SFU NatLangLab

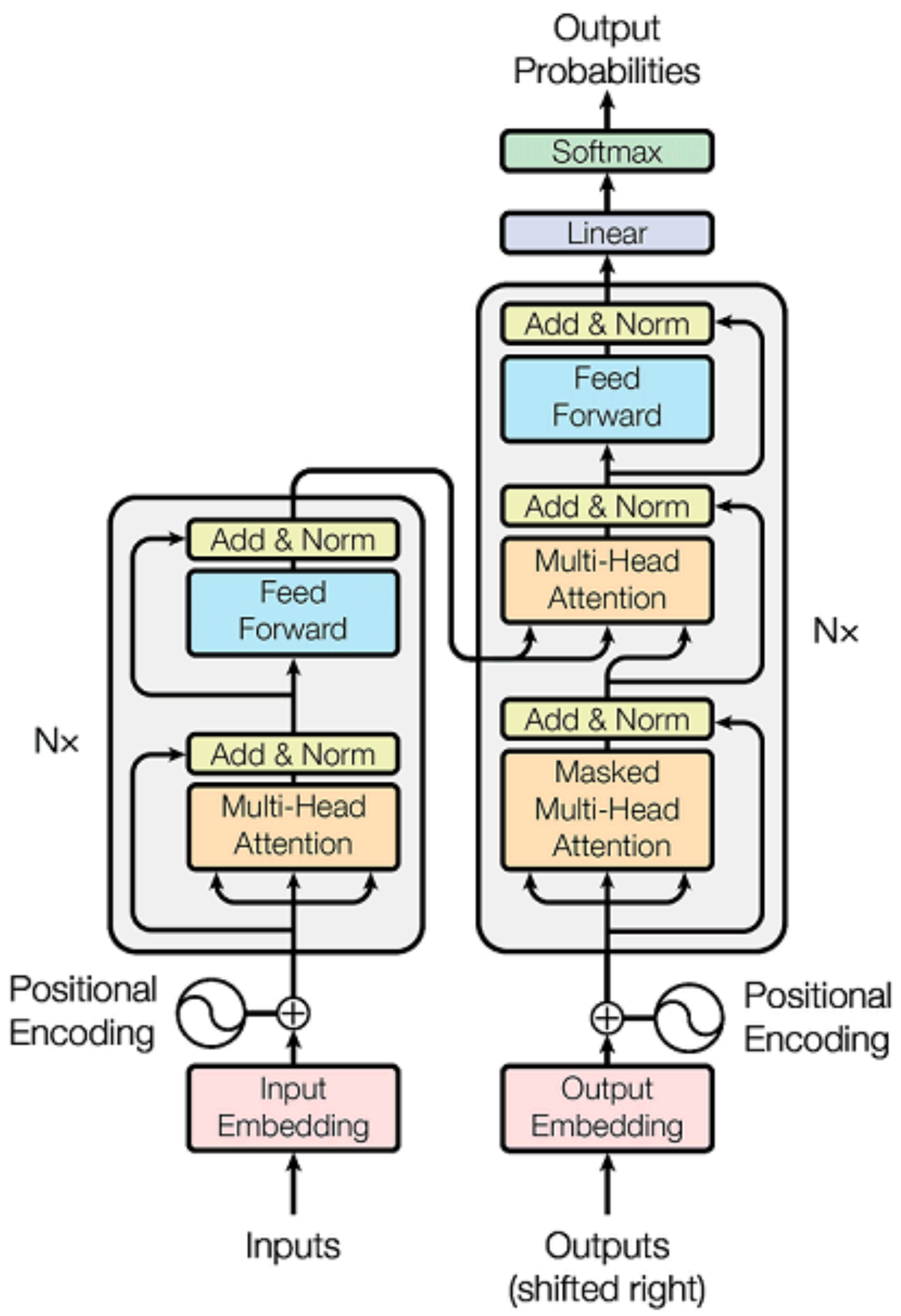
CMPT 413/713: Natural Language Processing

Modern LLM Architecture

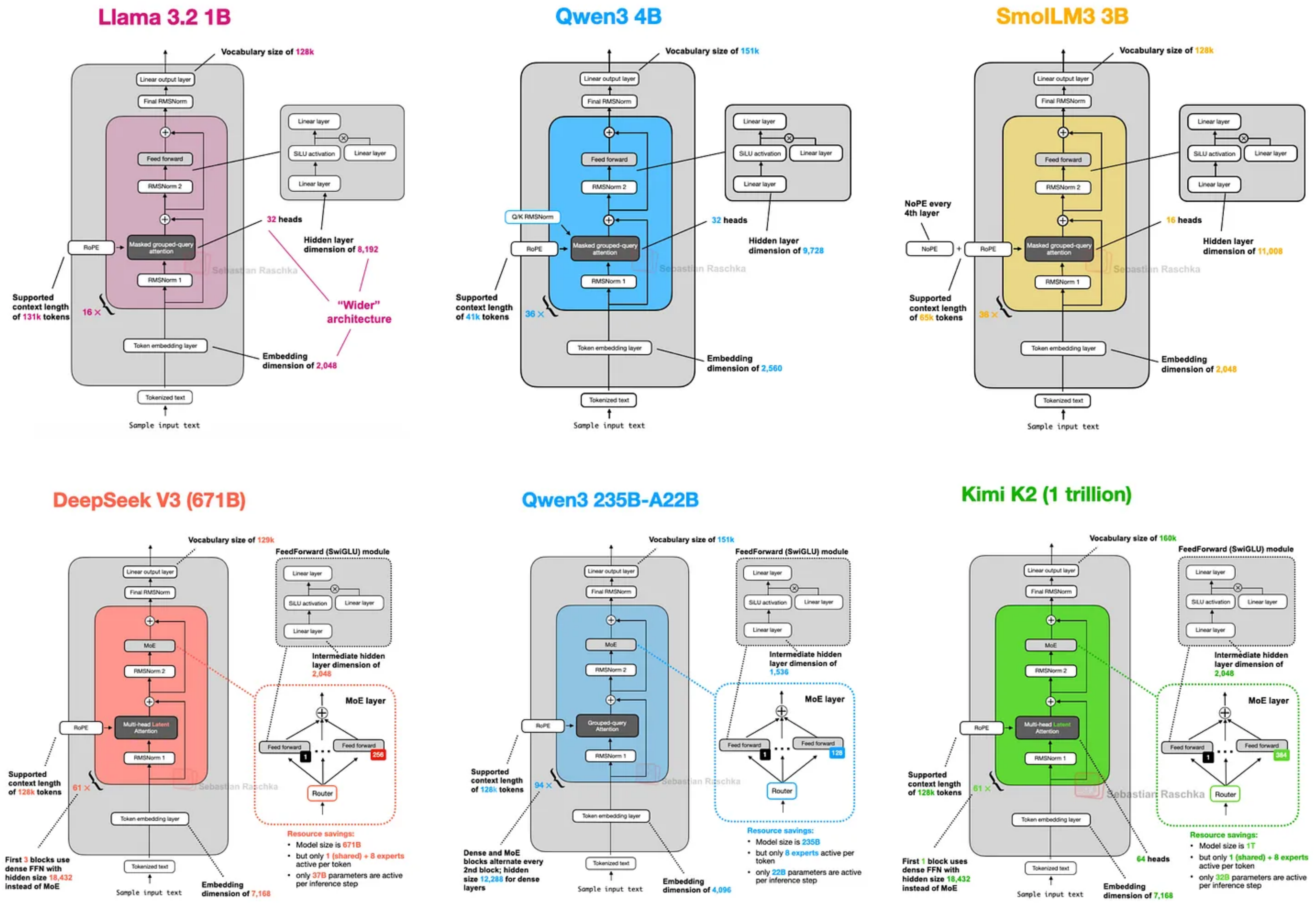
Spring 2026

2026-02-26

Lots of developments to the transformer block!



Transformers [2017]



https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison

PaLM

PaLM: Scaling Language Modeling with Pathways, Chowdhery et al, Google, 2022

Architecture

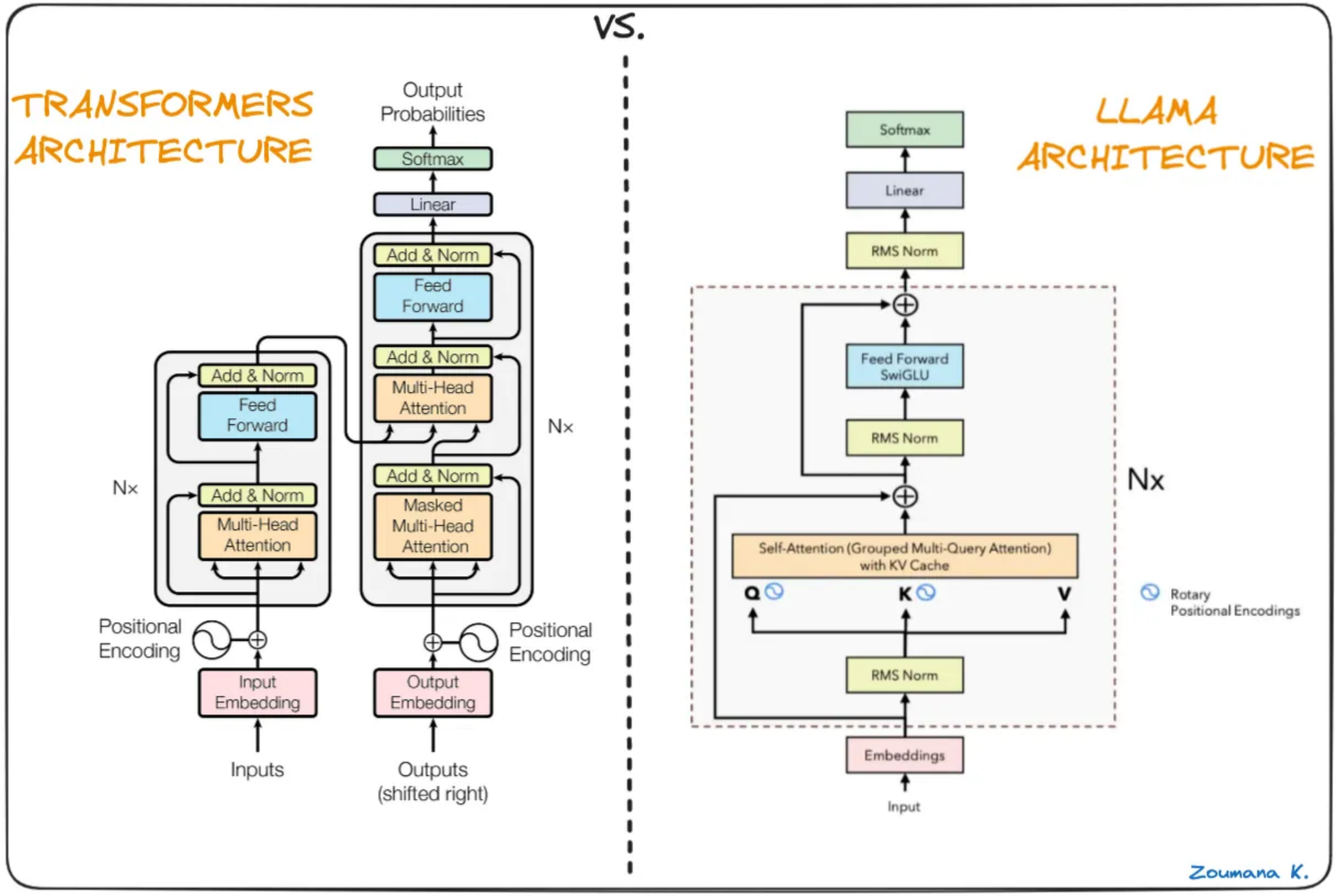
- SwiGLU activation: $\text{Swish}(xW) \otimes xV$
- Parallel layers
  - Serial: $y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x)))$
  - Parallel: $y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$
    - 15% faster training speed (degradation for small models 8B, but no degradation at 62B)
- Attention: Shared key-value across heads, query is still separately projected per head
- RoPE (rotary position) embeddings
- Shared input-output embeddings
- No biases: increased training stability
- Vocabulary: SentencePiece with 256k tokens

Training data

- 780 billion tokens of natural language + source code from github

LLaMa

	Vaswani et al.	LLaMa
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SwiGLU
Positional Encoding	Sinosoidal	RoPE
Attention	Full Multi-Head Attention	Grouped Multi-Query Attention



https://pub.towardsai.net/llama-explained-a70e71e706e9

Revisiting transformers

- Positional encoding
- Normalization
  - Layer normalization vs RMS
  - Post vs Pre-Layer norm
- Activation functions
- Attention variations

Revisiting transformers

- **Positional encoding**
- Normalization
  - Layer normalization vs RMS
  - Post vs Pre-Layer norm
- Activation functions
- Attention variations

Positional encoding

- Original transformer: fixed sinusoidal absolute embeddings
- Learned encoding
- Absolute vs relative
  - In most cases, it is the relative position between two words that matter (not their absolute position)
  - Relative encoding can be learned [Self-Attention with Relative Position Representations, Shaw et al. 2018]
- Rotary embeddings (RoPE)

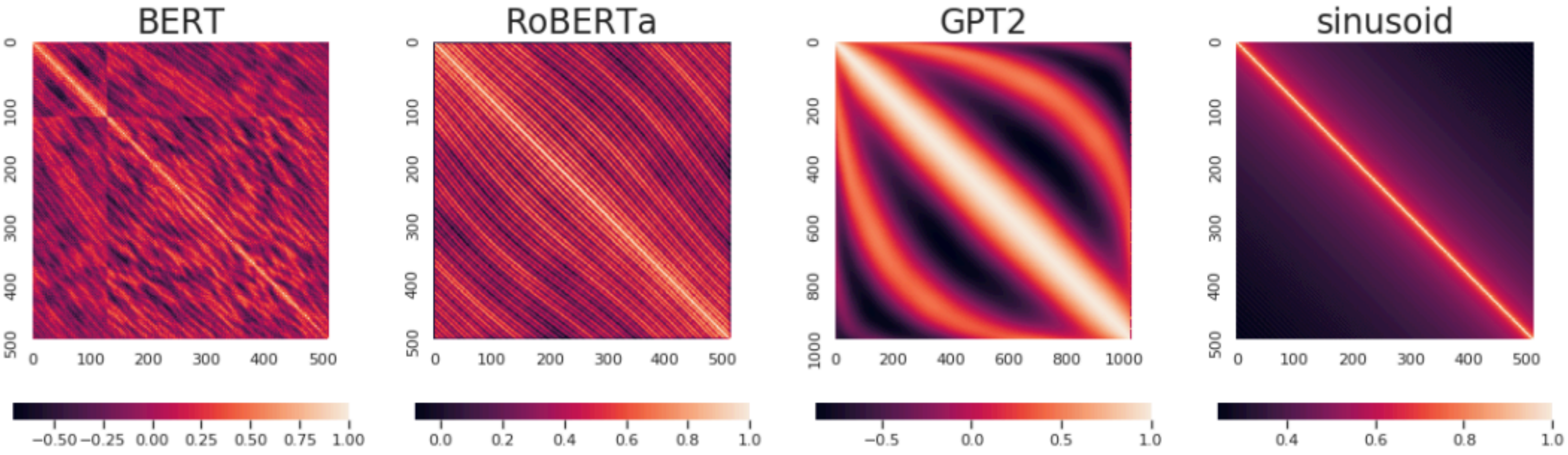
Learned encoding

- Advantage: Flexible, learned representations
- Disadvantage: bunch of extra parameters that need to be learned
- Disadvantage: impossible to extrapolate to longer sequences

Learned encoding

What do position embeddings learn?

- Visualize cosine similarity between position embeddings
- GPT-2 learned embeddings are quite good: can effectively predict absolute position using linear regression and relative ordering using logistic regression



What Do Position Embeddings Learn? [Wang and Chen 2020]

Learned encoding

What do position embeddings learn?

- Visualize cosine similarity between position embeddings
- GPT-2 learned embeddings are quite good: can effectively predict absolute position using linear regression and relative ordering using logistic regression

Absolute

Type	PE	MAE
Learned	BERT	34.14
	RoBERTa	6.06
	GPT-2	1.03
Pre-Defined	sinusoid	0.0

Relative

Type	PE	Error Rate
Learned	BERT	19.72%
	RoBERTa	7.23%
	GPT-2	1.56%
Pre-Defined	sinusoid	5.08%

What Do Position Embeddings Learn? [Wang and Chen 2020]

Relative encoding

- Learnable relative embeddings

$$f_q(\boldsymbol{x}_m) := \boldsymbol{W}_q \boldsymbol{x}_m$$
$$f_k(\boldsymbol{x}_n, n) := \boldsymbol{W}_k(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^k)$$
$$f_v(\boldsymbol{x}_n, n) := \boldsymbol{W}_v(\boldsymbol{x}_n + \tilde{\boldsymbol{p}}_r^v)$$

Self-Attention with Relative Position Representations
[Shaw et al. 2018]

- Modify attention scores to capture relative embedding

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + \boldsymbol{p}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{p}_n$$

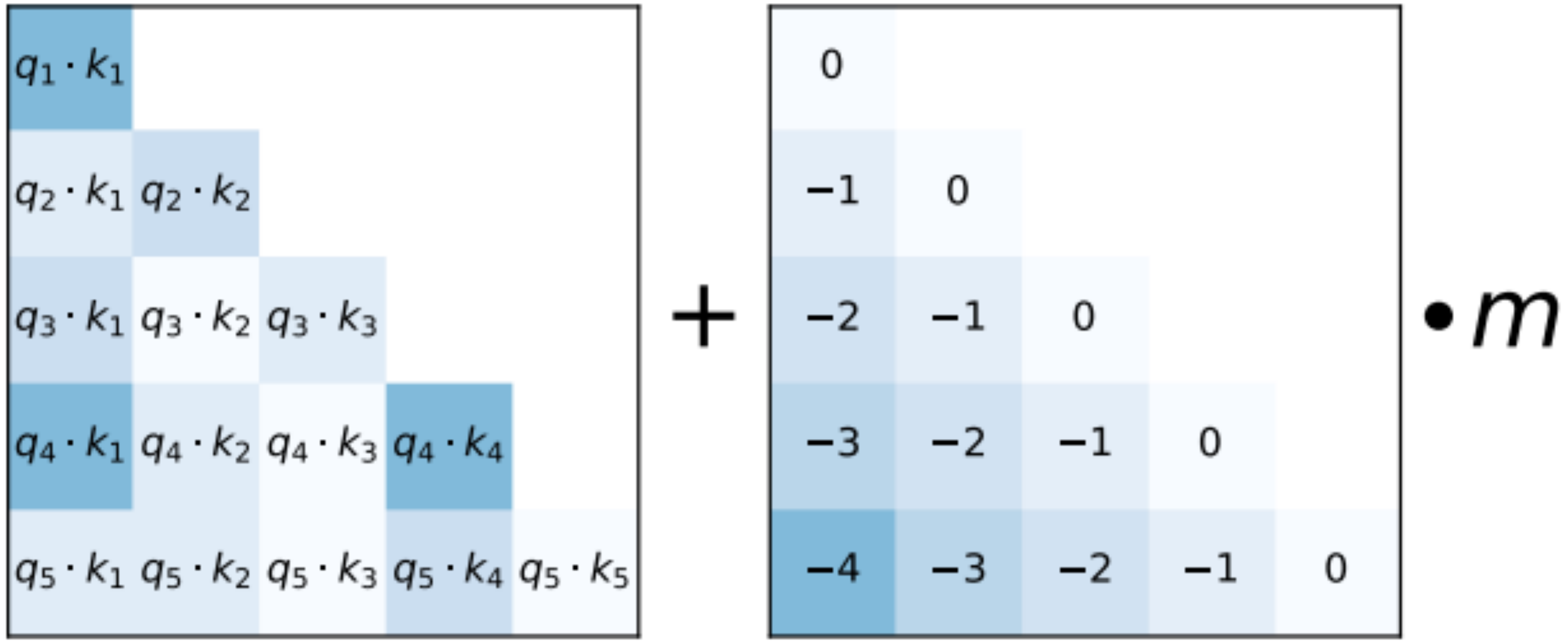
- Simplify to just learning a bias term

$$\boldsymbol{q}_m^\intercal \boldsymbol{k}_n = \boldsymbol{x}_m^\intercal \boldsymbol{W}_q^\intercal \boldsymbol{W}_k \boldsymbol{x}_n + b_{i,j}$$

