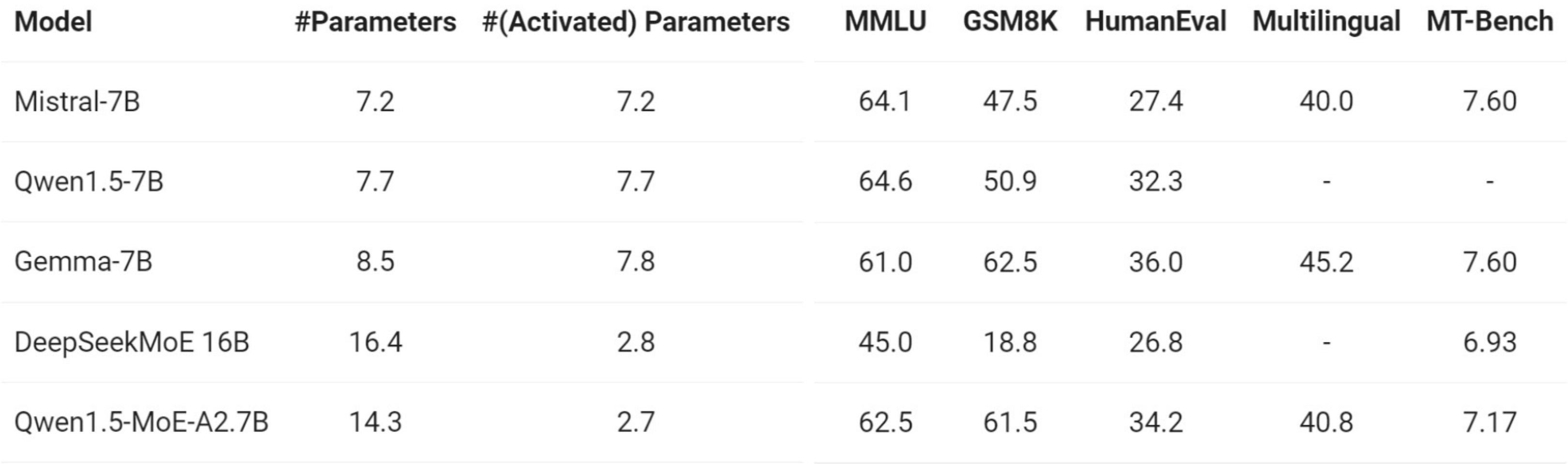
Model	**C-Eval**	**CMMLU**	**MMLU**	**HumanEval**	**MBPP**	**GSM8K**	**MATH**	**BBH**
Llama2-34B	-	-	62.6	22.6	$33.0^{\dagger}$	42.2	6.24	**44.1**
Deepseek-MoE (16B)	40.6	42.5	45.0	26.8	39.2	18.8	4.3	-
Mistral-7B	46.12	42.96	**62.69**	27.44	45.20	33.13	5.0	41.06
Gemma-7B	42.57	44.20	60.83	38.41	50.12	47.31	6.18	39.19
MiniCPM-2.4B	51.13	51.07	53.46	50.00	47.31	53.83	10.24	36.87
MiniCPM-MoE (13.6B)	**58.11**	**58.80**	58.90	**56.71**	**51.05**	**61.56**	**10.52**	39.22

Table 6: Benchmark results of MiniCPM-MoE. † means evaluation results on the full set of MBPP, instead of the hand-verified set (Austin et al., 2021). The evaluation results of Llama2-34B and Qwen1.5-7B are taken from their technical reports.

Simple MoE, shows gains from the base model with ~ 520B tokens for training

Another Upcycling example – Qwen MoE

Qwen MoE – Initialized from the Qwen 1.8B model top-k=4, 60 experts w/ 4 shared.



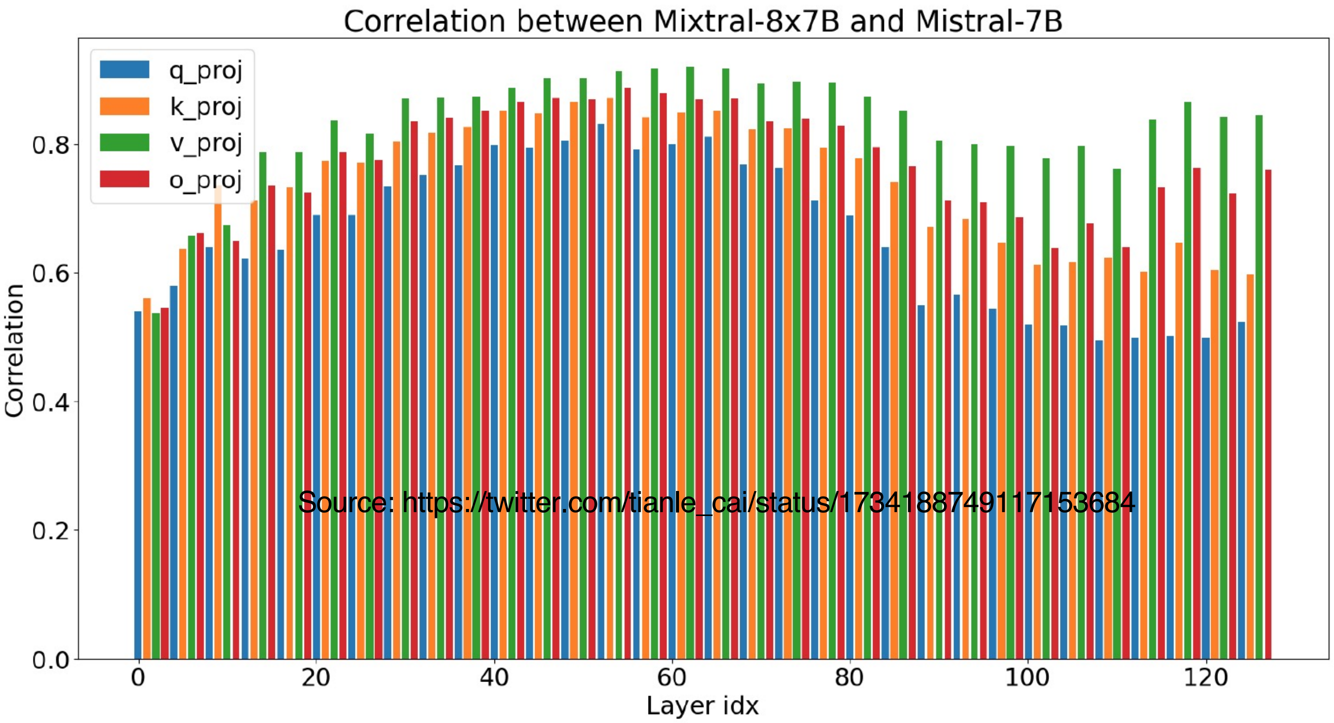
Model	#Parameters	#(Activated) Parameters	MMLU	GSM8K	HumanEval	Multilingual	MT-Bench
Mistral-7B	7.2	7.2	64.1	47.5	27.4	40.0	7.60
Qwen1.5-7B	7.7	7.7	64.6	50.9	32.3	-	-
Gemma-7B	8.5	7.8	61.0	62.5	36.0	45.2	7.60
DeepSeekMoE 16B	16.4	2.8	45.0	18.8	26.8	-	6.93
Qwen1.5-MoE-A2.7B	14.3	2.7	62.5	61.5	34.2	40.8	7.17

Similar architecture / setup to DeepSeekMoE, but one of the first (confirmed) upcycling successes

A remarkable Reduction of 75% in Training resource !

Upcycling example (?) Mixtral

Some people think Mixtral may also be upcycled



Source: https://twitter.com/tianle_cai/status/1734188749117153684

… but since Mixtral is only open weights and not open training code, we don't really know..

MoE Summary

- MoE take advantage of Sparsity – Not all inputs need the Full model
- Discrete Routing is Hard, but Top-K Heuristics seem to work
- Lots of Empirical Evidence has shown MoEs work, and are cost-effective

Quantization

Preliminaries: How Are Numbers Represented in Computers?

Different data types are represented by different numbers of bits:

- Char: 8 bits, 1 byte
- Int: 32 bits, 4 bytes

Common data types for training in machine learning:

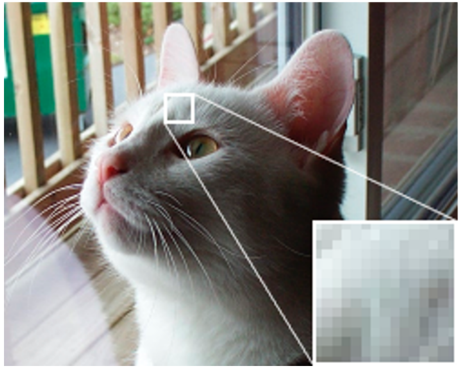
- Floating point 32: 4 bytes
- BFloat16: 2 bytes

We do not represent an *infinite set of* values!

