

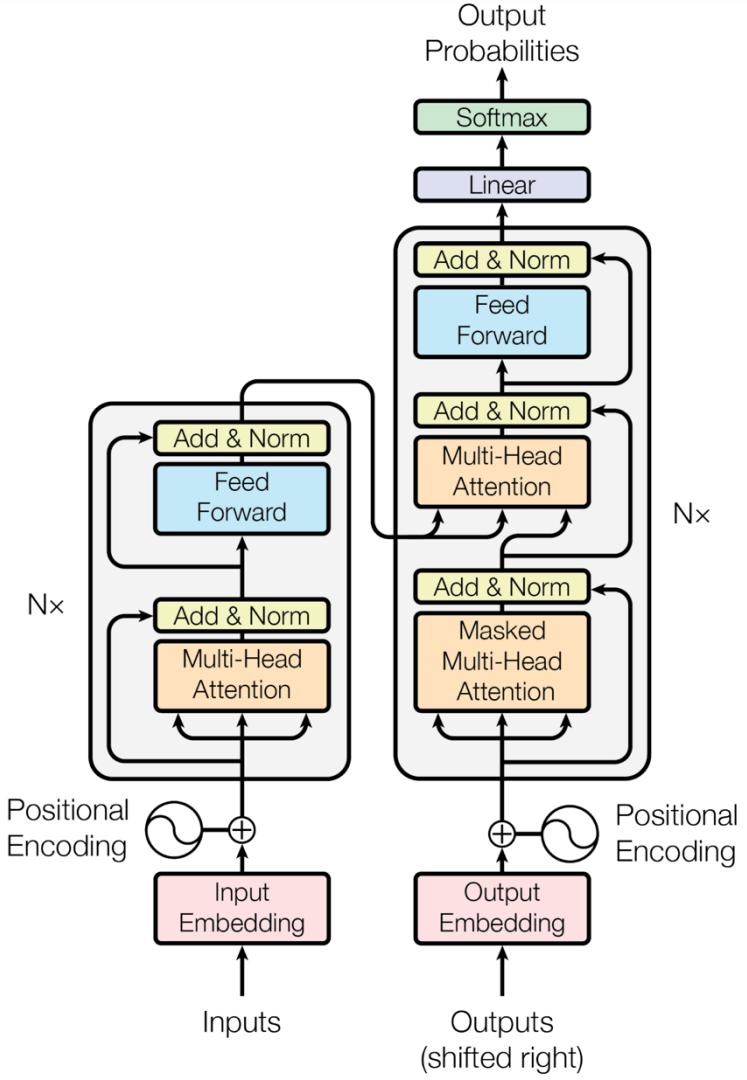


# Transformer Language Models

CSCI 601-471/671 (NLP: Self-Supervised Models)

<https://self-supervised.cs.jhu.edu/sp2025/>

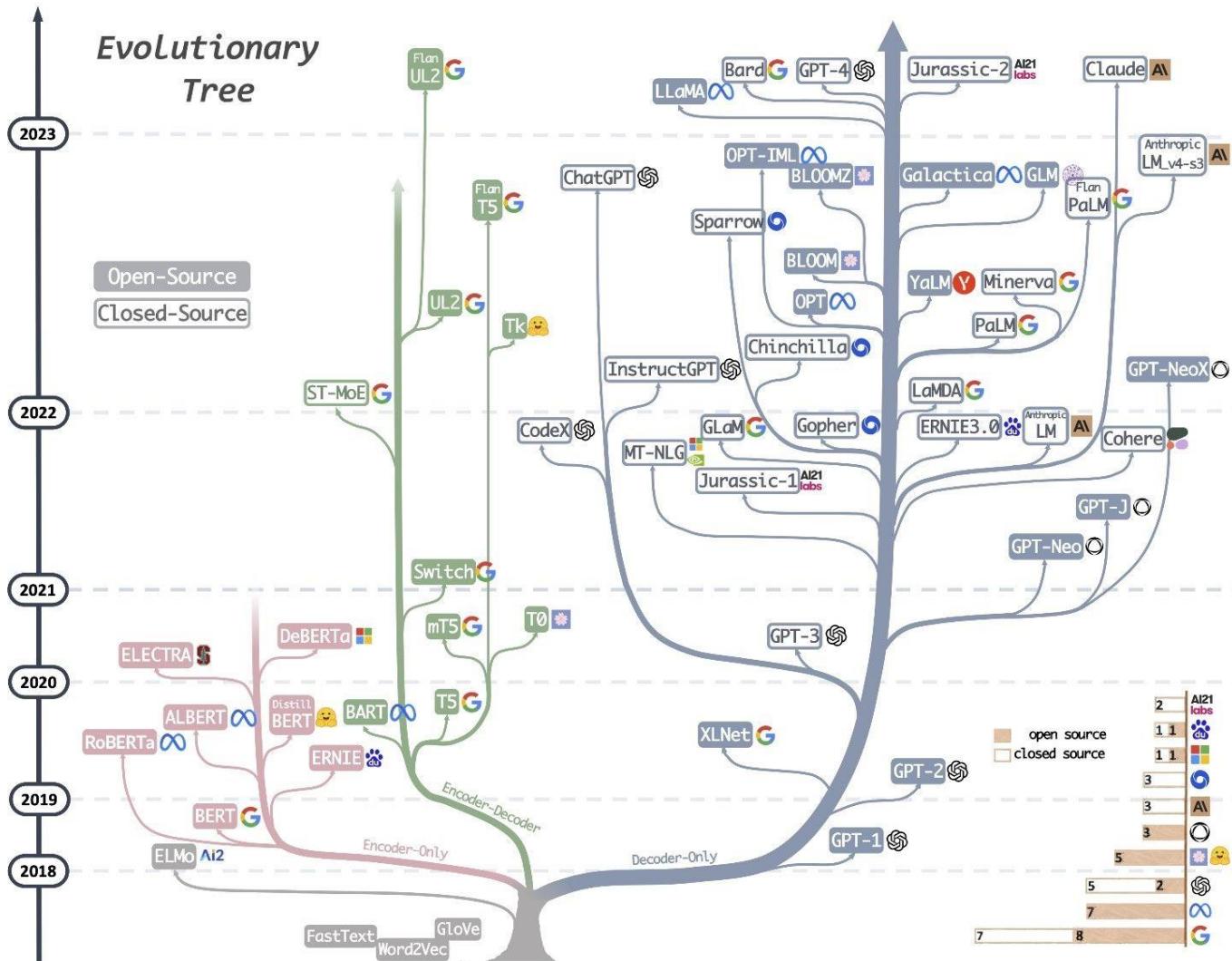
# Transformers: Recap



# After Transformer ...



# *Evolutionary Tree*



Yang et al. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond, 2023

# The Phases of Our Understanding

---

“Language modeling is a useful **subtask** for many NLP tasks”  
– everyone, pre-2018

“Language modeling is a useful **supertask** for many NLP tasks”  
– everyone, post-2018

# Chapter Plan

---

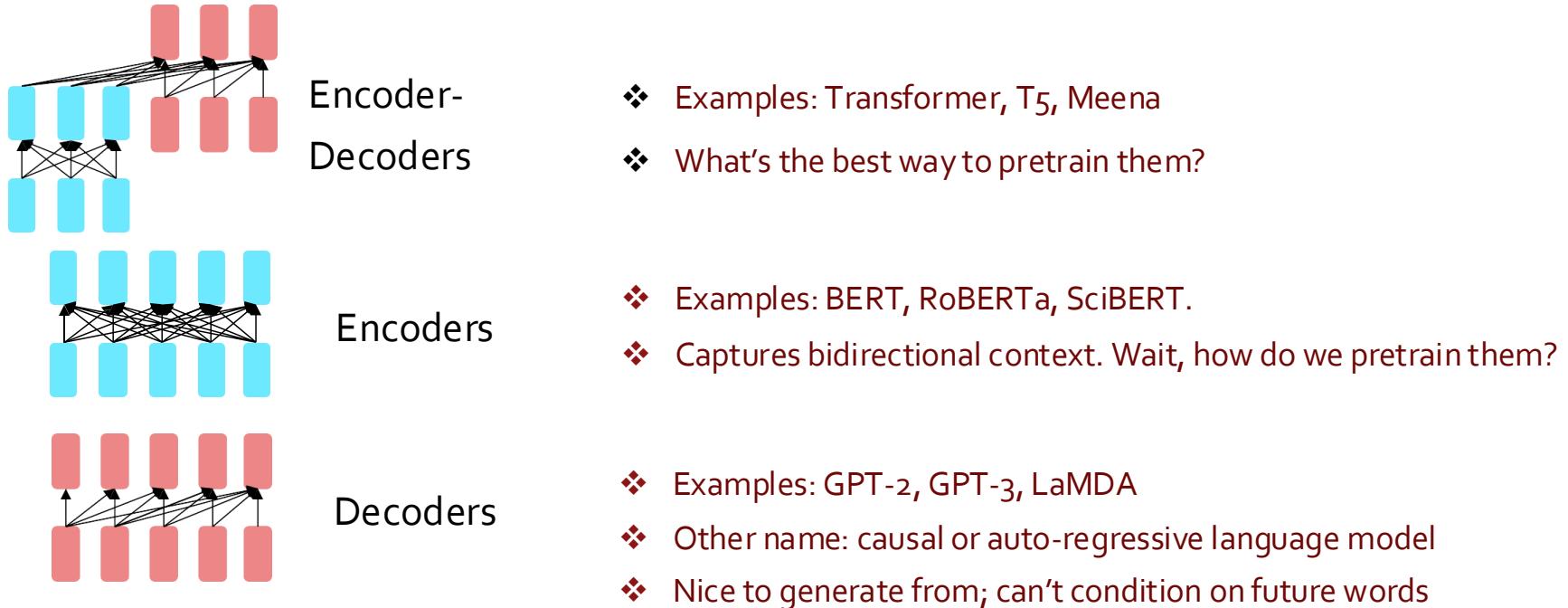
1. Transformer-based families of Language Models
2. Architectural variants
3. Thinking about pre-training data
4. Practical hacks and variants

**Chapter goal —** extending our understanding of training transformer language models.

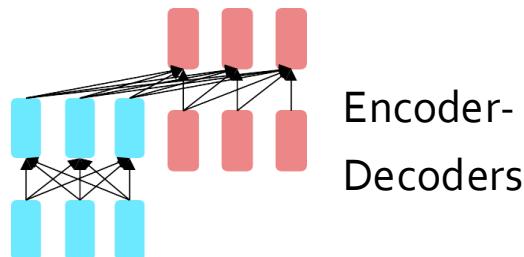
# Transformer Language Model Families

# Impact of Transformers

- A building block for a variety of LMs



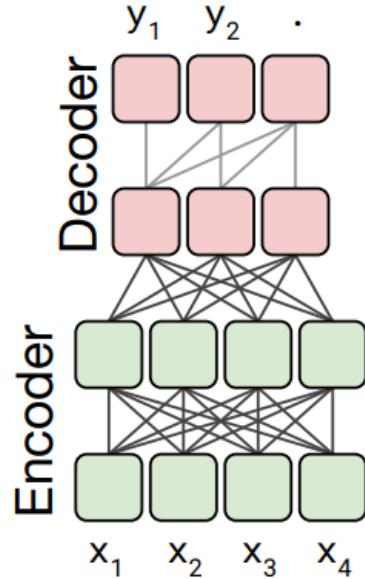
# Encoder-Decoder Family of Transformers



Encoder-  
Decoders

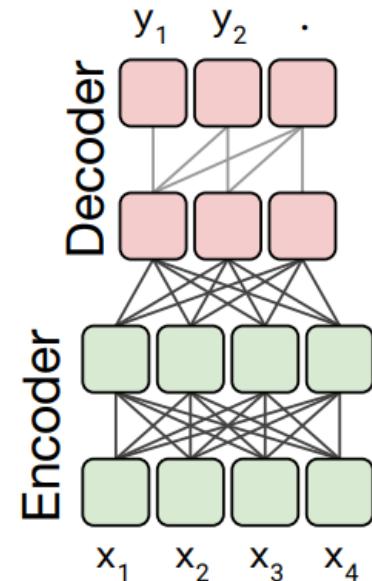
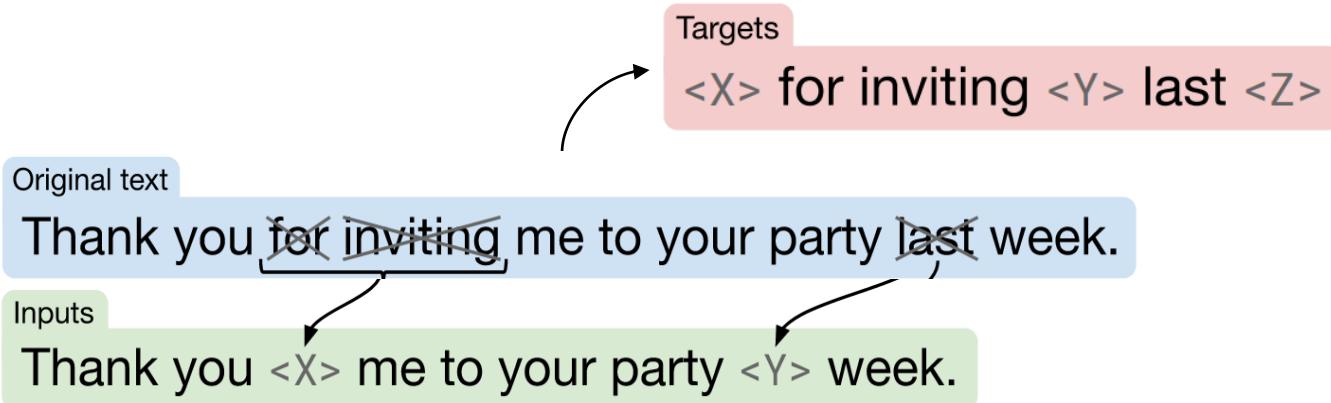
# Encoder-Decoder models: T5

- Architecture:
  - The **encoder** portion benefits from bidirectional context.
  - The **decoder** portion is used to train the whole model through language modeling.
  - Similar to the original Transformer enc-dec architecture.



# Encoder-Decoder models: T5

- Pretraining objective: Randomly corrupt tokens and replace with sentinel tokens ( $<X>$ ,  $<Y>$ ) that is unique over the example.



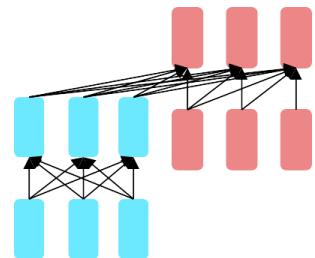
# Encoder-Decoder models: T5

Transformer Model	Parameters Count	Num Layers	Embedding Dimensions	Context/Sequence Length
<b>T5 Small</b>	~60M	6	512	512
<b>T5 Base</b>	~220M	12	768	512
<b>T5 Large</b>	~770M	24	1024	512
<b>T5-3B</b>	3B	24	1024	512
<b>T5-11B</b>	11B	24	1024	512

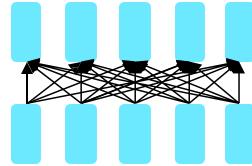
# Recap: Enc-dec models

---

- The most canonical form of Transformers.
- Notable example: T5.

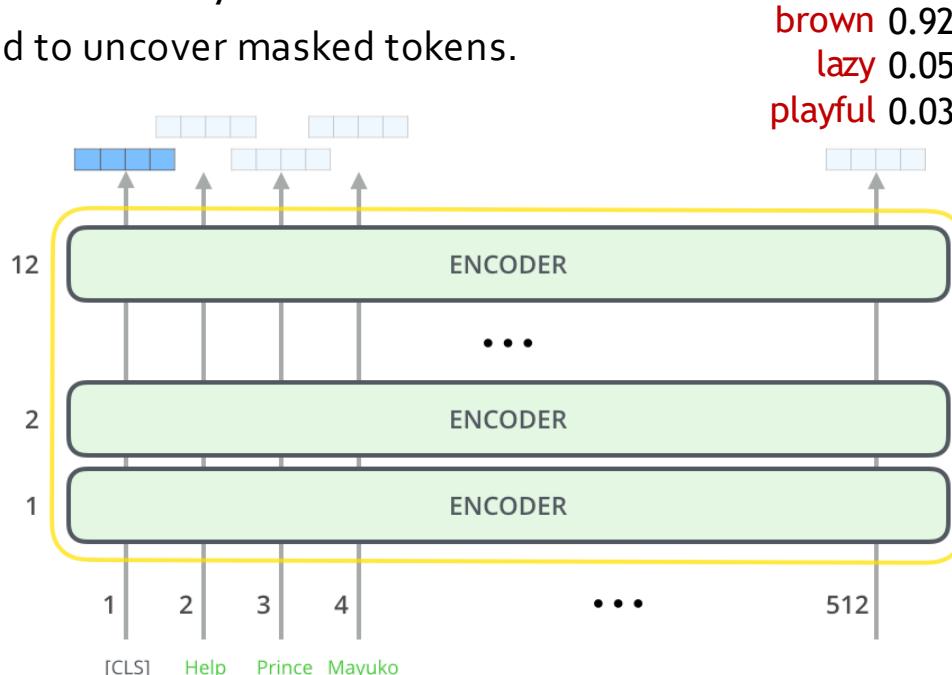


# Encoder-only Family of Transformers



# Encoder-only models (BERT)

- Transformer encoder-only
- BERT is trained to uncover masked tokens.



# Encoder-only models (BERT): Probing its predictions

- Masking words forces BERT to use context in both directions to predict the masked word.

Paris is the [MASK] of France.

Compute

Computation time on cpu: cached

capital	0.997
heart	0.001
center	0.000
centre	0.000
city	0.000

</> JSON Output

Maximize

# Encoder-only models (BERT): Probing its predictions

- Masking words forces BERT to use context in both directions to predict the masked word.

Today is Tuesday, so tomorrow is [MASK].

Compute

Computation time on cpu: cached



</> JSON Output

Maximize

# Encoder-only models (BERT): Pre-training Objectives

- **Token masking:** Randomly mask 15% of tokens and train the model to recover them.

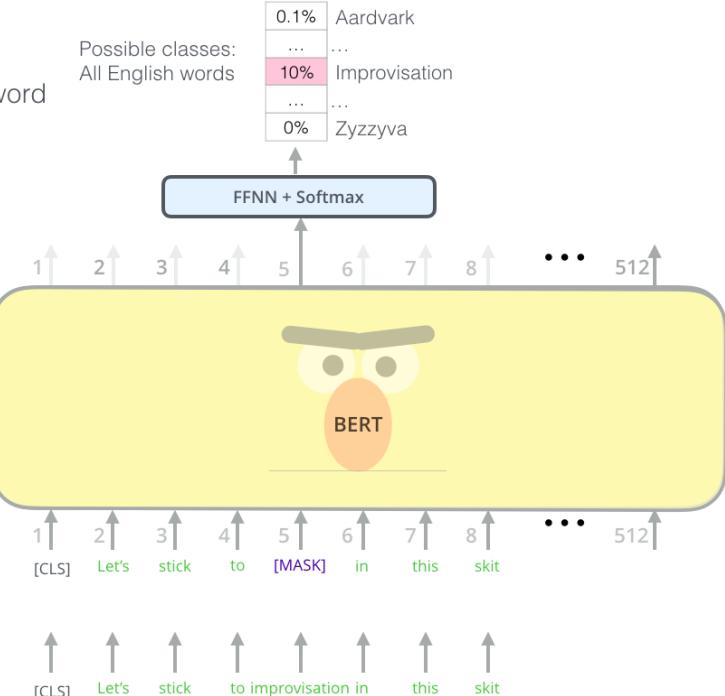
Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zzyzyva

Randomly mask  
15% of tokens

Input



# Encoder-only models (BERT): Pre-training Objectives

- **Token masking:** Randomly mask 15% of tokens and train the model to recover them.
  - Too little masking: Too **expensive** to train
  - Too much masking: **Underdefined**
    - (not enough info for the model to recover the masked tokens)
- **Sentence ordering:** Predict sentence ordering
  - Learns the relationships between sentences
  - 50% correct ordering, and 50% random incorrect ones

**Sentence A** = The man went to the store.

**Sentence B** = He bought a gallon of milk.

**Label** = IsNextSentence

**Sentence A** = The man went to the store.

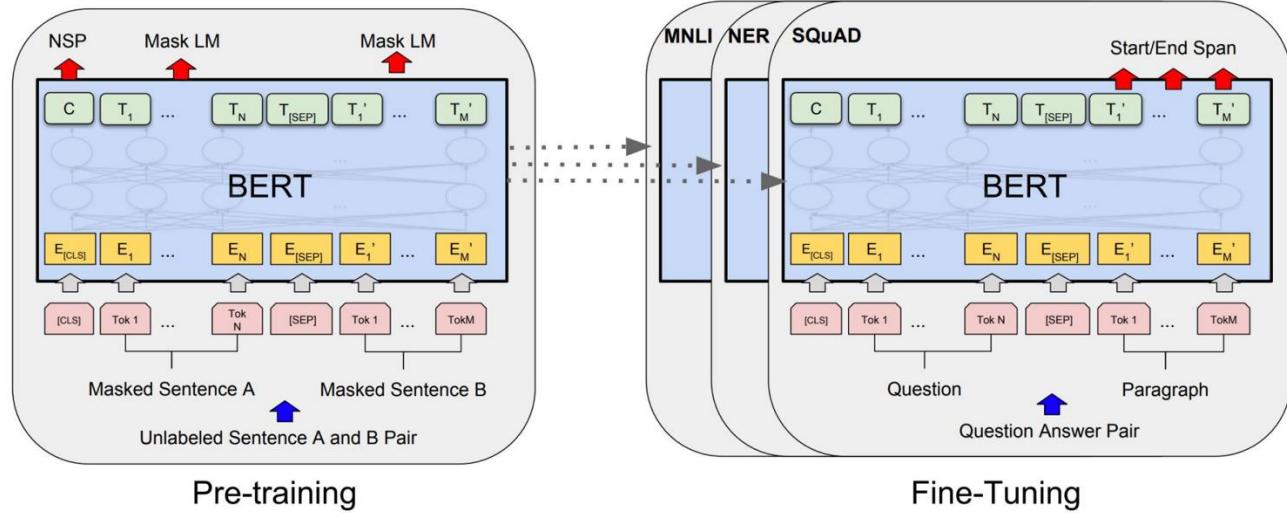
**Sentence B** = Penguins are flightless.

**Label** = NotNextSentence

# Encoder-only models (BERT): Fine-tune for tasks

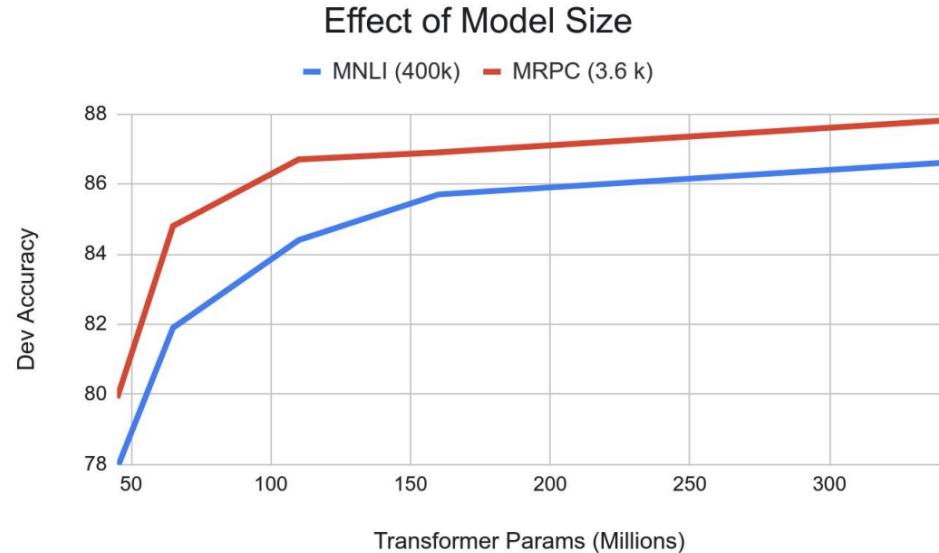
"Pretrain once, finetune many times."

