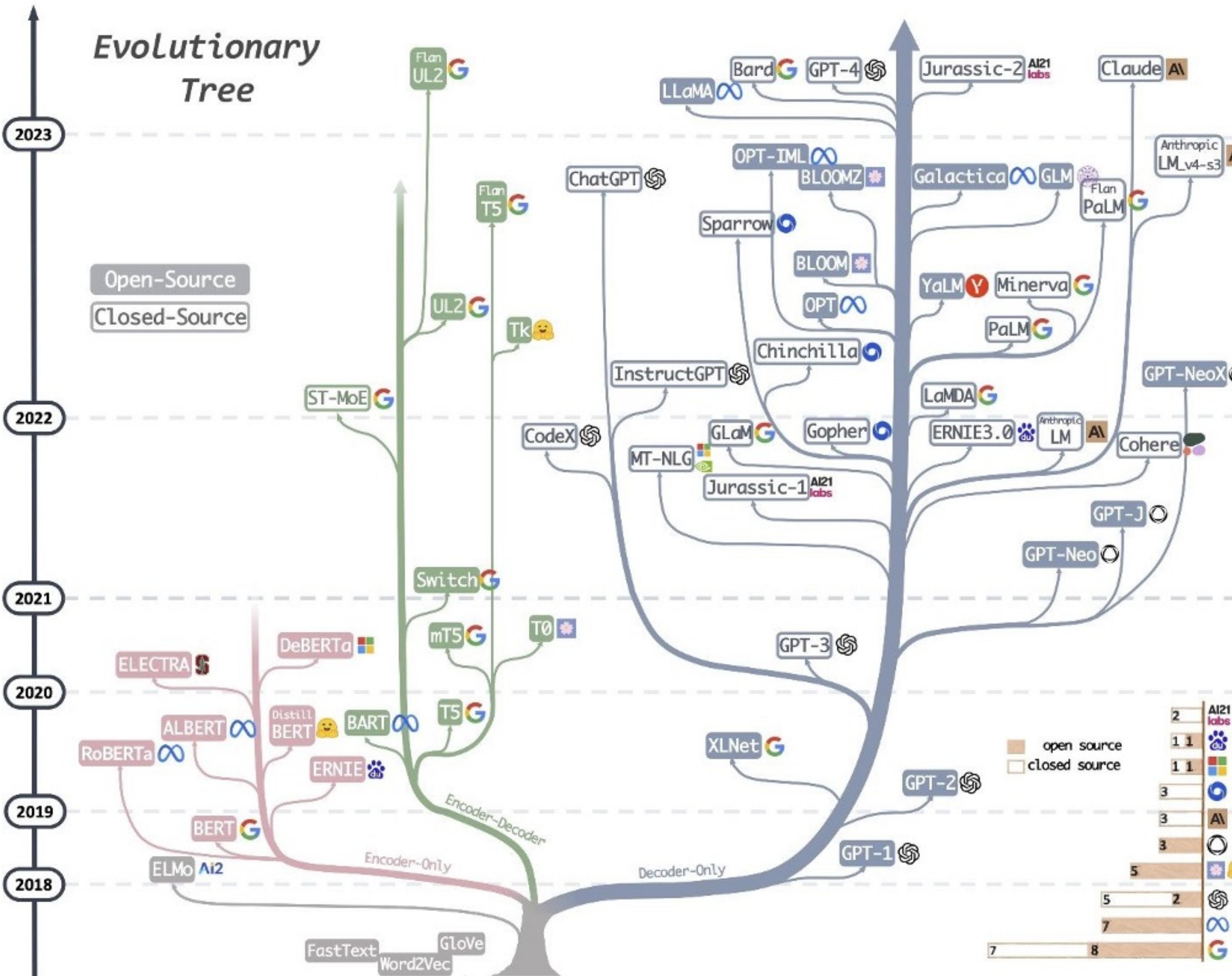


# Large Language Models (part 4)

Wei Xu

(Many slides from Greg Durrett, Daniel Khashabi, Tatsunori Hashimoto)

# Evolutionary Tree



Yang et al. (Apr. 2023)

# Open-source Efforts

# OPT: Open Pre-trained Transformer LMs

- ▶ GPT-3 models only released via API access.
- ▶ PALM not generally available outside Google
  - ▶ Doesn't support reproducible experiments.
- ▶ OPT (125M-66B-175B), to roughly matches GPT-3, with parameters are shared with the research community

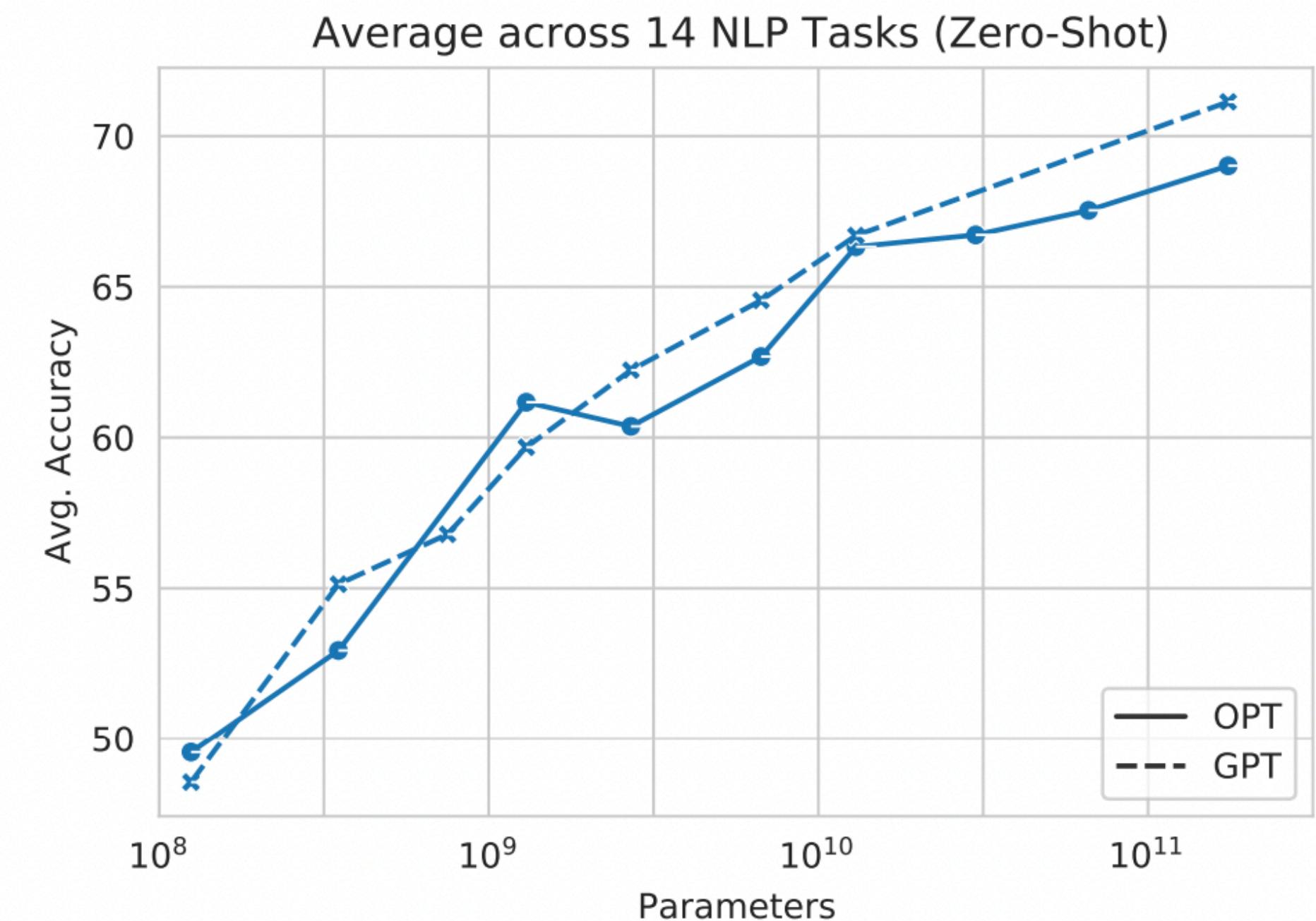


Figure 3: **Zero-shot NLP Evaluation Averages.** Across a variety of tasks and model sizes, OPT largely matches the reported averages of GPT-3. However, performance varies greatly per task: see Appendix A.

# OPT: Open Pre-trained Transformer LMs

- ▶ Includes 114 page logbook for training 175B model, interesting read

2021-11-16 11pm [Myle]: Run 12.08

- Previous run failed with mysterious error: “p2p\_plugin.c:141 NCCL WARN NET/IB : Got async event : port error”
- New log file:  
`/shared/home/namangoyal/checkpoints/175B/175B_run12.08.me_fp16.fsdp.gpf32.0.relu.transformer_lm_megatron.nlay96.emb12288.lrnpos.0emb_scale.bm_none.tps2048.gpt2.adam.b2_0.95.eps1e-08.cl1_0.lr0.00012.endlr6e-06.wu2000.dr0.1.atdr0.1.0emb_dr.wd0.1.ms8.uf1.mu143052.s1.ngpu992/train.log`

```
CKPT_DIR=/data/users/myleott/175B_run12.07.me_fp16.fsdp.gpf32.0.relu.transformer_lm_megatron.nlay96.emb12288.lrnpos.0emb_scale.bm_none.tps2048.gpt2.adam.b2_0.95.eps1e-08.cl1_0.lr0.00012.endlr6e-06.wu2000.dr0.1.atdr0.1.0emb_dr.wd0.1.ms8.uf1.mu143052.s1.ngpu992

BLOB_URL=""><<<SCRUBBED FOR RELEASE>>>

cd $CKPT_DIR
cp --recursive --include-pattern "checkpoint_5_13250*.pt" "$BLOB_URL" checkpoint_5_13250

export RESTORE_FILE=$CKPT_DIR/checkpoint_5_13250/checkpoint_5_13250.pt

export RUN_ID=175B_run12.08

INCLUDED_HOSTS=node-[1-38,40-89,91-94,96-119,121-128] \
python -m fb_sweep.opt.sweep_opt_en_lm_175b \
-n 124 -g 8 -t 1 \
-p $RUN_ID \
--checkpoints-dir /shared/home/namangoyal/checkpoints/175B/ \
--restore-file $RESTORE_FILE

After Launch:
sudo scontrol update job=1394 TimeLimit=UNLIMITED
sudo scontrol update job=1394 MailUser=<scrubbed> MailType=ALL
```

## 6 Considerations for Release

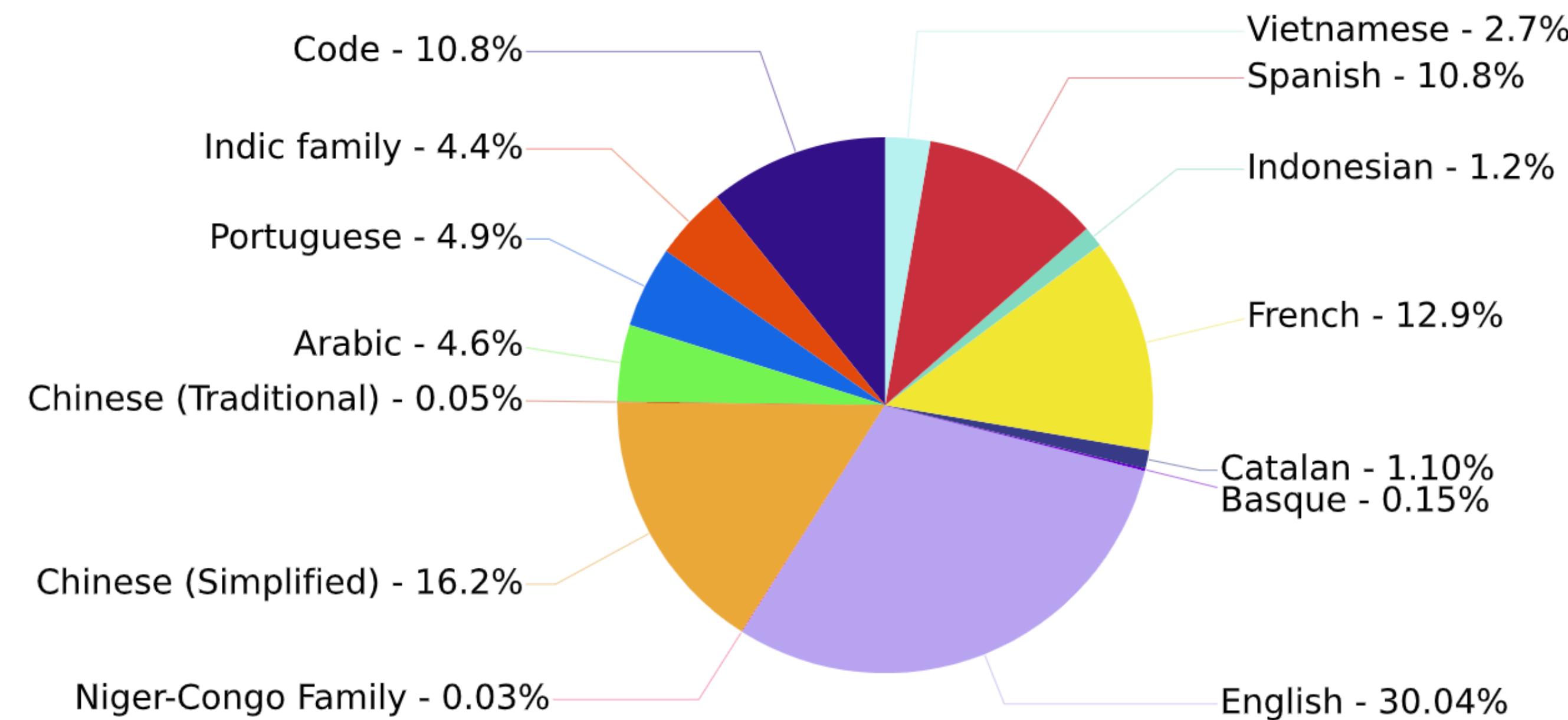
Following the recommendations for individual researchers generated by the Partnership for AI,<sup>7</sup> along with the governance guidance outlined by NIST,<sup>8</sup> we are disclosing all of the details involved in training OPT-175B through our logbook,<sup>9</sup> our code, and providing researchers access to model weights for OPT-175B, along with a suite of smaller baselines mirroring the setup for OPT-175B. We aim to be fully accountable for the development lifecycle of OPT-175B, and only through increasing transparency around LLM development can we start understanding the limitations and risks of LLMs before broader deployment occurs.

By sharing a detailed account of our day-to-day training process, we disclose not only how much compute was used to train the current version of OPT-175B, but also the human overhead required when underlying infrastructure or the training process itself becomes unstable at scale. These details

# Bloom

---

- ▶ A BigScience initiative, open-access, 176B parameter (GPT-2 architecture)
- ▶ 59 languages (46 natural language + 13 programming language)
- ▶ 1.6TB of pre-processed text



# LLaMA

- ▶ Released by Meta AI on **Feb 27, 2023**
- ▶ Weights of all models are publicly available (non-commercial license)

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: Model sizes, architectures, and optimization hyper-parameters.

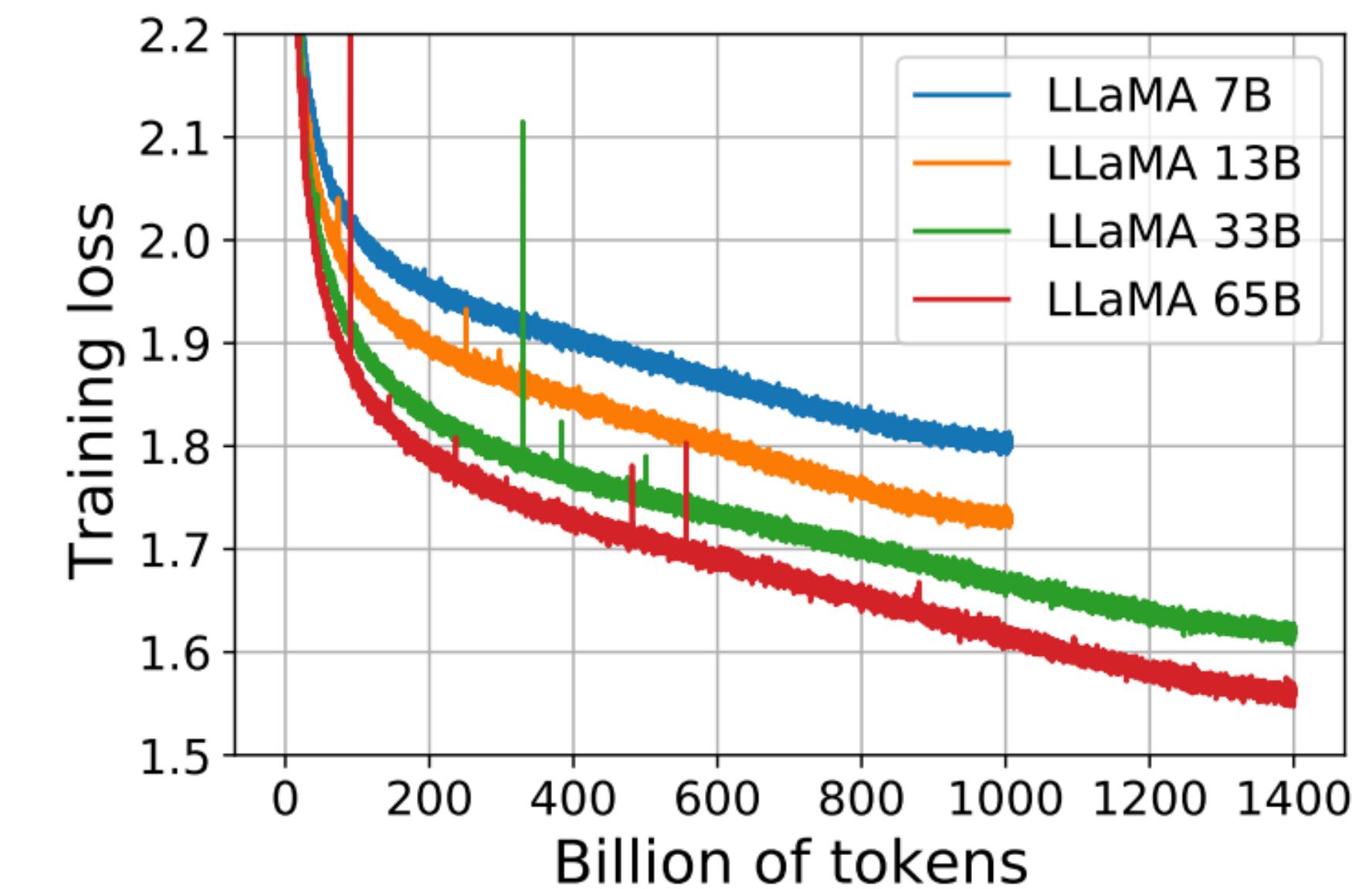


Figure 1: Training loss over train tokens for the 7B, 13B, 33B, and 65 models. LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

# LLaMA

---

- ▶ Trained only on publicly available data:
  - ▶ English CommonCrawl
  - ▶ C4 (another CommonCrawl dataset)
  - ▶ GitHub (from Google BigQuery)
  - ▶ Wikipedia of 20 languages,
  - ▶ Gutenberg and Books3 (from ThePile)
  - ▶ ArXiv (latex files)
  - ▶ StackExchange.
- ▶ Split all numbers into individual digits, and fall back to bytes for unknown UTF-8 characters

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

# LLaMA

---

- ▶ LLaMA-13B matches and outperforms OPT and (old) GPT-3 for zero-shot and few-shot performance

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	<b>88.0</b>	82.3	-	83.4	<b>81.1</b>	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	<b>80.0</b>	<b>57.8</b>	58.6
	65B	85.3	<b>82.8</b>	<b>52.3</b>	<b>84.2</b>	77.0	78.9	56.0	<b>60.2</b>

Table 3: **Zero-shot performance on Common Sense Reasoning tasks.**

		0-shot	1-shot	5-shot	64-shot
GPT-3	175B	14.6	23.0	-	29.9
Gopher	280B	10.1	-	24.5	28.2
Chinchilla	70B	16.6	-	31.5	35.5
PaLM	8B	8.4	10.6	-	14.6
	62B	18.1	26.5	-	27.6
	540B	21.2	29.3	-	39.6
LLaMA	7B	16.8	18.7	22.0	26.1
	13B	20.1	23.4	28.1	31.9
	33B	<b>24.9</b>	28.3	32.9	36.0
	65B	23.8	<b>31.0</b>	<b>35.0</b>	<b>39.9</b>

Table 4: **NaturalQuestions.** Exact match performance.