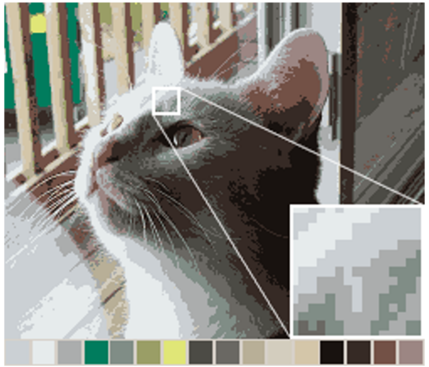
Data is organized by bytes and programs refer to data by address:

What is Quantization?

Quantization is the process of mapping a continuous signal (or a signal where values come from a large set) to a discretized signal (or one where values come from a smaller set)



Quantized to 24-bit colors



Quantized to 16-bit colors

Quantization error is the absolute value of the true value (e.g. in the continuous signal) minus the quantized value

https://en.wikipedia.org/wiki/Color_quantization

Quantization Motivation

Reduce Storage Required

Total Storage = Number of Parameters x Bit-width per Parameter

Reduce the FLOPs/Energy Required

For Addition: $O(N)$ FLOPs for N-bit-width values

For Multiplication: $O(N^2)$ FLOPs for N-bit-width values

N-bit Integer

Unsigned Integer: $[0, 2^n - 1]$

8-bits	0	1	0	0	0	1	1	0	
	x	x	x	x	x	x	x	x	
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	= 70

Signed Integer: We designate a bit for negative (1) or positive (0)

8-bits	1	1	0	0	0	1	1	0	
		x	x	x	x	x	x	x	
	-1	2^6	2^5	2^4	2^3	2^2	2^1	2^0	= -70

"Signed magnitude representation"

N-bit Integer

Unsigned Integer: $[0, 2^n - 1]$

8-bits	0	1	0	0	0	1	1	0	
	x	x	x	x	x	x	x	x	
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	= 70

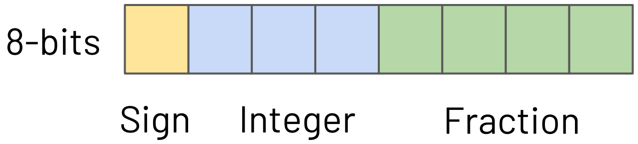
Signed Integer: We designate a bit for negative (1) or positive (0)

8-bits	1	0	1	1	1	0	1	0	
		x	x	x	x	x	x	x	
	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	= -70

"Two's complement representation"

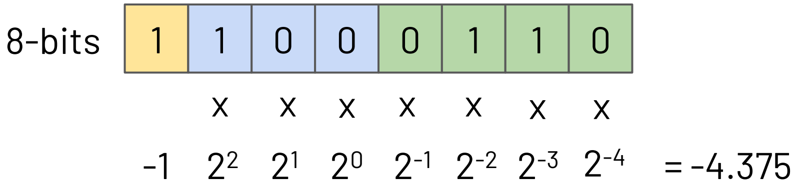
Fixed-Point Numbers

We have a fixed number of bits allocated to the integer vs. fraction representation.



8-bits

Sign Integer Fraction



8-bits

1	1	0	0	0	1	1	0	
	x	x	x	x	x	x	x	
-1	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	= -4.375

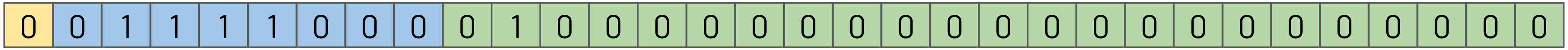
Floating-Point Numbers

Sign 8-bit Exponent 23-bit Fraction (“Mantissa”)

Value = $(-1)^{sign} \times (1 + Fraction) \times 2^{Exponent - Bias}$

- Computing $2^{Exponent}$ rather than the linear Exponent, allows increasing the range of values we can represent!
    - **Dynamic Range:** difference between the maximum and minimum value we can represent
- Bias is $2^{8-1}-1$. Subtracting the bias allows us to represent negative values in the exponent.

Example of FP32



0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

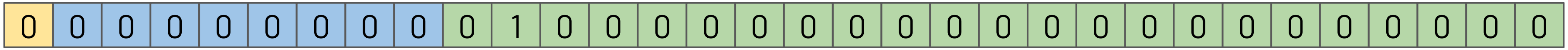
Sign 8-bit Exponent 23-bit Fraction ("Mantissa")

- Exponent: $2^6 + 2^5 + 2^4 + 2^3 = 120$
- Bias: $2^{8-1} - 1 = 127$
- Fraction: $2^{-2} = 0.25$

$$\text{Value} = (-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent - Bias}}$$

$$\text{Value} = (-1)^{0} \times (1 + 0.25) \times 2^{120-127} = 0.0097656..$$

Edge Cases of FP32



0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

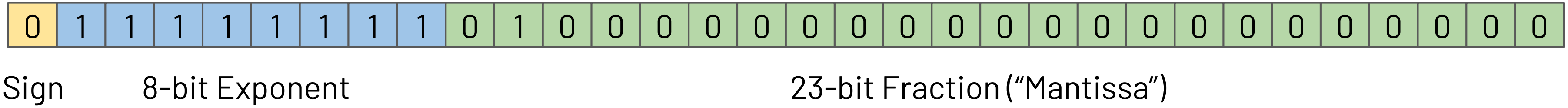
Sign 8-bit Exponent 23-bit Fraction ("Mantissa")

- Normal numbers: exponent is not all 0
  - This is what we looked at on the prior slides

- Subnormal numbers: exponent is all 0
  - Equation is set to: value = $(-1)^{sign}$ x Fraction x 2^{1-127}
  - **Value is 0** If exponent is all 0 and the fraction is 0
  - **Smallest positive subnormal value** we can represent = 2^{-23} x 2^{1-127}
  - **Largest positive subnormal value** we can represent = $(1 - 2^{-23})$ x 2^{1-127}

We cannot represent *infinitely small* values!

Edge Cases of FP32

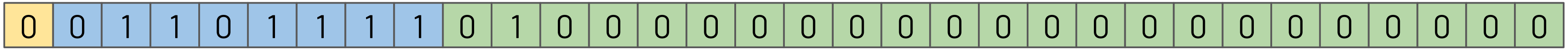


0	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign 8-bit Exponent 23-bit Fraction (“Mantissa”)

- Exponent is all 1
  - Non-0 fraction is NaN (Not a Number)
  - 0 fraction is + infinity or - infinity depending on the sign bit

Floating-Point Number (IEEE 754 Specification)



0	0	1	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign 8-bit Exponent 23-bit Fraction (“Mantissa”)

Exponent	Fraction=0	Fraction≠0	Equation
$00_H = 0$	±0	subnormal	$(-1)^{sign} \times \textbf{Fraction} \times 2^{1-127}$
$01_H \ldots FE_H = 1 \ldots 254$	normal		$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$
$FF_H = 255$	±INF	NaN	

Credits: MIT 6.5940 (Prof. Song Han)

Other Floating-Point Representations

Recall that the storage size and computational cost for our model grows with the number of bits per value. Can we use fewer bits to train or run inference?

	Exponent bits	Fraction bits	Total
IEEE 754 FP32	8	23	32
IEEE 754 FP16	5	10	16
Google Brain Float 16	8	7	16

FP16 ⟶ BF16: Goal is to increase the dynamic range. More stable training in practice.

Two Classical Quantization Methods

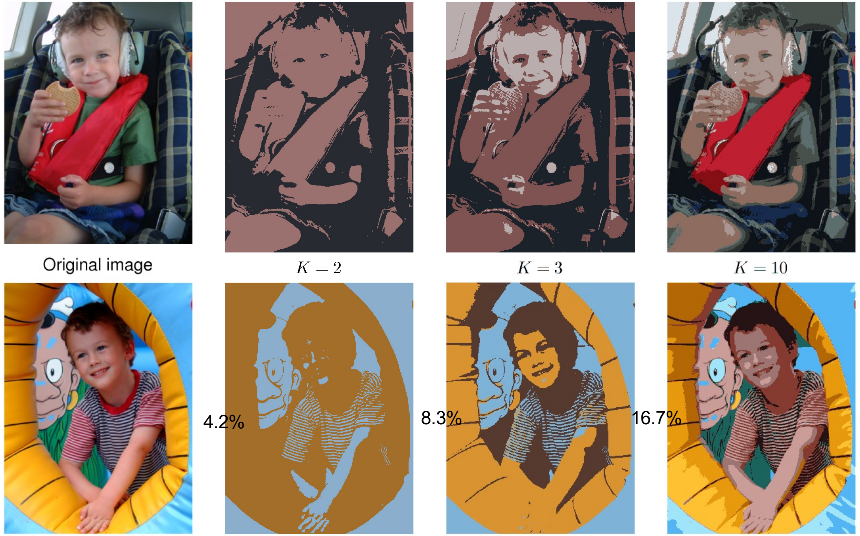
Original training: floating point weights and floating point computations

- K-means quantization: integer weights and floating point computations

- Linear quantization: integer weights and integer arithmetic

Examples of k-means clustering

- Clustering RGB vectors of pixels in images
- Compression of image file: N x 24 bits
    - Store RGB values of cluster centers: K x 24 bits
    - Store cluster index of each pixel: N x log K bits