- **Idea:** Make pre-trained model **usable** in **downstream tasks** (often classification)
- **Initialized** with pre-trained model parameters
- **Fine-tune** model parameters using labeled data from downstream tasks



# Encoder-only models (BERT): One of the Early Signs on the Effectiveness of Scale

- Going from 110M -> 340M params helps a lot
- Improvements have **not** plateaued!



# Encoder-only models (ModernBERT): Recent Reincarnation of BERT

- Essentially a BERT-like architecture but a few key changes:
  - Longer context:** Trained for context window of 8,192 tokens (vs. 512 in BERT)
  - MLP layer:** Drop the bias term to save costs.
  - More norms:** Add an extra normalization layer after embeddings.
  - Replaced activations:** Replaced GeLU activation with GeGLU (will talk about this)
  - Pos encoding:** Replaced the sine/cosine with rotary embeddings (will talk about this)

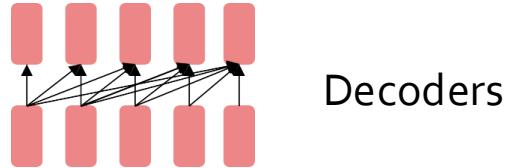
Model	IR (DPR)			IR (CoLBERT)		NLU	Code		
	BEIR	MLDR <sub>OOD</sub>	MLDR <sub>ID</sub>	BEIR	MLDR <sub>OOD</sub>	GLUE	CSN	SQA	
Large	BERT	38.9	23.3	31.7	49.5	28.5	85.2	41.6	60.8
	RoBERTa	41.4	22.6	36.1	49.8	28.8	88.9	47.3	68.1
	DeBERTaV3	25.6	7.1	19.2	46.7	23.0	<b>91.4</b>	21.2	19.7
	GTE-en-MLM	42.5	<b>36.4</b>	<b>48.9</b>	50.7	71.3	87.6	40.5	66.9
	ModernBERT	<b>44.0</b>	34.3	48.6	<b>52.4</b>	<b>80.4</b>	90.4	<b>59.5</b>	<b>83.9</b>

# Recap: Encoder-only models

---

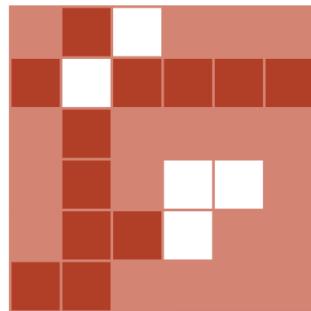
- Transformer-based decoder-only models trained on massive piles of data.
- Common use-cases:
  - Provide incredible framework **contextualized** embeddings of words.
  - It also allows **fine-tuning** on your particular task (usually top layers).
- However, they were **not** designed to generate text – unless you do additional work.

# Decoder-only Family of Transformers

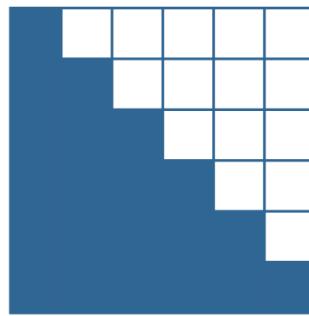


# Decoder-only (GPT)

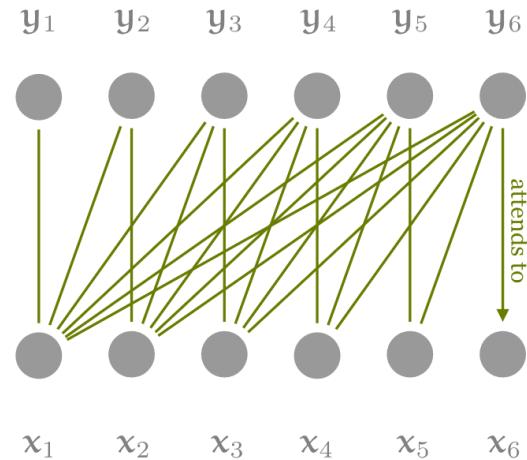
- Generate sequences where each token is predicted based on the previously generated tokens
- Use causal masking to ensure the causality
- Trained to maximize log-likelihood defined for next-token prediction.



raw attention weights



mask



# GPT4

- Transformer-based
  - The rest is .... mystery! 😊
  - Rumor: GPT-4 is a Mixture of Experts model (we'll talk about it).
  - If we're going based on costs, GPT4 is ~15-30 times costlier than GPT3. That should give you an idea how its likely size!
- Note, these language models involve **more than just pre-training.**
  - Pre-training provides the foundation based on which we build the model.
  - We will discuss the later stages (i.e., alignment) in a 2-3 weeks.

Model	Usage	
davinci-002	\$0.0020 / 1K tokens	
Model	Input	Output
gpt-4	\$0.03 / 1K tokens	\$0.06 / 1K tokens

# Other Available [Decoder] LMs

---

EleutherAI: GPT-Neo (6.7B), GPT-J (6B), GPT-NeoX (20B)

<https://huggingface.co/EleutherAI>

<https://6b.eleuther.ai/>

LLaMA, 65B: <https://github.com/facebookresearch/llama>

Mistral and Mixtral:

<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>

<https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1>

# Summary: Existing models

---

- There is a ton of models out there.
- We talked about a few: BERT, T5, GPT family.
- You should always check the existing leaderboards (e.g., ChatBotArena) to see what's the best and latest.
- Next, we're going to spend a quite a bit of time delving into design decisions for training LLMs.

# LMSys ChatArena

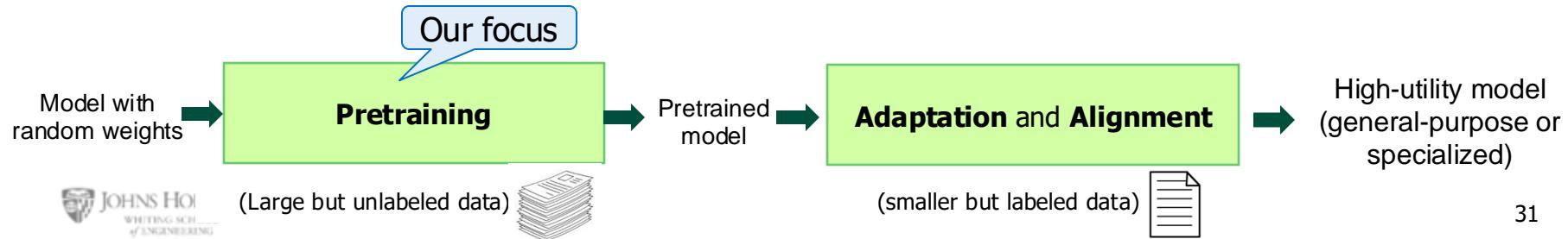
<https://lmarena.ai/>

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organization	License
1	1	<a href="#">chocolate_(Early_Grok-3)</a>	1403	+6/-6	9992	xAI	Proprietary
2	3	<a href="#">Gemini-2.0-Flash-Thinking-Exp-01-21</a>	1385	+4/-6	15083	Google	Proprietary
2	3	<a href="#">Gemini-2.0-Pro-Exp-02-05</a>	1380	+5/-6	13000	Google	Proprietary
2	1	<a href="#">ChatGPT-4o-latest_(2025-01-29)</a>	1377	+5/-5	13470	OpenAI	Proprietary
5	3	<a href="#">DeepSeek-R1</a>	1362	+7/-7	6581	DeepSeek	MIT
5	8	<a href="#">Gemini-2.0-Flash-001</a>	1358	+7/-7	10862	Google	Proprietary
5	3	<a href="#">o1-2024-12-17</a>	1352	+5/-5	17248	OpenAI	Proprietary
8	7	<a href="#">o1-preview</a>	1335	+3/-4	33169	OpenAI	Proprietary
8	8	<a href="#">Qwen2.5-Max</a>	1334	+5/-5	9282	Alibaba	Proprietary
8	7	<a href="#">o3-mini-high</a>	1332	+5/-9	5954	OpenAI	Proprietary
11	11	<a href="#">DeepSeek-V3</a>	1318	+4/-5	19461	DeepSeek	DeepSeek
11	13	<a href="#">Qwen-Plus-0125</a>	1311	+9/-7	5112	Alibaba	Proprietary
11	14	<a href="#">GLM-4-Plus-0111</a>	1310	+6/-9	5134	Zhipu	Proprietary
11	13	<a href="#">Gemini-2.0-Flash-Lite-Preview-02-05</a>	1309	+6/-5	10262	Google	Proprietary
12	12	<a href="#">o3-mini</a>	1306	+5/-6	12179	OpenAI	Proprietary
12	17	<a href="#">Step-2-16K-EXP</a>	1304	+7/-7	5130	StepFun	Proprietary
12	17	<a href="#">o1-mini</a>	1304	+4/-3	54944	OpenAI	Proprietary
12	13	<a href="#">Gemini-1.5-Pro-002</a>	1302	+3/-3	54970	Google	Proprietary

# Pre-training language models: Architectures

# Training Pipeline for LLMs

- There is extensive literature about best practices for pretraining
  - What choice of architectures are good?
  - How do you prepare pre-training data?
  - What considerations go into efficient training of the models?
  - ...
- Our goal in this chapter is to summarize the latest best and common practices.



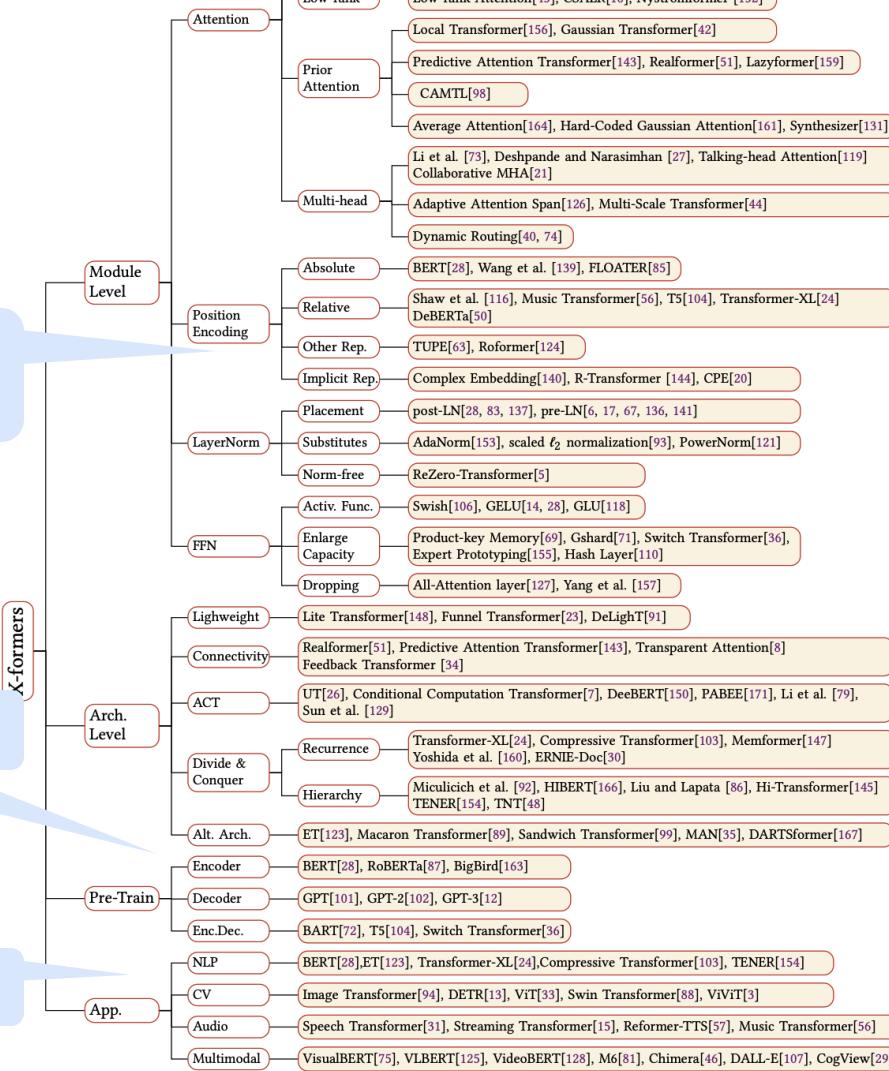
We will visit a few of these branches ...

But there is a lot that we do **not** cover ...

Variants of positional embeddings

Architectural choices

Multi-modal models



How consistent are the architectures  
used in existing LLMs?

# Another View of Architectural Variations

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

Low consensus  
(except pre-norm)

Most try to follow  
previous successful  
choices.

[Slide credit: Tatsu Hashimoto]



# When should we do normalization?

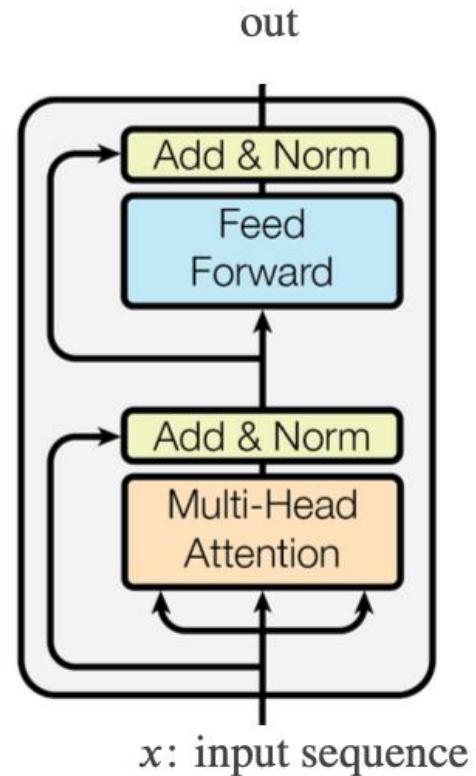


# Quiz: Pre-norm vs Post-norm

- Which is the original implementation?
- Which one is your favorite?

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

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



# Pre-norm vs Post-norm

- Pre-norm (right) is set up so that LayerNorm does not disrupt the residual stream (in gray).
- In theory, both should work fine.
- In practice, however, **Pre-norm is preferred over Post-norm.**

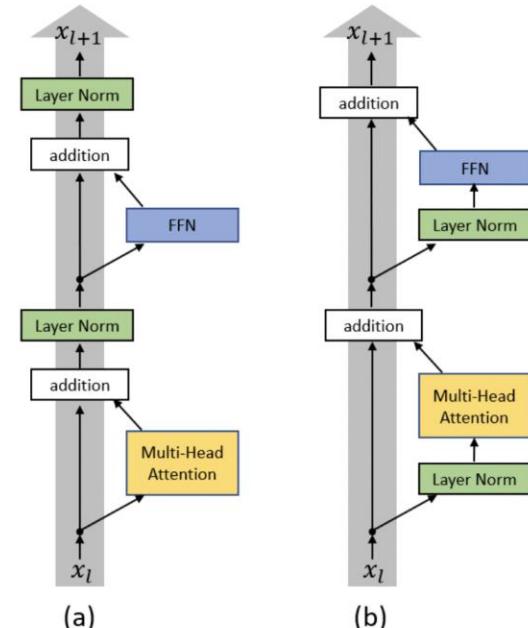
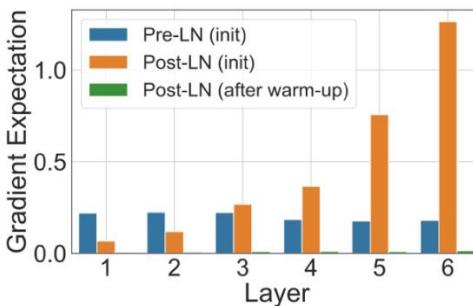


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

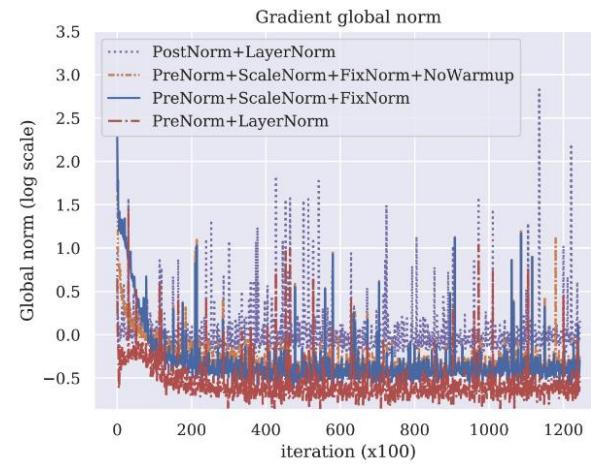
# Pre-norm vs Post-norm — Explanation?

- Stability, larger LRs for large networks and no need for warm up.

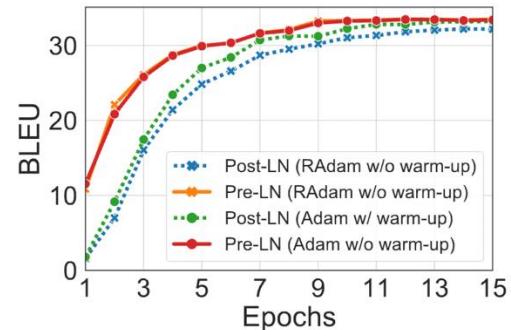
Gradient attenuation



Gradient spikes



No need for warm-up stage



(b) BLEU (IWSLT)

# Layer Norm vs RMSNorm

- Original transformer: **LayerNorm**
  - Normalizes the mean and variance across  $d_{model}$

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

**Notable models:**

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

- Many modern LMs: **RMSNorm**
  - Does not subtract mean or add a bias term

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

**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)
- Does this explanation make sense?
  - Matrix multiplies are the vast majority of FLOPs (and memory)
  - Non-matmul ops only make up 0.2% of our FLOPS
    - So perhaps it doesn't matter that GPUs compute non-matmul ops slower.

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

"Tensor Contraction" := matmuls

[Slide credit: Tatsu Hashimoto]

# Why RMSNorm?

- **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	$2.182 \pm 0.005$	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>

[Slide credit: Tatsu Hashimoto]



Is the “bias” term  
in FFNs necessary?



$$\text{FFN}(\mathbf{x}) = f(\mathbf{x} \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2$$

# The Bias Terms

- Most modern transformers don't have bias terms.
  - Original Transformer:

$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

and  $f$  was defined as ReLU:  $f(x) = \max(0, x)$

$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{\text{ff}}},$$
$$\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$$

- Most implementations (if they're not gated):

$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2$$

- Reasons: memory (similar to RMSnorm) and optimization stability.

[Slide credit: Tatsu Hashimoto]

# 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, which saves on wallclock time
- Bias term:
  - People more generally drop bias terms since the compute/param tradeoffs are not great.

[Slide credit: Tatsu Hashimoto]



# What activations $f(\cdot)$ should we use?



$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

# Activations

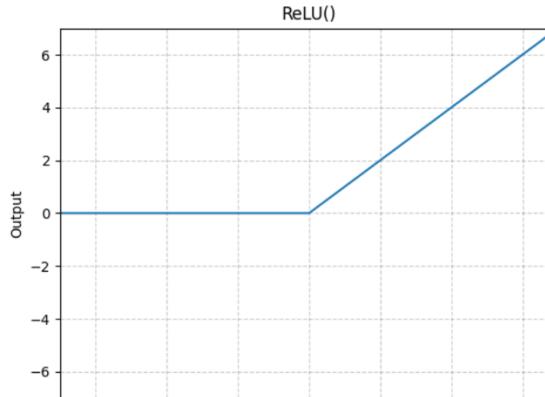
---

- No much consensus:  
ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU, ...

# Activations: ReLU vs GeLU

- **ReLU:**

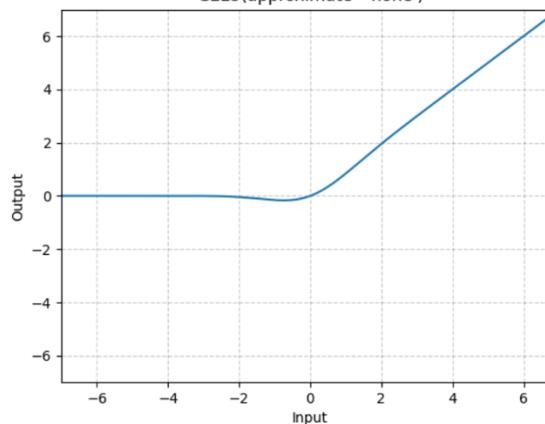
$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1)\mathbf{W}_2$$



**Notable models:**  
Original transformer, T5,  
Gopher, Chinchilla, OPT

- **GeLU:**

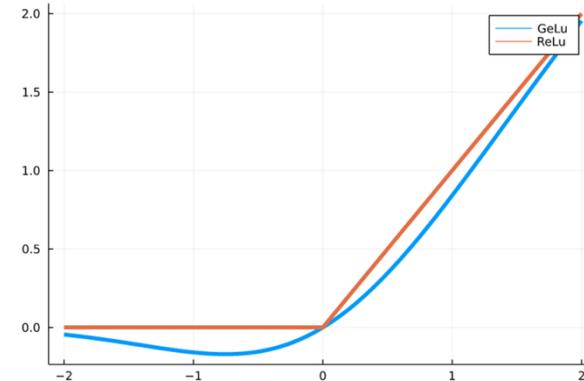
$$\text{FFN}(\mathbf{x}) = \text{GELU}(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2$$



**Notable models:**  
GPT1/2/3, GPTJ, GPT-  
Neox, BLOOM

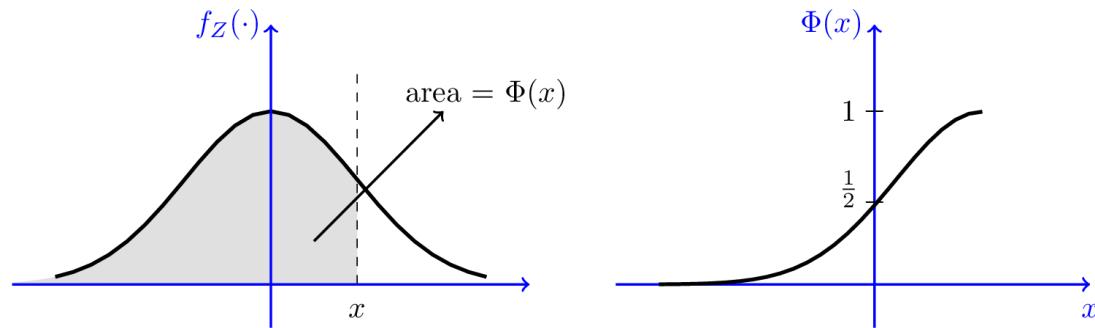
# GELU, in details

$$\text{FFN}(\mathbf{x}) = \text{GELU}(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2$$
$$\text{GELU}(\mathbf{y}) := \mathbf{y}\Phi(\mathbf{y})$$



- Here  $\Phi(\mathbf{y})$  the cumulative distribution function (CDF) of a normal distribution:

$$\Phi(y) = \frac{1}{2} \left( 1 + \operatorname{erf} \left( \frac{x}{\sqrt{2}} \right) \right)$$



# Activations: Gated activations (\*GLU)

- Gated activations modify the **first part** of the activations:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1)\mathbf{W}_2$$

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

$$\max(0, \mathbf{x}\mathbf{W}_1) \rightarrow \max(0, \mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V})$$

- This gives the gated variant (ReGLU) – note that we have an extra parameter  $\mathbf{V}$ :

$$\text{FFN}(\mathbf{x}) = (\max(0, \mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V}))\mathbf{W}_2.$$

[Slide credit: Tatsu Hashimoto]

# Activations: Gated activations variants

- **GeGLU**

$$\text{FFN}_{\text{GeGLU}}(\mathbf{x}; \mathbf{W}_1, \mathbf{W}_2, \mathbf{V}) = (\text{GELU}(0, \mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V}))\mathbf{W}_2.$$

**Notable models:**  
T5 v1.1, mT5, LaMDA

- **SwiGLU:** swish function is  $x * \text{sigmoid}(x)$ :

$$\text{FFN}_{\text{SwiGLU}}(\mathbf{x}; \mathbf{W}_1, \mathbf{W}_2, \mathbf{V}) = (\text{Swish}(0, \mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V}))\mathbf{W}_2.$$

**Notable models:**  
LLaMa, PaLM

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

[Slide credit: Tatsu Hashimoto]

# Do Gated Linear Units work?

- Yes, fairly consistently so.

