Lecture 3

Everything you didn't want to know about LM architecture and training

CS336

Tatsu H

Logistics

- Join the slack!

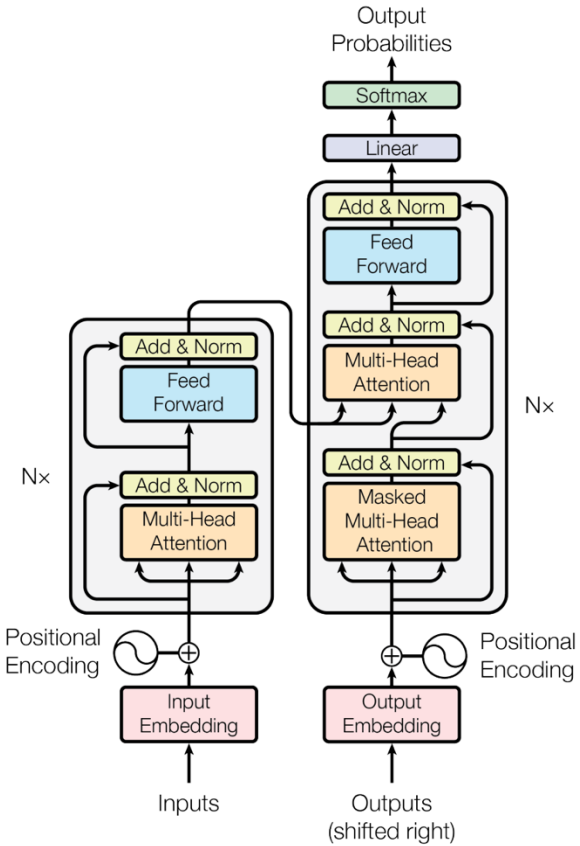
- Check to make sure you have the latest version of the assignment!

Outline and goals

- Quick recap of the ‘standard’ transformer (what you implement)
- What do most of the large LMs have in common?
- What are common variations to the architecture / training process?

Today’s theme: the best way to learn is hands-on experience
the second best way is to try to learn from others’ experience

Starting point: the 'original' transformer



Review: choices in the standard transformer

Position embedding: sines and cosines

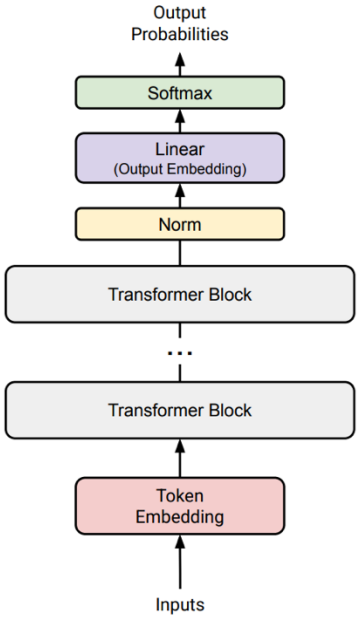
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

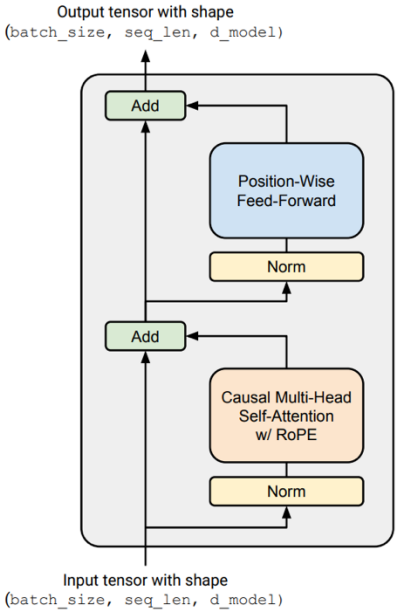
FFN: ReLU

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Norm type: post-norm, LayerNorm

What you implemented – simple, modern variant





Differences:

- **LayerNorm** is in front of the block
- **Rotary position embeddings (RoPE)**
- FF layers use **SwiGLU**, not ReLU
- Linear layers (and layernorm) have **no bias** (constant) terms

Why did we pick these?

What should you pick?

H…tures?

L…

FALCON2-11B TECHNICAL REPORT

Quentin Malartic* Nilabhra Roy Chowdhury Ruxandra Cojocaru Mugariya Farooq
… Sanath Narayan Ankit Singh
Mohammed Al-Yafeai Hamza Alobeidli
Kirill Fedyanin Reda Alami
…abi, United Arab Emirates

Meta

The Llama 3 Herd of Models

Llama Team, AI @ Meta[1]
[1]A detailed contributor list can be found in the appendix of this paper.

Nemotron-4 340B Technical Report

NVIDIA

Reka Core, Flash, and Edge: A Series of Powerful Multimodal Language Models

Aitor Ormazabal Che Zheng Cyprien de Masson d'Autume Dani Yogatama
Deyu Fu Donovan Ong Eric Chen Eugenie Lamprecht Hai Pham Isaac Ong
Kaloyan Aleksiev Lei Li Matthew Henderson Max Bain Mikel Artetxe
Nishant Relan Piotr Padlewski Qi Liu Ren Chen Samuel Phua
Yazheng Yang Yi Tay Yuqi Wang Zhongkai Zhu Zhihui Xie

Qwen2.5 Technical Report

Qwen Team

https://huggingface.co/Qwen

QWEN2 TECHNICAL REPORT

2025-03-12

Over 19 new *dense* model releases, many of them with minor architecture tweaks..

…Shane Arora* Akshita Bhagia* Yuling Gu* She…

Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan

Qwen Team, Alibaba Group*

…f a Small Language Model

…uch* Gabriel Martín Blázquez* Guilherme Penedo
…liček Agustín Piqueres Lajarín Vaibhav Srivastav
…uyen Clémentine Fourrier Ben Burtenshaw
… Cyril Zakka Mathieu Morlon

InternLM2 Technical Report

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, Dahua Lin

Shanghai AI Laboratory
SenseTime Group
The Chinese University of Hong Kong
Fudan University
internlm@pjlab.org.cn

Google DeepMind 2024-06-27

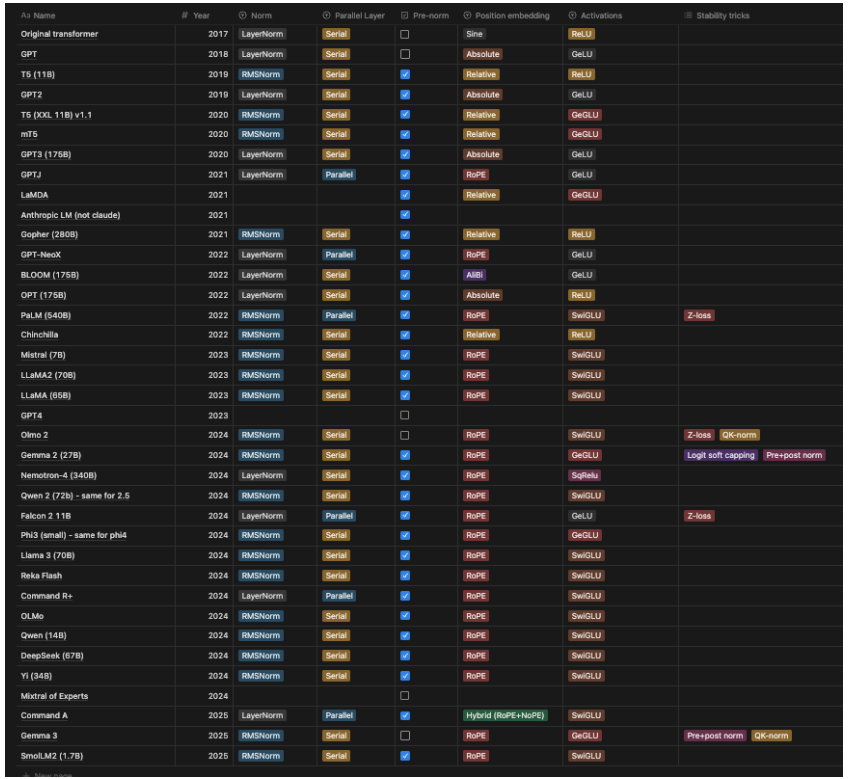
Gemma 2: Improving Open Language Models at a Practical Size

Gemma Team, Google DeepMind[1]

Phi-3 Tech…
…le Language…
Microsoft

Let’s look at the data (on dense architectures)

Learn from the many other models (and papers) out there



Name	Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	Stability tricks
Original transformer	2017	LayerNorm	Serial	☐	Sine	ReLU	
GPT	2018	LayerNorm	Serial	☐	Absolute	GeLU	
T5 (11B)	2019	RMSNorm	Serial	☑	Relative	ReLU	
GPT2	2019	LayerNorm	Serial	☑	Absolute	GeLU	
T5 (XXL 11B) v1.1	2020	RMSNorm	Serial	☑	Relative	GeGLU	
mT5	2020	RMSNorm	Serial	☑	Relative	GeGLU	
GPT3 (175B)	2020	LayerNorm	Serial	☑	Absolute	GeLU	
GPTJ	2021	LayerNorm	Parallel	☑	RoPE	GeLU	
LaMDA	2021			☑	Relative	GeGLU	
Anthropic LM (not claude)	2021			☑			
Gopher (280B)	2021	RMSNorm	Serial	☑	Relative	ReLU	
GPT-NeoX	2022	LayerNorm	Parallel	☑	RoPE	GeLU	
BLOOM (175B)	2022	LayerNorm	Serial	☑	AliBi	GeLU	
OPT (175B)	2022	LayerNorm	Serial	☑	Absolute	ReLU	
PaLM (540B)	2022	RMSNorm	Parallel	☑	RoPE	SwiGLU	Z-loss
Chinchilla	2022	RMSNorm	Serial	☑	Relative	ReLU	
Mistral (7B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU	
LLaMA2 (70B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU	
LLaMA (65B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU	
GPT4	2023			☐			
Olmo 2	2024	RMSNorm	Serial	☐	RoPE	SwiGLU	Z-loss QK-norm
Gemma 2 (27B)	2024	RMSNorm	Serial	☑	RoPE	GeGLU	Logit soft capping Pre+post norm
Nemotron-4 (340B)	2024	LayerNorm	Serial	☑	RoPE	SqRelu	
Qwen 2 (72b) - same for 2.5	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Falcon 2 11B	2024	LayerNorm	Parallel	☑	RoPE	GeLU	Z-loss
Phi3 (small) - same for phi4	2024	RMSNorm	Serial	☑	RoPE	GeGLU	
Llama 3 (70B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Reka Flash	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Command R+	2024	LayerNorm	Parallel	☑	RoPE	SwiGLU	
OLMo	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Qwen (14B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
DeepSeek (67B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Yi (34B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU	
Mixtral of Experts	2024			☐			
Command A	2025	LayerNorm	Parallel	☑	Hybrid (RoPE+NoPE)	SwiGLU	
Gemma 3	2025	RMSNorm	Serial	☐	RoPE	GeGLU	Pre+post norm QK-norm
SmolLM2 (1.7B)	2025	RMSNorm	Serial	☑	RoPE	SwiGLU	

We will talk through many major architecture and hyperparameter variants.