# LLaMA

- ▶ Transformer variations that have been used in different LLMs

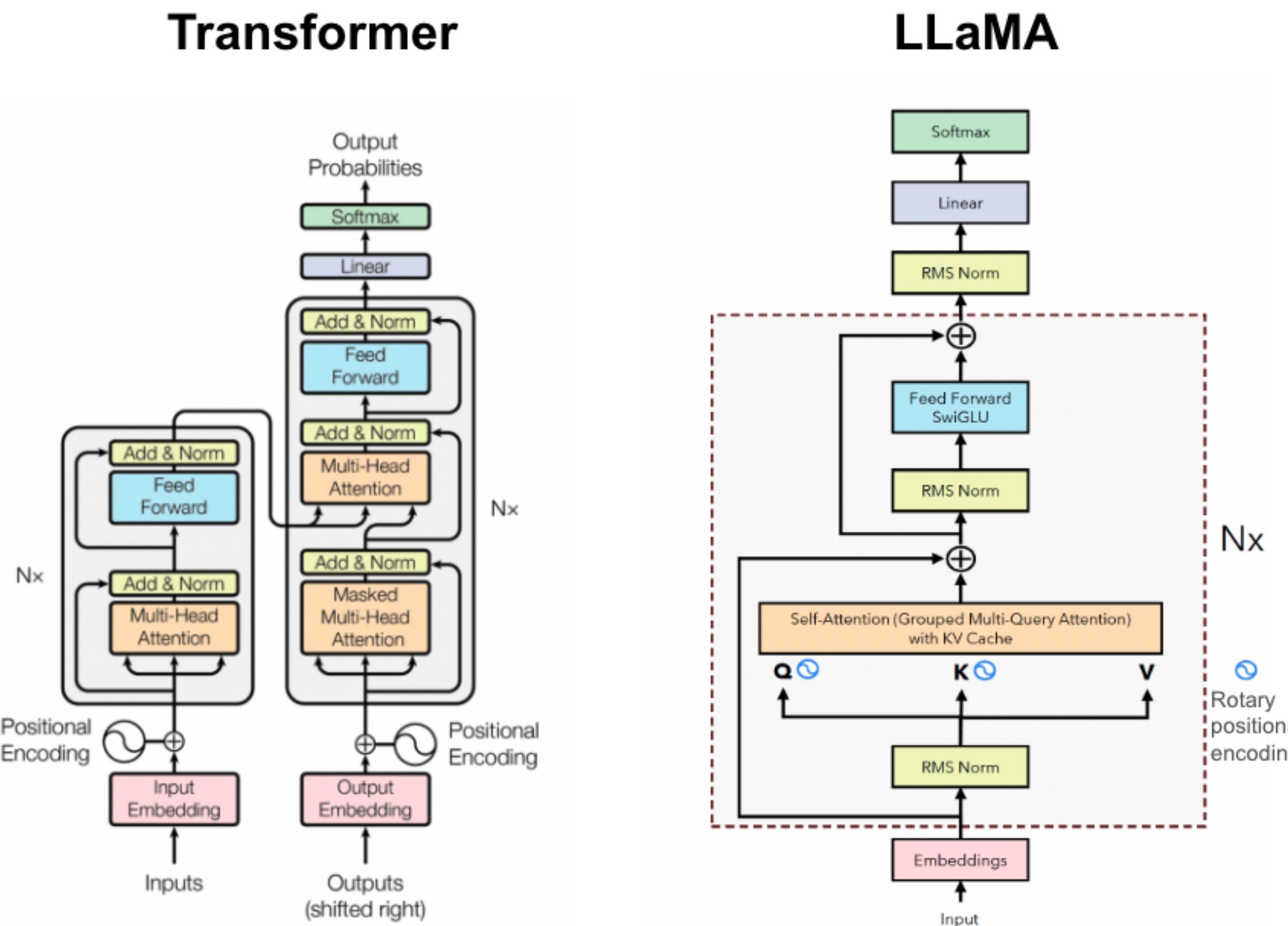


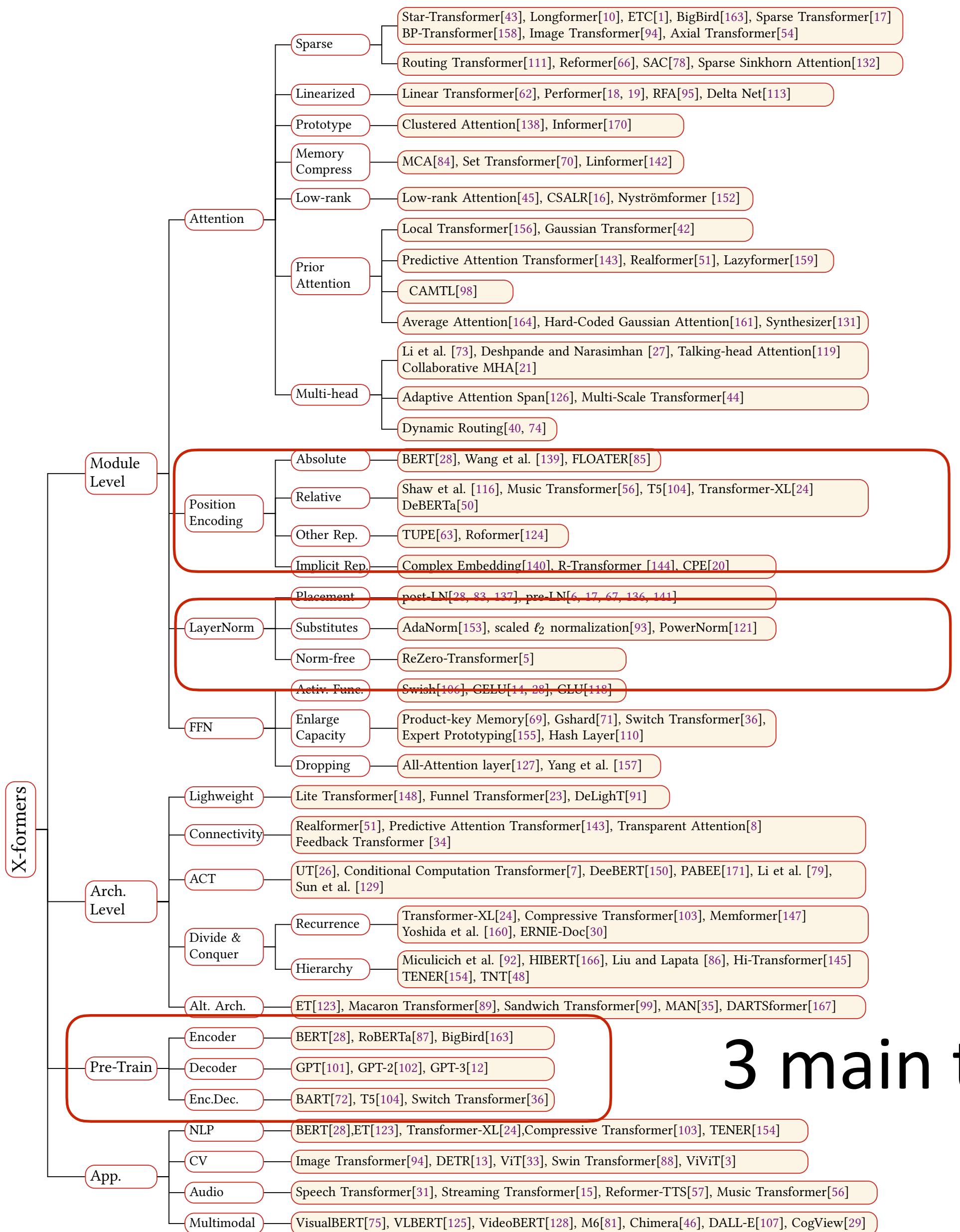
Image Credit: Rajesh Kavadiki

# LLaMA

---

- ▶ Transformer variations that have been used in different LLMs
- ▶ Pre-normalization layer using RMSNorm
- ▶ SwiGLU activation function — combines Swish and Gated Linear Unit (GLU), also used in Google's PaLM model
- ▶ Rotary positional embeddings (RoPE)
- ▶ AdamW Optimizer

# Transformer Variants



3 main types: encoder, decoder, enc-dec

Fig. 3. Taxonomy of Transformers

# Positional Embeddings

# LayerNorm

Lin et al. (2021)

# Normalization

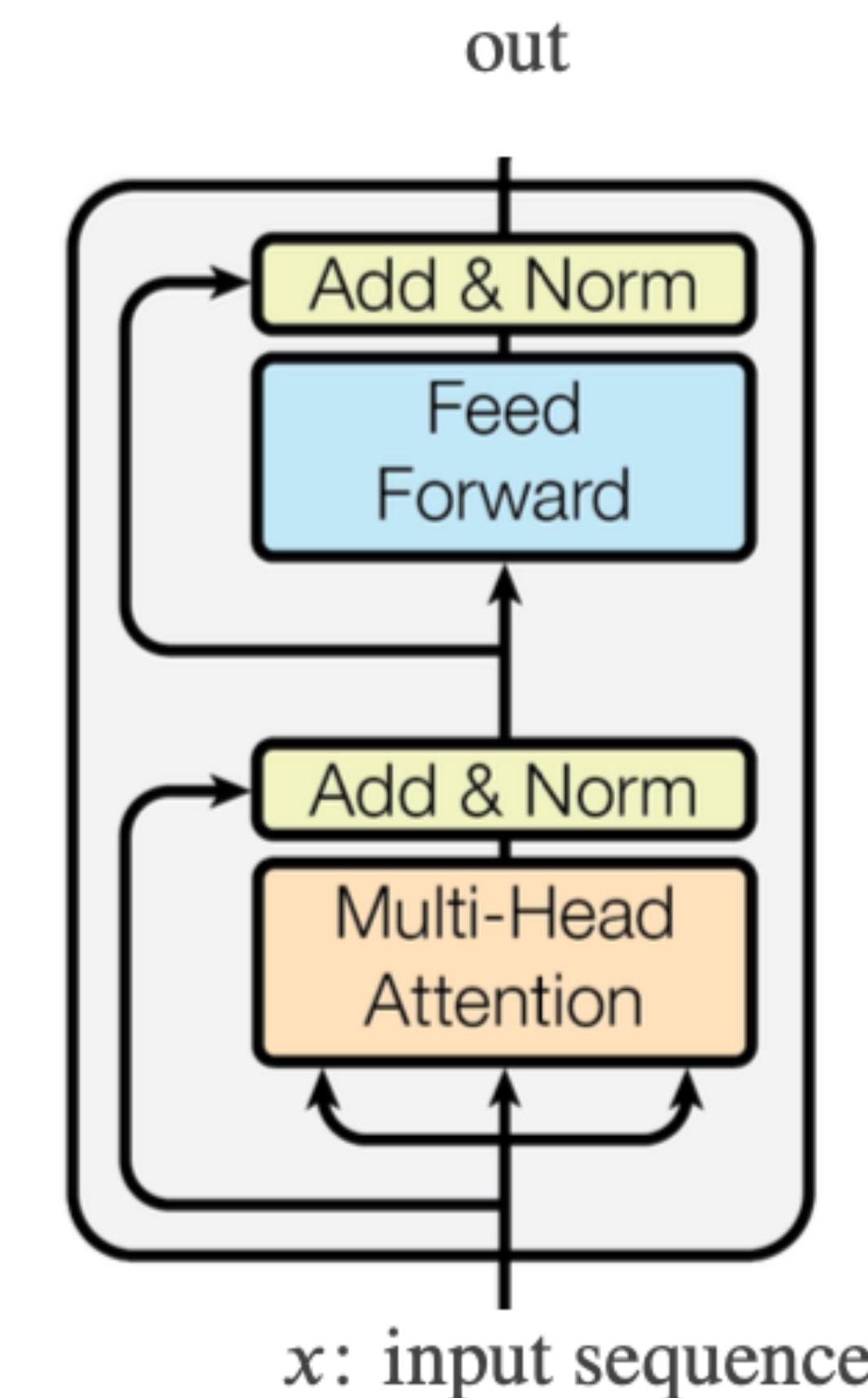
# Pre-norm vs. Post-norm

- Original Transformer — “Add” before “Norm”, or “Norm” before “Add”?

$$\text{LayerNorm}(x + \text{SubLayer}(x))$$

or

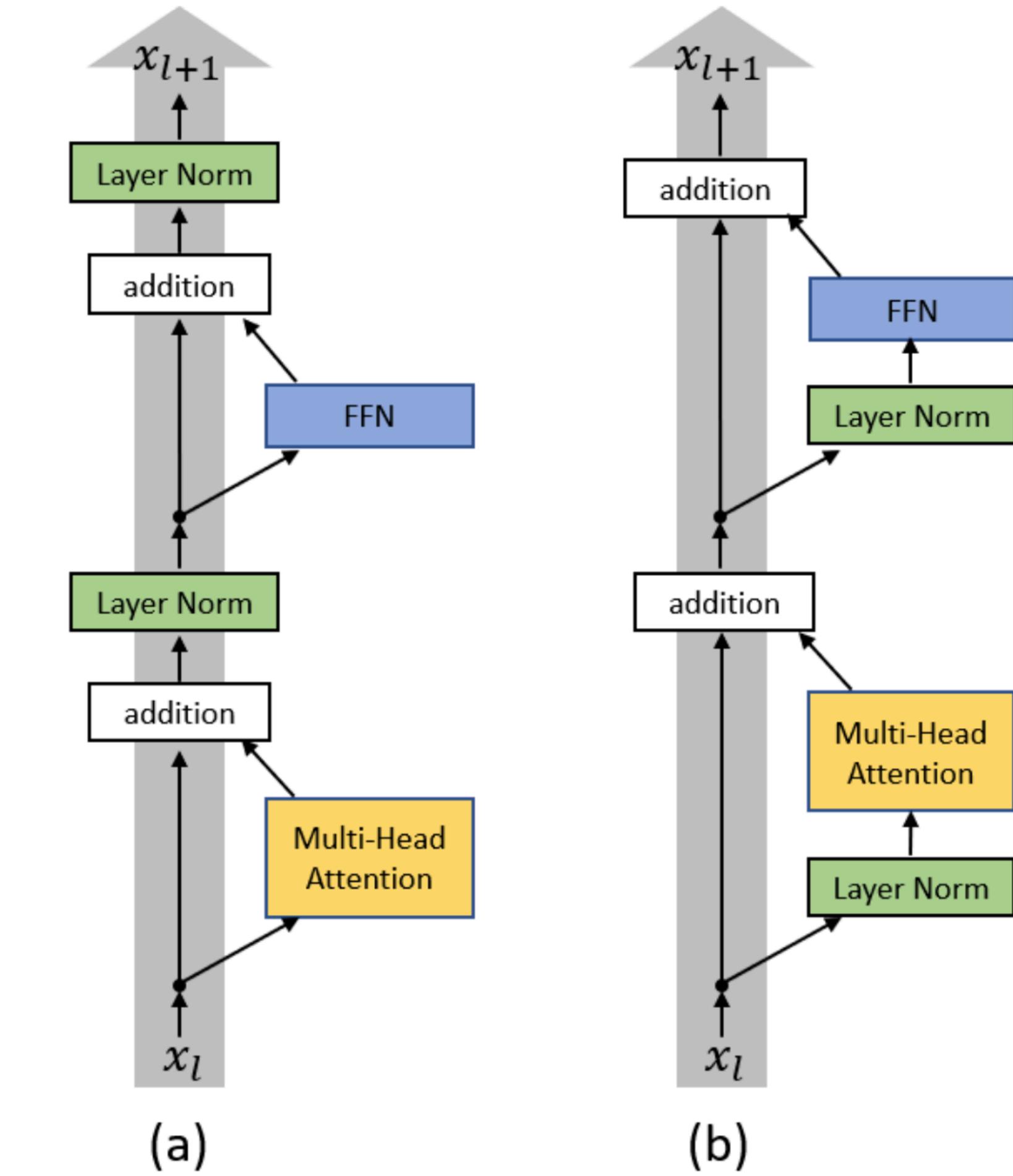
$$x + \text{SubLayer}(\text{LayerNorm}(x))$$



(Baevski & Auli, 2018; Child et al., 2019; Wang et al., 2019)