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

[Slide credit: Tatsu Hashimoto]

# Do gated linear units work?

- Yes, fairly consistently so.

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
<b>SwiGLU</b>	$223M$	$11.1T$	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>

[Slide credit: Tatsu Hashimoto]

# Recap: Gating, activations

---

- Many variations (ReLU, GeLU, \*GLU) across models.
- \*GLU isn't necessary for a good model (see GPT3)
- But evidence points towards somewhat consistent gains from Swi/GeGLU

# Serial vs Parallel layers

# Serial vs Parallel Layer

**Notable models:**  
GPTJ, PaLM, GPT-NeoX

- Normal transformer blocks are serial – they compute attention, then the MLP
  - Can they be parallelized? GPT-J introduced a simple change to do so!

- The standard “serial” formulation:

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

- The parallel formulation:

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

- Note, LayerNorm can be shared, and matrix multiplies can be fused

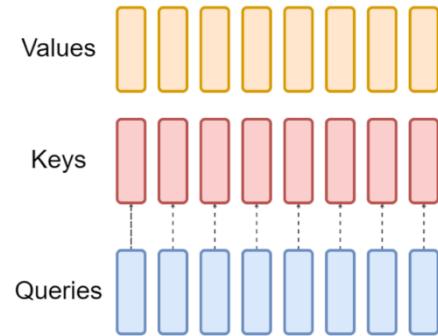
- From PaLM paper: "*The parallel formulation results in roughly 15% faster training speed at large scales ... Ablation experiments showed a small quality degradation at 8B scale but no quality degradation at 62B scale*"

# Recap

---

- **Pre-vs-post norm:**
  - Everyone does pre-norm (except OPT350M).
- **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 parallel layers have a compute win.

# Do you need all those keys?



# Self-Attention layer variations

---

- We're going to discuss a few variations of standard self-attention that are motivated by computational bottlenecks.
- Previously we talked about one bottleneck: # of arithmetic operations
- Now we're going to connect that to the # of read/writes from memory (IO)

# Diversion: Arithmetic Intensity

---

- Arithmetic Intensity of a program execution:  
 $(\# \text{ of floating-point operations}) / (\# \text{ of data bytes transferred to memory})$
- It helps determine whether a program is *compute-bound* or *memory-bound*:
  - If AI is **high**, performance is limited by how fast the GPU can compute.
  - If AI is **low**, performance is constrained by how fast data can be transferred between global memory and GPU cores.
- A good rule of thumb:
  - Memory-bound:  $\text{AI} < 10 \text{ FLOPs/byte}$
  - Balanced:  $10 \leq \text{AI} \leq 100 \text{ FLOPs/byte}$
  - Compute-bound:  $\text{AI} > 100 \text{ FLOPs/byte}$

# Quiz

- If a GPU kernel has **high** arithmetic intensity, which of the following is true?
  - A) Performance is mostly limited by memory bandwidth
  - B) Performance is mostly limited by compute throughput
  - C) Memory accesses dominate execution time
  - D) The workload is not well-suited for GPUs
- **Answer:** High AI means the GPU spends more time computing per byte of memory fetched, making it **compute-bound** rather than **memory-bound**. Hence, B.

# Arithmetic Intensity: An example

- We are going to compute AI for the first operation in Self-Attention.
- Note we assume that the full input sequence is given at once (e.g., training time).
- Given:  $\mathbf{x} \in \mathbb{R}^{b \times n \times d}$ ,  $\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{m}}$  we want to compute:  $\mathbf{x}\mathbf{W}_i^q$ . From last week:

Dimensions	Operation	Computations	IO
$\mathbf{x} \in \mathbb{R}^{b \times n \times d}$ , $\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{m}}$	$\mathbf{x}\mathbf{W}_i^q$ , $\mathbf{x}\mathbf{W}_i^k$ , $\mathbf{x}\mathbf{W}_i^v$ for $m$ heads	$O(bnd^2)$	$O(d^2 + 2bnd)$

$$\text{AI} = O\left(\frac{bnd^2}{d^2 + 2bnd}\right) = O\left(\left(\frac{d^2 + 2bnd}{bnd^2}\right)^{-1}\right) = O\left(\left(\frac{1}{bn} + \frac{2}{d}\right)^{-1}\right)$$

# Quiz

- Given:  $\mathbf{x} \in \mathbb{R}^{b \times n \times d}$ ,  $\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{m}}$  we know that the AI for computing  $\mathbf{x}\mathbf{W}_i^q$  is:

$$\text{AI} = O\left(\left(\frac{1}{bn} + \frac{1}{d}\right)^{-1}\right)$$

- This process is\_\_\_\_\_?
  - Memory-bound
  - Balanced
  - Compute-bound
- Answer:** Our AI is **large-ish**. Depending on hyperparams, this is either balanced or compute-bound.
  - If  $n = 10$  (sent len),  $b = 10$  (batch size),  $d = 512$  (hidden dim). Then  $\text{AI} = 71$ .
  - If  $n = 30$  (sent len),  $b = 20$  (batch size),  $d = 512$  (hidden dim). Then  $\text{AI} = 179$ .

# Arithmetic Intensity of Training Self-Attention

Bonus

Operation	Computations	IO	Arithmetic Intensity
$\mathbf{xW}_i^q, \mathbf{xW}_i^k, \mathbf{xW}_i^v$ for $m$ heads	$O(bnd^2)$	$O(d^2 + 2bnd)$	$O\left(\left(\frac{1}{d} + \frac{1}{bn}\right)^{-1}\right)$
$P_i \leftarrow \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d/m}}\right)$ for $m$ heads	$O(bn^2 d)$	$O(2bnd + bmn^2)$	$O\left(\left(\frac{m}{d} + \frac{1}{n}\right)^{-1}\right)$
$\text{head}_i \leftarrow P_i \mathbf{V}_i$ for $m$ heads	$O(bn^2 d)$	$O(2bnd + bmn^2)$	$O\left(\left(\frac{m}{d} + \frac{1}{n}\right)^{-1}\right)$
$Y = \text{Concat}(\text{head}_1, \dots, \text{head}_m) \mathbf{W}^o$	$O(bnd^2)$	$O(2bnd + d^2)$	$O\left(\left(\frac{1}{d} + \frac{1}{bn}\right)^{-1}\right)$

$b$ : batch size,

$n$ : sequence length,

$m$ : number of heads

$d$ : feature dimension in output of SA

$d/m$ : feature dimension inside each SA head

$d_{ff} = 4d$ : feature dimension inside FFN

16

en

All these AI values are large!  
We can continue running our  
GPUs during training! 

# Self-Attention Cost of Computation During Incremental (Autoregressive) Generation

Bonus

- Note that these numbers involve KV-caching.

Operation	Computations	IO	Arithmetic Intensity
$\mathbf{xW}_i^q, \mathbf{xW}_i^k, \mathbf{xW}_i^v$ for $m$ heads	$O(bd^2)$	$O(d^2 + 2bd)$	$O\left(\left(1/d + 1/b\right)^{-1}\right)$
These two rows have <b>low</b> AI. For example, if $n = 20$ (sent len), $h = 12$ (num heads), $d = 512$ (hidden dim), then AI = 0.93. Hence, our program is memory bound during inference! 🚨			$O\left(\left(1 + m/d + 1/n\right)^{-1}\right)$
Y = $\text{concat}(\mathbf{m}_1, \dots, \mathbf{m}_m)$	$O(bd)$	$O(bd + d)$	$O\left(\left(1 + m/d + 1/n\right)^{-1}\right)$
	$O(16bd^2)$	$O(2bd + 8d^2)$	$O\left(\left(1/d + 1/b\right)^{-1}\right)$

$b$ : batch size,

$n$ : sequence length **thus far**,

$m$ : number of heads

$d$ : feature dimension in output of SA

$d_{ff} = 4d$ : feature dimension inside FFN

# KV-Cache drag

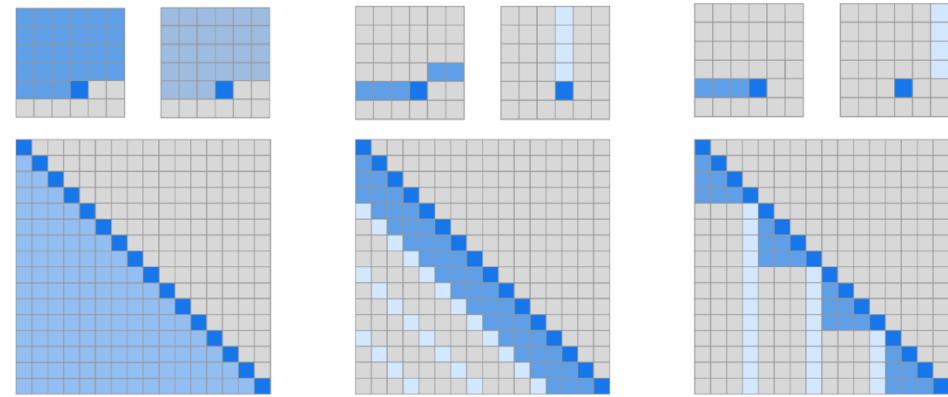
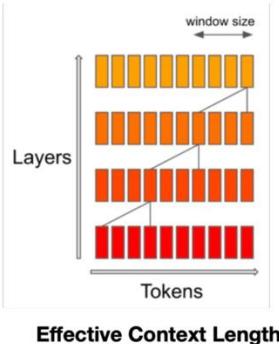
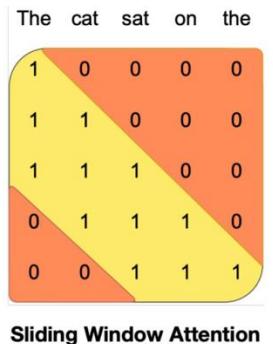
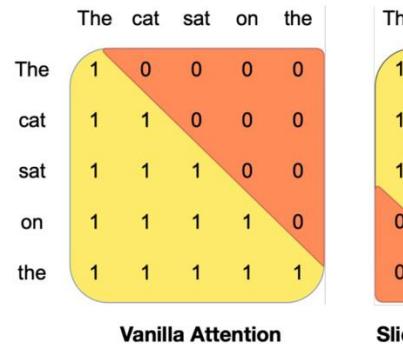
- Slowdown of autoregressive decoding.
  - As the sequence length grows, KV cache size increases, making cache lookup slower.
  - As we generate more output tokens (i.e. chatbot responding to user), throughput will slow down.
- For GPT2, this comes out to a modest ~36 MB assuming we use the max sequence length of 1024 tokens and a batch size of 1. For larger models, however, the KV Cache can take up GBs of memory.
  - [Try this calculator:](#)
- **Simple idea:** Retain only the last  $L$  tokens of the KV-cache and compute attention using these recent tokens:
  - Inference cost will be **constant**  $O(L)$  per token.

Batch Size:	Sequence Length:
1	1024

Model	Parameter Count	KV Cache Size
GPT-3 Small	125M	36.000 MB
GPT-3 Medium	350M	96.000 MB
GPT-3 Large	760M	144.000 MB
GPT-3 XL	1.3B	288.000 MB
GPT-3 2.7B	2.7B	320.000 MB

# Sparse / sliding window attention

- Right: Build sparse / structured attention that trades off expressiveness vs runtime.
- Left: Just use the main part of the strided pattern – let depth extend effective context (Mistral)



(a) Transformer

(b) Sparse Transformer (strided)

(c) Sparse Transformer (fixed)

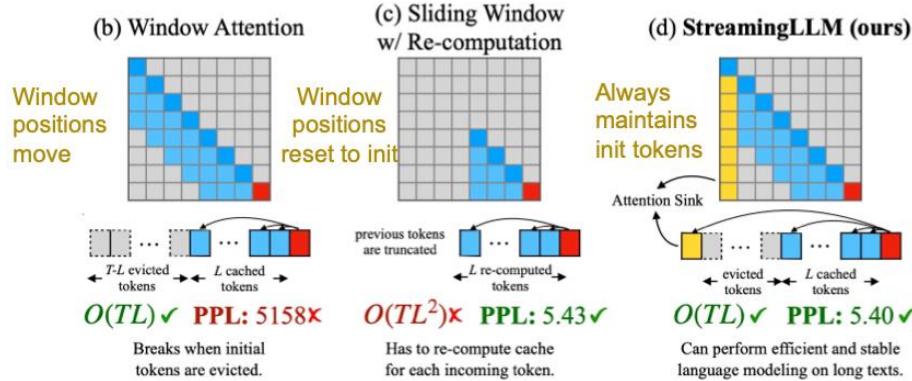
**Notable models:**  
GPT3 and Mistral

# Quiz

- What are the drawbacks of sliding window?
  1. If the model was not trained for sliding window, generation will be out-of-distribution and unstable.
  2. If uses few layers, it'll retains local/recent information and cannot see global context.
  3. After a while, it will forget the input text (e.g. the original instruction provided by the user).
  4. All of the above.

# Sliding Window Attention with “Sinks”

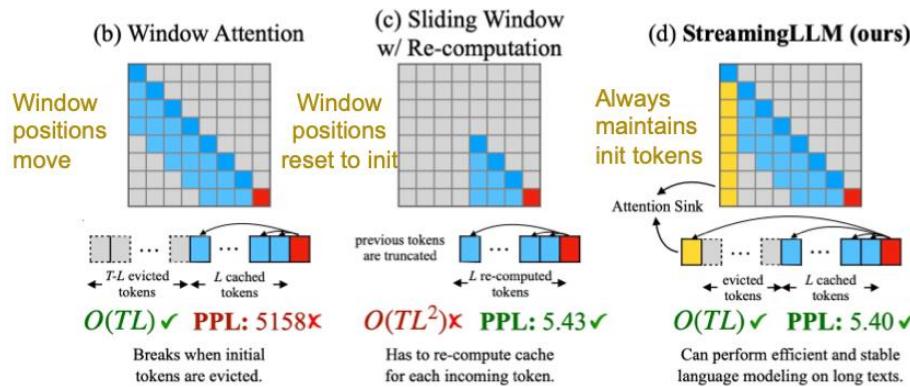
- **Idea:** We should better retain the initial tokens
  - **Intuition:** The model should hold on to the user prompt which kickstarted/instructed the LLM’s decoding
  - **During training:** The model always relies on tokens at initial positions.
    - We can’t suddenly remove initial **positions 1, 2, 3, ...** during inference.
    - Removing them results in a less stable inference (position encodings become OOD).



[Slide credit: Samet Oymak]

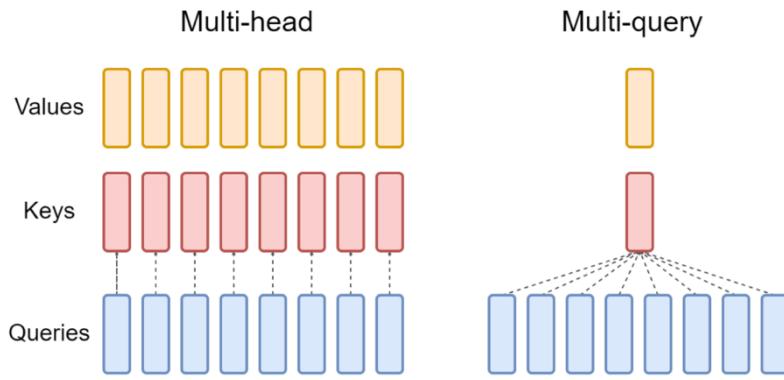
# Sliding Window Attention with “Sinks”

- **Standard Sliding Window Attention does work well** but it requires re-computation of KV cache to reset window’s positional encodings back to initial positions.
- StreamingLLM avoids this by always maintaining few initial positions (referred to as sinks).
  - Keeping initial tokens results in faster and more stable inference



# Multi-Query Attention (MQA)

- The idea is to reduce the memory-bandwidth cost of repeatedly loading the large "keys" and "values" tensors.
- Key idea** – have multiple queries, but just one dimension for keys and values.



Small PPL w/ MQA [[Shazeer 2019](#)]

Attention	$h$	$d_k, d_v$	$d_{ff}$	dev-PPL
multi-head	8	128	8192	<b>29.9</b>
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

# MQA in practice

```

13      # Independent queries, but shared keys and values
14      self.W_q = nn.Linear(embed_dim, embed_dim, bias=False) # Queries
15      self.W_kv = nn.Linear(embed_dim, 2 * self.head_dim, bias=False) # Shared Key and Value
16
17      self.out_proj = nn.Linear(embed_dim, embed_dim)

def forward(self, x):
    batch_size, seq_len, _ = x.shape

    # Compute Queries (B, L, D) → (B, L, H, D/H) → (B, H, L, D/H)
    Q = self.W_q(x).view(batch_size, seq_len, self.num_heads, self.head_dim).transpose(1, 2)

    # Compute shared Keys and Values (B, L, D) → (B, L, 2 * (D/H)) → (B, 1, L, D/H)
    KV = self.W_kv(x).view(batch_size, seq_len, 2, self.head_dim).permute(2, 0, 1, 3)
    K, V = KV[0].unsqueeze(1), KV[1].unsqueeze(1) # Shared across all heads

    # Scaled Dot-Product Attention
    attn_weights = torch.einsum("bhqd,bkhd→bhqk", Q, K) / (self.head_dim ** 0.5)
    attn_weights = torch.nn.functional.softmax(attn_weights, dim=-1)
    output = torch.einsum("bhqk,bkhd→bhqd", attn_weights, V)

    # Merge heads and apply output projection
    output = output.transpose(1, 2).reshape(batch_size, seq_len, self.embed_dim)
    return self.out_proj(output)

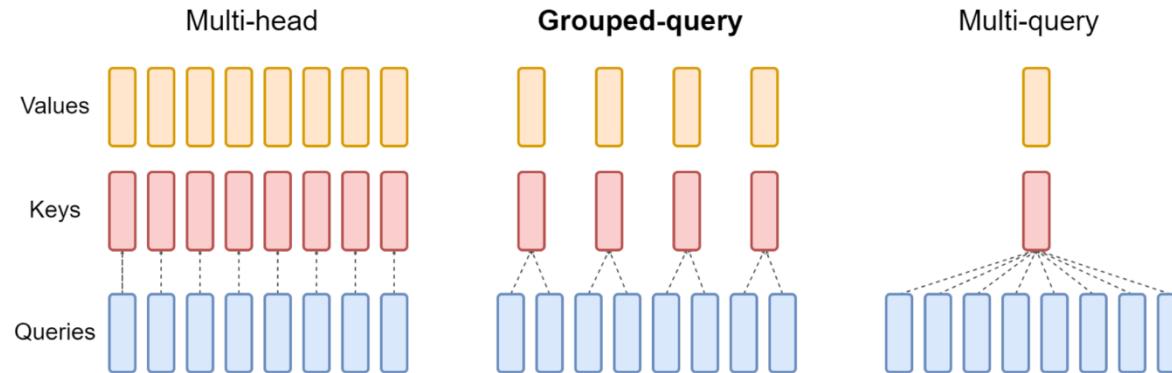
```

Script

# Grouped Query-Attention (GQA)

**Notable models:**  
Llama 2, Mistral, Qwen2

- An interpolation between “multi-head” attention and “multi-query” attention.



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

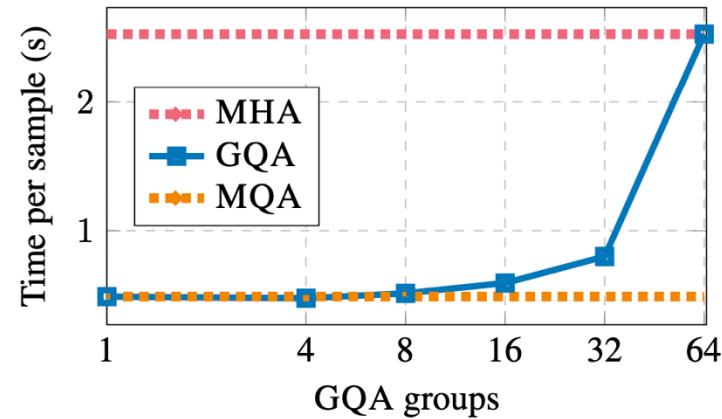
# Grouped Query-Attention (GQA)

- Does it actually work? Depends.

Output quality of various models; all these SA variants are on-par on quality.

Model	WMT	TriviaQA
	BLEU	F1
MHA-Large	27.7	78.2
MHA-XXL	28.4	81.9
MQA-XXL	28.5	81.3
GQA-8-XXL	28.4	81.6

Inference speed as a function of GQA group size — 8 heads gives you inference speed as good as 1 head!



# Recap

---

- SA's AI during inference is not good.
  - We're doing a lot of IO relative to computations (KV drag).
- Sliding window attention: sparsifying attention pattern by looking at nearby things.
- MQA and GQA: sharing attention keys and values.



# Parameter tying

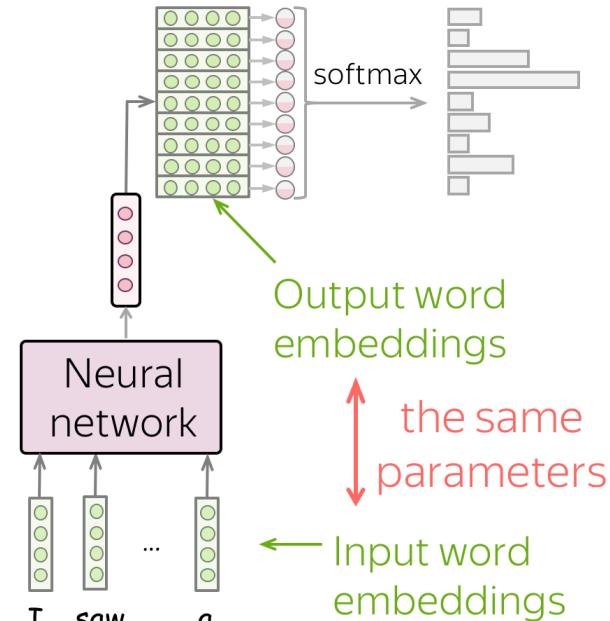


# Embedding parameter tying

- The same weight matrix is used for both the input embeddings and the output (projection) layer.

```
class TransformerWithTiedEmbeddings(nn.Module):  
    def __init__(self, vocab_size, d_model):  
        super().__init__()  
        self.embedding = nn.Embedding(vocab_size, d_model)  
        self.transformer = nn.Transformer(d_model=d_model)  
        self.output_layer = nn.Linear(d_model, vocab_size)  
  
        # Tying embeddings  
        self.output_layer.weight = self.embedding.weight
```

- Why?
  - Theoretical justification:** The input and output embeddings should exist in the same space.
  - Memory Efficiency:** reduce the # of trainable params.
  - Improved Generalization:** It enforces consistency between input vs output — the same representations are used in both encoding and decoding.



Is there a better way to encode  
positional information?

# Positional Embeddings: The Flavors

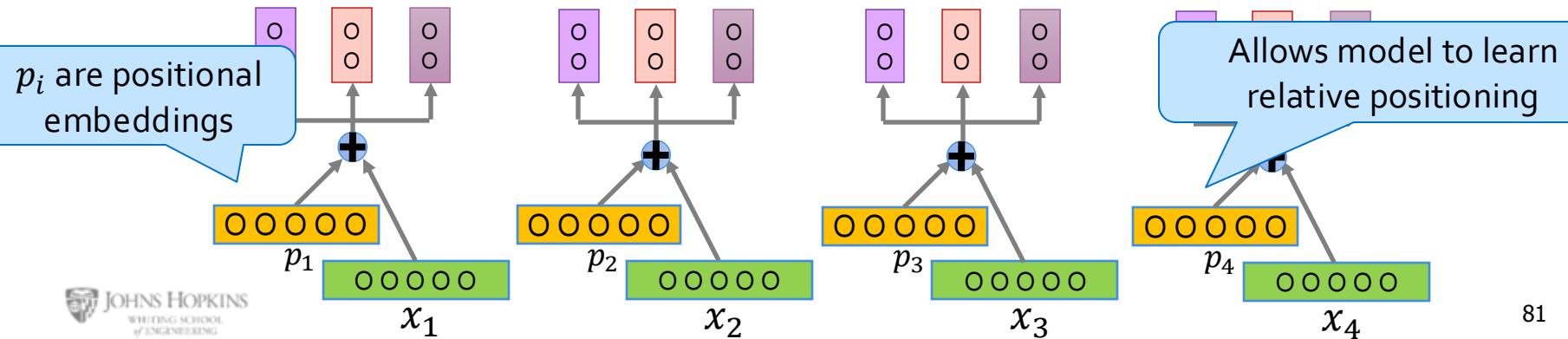
- Sine embeddings: add sines and cosines that enable localization

**Notable models:**  
Original Transformer

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

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

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



# Positional Embeddings: The Flavors

- **Sine embeddings:** add sines and cosines that enable localization

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

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

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

**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

- **Limitations:**

- We can have fixed encoding for each index training position (e.g., 1, 2, 3, ... 1000).
  - What happens if we get a sequence with 5000 words at test time?