- What do all these models have in common?
- What parts vary?
- What can we learn from this?

What are we going to cover?

Common architecture variations

- Activations, FFN
- Attention variants
- Position embeddings

Hyperparameters that (do or don’t) matter

- What is ff_dim? Do multi_head dims always sum to model_dim?
- How many vocab elements?

Stability tricks

Architecture variations..

Let’s think about the core architecture piece

Name	Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations
Original transformer	2017	LayerNorm	Serial	☐	Sine	ReLU
GPT	2018	LayerNorm	Serial	☐	Absolute	GeLU
T5 (11B)	2019	RMSNorm	Serial	☑	Relative	ReLU
GPT2	2019	LayerNorm	Serial	☑	Sine	GeLU
T5 (XXL 11B) v1.1	2020	RMSNorm	Serial	☑	Relative	GeGLU
mT5	2020	RMSNorm	Serial	☑	Relative	GeGLU
GPT3 (175B)	2020	LayerNorm	Serial	☑	Sine	GeLU
GPTJ	2021	LayerNorm	Parallel	☑	RoPE	GeLU
LaMDA	2021			☑	Relative	GeGLU
Gopher (280B)	2021	RMSNorm	Serial	☑	Relative	ReLU
GPT-NeoX	2022	LayerNorm	Parallel	☑	RoPE	GeLU
BLOOM (175B)	2022	LayerNorm	Serial	☑	AliBi	GeLU
OPT (175B)	2022	LayerNorm	Serial	☑	Absolute	ReLU
PaLM (540B)	2022	RMSNorm	Parallel	☑	RoPE	SwiGLU
Chinchilla	2022	RMSNorm	Serial	☑	Relative	ReLU
Mistral (7B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA2 (70B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA (65B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
Qwen (14B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
DeepSeek (67B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Yi (34B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU

High level view:

- Low consensus (except pre-norm)
- Trends toward ‘LLaMA-like’ architectures

Pre-vs-post norm

The one thing *everyone* agrees on (in 2024)

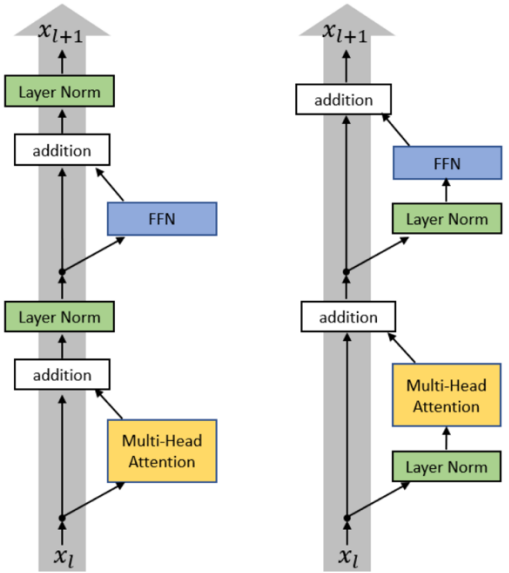


Figure from Xiong 2020

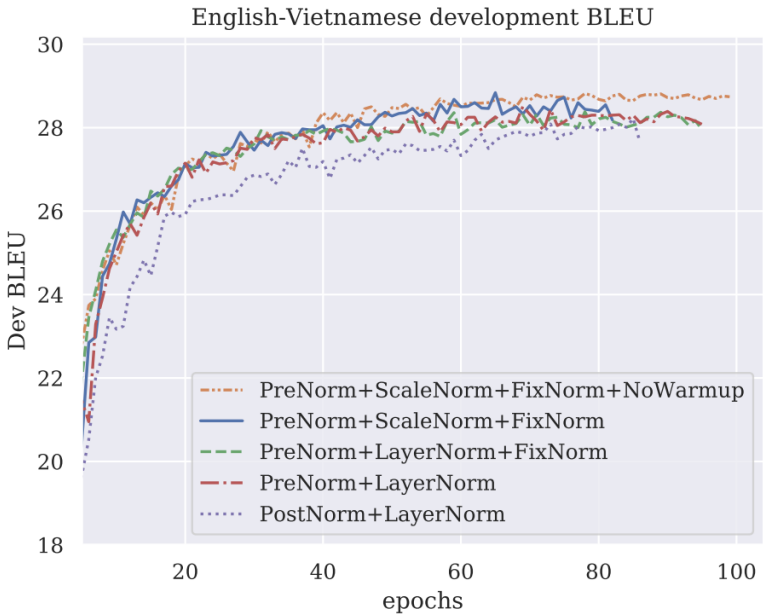
Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \cdots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x_{l+1,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l+1,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
	Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{L+1,i}^{pre})$

Set up LayerNorm so that it doesn't affect the main residual signal path (on the left)

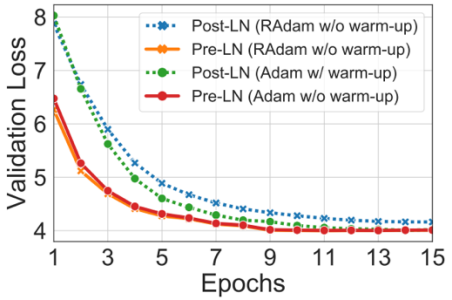
Almost all modern LMs use pre-norm (but BERT was post-norm)

(One somewhat funny exception – OPT350M. I don't know why this is post-norm)

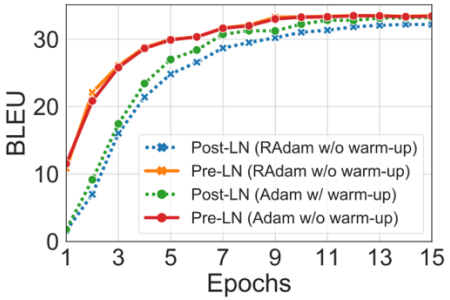
Pre-vs-post-norm, the data



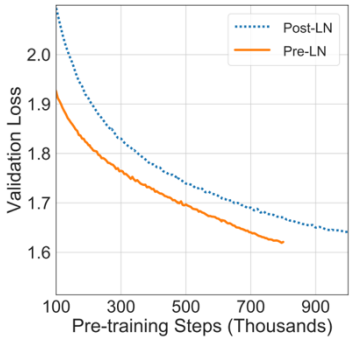
Salazar and Ngyuen 2019



(a) Validation Loss (IWSLT)



(b) BLEU (IWSLT)

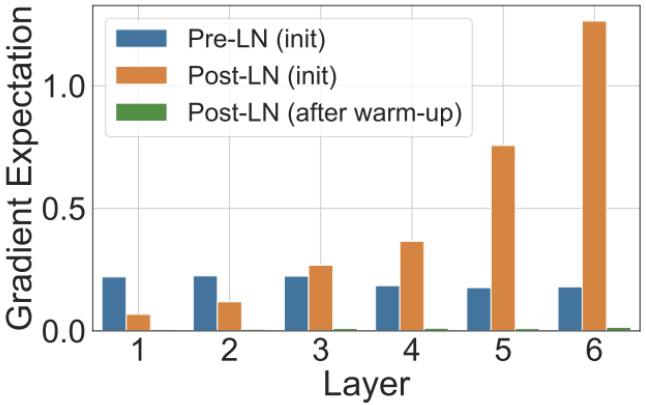


(a) Validation Loss on BERT

Figure from Xiong 2020

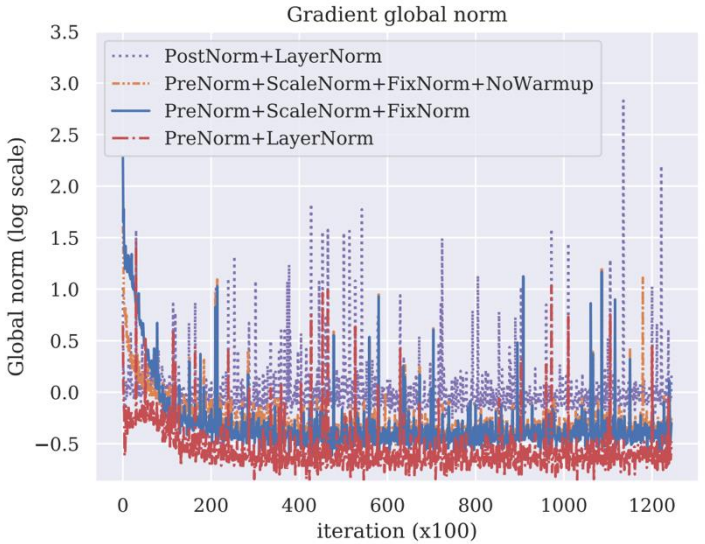
Pre-vs-post norm, explanations?