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [Raffel et al. 2018]

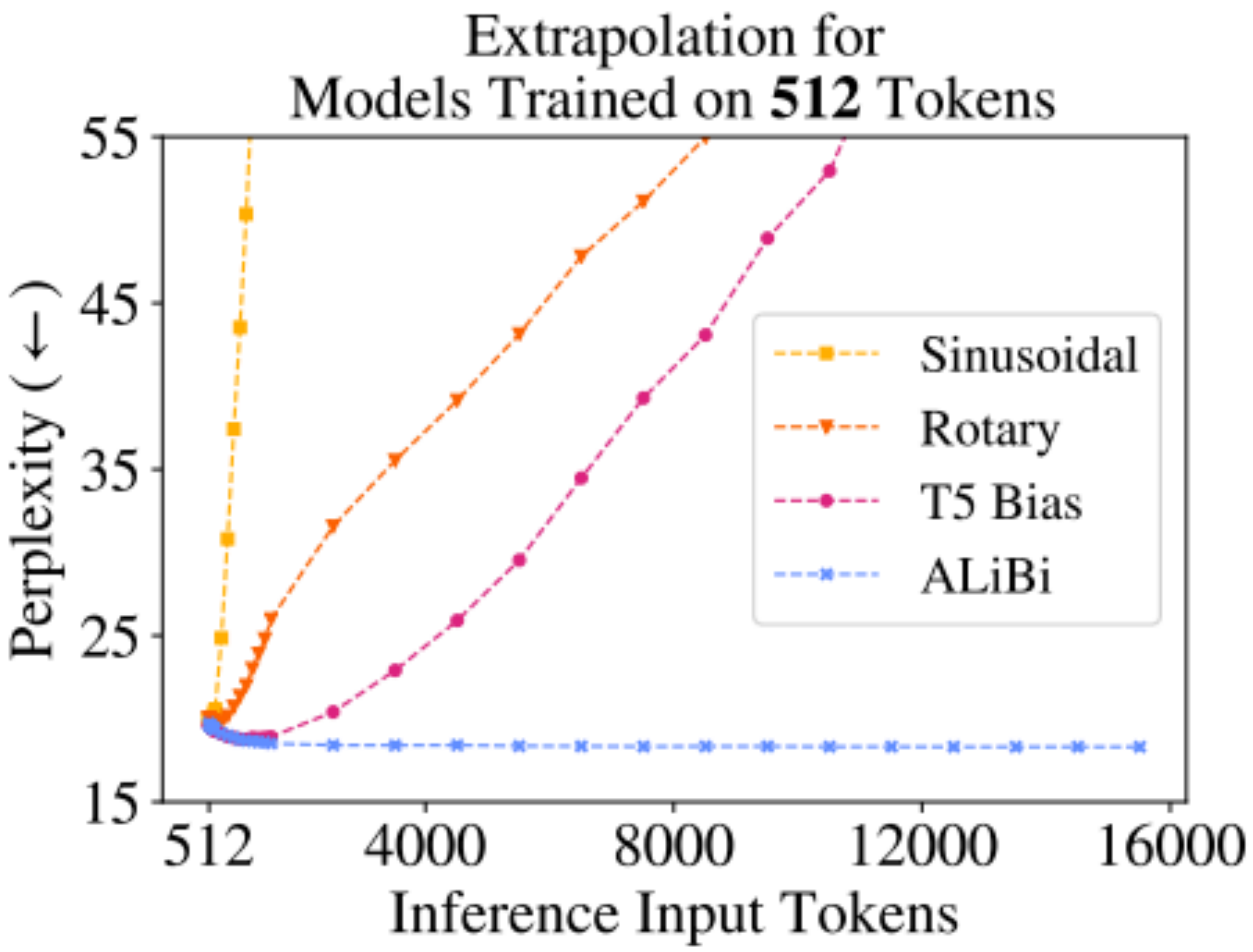
Attention with Linear Biases (ALiBi)

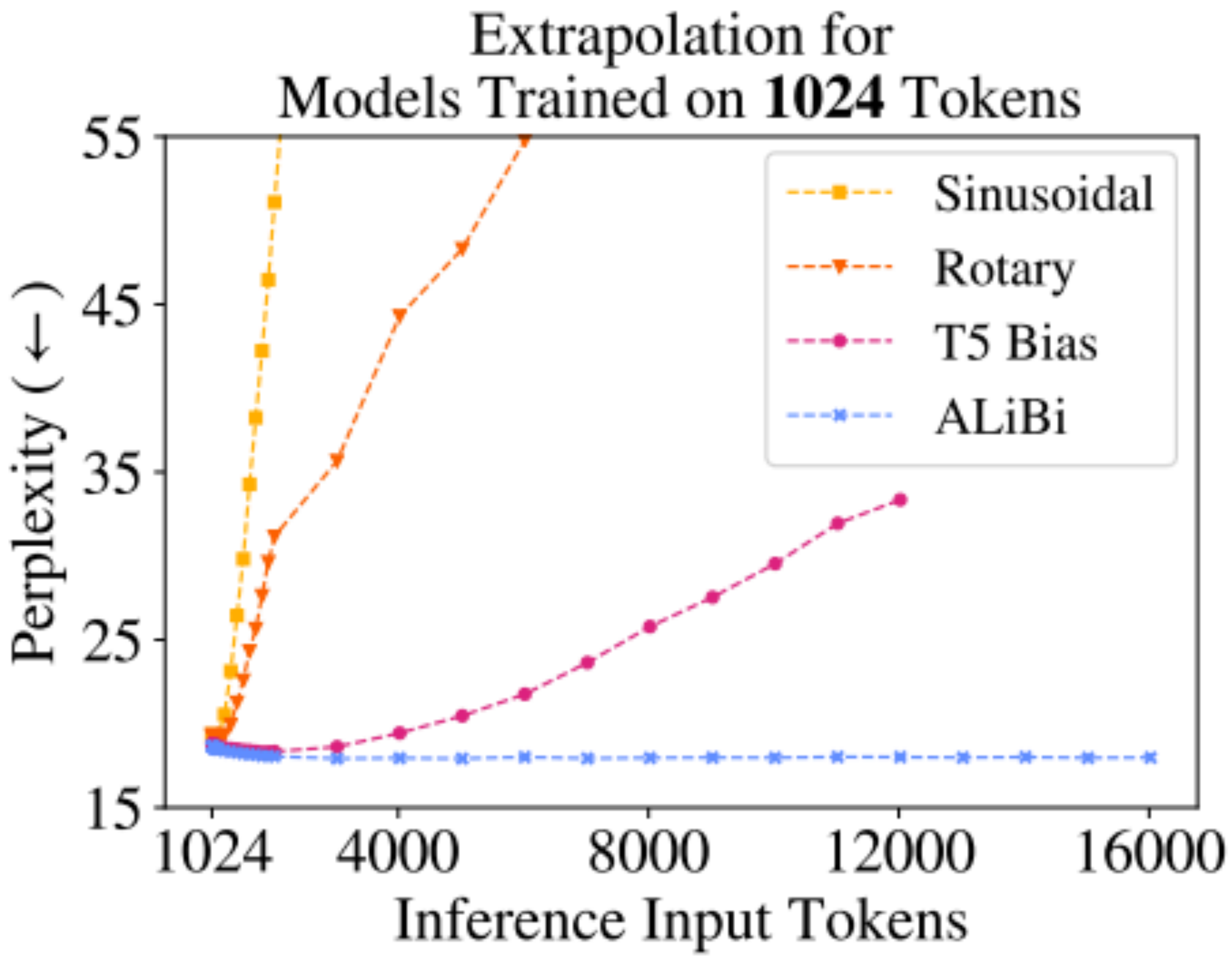
- No explicit position embedding
- Bias query-key attention scores with fixed penalty that is proportional to the distance
- Allows for better extrapolation to long sequences at test time



Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation [Press et al. 2021]

Attention with Linear Biases (ALiBi)





Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation [Press et al. 2021]

Rotary encoding

- Design absolute embeddings so the dot product result in function of relative position

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m - n)$$

- **Rotary Position Embedding (RoPE)**: Apply rotation to encode positional encoding (vs using addition).

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}^d_{\Theta,m} \mathbf{W}_{\{q,k\}} \mathbf{x}_m$$

  - For dimension d, break down into $d/2$ 2D subspace.
    - For vector of length d, group into $d/2$ pairs of numbers - each pair is a vector in 2D.
    - Encode positional encoding m on each pair by rotating by an angle $m\theta$

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding

- Design absolute embeddings so the dot product result in function of relative position

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m-n)$$

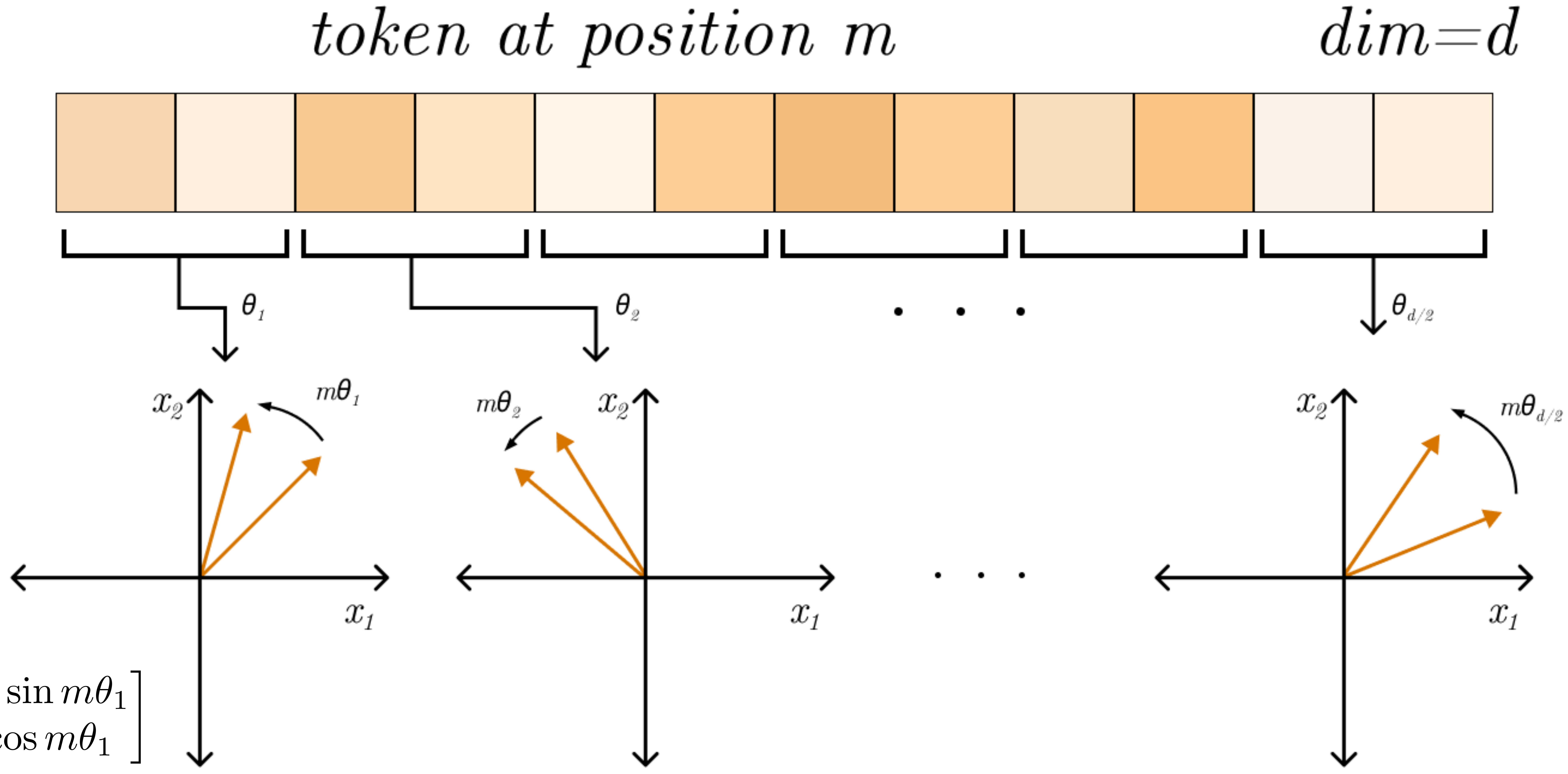
- **Rotary Position Embedding (RoPE)**: Apply rotation to encode positional encoding (vs using addition).

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}^d_{\Theta,m}\mathbf{W}_{\{q,k\}}\mathbf{x}_m$$

$$R^d_{\Theta,m} = \begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{\frac{d}{2}} & -\sin m\theta_{\frac{d}{2}} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{\frac{d}{2}} & \cos m\theta_{\frac{d}{2}} \end{bmatrix}$$

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding



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

https://iclr-blogposts.github.io/2025/blog/positional-embedding/

Rotary encoding

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}^d_{\Theta,m}\mathbf{W}_{\{q,k\}}\mathbf{x}_m$$

Why it works?

- For dimension d, break down into $d/2$ 2D subspace.
  - For vector of length d, group into $d/2$ pairs of numbers - each pair is a vector in 2D.
  - Encode positional encoding m on each pair by rotating by an angle $m\theta$

$$R^d_{\Theta,m} = \begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{\frac{d}{2}} & -\sin m\theta_{\frac{d}{2}} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{\frac{d}{2}} & \cos m\theta_{\frac{d}{2}} \end{bmatrix}$$

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding

More efficient form

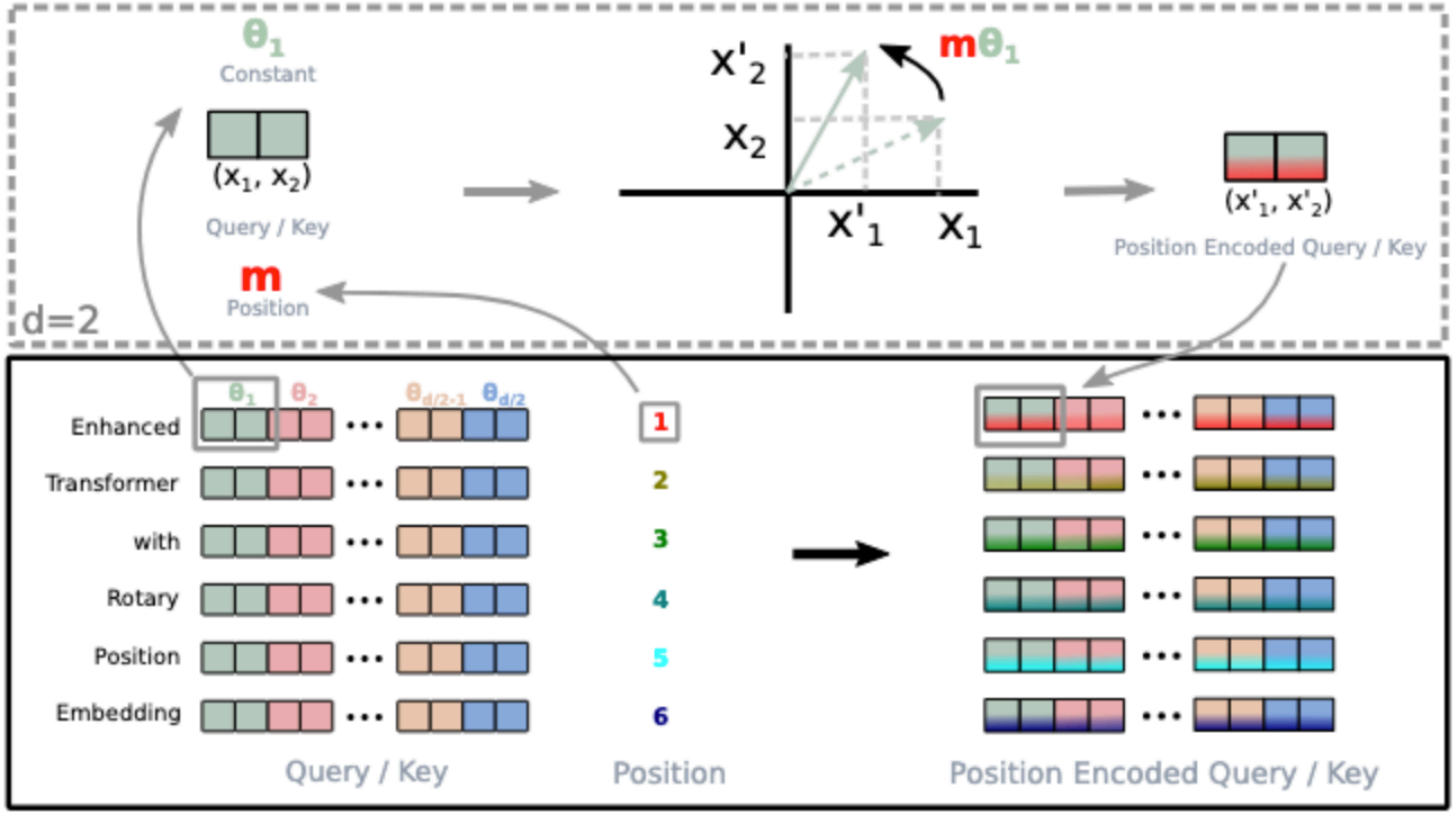
- With just element wise multiply and addition

$$R^d_{\Theta,m}\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{bmatrix} \otimes \begin{bmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{\frac{d}{2}} \\ \cos m\theta_{\frac{d}{2}} \end{bmatrix} + \begin{bmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{bmatrix} \otimes \begin{bmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{\frac{d}{2}} \\ \sin m\theta_{\frac{d}{2}} \end{bmatrix}$$

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1 \ldots d/2]\}$$

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding



RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding

Benefits

- Absolute encoding that captures relative information
- Computationally efficient
- Integrates well into KV cache