- We want something that can generalize to arbitrary sequence lengths.

# Positional Embeddings: The Flavors

- **Sine embeddings:** add sines and cosines that enable localization

▪ Sine embeddings: add sines and cosines that enable localization

**Notable models:**  
Original Transformer

▪ Absolute embeddings: add a position vector to the embedding

▪ Relative embeddings: add a vector to the attention computation

$$QK_{ij} = x_i^T W_q^T W_k x_j + P_{ij}$$

- Intuition: encoding the relative positions, for example based on the distance of the tokens in a local window to the current token.

- **Absolute embeddings:** add a position vector to the embedding

▪ Sine embeddings: add sines and cosines that enable localization

**Notable models:**  
GPT1/2/3 - OPT

▪ Absolute embeddings: add a position vector to the embedding  
▪ Relative embeddings: add a vector to the attention computation  
 $QK_{ij} = x_i^T W_q^T W_k x_j + P_{ij}$   
○ Intuition: encoding the relative positions, for example based on the distance of the tokens in a local window to the current token.

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

$$QK_{ij} = x_i^T W_q^T W_k x_j + P_{ij}$$

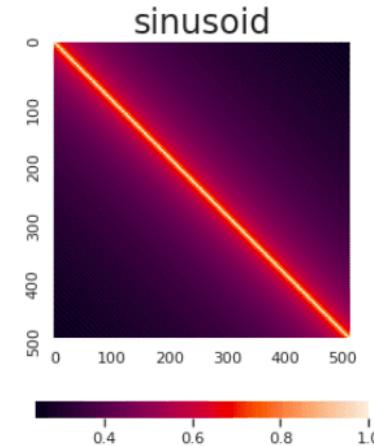
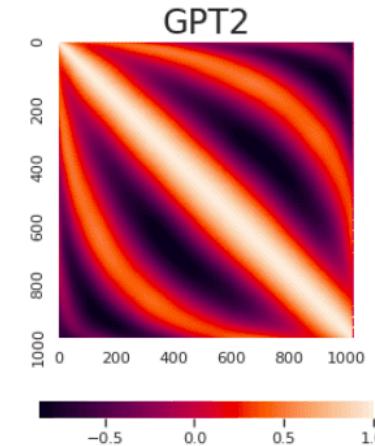
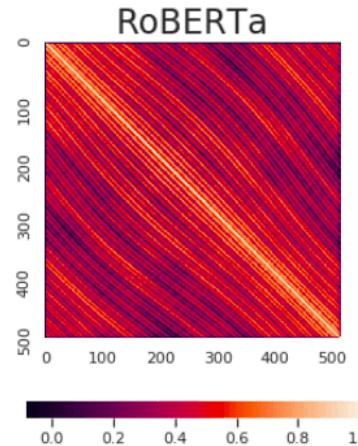
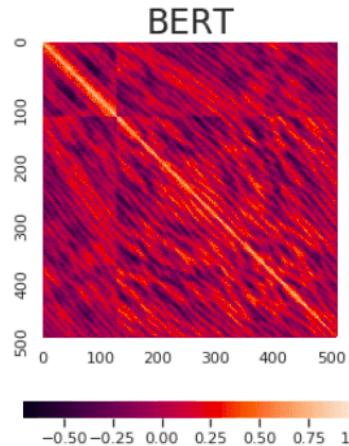
**Notable models:**  
T5, Gopher, Chinchilla

- Intuition: encoding the relative positions, for example based on the distance of the tokens in a local window to the current token.

# A Unified Perspective on Relative Positional Encoding

- You can rewrite the statement from the previous slide in the following form:

$$QK_{ij} = \mathbf{x}_i^T W_q^T W_k \mathbf{x}_j + \mathbf{P}_{ij}$$



# A Unified Perspective on Relative Positional Encoding

- We are input sequence  $x_0, x_1, \dots$  and
  - Then the unnormalized attention value between position  $i$ , and  $j$  is:

$$QK_{ij} = (\mathbf{W}_q \mathbf{x}_i)^T (\mathbf{W}_k \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j$$

- Now also assume that positional embeddings are added to  $x_i$ , i.e., they're  $\mathbf{x}_i + \mathbf{p}_i$

$$QK_{ij} = (\mathbf{W}_q [\mathbf{x}_i + \mathbf{p}_i])^T (\mathbf{W}_k [\mathbf{x}_j + \mathbf{p}_j]) = \underbrace{\mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j}_{\text{The original attention term: how much attention should we pay to word } x_j \text{ given word } x_i} + \underbrace{\mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{p}_j}_{\text{How much attention should we pay to word } x \text{ given the position } p} + \underbrace{\mathbf{p}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j}_{\text{How much attention should position } p_i \text{ attend to position } p_j} + \underbrace{\mathbf{p}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{p}_j}_{}$$

The original attention term:  
how much attention should we  
pay to word  $x_j$  given word  $x_i$

How much attention  
should we pay to word  $x$   
given the position  $p$

How much attention  
should position  $p_i$  should  
attend to position  $p_j$

# Relative Positional Encoding

- There have been various choices:
  - T5 models simplify this into learnable relative embeddings  $\mathbf{P}_{ij}$  such that:
$$QK_{ij} = \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j + \mathbf{P}_{ij}$$
  - DeBERTa learns relative positional embeddings  $\tilde{\mathbf{p}}_{i-j}$  such that:
$$QK_{ij} = \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j + \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \tilde{\mathbf{p}}_{i-j} + \tilde{\mathbf{p}}_{i-j}^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j$$
  - Transformer-XL learns relative positional embeddings  $\tilde{\mathbf{p}}_{i-j}$  and trainable vectors  $\mathbf{u}, \mathbf{v}$  s.t.:
$$QK_{ij} = \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j + \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \tilde{\mathbf{p}}_{i-j} + \mathbf{u}^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j + \mathbf{v}^T \mathbf{W}_q^T \mathbf{W}_k \tilde{\mathbf{p}}_{i-j}$$
  - ALiBi learns a scalar  $m$  such that:
$$QK_{ij} = \mathbf{x}_i^T \mathbf{W}_q^T \mathbf{W}_k \mathbf{x}_j - m |i - j|$$

# Recap

- **Sine embeddings:** add sines and cosines that enable localization
- **Absolute embeddings:** add a position vector to the embedding
- **Relative embeddings:** add a vector to the attention computation
- **RoPE embeddings:** (next slide)

**Notable models:**  
Original Transformer

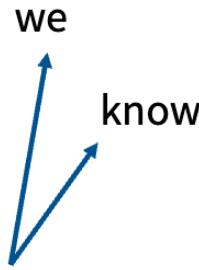
**Notable models:**  
GPT1/2/3 - OPT

**Notable models:**  
T5, Gopher,  
Chinchilla, Deberta  
Transformer-XL,

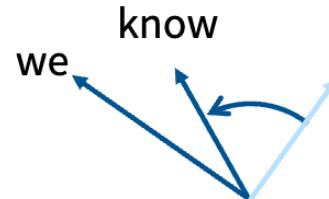
**Notable models:**  
GPTJ, PaLM, LLaMA

# Rotary Positional Encoding (RoPE)

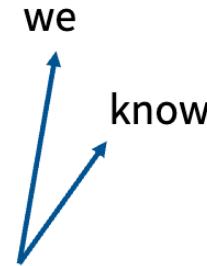
- We want our embeddings to be invariant to absolute position.
- We know that inner products are invariant to arbitrary rotation.



Position independent  
embedding



Embedding  
“of course we know”  
Rotate by ‘2 positions’



Embedding  
“we know that”  
Rotate by ‘0 positions’

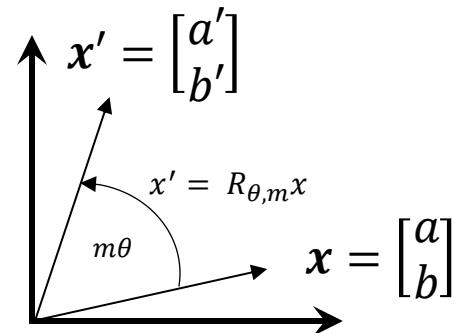
[Slide credit: Tatsu Hashimoto]

# Thinking About Rotation Matrix

- 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  $m$ .

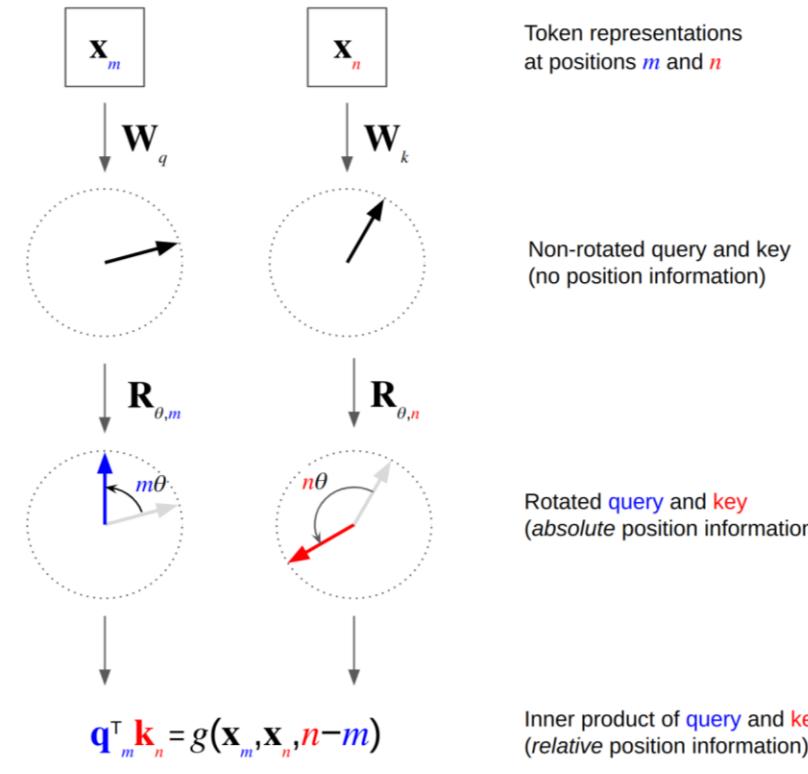


# Rotary Positional Encoding (RoPE)

- Drop the additive positional encoding and make it multiplicative.

$$\begin{aligned} qk_{mn} &= (R_{\theta,m}W_qx_m)^T(R_{\theta,n}W_kx_n) \\ &= x_m^T W_q^T R_{\theta,m}^T R_{\theta,n} W_k x_n \end{aligned}$$

- $\theta$ : the size of rotation
  - $R_{\theta,m}$ : rotation matrix, rotates a vector it gets multiplied to proportional to  $\theta$  and the position index  $m$ .
- Intuition: **nearby** words have **smaller relative rotation**.



# Thinking About Rotation Matrix

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

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

# RoPE in its General Form

$$qk_{mn} = (R_{\Theta,m}^d W_q x_m)^T (R_{\Theta,m}^d W_k x_n),$$

- where  $R_{\Theta,m}^d$  is a  $d$ -dimensional rotation matrix.
- Since  $R_{\Theta,m}^d$  is a sparse matrix, its multiplication is implemented via dense operations:

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

# 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

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

Multiply  
query/key inputs

...

Same stuff as the usual multi-head self attention below

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

[Slide credit: Tatsu Hashimoto]

# Recap

- **Sine embeddings:** add sines and cosines that enable localization
- **Absolute embeddings:** add a position vector to the embedding
- **Relative embeddings:** add a vector to the attention computation
- **RoPE embeddings:** uses rotations to encode relative distances.

**Notable models:**  
Original Transformer

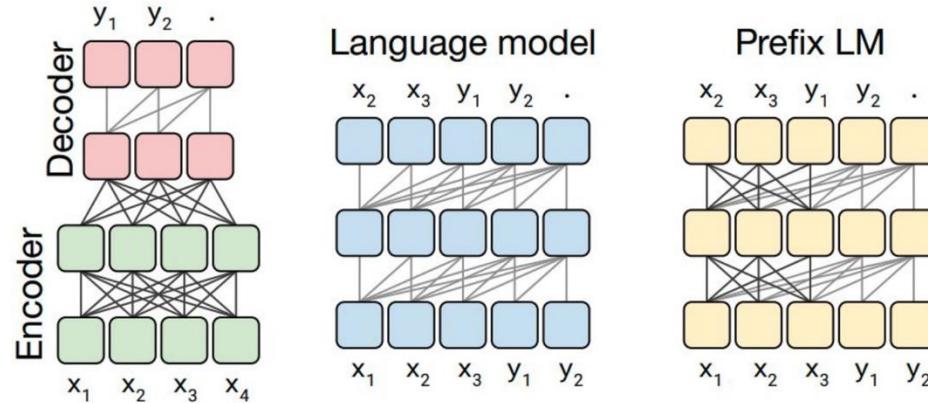
**Notable models:**  
GPT1/2/3 - OPT

**Notable models:**  
T5, Gopher,  
Chinchilla, Deberta  
Transformer-XL,

**Notable models:**  
GPTJ, PaLM, LLaMA

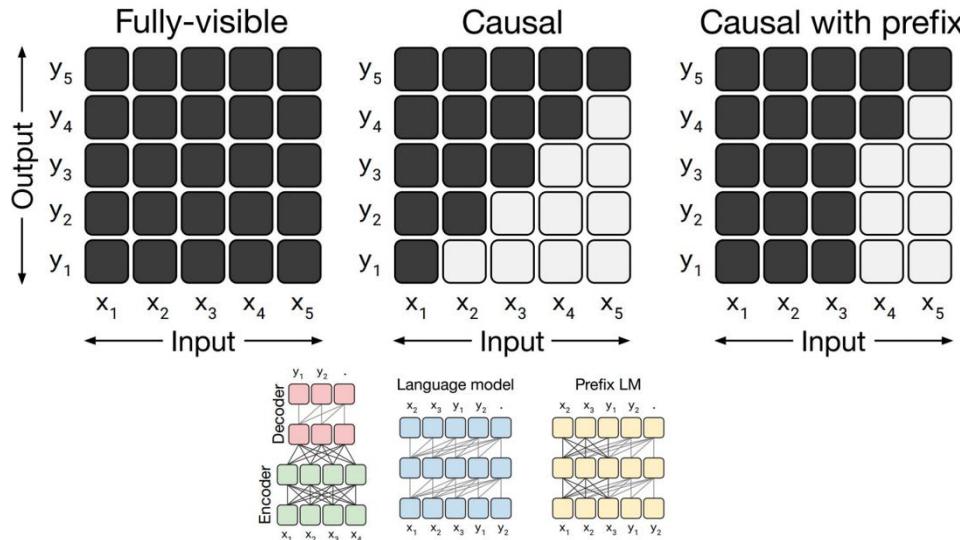
Which overall architecture  
should I use?

# Architectures: Different Choices



# Architectures: Different Attention Masks

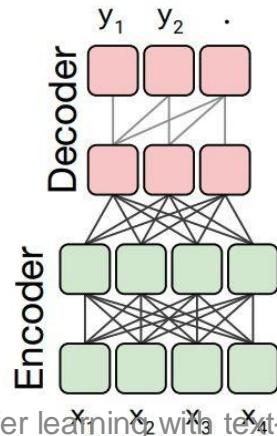
- **Fully visible** mask allows the self attention mechanism to attend to the full input.
- A **causal mask** doesn't allow output elements to look into the future.
- **Causal mask with prefix** allows to fully-visible masking on a portion of input.



# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>

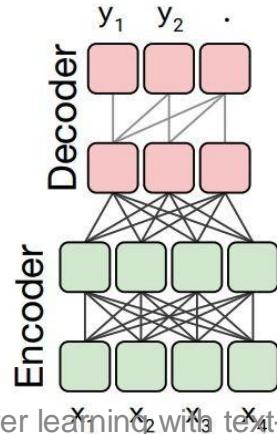


# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>

Input: Thank you for <X> me to your party  
 <Y>. Target: <X> inviting <Y> last week.

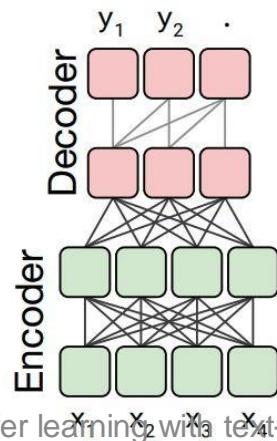


# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>

Number of parameters

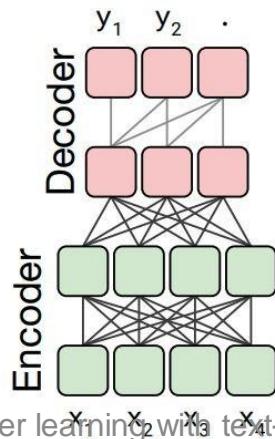


# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>

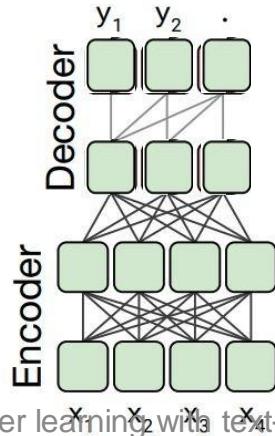
Number of FLOPS



# Architectural Variants: Experiments

Evaluated for classification tasks.

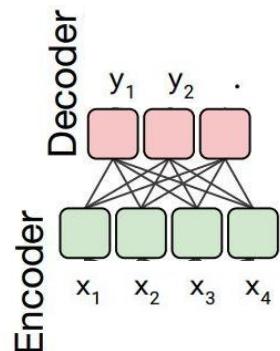
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder Enc-dec, shared	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>



# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95

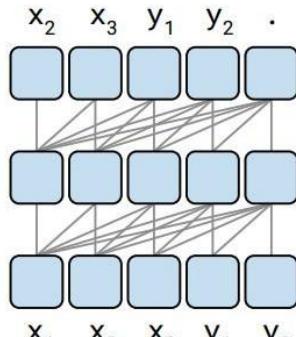


# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86

Language model



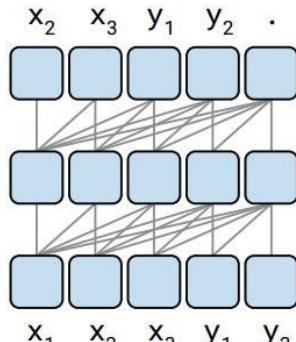
# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86

Language model is decoder-only

Language model



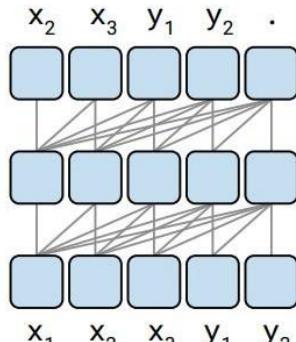
# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86

LM looks at both input and target, while encoder only looks at input sequence and decoder looks at output sequence.

Language model

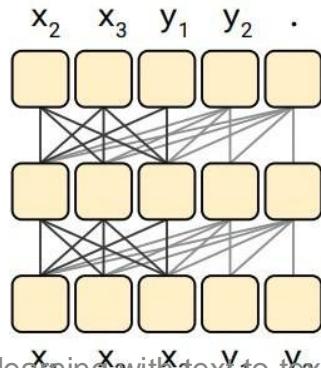


# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	$P$	$M$	81.82	18.61	78.94	68.11	26.43	37.98	27.39

Prefix LM



# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	$P$	$M$	81.82	18.61	78.94	68.11	26.43	37.98	27.39

- Takeaways:

1. Halving the number of layers in encoder and decoder hurts the performance.

# Architectural Variants: Experiments

Evaluated for classification tasks.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	$P$	$M$	81.82	18.61	78.94	68.11	26.43	37.98	27.39

- Takeaways:

1. Halving the number of layers in encoder and decoder hurts the performance.
2. Performance of Enc-Dec with shared params is almost on-par with prefix LM.



# Overall architecture



# Architecture Hyperparams

---

There are a ton of question regarding architecture hyperparameters:

- How much bigger should the feedforward size be compared to hidden size?
- How many heads? Should # of heads always divide hidden size?
- Should we make our model wide or deep?

# The Surprising Consensus #1: FFN Dimension Ratio

---

- Feedforward – model dimension ratio:

$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$
$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{\text{ff}}},$$
$$\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$$

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

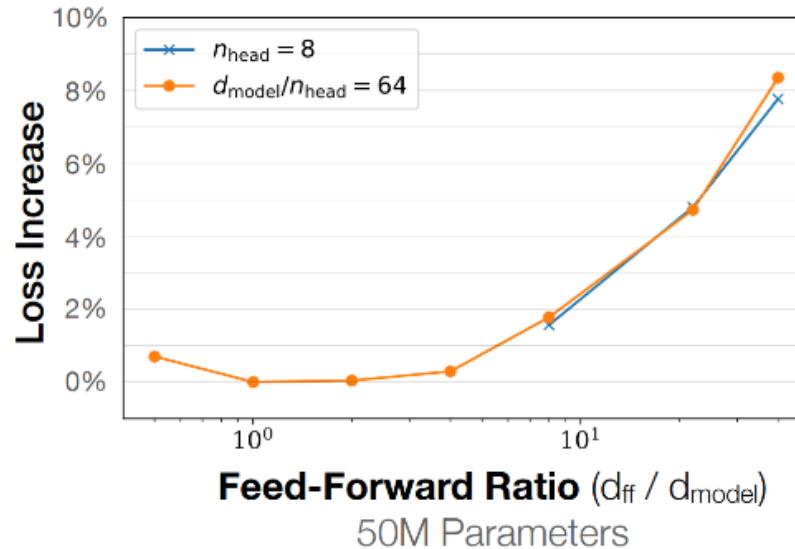
$$d_{\text{ff}} = 4d$$

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

[Slide credit: Tatsu Hashimoto]

# Why this range of multipliers?

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



# Exception #1 — GLU/Gated variants

- Remember that GLU variants scale down by 2/3 rd. This means most GLU variants have  $d_{ff} = \frac{8}{3} \times d$ . 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

[Slide credit: Tatsu Hashimoto]

# Exception #2 - T5

- As we have (and will) see, most LMs 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  $d_{\text{ff}} = 65,536$   
 $d = 1024$
- 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  $d_{\text{ff}}$  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.

# The Surprising Consensus #2: Model Dimension Ratio

- Remember:

$$\text{head}_i = \text{Attention}(\mathbf{x}\mathbf{W}_i^q, \mathbf{x}\mathbf{W}_i^k, \mathbf{x}\mathbf{W}_i^v)$$
$$\text{MultiHeadedAttention}(\mathbf{x}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^o$$

In practice, we use a reduced dimension for each head.

$$\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{h}}, \quad \mathbf{W}_i^k \in \mathbb{R}^{d \times \frac{d}{h}}, \quad \mathbf{W}_i^v \in \mathbb{R}^{d \times \frac{d}{h}}, \quad \mathbf{W}^o \in \mathbb{R}^{d \times d}$$

- The consensus: dim of head ( $\frac{d}{h}$ ) x num-heads ( $h$ ) = model-dim ( $d$ )
- This doesn't have to be true: we can have head-dimensions > model-dim / num-heads. The matrix ( $\mathbf{W}^o$ ) can take care of projection to model-dim.
  - But most models do follow this guideline

# Heads vs model dim

- Some examples of this hyperparameter:

	$h$	$d/h$	$d$	num-heads x head-dim / model-dim
	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.