Original image $K = 2$ $K = 3$ $K = 10$

Side Notes:

$O(kn^2)$ 1-D k-means algorithm via Dynamic Programming: https://pmc.ncbi.nlm.nih.gov/articles/PMC5148156/

k-means is NP-hard: https://cseweb.ucsd.edu/~avattani/papers/kmeans_hardness.pdf

k-means is NP-hard even for k=2: https://cseweb.ucsd.edu/~dasgupta/papers/kmeans.pdf

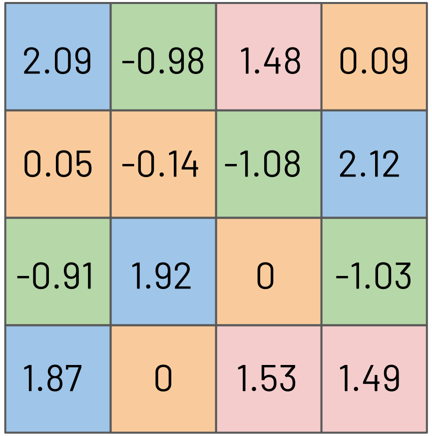
Method 1: K-Means Quantization

Approach during inference

1. Given our original full precision (e.g. FP32) weight matrix, we cluster the values

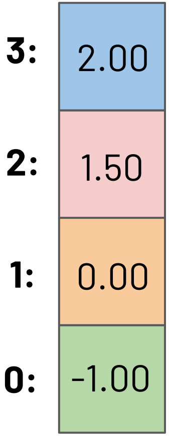
2. We store a index of the cluster number (integer) to the cluster centroid (floating point)

Original FP32 Weights



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Centroids

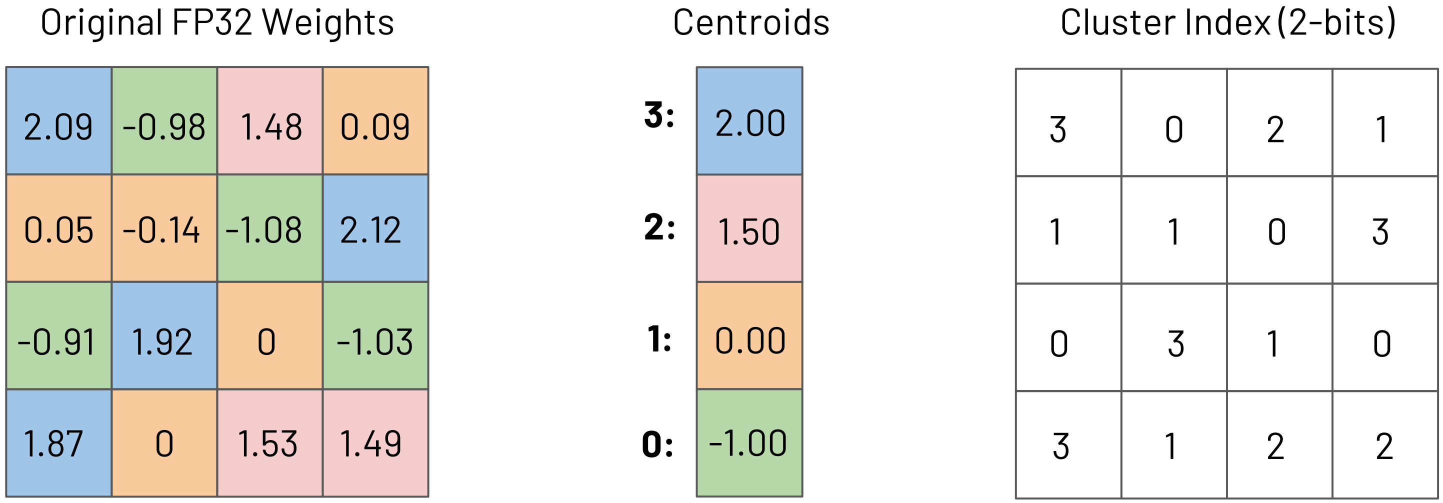


3:	2.00
2:	1.50
1:	0.00
0:	-1.00

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Method 1: K-Means Quantization

3. We perform computations by reconstructing the weight matrix, using the map



Original FP32 Weights

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

Cluster Index (2-bits)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Method 1: K-Means Quantization

Analysis.

Let S be the size of the index. Let the original DxD weight matrix be in FP32.

Storage savings (reduced memory fetch):

$$(32 \times D^2) - (\log_2(S) \times D^2 + S \times 32)$$

At D = 4, the savings are 64B - 20B = 44B (**3.2x compression**)

As S << D, we see larger savings

Note that there are no computation savings (computation is still FP32)

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Method 1: K-Means Quantization

Approach during fine-tuning/training the shared weights (centroid vector). We have the same Sx1 dimensional centroids vector as before.

1. We compute gradients with respect to the loss function for the centroids vector

$$dL / dC_k = \text{SUM}_{i,j}\, dL/dW_{ij}, \text{ for } W_{i,j} \text{ in centroid } C_k$$

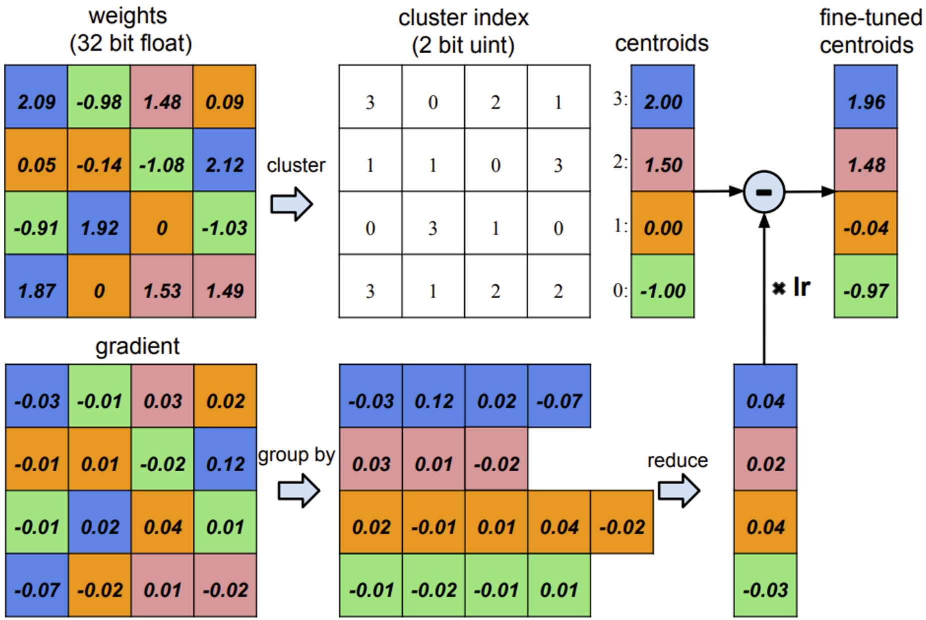
1. We use the gradient vector for the centroids to update the centroid vector

$$C^{(2)} = C^{(1)} - \text{lr} \times (dL/dC^{(1)})$$

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Method 1: K-Means Quantization

Summarizing the overall procedure:



weights (32 bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

cluster index (2 bit uint)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

	centroids	fine-tuned centroids
3:	2.00	1.96
2:	1.50	1.48
1:	0.00	-0.04
0:	-1.00	-0.97

gradient

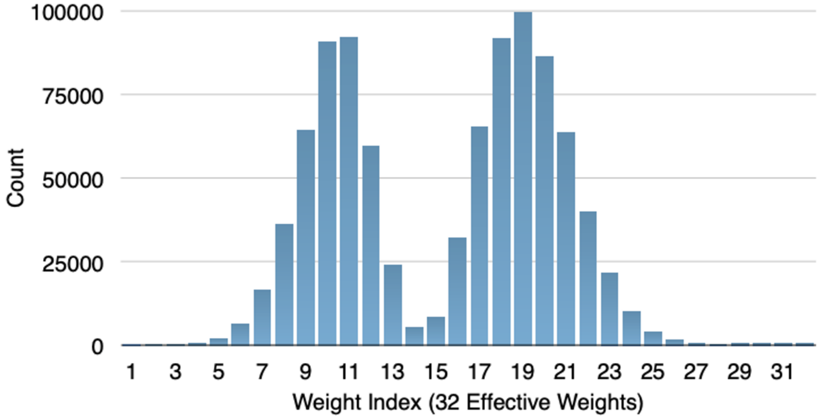
-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