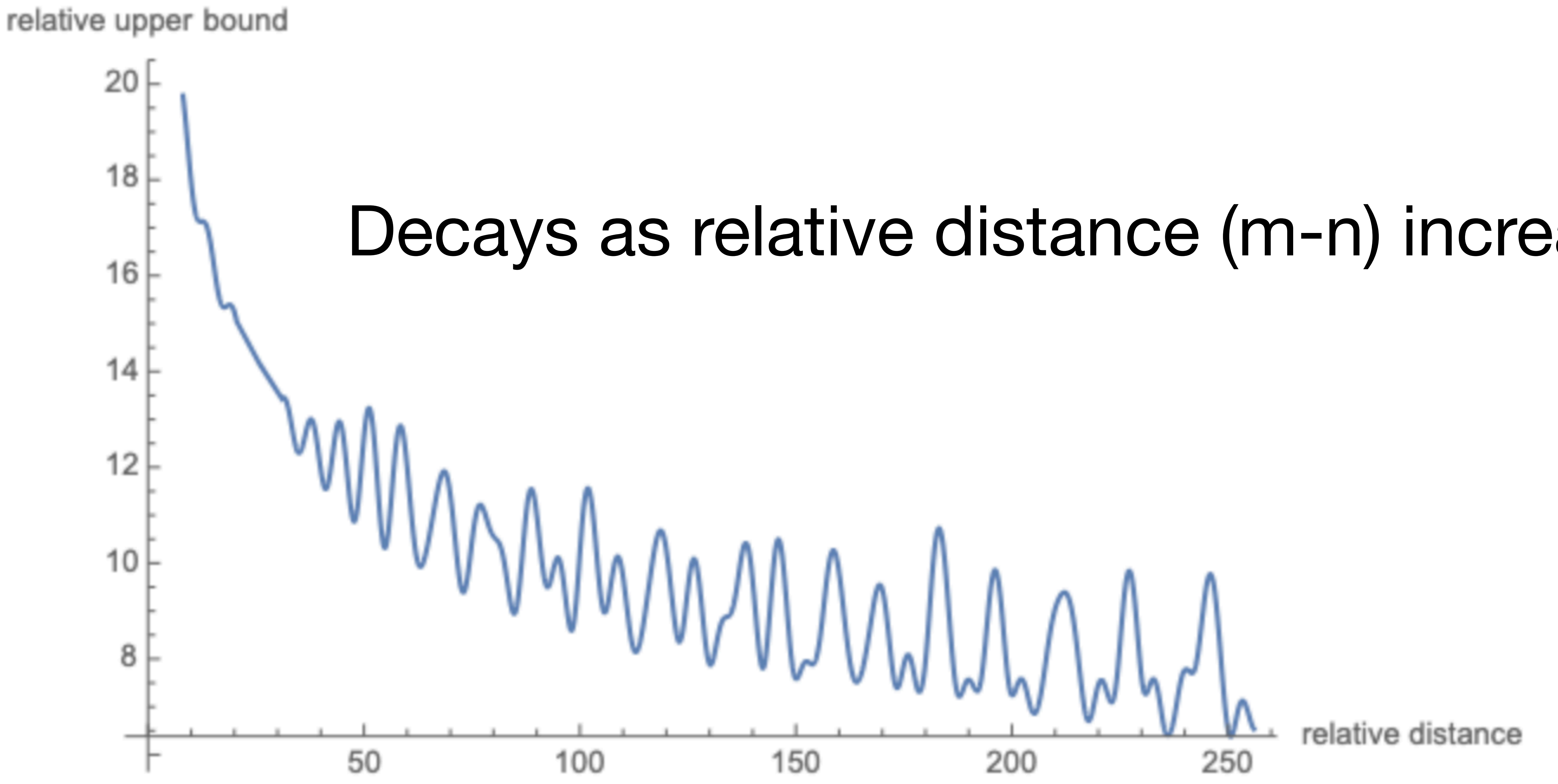
RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Rotary encoding

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}^d_{\Theta,m}\mathbf{W}_{\{q,k\}}\mathbf{x}_m$$

Dot product

$$\mathbf{q}_m^\intercal \mathbf{k}_n = (\mathbf{R}^d_{\Theta,m}\mathbf{W}_q\mathbf{x}_m)^\intercal(\mathbf{R}^d_{\Theta,n}\mathbf{W}_k\mathbf{x}_n) = \mathbf{x}_m^\intercal\mathbf{W}_q^\intercal(\mathbf{R}^{d\intercal}_{\Theta,m}\mathbf{R}^d_{\Theta,n})\mathbf{W}_k\mathbf{x}_n$$



Decays as relative distance (m-n) increases

Figure 2: Long-term decay of RoPE.

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al. 2021]

Current LLMs use RoPE

- Advantage of RoPE may not hold for large models

Model size	Pos. Emb.	Performance zs-main ↑	zs-small ↑	ppl-pile ↓
Likely threshold (1-σ)		±1.1	±0.4	±0.002
1B	ALiBi	49.2	43.1	0.895
	URPE	49.6	43.1	0.885
	RoPE	**50.0**	**44.2**	**0.883**
3B	ALiBi	**54.4**	49.8	0.807
	RoPE	**54.4**	**50.5**	**0.799**

The Falcon Series of Open Language Models
[Almazrouei et al. 2023]

- But can depend on value of theta
  - Recent models use $\theta = 500K$ (vs initial $\theta = 10K$) for long context

LLaMa, OLMo

Comparing RoPE variants

Adjusted base frequency (ABF)

- RoPE with $\theta = 10K$ vs $\theta = 500K$

PE	Books	CC	Wikipedia
RoPE	6.548	6.816	3.802
RoPE PI	6.341	6.786	3.775
RoPE ABF	**6.323**	**6.780**	**3.771**
xPos ABF	6.331	**6.780**	**3.771**

Effective Long-Context Scaling of Foundation Models [Xiong et al. 2023]

- Position interpolation (PI)

$$\mathbf{f}'(\mathbf{x}, m) = \mathbf{f}\left(\mathbf{x}, \frac{mL}{L'}\right)$$

Position interpolation: Rescale from new length L' to original length L

Extending Context Window of Large Language Models via Positional Interpolation [Chen et al. 2023]

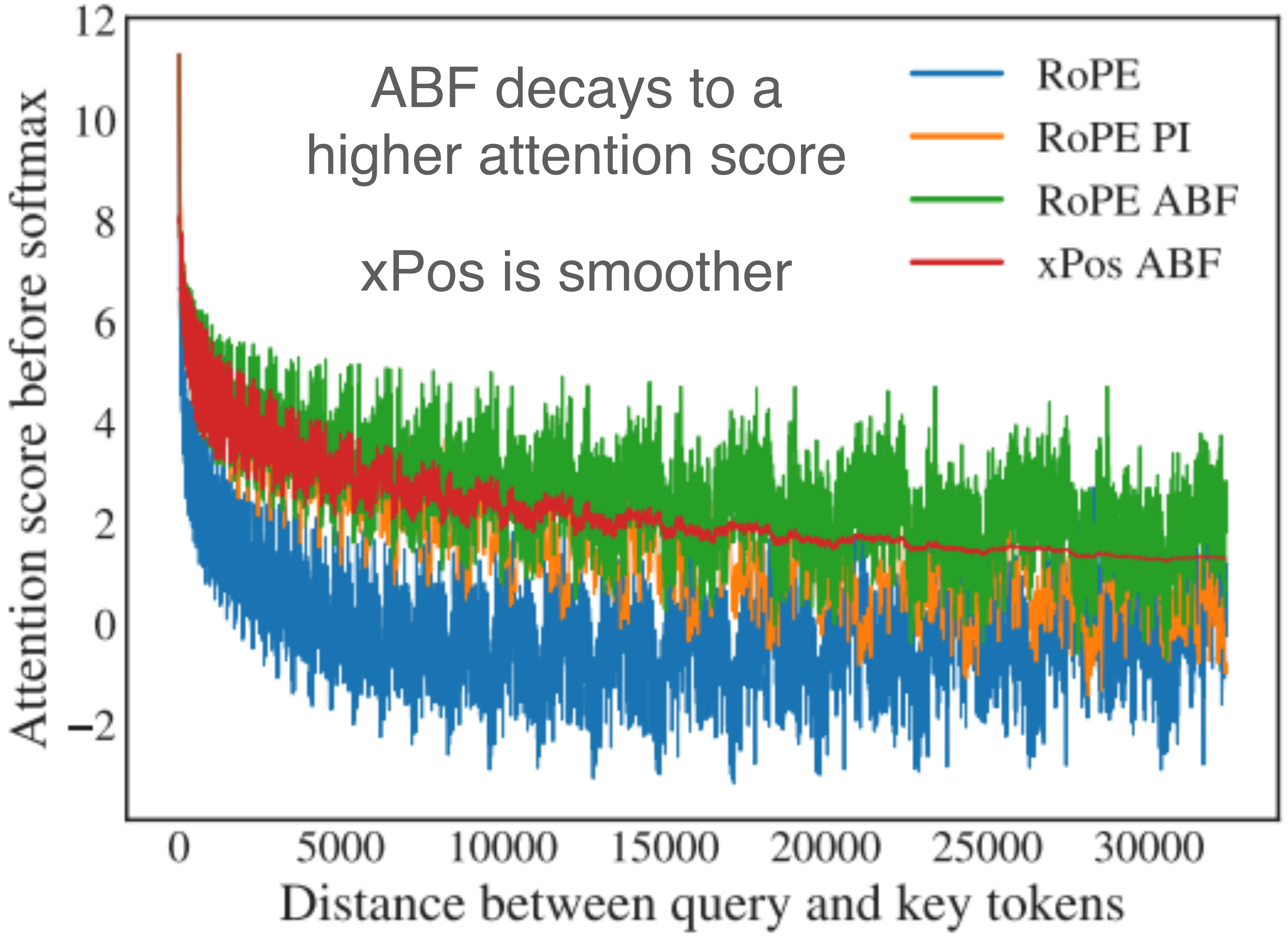
- xPos - Variant of RoPE

A length-extrapolatable transformer [Sun et al. 2022]

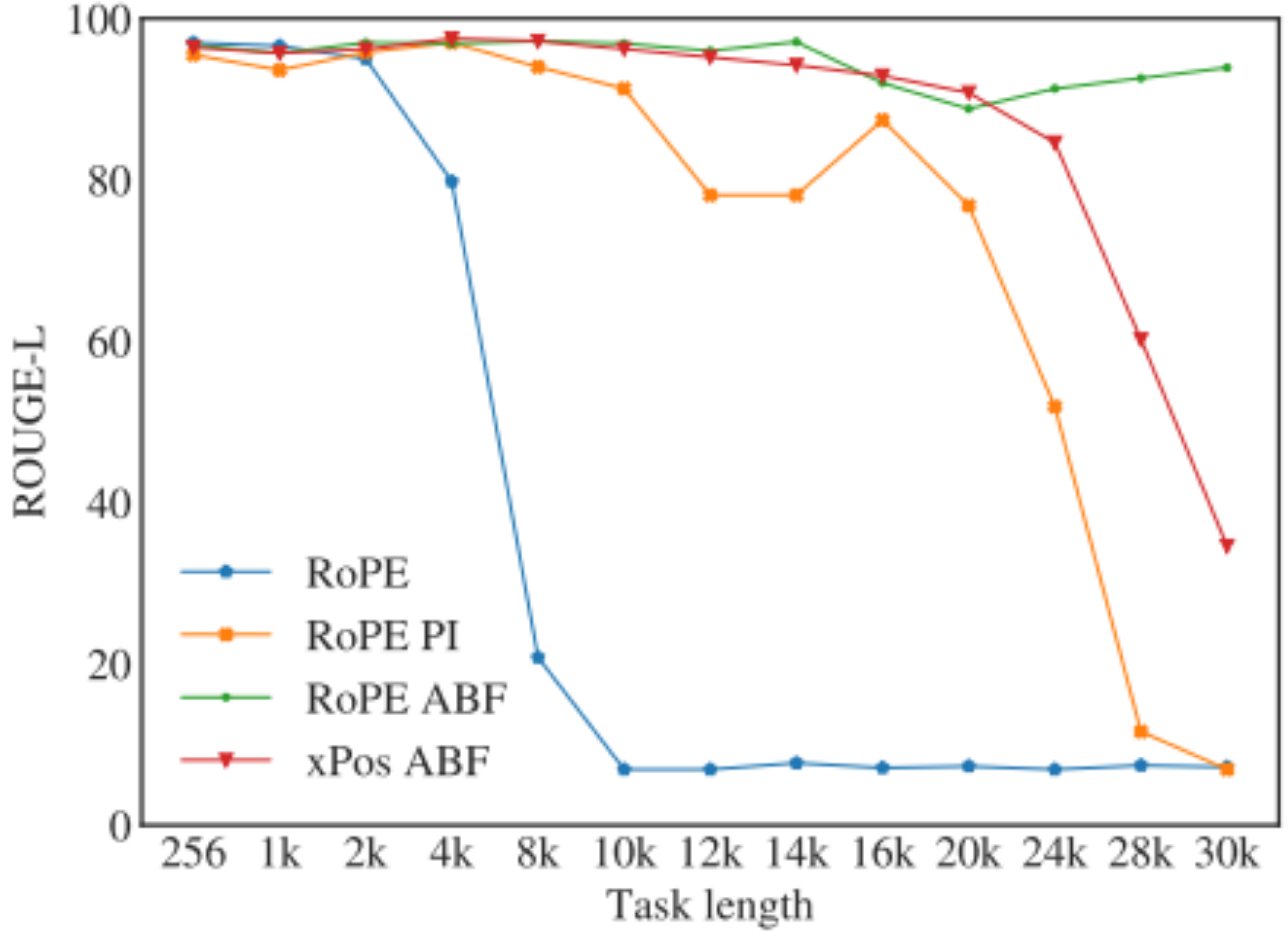
Current LLMs use RoPE

Adjusted base frequency (ABF)

- Comparing RoPE with $\theta = 10K$ vs $\theta = 500K$



First sentence retrieval

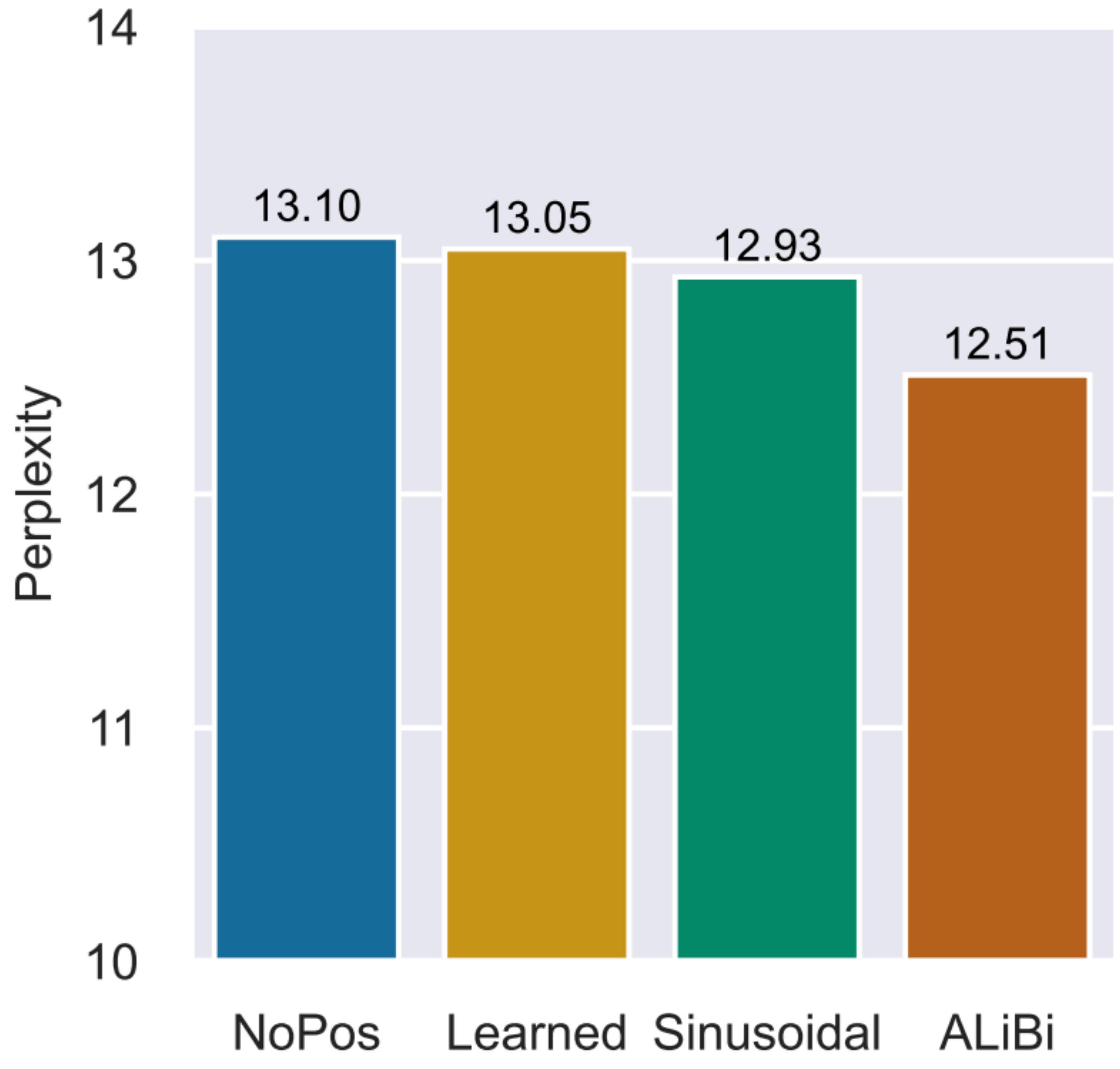


Effective Long-Context Scaling of Foundation Models [Xiong et al. 2023]

How important is positional encoding?

Can we remove them?

- Train 1.3B parameter autoregressive LM on the Pile
- LM perplexity comparable to learned embedding
- Hypothesis: causal attention allows for encoding of absolute position of tokens

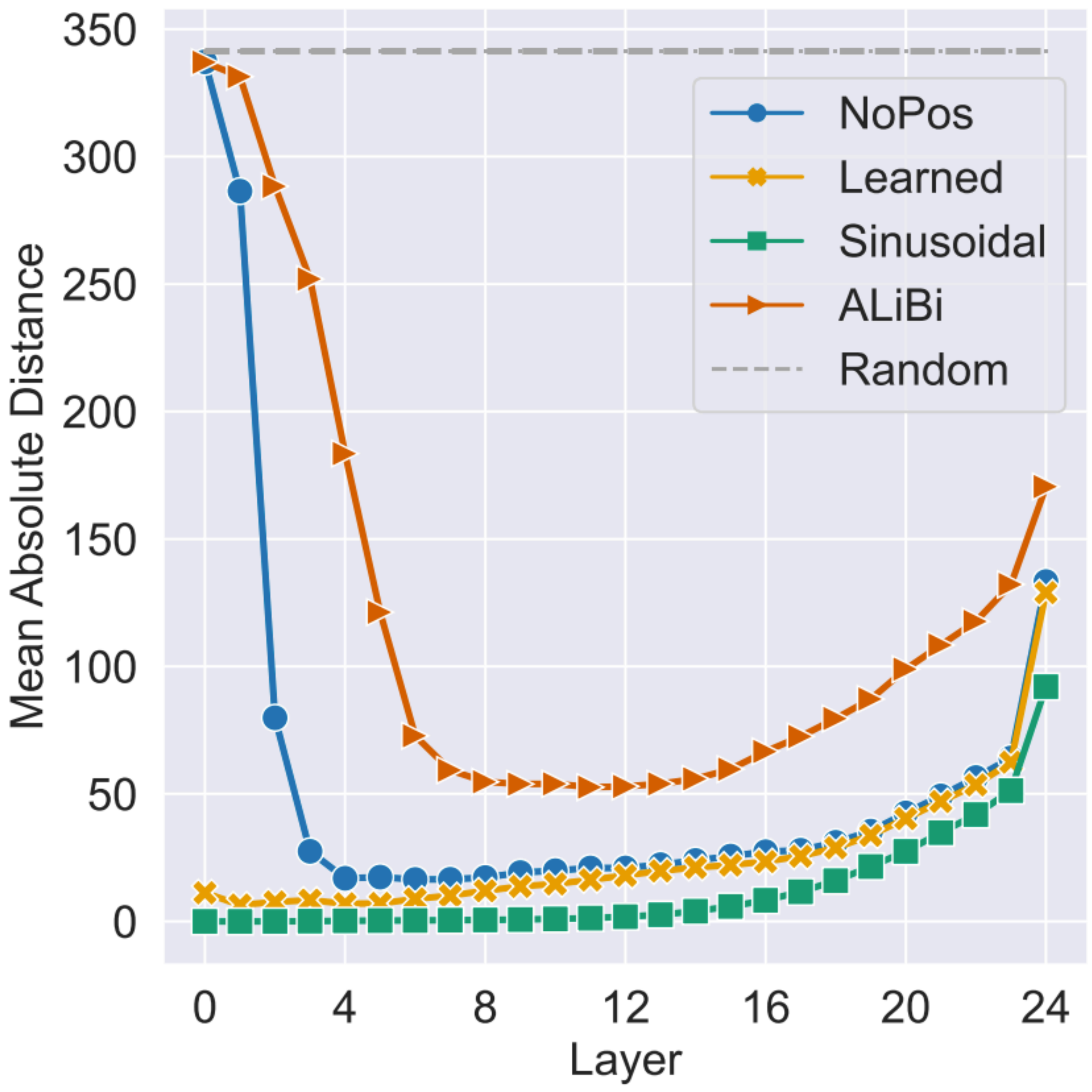


Transformer Language Models without Positional Encodings Still Learn Positional Information [Haviv et al. 2022]

How important is positional encoding?