Gradient attenuation [Xiong 2020]



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

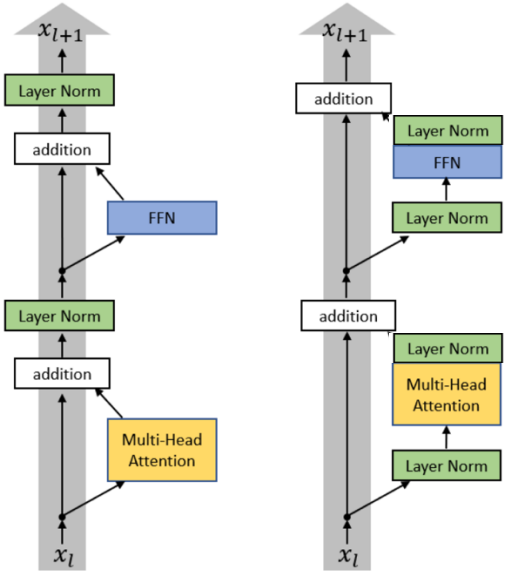
Gradient spikes [Salazar and Ngyuen]



Original stated advantage– removing warmup.
Today – stability and larger LRs for large networks

New things – ‘double’ norm.

If putting LayerNorms in residual streams is bad.. Why not post-norm outside the stream?



Recent models: Grok, Gemma 2. Olmo 2 *only* does non-residual post norm

LayerNorm vs RMSNorm

Original transformer: **LayerNorm** – normalizes the mean and variance across d_{model}

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Notable models:

GPT3/2/1, OPT, GPT-J, BLOOM

Many modern LMs: **RMSNorm** – does not subtract mean or add a bias term

$$y = \frac{x}{\sqrt{||x||_2^2 + \varepsilon}} * \gamma$$

Notable models:

LLaMA-family, PaLM, Chinchilla, T5

Why RMSNorm?

Modern explanation – it's faster (and just as good).

- **Fewer operations** (no mean calculation)
- **Fewer parameters** (no bias term to store)

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Does this explanation make sense?

Operator class	% flop
△ Tensor contraction	99.80
□ Stat. normalization	0.17
○ Element-wise	0.03

Matrix multiplies are the *vast* majority of FLOPs (and memory)

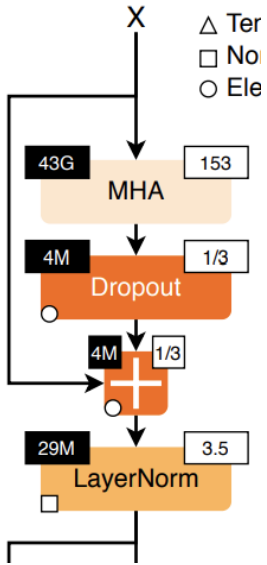
[Ivanov et al 2023]

Why RMSNorm (2)

Important lesson: FLOPS are not runtime! (we will discuss this in far more detail later)

Operator class	% flop	% Runtime
△ Tensor contraction	99.80	61.0
□ Stat. normalization	0.17	25.5
○ Element-wise	0.03	13.5

RMSNorm can still matter due to the importance of *data movement*



Left top (”43G”) is FLOPS
Right top (“153”) is the FLOP-to-memory ratio

[Ivanov et al 2023]

RMSNorm - validation

RMSNorm runtime (and surprisingly, perf) gains have been seen in papers

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
RMS Norm	$223M$	$11.1T$	3.68	2.167 ± 0.008	**1.821**	**75.45**	**17.94**	**24.07**	**27.14**
Rezero	$223M$	$11.1T$	3.51	2.262 ± 0.003	1.939	61.69	15.64	20.90	26.37
Rezero + LayerNorm	$223M$	$11.1T$	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Rezero + RMS Norm	$223M$	$11.1T$	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	$223M$	$11.1T$	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31

Narang et al 2020

More generally: dropping bias terms

Most modern transformers don’t have bias terms.

Original Transformer:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Most implementations (if they’re not gated):

$$FFN(x) = \sigma(xW_1)W_2$$

Reasons: memory (similar to RMSnorm) and optimization stability

LayerNorm: recap

- Basically everyone does pre-norm.
  - Intuition – keep the good parts of residual connections
  - Observations – nicer gradient propagation, fewer spike
  - Some people add a second norm outside the residual stream (NOT post-norm)

- Most people do RMSnorm
  - In practice, works as well as LayerNorm
  - But, has fewer parameters to move around, which saves on wallclock time
  - People more generally drop bias terms since the compute/param tradeoffs are not great.

Activations

A whole zoo of activations ..

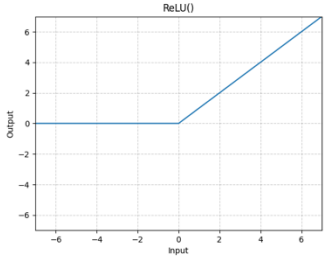
ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU

What are these things? What do people use? Does it matter?

A few of the common activations

ReLU

$$FF(x) = \max(0, xW_1)\, W_2$$

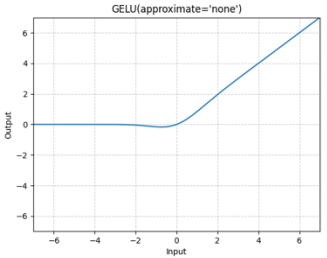


Notable models:

Original transformer, T5, Gopher, Chinchilla, OPT

GeLU

$$FF(x) = \mathrm{GELU}(xW_1)W_2$$
$$GELU(x) \coloneqq x\Phi(x)$$



Notable models:

GPT1/2/3, GPTJ, GPT-Neox, BLOOM

SwiGLU / GeGLU (next slide..)

Notable models:

Llama, PaLM,T5 v1.1, *most* models post 2023

Gated activations (*GLU)

GLUs modify the ‘first part’ of a FF layer

$$FF(x) = \max(0, xW_1)\, W_2$$

Instead of a linear + ReLU, augment the above with an (entrywise) linear term

$$\max(0, xW_1) \rightarrow \max(0, xW_1) \otimes (xV)$$

This gives the gated variant (ReGLU) – note that we have an extra parameter (V)

$$\mathrm{FF}_{\mathrm{ReGLU}}(x) = (\max(0, xW_1) \otimes xV)\, W_2$$

Gated variants of standard FF layers

GeGLU

$$\mathrm{FFN}_{\mathrm{GEGLU}}(x, W, V, W_2) = (\mathrm{GELU}(xW) \otimes xV)W_2$$

Notable models:

T5 v1.1, mT5, LaMDA, Phi3, Gemma 2, Gemma 3

SwiGLU (swish is $x * \mathrm{sigmoid}(x)$)

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

Notable models:

LLaMa 1/2/3, PaLM, Mistral, OlMo, *most models post 2023*

Note: Gated models use smaller dimensions for the d_{ff} by 2/3

Do gated linear units work?

Yes, fairly consistently so.

	Score Average	CoLA MCC	SST-2 Acc
FFN_{ReLU}	83.80	51.32	94.04
FFN_{GELU}	83.86	53.48	94.04
$\text{FFN}_{\text{Swish}}$	83.60	49.79	93.69
FFN_{GLU}	84.20	49.16	94.27
$\text{FFN}_{\text{GEGLU}}$	84.12	53.65	93.92
$\text{FFN}_{\text{Bilinear}}$	83.79	51.02	**94.38**
$\text{FFN}_{\text{SwiGLU}}$	84.36	51.59	93.92
$\text{FFN}_{\text{ReGLU}}$	**84.67**	**56.16**	**94.38**
[Raffel et al., 2019]	83.28	53.84	92.68
ibid. stddev.	0.235	1.111	0.569

Shazeer 2020

Do gated linear units work (2)?

Yes, with other works corroborating Shazeer 2020

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ
Vanilla Transformer	$223M$	$11.1T$	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02
GeLU	$223M$	$11.1T$	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**
Swish	$223M$	$11.1T$	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**
ELU	$223M$	$11.1T$	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02
GLU	$223M$	$11.1T$	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34
GeGLU	$223M$	$11.1T$	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**
ReGLU	$223M$	$11.1T$	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**
SeLU	$223M$	$11.1T$	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75
SwiGLU	$223M$	$11.1T$	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**
LiGLU	$223M$	$11.1T$	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**
Sigmoid	$223M$	$11.1T$	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02
Softplus	$223M$	$11.1T$	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**

Narang et al 2020

Gating, activations

- **Many variations (ReLU, GeLU, *GLU) across models.**

- ***GLU isn't *necessary* for a good model (see GPT3), but it's probably helpful**

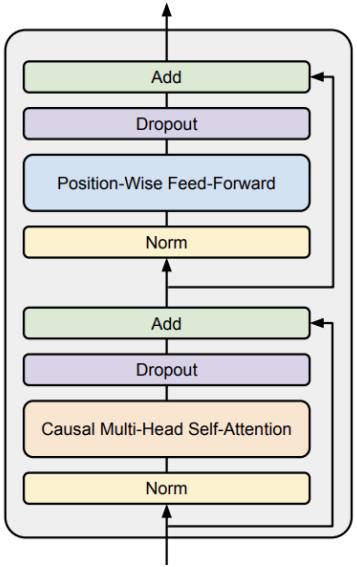
  Otuer, recent outlier models..

  Nemotron 340B (Squared ReLU), Falcon 2 11b (ReLU)

- **But evidence points towards somewhat consistent gains from Swi/GeGLU**

Serial vs Parallel layers

Normal transformer blocks are *serial* – they compute attention, then the MLP



Could we parallelize the transformer block?

Parallel layers

A few models (GPTJ, PaLM, GPT-NeoX) do parallel layers. Originally in GPT-J

> **Parallel Layers** – We use a "parallel" formulation in each Transformer block (Wang & Komatsuzaki, 2021), rather than the standard "serialized" formulation. Specifically, the standard formulation can be written as:
>
> $$y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x)))$$
>
> Whereas the parallel formulation can be written as:
>
> $$y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$$
>
> The parallel formulation results in roughly 15% faster training speed at large scales, since the MLP and Attention input matrix multiplications can be fused. Ablation experiments showed a small quality degradation at 8B scale but no quality degradation at 62B scale, so we extrapolated that the effect of parallel layers should be quality neutral at the 540B scale.

If implemented right, LayerNorm can be shared, and matrix multiplies can be fused

Recent Models: Cohere Command A, Falcon 2 11B, Command R+

Summary: architectures

Pre-vs-post norm:

- Everyone does pre-norm (except OPT350M), likely with good reason.