# Pre-norm vs. Post-norm

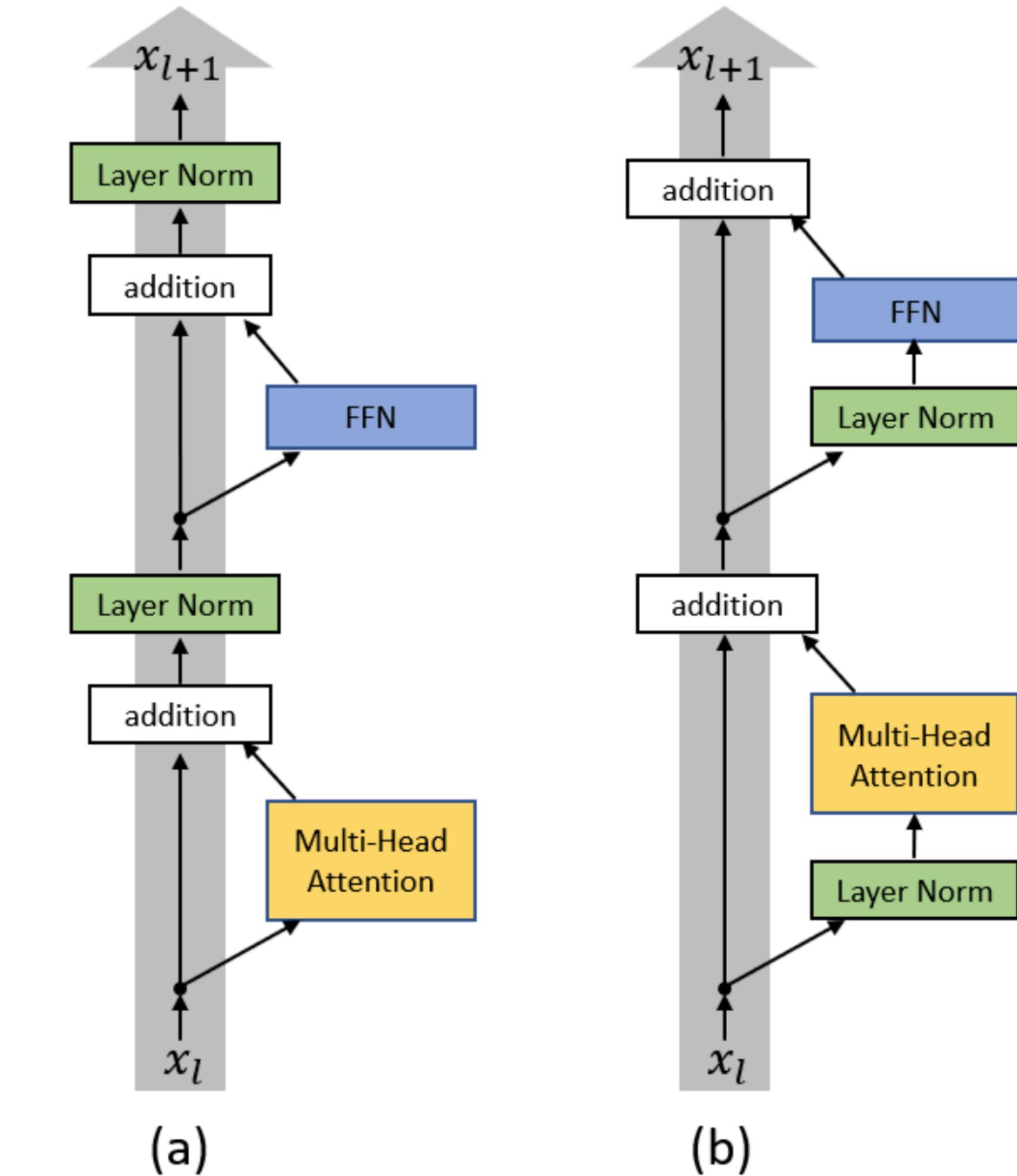
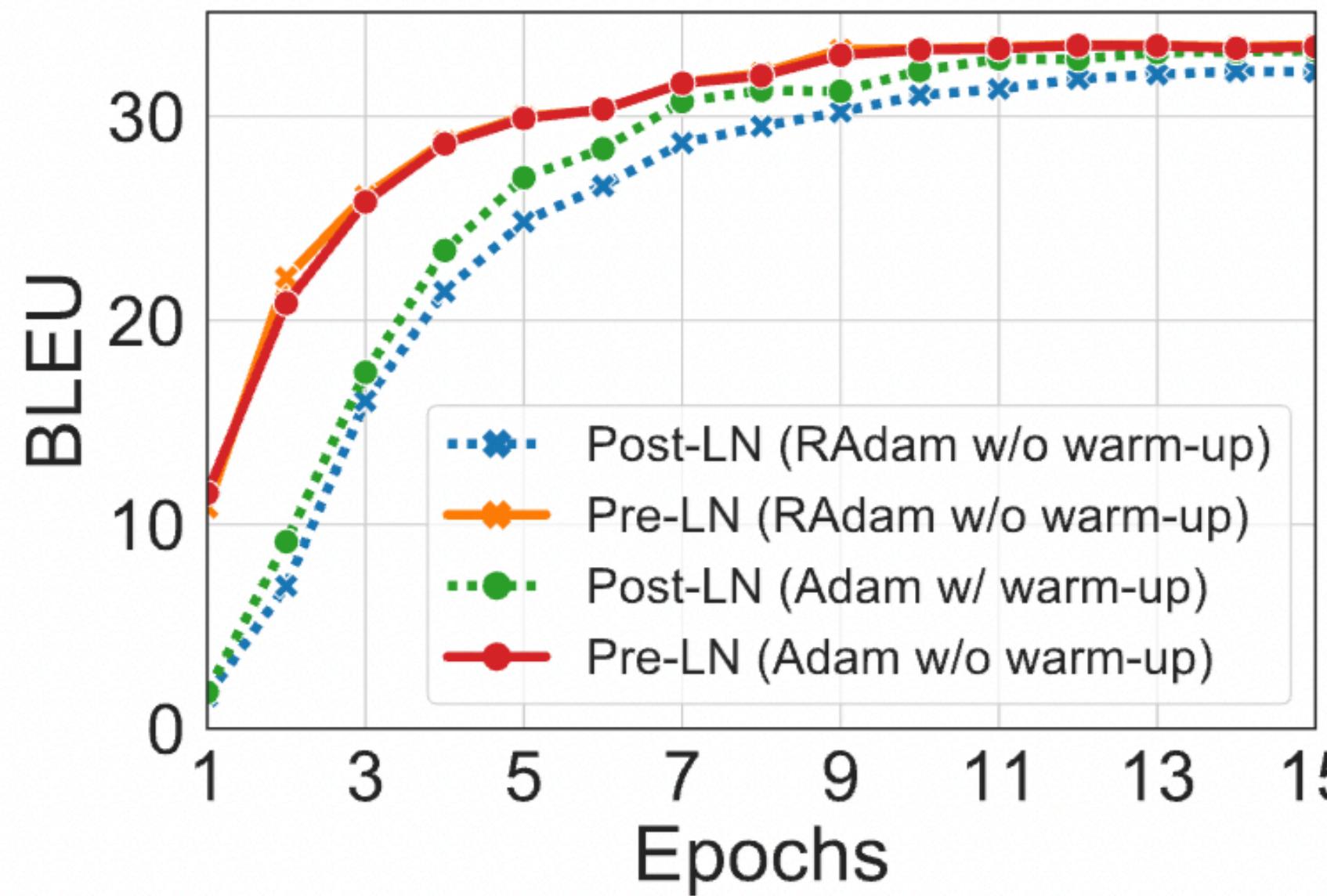
- ▶ Pre-normalization Transformer (b) to have better-behaved gradients at initialization than in the original Transformer (a)



Xiong et al. (2020)

# Pre-norm vs. Post-norm

- ▶ Pre-normalization Transformer (b) to have better-behaved gradients at initialization than in the original Transformer (a)
- ▶ In (b), LayerNorm does not disrupt residual

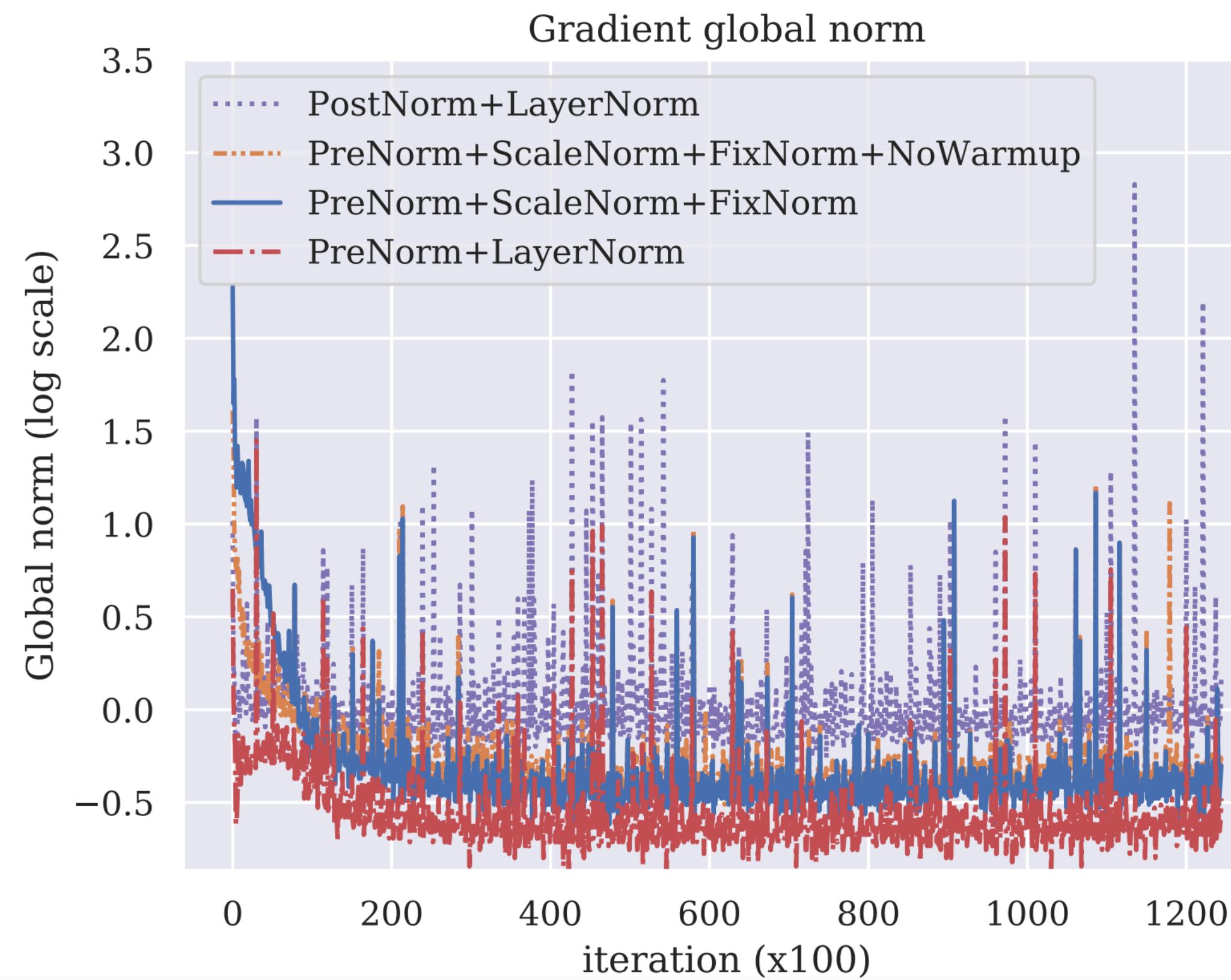


Xiong et al. (2020)

# Pre-norm vs. Post-norm

---

- ▶ Post-norm produces noisy gradients with many tall spikes, needs warm up
- ▶ Pre-norm has fewer noisy gradients with smaller sizes, even without warmup



# RMSNorm

- RMSNorm (a) instead of standard LayerNorm (b)

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

root mean square

(a)

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

mean

variance

(b)

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

(used in LLaMA1/2/3, PaLM, T5 ...)

(used in GPT1/2/3, OPT ...)

Zhang and Sennrich (2019)

# RMSNorm

- ▶ RMSNorm (a) instead of standard LayerNorm (b)

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

root mean square

(a)

only re-scaling invariance, skip re-centering  
more computationally efficient

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

mean

variance

(b)

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

Zhang and Sennrich (2019)

# RMSNorm

- ▶ RMSNorm (a) instead of standard LayerNorm (b)

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

root mean square

(a)

only re-scaling invariance, skip re-centering  
more computationally efficient ??

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

mean

variance

(b)

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

Zhang and Sennrich (2019)

# RMSNorm

- ▶ Matrix multiplications make up the majority of GPU FLOPs (and memory)
- ▶ Below is runtime analysis of the encoder layer of (Transformer-based) BERT

Matrix-matrix  
multiplication

softmax & layer  
normalization

*Table 1. Proportions for operator classes in PyTorch.*

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

biases, dropout, activations, and residual connections

more computationally efficient ??

Ivanov et al. (2021)

# RMSNorm

---

- ▶ Yet, RMSNorm runtime gains have been observed in papers

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	$223M$	$11.1T$	3.50	$2.182 \pm 0.005$	1.838	71.66	17.78	23.02	26.62
RMS Norm	$223M$	$11.1T$	3.68	$2.167 \pm 0.008$	<b>1.821</b>	<b>75.45</b>	<b>17.94</b>	<b>24.07</b>	<b>27.14</b>

more computationally efficient ??

Narang et al. (2021)

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
GeLU	223M	11.1T	3.58	2.179 ± 0.003	1.838	<b>75.79</b>	<b>17.86</b>	<b>25.13</b>	26.47
Swish	223M	11.1T	3.62	2.186 ± 0.003	1.847	<b>73.77</b>	17.74	<b>24.34</b>	<b>26.75</b>
ELU	223M	11.1T	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	223M	11.1T	3.59	2.174 ± 0.003	<b>1.814</b>	<b>74.20</b>	<b>17.42</b>	24.34	<b>27.12</b>
GeGLU	223M	11.1T	3.55	2.130 ± 0.006	<b>1.792</b>	<b>75.96</b>	<b>18.27</b>	<b>24.87</b>	<b>26.87</b>
ReGLU	223M	11.1T	3.57	2.145 ± 0.004	<b>1.803</b>	<b>76.17</b>	<b>18.36</b>	<b>24.87</b>	<b>27.02</b>
SeLU	223M	11.1T	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	223M	11.1T	3.53	2.127 ± 0.003	<b>1.789</b>	<b>76.00</b>	<b>18.20</b>	<b>24.34</b>	<b>27.02</b>
LiGLU	223M	11.1T	3.59	2.149 ± 0.005	<b>1.798</b>	<b>75.34</b>	<b>17.97</b>	<b>24.34</b>	26.53
Sigmoid	223M	11.1T	3.63	2.291 ± 0.019	1.867	<b>74.31</b>	17.51	23.02	26.30
Softplus	223M	11.1T	3.47	2.207 ± 0.011	1.850	<b>72.45</b>	17.65	<b>24.34</b>	<b>26.89</b>
RMS Norm	223M	11.1T	3.68	2.167 ± 0.008	<b>1.821</b>	<b>75.45</b>	<b>17.94</b>	<b>24.07</b>	<b>27.14</b>
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
24 layers, $d_{ff} = 1536, H = 6$	224M	11.1T	3.33	2.200 ± 0.007	1.843	<b>74.89</b>	17.75	<b>25.13</b>	<b>26.89</b>
18 layers, $d_{ff} = 2048, H = 8$	223M	11.1T	3.38	2.185 ± 0.005	<b>1.831</b>	<b>76.45</b>	16.83	<b>24.34</b>	<b>27.10</b>
8 layers, $d_{ff} = 4608, H = 18$	223M	11.1T	3.69	2.190 ± 0.005	1.847	<b>74.58</b>	17.69	<b>23.28</b>	<b>26.85</b>
6 layers, $d_{ff} = 6144, H = 24$	223M	11.1T	3.70	2.201 ± 0.010	1.857	<b>73.55</b>	17.59	<b>24.60</b>	<b>26.66</b>
Block sharing	65M	11.1T	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	45M	9.4T	4.21	2.631 ± 0.305	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	20M	9.1T	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	170M	11.1T	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	144M	11.1T	3.70	2.352 ± 0.029	2.082	67.93	16.13	<b>23.81</b>	26.08
Factorized Embedding	227M	9.4T	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	202M	9.1T	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	248M	11.1T	3.55	2.192 ± 0.002	1.840	<b>71.70</b>	17.72	<b>24.34</b>	26.49
Tied decoder input and output embeddings	248M	11.1T	3.57	2.187 ± 0.007	<b>1.827</b>	<b>74.86</b>	17.74	<b>24.87</b>	<b>26.67</b>
Untied embeddings	273M	11.1T	3.53	2.195 ± 0.005	<b>1.834</b>	<b>72.99</b>	17.58	<b>23.28</b>	26.48
Adaptive input embeddings	204M	9.2T	3.55	2.250 ± 0.002	1.899	66.57	16.21	<b>24.07</b>	<b>26.66</b>
Adaptive softmax	204M	9.2T	3.60	2.364 ± 0.005	1.982	<b>72.91</b>	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.8T	3.43	2.229 ± 0.009	1.914	<b>71.82</b>	17.10	23.02	25.72
Mixture of softmaxes	232M	16.3T	2.24	2.227 ± 0.017	<b>1.821</b>	<b>76.77</b>	17.62	22.75	<b>26.82</b>
Transparent attention	223M	11.1T	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	<b>26.80</b>
Dynamic convolution	257M	11.8T	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.4T	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.9T	3.09	2.220 ± 0.003	1.863	<b>73.67</b>	10.76	<b>24.07</b>	26.58
Synthesizer (dense)	224M	11.4T	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	<b>26.63</b>
Synthesizer (dense plus)	243M	12.6T	3.22	2.191 ± 0.010	1.840	<b>73.98</b>	16.96	<b>23.81</b>	<b>26.71</b>
Synthesizer (dense plus alpha)	243M	12.6T	3.01	2.180 ± 0.007	<b>1.828</b>	<b>74.25</b>	17.02	<b>23.28</b>	26.61
Synthesizer (factorized)	207M	10.1T	3.94	2.341 ± 0.017	1.968	62.78	15.39	<b>23.55</b>	26.42
Synthesizer (random)	254M	10.1T	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.0T	3.63	2.189 ± 0.004	1.842	<b>73.32</b>	17.04	<b>24.87</b>	26.43
Synthesizer (random plus alpha)	292M	12.0T	3.42	2.186 ± 0.007	<b>1.828</b>	<b>75.24</b>	17.08	<b>24.08</b>	26.39
Universal Transformer	84M	40.0T	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.7T	3.20	2.148 ± 0.006	<b>1.785</b>	<b>74.55</b>	<b>18.13</b>	<b>24.08</b>	<b>26.94</b>
Switch Transformer	1100M	11.7T	3.18	2.135 ± 0.007	<b>1.758</b>	<b>75.38</b>	<b>18.02</b>	<b>26.19</b>	<b>26.81</b>
Funnel Transformer	223M	1.9T	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.0T	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	421M	386.6T	0.25	2.155 ± 0.003	<b>1.798</b>	<b>75.16</b>	17.04	<b>23.55</b>	<b>26.73</b>

Table 1: Results for all architecture variants. The baseline model is the vanilla Transformer with relative attention. The early loss represents the mean and standard deviation of perplexity at 65,536 steps. The final perplexity is reported at the end of pre-training (524,288 steps). SGLUE refers to SuperGLUE and WebQ refers to WebQuestions dataset. We report average, ROUGE-2, accuracy, and BLEU score for SuperGLUE, XSum, WebQuestions, and WMT EnDe, respectively, on the validation sets. **Note:** Results on WMT English to German are reported **without any pre-training**. The scores which outperform the vanilla Transformer are highlighted in **boldface**.

# Bias Term

---

- ▶ Standard feedforward network layer:

$$\text{FFN}(x, W_1, W_2, b_1, b_2) = f(xW_1 + b_1)W_2 + b_2$$

- ▶ Original Transformer uses ReLU as activation function

$$\text{FFN}(x, W_1, W_2, b_1, b_2) = \max(0, xW_1 + b_1)W_2 + b_2$$

- ▶ Many implementations (if they are not gated), e.g. T5, PaLM, DALL-E-mini ...

$$\text{FFN}_{\text{ReLU}}(x, W_1, W_2) = \max(xW_1, 0)W_2$$

# Recap so far ...

---

- ▶ Basically everyone does pre-norm
  - Intuition: keep the good parts of residual connections
  - Observations: nicer gradient propagation, fewer spike
- ▶ Most people do RMSnorm
  - In practice, works as well as LayerNorm
  - But, has fewer parameters to move around, saves on wallclock time
- ▶ People more generally drop bias terms
  - since the compute/param tradeoffs are not great.
  - without compromising performance

# Activation Function

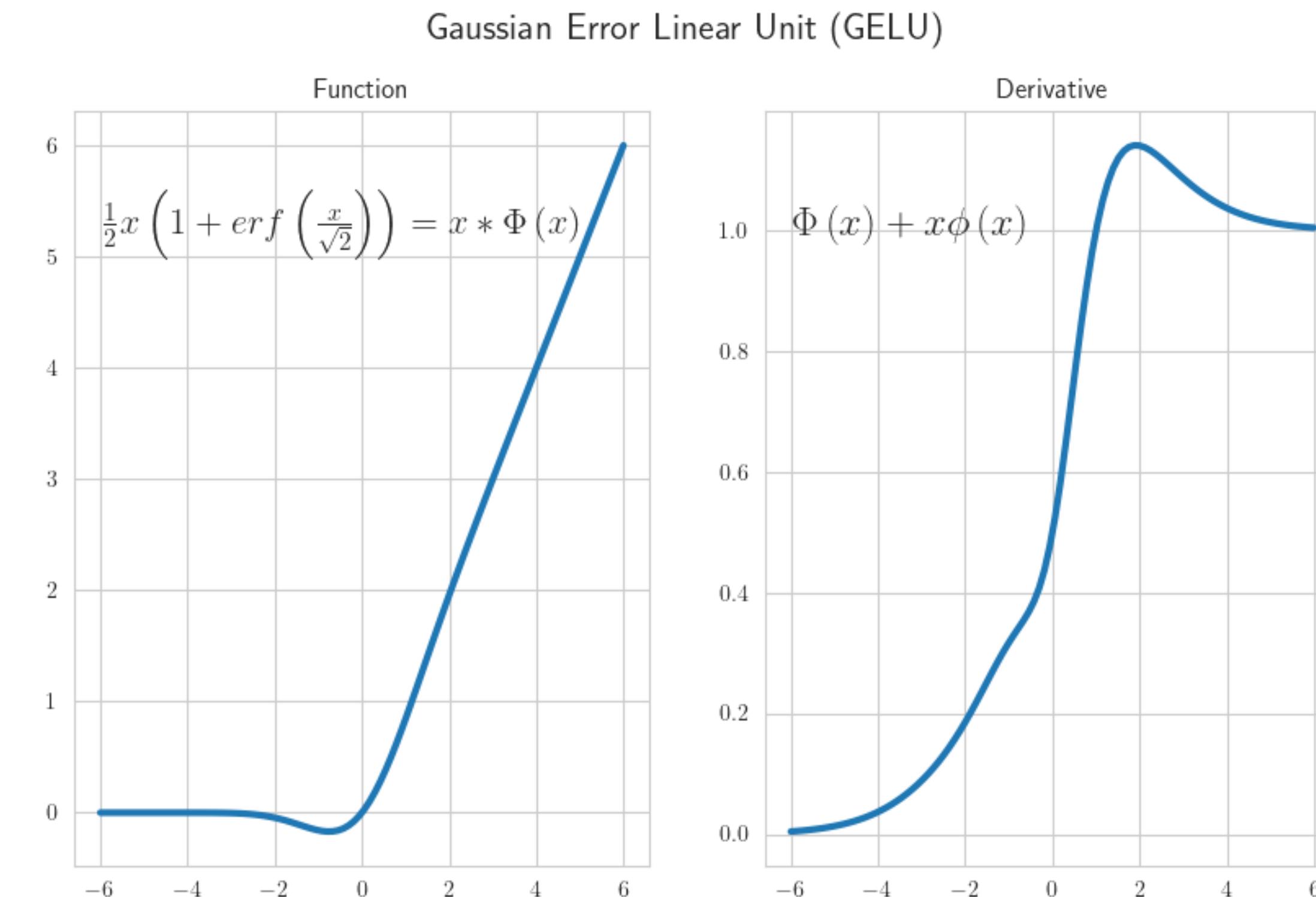
# Activation Functions

---

- ▶ A lot different ones used in training LLMs:

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

- ▶ Not much consensus ...