Can we remove them?

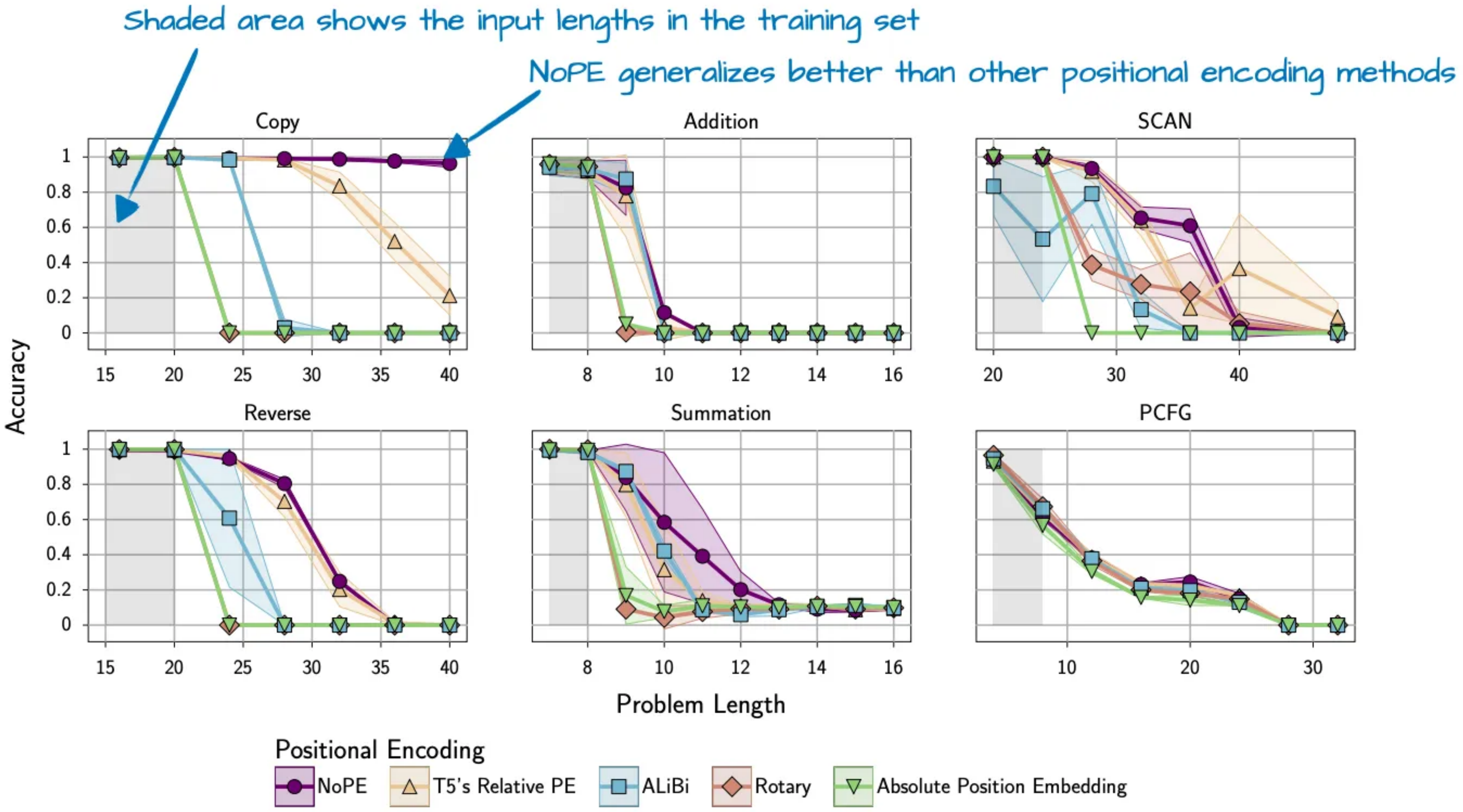
- Model can learn to encode absolute position
- Train 2-layer FF ReLU network (hooked up to transformer layer output representations) to predict absolute position (0 to 1023) of each token (as multi class classification)
- LM weights are frozen
- Later layers can predict position



Transformer Language Models without Positional Encodings Still Learn Positional Information [Haviv et al. 2022]

NoPE (no positional embedding)

- Interleave NoPE layers with layers with positional embeddings



The Impact of Positional Encoding on Length Generalization in Transformers [Kazemnejad et al. 2023]

(Annotated version from https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison)

Revisiting transformers

- Positional encoding
- **Normalization**
  - Layer normalization vs RMS
  - Post vs Pre-Layer norm
- Activation functions
- Attention variations

Add & Norm

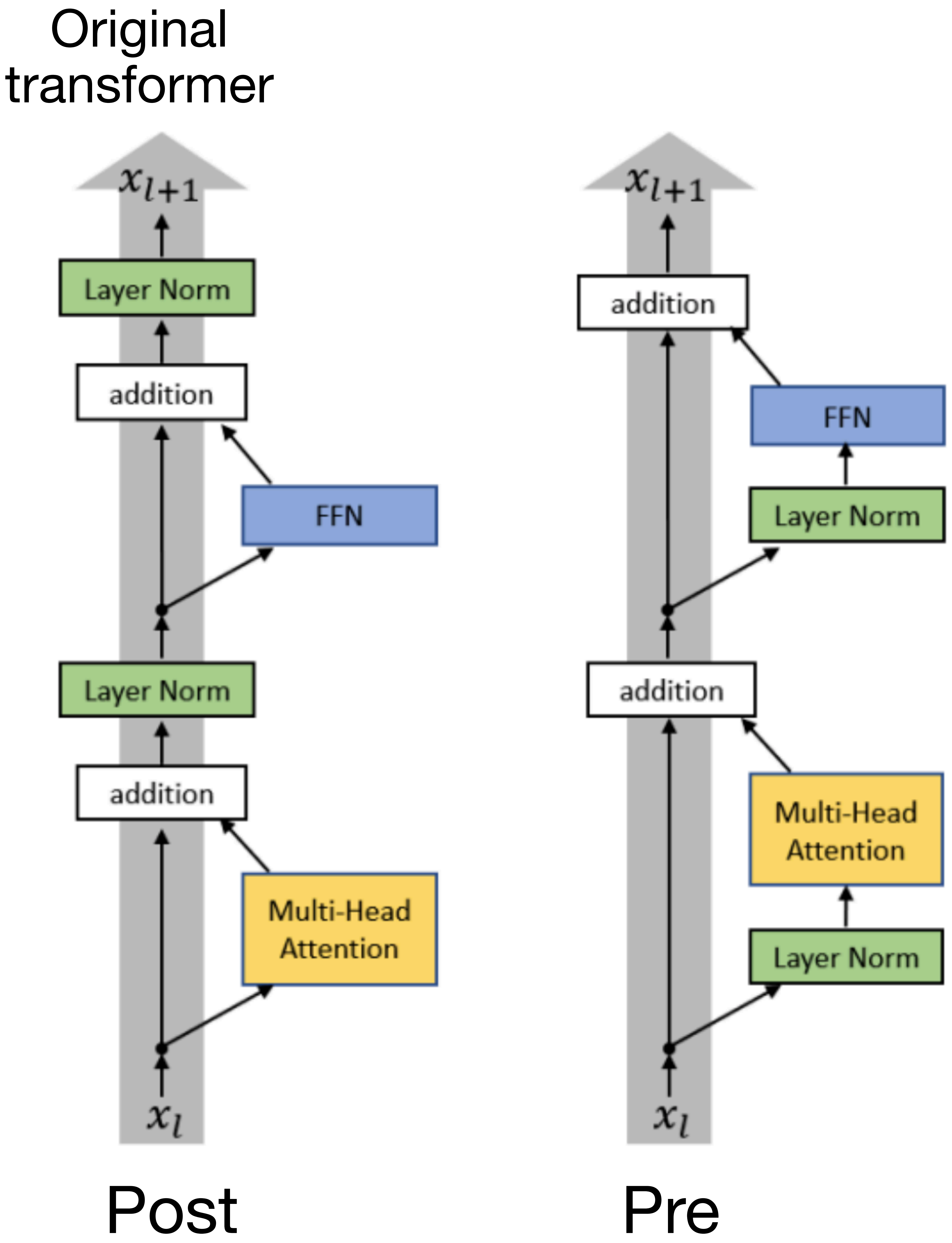
Residual Connections and Layer Norm

- Combine residual connection and layer norm into a single "Add & Norm" component
- Three choices:
  - Pre-norm (input): $\mathbf{z}^{\ell+1} = f(\mathrm{LN}(\mathbf{z}^{\ell})) + \mathbf{z}^{\ell}$ https://arxiv.org/abs/2002.04745
  - Res-Post-norm (output): $\mathbf{z}^{\ell+1} = \mathrm{LN}(f(\mathbf{z}^{\ell})) + \mathbf{z}^{\ell}$ https://arxiv.org/pdf/2111.09883
    - (Does not combine well with dropout)
  - Post-norm: $\mathbf{z}^{\ell+1} = \mathrm{LN}(f(\mathbf{z}^{\ell}) + \mathbf{z}^{\ell})$ Original transformer
- Pre-norm leads to faster training. Res-Post-norm has less spikes.

Layer normalization

Pre- vs Post- layer normalization

- Moving layer normalization to before the MHA and FFN block helps improve training stability
- Warm-up of learning rate is critical for training when using Post-LN

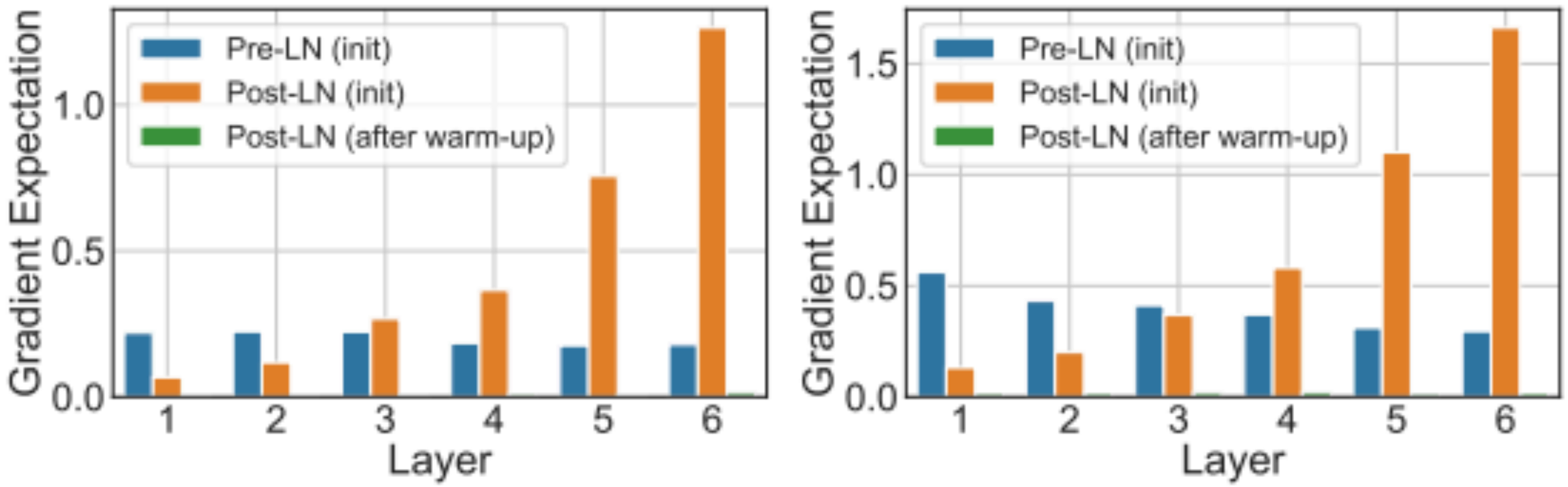


On Layer Normalization in the Transformer Architecture [Xiong et al. 2020]

Layer normalization

Pre- vs Post- layer normalization

- Warmup very important for Post-LN
- Gradient for Post-LN varies across layers during warmup, and is very small after warmup



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers

On Layer Normalization in the Transformer Architecture [Xiong et al. 2020]

Replace LayerNorm with RMSNorm

LayerNorm

- changes input features to have mean 0 and variance 1 per layer.
- Adds two more learnable parameters (vectors)
  - Gain **g** and bias **b**

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} x_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i^l - \mu^l)^2}$$

$$h_i = \frac{g_i}{\sigma_i}(x_i - \mu_i) + b_i$$

RMSNorm

- Simplifies LayerNorm by removing the mean and bias terms

$$\mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

$$\mathrm{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\mathrm{RMS}(\mathbf{x})} \cdot \mathbf{g}$$

Root Mean Square Layer Normalization
[Zhang and Sennrich, 2019]

Revisiting transformers

- Positional encoding
- Normalization
    - Layer normalization vs RMS
    - Post vs Pre-Layer norm
- **Activation functions**
- Attention variations

Activation function

Gated linear unit (GLU)

- Output of a linear transformation of x is modulated by a gate

$$\text{GLU} = \sigma(xW) \otimes xV$$

$$\text{SwiGLU} = \text{Swish}(xW) \otimes xV$$

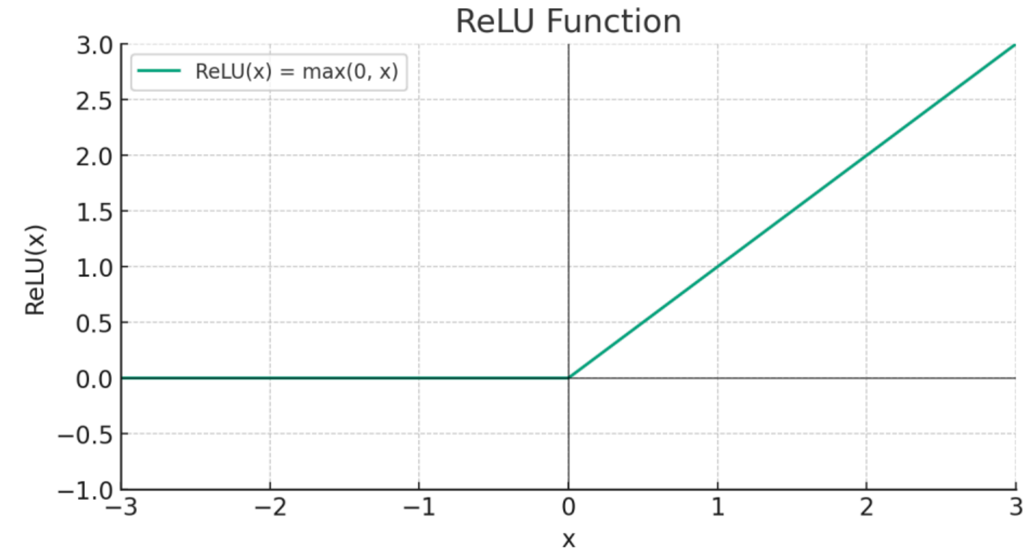
Squared ReLU

Outperforms ReLU, competitive with SwiGLU

- Sparse computations
- Used in Primer

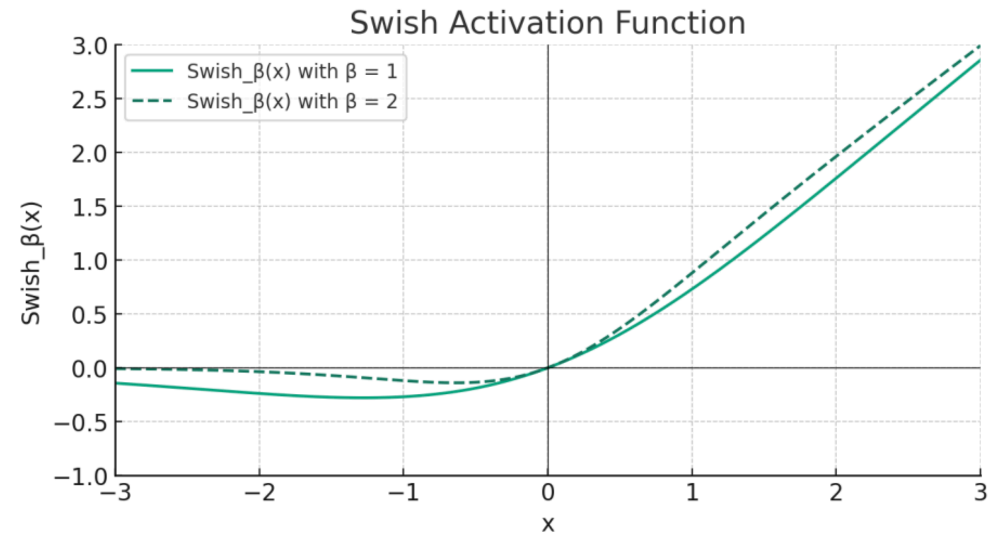
- Vaswani et al.: ReLU

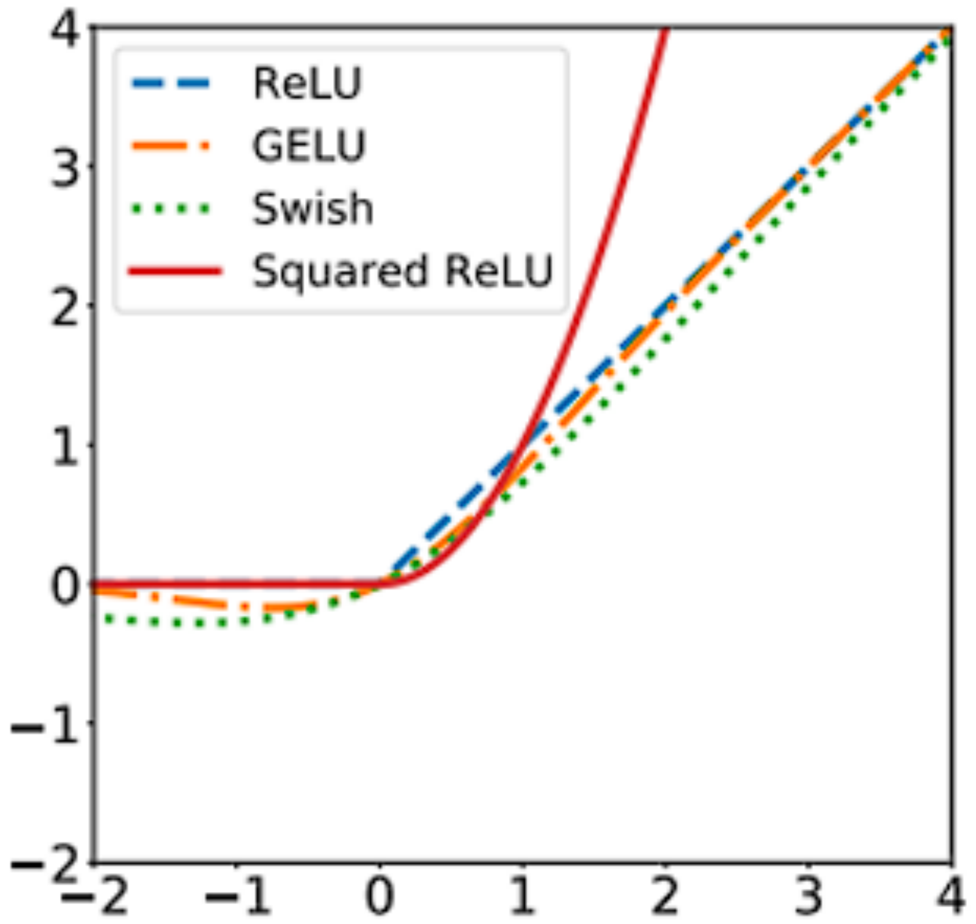
$$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$$