Layer vs RMSnorm:

- RMSnorm has clear compute wins, sometimes even performance

Gating:

- GLUs seem generally better, though differences are small

Serial vs parallel layers:

- No extremely serious ablations, but has a compute win.

Name	Year	Norm	Parallel Layer	Pre-norm	Position embedding	Activations
Original transformer	2017	LayerNorm	Serial	☐	Sine	ReLU
GPT	2018	LayerNorm	Serial	☐	Absolute	GeLU
T5 (11B)	2019	RMSNorm	Serial	☑	Relative	ReLU
GPT2	2019	LayerNorm	Serial	☑	Absolute	GeLU
T5 (XXL 11B) v1.1	2020	RMSNorm	Serial	☑	Relative	GeGLU
mT5	2020	RMSNorm	Serial	☑	Relative	GeGLU
GPT3 (175B)	2020	LayerNorm	Serial	☑	Absolute	GeLU
GPTJ	2021	LayerNorm	Parallel	☑	RoPE	GeLU
LaMDA	2021			☑	Relative	GeGLU
Anthropic LM (not claude)	2021			☑		
Gopher (280B)	2021	RMSNorm	Serial	☑	Relative	ReLU
GPT-NeoX	2022	LayerNorm	Parallel	☑	RoPE	GeLU
BLOOM (175B)	2022	LayerNorm	Serial	☑	AliBi	GeLU
OPT (175B)	2022	LayerNorm	Serial	☑	Absolute	ReLU
PaLM (540B)	2022	RMSNorm	Parallel	☑	RoPE	SwiGLU
Chinchilla	2022	RMSNorm	Serial	☑	Relative	ReLU
Mistral (7B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA2 (70B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
LLaMA (65B)	2023	RMSNorm	Serial	☑	RoPE	SwiGLU
GPT4	2023			☐		
Baichuan 2	2023	RMSNorm	Serial	☑	AliBi	SwiGLU
Olmo 2	2024	RMSNorm	Serial	☐	RoPE	SwiGLU
Gemma 2 (27B)	2024	RMSNorm	Serial	☑	RoPE	GeGLU
Nemotron-4 (340B)	2024	LayerNorm	Serial	☑	RoPE	SqRelu
Qwen 2 (72b) - same for 2.5	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Falcon 2 11B	2024	LayerNorm	Parallel	☑	RoPE	GeLU
Phi3 (small) - same for phi4	2024	RMSNorm	Serial	☑	RoPE	GeGLU
Llama 3 (70B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Reka Flash	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Command R+	2024	LayerNorm	Parallel	☑	RoPE	SwiGLU
OLMo	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Qwen (14B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
DeepSeek (67B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Yi (34B)	2024	RMSNorm	Serial	☑	RoPE	SwiGLU
Mixtral of Experts	2024			☐		
Command A	2025	LayerNorm	Parallel	☑	Hybrid (RoPE+NoPE)	SwiGLU
Gemma 3	2025	RMSNorm	Serial	☐	RoPE	GeGLU
SmolLM2 (1.7B)	2025	RMSNorm	Serial	☑	RoPE	SwiGLU

Many variations in position embeddings

Sine embeddings: add sines and cosines that enable localization

$$Embed(x, i) = v_x + PE_{pos}$$

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Notable models:

Original transformer

Absolute embeddings: add a position vector to the embedding

$$Embed(x, i) = v_x + u_i$$

Notable models:

GPT1/2/3, OPT

Relative embeddings: add a vector to the *attention computation*

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Notable models:

T5, Gopher, Chinchilla

Rope embeddings (next slides..)

Notable models:

GPTJ, PaLM, LLaMA

Most 2024+ models

RoPE: rotary position embeddings

High level thought process: a *relative* position embedding should be some $f(x, i)$ s.t.

$$\langle f(x,i), f(y,j) \rangle = g(x, y, i-j)$$

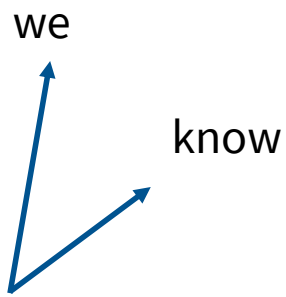
That is, the attention function *only* gets to depend on the relative position (i-j). How do existing embeddings not fulfill this goal?

- **Sine:** Has various cross-terms that are not relative
$$\langle Embed(x,i), Embed(y,i) \rangle = \langle v_x, v_y \rangle + \langle PE_i, v_y \rangle \ldots$$
- **Absolute:** obviously not relative
- **Relative embeddings:** $e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$ is not an inner product

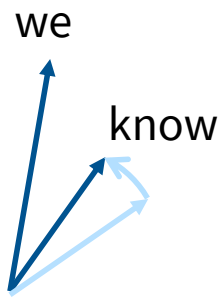
RoPE: rotary position embeddings

How can we solve this problem?

- We want our embeddings to be invariant to absolute position
- We know that inner products are invariant to arbitrary rotation.

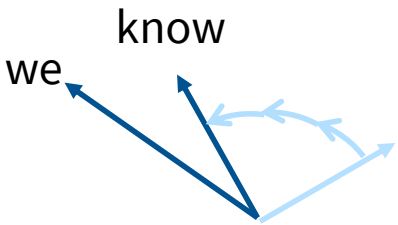


Position independent embedding



Embedding
“we know that”

Rotate we by ‘0 positions’
know by ‘1 positions’

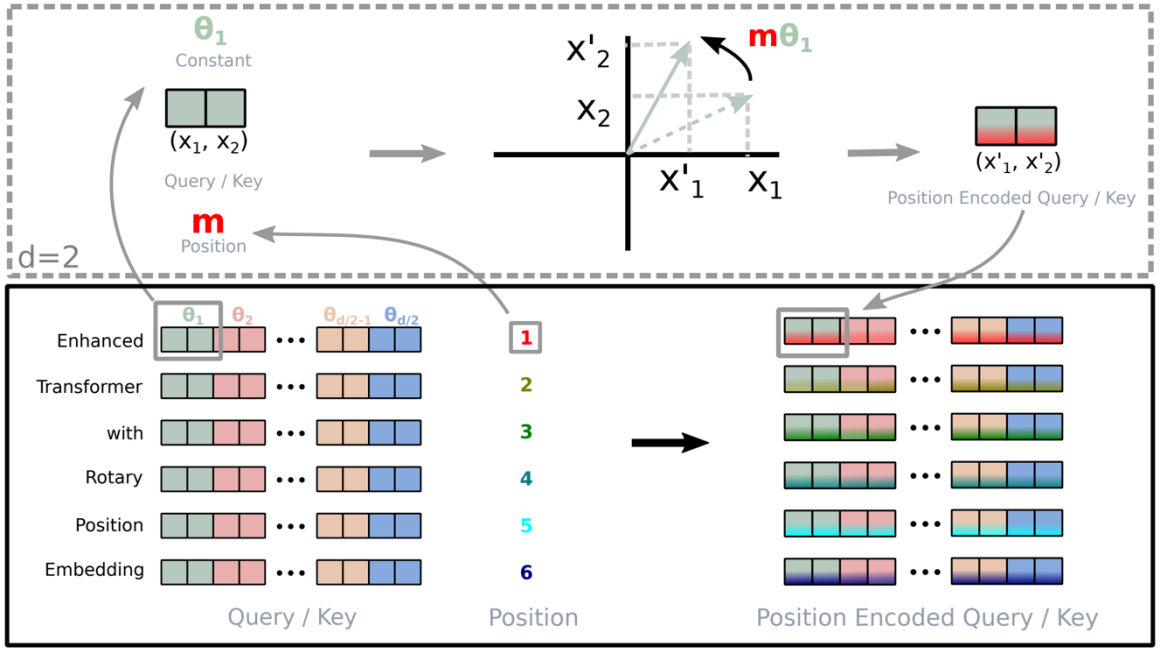


Embedding
“of course we know”

Rotate we by ‘2 positions’
Rotate know by ‘3 positions’

RoPE: rotary position embeddings

There are many rotations, which one do you pick?



[Su et al 2021]

Just pair up the coordinates and rotate them in 2d (motivation: complex numbers)

The actual RoPE math

Multiply with sines and cosines

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

Difference with sine embeddings – not additive, no cross terms

Implementation and code for RoPE

Usual attention stuff

```
query_states = self.q_proj(hidden_states)
key_states = self.k_proj(hidden_states)
value_states = self.v_proj(hidden_states)

# Flash attention requires the input to have the shape
# batch_size x seq_length x head_dim x hidden_dim
# therefore we just need to keep the original shape
query_states = query_states.view(bsz, q_len, self.num_heads, self.head_dim).transpose(1, 2)
key_states = key_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)
value_states = value_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)
```

Get the RoPE matrix cos/sin

Multiply query/key inputs

```
cos, sin = self.rotary_emb(value_states, position_ids)
query_states, key_states = apply_rotary_pos_emb(query_states, key_states, cos, sin)
```

…

Same stuff as the usual multi-head self attention below

Note: embedding at *each attention operation* to enforce position invariance

Hyperparameters

Transformer hyperparameter questions you might have had in 224n..

- How much bigger should the feedforward size be compared to hidden size?
- How many heads, and should num_heads always divide hidden size?
- What should my vocab size be?

And other model setting questions

- Do people even regularize these huge LMs?
- How do people scale these models - very deep or very wide?

Surprising (?) consensus hyperparameter 1

Feedforward – model dimension ratio.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

There are two dimensions that are relevant – the feedforward dim (d_{ff}) and model dim (d_{model}). What should their relationship be?

$$\boldsymbol{d_{ff} = 4\, d_{model}}$$

This is *almost always* true. There's just a few exceptions.

Exception #1 – GLU variants

Remember that GLU variants scale down by 2/3rd. This means most GLU variants have $d_{ff} = \frac{8}{3} d_{model}$. This is mostly what happens. Some notable such examples.