# 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

- Note here width is the hidden dimension, **not** the context window width.

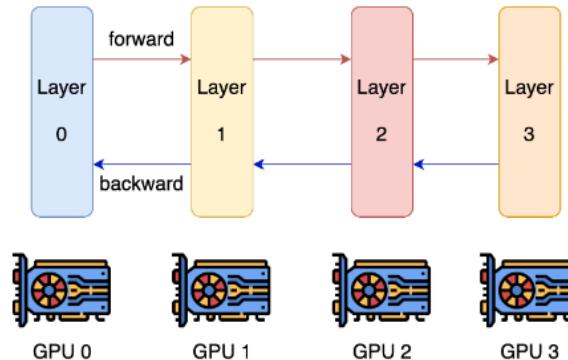
[Slide credit: Tatsu Hashimoto]

# Considerations about aspect ratio

- Extremely deep models are harder to parallelize

**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

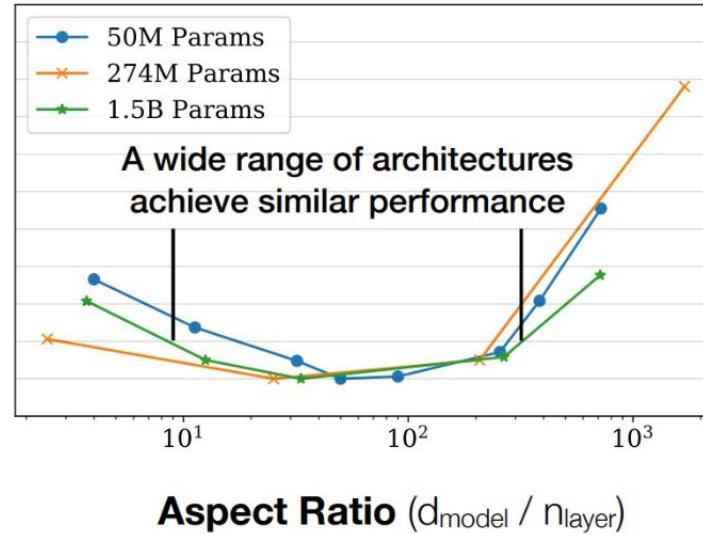
[Tay et al 2021]



[Slide credit: Tatsu Hashimoto]

# Evidence on aspect ratio scaling

Wide range of 'good' values (100-200)



[Kaplan et al 2020]

# Recap of architecture hyperparams

---

- Feedforward dimension / model dimension
  - 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 not much validation
- Aspect ratio
  - Wide range of ‘good’ values (100-200). Systems concerns dictate the value.



# Tokenizers



# What Tokenizers do people use?

- 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

- Important property – all of these tokenizers are *mostly\* invertible*.
  - \* except the ones that do lowercasing and aggressive normalization

[Slide credit: Tatsu Hashimoto]

# 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
BLOOM	250680
DeepSeek	100000
Qwen 15B	152064
Yi	64000

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

# Dealing with white spaces

Multi-whitespace  
tokenization (GPT-NeoX)

GPT-2

```
def fibRec(n):  
    if n < 2:  
        return n  
    else:  
        return fibRec(n-1) + fibRec(n-2)
```

55 tokens

GPT-NeoX-20B

```
def fibRec(n):  
    if n < 2:  
        return n  
    else:  
        return fibRec(n-1) + fibRec(n-2)
```

39 tokens

# Dealing with numbers

3.141592653589793238462643383279502884197169399  
375105820974944592307816406286208998628034825342  
117067982148086513282306647093844609550582231725  
359408128481117450284102701938521105559644622948  
954930381964428810975665933446128475648233786783  
165271201909145648566923460348610454326648213393  
607260249141273724587006606315588174881520920962

**GPT-4 and GPT-4o** tokenizers broke down numerical sequences into groups of 3.

<bos>3.14159265358979323846264338327950288419716  
939937510582097494459230781640628620899862803482  
534211706798214808651328230664709384460955058223  
172535940812848111745028410270193852110555964462  
294895493038196442881097566593344612847564823378  
678316527120190914564856692346034861045432664821  
339360726024914127372458700660631558817488152092

**Mixral, Llama, DeepSeek, and Gemma** tokenizers broke down numerical sequences into a separate token for each digit.

**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.

# Tokenizers

---

- Everyone uses invertible subword tokenizers (BPE, Unigram) for good reason.
- For math and code, careful manual handling of whitespace and numbers can help.

# Summary of LLM architectures

- 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	□	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/go...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 claudie)	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...48.pdf	2022	BPE	50257	LayerNorm	Parallel	✓	RoPE	GeLU	□	4	44	
GPT4	OPEN	Ad	arxiv.org/abs...08774	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	

# Pre-training language models: Pre-training data

# The pre-training data size and sources

- They vary quite a bit!
- They used to be in billions of tokens; now they're north of trillions.

Model Name	Release	Pre-training data #Tokens	Training Dataset
BERT	2018	3.3B	BooksCorpus (800M), English Wikipedia (2.5B)
GPT-1	2018	13B	BooksCorpus
GPT-2	2019	40B	WebText: scraping outbound links from Reddit post with >= 3 karma
T5	2019	34B	C4 which is the cleaned up version of CommonCrawl
GPT-3	2020	400B	Common Crawl (filtered), WebText2, Mystery books!! (Books1, Books2), Wikipedia
Gopher	2021	1.4T	MassiveText
BLOOM	2022	350B	ROOTS corpus, a dataset comprising hundreds of sources in 46 natural and 13 programming languages (59 in total)
PaLM	2022	2.81T	Web documents, books, Wikipedia, conversations, GitHub code
LaMDA	2022	1.56T	Public dialog data and web documents
Chinchilla	2022	1.4T	MassiveText
LLaMA2	2023	2.0T	A new mix of publicly available online data
GPT-4	2023	?	?
Claude-3	2023	?	?
OLMo 2	2024	5.6T	OLMo-Mix-1124(stage1) + Dolmino-Mix-1124(stage 2)
Qwen2.5	2024	7T	
DeepSeek (V3)	2024	14.8T	GitHub's Markdown and StackExchange
LLaMA3	2024	15T	A new mix of publicly available online data

# Where do we begin to collect data?

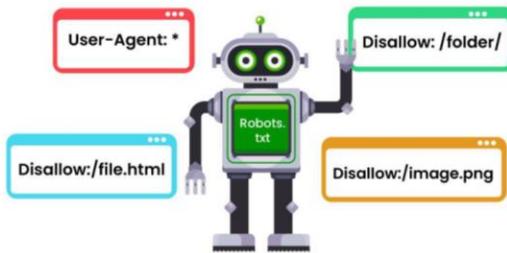
---

- Where do I find a very large dataset?
  - Crawling web is non-trivial (unless you're OpenAI or Google with ton of resources).
  - But if you have to do it, be aware that websites have their own permissions regarding which parts of their content, if any, can be crawled. (next slide)
  - The alternative is to look for websites that have done the crawling for you.

# Robots.txt



- A plain text file that tells web crawlers which parts of a website they can access.
- When a web crawler visits a website, it first checks the robots.txt file (if available) before crawling other pages.



- AI companies release the details of their crawlers:  
<https://platform.openai.com/docs/bots/>

```
# Squarespace Robots Txt
```

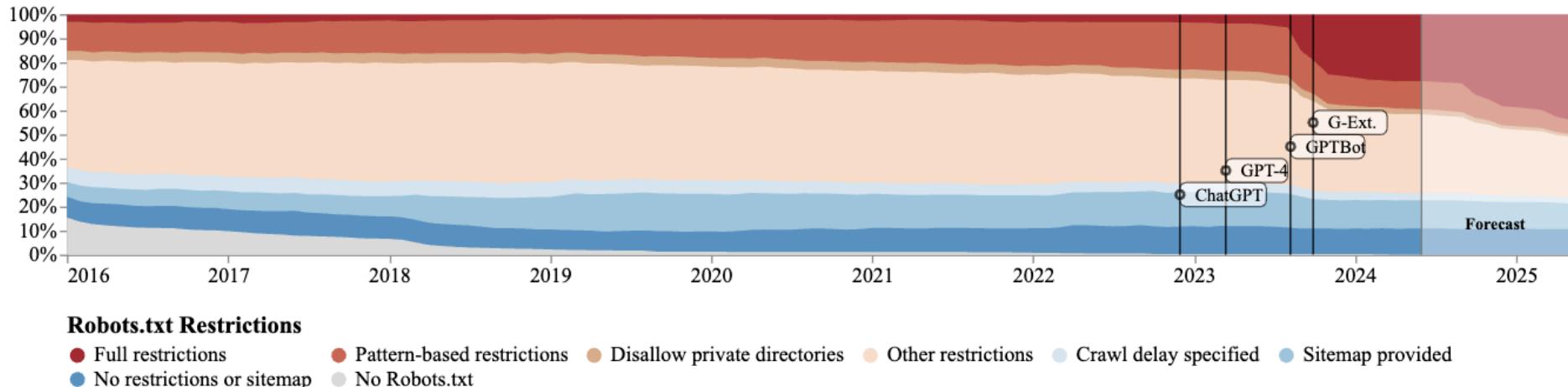
```
User-agent: GPTBot
User-agent: ChatGPT-User
User-agent: CCBot
User-agent: anthropic-ai
User-agent: Google-Extended
User-agent: AdsBot-Google
User-agent: AdsBot-Google-Mobile
User-agent: AdsBot-Google-Mobile-Apps
User-agent: *
Disallow: /
```

More examples:

<https://www.youtube.com/robots.txt>  
<https://www.jhu.edu/robots.txt>

# Robots.txt's are becoming increasingly more restrictive

- A longitudinal analyses show that in the past few years, a major chunk of websites have restricted their data to AI crawlers.



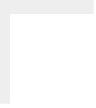
# CommonCrawl



- A non-profit organization that release a new crawl of the internet every month they.
  - So far, there have been ~100 crawls from 2008-2024.
  - In 2016, a crawl took 10-12 days on 100 machines. They used [Apache Nutch](#).
  - This is **not** a complete of the internet. Crawls have some overlap but try to diversify.
    - Common Crawl follows links from previously crawled pages.
  - Also note, it respects robots.txt
- CC is a common sources of pre-training data.
  - **WARC**: The raw HTTP responses, including full web pages.
  - **WAT**: The metadata summary from WARC files.
  - **WET**: The extracted plaintext from WARC files, stripping out HTML and other non-textual content.

Data Type	File List	#Files	Total Size Compressed (TiB)
Segments	<a href="#">segment.paths.gz</a>	100	
WARC	<a href="#">warc.paths.gz</a>	90000	76.08
WAT	<a href="#">wat.paths.gz</a>	90000	17.68
WET	<a href="#">wet.paths.gz</a>	90000	7.00
Robots.txt files	<a href="#">robotstxt.paths.gz</a>	90000	0.15
Non-200 responses	<a href="#">non200responses.paths.gz</a>	90000	2.59
URL index files	<a href="#">cc-index.paths.gz</a>	302	0.19
Columnar URL index files	<a href="#">cc-index-table.paths.gz</a>	900	0.22

<https://data.commoncrawl.org/crawl-data/CC-MAIN-2024-30/index.html>



# CC is messy. Is that a concern?

Besides quantity, the choice of dataset is also critical

# Garbage in Garbage OUT



# C4: A cleaned up pre-training dataset

---

- C4: Colossal Clean Crawled Corpus
  - The course is CommonCrawl.
  - English language only
  - 750GB after ton of filtering
- Notice that the unfiltered data is quite large.
  - Common Crawl is mostly not useful natural language

Data set	Size
★ C4	745GB
C4, unfiltered	6.1TB

# C4: The Data

Remove any:

- References to Javascript
- Pages with "{} (no code), "[Lorem ipsum](#)" text (dummy text), "terms of use", etc.
- Pages with "[bad words](#)".

Menu

Lemon

Introduction

The lemon, Citrus Limon (L.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae. The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Article

The origin of the le

vn, though

Retain:

- Sentences with terminal punctuation marks
- Pages with at least 5 sentences, sentences with at least 3 words

Please enable JavaScript to use our site.

Home

Products

Shipping

Contact

FAQ

Dried Lemons, \$3.59/pound

Organic dried lemons from our farm in California.

Lemons are harvested and sun-dried for maximum flavor.

Good in soups and on popcorn.

The lemon, Citrus Limon (L.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.

The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

orem ipsum dolor sit amet, consectetur adipiscing elit.

Curabitur in tempus quam. In mollis et ante at consectetur.

Aliquam erat volutpat.

Donec at lacinia est.

Duis semper, magna tempor interdum suscipit, ante elit molestie urna, eget efficitur risus nunc ac elit.

Fusce quis blandit lectus.

Mauris at mauris a turpis tristique lacinia at nec ante.

Aenean in scelerisque tellus, a efficitur ipsum:

Integer justo enim, ornare vitae sem non, mollis fermentum lectus.

Mauris ultrices nisl at libero porta sodales in ac orci.

```
function Ball(r) {  
    this.radius = r;  
    this.area = pi * r ** 2;  
    this.show = function(){  
        drawCircle(r);  
    }  
}
```

# Pre-training Data: Experiment

- Takeaway:
  - Clean and compact data is better than large, but noisy data.
  - Pre-training on in-domain data helps.

Data set	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	<b>19.24</b>	80.88	71.36	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21



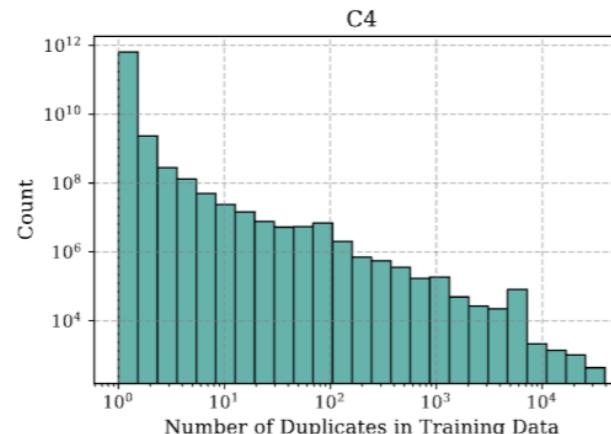
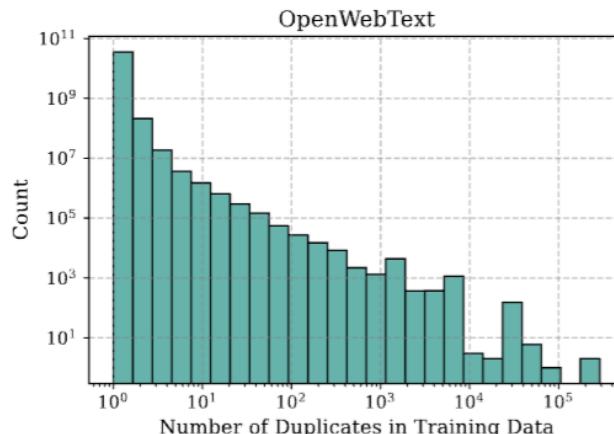
Does it matter that my  
data has ton of repetitions?



# Pre-training Data Duplicates

- There is a non-negligible number of duplicates in any pre-training data.

	% train examples with dup in train	% valid with dup in train
C4	3.04%	1.59%
RealNews	13.63%	1.25%
LM1B	4.86%	0.07%
Wiki40B	0.39%	0.26%



# Pre-training Data Duplicates

- There is a non-negligible number of duplicates in any pre-training data.
- Maybe we should not spend our training budget re-learning things we have already seen.

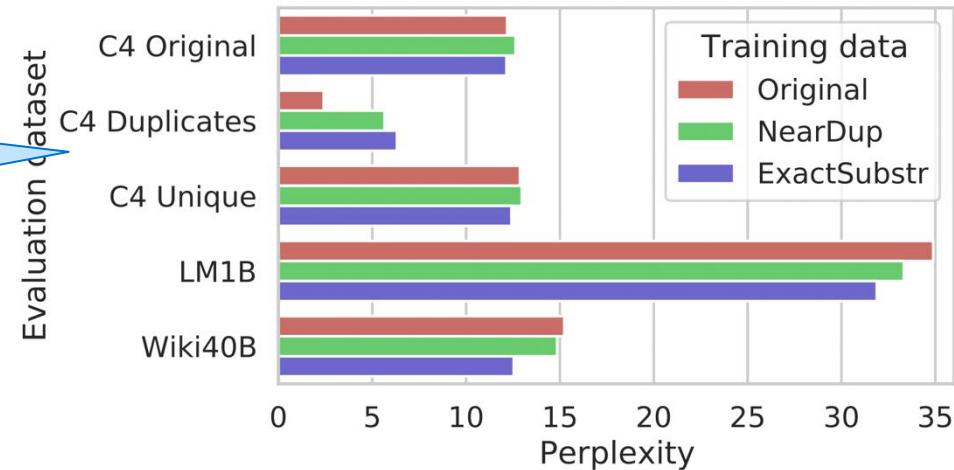
Dataset	Example	Near-Duplicate Example
Wiki-40B	\n_START_ARTICLE_\nHum Award for Most Impactful Character \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]	\n_START_ARTICLE_\nHum Award for Best Actor in a Negative Role \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]
LM1B	I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .	I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .
C4	Affordable and convenient holiday flights take off from your departure country, "Canada". From May 2019 to October 2019, Condor flights to your dream destination will be roughly 6 a week! Book your Halifax (YHZ) - Basel (BSL) flight now, and look forward to your "Switzerland" destination!	Affordable and convenient holiday flights take off from your departure country, "USA". From April 2019 to October 2019, Condor flights to your dream destination will be roughly 7 a week! Book your Maui Kahului (OGG) - Dubrovnik (DBV) flight now, and look forward to your "Croatia" destination!

# Deduplicating Data Improves LMs

- Models: GPT-2-like (1.5B param) models
- On these datasets:
  - C4 : the original training data
  - C4-NearDup: C4 excluding exact duplicates
  - C4-ExactSubs: C4 excluding near-duplicates

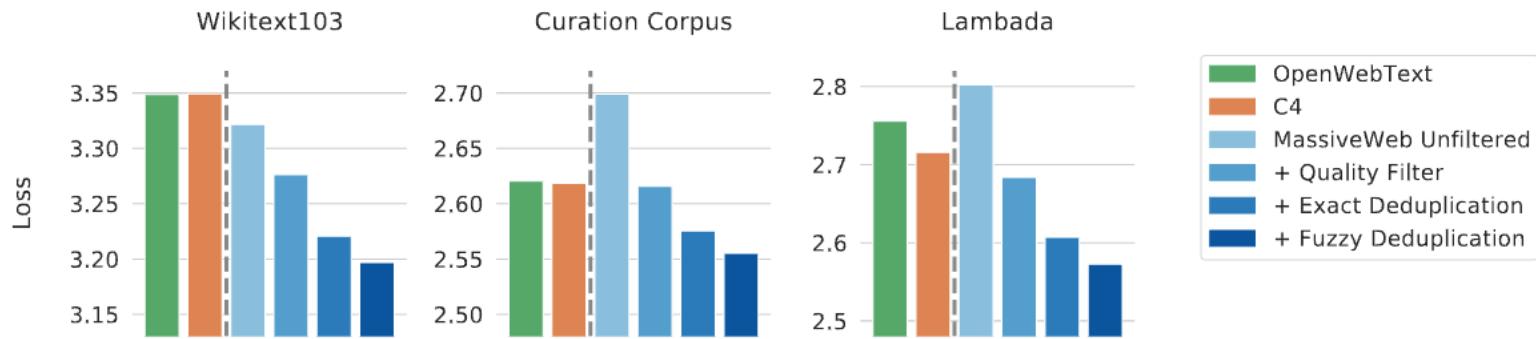
Except when evaluated on duplicate evaluation data!

Training on deduplicated data always leads to lower PPL!



# Deduplicating Data Improves LMs

- Another evidence from Gopher paper: Performance of 1.4B parameter models (lower is better) trained on OpenWebText, C4, and versions of MassiveWeb with progressively more pre-processing stages added.
- Applying a quality filter and de-duplication stages significantly improves quality.





# How can I do my own deduplication?



# How do you scale data deduplication?

---

- Pre-training is huge. Naively deduplicating the data is going to take forever!!
- How do you deduplicate it? Here are a few options:
  - SuffixArray
  - MinHash
  - BloomFilters
  - Embedding-based dedup

# The simplest: hashing documents

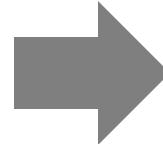
- Hash all documents, so each document receives one unique hash.
- **Efficiency:** This will be fast.
- **Granularity:**
  - This will be sensitive to small changes; any change in the document (e.g., one word change) would change its hash.
  - Also, we're deduplicating full documents.
- Different choices of hashing functions (trade off between efficiency vs collision):
  - Collision:  $h(x) = h(y)$ , if  $x \neq y$ .
  - Cryptographic hashing (SHA-256, SHA-3, BLAKE2); collision resistant but slow.
  - DJB2, MurmurHash, CityHash: Not collision resistant but fast.

# What are Suffix Arrays?

- A common approach is using **Suffix arrays** — A suffix array for a string  $T$  (of length  $m$ ) is an array of integers  $[0, m)$  that correspond to suffixes of  $T\$$ , stored in sorted order.
  - Example:  $T = \text{"abaaba\$"}$
- Space complexity:
  - $O(m)$

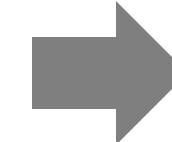
0	abaaba\$
1	baaba\$
2	aaba\$
3	aba\$
4	ba\$
6	a\$
7	\$

Sort suffixes  
lexicographically



6	\$
5	a\$
2	aaba\$
3	aba\$
0	abaaba\$
4	ba\$
1	baaba\$

Now you can  
drop the strings



6
5
2
3
0
4
1

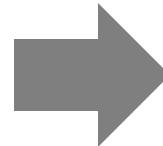
# What are Suffix Arrays?

- A common approach is using **Suffix arrays** — A suffix array for a string  $T$  (of length  $m$ ) is an array of integers  $[0, m)$  that correspond to suffixes of  $T\$$ , stored in sorted order.
  - Example:  $T = \text{"abaaba\$"}$

- Space complexity:
  - $O(m)$

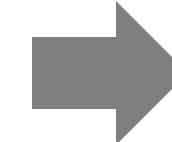
0	abaaba\$
1	baaba\$
2	aaba\$
3	aba\$
4	ba\$
6	a\$
7	\$

Sort suffixes  
lexicographically



6	\$
5	a\$
2	aaba\$
3	aba\$
0	abaaba\$
4	ba\$
1	baaba\$

Now you can  
drop the strings



6
5
2
3
0
4
1

- You don't need the suffixes since, given their index, you can look them up from  $T$ .

# Suffix arrays: querying

- Querying: Is  $P$  a substring of  $T$ ?
- Two crucial observations:
  1. For  $P$  to be a substring, it must be a prefix of  $\geq 1$  of  $T$ 's suffixes.
  2. Suffixes sharing a prefix are consecutive in the suffix array.

- Example: Given SA of  $T = \text{"abaaba\$"}$  find the indices (if any) of substring  $P = \text{"aba"}$ .
- In practice, we can use binary search to check whether  $P$  is a prefix of any suffix.
- Complexity:  $O(n \log m)$ 
  - o for  $m = \text{len}(T)$  and  $n = \text{len}(P)$
  - o See an example [here](#).

6	\$
5	a\$
2	aaba\$
3	aba\$
0	abaaba\$
4	ba\$
1	baaba\$

Now you can drop the strings

6
5
2
3
0
4
1

# LCS with SuffixArrays

- Suffix arrays also allows us to quickly check overlap between pairs of documents.
- Querying: Given SA of T, what is its Longest Common Subsequence (LCS) with P?
- This can also be done with binary search  $O(n \log m)$  for  $m = \text{len}(T)$  and  $n = \text{len}(P)$ .
- See an example [here](#).