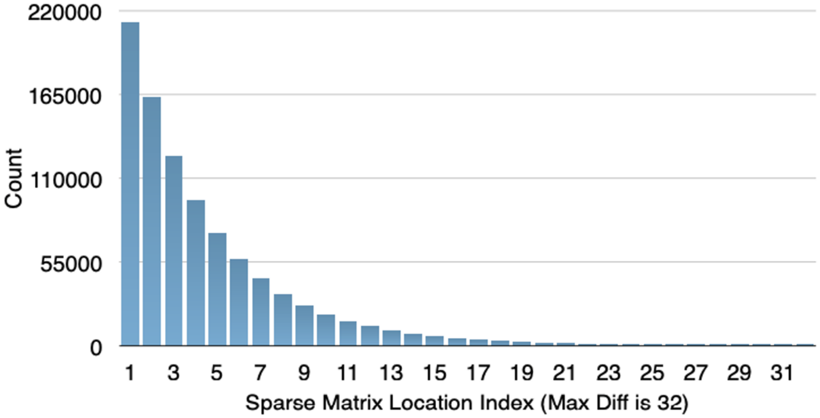
-0.03	0.12	0.02	-0.07		0.04
0.03	0.01	-0.02			0.02
0.02	-0.01	0.01	0.04	-0.02	0.04
-0.01	-0.02	-0.01	0.01		-0.03

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Method 1: K-Means Quantization *with* Huffman Encodings

Looking at the quantized weights and centroid indices for the last layer of an AlexNet (vision) model:

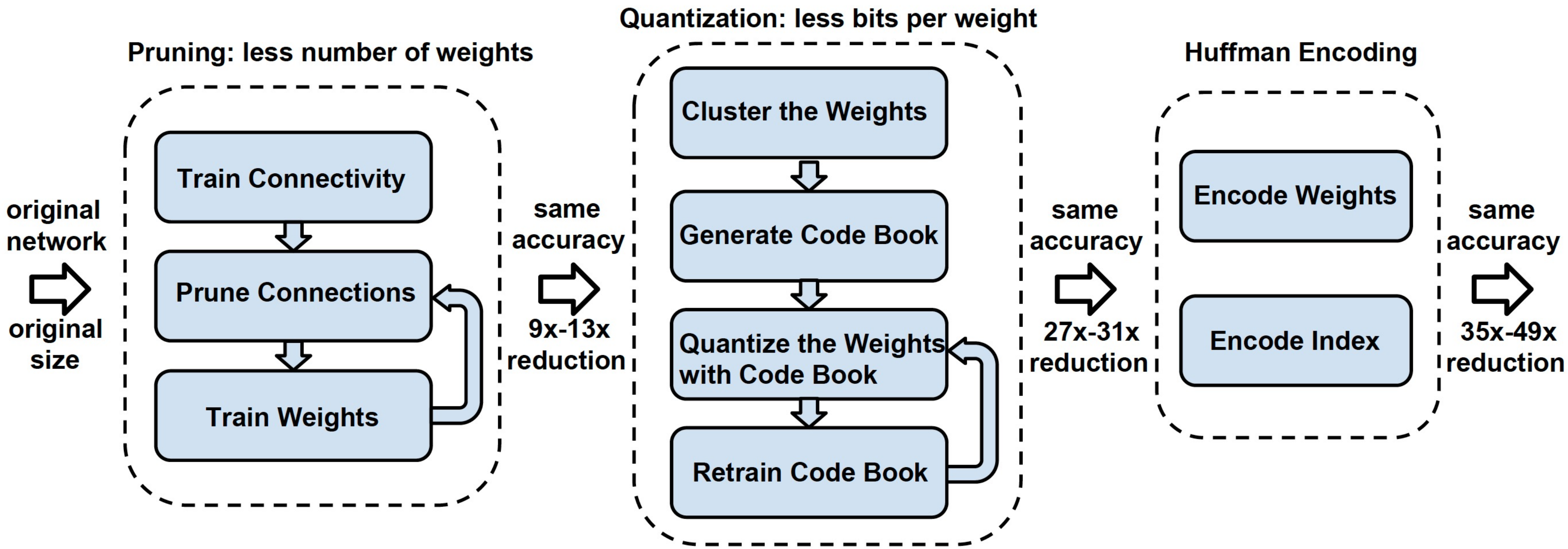




Huffman codes (Van Leeuwen, 1976) use variable-length codewords to encode distributions. We could get away with *fewer bits* for more frequent values! Results in >20%+ memory savings above!

Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Summary of Deep Compression

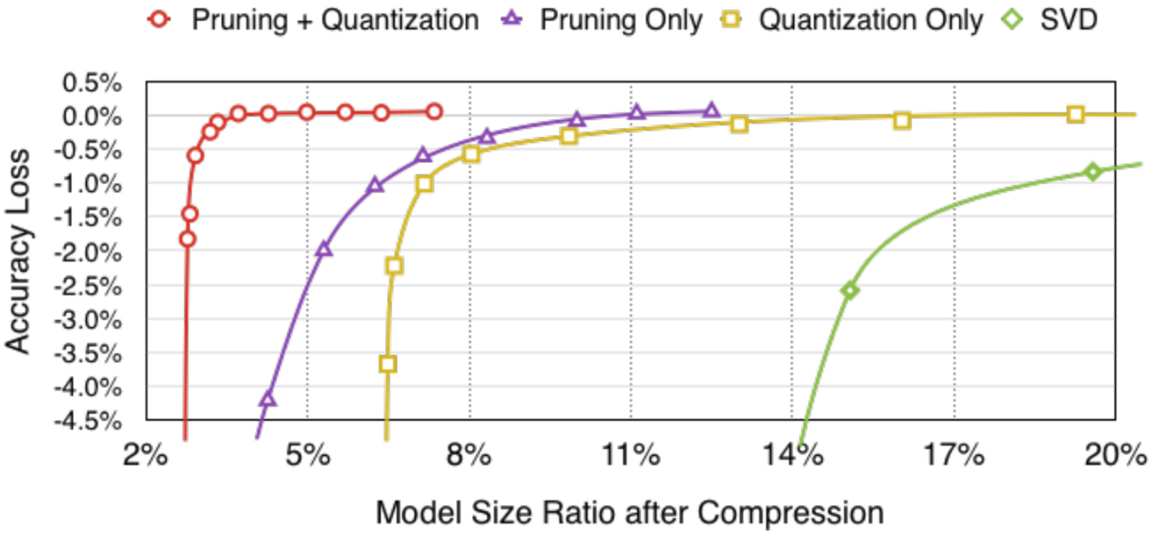


Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Deep Compression Results

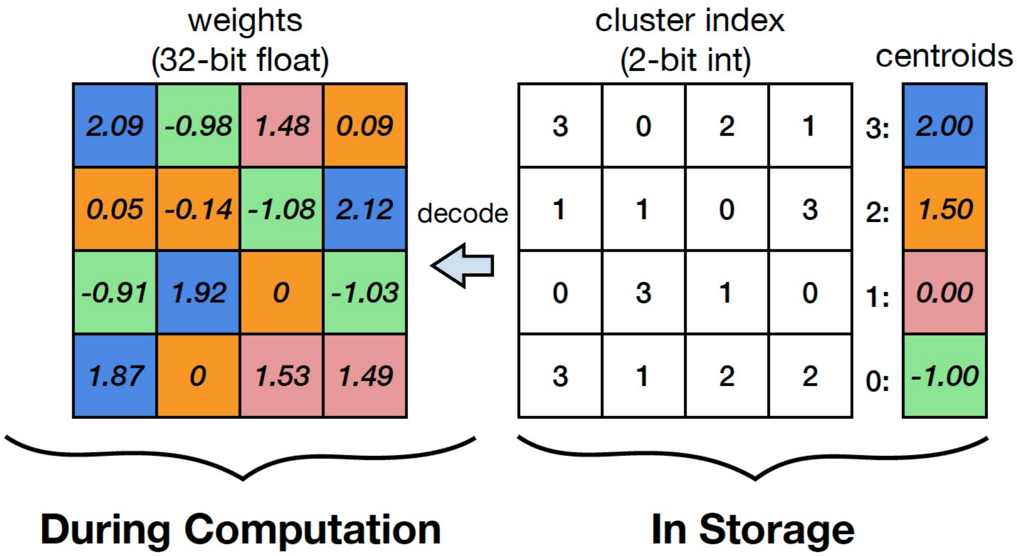
Takeaways:

- For each method, as we increase the compression further, accuracy decreases
- Pruning and quantization work well together vs. one of them alone



Song Han et al., *Deep Compression*, 2016 ICLR (Best Conference Paper)

Recap: K-Means-based Weight Quantization



weights (32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

cluster index (2-bit int)