# SwiGLU in LLaMA

---

- ▶ SwiGLU activation function — combines Swish and Gated Linear Unit (GLU), also used in Google's PaLM model
- ▶ Feedforward layer in the Transformer using ReLU (with no bias shown here):

$$\text{FFN}_{\text{ReLU}}(x, W_1, W_2) = \max(xW_1, 0)W_2$$

- ▶ Replace ReLU by Swish:

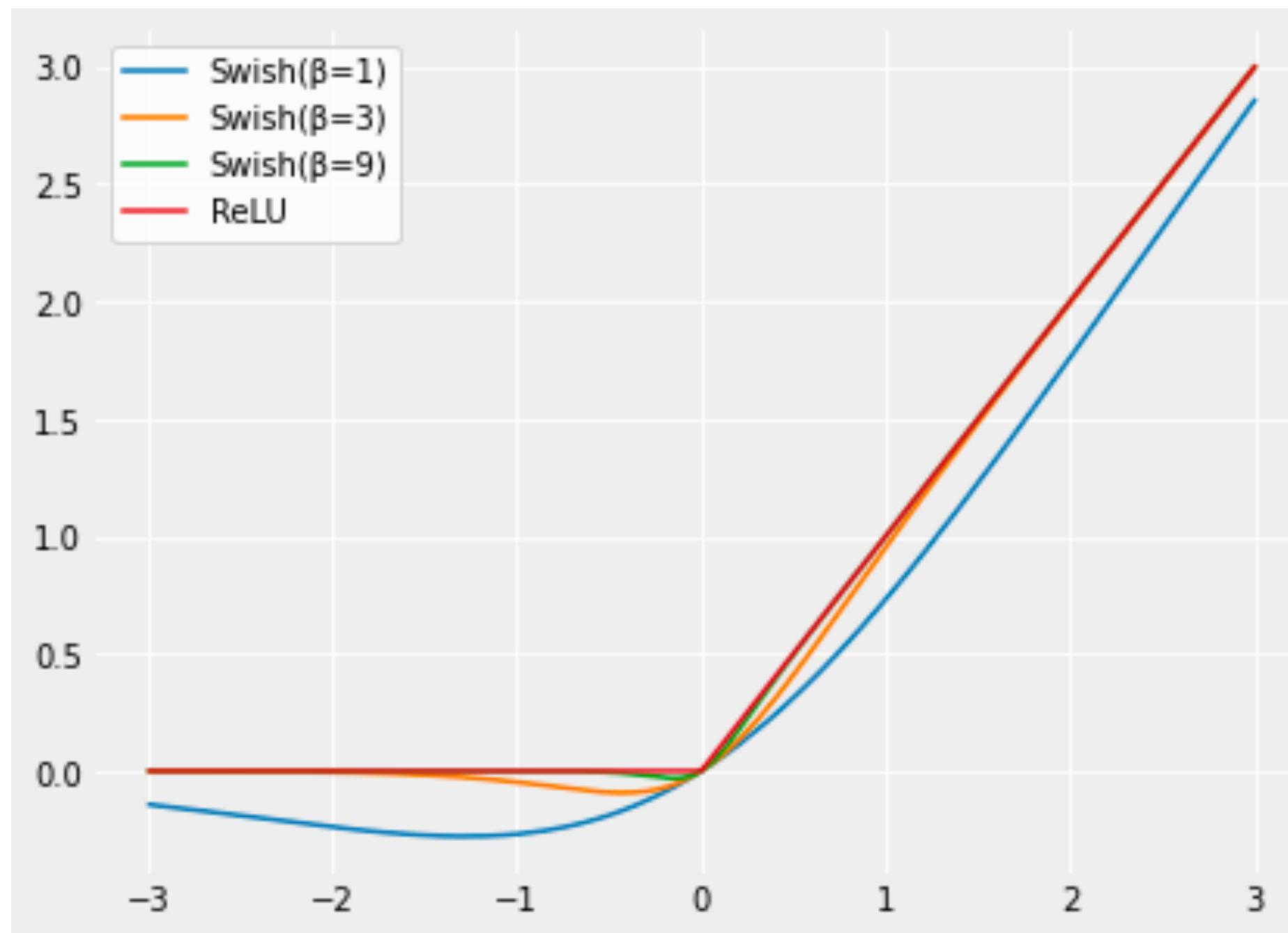
$$\text{FFN}_{\text{Swish}}(x, W_1, W_2) = \text{Swish}_1(xW_1)W_2$$

Shazeer (2020)

# Swish Activation

---

- ▶ Swish can be loosely viewed as a smooth function which nonlinearly interpolates between the linear function and ReLU

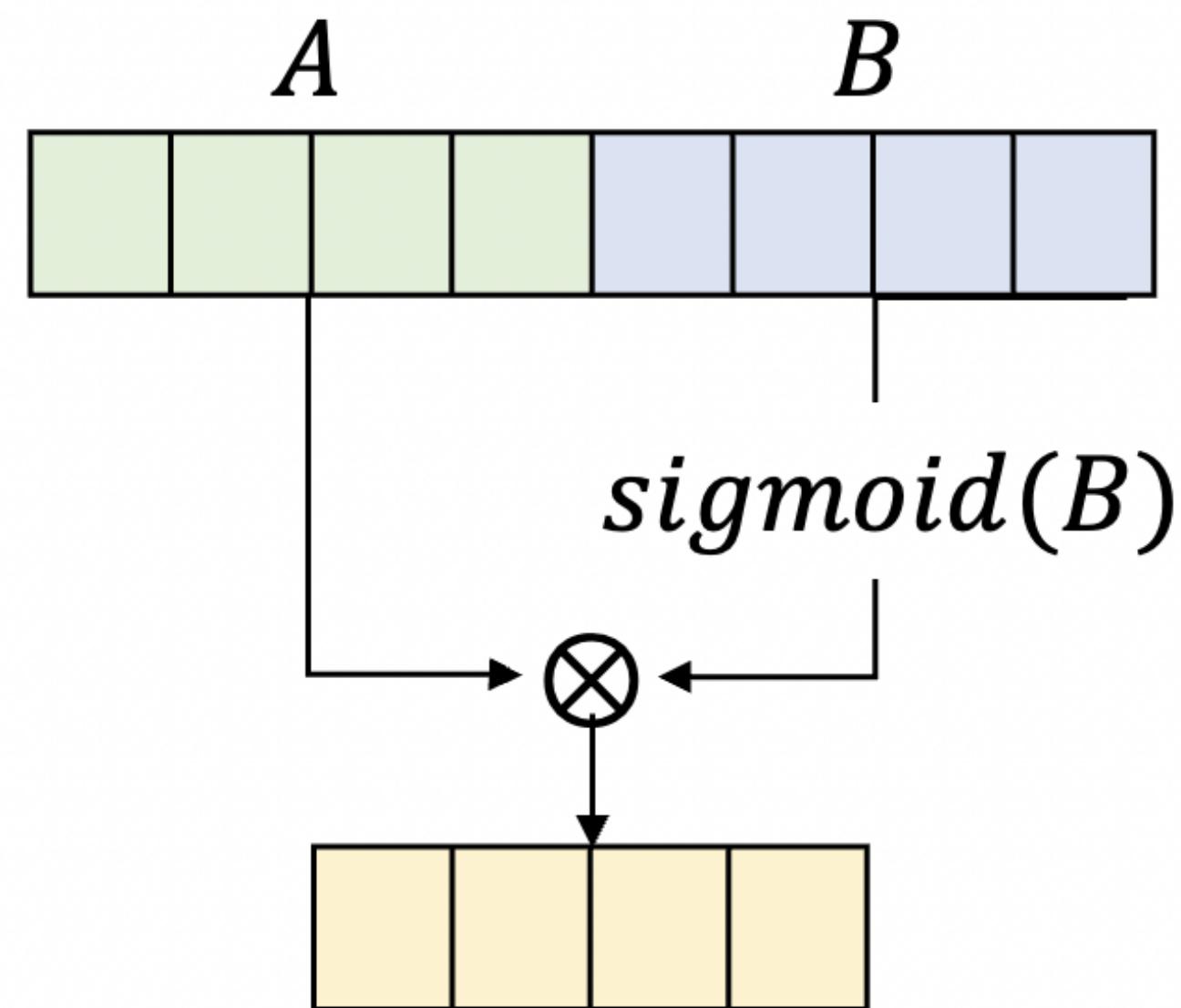


$$\begin{aligned} \text{Swish}_\beta(x) &= x * \text{sigmoid}(\beta x) \\ &= \frac{x}{1+e^{-\beta x}} \end{aligned}$$

# Gated Linear Unit (GLU)

---

- ▶ Similar to the gating mechanism in LSTM.
- ▶ Element-wise product of two linear transformations of the input, one is sigmoid-activated.



$$\text{GLU}(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$

# Sigmoid of a Vector

---

$$\begin{array}{c} \textbf{X} \\ \begin{bmatrix} 3 \\ 1.75 \\ -2 \\ 0.5 \end{bmatrix} \end{array} \xrightarrow{\text{Sigmoid}} \begin{array}{c} \textbf{Out} \\ \begin{bmatrix} 0.95 \\ 0.85 \\ 0.12 \\ 0.62 \end{bmatrix} \end{array}$$

Not a probability distribution

$$\begin{array}{c} \textbf{X} \\ \begin{bmatrix} 3 \\ 1.75 \\ -2 \\ 0.5 \end{bmatrix} \end{array} \xrightarrow{\text{SoftMax}} \begin{array}{c} \textbf{Out} \\ \begin{bmatrix} 0.725 \\ 0.21 \\ 0.005 \\ 0.06 \end{bmatrix} \end{array}$$

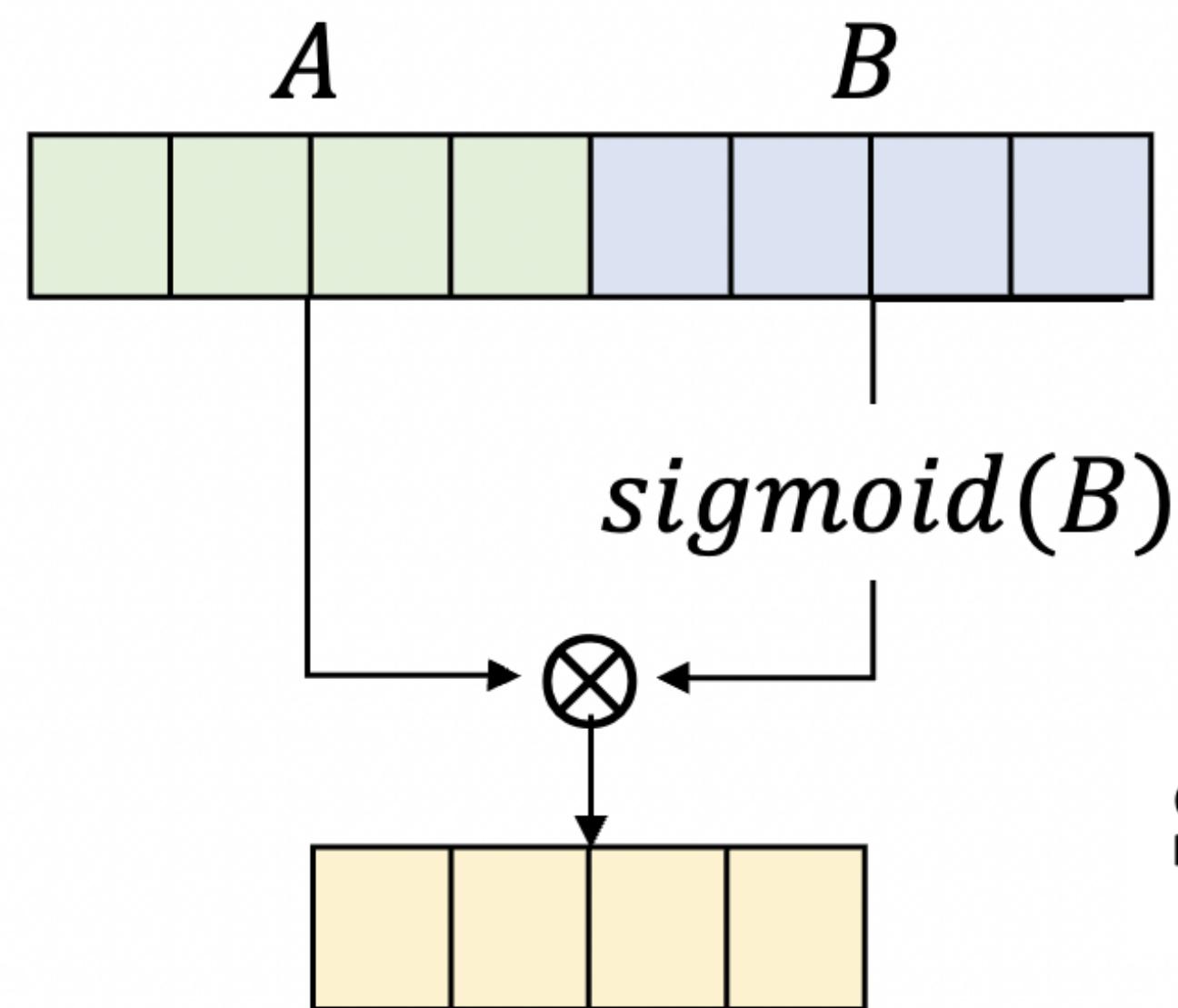
Probability distribution

Image Credit: Gabriel Furnieles

# SwiGLU

---

- ▶ SwiGLU activation function – combines Swish and Gated Linear Unit (GLU), also used in Google's PaLM model



$$\text{GLU}(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_\beta(xW + b) \otimes (xV + c)$$

# SwiGLU in LLaMA

---

- ▶ SwiGLU activation function — combines Swish and Gated Linear Unit (GLU), also used in Google's PaLM model
- ▶ Feedforward layer in the Transformer using ReLU (with no bias shown here):

$$\text{FFN}_{\text{ReLU}}(x, W_1, W_2) = \max(xW_1, 0)W_2$$

- ▶ Replace ReLU by Swish or SwiGLU:

$$\text{FFN}_{\text{Swish}}(x, W_1, W_2) = \text{Swish}_1(xW_1)W_2$$

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

Shazeer (2020)

# LLaMA

---

- ▶ SwiGLU activation function — combines Swish and Gated Linear Unit (GLU), also used in Google's PaLM model

Training Steps	65,536	524,288
FFN <sub>ReLU</sub> ( <i>baseline</i> )	1.997 (0.005)	1.677
FFN <sub>GELU</sub>	1.983 (0.005)	1.679
FFN <sub>Swish</sub>	1.994 (0.003)	1.683
FFN <sub>GLU</sub>	1.982 (0.006)	1.663
FFN <sub>Bilinear</sub>	1.960 (0.005)	1.648
FFN <sub>GEGLU</sub>	<b>1.942</b> (0.004)	<b>1.633</b>
FFN <sub>SwiGLU</sub>	<b>1.944</b> (0.010)	<b>1.636</b>
FFN <sub>ReGLU</sub>	1.953 (0.003)	1.645

← Held-out log-perplexity  
on C4 corpus (used in T5 model)

Shazeer (2020)

# Gated Linear Unit (GLU)

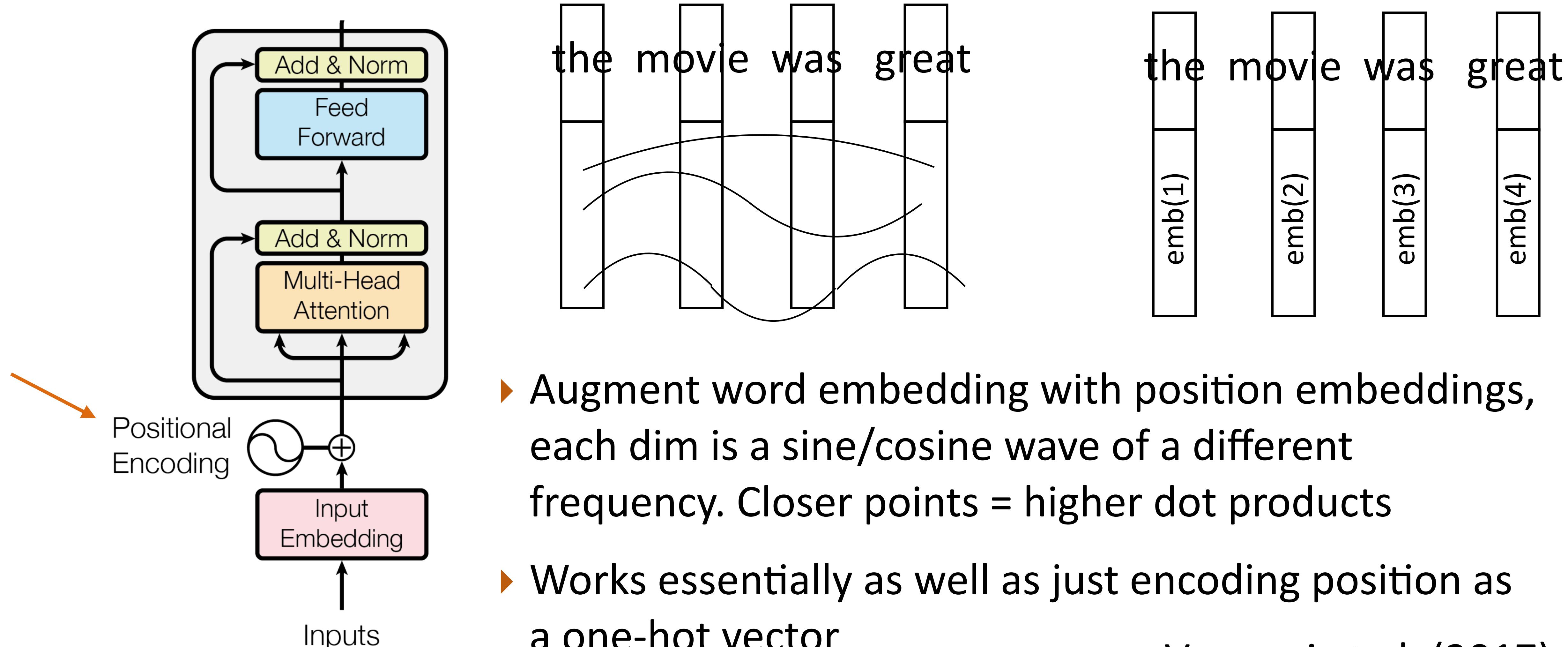
- ▶ GLU variants generally works pretty well