Find more about these algorithms in Ben Langmead's course: <https://www.langmead-lab.org/teaching.html>

# Deduplication with Suffix Arrays

- Concatenating all text in the corpus together and then sorting each suffix.
- By scanning this sorted list, substrings with a common prefix can be identified by scanning the prefixes of neighboring elements in the sorted list.
- This latter step can be done in an embarrassingly parallel fashion.
- **Granularity:**
  - Note SAs can only do exact deduplication!
  - But it can allow you to do deduplication on substrings/sub-documents.
- **Hyperparameter:** the length of overlap
  - Lee et al. deduplicated substrings that are at least 50 tokens long.

See example here: <https://github.com/google-research/deduplicate-text-datasets/blob/master/README.md>  
Uses MinHash: [Lee et al. Deduplicating Training Data Makes Language Models Better, 2020](#)

# Deduplication with MinHash

- **MinHash** is a locality-sensitive hashing technique used to group sets into collections based on their Jaccard similarity.
  - Note, unlike SuffixArrays, MinHash can do “fuzzy” deduplication!
  - Hyperparameters: the n-gram-size, and the number of permutations used.
  - Lee et al used:
    - n-gram-size of 5 tokens and Jaccard sim < 0.8;
    - 9K permutations, split into 450 buckets of 20 hashes each.
  - Li et al. used: 1,395 permutations, split into 93 buckets of size 15.

Uses MinHash: [Lee et al. Deduplicating Training Data Makes Language Models Better, 2020](#)

Uses MinHash: Li et al. [DataComp-LM: In search of the next generation of training sets for language models, 2024](#)

Also see: <https://blog.nelhage.com/post/fuzzy-dedup/>

# Deduplication with BloomFilters

- **Bloom filters** are a data structure that enable space-efficient set membership queries.
  - A Bloom filter maintains a sketch of a set (in sublinear space) that supports an
    - insert operation,
    - a probabilistic membership\_query operation.
      - Note: The latter operation has no false negatives (i.e., return False for an element in the set), but it may occasionally return a false positive (i.e., return True for an element not in the set).
- **Efficiency:** Li et al. say that BF is “vastly more efficient than a MinHash and SuffixArrays.”
- **Granularity:**
  - Can be used for both exact dedup (like Sondaini et al) and “fuzzy” dedup!
  - Caveat: MinHash performs doc-level deduplication at a document vs. document level, whereas BFF performs document-level deduplication at a document vs. corpus level.
- Hyperparams: Number of hashers which determines the false positive rate.

BloomFilters: [Space/time trade-offs in hash coding with allowable errors, 1970](#)

Uses BloomFilter: [Sondaini et al. Dolma: An open corpus of three trillion tokens for language model pretraining research, 2024](#) <https://github.com/allenai/bff>

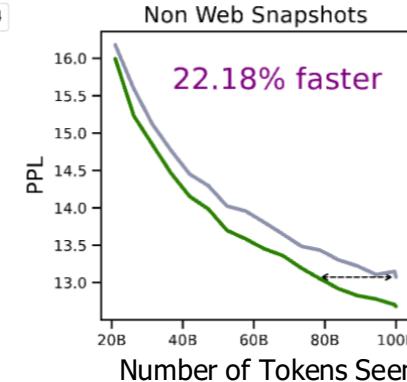
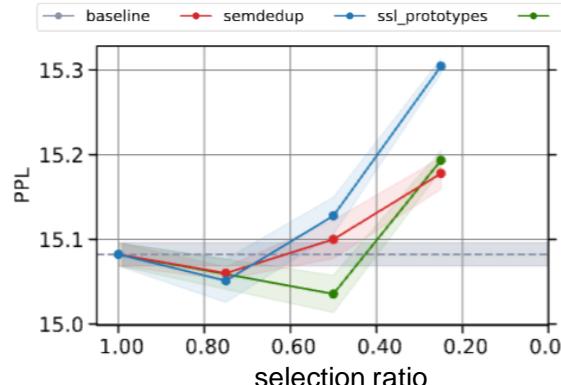
# Comparison between dedup algorithms

- Single methods: BF better than any other method standalone.
- Combination: The competitive approaches are last row (exact -> MH -> SA) and BF-only. The former leads to more compact data.

Exact Dedup	MinHash	Suffix Array	Bloom Filter	Tokens	Removal Rate	CORE	Δ from Baseline
✗	✗	✗	✗	76B	00%	40.1	+0.0
✓	✗	✗	✗	66B	13%	41.0	+0.9
✗	✓	✗	✗	62B	18%	40.9	+0.8
✗	✗	✓	✗	51B	33%	41.4	+1.3
✗	✗	✗	✓	<b>56B</b>	<b>26%</b>	<b>41.7</b>	<b>+1.6</b>
Individual technique	✓	✓	✗	58B	24%	40.2	+0.1
	✓	✗	✓	49B	36%	41.3	+1.3
	✗	✓	✓	48B	37%	41.2	+1.2
	✓	✓	✓	<b>45B</b>	<b>41%</b>	<b>41.7</b>	<b>+1.6</b>
Combined techniques							

# Deduplication in embedding space

- D4 performs dedup in embedding space of sentences by a pre-trained sentence embedder:
  - (1) deduplication: drop data points in epsilon-ball around each data point.
  - (2) diversification: k-means to cluster points and drop those far from centroids
- Does it work?
  - Yes, it gives 22% training speedup over baseline (random selection).
  - Is it better than MinHash? Depends 😐



# Deduplication: Recap

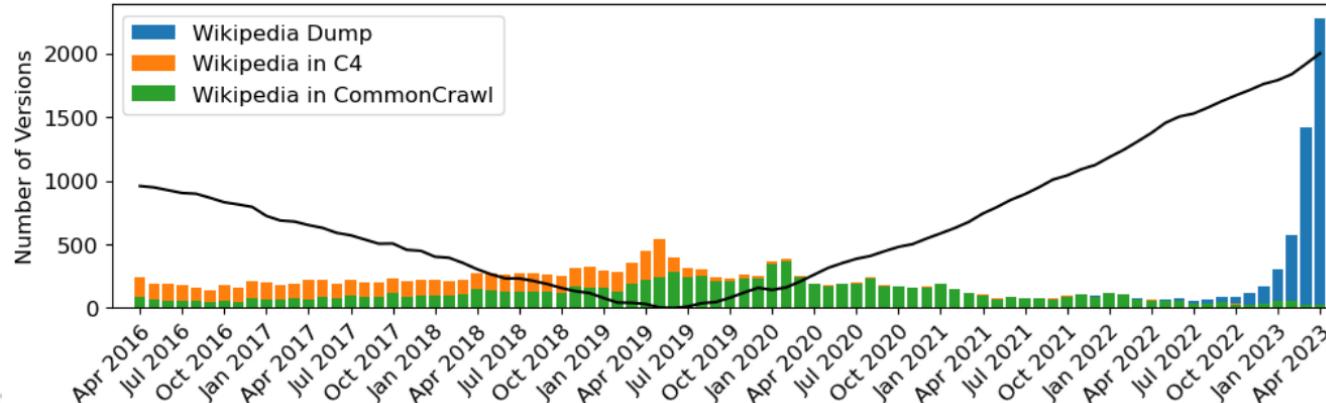
---

- Does it matter that my data has tons of repetitions? Yes, one should do careful dedup.
- How can I do my own deduplication?
  - Scaling it up requires advanced data structures.
  - So far, there is no clear winner between these algorithms. A “kitchen sink” approach that mixes dedup algorithms is generally best, but it’s an empirical exercise.
  - BF is generally preferred since it’s cheaper/faster.

# Should I worry about old data in my pre-training?

# Prevalence of stale data

- Breakdown of old versions of Wikipedia in RedPejamas
- RedPejamas which is based on dumps from C4, CC and a recent Wikipedia dump.
- The bars below show the breakdown of older versions of Wikipedia in RedPajamas.
  - There is a ton of old Wikipedia versions in RedPejamas! 😱
- The solid trend is the perplexity of a pre-trained model on temporal instances of Wikipedia.
  - The significant stale training data in has skewed PPL toward older versions of Wikipedia.



Should I worry about skew of  
the data mixtures in my pre-training?

# Data Mixtures

- Your dataset mixture will determine the versatility of the resulting model.
- Data in the world is always skewed. For example,
  - English has a lot more language than other domains.
  - Reddit is a lot larger than science papers.
- A uniform “weight” of data during pre-training is not good since overrepresented domains would dominate (e.g., your model would be a better at English than Azeri).
- Overamplifying underrepresented domains also runs risk of overfitting.
- So, there is a lot of research on finding good balance.

# Language filtering

- Many works limit their data to English.
- Chinese models (e.g., Qwen and DeepSeek) are mostly English + Chinese.
- The issue is the difficulty curating high-quality data. Also cost training on more data.
- GPT-4, Claude, Gemini are all multilingual.
- How do people identify languages? A popular choice is [fastText](#) which supports 176 langs.
- Danger in English-only filtering:
  - accidentally filtering out dialect of English.
  - Ill-defined for code-switching (e.g., English + Chinese).



# Few notable data pipelines

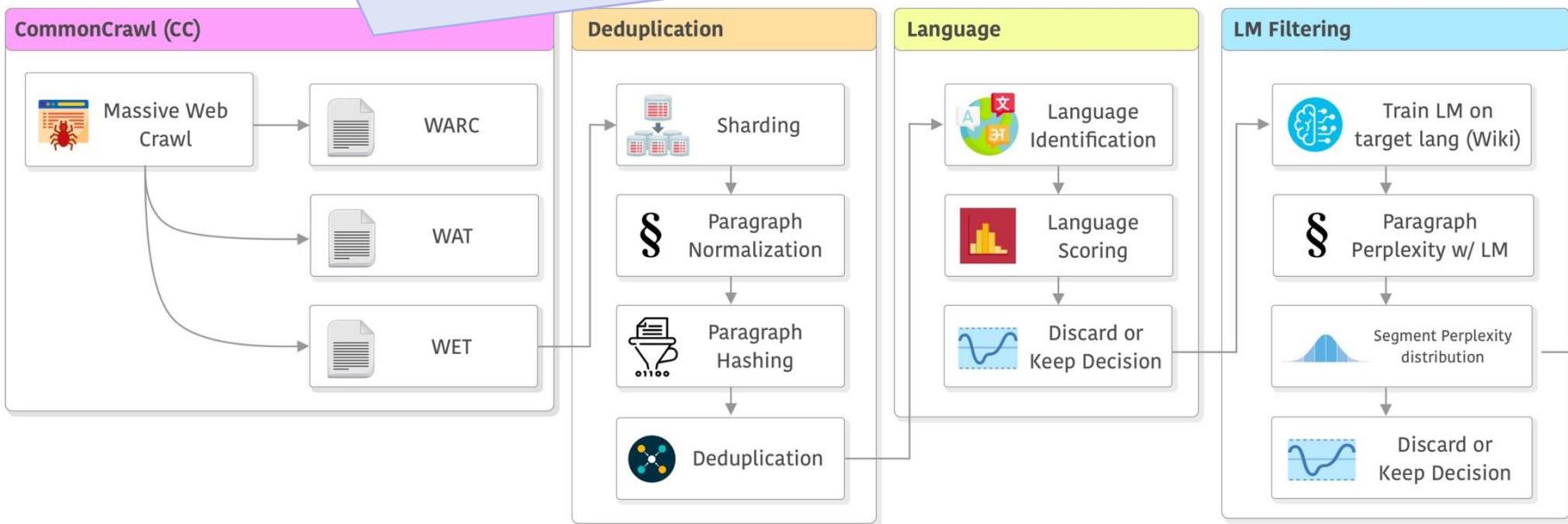


# LLaMA 1's Data Pipeline

Starts with the massive crawled data by CommonCrawl.

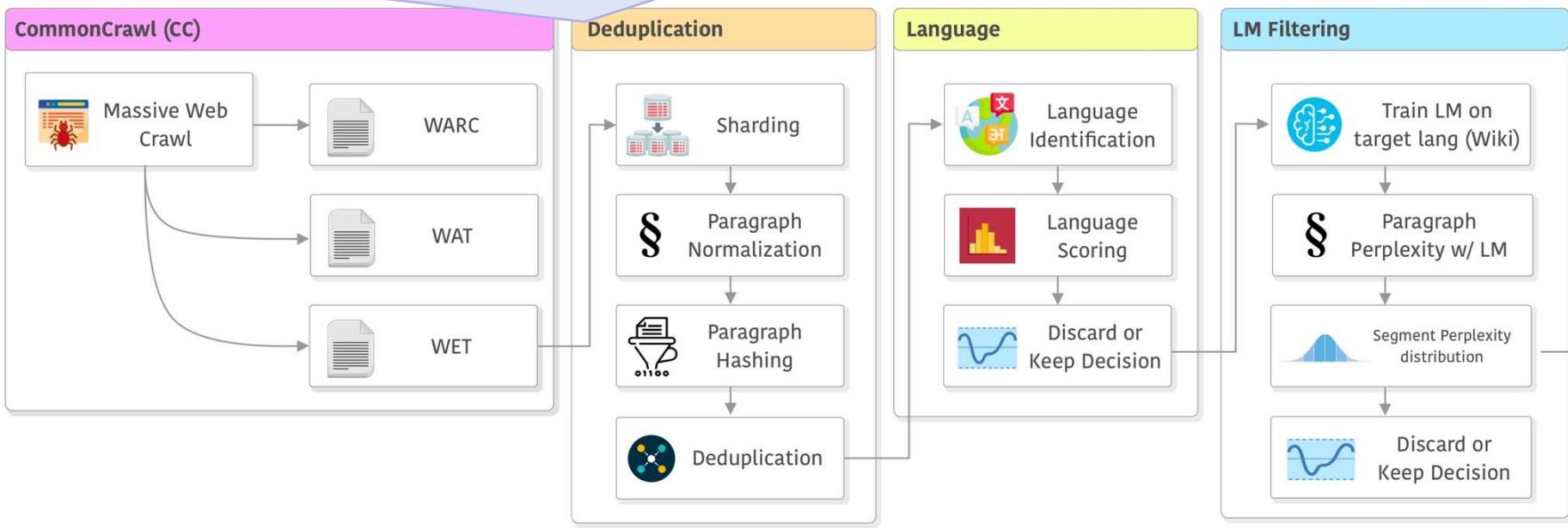
The WET format that contains textual information.

WARC is raw, WAT is metadata, WET is text+some metadata.



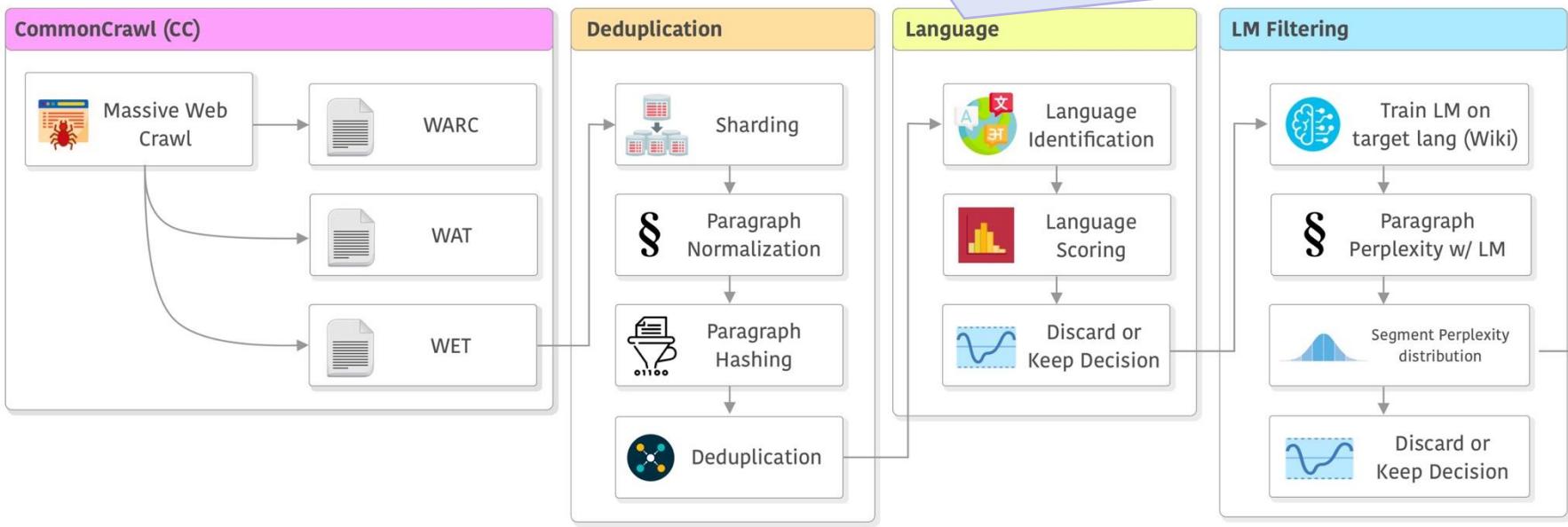
# LLaMA 1's Data Pipeline

Shard WET content into shards of 5GB each (one CC snapshot can have 30TB). Then you normalize paragraphs (lowercasing, numbers as placeholders, etc), compute per-paragraph hashes and then duplicate them.



# LLaMA 1's Data Pipeline

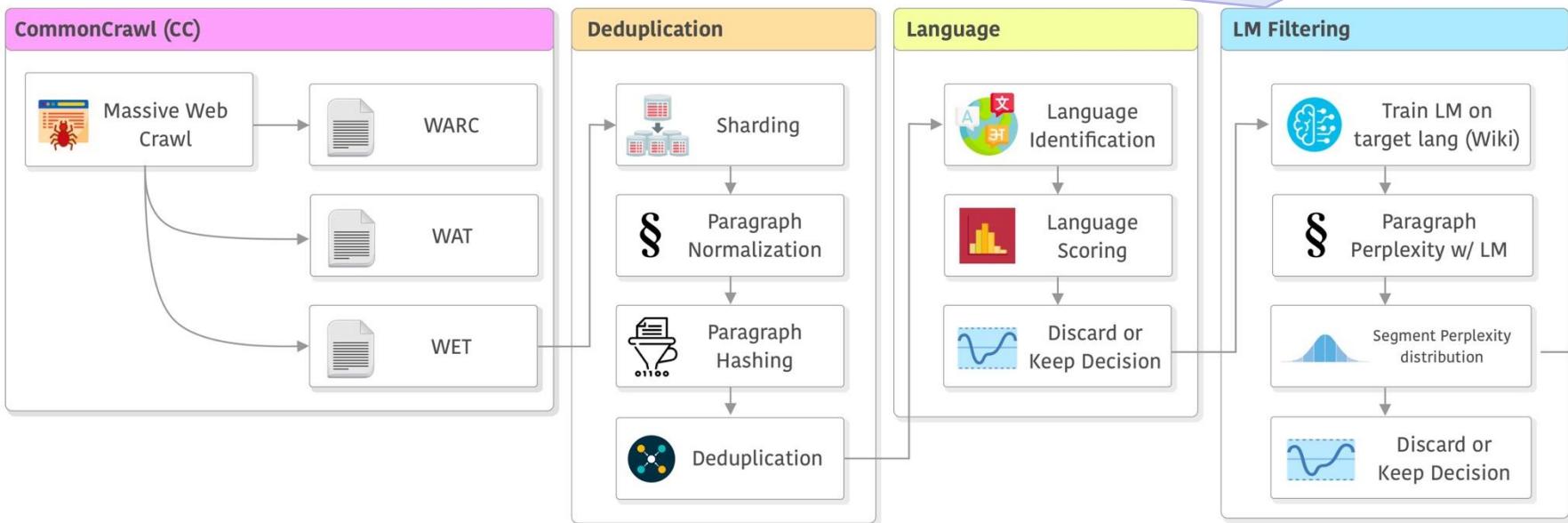
Perform language identification and decide whether to keep or discard languages.  
The order of when you do this in the pipeline can impact the language discrimination quality.



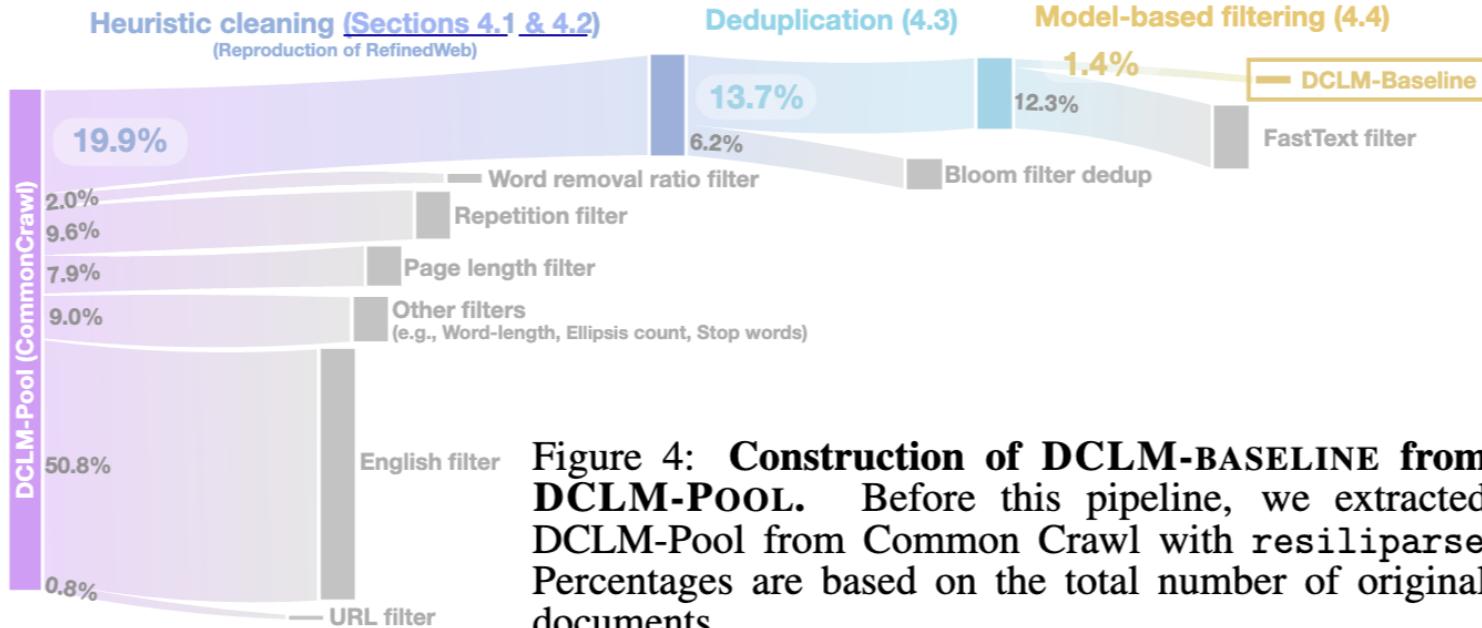
# LLaMA 1's Data Pipeline

Do further quality filtering: Train a simple LM (n-gram) on target languages using Wikipedia, then compute per-paragraph perplexity on the rest of the data:

- Very high PPL: Very different than Wiki and likely low-quality → Drop
- Very low PPL: Very similar or near duplicates to Wiki → Drop



# DataDecomp-LM filtering pipeline



**Figure 4: Construction of DCLM-BASELINE from DCLM-POOL.** Before this pipeline, we extracted DCLM-Pool from Common Crawl with `resiliparse`. Percentages are based on the total number of original documents.