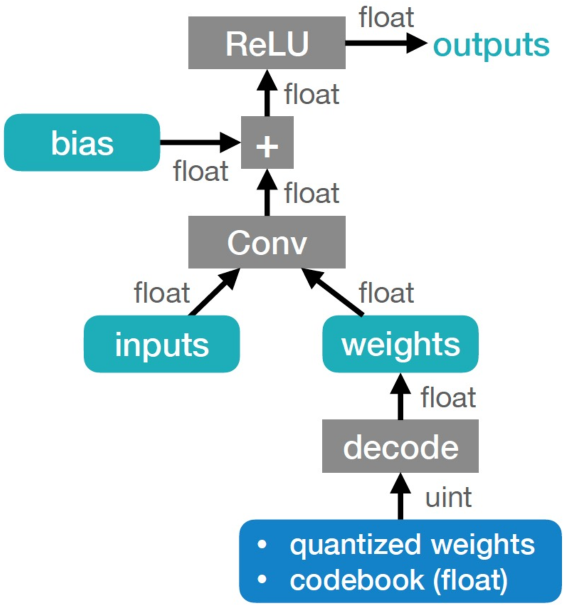
3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

centroids

3: 2.00
2: 1.50
1: 0.00
0: -1.00

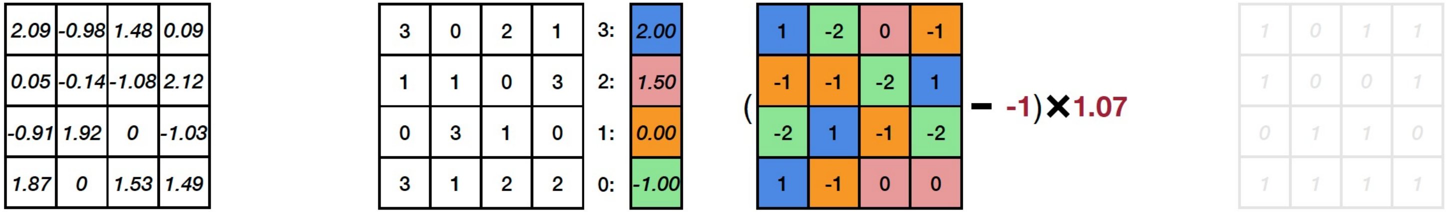
During Computation

In Storage



- The weights are decompressed using a lookup table (*i.e.*, codebook) during runtime inference.
- K-Means-based Weight Quantization only saves storage cost of a neural network model.
  - All the computation and memory access are still floating-point.

Motivation for Linear Quantization



2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

3: 2.00
2: 1.50
1: 0.00
0: -1.00

(

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

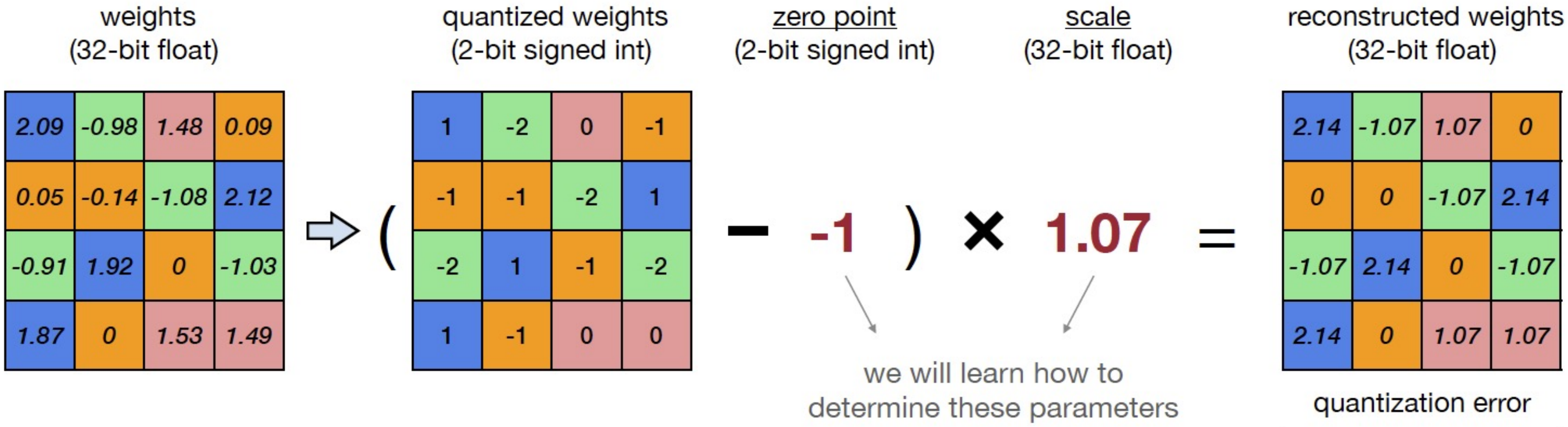
− -1) × 1.07

1	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1

		K-Means-based Quantization	**Linear Quantization**	**Binary/Ternary Quantization**
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	

Method 2: Linear Quantization

An affine mapping of integers to real numbers



weights (32-bit float) ⇨ (quantized weights (2-bit signed int) − zero point (2-bit signed int)) × scale (32-bit float) = reconstructed weights (32-bit float)

$$\begin{bmatrix}2.09 & -0.98 & 1.48 & 0.09\\ 0.05 & -0.14 & -1.08 & 2.12\\ -0.91 & 1.92 & 0 & -1.03\\ 1.87 & 0 & 1.53 & 1.49\end{bmatrix} \Rightarrow \left(\begin{bmatrix}1 & -2 & 0 & -1\\ -1 & -1 & -2 & 1\\ -2 & 1 & -1 & -2\\ 1 & -1 & 0 & 0\end{bmatrix} - -1 \right) \times 1.07 = \begin{bmatrix}2.14 & -1.07 & 1.07 & 0\\ 0 & 0 & -1.07 & 2.14\\ -1.07 & 2.14 & 0 & -1.07\\ 2.14 & 0 & 1.07 & 1.07\end{bmatrix}$$

we will learn how to determine these parameters

Binary	Decimal
01	1
00	0
11	-1
10	-2

quantization error

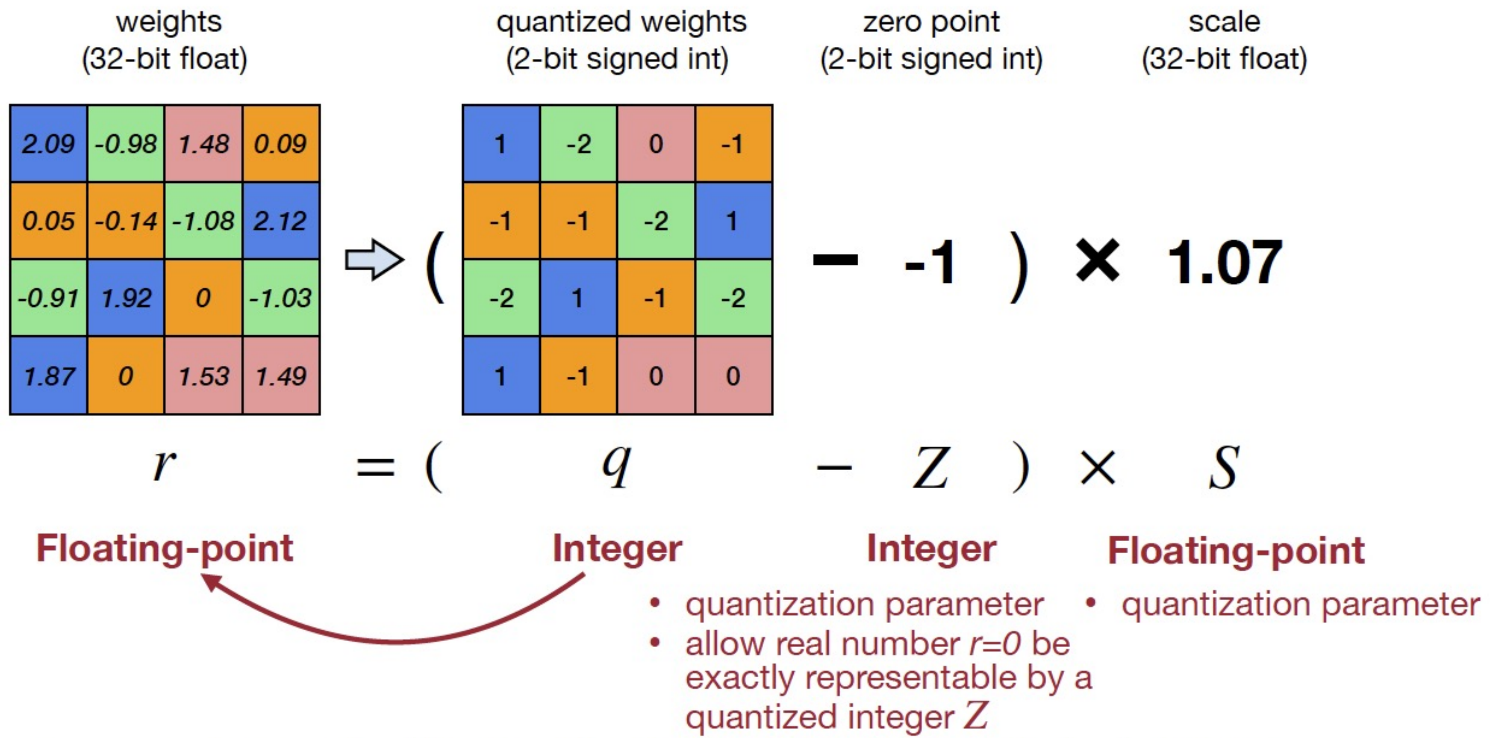
-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

[MIT 6.5940 Fall 2023 TinyML and Efficient Deep Learning Computing](#)

[[1712.05877] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference](#)

Method 2: Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



weights (32-bit float)

quantized weights (2-bit signed int)

zero point (2-bit signed int)

scale (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - -1 \right) \times 1.07$$

$$r = (q - Z) \times S$$

Floating-point

Integer

Integer

- quantization parameter
- allow real number $r=0$ be exactly representable by a quantized integer Z