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

# Positional Embeddings

# Positional Embeddings

- ▶ Sine embeddings in the original Transformer:



Vaswani et al. (2017)

# Positional Embeddings

- ▶ Sine embeddings in the original Transformer:

```
1 P = getPositionEncoding(seq_len=100, d=512, n=10000)
2 cax = plt.matshow(P)
3 plt.gcf().colorbar(cax)
```

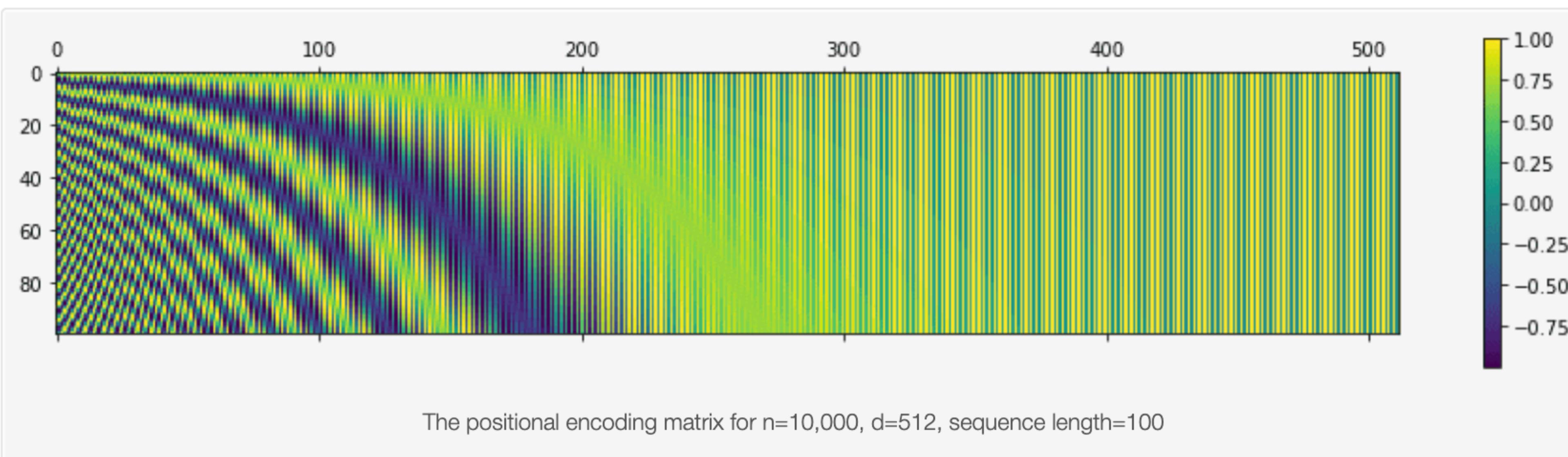
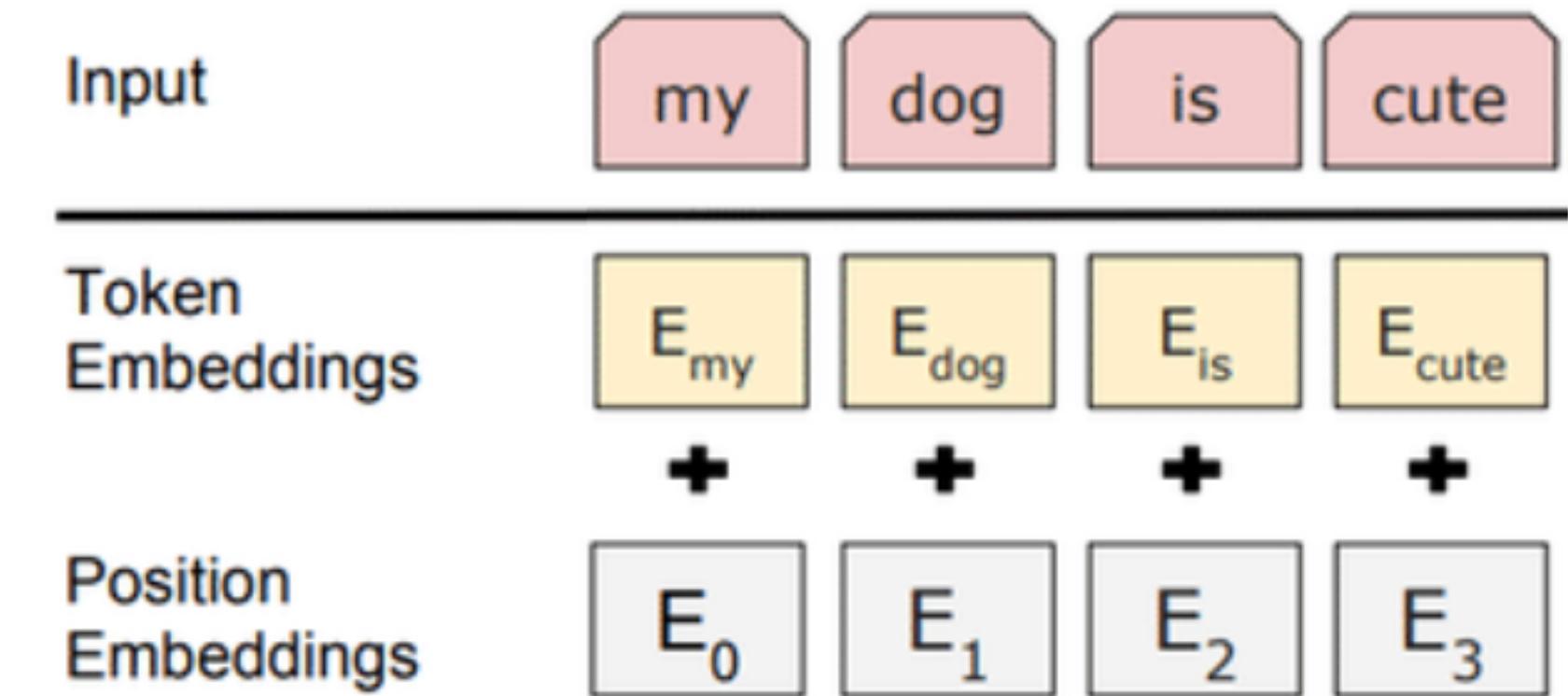


Image Credit: Mehreen Saeed

# Positional Embeddings

---

- ▶ Absolute positional embeddings are added to input token embeddings
  - fixed encoding for each position (e.g., sine embeddings)
  - or learned encoding for each position



- ▶ Limitations:
  - embedding for each position (e.g., 1, 2, 3, ..., 1000, ... 5000)
  - can't generalize well to arbitrary long sequences
  - can't capture relative distance between two tokens

# Positional Embeddings

---

- ▶ Relative positional embedding encodes the distance between tokens

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

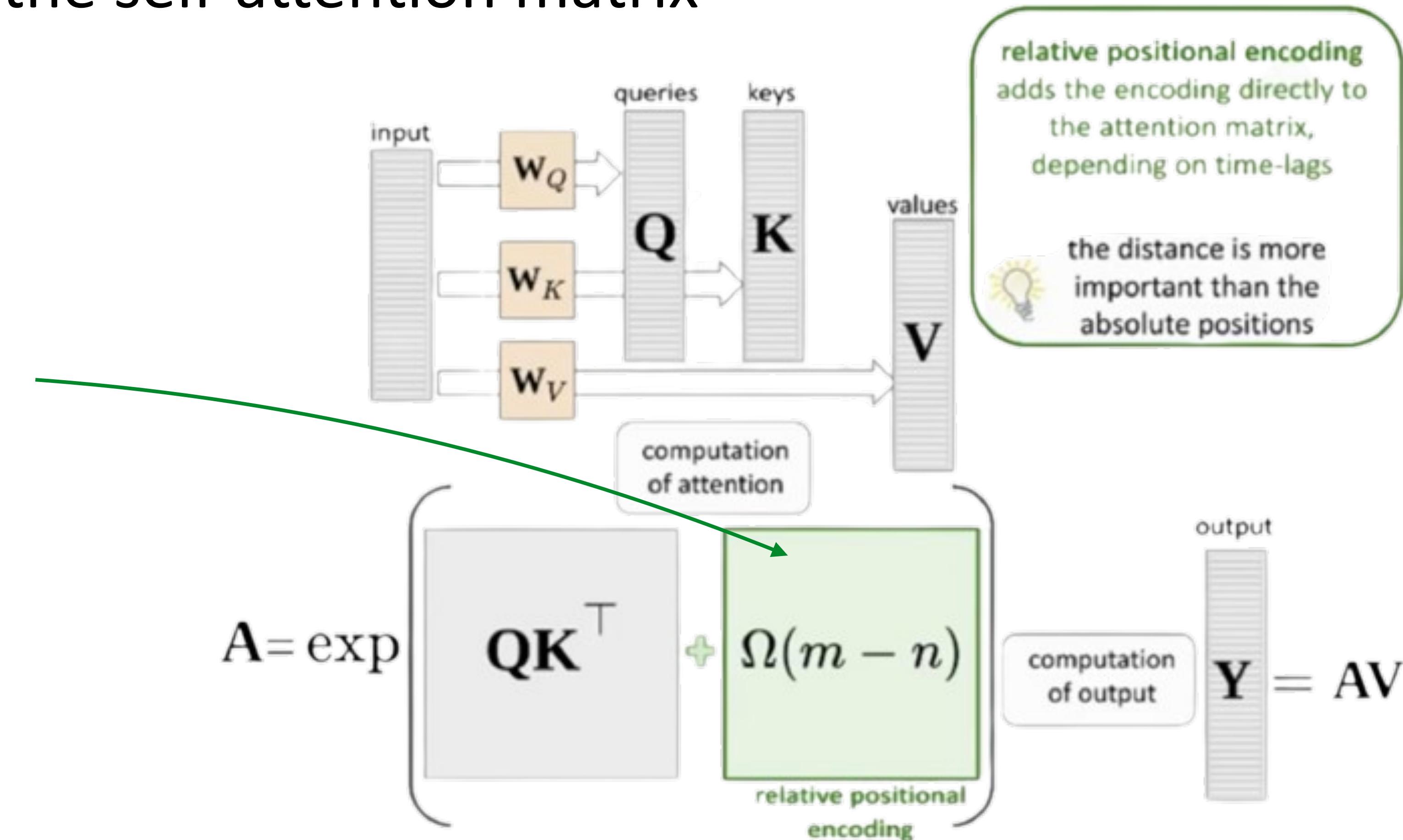
Relative Positions Pattern  
in 5 token Attention matrix

# Positional Embeddings

- ▶ Relative positional embedding encodes the distance between tokens
- ▶ added directly to the self-attention matrix

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

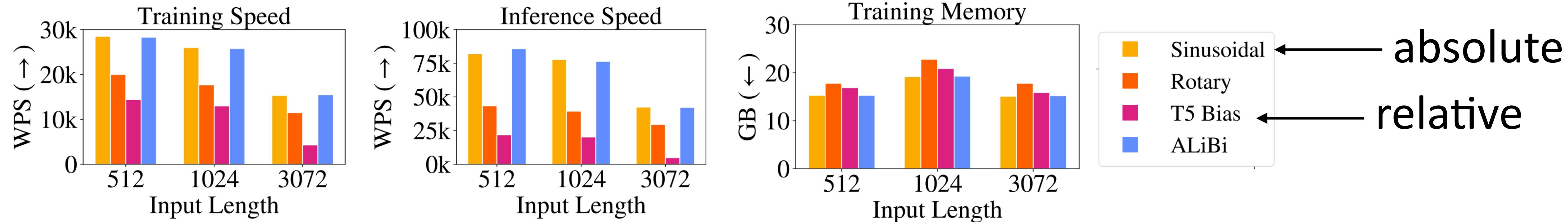
Relative Positions Pattern  
in 5 token Attention matrix



Shaw et al. (2018)

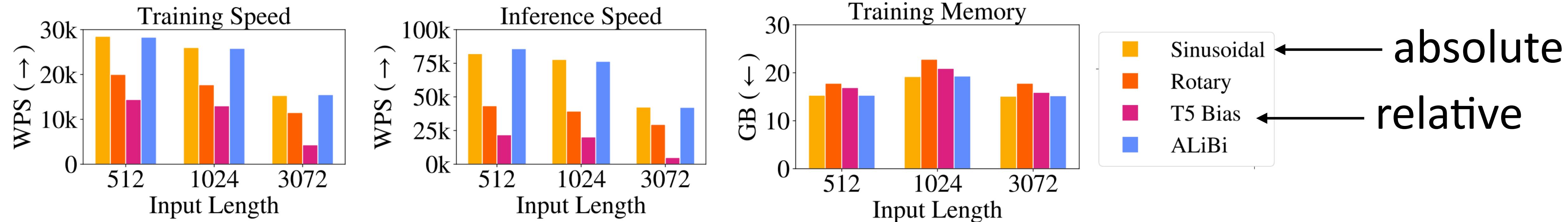
# Positional Embeddings

- ▶ Relative positional embedding encodes the distance between tokens
- ▶ added directly to the self-attention matrix
- ▶ Limitations: slow



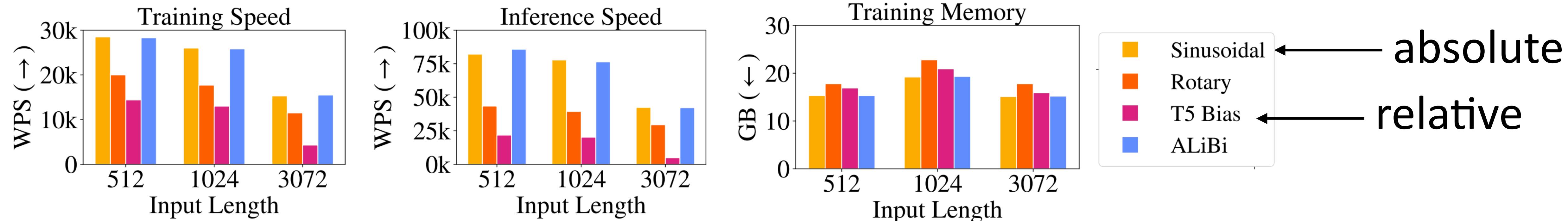
# Positional Embeddings

- ▶ Relative positional embedding encodes the distance between tokens
- ▶ added directly to the self-attention matrix
- ▶ Limitations: slow (why?)



# Positional Embeddings

- ▶ Relative positional embedding encodes the distance between tokens
  - ▶ added directly to the self-attention matrix
  - ▶ Limitations: slow (why? changes KV values all the time, can't do KV caching)



# KV Caching

- ▶ Accelerate LLM inference by reducing redundant computations
- ▶ Have the key and value projections cached

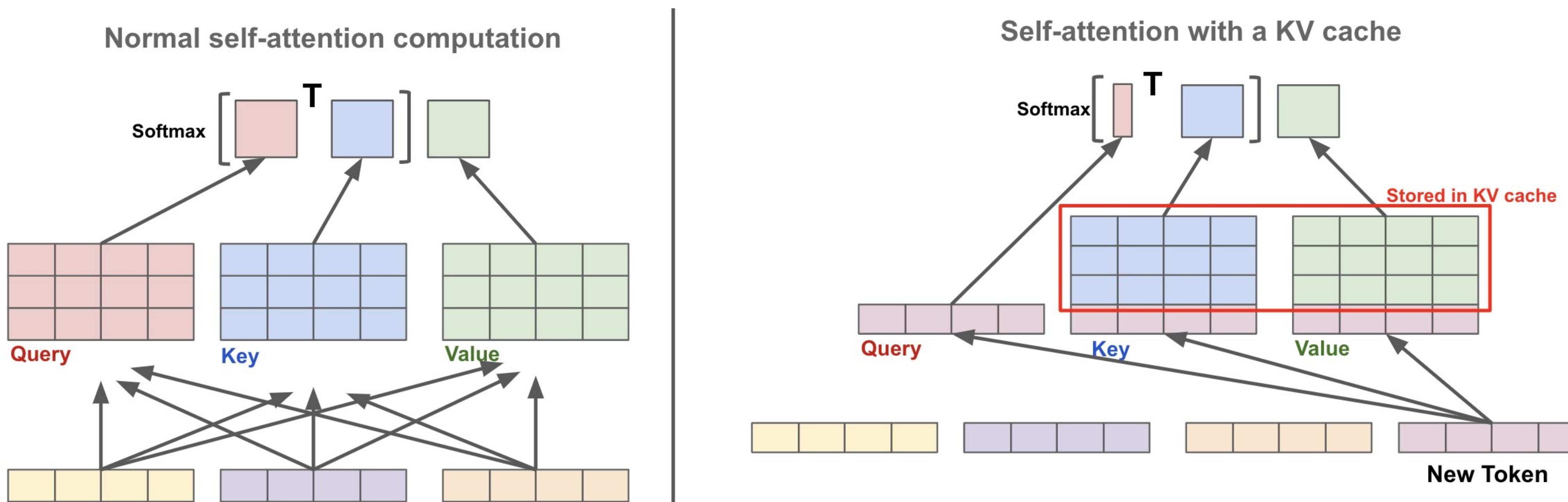


Image Credit: Cameron R. Wolfe

# Rotary Positional Embeddings

- Rotary Positional Embeddings (RoPE)