Model	d_{ff}/d_{model}
PaLM	4
Mistral 7B	3.5
LLaMA-2 70B	3.5
LLaMA 70B	2.68
Qwen 14B	2.67
DeepSeek 67B	2.68
Yi 34B	2.85
T5 v1.1	2.5

Models are roughly in this range, though PaLM, LLaMA2 and Mistral are slightly larger

Exception #2 – T5

As we have (and will) see, most LMs are have boring, conservative hyperparameters. One exception is T5 [Raffel et al 2020] which has some *very bold* settings.

In particular, for the 11B model, they set

$$d_{ff} = 65{,}536$$
$$d_{model} = 1024$$

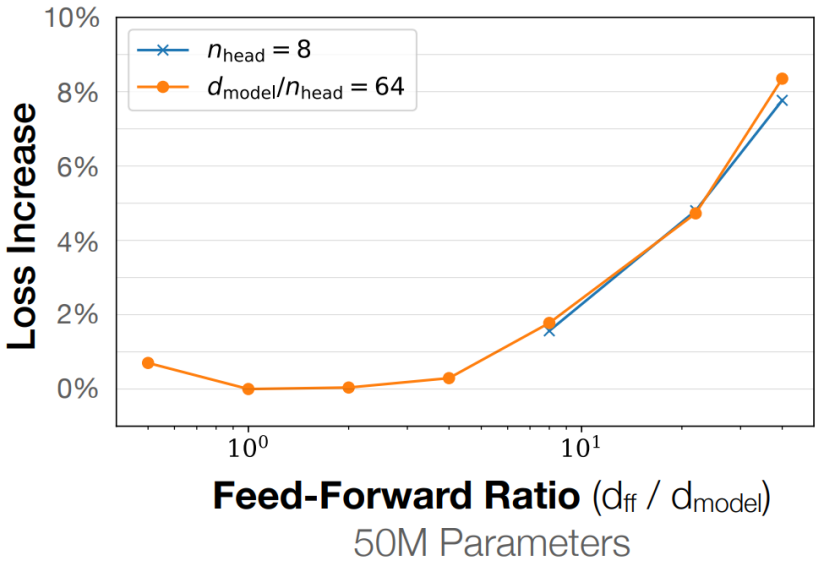
For an astounding 64-times multiplier.

> for "11B" we use $d_{\text{ff}} = 65{,}536$ with 128-headed attention producing a model with about 11 billion parameters. We chose to scale up d_{ff} specifically because modern accelerators (such as the TPUs we train our models on) are most efficient for large dense matrix multiplications like those in the Transformer's feed-forward networks.

Other, recent exceptions – Gemma 2 (8x), SmolLM/Gemma 3 (4x, GLU)

Why this range of multipliers?

Empirically, there's a basin between 1-10 where this hyperparameter is near-optimal



[Kaplan+ 2020]

What can we learn from the model-dim hyperparam?

- The ‘default’ choices of $d_{ff} = 4d_{model}$ and $d_{ff} = 2.66d_{model}$ have worked well for nearly all modern LLMs.

- But T5 does show that even radical choices of $d_{ff} = 64d_{model}$ can work. This hyperparameter choice isn’t written in stone.

- That said, T5 has a follow-up model (T5 v1.1) that is ‘improved’ and uses a much more standard 2.5 multiplier on GeGLU, so the 64-times multiplier is likely suboptimal.

Surprising (?) consensus hyperparameter 2

Head-dim*num-heads to model-dim ratio. As a reminder, slide from 224n.

Multi-head self-attention is computationally efficient

- Even though we compute h many attention heads, it's not really more costly.
  - We compute $XQ \in \mathbb{R}^{n \times d}$, and then reshape to $\mathbb{R}^{n \times h \times d/h}$. (Likewise for XK, XV.)
  - Then we transpose to $\mathbb{R}^{h \times n \times d/h}$; now the head axis is like a batch axis.
  - Almost everything else is identical, and the **matrices are the same sizes.**

This doesn't *have to* be true: we can have head-dimensions > model-dim / num-heads.

But most models do follow this guideline

How many heads, whats the model dim?

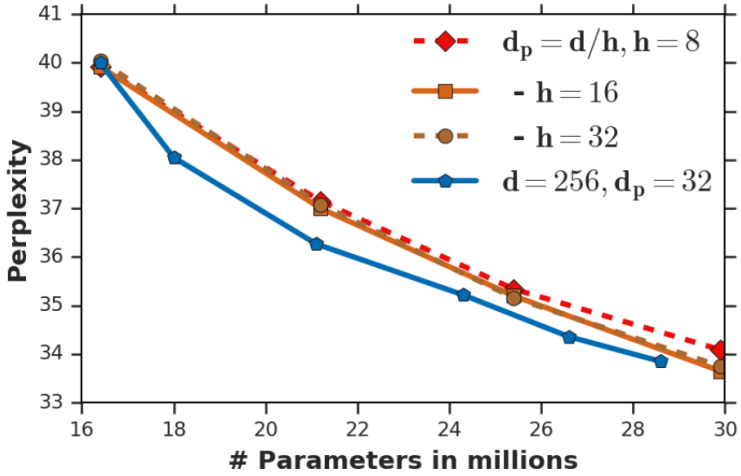
Some examples of this hyperparameter

	Num heads	Head dim	Model dim	Ratio
GPT3	96	128	12288	1
T5	128	128	1024	16
T5 v1.1	64	64	4096	1
LaMDA	128	128	8192	2
PaLM	48	258	18432	1.48
LLaMA2	64	128	8192	1

Most models have ratios around 1 – notable exceptions by some google models.

Evidence for 1-1 ratio?

There have been papers written against the 1-1 ratio [Bhojanapalli et al 2020]



But we don’t seem to be seeing significant ‘low rank bottlenecks’ in practice..

Aspect ratios

Should my model be deep or wide? *How* deep and how wide?

Most models are surprisingly consistent on this one too!

Sweet spot?

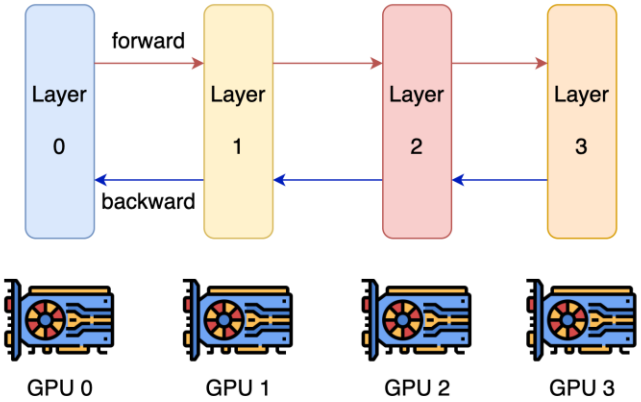
Model	d_{model}/n_{layer}
BLOOM	205
T5 v1.1	171
PaLM (540B)	156
GPT3/OPT/Mistral/Qwen	128
LLaMA / LLaMA2 / Chinchila	102
T5 (11B)	43
GPT2	33

Considerations about aspect ratio

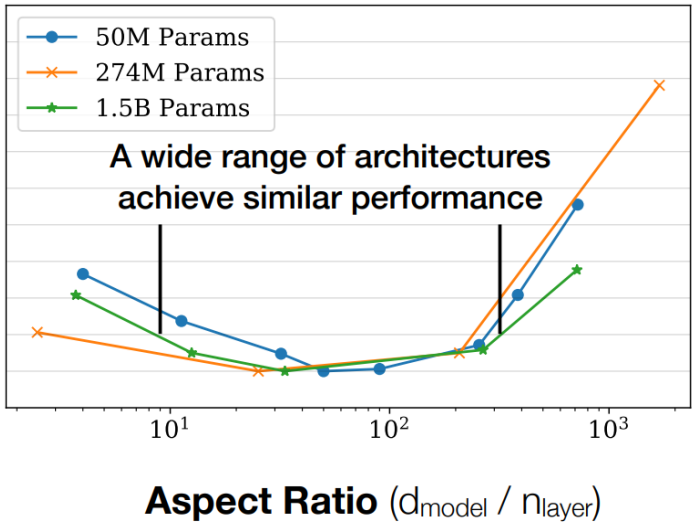
Extremely deep models are harder to parallelize and have higher latency

> **The Limits of Depth vs Width** We note an obvious limitation with our advice. Scaling depth has an obvious limiter, i.e., they are non-parallelizable across different machines or devices and every computation has to always wait for the previous layer. This is unlike width, which can be easily parallelizable over thousands or hundreds of thousands of devices. Within the limitation of scaling

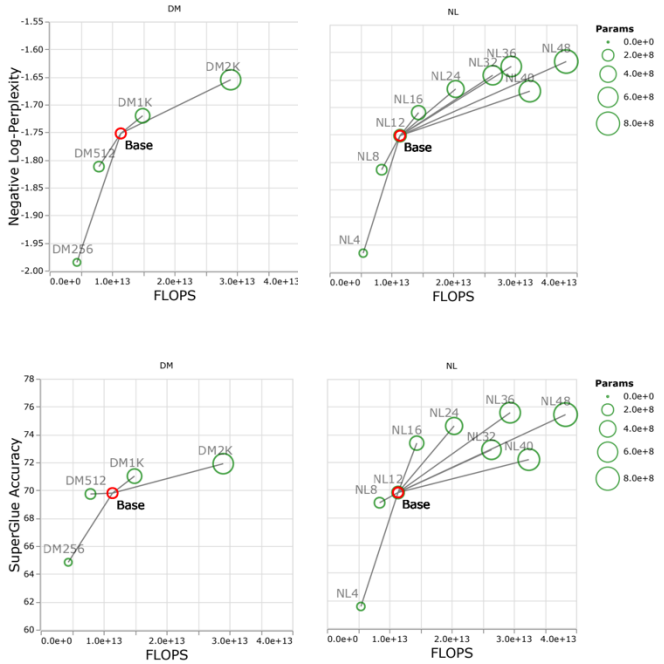
[Tay et al 2021]



Evidence on aspect ratio scaling



[Kaplan et al 2020]



[Tay et al 2021]

What are typical vocabulary sizes?