Floating-point

- quantization parameter

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

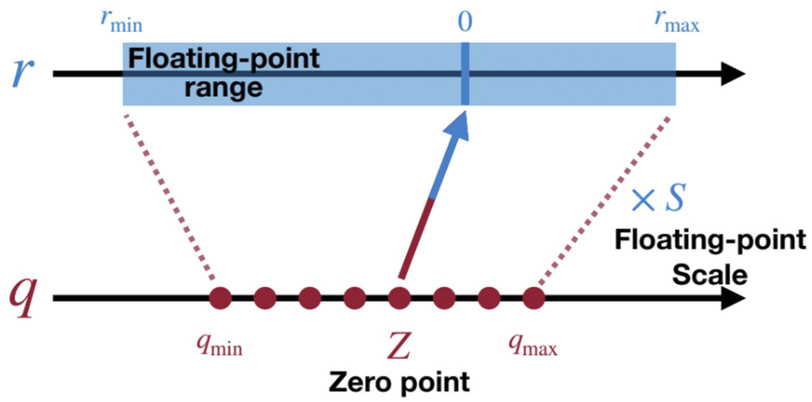
[MIT 6.5940 Fall 2023 TinyML and Efficient Deep Learning Computing]

[[1712.05877] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference]

Method 2: Linear Quantization

Knowns:

- We know $r_{\min}$ and $r_{\max}$ from our original weight matrix
- We know $q_{\min}$ and $q_{\max}$: if we are quantizing to bits (N), the range is (-2^{N-1} through $2^{N-1} - 1$)



Given all those values, we can solve for S and Z (two equations, two unknowns):

1. $r_{\max} = S(q_{\max} - Z)$
2. $r_{\min} = S(q_{\min} - Z)$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$\downarrow$$

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$\downarrow$$

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

Bit Width	$q_{\min}$	$q_{\max}$
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Linear Quantized Matrix Multiplication

Can we use integer computation instead of floating point with our linearly quantized weights?

To compute matmul $\mathbf{Y} = \mathbf{WX}$

where $\mathbf{Y} = S_Y (q_Y - Z_Y)$, $\mathbf{W} = S_W (q_W - Z_W)$, $\mathbf{X} = S_X (q_X - Z_X)$

Rearranging the terms:

$$\mathbf{Y} = \mathbf{WX}$$

$$\Rightarrow \quad \mathbf{q_Y} = \boxed{\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}} \underbrace{\boxed{\left(\mathbf{q_W q_X} - Z_\mathbf{W} \mathbf{q_X} \overbrace{- Z_\mathbf{X} \mathbf{q_W} + Z_\mathbf{W} Z_\mathbf{X}}^{\text{Precompute}}\right)}}_{\substack{\textit{N}\text{-bit Integer Multiplication} \\ \text{32-bit Integer Addition/Subtraction}}} + \underbrace{\boxed{Z_\mathbf{Y}}}_{\substack{\textit{N}\text{-bit Integer} \\ \text{Addition}}}$$

Empirically, the scale $\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}$ is always in the interval (0, 1). $\Rightarrow$ $\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}} = \underset{\text{Bit Shift}}{2^{-n}} \overset{\text{Fixed-point Multiplication}}{M_0}$, where $M_0 \in [0.5, 1)$

Linear Quantized Matrix Multiplication

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication.

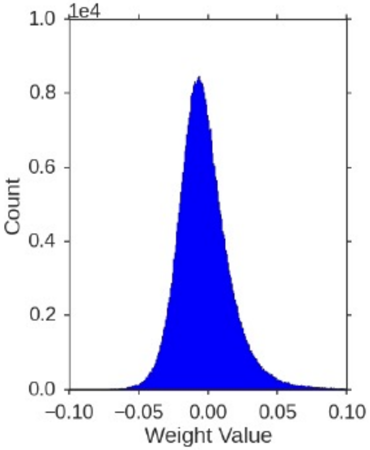
$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$$\mathbf{q_Y} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \left(\boxed{\mathbf{q_W}\mathbf{q_X} - Z_{\mathbf{W}}\mathbf{q_X} \overset{\text{Precompute}}{- Z_{\mathbf{X}}\mathbf{q_W} + Z_{\mathbf{W}} Z_{\mathbf{X}}}} \right) + \boxed{Z_{\mathbf{Y}}}$$

Rescale to
***N*-bit Integer**

***N*-bit Integer Multiplication**
32-bit Integer Addition/Subtraction

***N*-bit Integer**
Addition

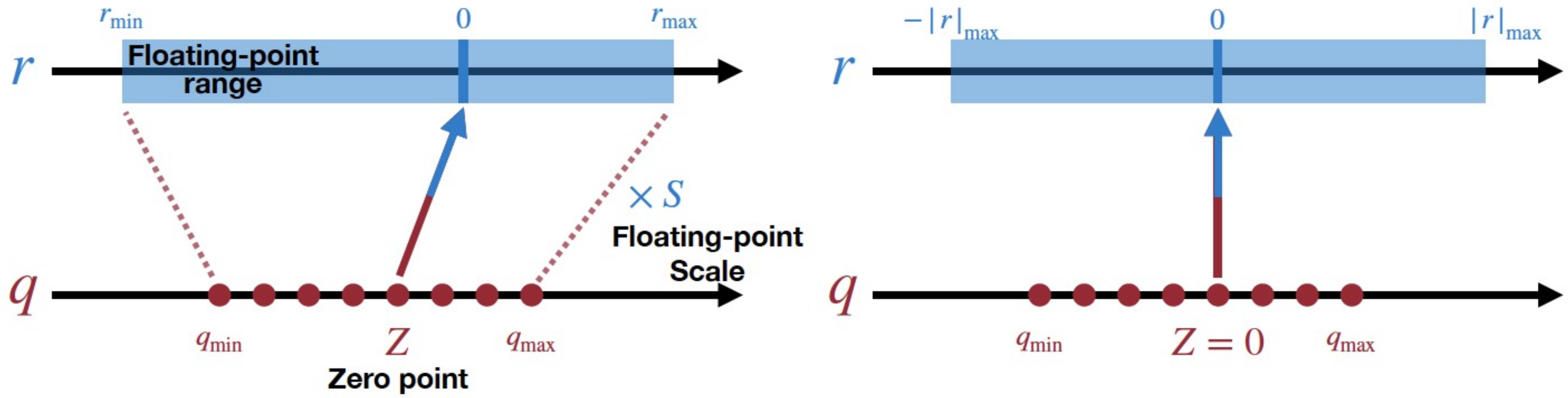


$$Z_{\mathbf{W}} = 0?$$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Symmetric Linear Quantization

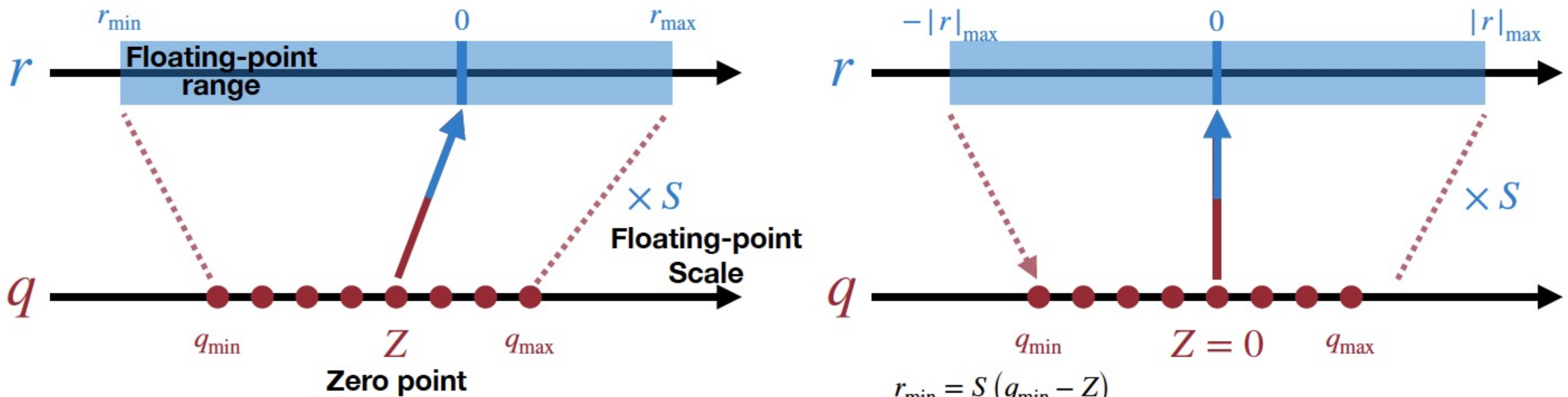
Zero point $Z = 0$ and Symmetric floating-point range



Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Symmetric Linear Quantization