- LLaMa: Swish/SiLU (Hendricks and Gimpel 2016)

$$\text{Swish}(\mathbf{x}; \beta) = \mathbf{x} \odot \sigma(\beta \mathbf{x})$$





Revisiting transformers

- Positional encoding
- Normalization
  - Layer normalization vs RMS
  - Post vs Pre-Layer norm
- Activation functions
- **Attention variations**

QK norm

- Normalize query and key to unit vectors

$$A(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

→

γ is learnable parameter

$$A(Q, K, V) = \mathrm{softmax}\left(\gamma \cdot Q' K'^T\right)V$$

$$Q'_i = \frac{Q_i}{\|Q_i\|}, K'_j = \frac{K_j}{\|K_j\|}$$

- Variants using LayerNorm

- Training stability

Query-Key Normalization for Transformers [Henry et al. 2020]

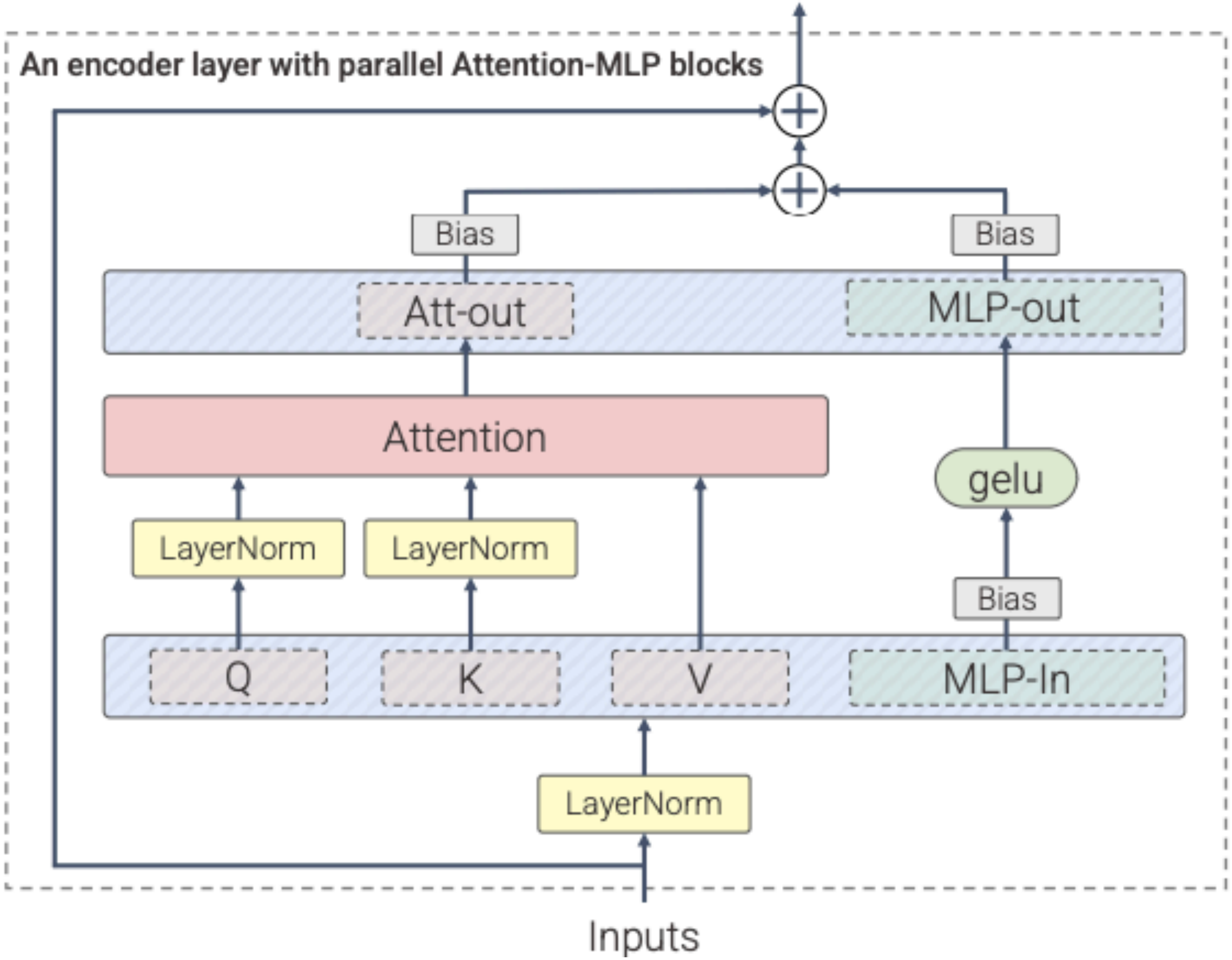
Parallel attention and QK norm

- Parallel attention

$$y' = \text{LayerNorm}(x),$$
$$y = x + \text{MLP}(y') + \text{Attention}(y').$$

- QK norm

$$\text{softmax}\left[\frac{1}{\sqrt{d}}\text{LN}(XW^Q)(\text{LN}(XW^K))^T\right]$$



Scaling Vision Transformers to 22 Billion Parameters [Dehghani et al. 2023]

QK norm

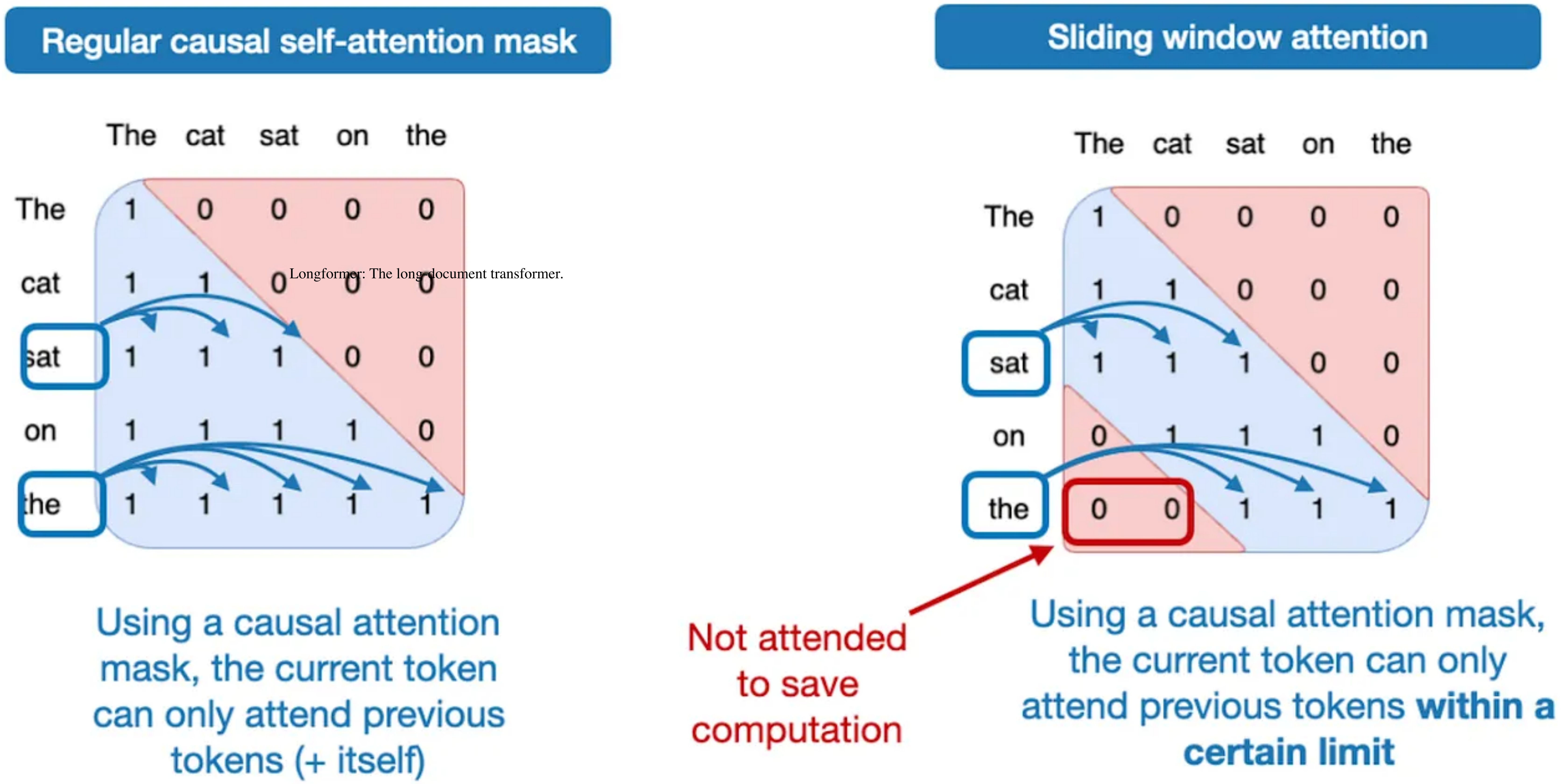
- Prevents exploding attention logits (which leads to almost one-hot attention weights with near-zero entropy)

Figure 1: Effect of query/key normalization on an 8B parameter model.

Scaling Vision Transformers to 22 Billion Parameters [Dehghani et al. 2023]

Sliding window attention

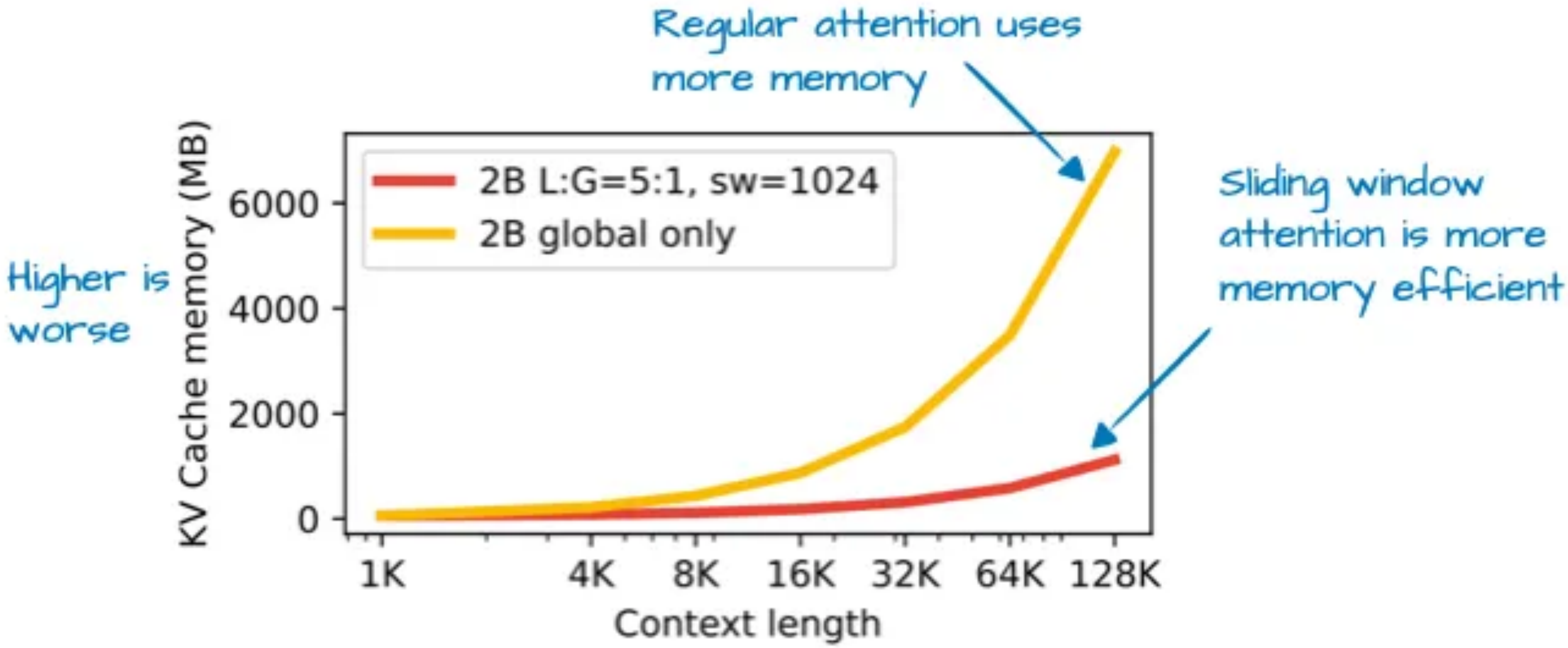
- Limited context / local attention
- Interleave with global attention layers



https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison

Longformer: The Long-Document Transformer [Beltagy et al. 2020]

Sliding window attention



Gemma 3 [Gemma Team. 2025]

(Annotated version from https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison)

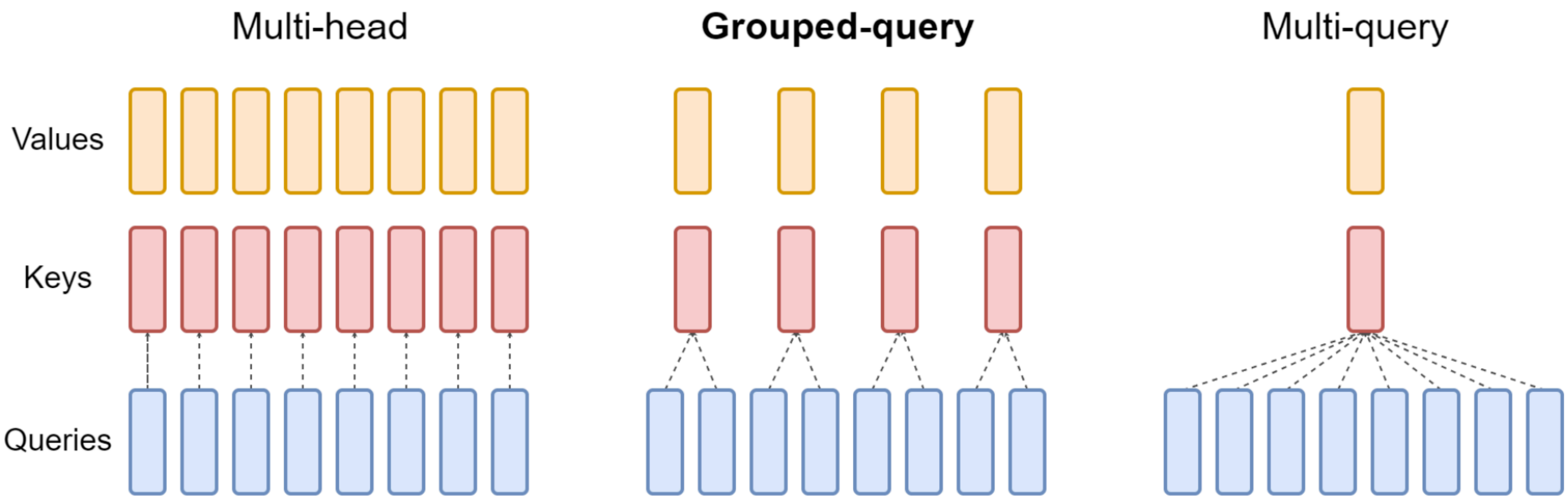
Sliding window attention

Gemma 3 [Gemma Team. 2025]

(Annotated version from https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison)

Grouped multi-query attention

- Reduce number of heads used for keys and values
- Shared values and keys across heads



Fast Transformer Decoding: One Write-Head is All You Need [Shazeer 2019]

GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints [Ainslie et al. 2023]

Multi-head latent attention (MLA)

$$c_n^{KV} = W^{DKV} x_n$$
$$K = W^{UK} C_{1:n}^{KV}$$
$$V = W^{UV} C_{1:n}^{KV}$$