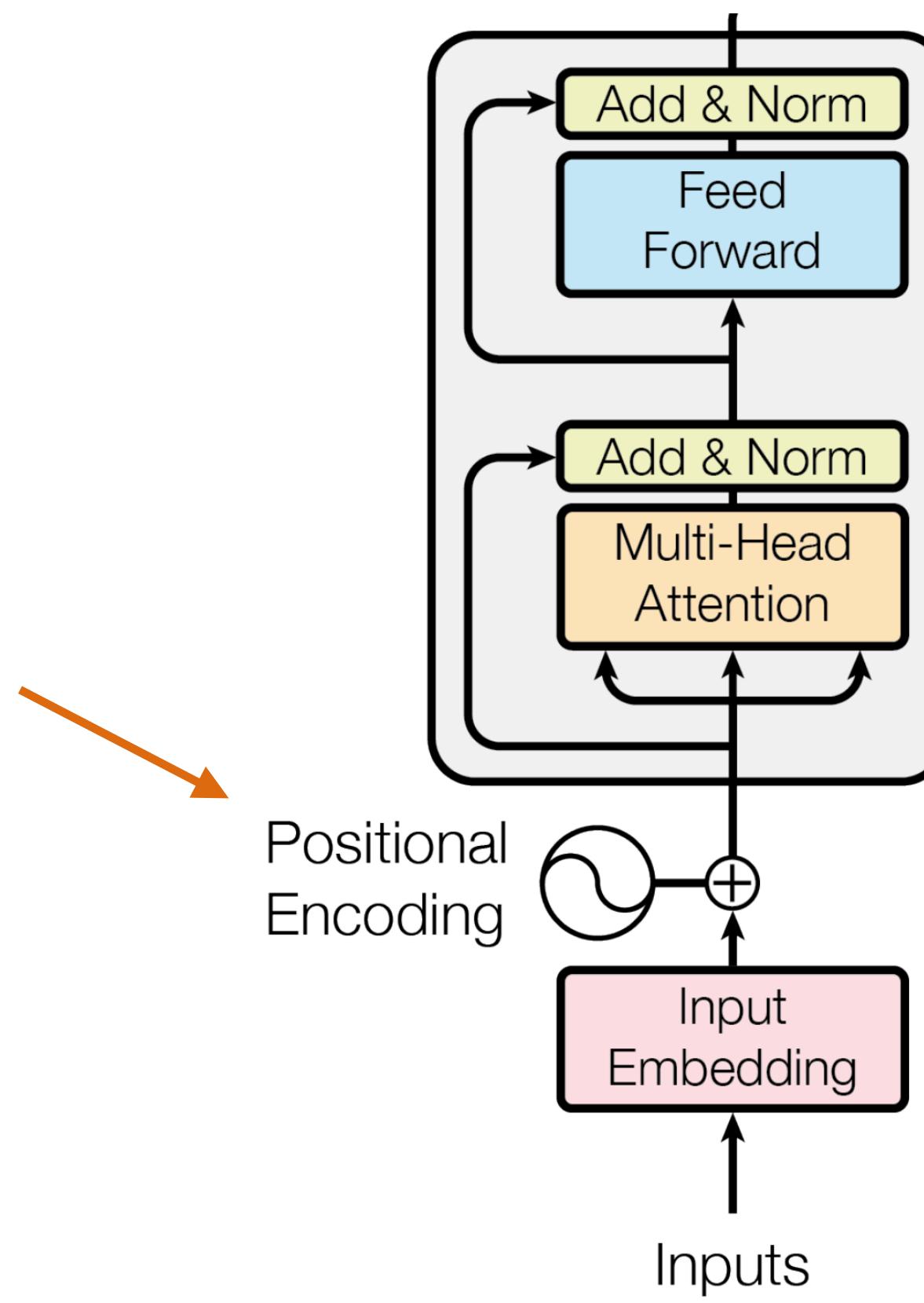
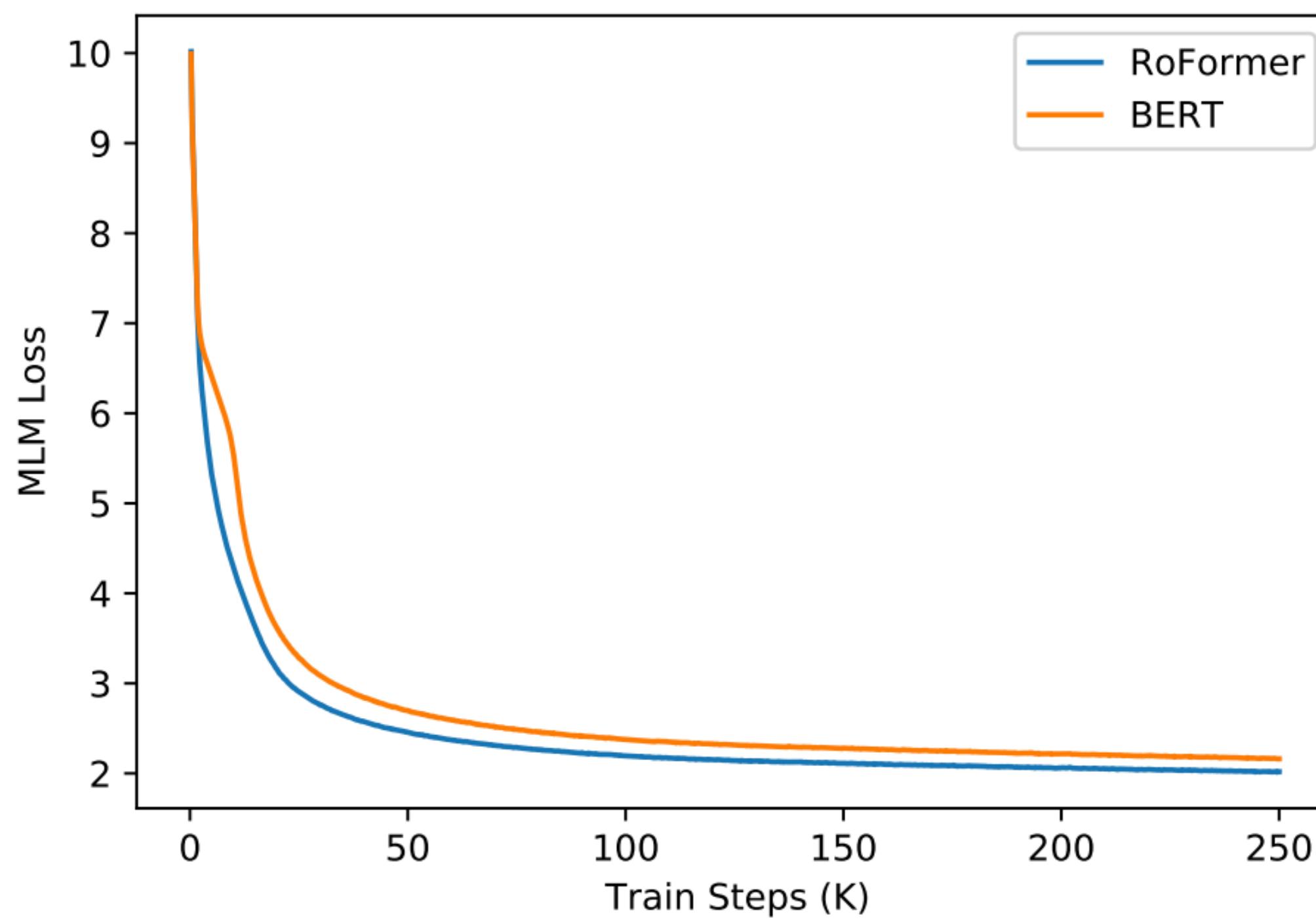


Figure 3: Evaluation of RoPE in language modeling pre-training. Left: training loss for BERT and RoFormer.

Su et al. (2021)

# Rotary Position Embeddings (RoPE)

---

---

## RoFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

---

**Jianlin Su**  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
[bojonesu@wezhuiyi.com](mailto:bojonesu@wezhuiyi.com)

**Yu Lu**  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
[julianlu@wezhuiyi.com](mailto:julianlu@wezhuiyi.com)

**Shengfeng Pan**  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
[nickpan@wezhuiyi.com](mailto:nickpan@wezhuiyi.com)

**Bo Wen**  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
[brucewen@wezhuiyi.com](mailto:brucewen@wezhuiyi.com)

**Yunfeng Liu**  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
[glenliu@wezhuiyi.com](mailto:glenliu@wezhuiyi.com)

April 21, 2021

### ABSTRACT

Position encoding in transformer architecture provides supervision for dependency modeling between elements at different positions in the sequence. We investigate various methods to encode positional information in transformer-based language models and propose a novel implementation named Rotary Position Embedding(RoPE). The proposed RoPE encodes absolute positional information with rotation matrix and naturally incorporates explicit relative position dependency in self-attention formulation. Notably, RoPE comes with valuable properties such as flexibility of being expand to any sequence lengths, decaying inter-token dependency with increasing relative distances, and capability of equipping the linear self-attention with relative position encoding. As a result, the enhanced transformer with rotary position embedding, or RoFormer, achieves superior performance in tasks with long texts. We release the theoretical analysis along with some preliminary experiment results on Chinese data. The undergoing experiment for English benchmark will soon be updated.

# Rotary Position Embeddings (RoPE)

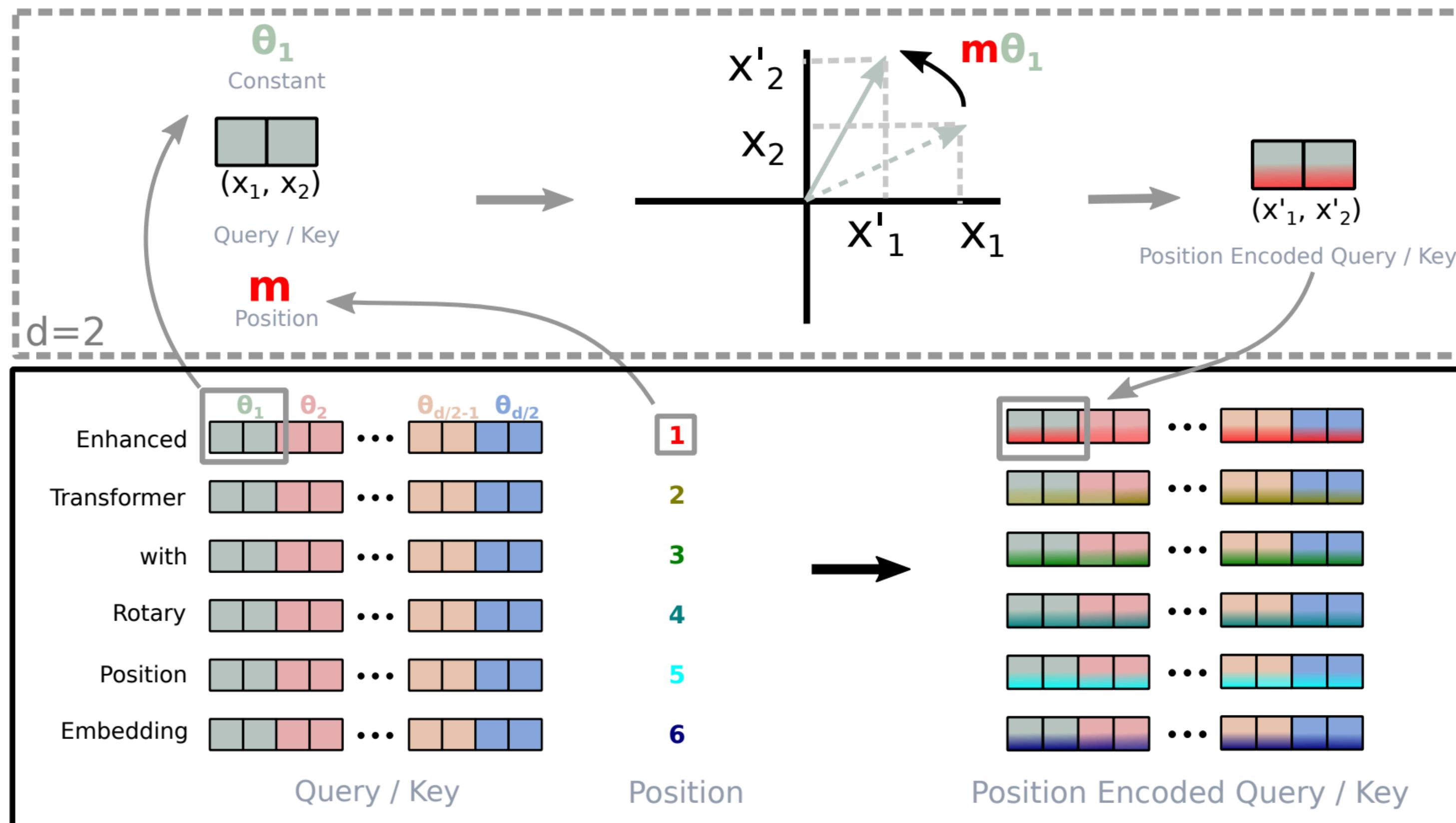


Figure 1: Implementation of Rotary Position Embedding(RoPE).

Su et al. (2021)

<https://www.youtube.com/watch?v=o29P0Kpobz0>

# Rotary Position Embeddings (RoPE)

- Rotation instead of addition!

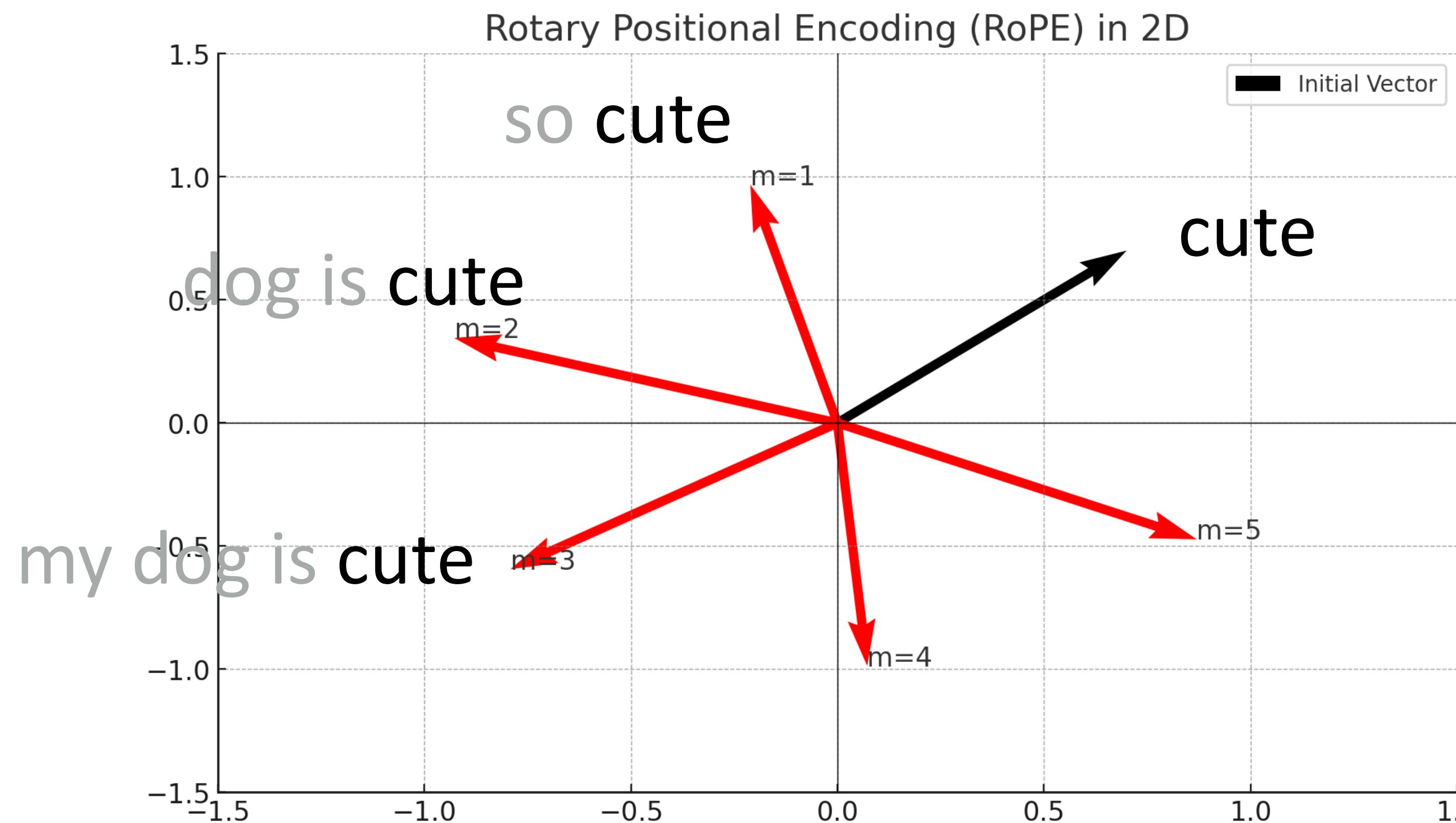
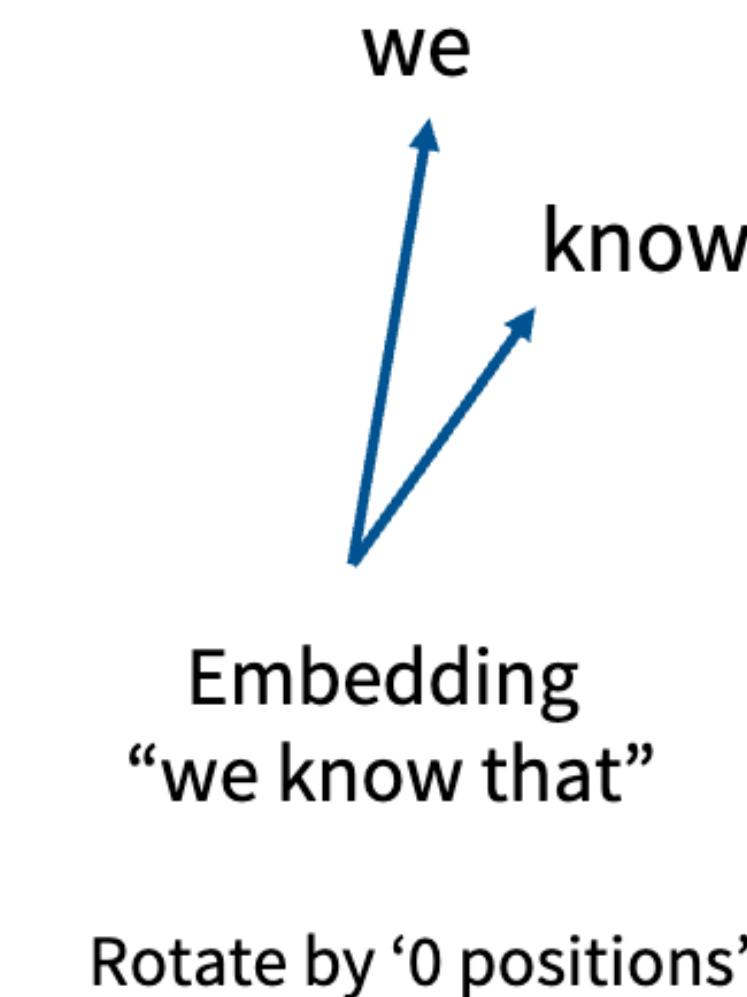
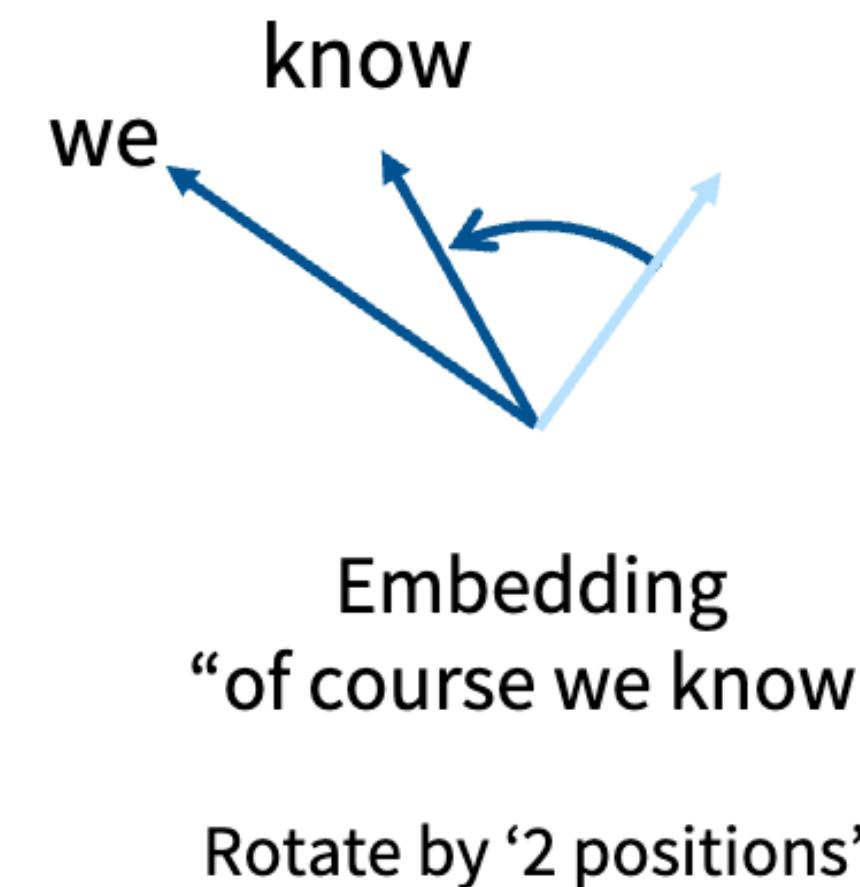
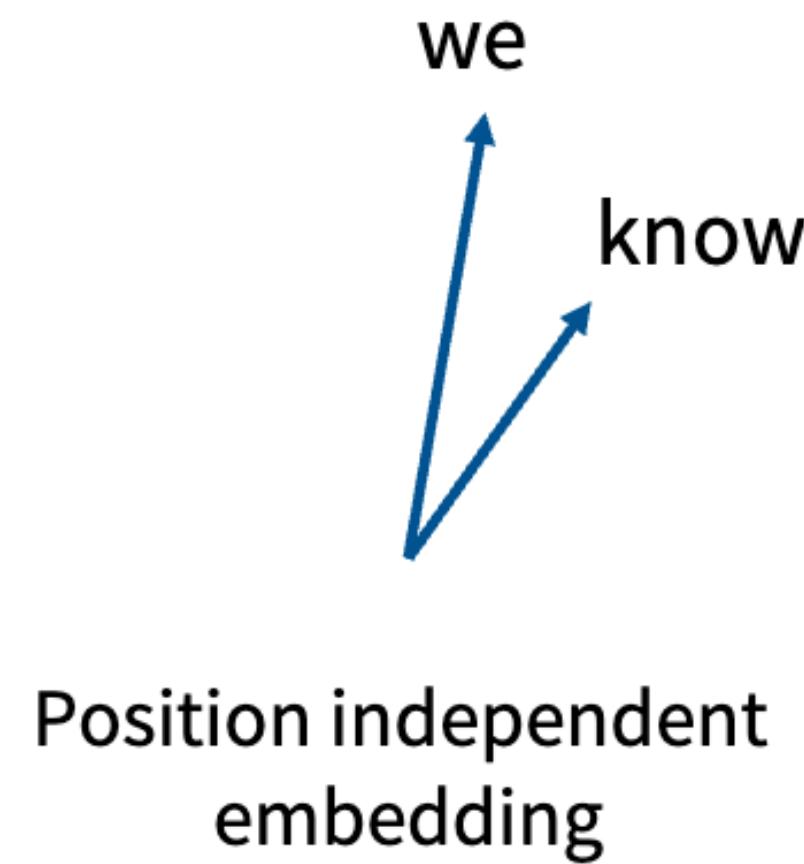


Image Credit: Sushant Kumar

# Rotary Position Embeddings (RoPE)

---

- ▶ Rotation instead of addition, such that
  - embeddings are invariant to absolute position
  - inner products are invariant to arbitrary rotation
- ▶ captures both absolute position and relative distance!