Full range mode



$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

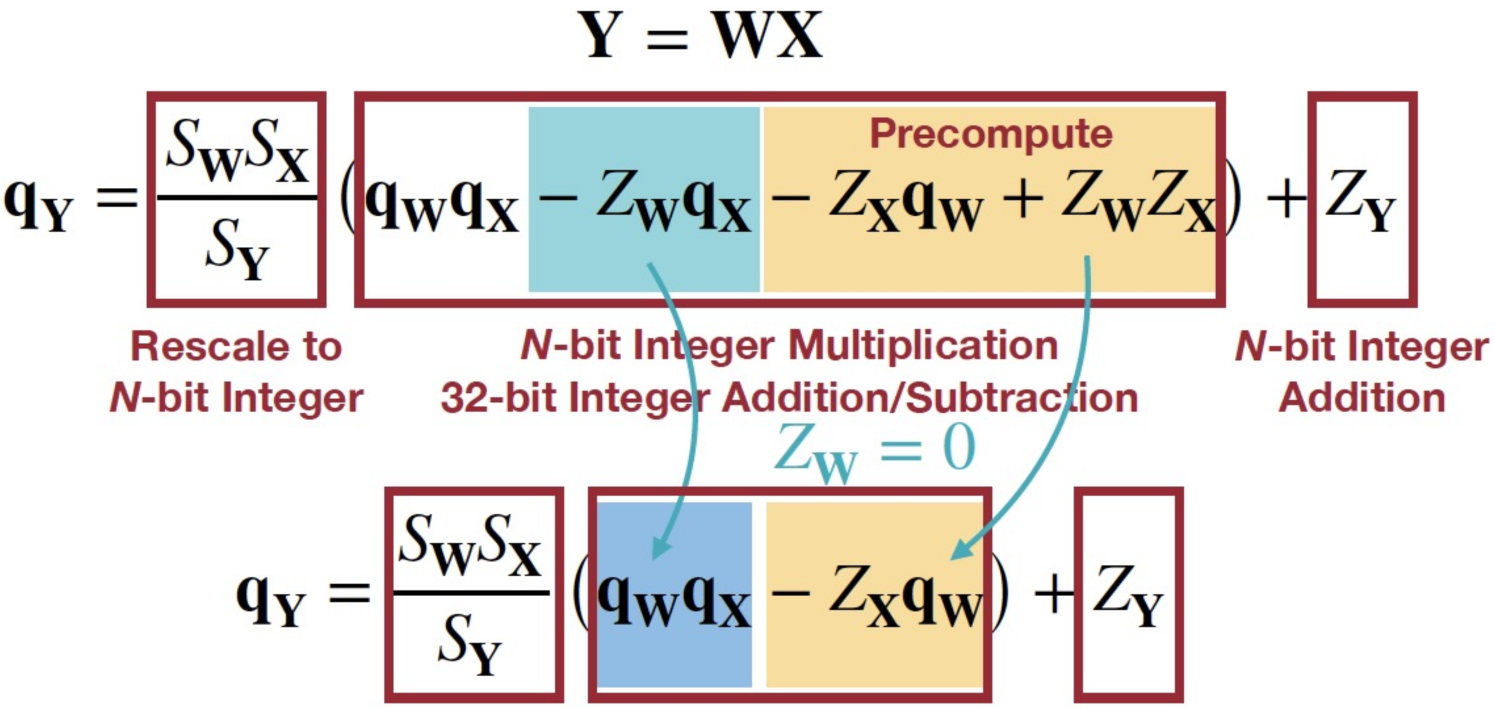
$$S = \frac{r_{\min}}{q_{\min} - Z} = \frac{-|r|_{\max}}{q_{\min}} = \frac{|r|_{\max}}{2^{N-1}}$$

Bit Width	$q_{\min}$	$q_{\max}$
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Linear Quantized Fully-Connected Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication, when Zw=0.



$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

Precompute

$$\mathbf{q_Y} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_{\mathbf{W}}\mathbf{q_X} - Z_{\mathbf{X}}\mathbf{q_W} + Z_{\mathbf{W}}Z_{\mathbf{X}} \right)} + \boxed{Z_{\mathbf{Y}}}$$

Rescale to *N*-bit Integer

N-bit Integer Multiplication
32-bit Integer Addition/Subtraction

N-bit Integer Addition

$Z_{\mathbf{W}} = 0$

$$\mathbf{q_Y} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \boxed{\left(\mathbf{q_W}\mathbf{q_X} - Z_{\mathbf{X}}\mathbf{q_W} \right)} + \boxed{Z_{\mathbf{Y}}}$$

Linear Quantized Fully-Connected Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

$$\downarrow \; Z_{\mathbf{W}} = 0$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}} S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}}\right) + S_{\mathbf{b}}\left(\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{b}}\right)$$

$$\downarrow \; Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}} S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}} + \mathbf{q}_{\mathbf{b}}\right)$$

Linear Quantized Fully-Connected Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0 \ \Big\downarrow \ Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}} S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}} + \mathbf{q}_{\mathbf{b}}\right)$$

$$\Big\downarrow$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} + \overset{\text{Precompute}}{\boxed{\mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}}}}\right) + Z_{\mathbf{Y}}$$

$$\Big\downarrow \ \mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}}$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} + \mathbf{q}_{bias}\right) + Z_{\mathbf{Y}}$$

Linear Quantized Fully-Connected Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- So far, we ignore bias. Now we consider the following fully-connected layer with bias.

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

$$\Big\downarrow \quad \begin{aligned} & Z_{\mathbf{W}} = 0 \\ & Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}} \\ & \mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - Z_{\mathbf{X}} \mathbf{q}_{\mathbf{W}} \end{aligned}$$

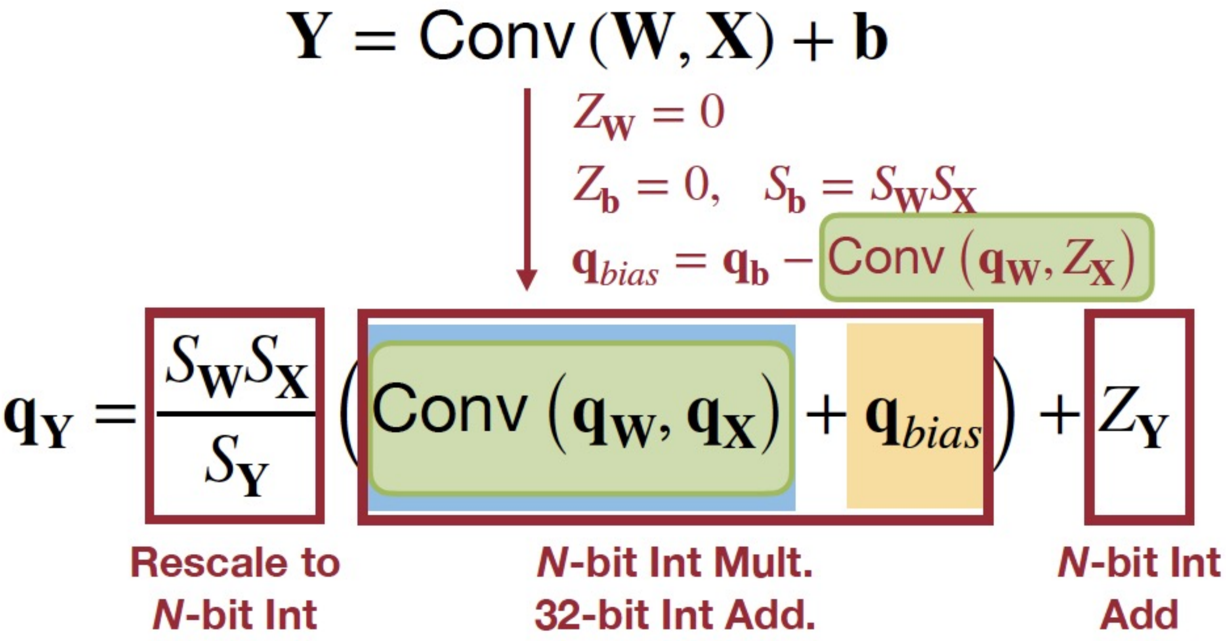
$$\mathbf{q}_{\mathbf{Y}} = \underbrace{\boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}}}_{\text{Rescale to } N\text{-bit Int}} \underbrace{\boxed{(\mathbf{q}_{\mathbf{W}} \mathbf{q}_{\mathbf{X}} + \mathbf{q}_{bias})}}_{N\text{-bit Int Mult. 32-bit Int Add.}} + \underbrace{\boxed{Z_{\mathbf{Y}}}}_{N\text{-bit Int Add}}$$

Note: both $\mathbf{q}_{\mathbf{b}}$ ***and*** $\mathbf{q}_{bias}$ ***are 32 bits.***

Linear Quantized Convolution Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following convolution layer.



$$\mathbf{Y} = \mathrm{Conv}\left(\mathbf{W}, \mathbf{X}\right) + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0$$

$$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - \boxed{\mathrm{Conv}\left(\mathbf{q}_{\mathbf{W}}, Z_{\mathbf{X}}\right)}$$

$$\mathbf{q}_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \left(\boxed{\boxed{\mathrm{Conv}\left(\mathbf{q}_{\mathbf{W}}, \mathbf{q}_{\mathbf{X}}\right)} + \boxed{\mathbf{q}_{bias}}} \right) + \boxed{Z_{\mathbf{Y}}}$$

Rescale to *N*-bit Int

***N*-bit Int Mult.**
32-bit Int Add.

***N*-bit Int Add**

Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.