Monolingual models – 30-50k vocab

Model	Token count
Original transformer	37000
GPT	40257
GPT2/3	50257
T5/T5v1.1	32128
LLaMA	32000

Multilingual / production systems 100-250k

Model	Token count
mT5	250000
PaLM	256000
GPT4	100276
Command A	255000
DeepSeek	100000
Qwen 15B	152064
Yi	64000

Monolingual vocabs don’t need to be huge, but multilingual ones do

Dropout and other regularization

Do we need regularization during pretraining?

Arguments against:

- There is *a lot* of data (trillions of tokens), more than parameters.
- SGD only does a single pass on a corpus (hard to memorize)

This is all quite reasonable.. but what do people do in practice?

Dropout and weight decay in practice

Model	Dropout*	Weight decay
Original transformer	0.1	0
GPT2	0.1	0.1
T5	0.1	0
GPT3	0.1	0.1
T5 v1.1	0	0
PaLM	0	(variable)
OPT	0.1	0.1
LLaMA	0	0.1
Qwen 14B	0.1	0.1

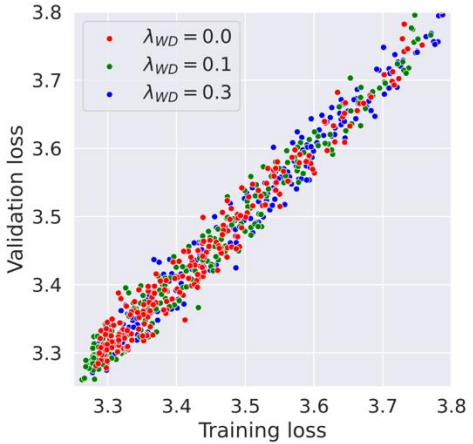
Many older models used dropout during pretraining

Newer models (except Qwen) rely only on weight decay

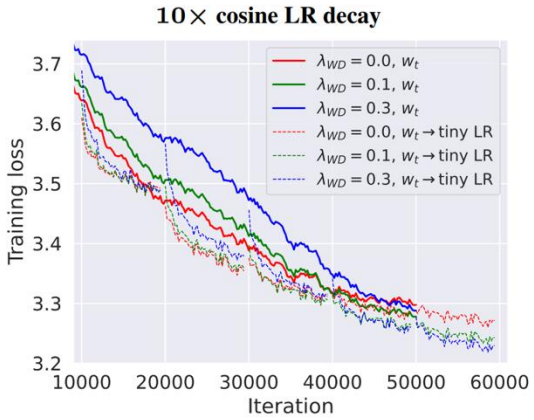
* Most of the times papers just don't discuss dropout. On open models, this closely matches not doing dropout. This may not be true of closed models.

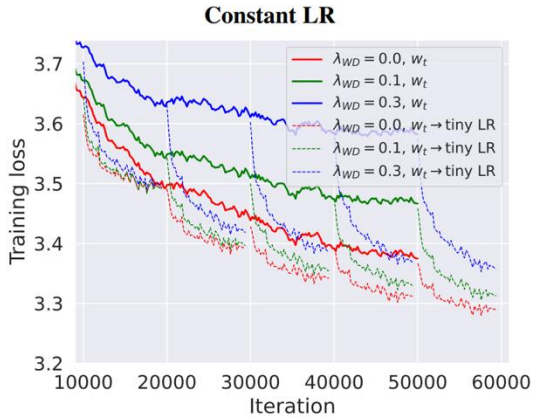
Why weight decay LLMs?

[Andriushchenko et al 2023] has interesting observations about LLM weight decay



It's not to control overfitting





Weight decay interacts with learning rates (cosine schedule)

Summary: hyperparameters

Feedforward

- Factor-of-4 rule of thumb (8/3 for GLUs) is standard (with some evidence)

Head dim

- Head dim*Num head = D model is standard – but low to no validation

Aspect ratio

- Wide range of ‘good’ values (100-200). Systems concerns dictate the value

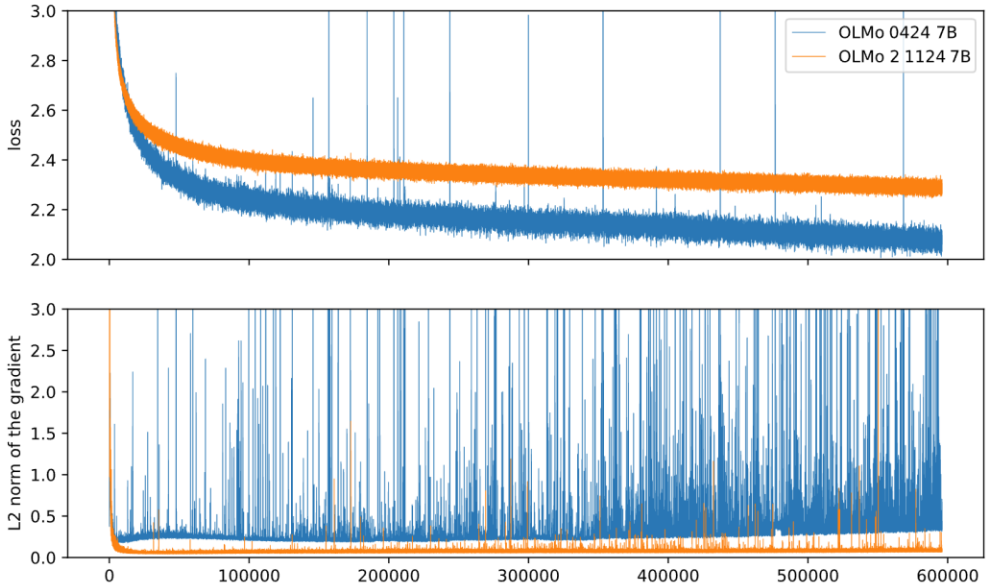
Regularization

- You still ‘regularize’ LMs but its effects are primarily on optimization dynamics

Name	Year	MLP factor	Aspect ratio (d/layer)	weight decay	drop_rate
Original transformer	2017	4	85	0	0.1
GPT	2018	4	64	0.1	0.1
T5 (11B)	2019	64	43	0	0.1
GPT2	2019	4	33	0.1	0.1
T5 (XXL 11B) v1.1	2020	2.5	171	0	0
mT5	2020	2.5	171	0	0
GPT3 (175B)	2020	4	128	0.1	0.1
GPTJ	2021		146	0.1	0
LaMDA	2021	8	128		
Anthropic LM (not claude)	2021	4	128		
Gopher (280B)	2021	4	205		
GPT-NeoX	2022	4	140	0.01	0
BLOOM (175B)	2022	4	205	0.1	0
OPT (175B)	2022	4	128	0.1	0.1
PaLM (540B)	2022	4	156		0
Chinchilla	2022	4	102		
Baichuan 2	2023	2.68	128	0.1	0
Mistral (7B)	2023	3.5	128	0.1	0
LLaMA2 (70B)	2023	3.5	102	0.1	0
LLaMA (65B)	2023	2.6875	102	0.1	0
GPT4	2023		0		
Olmo 2	2024	2.6875	128		
Gemma 2 (27B)	2024	8	100		
Nemotron-4 (340B)	2024	4	192		0
Qwen 2 (72b) - same for 2.5	2024	3.609	102		
Falcon 2 11B	2024	4	68	0.1	
Phi3 (small) - same for phi4	2024	3.5	128		
Llama 3 (70B)	2024	3.5	102		0
Reka Flash	2024		0		
Command R+	2024	2.75	192		
OLMo	2024	2.6875	128	0.1	0
Qwen (14B)	2024	2.675	128	0.1	0.1
DeepSeek (67B)	2024	2.6875	86	0.1	0
Yi (34B)	2024	2.857142	119	0.1	0
Mixtral of Experts	2024		0		
Command A	2025		0		
Gemma 3	2025	4	87		
SmolLM2 (1.7B)	2025	4	85		

Stability tricks

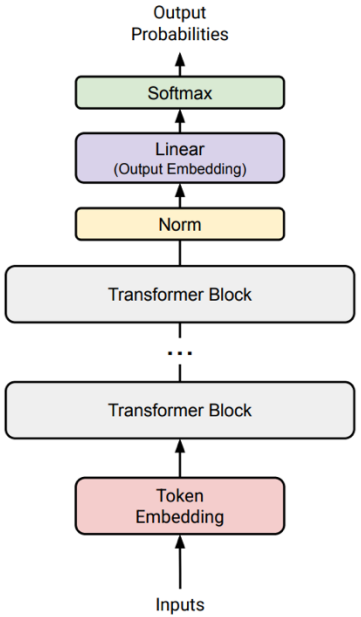
Recently, lots of attention on *stable training*

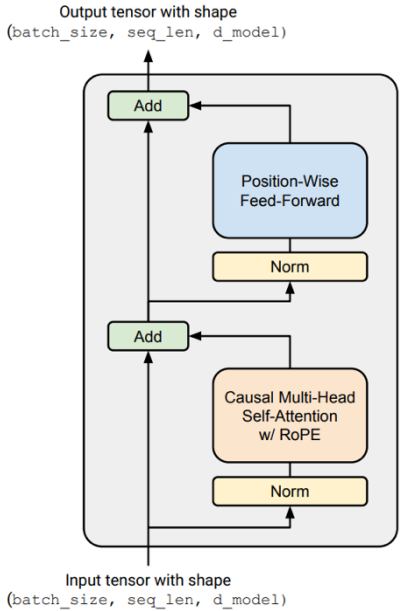


Don’t train models that look like the blue curve!

Where do the issues arise? Beware of softmaxes!

Softmaxes – can be ill-behaved due to exponentials / divison by zero





Output softmax stability – the ‘z-loss’

Recall the softmax calculation

$$\begin{aligned} \log(P(x)) &= \log\left(\frac{e^{U_r(x)}}{Z(x)}\right) \\ &= U_r(x) - \log(Z(x)) \\ Z(x) &= \Sigma_{r'=1}^{|V|} e^{U_{r'}(x)} \end{aligned}$$

$$\begin{aligned} L &= \sum_i \left[\log(P(x_i)) - \alpha(\log(Z(x_i)) - 0)^2\right] \\ &= \sum_i \left[\log(P(x_i)) - \alpha \log^2(Z(x_i))\right] \end{aligned}$$

[From Devlin 2014]

This is useful for stability! PaLM pioneered this ‘z loss’ trick.