- Use low rank matrix C^{KV}
- Project from latent space to full KV

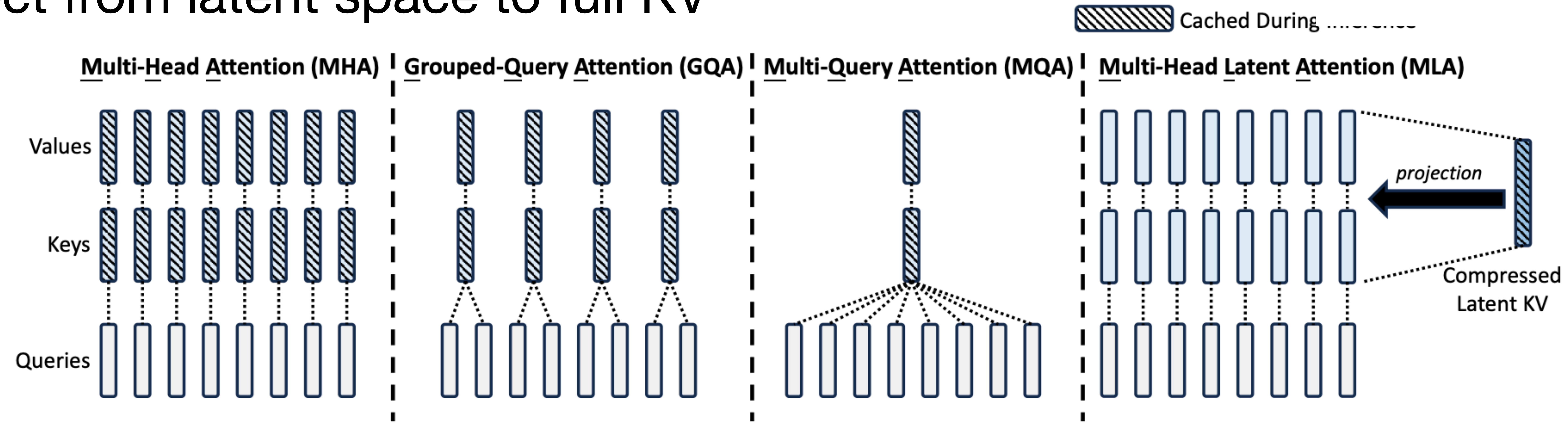
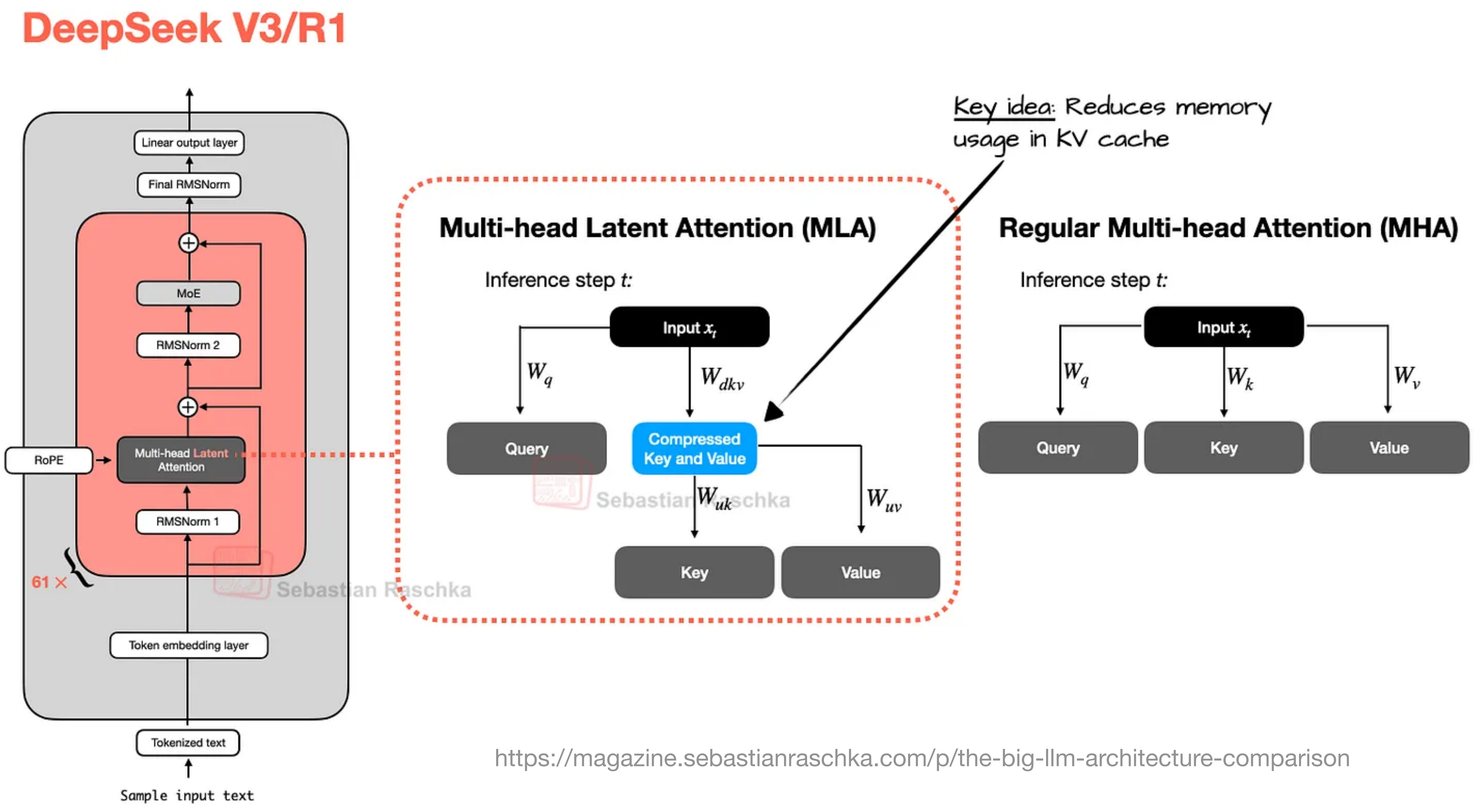


Figure 3 | Simplified illustration of Multi-Head Attention (MHA), Grouped-Query Attention (GQA), Multi-Query Attention (MQA), and Multi-head Latent Attention (MLA). Through jointly compressing the keys and values into a latent vector, MLA significantly reduces the KV cache during inference.

DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model [DeepSeek AI 2024]

Multi-head latent attention (MLA)



https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison

Gated value embedding

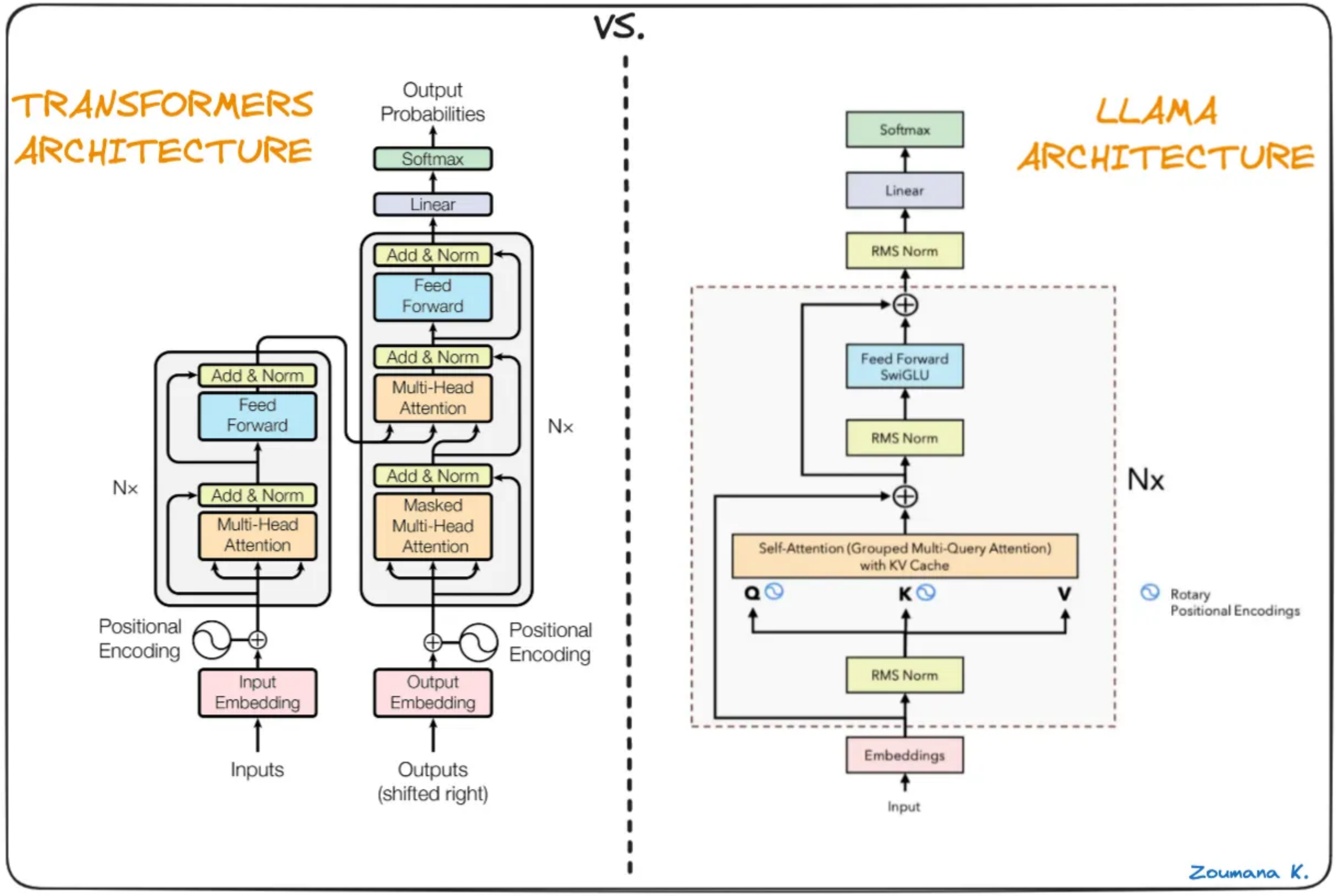
- Add gated value embedding (at alternating layers)
- Additional expressivity (more parameters) with minimal compute overhead

```
ve = value_embeds[layer_idx](token_ids)  # (B, T, kv_dim)
gate = 2 * sigmoid(ve_gate(x[:, :, :32]))  # range (0, 2)
v = v + gate * ve
```

https://github.com/karpathy/nanochat/discussions/481

LLaMa

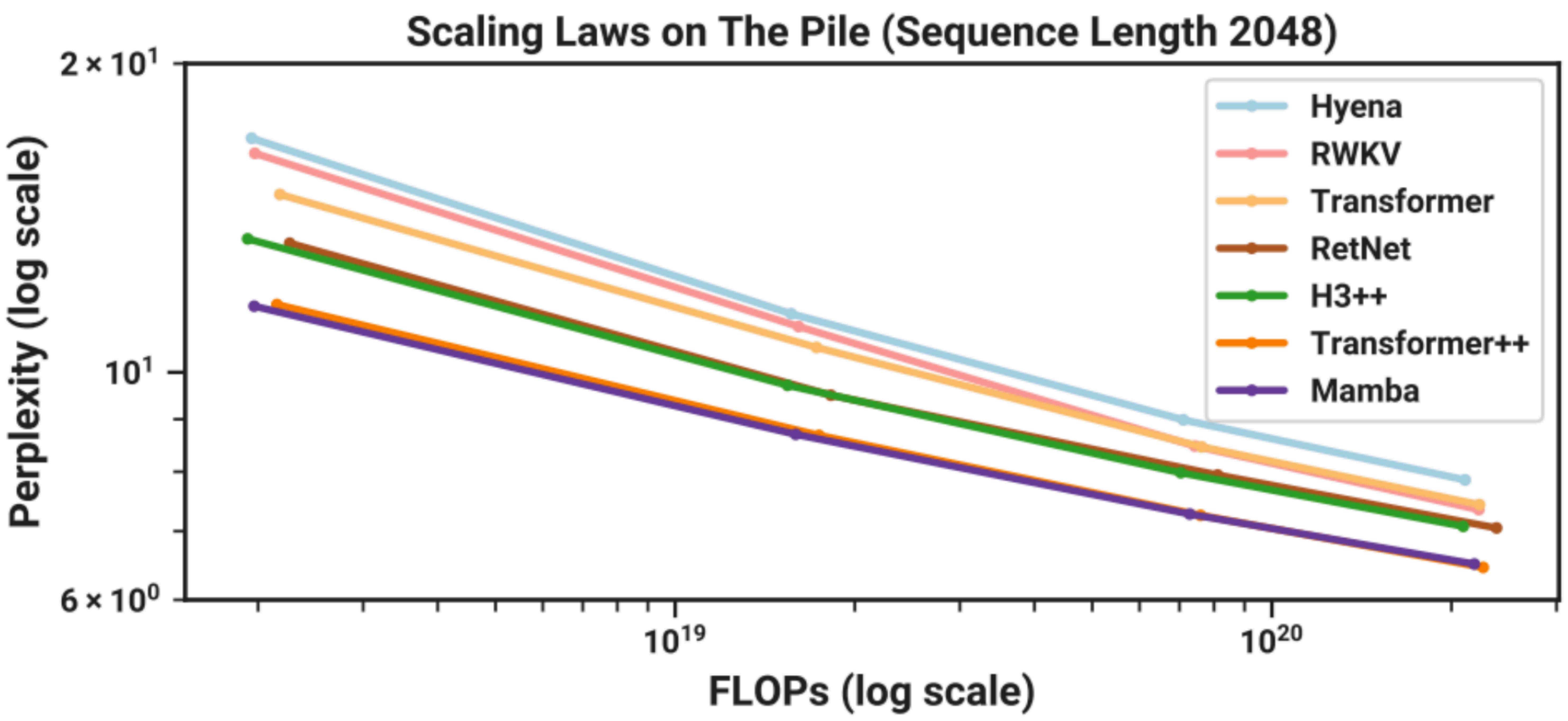
	Vaswani et al.	LLaMa
Norm Position	Post	Pre
Norm Type	LayerNorm	RMSNorm
Non-linearity	ReLU	SwiGLU
Positional Encoding	Sinosoidal	RoPE
Attention	Full Multi-Head Attention	Grouped Multi-Query Attention



https://pub.towardsai.net/llama-explained-a70e71e706e9

Impact of these changes

- Transformer: Vaswani et al.
- Transformer++: basically LLaMa



- Stronger architecture is ≈10x more efficient!

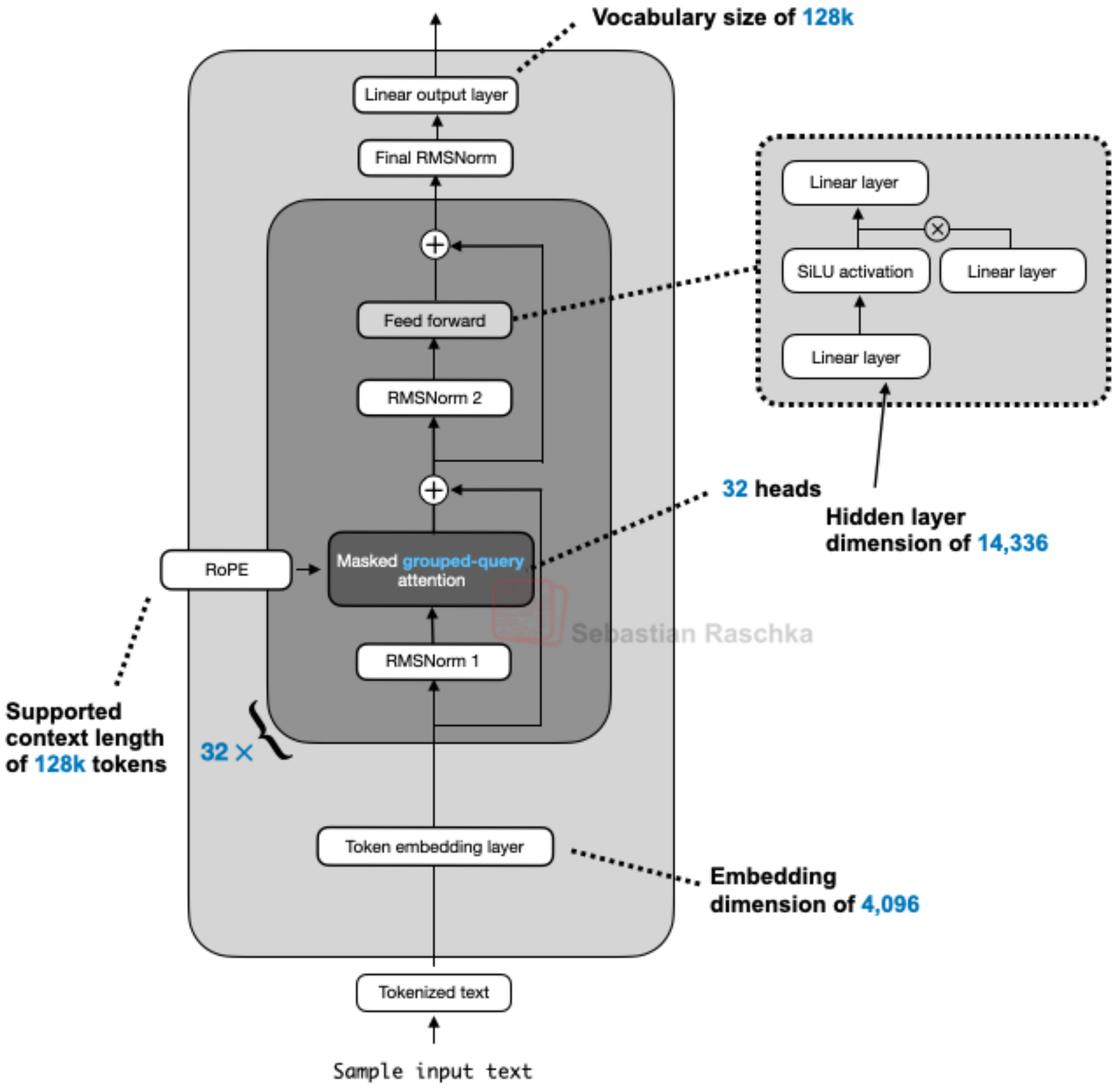
Figure from https://arxiv.org/pdf/2312.00752 [Gu and Dao, 2023]

OLMo 2 architecture

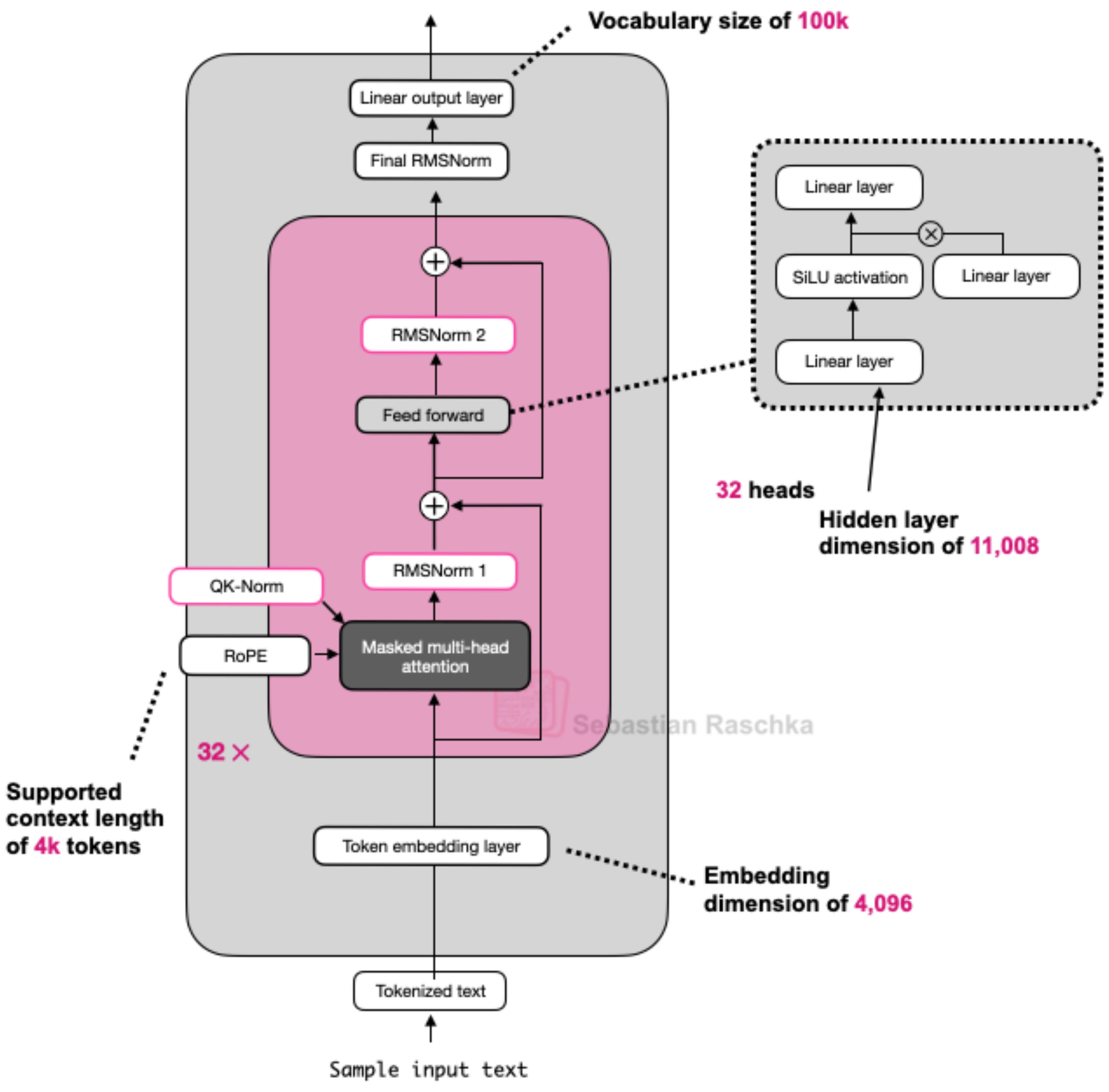
	OLMo 1 (0224)	OLMo-0424	OLMo 2
Biases	None	None	None
Activation	SwiGLU	SwiGLU	SwiGLU
RoPE θ	$1 \cdot 10^4$	$1 \cdot 10^4$	$5 \cdot 10^5$
QKV Normalization	None	Clip to 8	QK-Norm
Layer Norm	non-parametric	non-parametric	RMSNorm
Layer Norm Applied to	Inputs	Inputs	Outputs
Z-Loss Weight	0	0	10^{-5}
Weight Decay on Embeddings	Yes	Yes	No