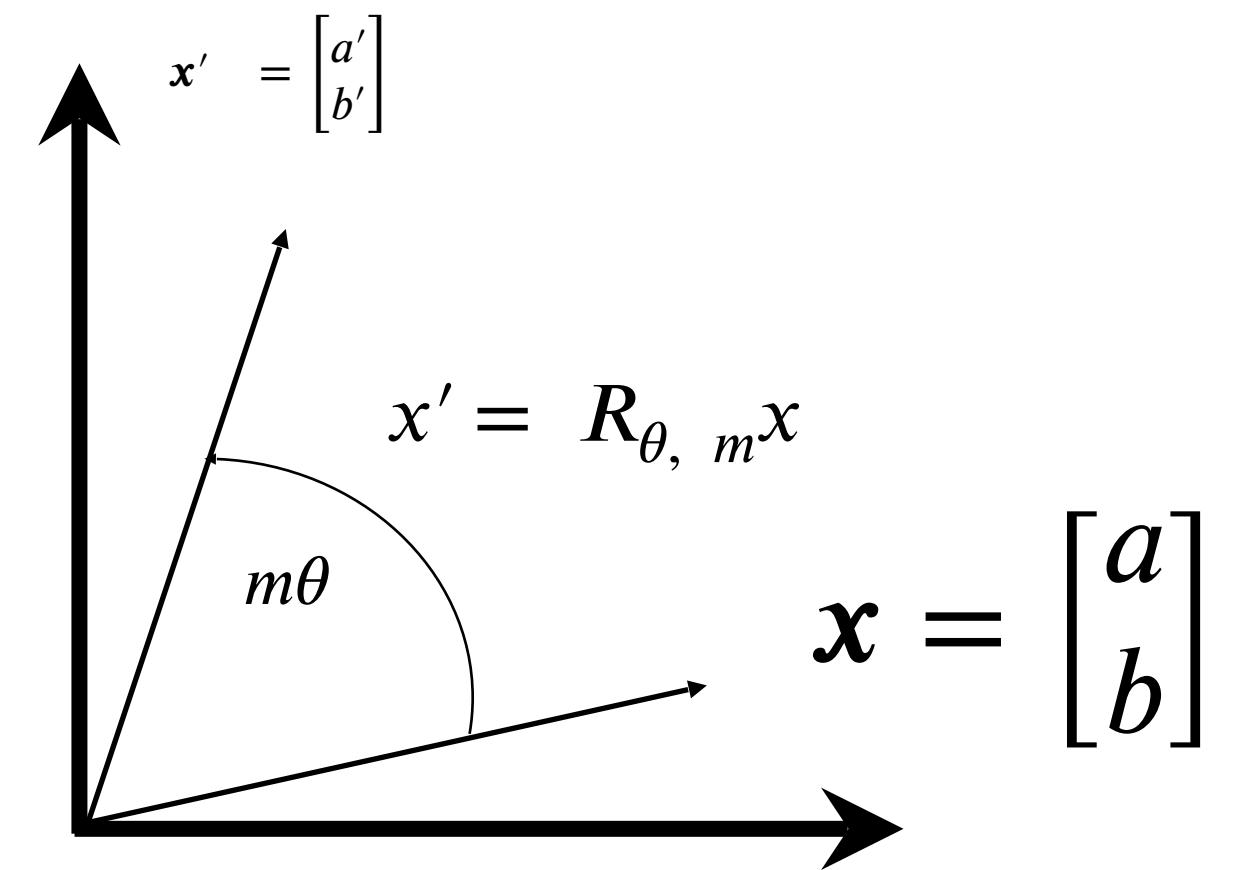
# Rotary Position Embeddings (RoPE)

---

- In 2D, a rotation matrix can be defined in the following form:

$$R_{\theta, m} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$

- The rotation increases with increasing  $\theta$  and  $m$ .
- Apply rotation after getting Q and K vectors (not V)



# Rotary Position Embeddings (RoPE)

---

- ▶ In practice, rotate  $d$  dimensional embedding matrices.
- ▶ Idea: rotate different dimensions with different angles  $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \dots, \theta_{d/2}\}$

$$\mathbf{R}_{\Theta,t}^d = \begin{pmatrix} \cos t\theta_1 & -\sin t\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin t\theta_1 & \cos t\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos t\theta_2 & -\sin t\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin t\theta_2 & \cos t\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos t\theta_{d/2} & -\sin t\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin t\theta_{d/2} & \cos t\theta_{d/2} \end{pmatrix}$$

# Rotary Position Embeddings (RoPE)

---

- A more computational efficient realization, taking advantage of the sparsity

$$\mathbf{R}_{\Theta,t}^d \mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{d-1} \\ u_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos t\theta_1 \\ \cos t\theta_2 \\ \cos t\theta_2 \\ \vdots \\ \cos t\theta_{d/2} \\ \cos t\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -u_2 \\ u_1 \\ -u_4 \\ u_3 \\ \vdots \\ -u_d \\ u_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin t\theta_1 \\ \sin t\theta_1 \\ \sin t\theta_2 \\ \sin t\theta_2 \\ \vdots \\ \sin t\theta_{d/2} \\ \sin t\theta_{d/2} \end{pmatrix}$$

# Rotary Position Embeddings (RoPE)

---

- Inner product decays as relative distance increases

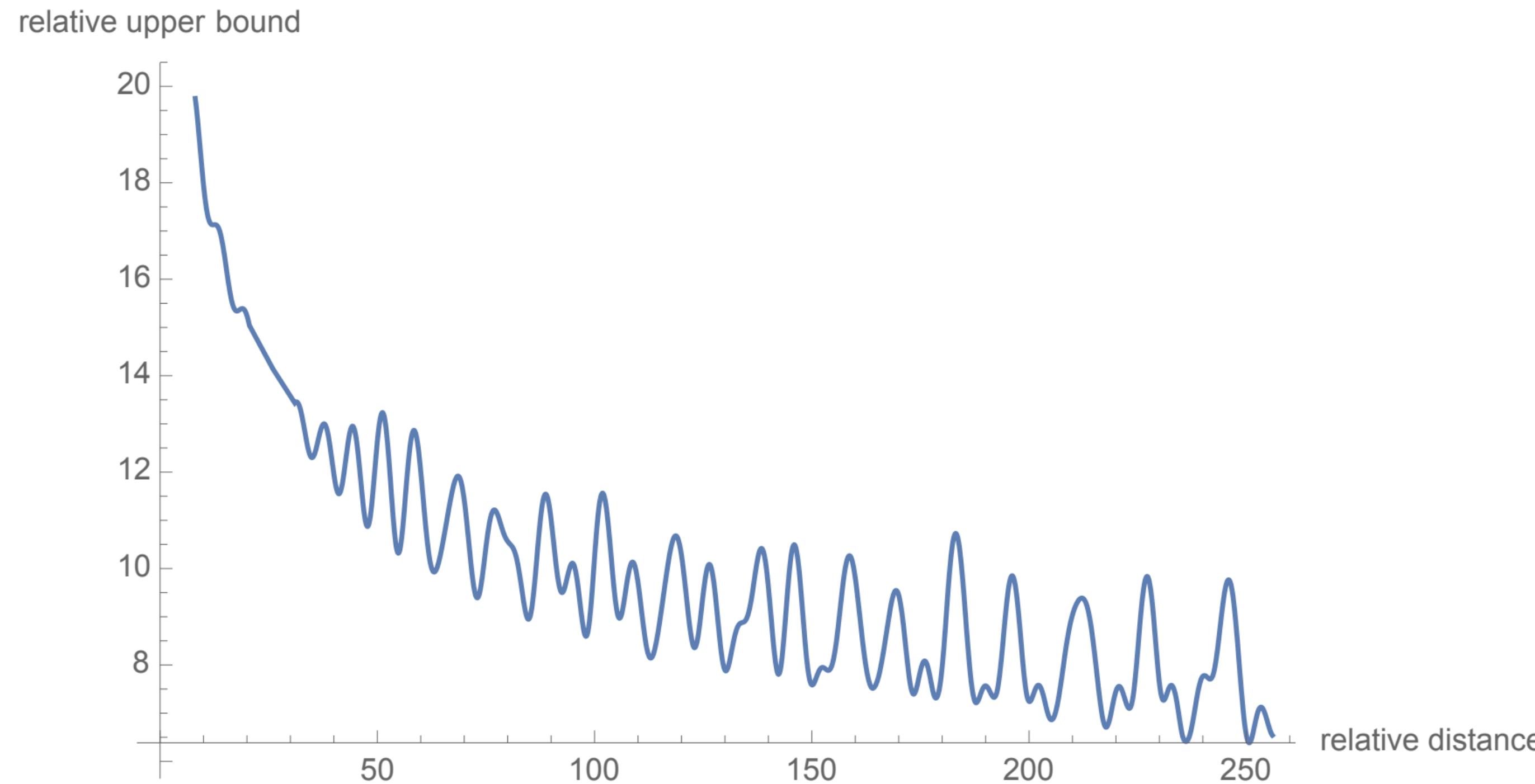


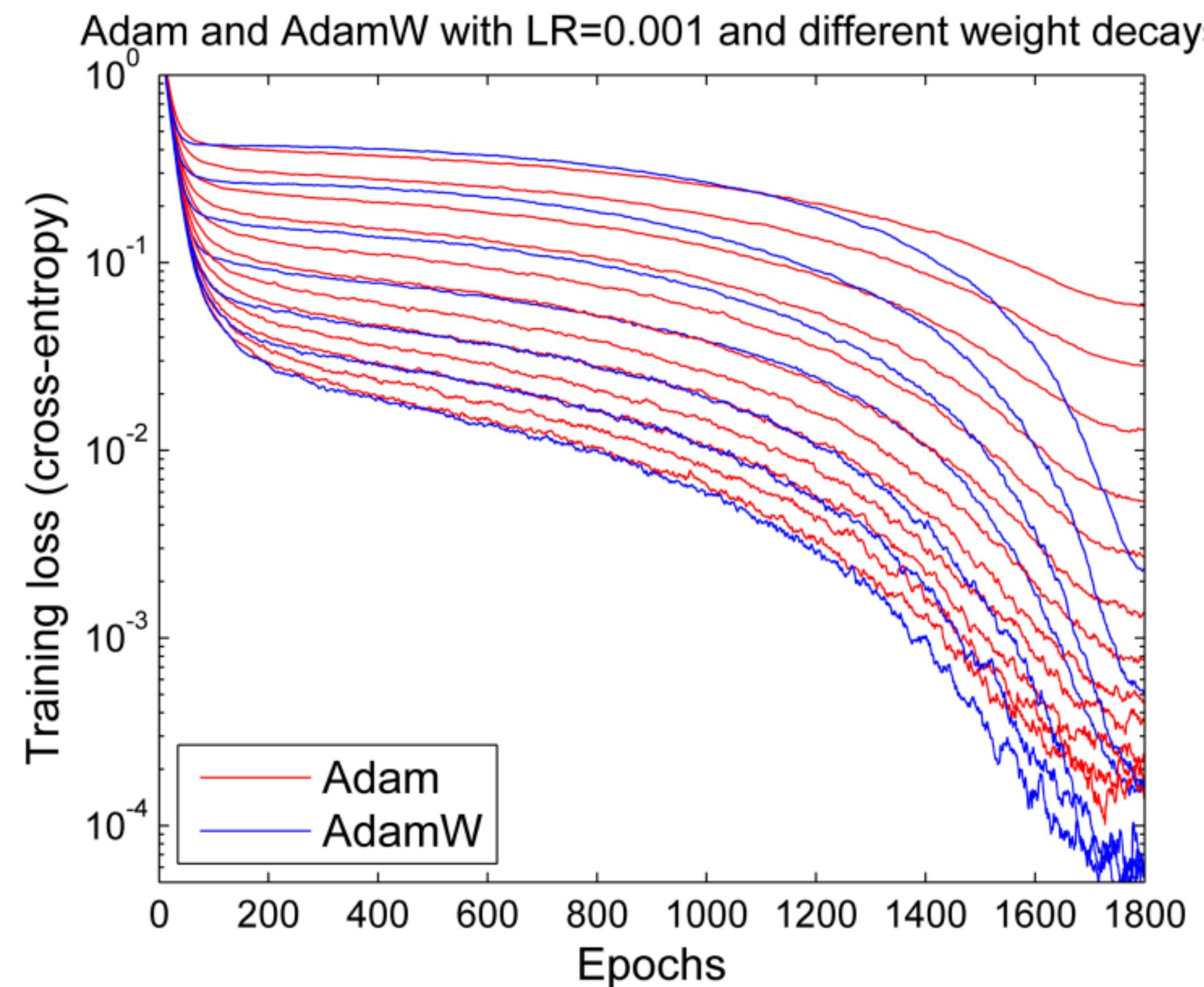
Figure 2: Long-term decay of RoPE.

# Optimization

# AdamW

---

- ▶ AdamW (Adam w/ weight decay) optimizer



Loshchilov and Hutter (2017)

<https://towardsdatascience.com/why-adamw-matters-736223f31b5d>

# AdamW Optimizer

## ► AdamW (Adam w/ weight decay)

$$x_t \leftarrow x_{t-1} - \alpha \frac{\beta_1 m_{t-1} + (1 - \beta_1)(\nabla f_t + w x_{t-1})}{\sqrt{v_t} + \epsilon}$$

weight is regularized less when v is large  
(insert line 6,7,8 into line 12; ignore 9,10)

**Algorithm 2** Adam with L<sub>2</sub> regularization and

---

```
1: given  $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $x_{t=0} \in \mathbb{R}^n$ , first moment vector  $m_{t=0} \leftarrow \mathbf{0}$ , second moment vector  $v_{t=0} \leftarrow \mathbf{0}$ , schedule multiplier  $\eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(x_{t-1}) \leftarrow \text{SelectBatch}(x_{t-1})$       ▷ select batch and
         return the corresponding gradient
6:    $g_t \leftarrow \nabla f_t(x_{t-1}) + w x_{t-1}$ 
7:    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$       ▷ here and below all
         operations are element-wise
8:    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
9:    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$       ▷  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$       ▷  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$       ▷ can be fixed, decay, or
         also be used for warm restarts
12:   $x_t \leftarrow x_{t-1} - \eta_t \left( \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $x_t$ 
```

---

# AdamW Optimizer

- ▶ AdamW (Adam w/ weight decay) optimizer

weight decay after (first and second moments of) gradient calculation for parameter-wise adaptive learning rate

**Algorithm 2** — ~~Adam with  $L_2$  regularization~~ and   
 Adam with weight decay (AdamW)

---

```
1: given  $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\mathbf{x}_{t=0} \in \mathbb{R}^n$ , first moment vector  $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$ , second moment vector  $\mathbf{v}_{t=0} \leftarrow \mathbf{0}$ , schedule multiplier  $\eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow \text{SelectBatch}(\mathbf{x}_{t-1})$       ▷ select batch and
         return the corresponding gradient
6:    $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1})$  +  $w\mathbf{x}_{t-1}$ 
7:    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$       ▷ here and below all
         operations are element-wise
8:    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ 
9:    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$       ▷  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$       ▷  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$  ▷ can be fixed, decay, or
         also be used for warm restarts
12:   $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \left( \alpha \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + w \mathbf{x}_{t-1} \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $\mathbf{x}_t$ 
```

---

Loshchilov and Hutter (2017)

<https://towardsdatascience.com/why-adamw-matters-736223f31b5d>

# Tokenization

# Tokenization

---

- ▶ The non-google world uses BPE. Google uses the SentencePiece library, which (sometimes) refers to a non-BPE subword tokenizer

Model	Tokenizer
Original transformer	BPE
GPT 1/2/3	BPE
T5 / mT5 / T5v1.1	SentencePiece (Unigram)
Gopher/Chinchilla	SentencePiece (??)
PaLM	SentencePiece (??)
LLaMA	BPE

# Tokenization

---

Monolingual models  
(30-50k vocab)