> We additionally use an auxiliary loss of $z\_loss = 10^{-4} \cdot \log^2 Z$ to encourage the softmax normalizer $\log(Z)$ to be close to 0, which we found increases the stability of training.

Other examples: Baichuan 2 (2023), DCLM (2024), OLMo 2 (2025)

Attention softmax stability – the ‘QK norm’

The query and keys ar softmax operation.

Other examples: DCLM, OLMo2, Gemma 2

Originally from vision and multimodal models [Dehgani 2023, Idefcs, Chameleon]

Logit soft-capping.

Soft-capping the logits to some maximum value via Tanh

> **Logit soft-capping**. We cap logits (Bello et al., 2016) in each attention layer and the final layer such that the value of the logits stays between −soft_cap and +soft_cap. More specifically, we cap the logits with the following function:
>
> $$\text{logits} \leftarrow \text{soft\_cap} * \tanh(\text{logits}/\text{soft\_cap}).$$
>
> We set the soft_cap parameter to 50.0 for the self-attention layers and to 30.0 for the final layer.

Prevents logits from blowing up, but also might have perf issues?

Table 4: Models perplexity with confidence interval ±0.1 at 95% level.

bf16 baseline	*soft_cap*	*QKV_norm*	*QK_norm_cap*	*QK_norm*	*QK_FC_norm*
11.19	11.24	10.85	11.00	10.84	10.87

Attention heads

Most models don’t touch the attention heads much at all with a few minor exceptions..

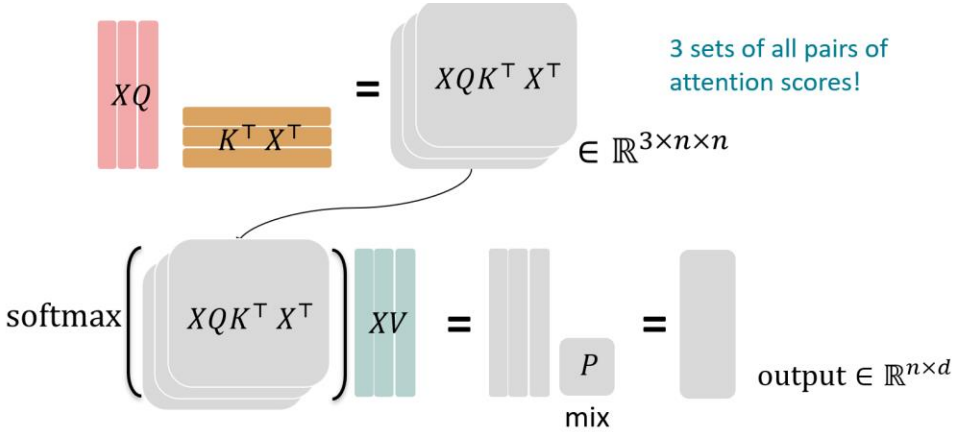
GQA / MQA : Saving inference costs by reducing the number of heads

Sparse or sliding window attention (GPT4/Mistral): restricting the attention pattern to reduce compute cost

Exotic SSM stuff (Jamba, Falcon 3, etc): not covered (sorry!)

GQA/MQA – Reducing attention head cost

Let's think about the compute involved for attention



Total arithmetic operations (bnd^2), **total memory accesses** $(bnd + bhn^2 + d^2)$ (X, softmax, projection)

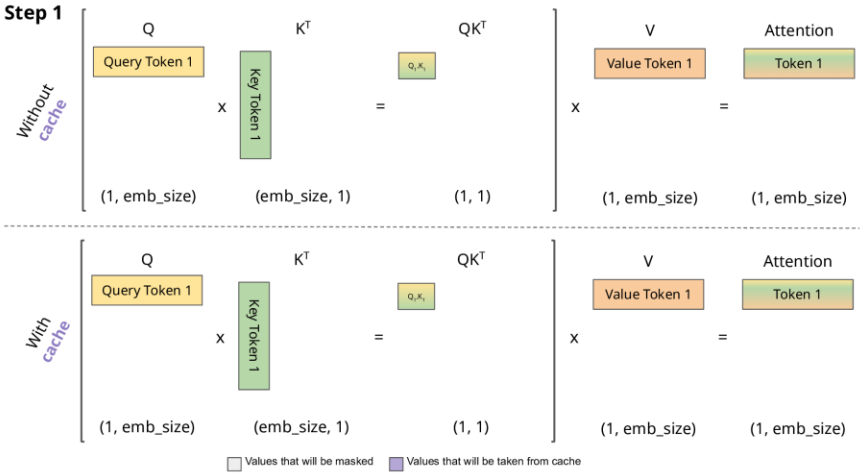
Arithmetic intensity is high $O\left(\left(\frac{1}{k} + \frac{1}{bn}\right)^{-1}\right)$ - we can keep our GPUs running

GQA/MQA – Reducing attention head cost

What about the *incremental* case when we generate text?

Key difference: can't parallelize the generation process – needs to be step by step

In this case – we need to incrementaly re-compute/update attention via the 'KV cache'



[Animation from https://medium.com/@joaolages/kv-caching-explained-276520203249]

GQA/MQA – Reducing attention head cost

What's the incremental arithmetic intensity?

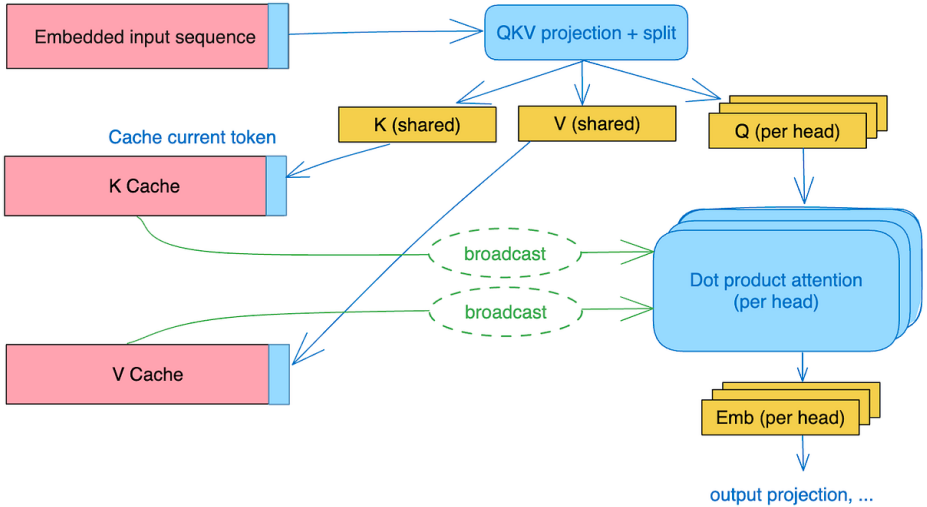
Total arithmetic operations (bnd^2), **total memory accesses** $(\overset{\text{X}}{bn^2d} + \overset{\text{projection}}{nd^2})$

Arithmetic intensity is not good $O\left(\left(\frac{n}{d} + \frac{1}{b}\right)^{-1}\right)$ - need large batches + short seq length (n) or big model dimensions (d)

Is there some way around this? The n/d term is difficult to reduce.

MQA – just have fewer key dimensions.

Key idea – have multiple queries, but just one dimension for keys and values



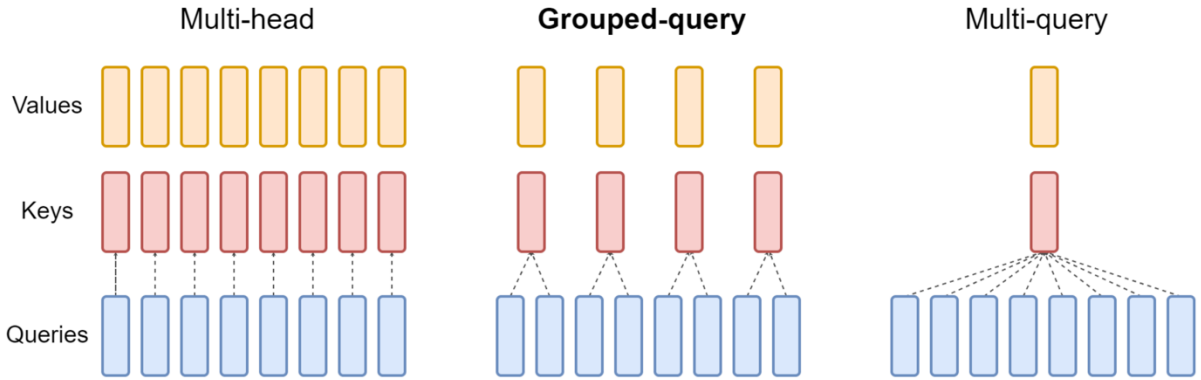
We have much fewer items to move in and out of memory (KV Cache)

Total memory access $(bnd + bn^2k + nd^2)$, **Arithmetic intensity** $O\left(\left(\frac{1}{d} + \frac{n}{dh} + \frac{1}{b}\right)^{-1}\right)$

[figure from https://blog.fireworks.ai/multi-query-attention-is-all-you-need-db072e758055]

Recent extension – GQA

Don’t go all the way to one dimension of KV – have fewer dims



Simple knob to control expressiveness (key-query ratio) and inference efficiency

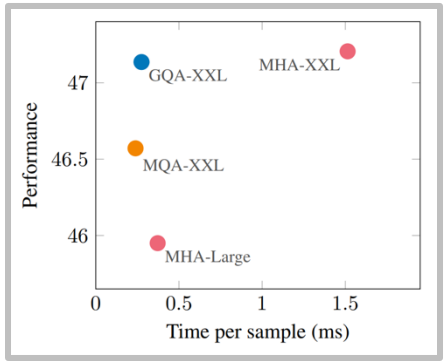
Does MQA hurt? Sometimes..