# Few cleaned-up pre-training datasets

Dataset	Example models	Tokens	Source	License	Lang
C4 (Raffel et al. 2020)	T5	165B	CC	ODC-BY	English
The Pile (Gao et al. 2020)	GPT-J, Pythia	300B	22 datasets including CC, books, code, news	Varies by dataset subset	English
RedPejamas (Weber et al. 2024)	Llama 1	1.2T	CC, C4, Github, Arxiv, Books, Wikipedia, StackExchange	Varies by dataset subset	English
RefinedWeb (Penedo et al. 2023)	Falcon	600B	CC	ODC-BY 1.0	English
Dolma (Soldaini et al. 2024)	OLMo	3T	CC, C4, Gutenberg, Github, Wikipedia, Wikibooks	ImpACT MR	English
DataComp-LM (Li et al. 2024)	SmollM2, DCLM	240T	CC	?	English

# The Pile

- Pile-CC: From Common Crawl; uses [justText](#) to extract useful text.
- PubMed Central: 5M NIH funded papers and public.
- arXiv: preprint for research papers since 1991 (uses latex).
- Gutenberg [PG-19](#): Online books (before 2019) with copyright clearance.
- Books3 is a collection of ~200K books. Has been [subject of lawsuits](#).
- StackExachange: Q&A format is close to real applications.
- Github: Content is not just the code.
  - Note, [GH archive](#) has regular snapshots of Github (commits, forks, etc.)

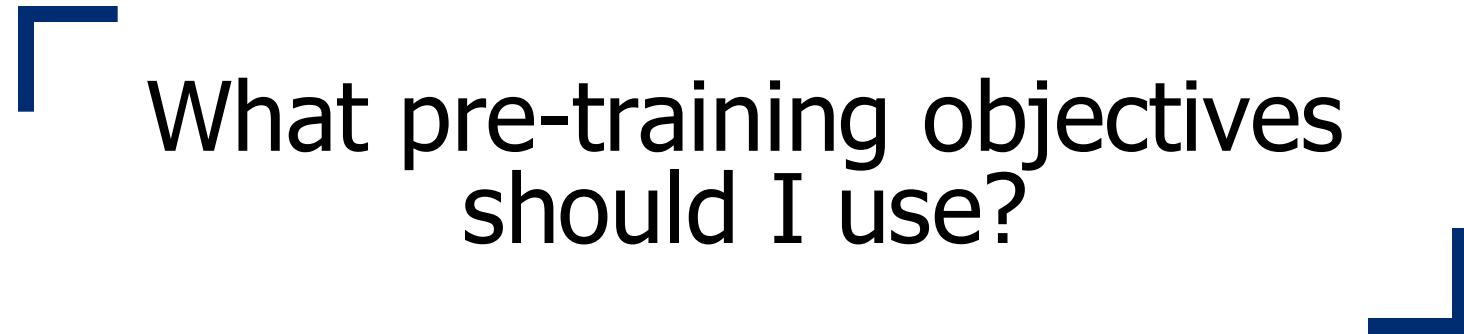
Component	Raw Size
Pile-CC	227.12 GiB
PubMed Central	90.27 GiB
Books3 <sup>†</sup>	100.96 GiB
OpenWebText2	62.77 GiB
ArXiv	56.21 GiB
Github	95.16 GiB
FreeLaw	51.15 GiB
Stack Exchange	32.20 GiB
USPTO Backgrounds	22.90 GiB
PubMed Abstracts	19.26 GiB
Gutenberg (PG-19) <sup>†</sup>	10.88 GiB
OpenSubtitles <sup>†</sup>	12.98 GiB
Wikipedia (en) <sup>†</sup>	6.38 GiB
DM Mathematics <sup>†</sup>	7.75 GiB
Ubuntu IRC	5.52 GiB
BookCorpus2	6.30 GiB
EuroParl <sup>†</sup>	4.59 GiB
HackerNews	3.90 GiB
YoutubeSubtitles	3.73 GiB
PhilPapers	2.38 GiB
NIH ExPorter	1.89 GiB
Enron Emails <sup>†</sup>	0.88 GiB
<b>The Pile</b>	<b>825.18 GiB</b>

# Summary: preparing pre-training data

---

- Data does not fall from the sky. You have to work to get it!
- **Finding large data:** CommonCrawl has a ton of crawled dumps, but not the only one.
- **Cleaning data** can save tons of compute and even give you gains.
- **Repetitions** are often a waste of compute and deteriorate model quality.
- **Scaling deduplication** requires advanced data structures.
- **Old data** old data may skew your model predictions, but it depends on your application.
- **Data mixtures** are quite important, though depend on your downstream application.

# Pre-training language models: The actual training



# What pre-training objectives should I use?

# On Pre-training Objectives

- So far, the dominant objective we have seen is “next-token” prediction.
- In reality any “marginal” observations about language can be a source of supervision.

# Objectives

- Prefix language modeling
  - **Input:** Thank you for inviting
  - **Output:** me to your party last week
- BERT-style denoising
  - **Input:** Thank you <M> <M> me to your party  
apple week
  - **Output:** Thank you for inviting me to your  
party last week
- Deshuffling
  - **Input:** party me for your to. last fun you  
inviting week Thanks.
  - **Output:** Thank you for inviting me to your  
party last week
- IID noise, replace spans
  - **Input:** Thank you <X> me to your party <X> week
  - **Output:** <X> for inviting <Y> last <Z>
- IID noise, drop tokens
  - **Input:** Thank you me to your party week .
  - **Output:** for inviting last

# Objectives: Experiments

- All the variants perform similarly
- “Replace corrupted spans” and “Drop corrupted tokens” are more appealing because target sequences are shorter, speeding up training.

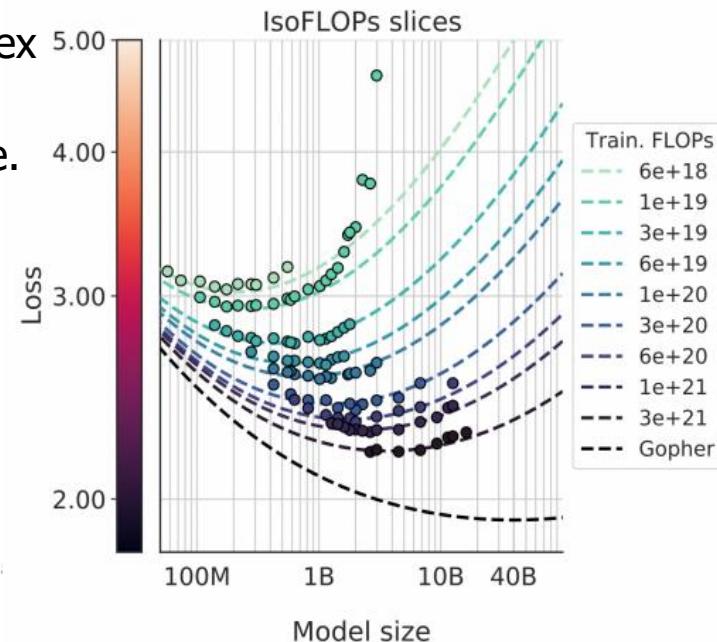
Assuming Enc-Dec architecture.  
Evaluated for classification tasks.

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	<b>26.86</b>	39.73	<b>27.49</b>
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62
BERT-style (Devlin et al., 2018)	82.96	19.17	<b>80.65</b>	69.85	26.78	<b>40.03</b>	27.41
★ Replace corrupted spans	83.28	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	39.82	<b>27.65</b>
Drop corrupted tokens	<b>84.44</b>	<b>19.31</b>	<b>80.52</b>	68.67	<b>27.07</b>	39.76	<b>27.82</b>

How should we select the  
right hyperparams?

# IsoPlots: Tradeoffs at a smaller scale

- The performance of your model depends on a complex combination of many factors.
- Goal: find the best combinations, for a fixed compute.
- It's good to change various parameter (e.g., training data, size, or other hyperparams) and see how it's quality changes.



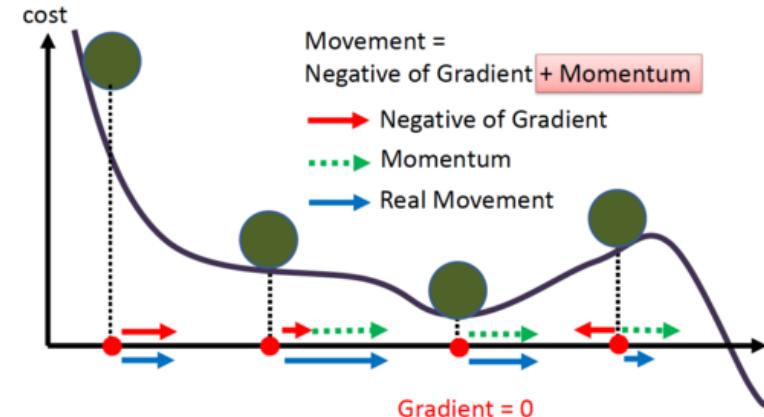


# How should I train the model?



# Optimizers

- Most modern models use “AdamW” optimizer (not vanilla Gradient Descent).
  - Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order “momentums”.
  - “W” because it decouples “weight decay” from “learning rate”. (Details out of scope for us. See the cited paper.)



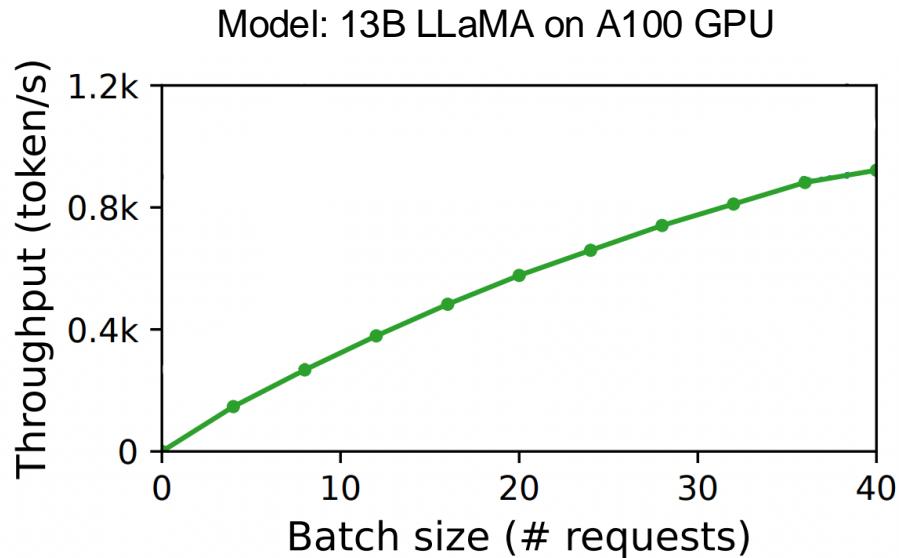
<https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>

<https://pytorch.org/docs/stable/generated/torch.optim.Adam.html>

[Decoupled Weight Decay Regularization, 2017]

# Batching Data

- Previously we talked about the importance of batching data
- GPUs are faster at Tensor operations and hence, we want to do batch processing
- The larger batch of data, the faster they get processed.
- Alas, the speedup is often sub-linear (e.g., 2x larger batch leads to less than 2x speedup).



# Batch sizes: some known statistics

[LLaMA: Open and Efficient Foundation Language Models, 2023](#)

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

[The Llama 3 Herd of Models, 2024](#)

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

**Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training.** See text and Figure 5 for descriptions of each type of parallelism.

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

is set to 1.0. We do not employ the batch size scheduling strategy for it, and it is trained with a constant batch size of 4608 sequences. During pre-training, we set the maximum sequence

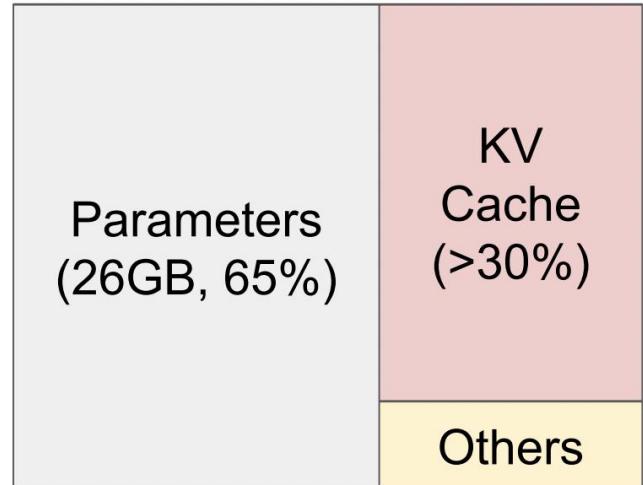
# Can I fit this model in which GPU?

---

- One of the followings:
  - You have a model a model and want to find the right GPU for it.
  - You have a GPU and want to find the largest model to fit in.
- What should we do?
  - The memory taken up by a model depends on:
    - Model parameters
    - Activations: notice that these increase with larger batch and seq length
    - Gradients (of training)

# The Memory Usage

- Here is the memory usage of an NVIDIA A100 when serving (i.e., no training)
  - Model: 13B LLaMA
  - Batch size of 10
- ~65% of your GPU memory is the model parameters that **never change**
- ~32% of your memory are KV tensors that **change for each input.**
  - This KV cache will increase for larger batch sizes.
  - Managing this part of the memory is key for efficient training.



NVIDIA A100 40GB

# How many parameters does my Transformer have?

Bonus

- Let's count the number of parameters:
- The self-attention block params:
  - $3 \times \left(d \times \frac{d}{m}\right) \times m + d^2 = 4d^2$
- The FFN block params:
  - $2 \times (d \times d_{\text{ff}})$
- So, in total:  $4d^2 + 2dd_{\text{ff}}$
- The ratio of SA/FFN parameters is  $\frac{2d}{d_{\text{ff}}}$  and  $d_{\text{ff}}$  is usually 2-4 larger than  $d$ .
- In most models, roughly 2/3 of transformer parameters are feedforward blocks
- Notice that the num of params is independent of seq length ( $n$ ) or batch size ( $b$ )!
  - So, in theory you should be able to run your SA on sequences of any length!
    - (but would it work on longer sequences? -- more on this later)

$$\mathbf{W}_i^q \in \mathbb{R}^{d \times \frac{d}{m}}, \mathbf{W}_i^k \in \mathbb{R}^{d \times \frac{d}{m}}, \mathbf{W}_i^v \in \mathbb{R}^{d \times \frac{d}{m}}, \mathbf{W}^o \in \mathbb{R}^{d \times d}$$
$$\text{head}_i \leftarrow \text{Attention}(\mathbf{x}\mathbf{W}_i^q, \mathbf{x}\mathbf{W}_i^k, \mathbf{x}\mathbf{W}_i^v)$$
$$\mathbf{x} \leftarrow \text{MHAttention}(\mathbf{x}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^o$$

$$\mathbf{x} \leftarrow f(\mathbf{x} W_1 + b_1)W_2 + b_2$$
$$W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}, W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$$

(note, not showing layer-norm and residuals)

$m$ : number of heads

$d$ : feature dimension in output of SA

# 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

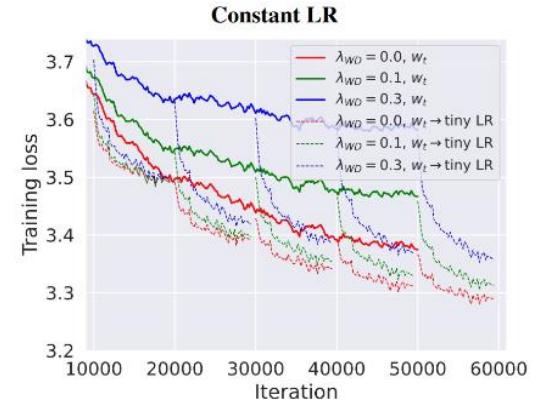
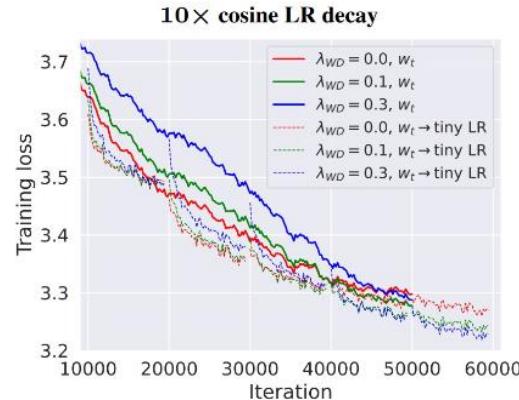
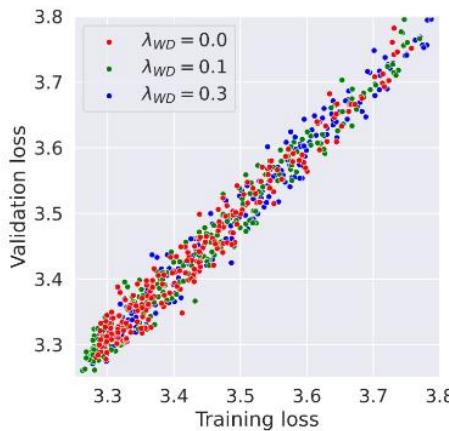
Newer models (except Qwen) rely only on weight decay

\* 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.

[Slide credit: Tatsu Hashimoto]

# Why weight decay LLMs?

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

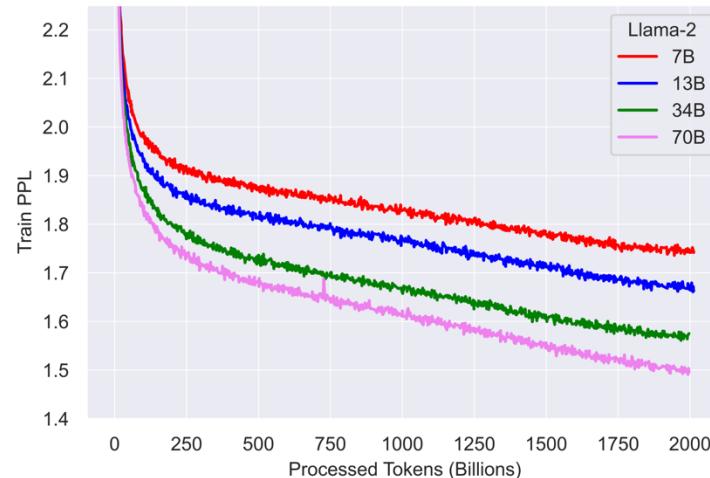
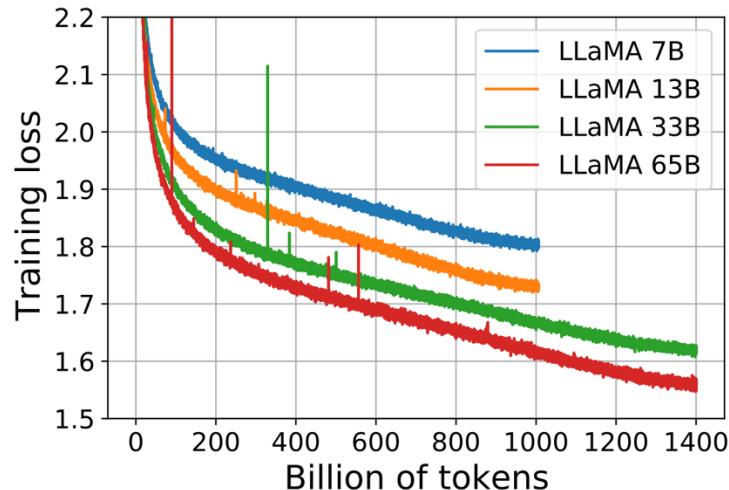


It's not to control overfitting

Weight decay interacts with learning rates (cosine schedule)

# Convergence

- In practice, your model's loss should continue to go down with more training on more data.
- So, the real bottlenecks are:
  - (1) compute
  - (2) data
- Sometimes training diverges (spikes in the loss), at which point practitioners usually restart training from an earlier checkpoint.



# Staged pre-training

- Few models do staged pre-training (e.g., llama3).
  1. Start with pre-training indiscriminative on all sorts of data (including short data).
  2. Do continued pre-training on long text.
  3. Annealing (learning rate going to zero)

# Recap of training LLMs

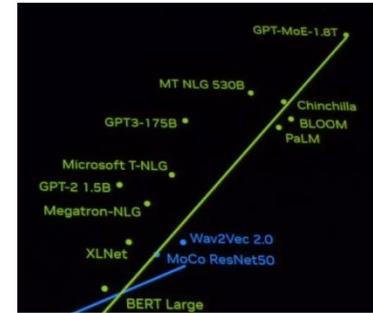
---

- **IsoPlots:** for a fixed compute, which combination of parameters give you the best bang for the buck.
- Careful batching makes your training go brrr!
- Memory usage can be tricky since there are various moving parts.
  - More on distributed training later on.
- Dropout is less common but you still ‘regularize’ LMs via large-scale training.

# Mixture of Experts (MoE)

Slide credit to Tatsu Hashimoto and Samet Oymak  
for earlier versions of these slides.

# Mixture of Experts (MoE)



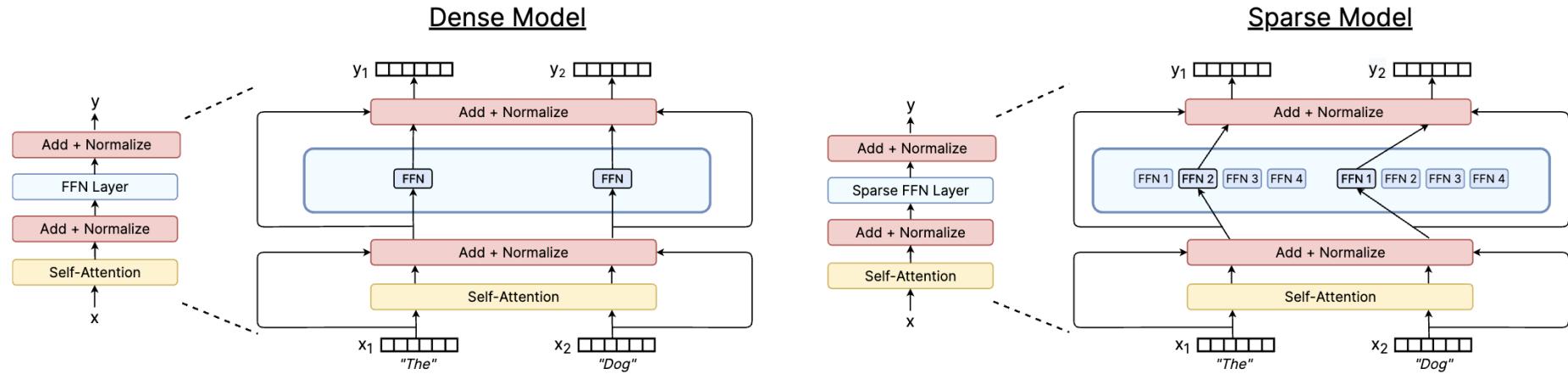
GPT4 (?)