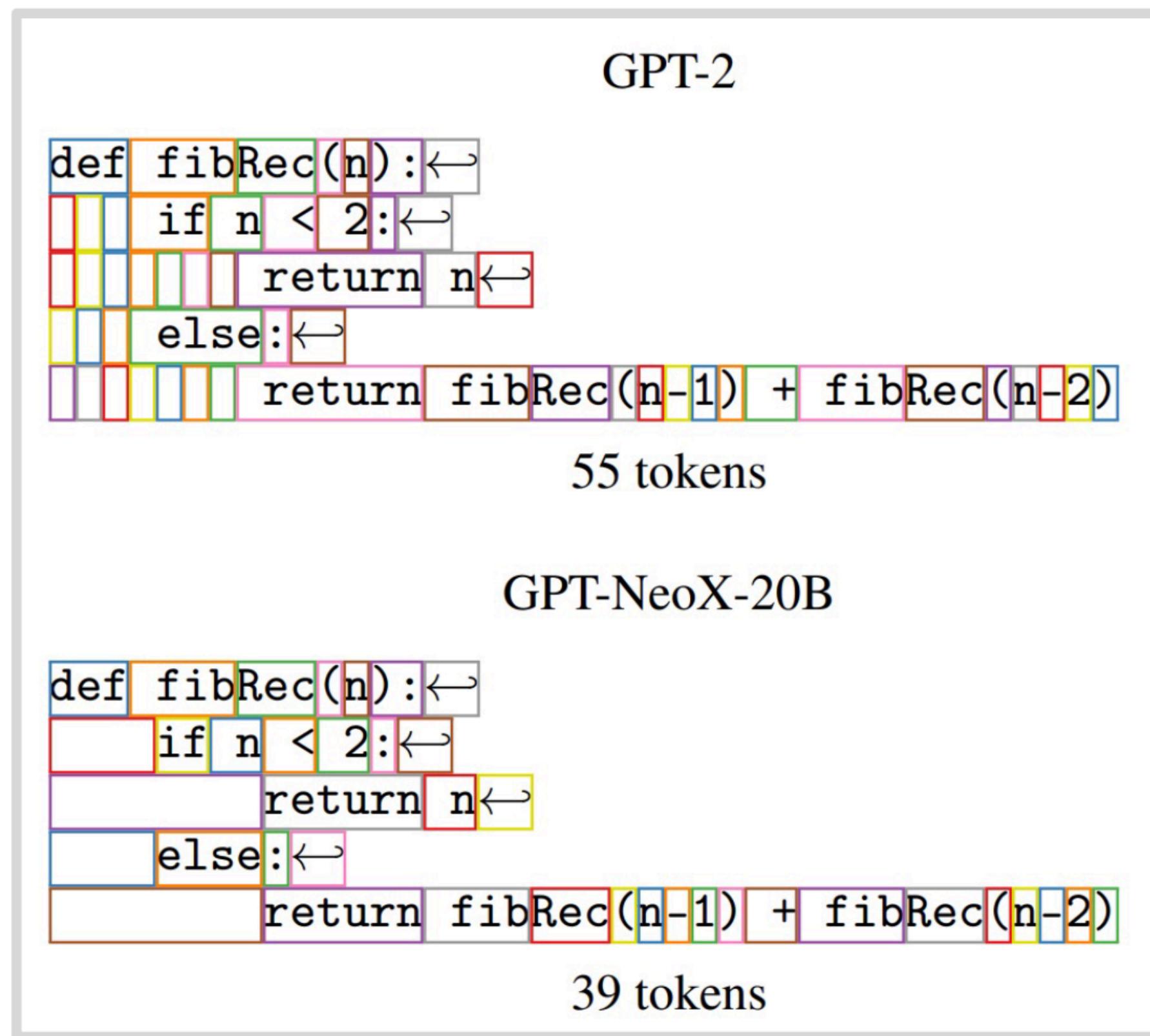
Model	Token count
Original transformer	37000
GPT	40257
GPT2/3	50257
T5/T5v1.1	32128
LLaMA	32000

Multilingual / Production Systems  
(100-250k vocab)

Model	Token count
mT5	250000
PaLM	256000
GPT4	100276
BLOOM	250680
DeepSeek	100000
Qwen 15B	152064
Yi	64000

# Tokenization

- ▶ Different treatments for white space, and digits ... mainly for math/code



Multi-whitespace tokenization

**Tokenizer.** We tokenize the data with the byte-pair encoding (BPE) algorithm (Sennrich et al., 2015), using the implementation from Sentence-Piece (Kudo and Richardson, 2018). Notably, we split all numbers into individual digits, and fallback to bytes to decompose unknown UTF-8 characters.

Individual digit tokenization (LLaMA/DeepSeek)

# What are being used?

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

Mostly follow previous successful choices.

# What are being used?

- ▶ There are many architectural variations.
- ▶ Major differences? Position embeddings, activations, tokenization
- ▶ This is an evolving field; a lot of empirical analysis is going into identifying best practices.

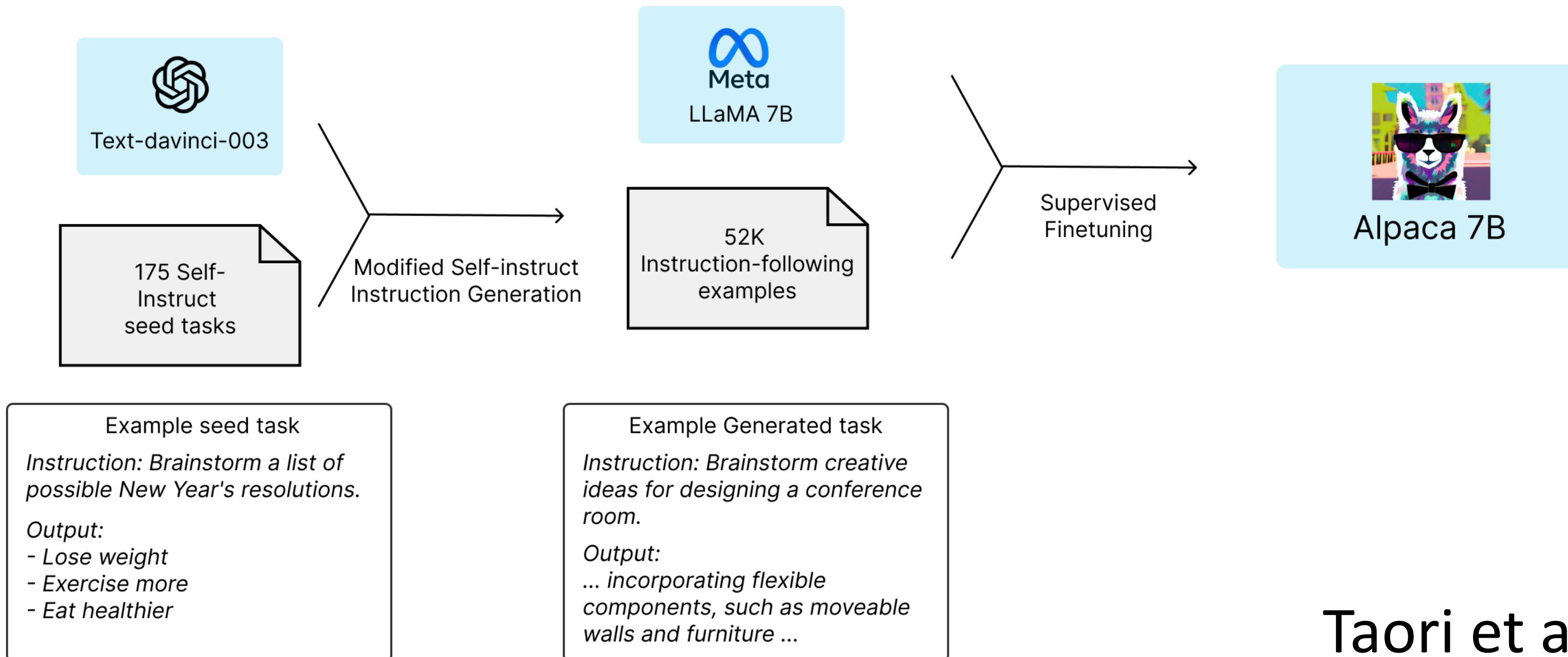
Aa Name	Has pa...	Link	# Year	Tokenizer type	# Vocab count	Norm	Parallel Layer	Pre-norm	Position embedding	Activations	MoE	# MLP factor	# num_layers	# model_dim
Original transformer	Yes	arxiv.org/abs...03762	2017	BPE	37000	LayerNorm	Serial	<input type="checkbox"/>	Sine	ReLU	<input type="checkbox"/>	4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

Image Credit: Tatsu Hashimoto

# Other Open-source Efforts

# Alpaca

- ▶ Released by Stanford on March 13, 2023
- ▶ Fine-tuned Meta's LLaMA-7B on 52k instruction-following demonstrated generated (Self-Instruct) using GPT-3.5 (text-davinci-003) for \$500.



Taori et al. (2022)

# Self-Instruct

- ▶ Address the labor-intense process for creating human-written instructions

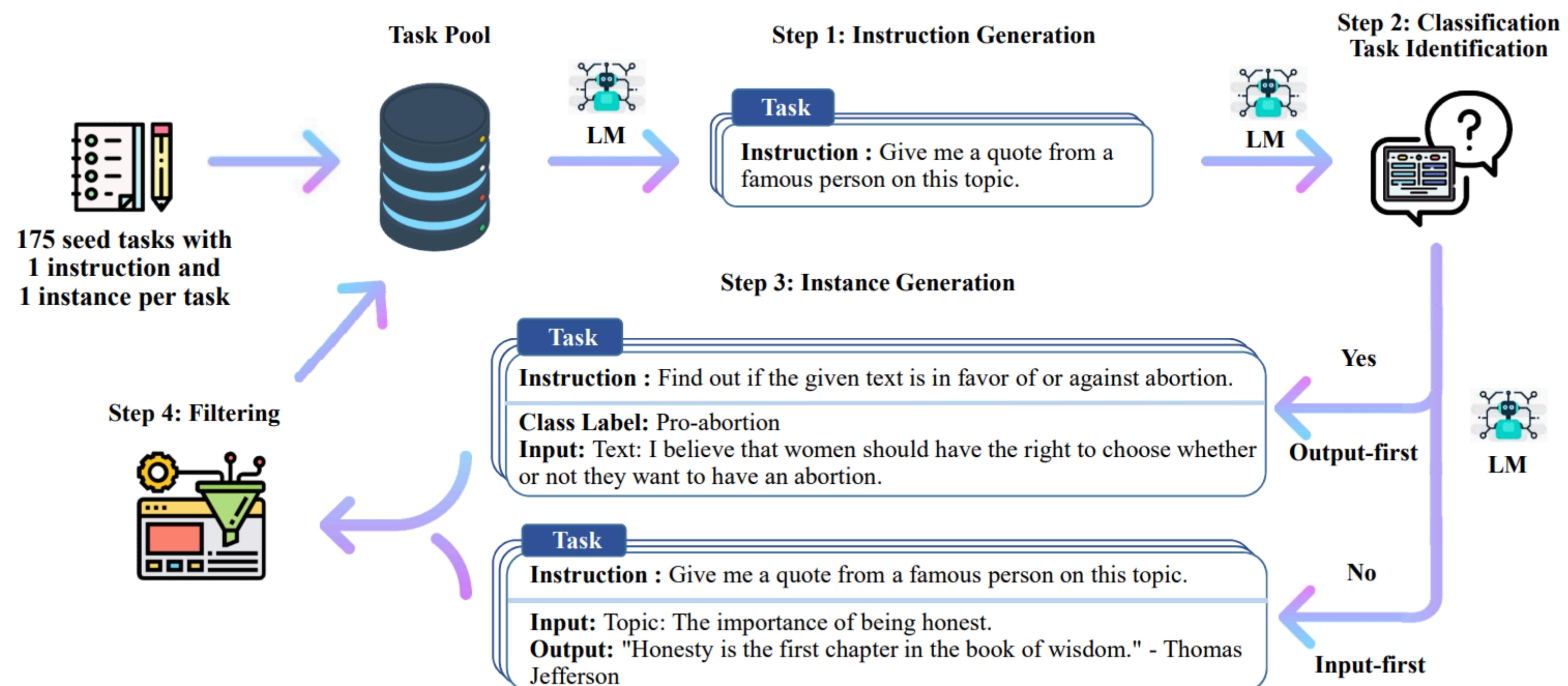


Figure 1: A high-level overview of SELF-INSTRUCT. The process starts with a small seed set of tasks (one instruction and one input-output instance for each task) as the task pool. Random tasks are sampled from the task pool, and used to prompt an off-the-shelf LM to generate both new instructions and corresponding instances, followed by filtering low-quality or similar generations, and then added back to the initial repository of tasks. The resulting data can be used for the instruction tuning of the language model itself later to follow instructions better. Tasks shown in the figure are generated by GPT3. See Table 10 for more creative examples.

# Self-Instruct

---

- ▶ Using multiple prompting templates to (a) generate the instruction, (b) classifying whether an instruction represents a classification task or not, (c) generating non-classification or classification instances

Come up with a series of tasks:

```
Task 1: {instruction for existing task 1}
Task 2: {instruction for existing task 2}
Task 3: {instruction for existing task 3}
Task 4: {instruction for existing task 4}
Task 5: {instruction for existing task 5}
Task 6: {instruction for existing task 6}
Task 7: {instruction for existing task 7}
Task 8: {instruction for existing task 8}
Task 9:
```

Table 6: Prompt used for generating new instructions. 8 existing instructions are randomly sampled from the task pool for in-context demonstration. The model is allowed to generate instructions for new tasks, until it stops its generation, reaches its length limit or generates “Task 16” tokens.

# Self-Instruct

---

Given the classification task definition and the class labels, generate an input that corresponds to each of the class labels. If the task doesn't require input, just generate the correct class label.

Task: Classify the sentiment of the sentence into positive, negative, or mixed.

Class label: mixed

Sentence: I enjoy the flavor of the restaurant but their service is too slow.

Class label: Positive

Sentence: I had a great day today. The weather was beautiful and I spent time with friends.

Class label: Negative

Sentence: I was really disappointed by the latest superhero movie. I would not recommend it.

...

Task: Tell me the first number of the given list.

Class label: 1

List: 1, 2, 3

Class label: 2

List: 2, 9, 10

Task: Which of the following is not an input type? (a) number (b) date (c) phone number (d) email address (e) all of these are valid inputs.

Class label: (e)

Task: {instruction for the target task}

Table 9: Prompt used for the output-first approach of instance generation. The model is prompted to generate the class label first, and then generate the corresponding input. This prompt is used for generating the instances for classification tasks.

# Self-Instruct

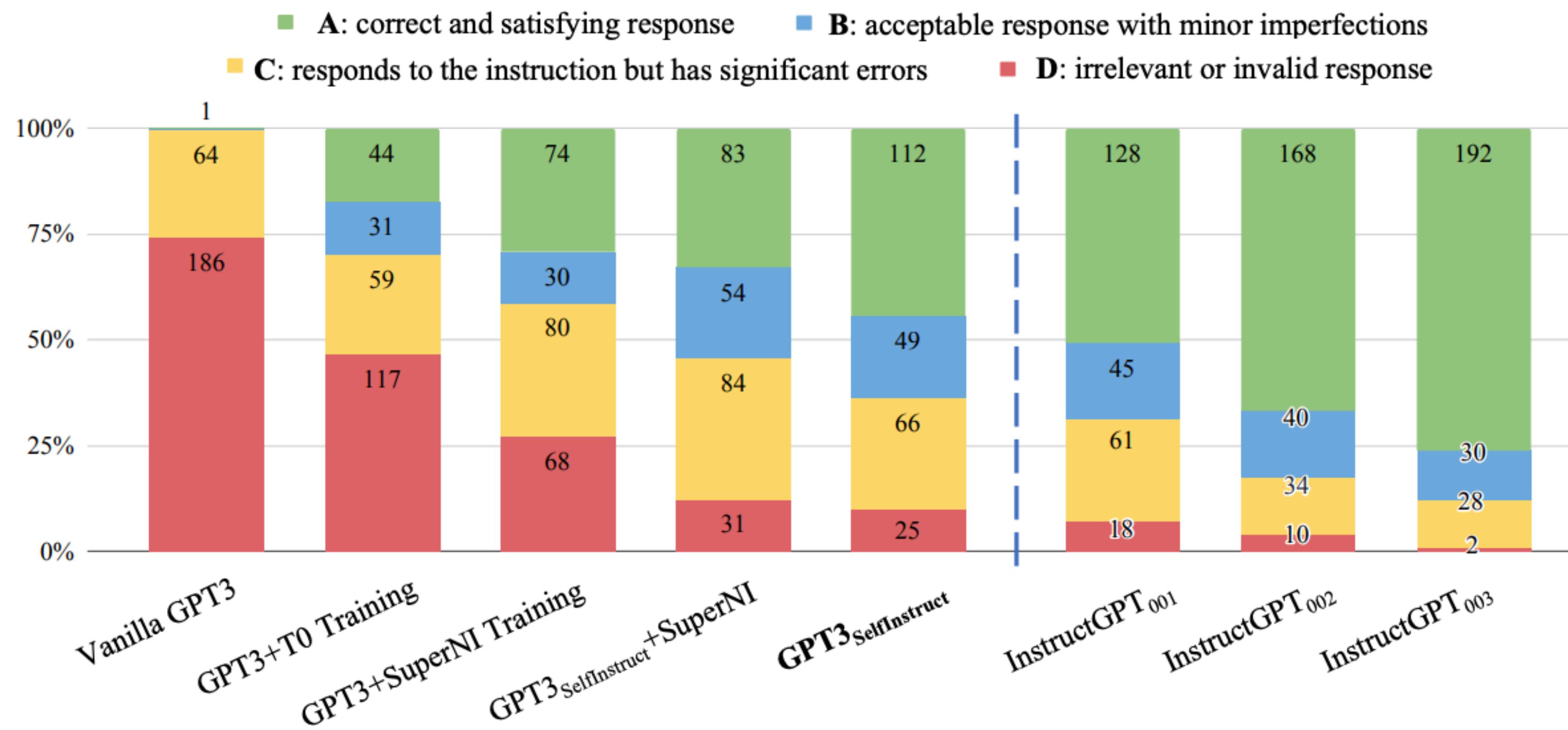


Figure 5: Performance of GPT3 model and its instruction-tuned variants, evaluated by human experts on our 252 user-oriented instructions (§5.4). Human evaluators are instructed to rate the models’ responses into four levels. The results indicate that GPT3<sub>SELF-INST</sub> outperforms all the other GPT3 variants trained on publicly available instruction datasets. Additionally, GPT3<sub>SELF-INST</sub> scores nearly as good as InstructGPT<sub>001</sub> (c.f., [footnote 1](#)).

# OLMo

---

- ▶ Released by AI2 on Feb 28, 2024
- ▶ Open-source not only the training code and model weights, but the full pre-training data (Dolma dataset) and intermediate checkpoints

<b>Size</b>	<b>Layers</b>	<b>Hidden Size</b>	<b>Attention Heads</b>	<b>Tokens Trained</b>
1B	16	2048	16	2T
7B	32	4086	32	2.46T
65B*	80	8192	64	

Table 1: OLMo model sizes and the maximum number of tokens trained to.

\* *At the time of writing our 65B model is still training.*

# Chatbot Arena: Elo Rankings

- ▶ Accepted as one of the premiere rankings for LLMs
- ▶ Style control was introduced as it was believed that the "style" of responses had a big effect

Rank*(UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organization	License
1	2	<a href="#">Grok-3-Preview-02-24</a>	1407	+7/-7	7580	xAI	Proprietary
1	1	<a href="#">GPT-4.5-Preview</a>	1404	+7/-9	6024	OpenAI	Proprietary
3	6	<a href="#">Gemini-2.0-Flash-Thinking-Exp-01-21</a>	1384	+5/-5	19837	Google	Proprietary
3	3	<a href="#">Gemini-2.0-Pro-Exp-02-05</a>	1380	+4/-4	17695	Google	Proprietary
3	2	<a href="#">ChatGPT-4o-latest_(2025-01-29)</a>	1375	+4/-5	19587	OpenAI	Proprietary
6	4	<a href="#">DeepSeek-R1</a>	1361	+5/-6	10474	DeepSeek	MIT
6	10	<a href="#">Gemini-2.0-Flash-001</a>	1355	+4/-5	15416	Google	Proprietary
6	3	<a href="#">o1-2024-12-17</a>	1353	+4/-4	22010	OpenAI	Proprietary
9	10	<a href="#">Gemma-3-27B-it</a>	1339	+9/-11	3870	Google	Gemma
9	10	<a href="#">Qwen2.5-Max</a>	1338	+5/-5	14258	Alibaba	Proprietary
9	7	<a href="#">o1-preview</a>	1335	+4/-4	33195	OpenAI	Proprietary
9	10	<a href="#">o3-mini-high</a>	1328	+6/-5	11409	OpenAI	Proprietary

leaderboard on Mar 12, 2025