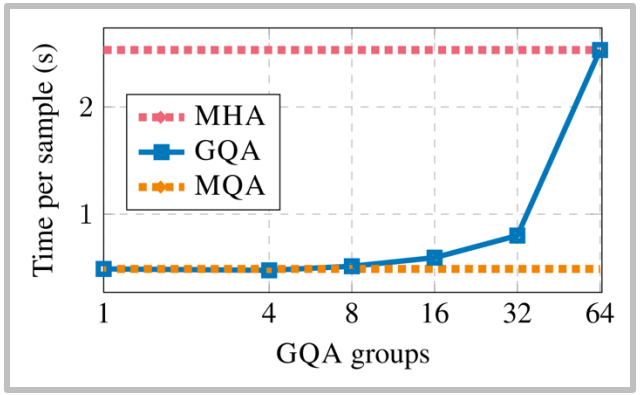
Small PPL hit w/ MQA [Shazeer 2019]

Table 3: Billion-Word LM Benchmark Results.

Attention	h	d_k, d_v	d_{ff}	dev-PPL
multi-head	8	128	8192	**29.9**
multi-query	8	128	9088	30.2
multi-head	1	128	9984	31.2
multi-head	2	64	9984	31.1
multi-head	4	32	9984	31.0
multi-head	8	16	9984	30.9

Low/no hit w/ GQA [Ainslie 2023]

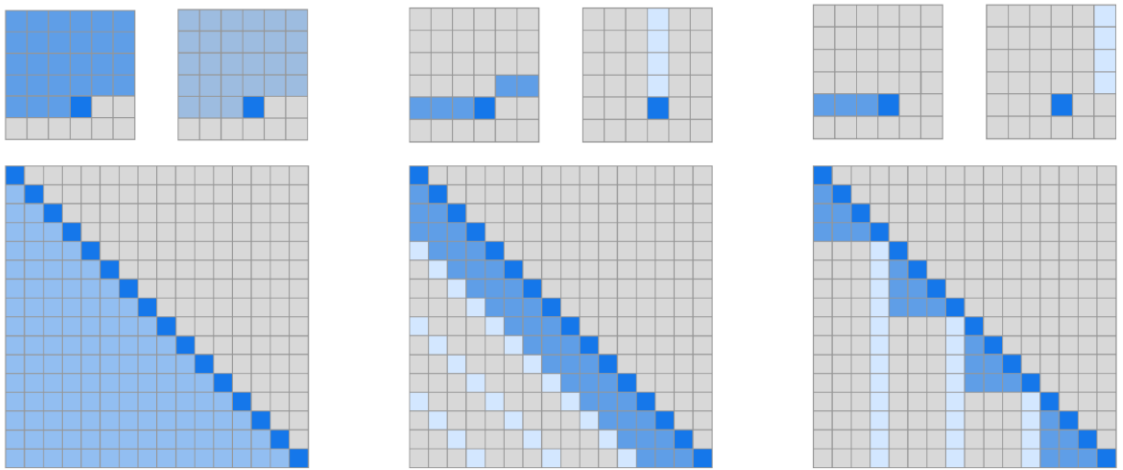




Sparse / sliding window attention

Attending to the entire context can be expensive (quadratic).

Build sparse / structured attention that trades off expressiveness vs runtime (GPT3)

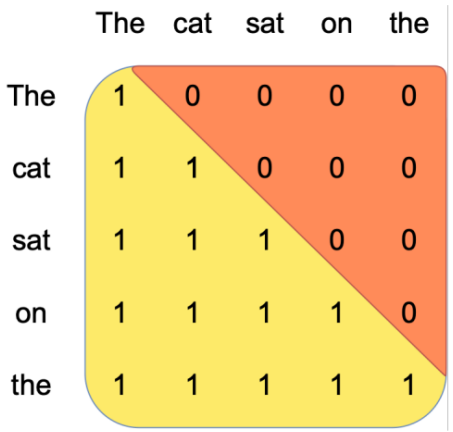


(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

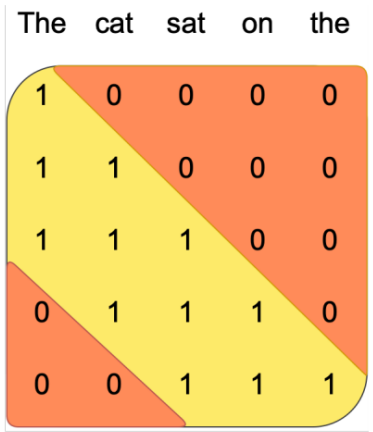
[Child et al 2019]

Sliding window attention

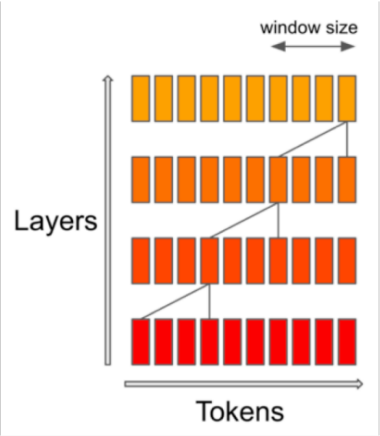
Another variation on this idea – sliding window attention



Vanilla Attention



Sliding Window Attention

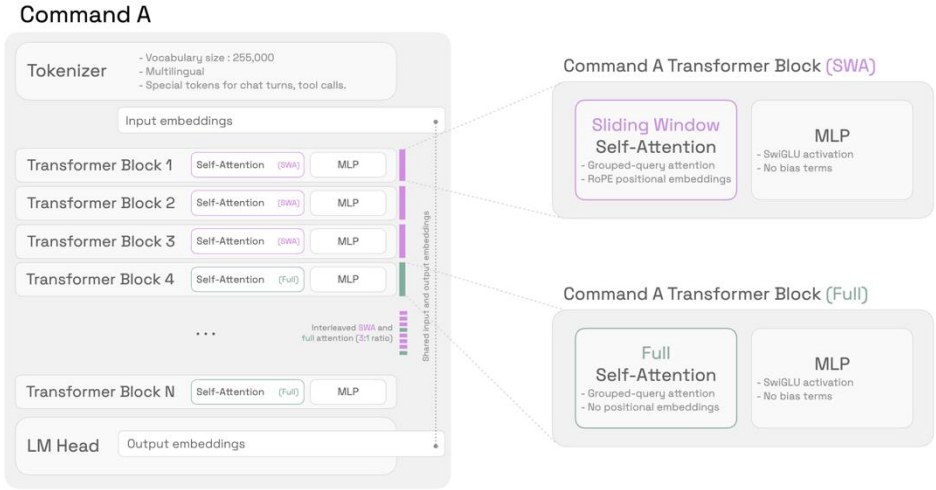


Effective Context Length

Just use the main part of the strided pattern – let depth extend effective context (Mistral)

Current standard trick – interleave ‘full’ and ‘LR’ attention

From Cohere Command A – Every 4th layer is a full attention



Long-range info via NoPE, short-range info via RoPE + SWA.

Other models – LLaMA 4, Gemma does SWA+Full RoPE.

Recap, conclusion, etc.

Many aspects (arch, hparams) of transformers are in common across the big LMs

Name	Has pa...	Link	Year	Tokenizer type	Vocab count	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	MoE	MLP factor	num_layers	model_dim
Original transformer	Yes	arxiv.org/abs...03762	2017	BPE	37000	LayerNorm	Serial	☐	Sine	ReLU	☐	4	6	
GPT	Yes	cdn.openai.com/res...er.pdf	2018	BPE	40257	LayerNorm	Serial	☐	Absolute	GeLU	☐	4	12	
GPT2	Yes	cdn.openai.com/bet...rs.pdf	2019	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	48	
T5 (11B)	Yes	arxiv.org/abs...10683	2019	SentencePiece	32128	RMSNorm	Serial	☑	Relative	ReLU	☐	64	24	
GPT3 (175B)	Yes	arxiv.org/abs...14165	2020	BPE	50257	LayerNorm	Serial	☑	Sine	GeLU	☐	4	96	
mT5	Yes	arxiv.org/abs...11934	2020	SentencePiece	250000	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
T5 (XXL 11B) v1.1	Kind of	github.com/goo...d#t511	2020	SentencePiece	32128	RMSNorm	Serial	☑	Relative	GeGLU	☐	2.5	24	
Gopher (280B)	Yes	arxiv.org/abs...11446	2021	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
Anthropic LM (not claude)	Yes	arxiv.org/abs...00861	2021	BPE	65536			☑			☐	4	64	
LaMDA	Yes	arxiv.org/abs...08239	2021	BPE	32000			☑	Relative	GeGLU	☐	8	64	
GPTJ	Kind of	huggingface.co/Ele...t-j-6b	2021	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐		28	
Chinchilla	Yes	arxiv.org/abs...15556	2022	SentencePiece	32000	RMSNorm	Serial	☑	Relative	ReLU	☐	4	80	
PaLM (540B)	Yes	arxiv.org/abs...02311	2022	SentencePiece	256000	RMSNorm	Parallel	☑	RoPE	SwiGLU	☐	4	118	
OPT (175B)	Yes	arxiv.org/abs...01068	2022	BPE	50272	LayerNorm	Serial	☑	Absolute	ReLU	☐	4	96	
BLOOM (175B)	Yes	arxiv.org/abs...05100	2022	BPE	250680	LayerNorm	Serial	☑	AliBi	GeLU	☐	4	70	
GPT-NeoX	Yes	arxiv.org/pdf...45.pdf	2022	BPE	50257	LayerNorm	Parallel	☑	RoPE	GeLU	☐	4	44	
GPT4 OPEN	Ad	arxiv.org/abs...08774	2023	BPE	100000			☐			☐			
LLaMA (65B)	Yes	arxiv.org/abs...13971	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	2.6875	80	
LLaMA2 (70B)	Yes	arxiv.org/abs...09288	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	80	
Mistral (7B)	Yes	arxiv.org/abs...06825	2023	BPE	32000	RMSNorm	Serial	☑	RoPE	SwiGLU	☐	3.5	32	

Major differences? Position embeddings, activations, tokenization