2 OLMo 2 Furious [AI2 OLMo team, 2024]

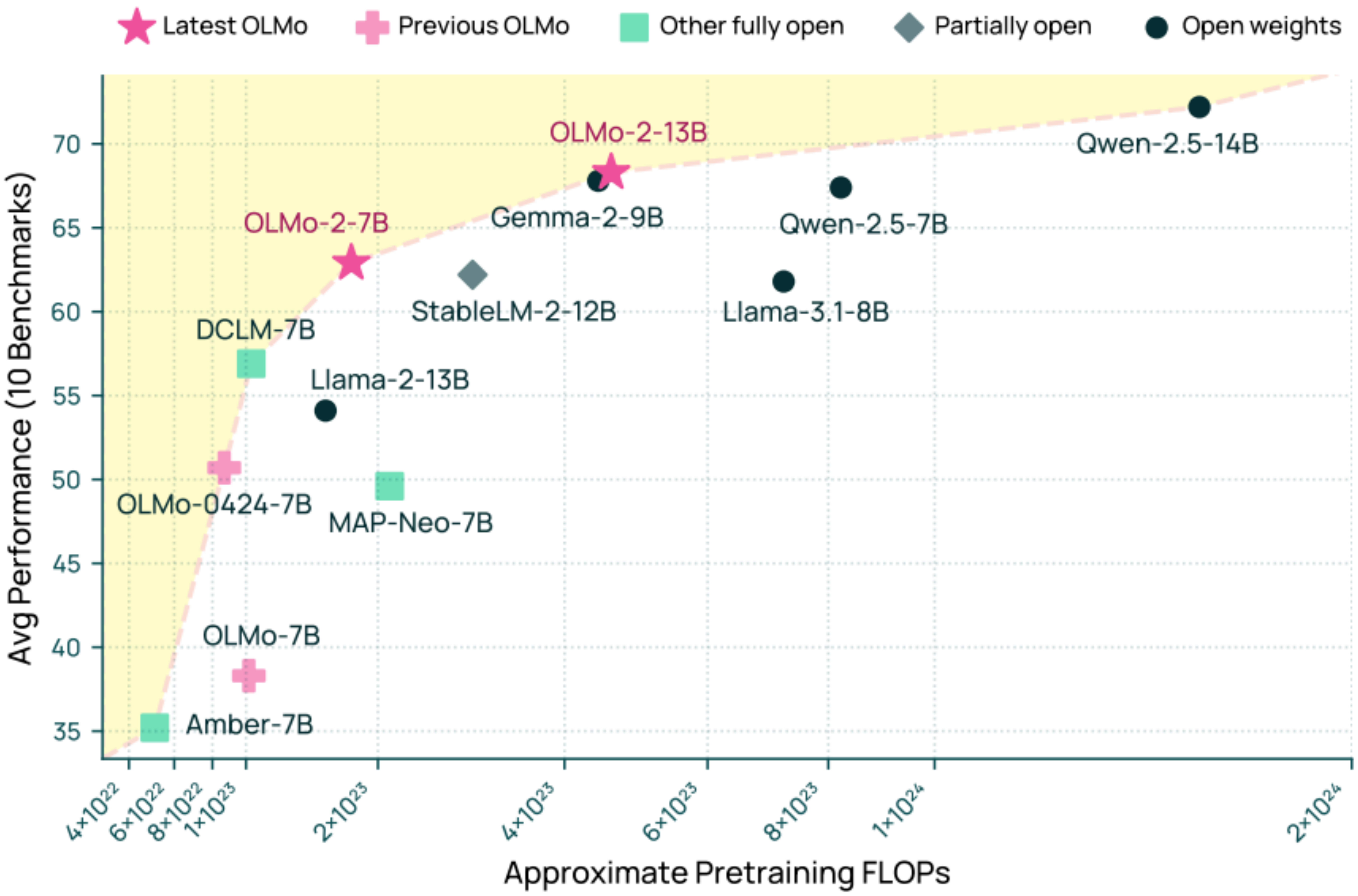
Llama 3 8B



OLMo 2 7B



Development of Open LLMs



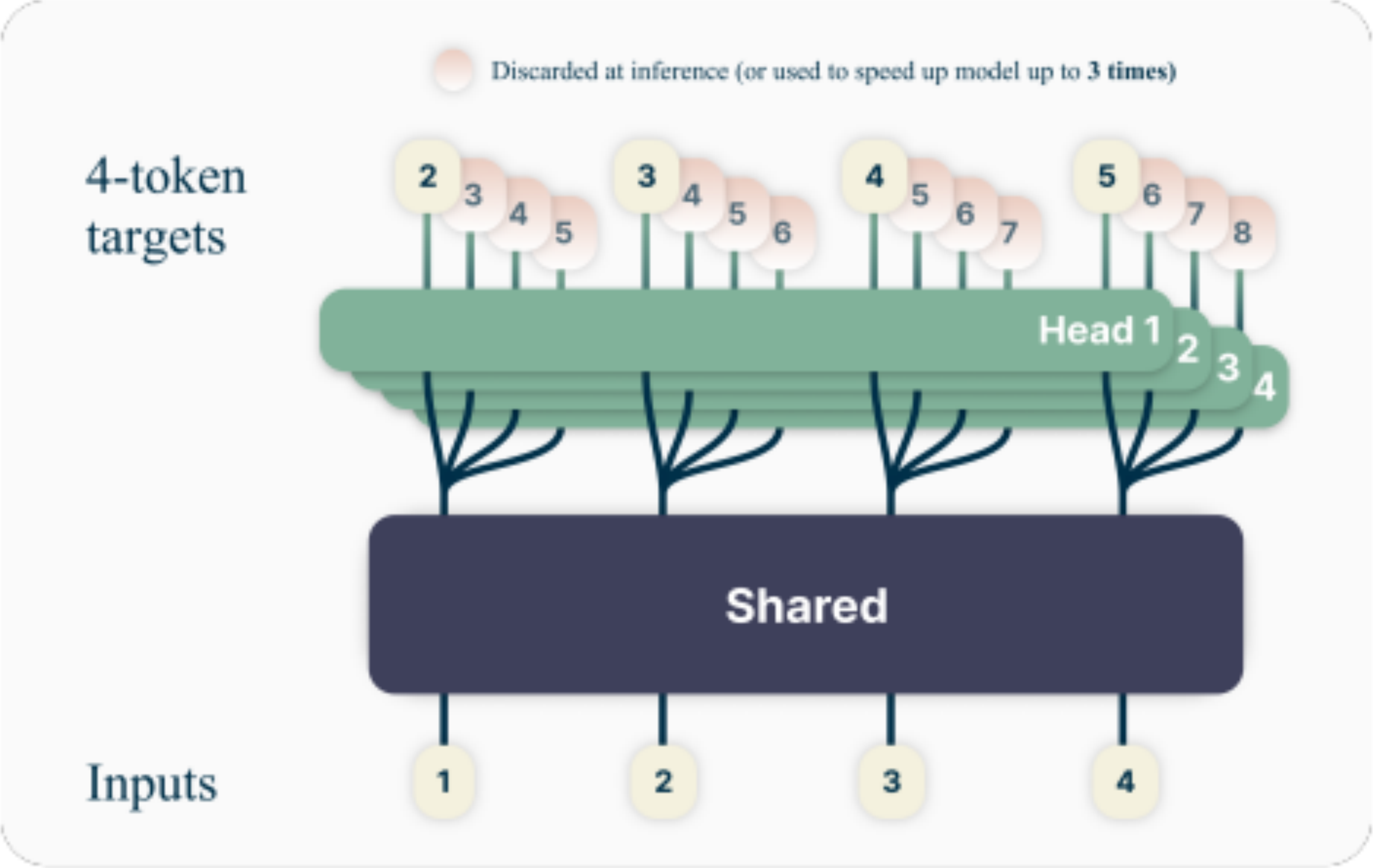
2 OLMo 2 Furious [AI2 OLMo team, 2024]

Other aspects of variation

- Training objective
- Scaling up with Mixture-of-Experts
- Optimizer
- Efficient training

Multi-token prediction

- Predict next tokens with shared trunk



- Gains due to multi-token prediction

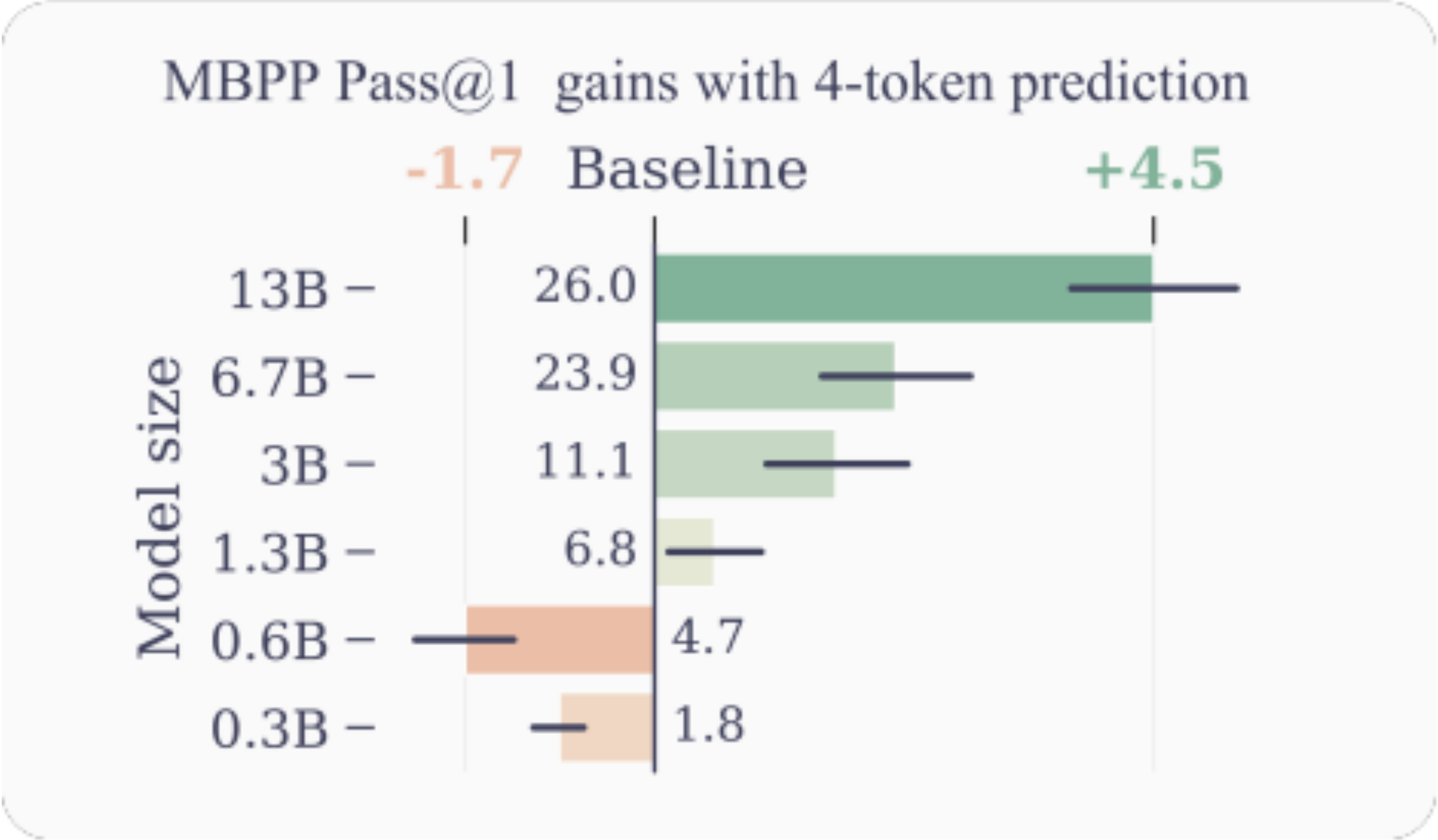


Figure from https://arxiv.org/pdf/2404.19737 [Gloeckle et al. 2024]

Multi-token prediction

- Predict in sequence to avoid large memory demands

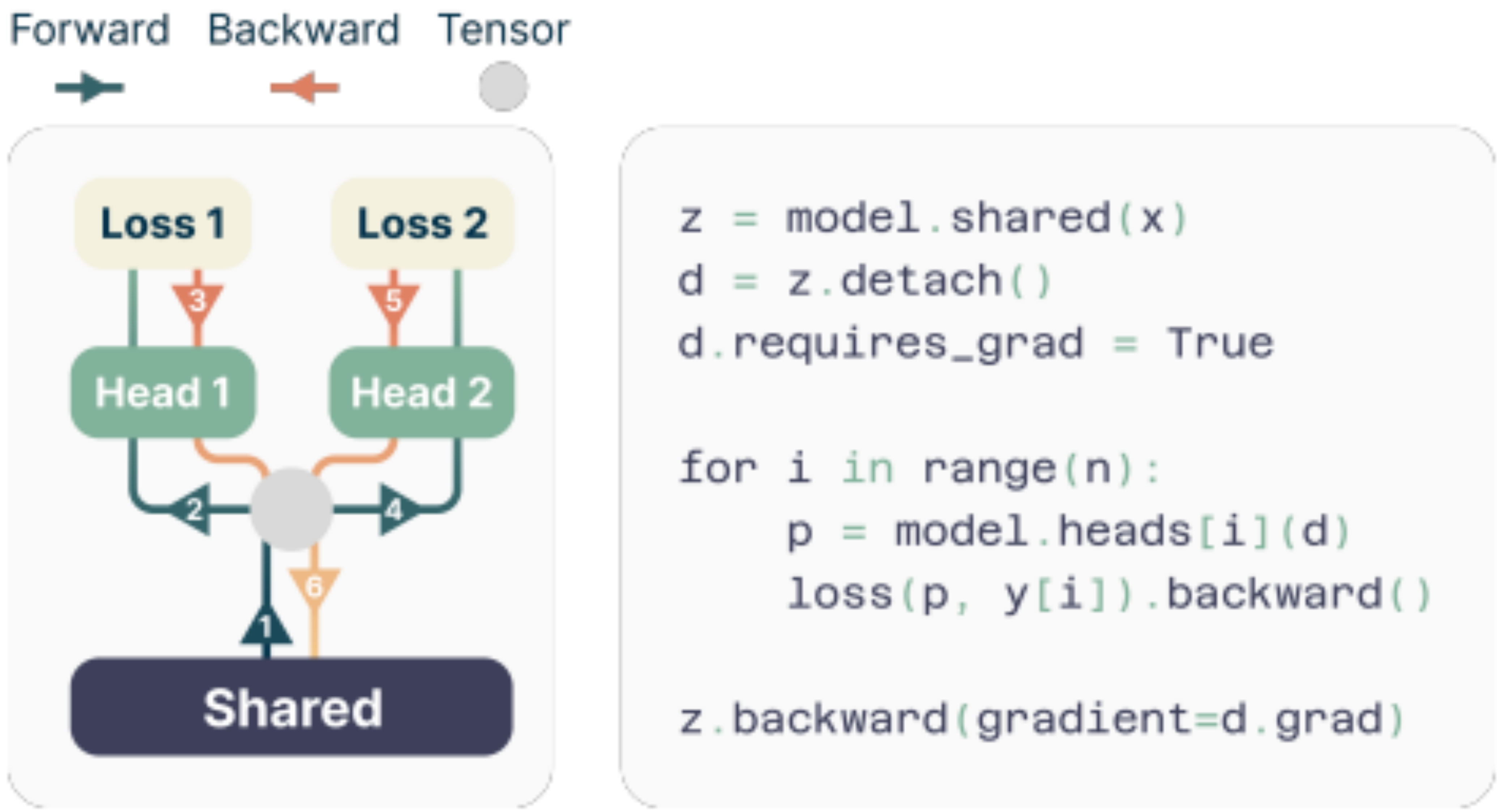


Figure 2: **Order of the forward/backward in an n-token prediction model with $n = 2$ heads.** By performing the forward/backward on the heads in sequential order, we avoid materializing all unembedding layer gradients in memory simultaneously and reduce peak GPU memory usage.