Linear Quantized Convolution Layer

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following convolution layer.

$$\mathbf{Y} = \mathrm{Conv}\,(\mathbf{W}, \mathbf{X}) + \mathbf{b}$$

$$Z_{\mathbf{W}} = 0$$

$$Z_{\mathbf{b}} = 0, \quad S_{\mathbf{b}} = S_{\mathbf{W}} S_{\mathbf{X}}$$

$$\mathbf{q}_{bias} = \mathbf{q}_{\mathbf{b}} - \boxed{\mathrm{Conv}\,(\mathbf{q}_{\mathbf{W}}, Z_{\mathbf{X}})}$$

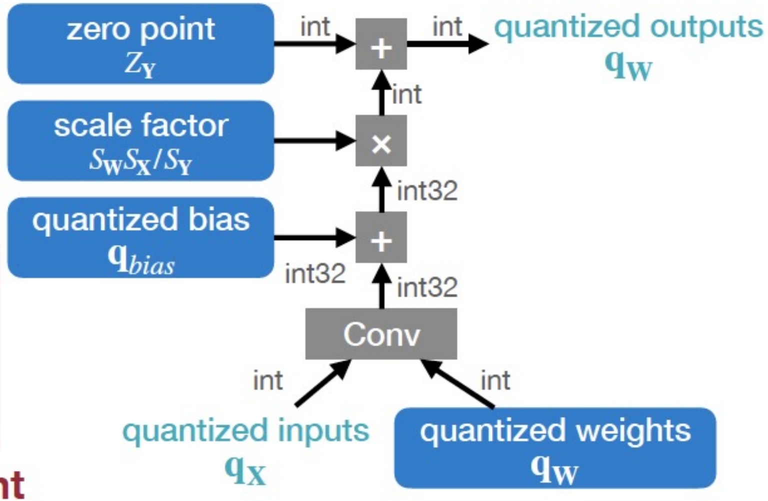
$$\mathbf{q}_{\mathbf{Y}} = \boxed{\frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}} \left(\boxed{\boxed{\mathrm{Conv}\,(\mathbf{q}_{\mathbf{W}}, \mathbf{q}_{\mathbf{X}})} + \boxed{\mathbf{q}_{bias}}} \right) + \boxed{Z_{\mathbf{Y}}}$$

Rescale to *N*-bit Int

***N*-bit Int Mult. 32-bit Int Add.**

***N*-bit Int Add**

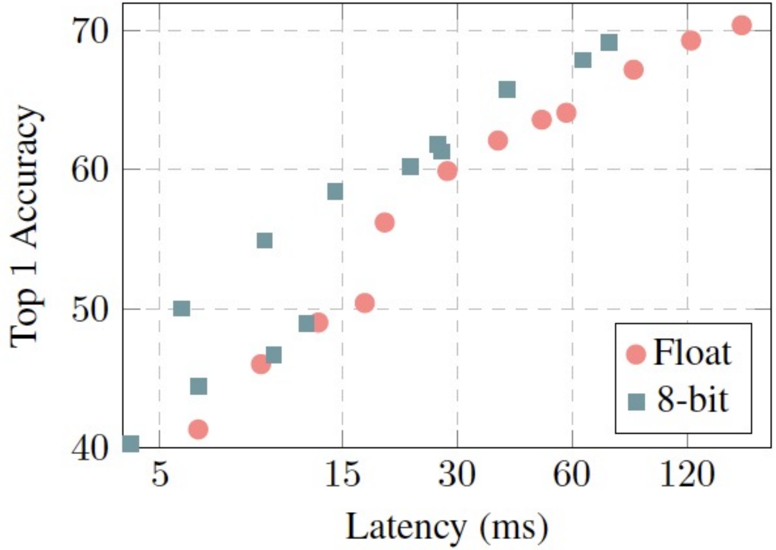
Note: both $\mathbf{q}_{\mathbf{b}}$ and $\mathbf{q}_{bias}$ are 32 bits.



INT8 Linear Quantization Results

An affine mapping of integers to real numbers $r = S(q - Z)$

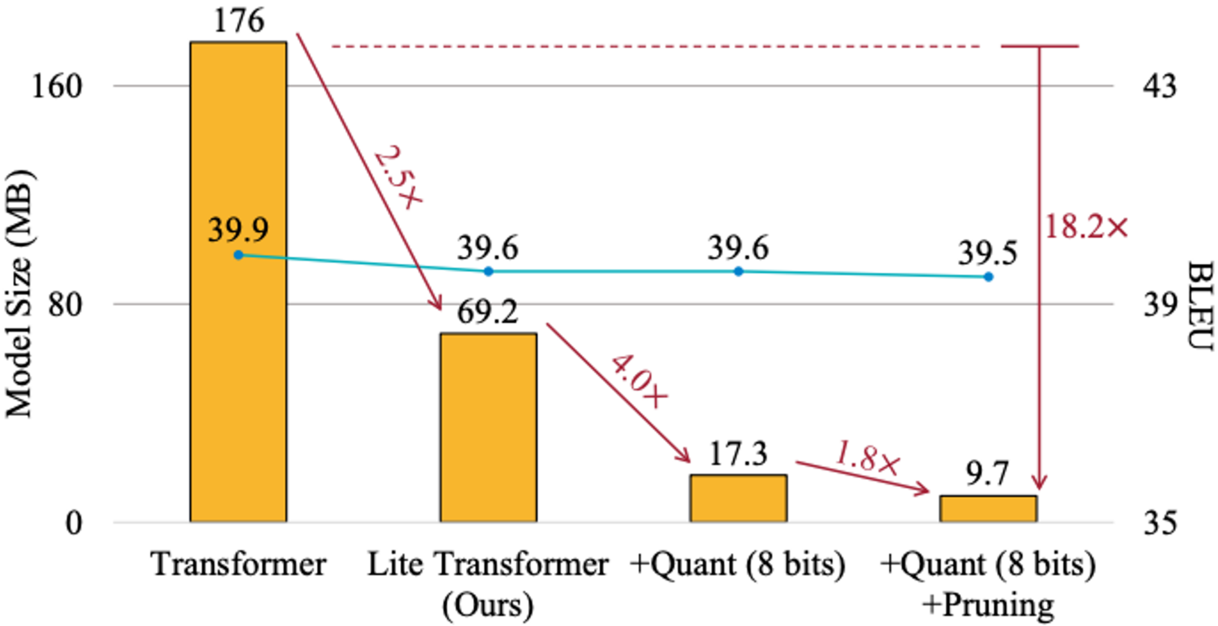
Neural Network	ResNet-50	Inception-V3
Floating-point Accuracy	76.4%	78.4%
8-bit Integer-quantized Acurracy	74.9%	75.4%



Latency-vs-accuracy tradeoff of float vs. integer-only MobileNets on ImageNet using Snapdragon 835 big cores.

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Applications to Transformers



[2004.11886] Lite Transformer with Long-Short Range Attention, ICLR2020

References for Neural Networks Quantization

1. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng et al., IEEE 2020]
2. Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]
3. Deep Compression [Han et al., ICLR 2016]
4. Neural Network Distiller: https://intellabs.github.io/distiller/algo_quantization.html
5. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
6. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations [Courbariaux et al., NeurIPS 2015]
7. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. [Courbariaux et al., Arxiv 2016]
8. XNOR-Net: ImageNet Classification using Binary Convolutional Neural Networks [Rastegari et al., ECCV 2016]
9. Ternary Weight Networks [Li et al., Arxiv 2016]
10. Trained Ternary Quantization [Zhu et al., ICLR 2017]

MIT 6.5940 Fall 2024 TinyML and Efficient Deep Learning Computing
[1712.05877] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

More References for Neural Networks Quantization

1. Deep Compression [Han et al., ICLR 2016]
2. Neural Network Distiller: https://intellabs.github.io/distiller/algo_quantization.html
3. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]
4. Data-Free Quantization Through Weight Equalization and Bias Correction [Markus et al., ICCV 2019]
5. Post-Training 4-Bit Quantization of Convolution Networks for Rapid-Deployment [Banner et al., NeurIPS 2019]
6. 8-bit Inference with TensorRT [Szymon Migacz, 2017]
7. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper [Raghuraman Krishnamoorthi, arXiv 2018]
8. Neural Networks for Machine Learning [Hinton et al., Coursera Video Lecture, 2012]
9. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Bengio, arXiv 2013]
10. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or –1. [Courbariaux et al., Arxiv 2016]
11. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients [Zhou et al., arXiv 2016]
12. PACT: Parameterized Clipping Activation for Quantized Neural Networks [Choi et al., arXiv 2018]
13. WRPN: Wide Reduced-Precision Networks [Mishra et al., ICLR 2018]
14. Towards Accurate Binary Convolutional Neural Network [Lin et al., NeurIPS 2017]
15. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights [Zhou et al., ICLR 2017]
16. HAQ: Hardware-Aware Automated Quantization with Mixed Precision [Wang et al., CVPR 2019]