Figure from https://arxiv.org/pdf/2404.19737 [Gloeckle et al. 2024]

Multi-token prediction

- D sequential modules for predicting D tokens

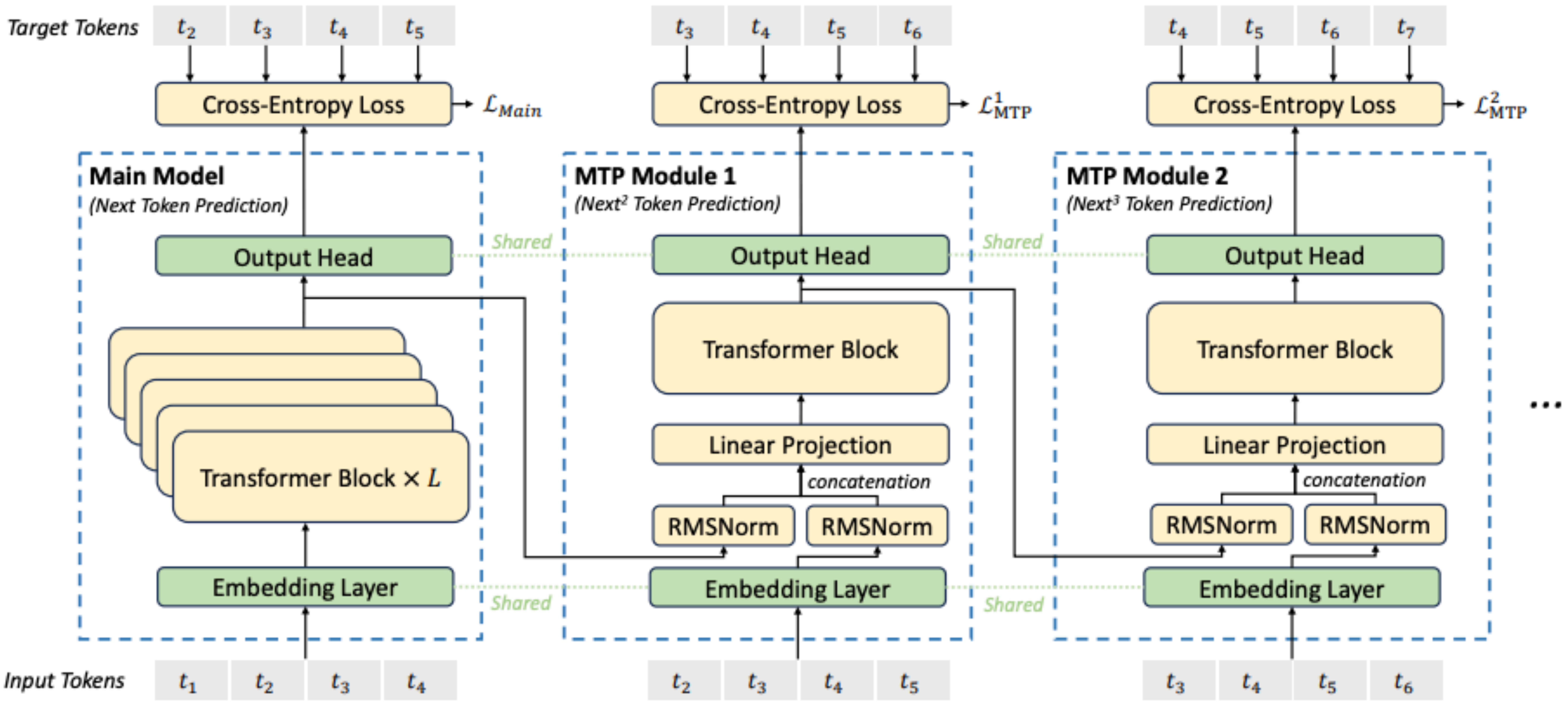
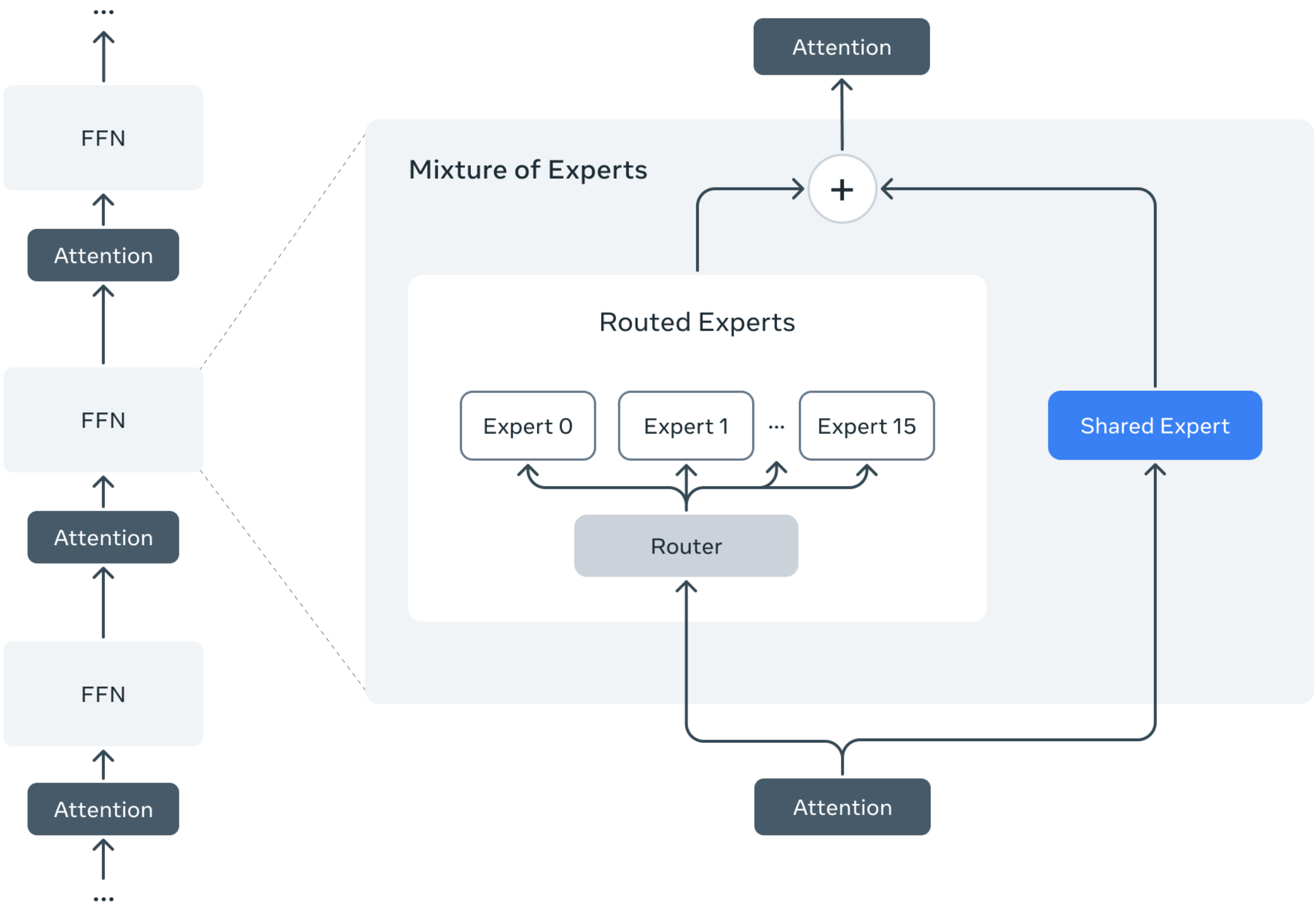


Figure from https://arxiv.org/pdf/2412.19437 [DeepSeek-V3 2025]

Mixture of Experts

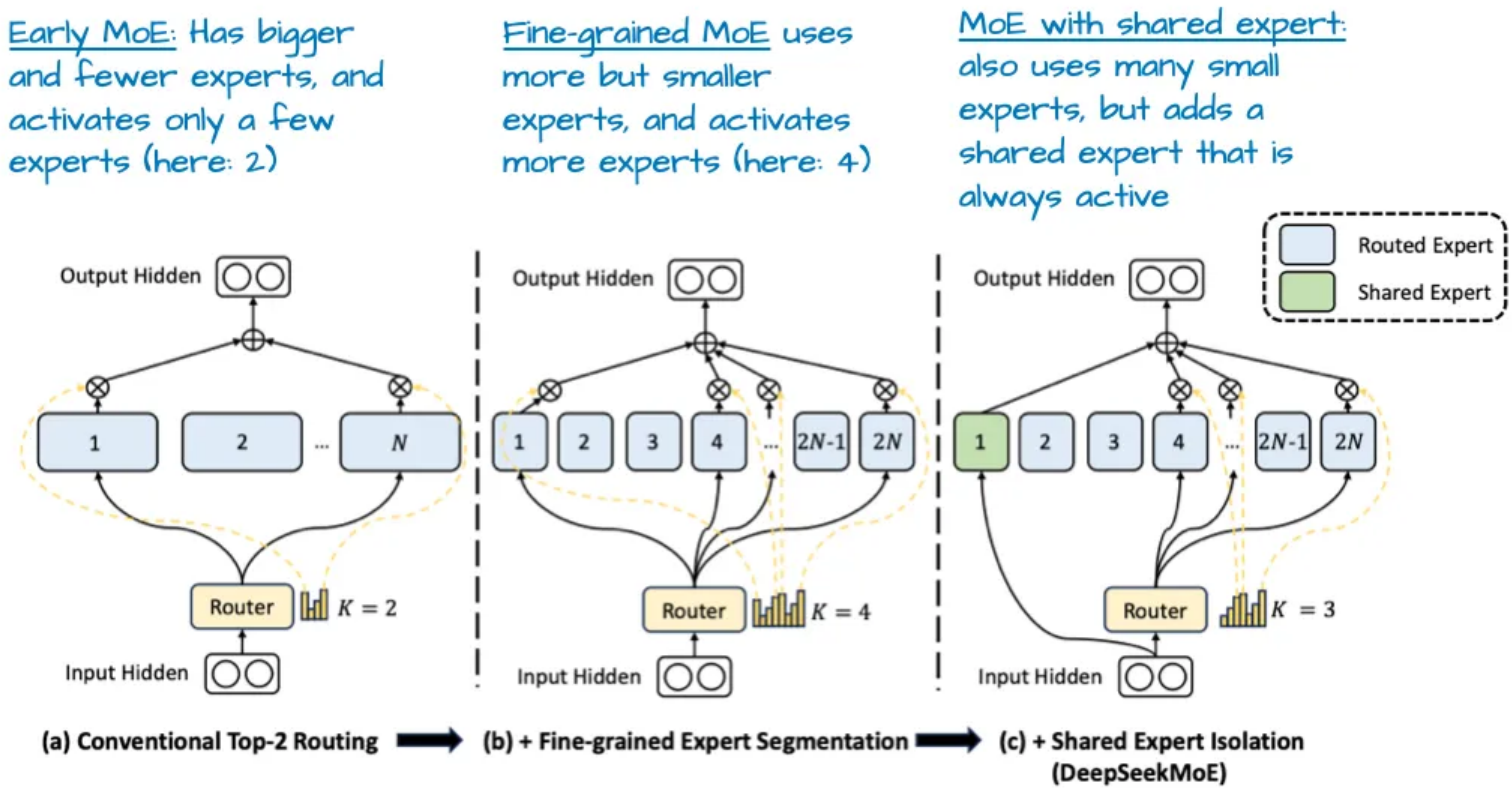
LLama 4 - 17B active parameters, 400B total parameters
Alternating dense and MoE layers
MOE layer: 128 routed expert, 1 shared

- Allows for large capacity with low computational cost per token
- Router that selects which expert (individual neural networks) to use



https://ai.meta.com/blog/llama-4-multimodal-intelligence/

Mixture of experts over time

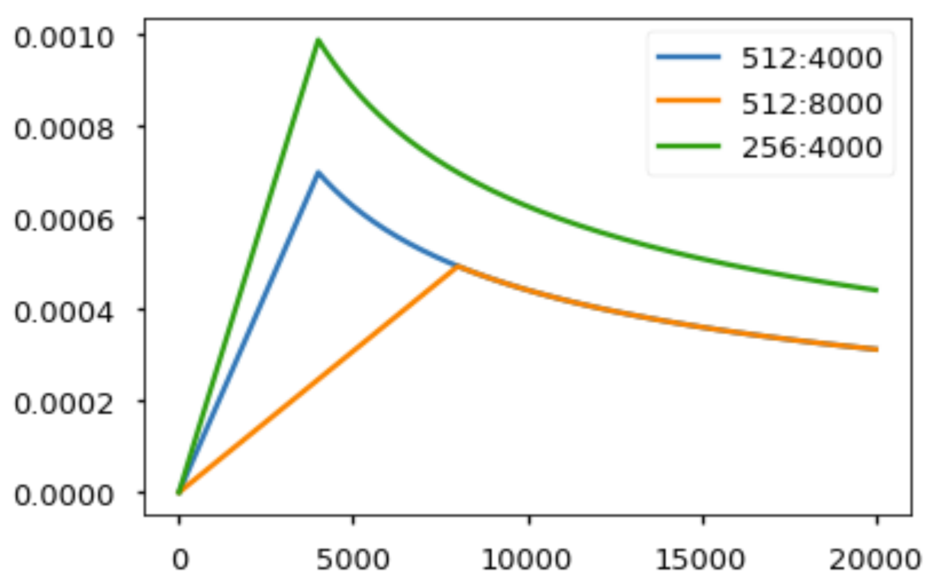


DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models [Dai et al. 2024]
(Annotated version from https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison)

Optimizer

- SGD: Update in the direction of reducing loss
- Adam: Add momentum and normalize by the stddev of the outputs
- AdamW: properly applies weight decay for regularization to Adam
- Adam with learning rate schedule:

$$\text{learning\_rate} = d_{\text{model}}^{-0.5} \cdot \min(\text{step\_num}^{-0.5}, \text{step\_num}^{-0.5} \cdot \text{warmup\_steps}^{-1.5})$$



Muon Optimizer

- Orthogonalize the update

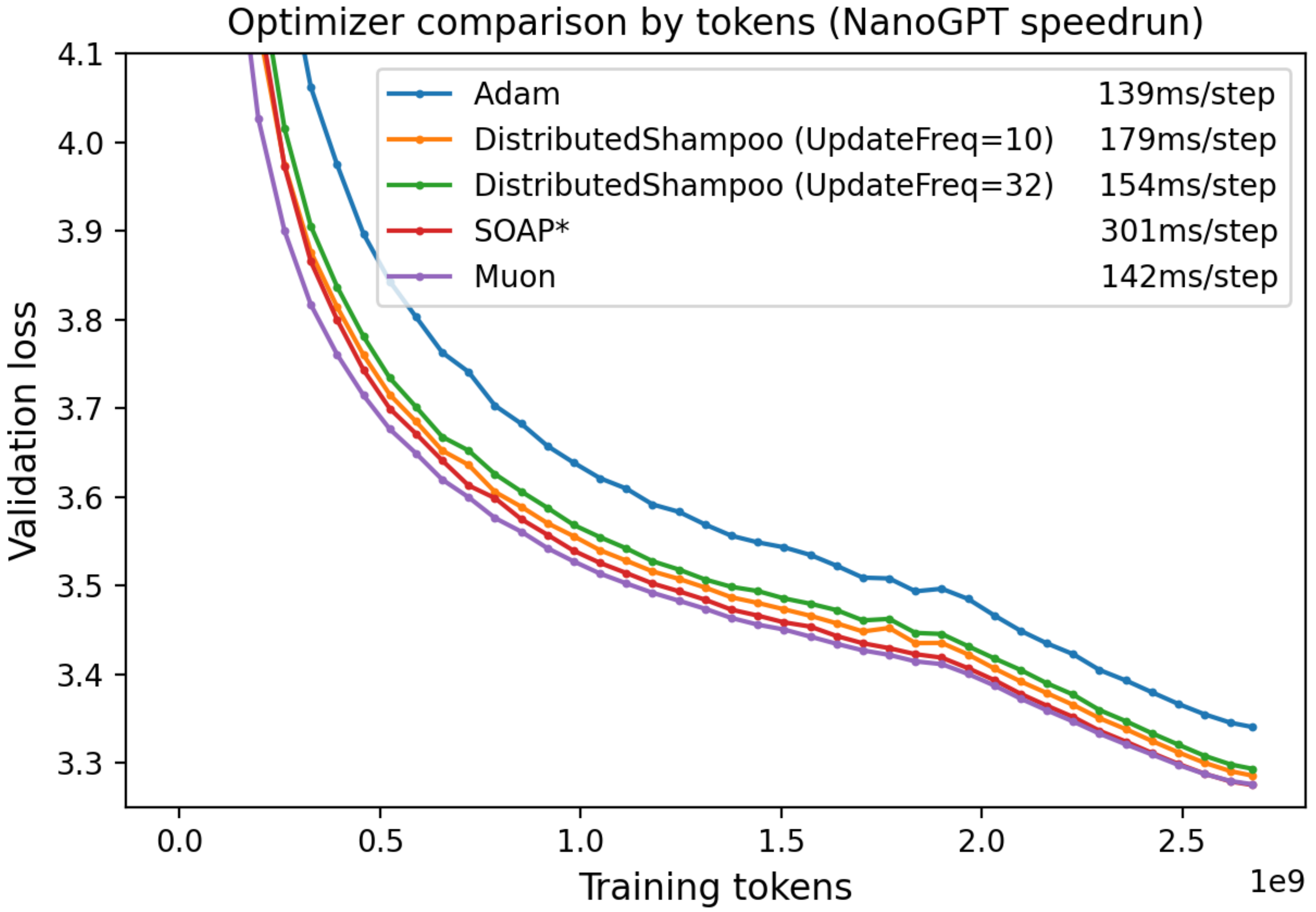
Algorithm 2 Muon

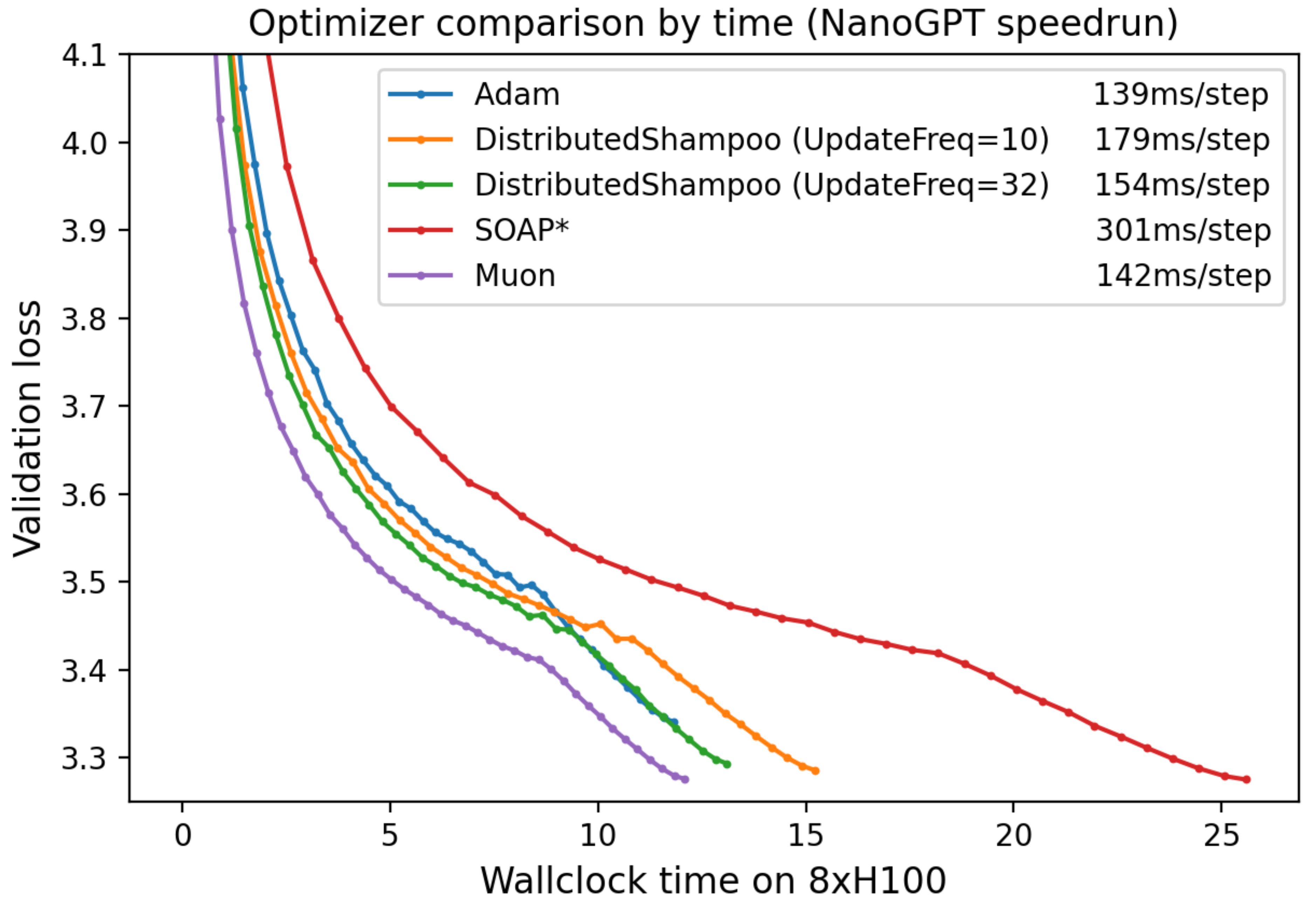
Require: Learning rate η, momentum μ

Initialize $B_0 \leftarrow 0$
for $t = 1, \ldots$ **do**
  Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
  $B_t \leftarrow \mu B_{t-1} + G_t$
  $O_t \leftarrow \text{NewtonSchulz5}(B_t)$
  Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
end for
return θ_t

- Use for weight matrices, use AdamW for embeddings

https://kellerjordan.github.io/posts/muon/





*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

Resources

- LLM architectures
    - https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison
    - https://magazine.sebastianraschka.com/p/a-dream-of-spring-for-open-weight
- Andrej Karpathy - NanoGPT journey
    - https://github.com/karpathy/nanochat/discussions/481