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

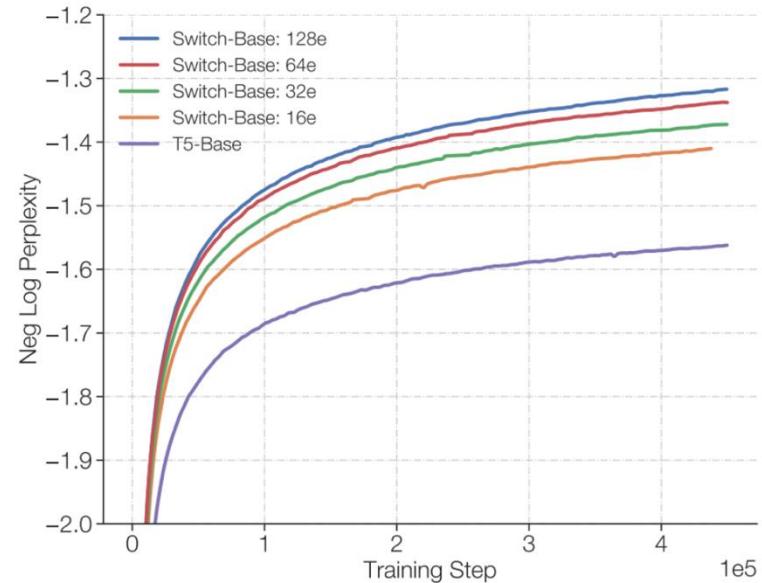
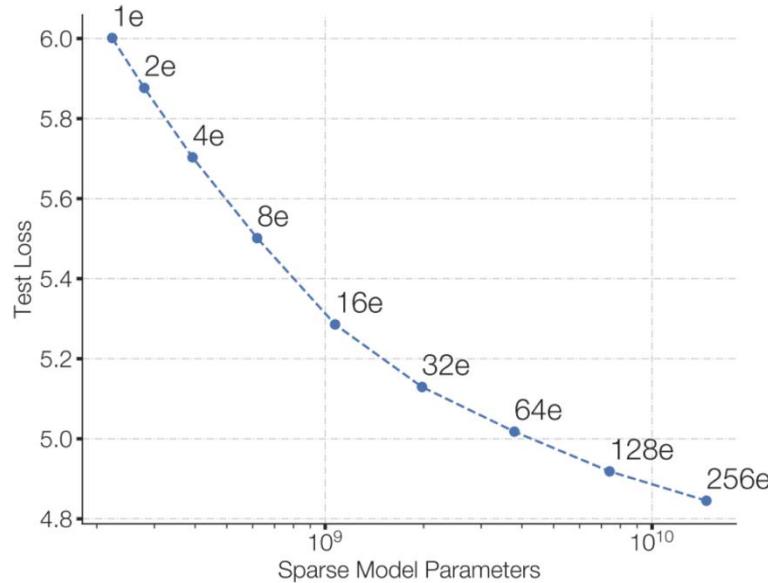
# Mixture of Experts (MoE)

- Two main elements (NNs):
  - **Sparse MoE layer:** Instead of using the dense FFN, sparse FFNs are used.
  - **A gate networking/router:** It determines which tokens are sent to which experts.
- You can increase the # experts without affecting FLOPs



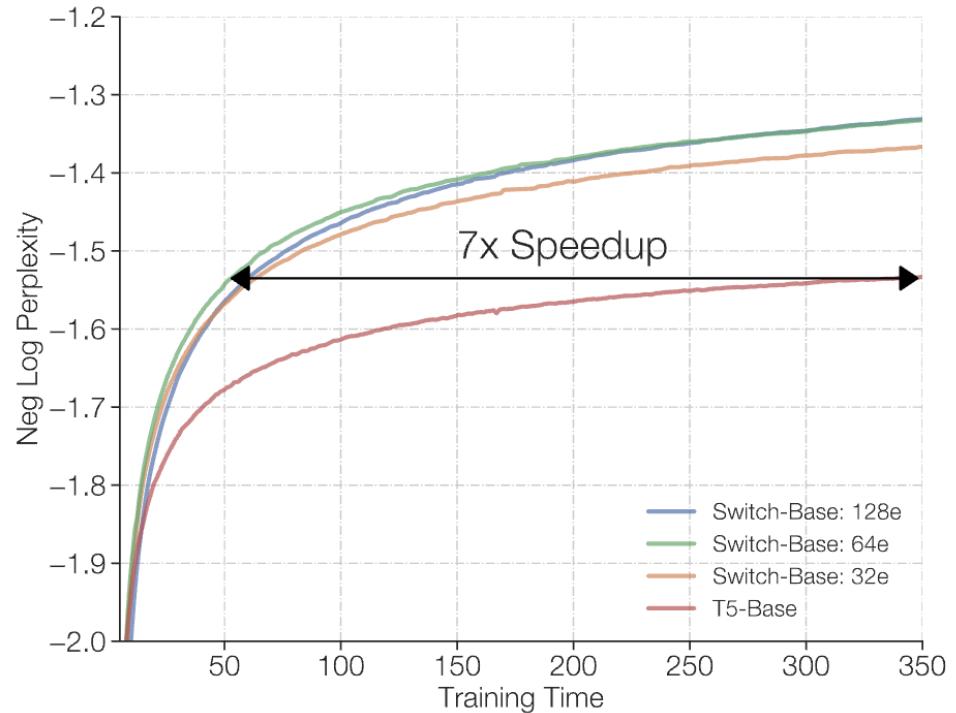
# Why are MoE's getting popular?

- Same FLOP, more param does better



# Why are MoE's getting popular?

- Faster training over a dense (non-MOE) model



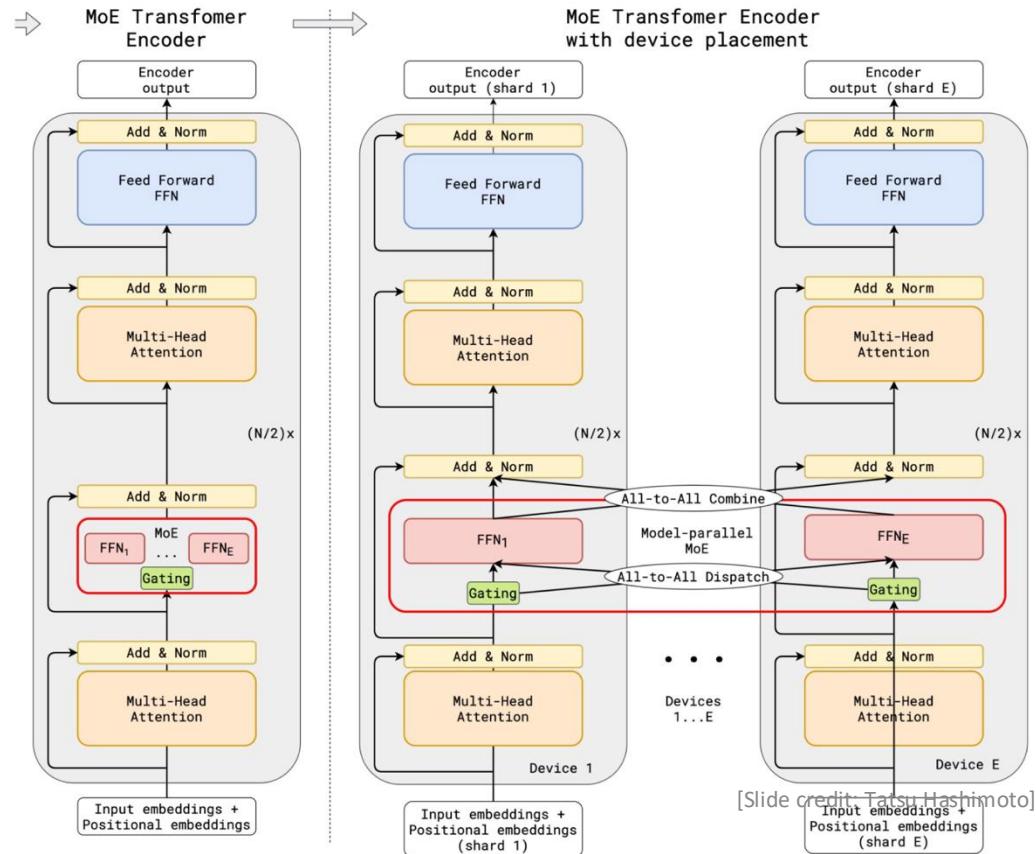
# **Why are MoE's getting popular?**

---

- Have faster inference compared to the dense models of the same size model

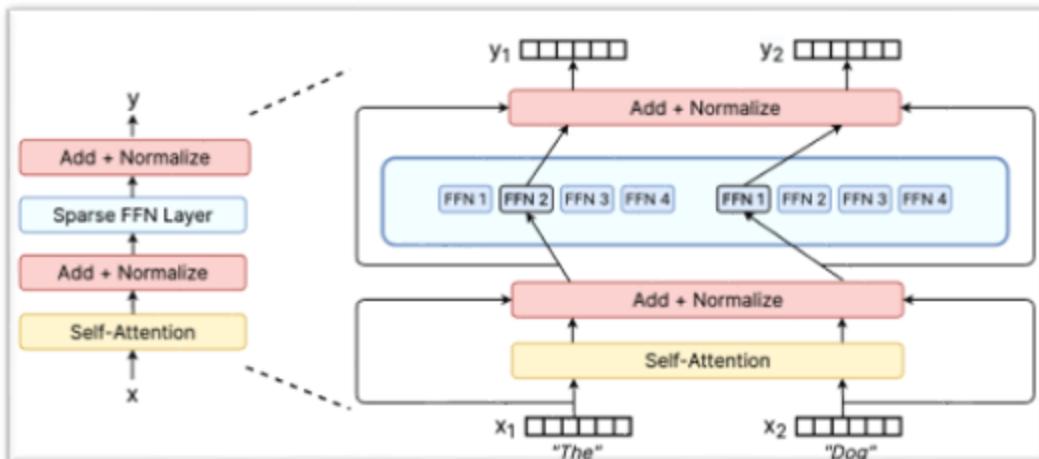
# Why are MoE's getting popular?

- Parallelizable to many devices (more on this in a bit)
- MoEs parallelize nicely since each FFN (expert) can fit in a device.

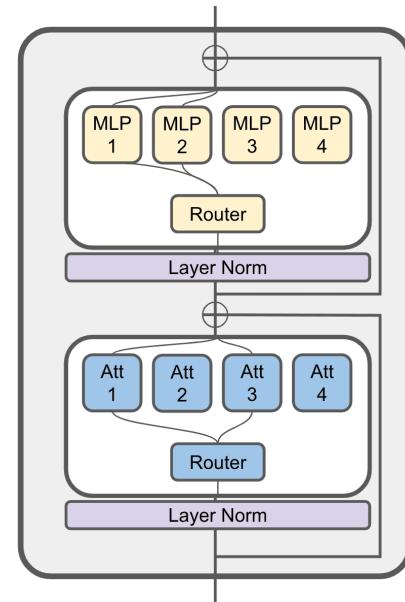


# MoE variants

Typical: replace MLP with MoE layer



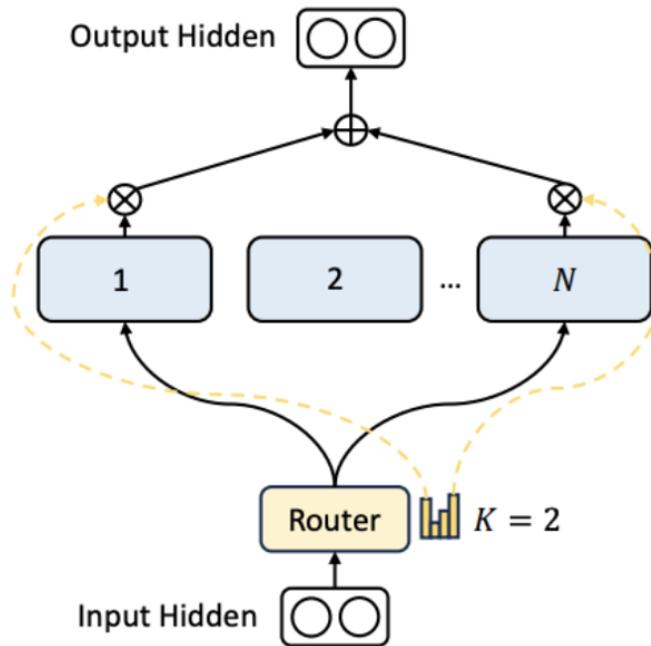
Less common: MoE for attention heads



[ModuleFormer, JetMoE]

# Top-k routing, intuitively

- Most models use the class top-k routing which involves 3 steps:
  - **(1) Scoring:** Produces a distribution over the experts.
  - **(2) Routing:** identify the set of top-k experts and assign their scores:
  - **(3) weighted sum among top-k:** creates weighted average of experts summed with the residuals.



# Top-k routing, in detail

This is how DeepSeek and Grok implement MoE layer.

- Most models use the class top-k routing which involves 3 steps:
  - **(1) Scoring:** Suppose the input feature (the input to MoE layer) is  $x$ . The gates are selected by a logistic regression (i.e., linear scoring + softmax) which produces a distribution over the experts.

$$s = \text{Softmax}(xW_r) \text{ where } W_r \text{ are the trainable params}$$

- **(2) Routing:** identify the set of top-k experts and assign their scores:

$$g_i = \begin{cases} s_i & s_i \in \text{TopK}(\{s_j \mid 1 \leq j \leq N\}, K) \\ 0 & \text{o.w.} \end{cases}$$

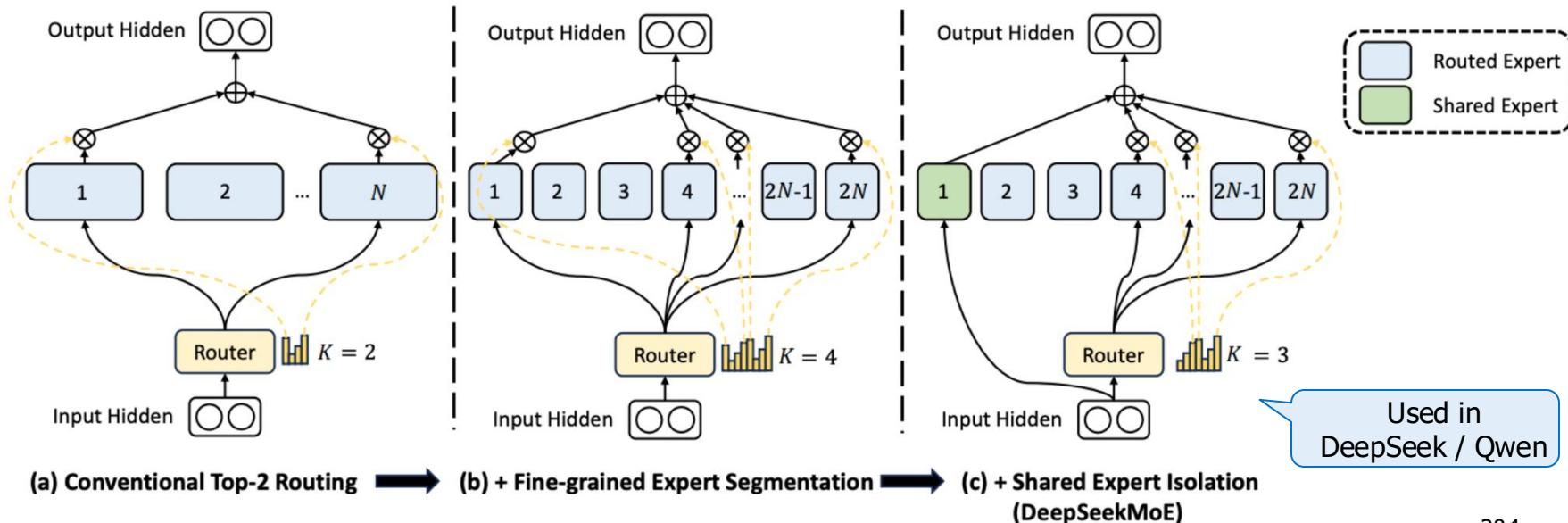
- **(3) weighted sum among top-k:**

$$y = \sum_i g_i \text{FFN}_i(x) + x$$

Mixtral and DBRX  
softmax after the TopK

# Recent variations: shared experts

- Smaller, larger number of experts + a few shared experts that are always on.
- The idea is to have induce more complementarity among experts, by having a shared expert that takes the care of easy/common skills.



# Various ablations from the DeepSeek paper

- More experts, shared experts all seem to generally help

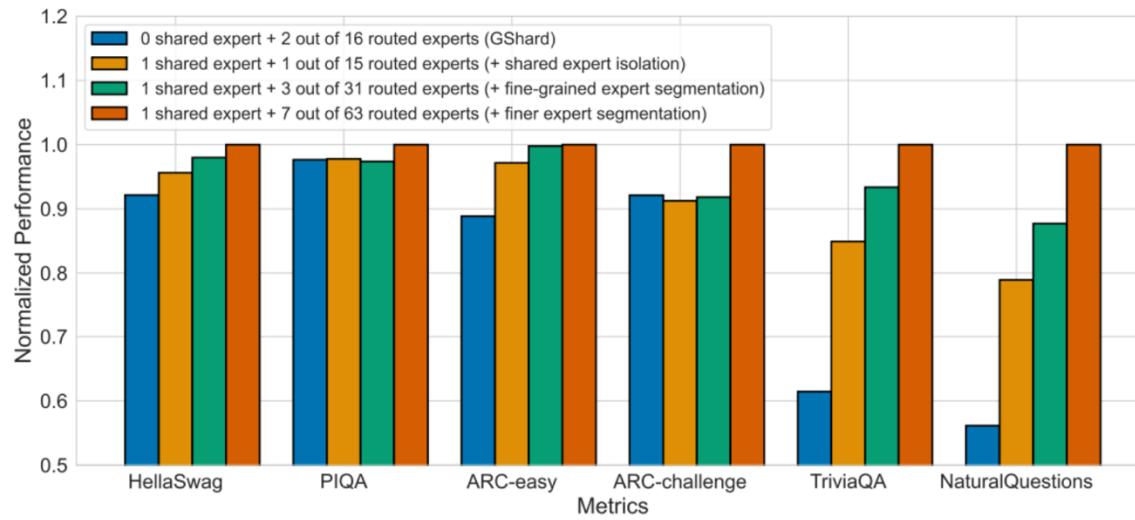


Figure 3 | Ablation studies for DeepSeekMoE. The performance is normalized by the best performance for clarity in presentation. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute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

# Why haven't MoEs been more popular?

- Infrastructure is complex / advantages on multi node.

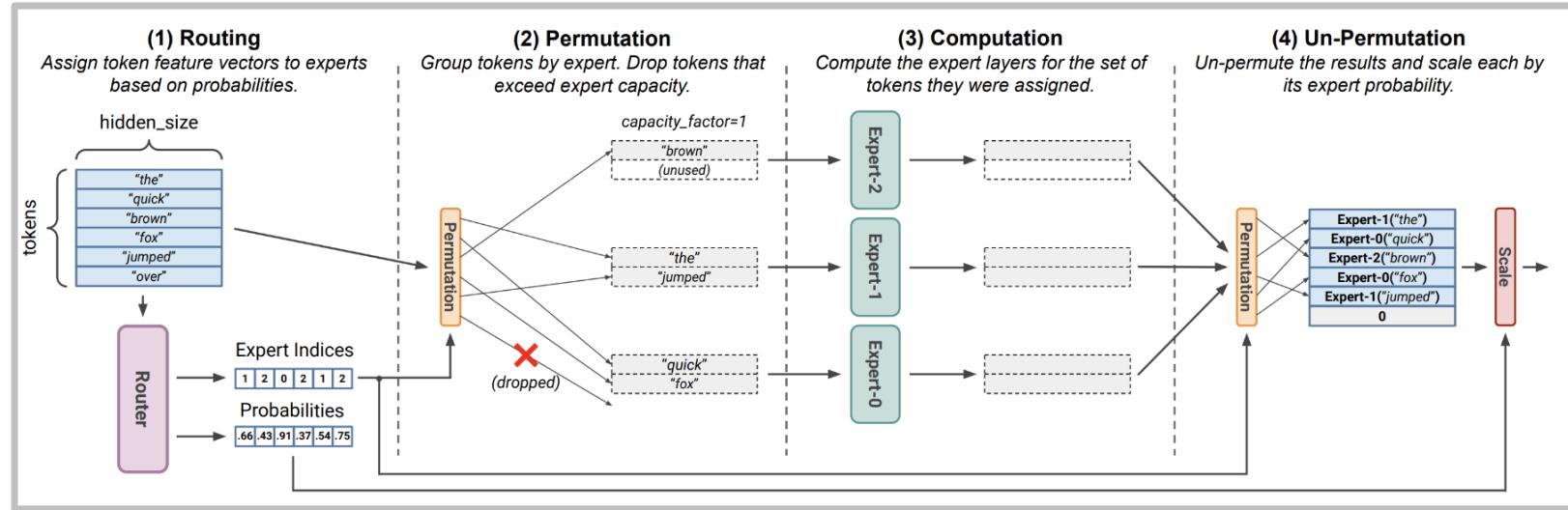
At a high level, sparsity is good when you have many accelerators (e.g. GPU/TPU) to host all the additional parameters that comes when using sparsity. Typically models are trained using data-parallelism where different machines will get different slices of the training/inference data. The machines used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

# Why haven't MoEs been more popular?

- **Training stability:** Because of the discrete nature of MoE's decisions, small changes in router weights can have disproportionate effect in the outcomes.
  - One solution is adding stochasticity during training to encourage exploration.
- **Redundancy and hybridity:** There is a tendency for multiple experts to converge in learning similar information. This dilutes the specialization of experts and results in overlapping knowledge domains and inefficient use of parameters.
  - One solution is using shared experts (used by DeepSeek).
- **Load balancing:** The imbalance calls to few popular experts makes MoE inefficient. During training, the gating network may converge to few experts which may continue to self-reinforce as favored experts are trained quicker and hence selected more.
  - One common solution is using an auxiliary loss to encourage giving all experts equal importance.
- **Complex infrastructure:** Often you need a lot of GPU memory to fit your model and run it efficiently.
  - A lot to discuss on this but beyond the scope of our class.

# Side issue – stochasticity of MoE models

- There was speculation that GPT-4's stochasticity was due to MoE.
- Why would a MoE have additional randomness?



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

[Slide credit: Tatsu Hashimoto]

# Summary

---

- MoEs take advantage of sparsity – not all inputs need the full model
- Discrete routing is hard, but top-k heuristics seem to work
- Lots of empirical evidence now that MoEs work, and are cost-effective

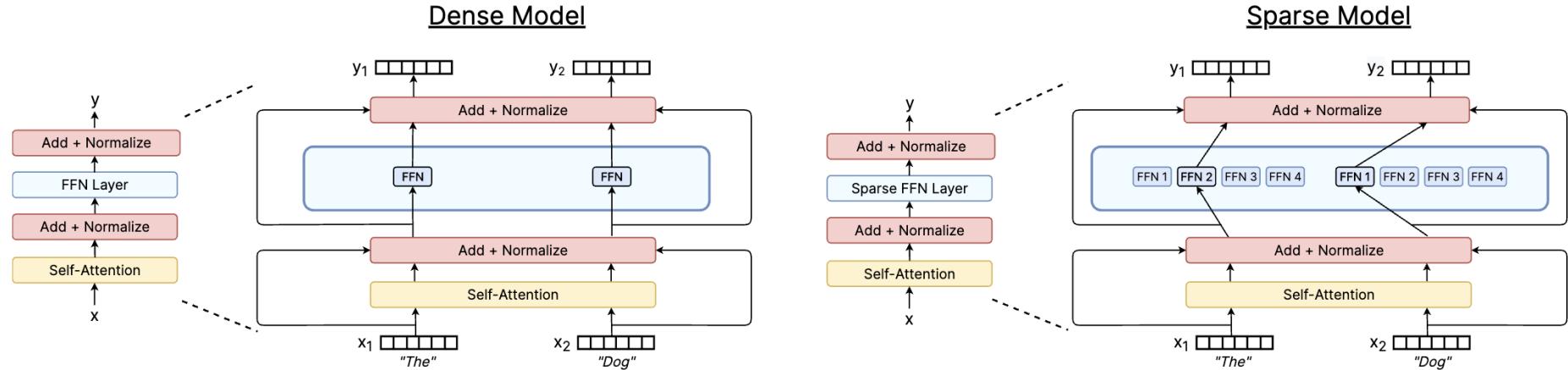


# Bonus content on MoE



# Mixture of Experts (MoE)

- Two main elements (NNs):
  - **Sparse MoE layer:** Instead of using the dense FFN, sparse FFNs are used.
  - **A gate networking/router:** It determines which tokens are sent to which experts.
- You can increase the # experts without affecting FLOPs



# MoE variants

- Routing function
- Expert sizes
- Training objectives

# Variations of routing function

- **Observation:** choosing experts based on the input usually entails a discrete selection (i.e. which expert to use), which complicates backprop relying on differentiability.
- The pioneering work of Shazeer et al. 2017 formulated routed function that was adopted and adapted by many follow-on works. Here is how it worked:
  1. Top- $k$  routing function which takes as an input a token representation  $x$ ,
  2. Then routes it to the top- $k$  experts out of the set  $N$  experts.

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_j^N e^{h(x)_j}}. \quad h(x) = W_r \cdot x \quad \text{trainable variable } W_r$$

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x). \quad \text{denote the set of selected top-}k \text{ expert indices as } \mathcal{T}.$$

# Routing function

- Many of the routing algorithms boil down to “choose top k”

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

Each token chooses top-k expert

Tokens							
				T1	T2	T3	
				E1	<b>Choose Top-K</b>		
				E2	0.51	-0.25	1.58
				E3	-1.32	1.97	0.1
				E4	2.25	2.61	0.02
				E5	-2.81	-0.68	-0.41

Each expert chooses top-k token

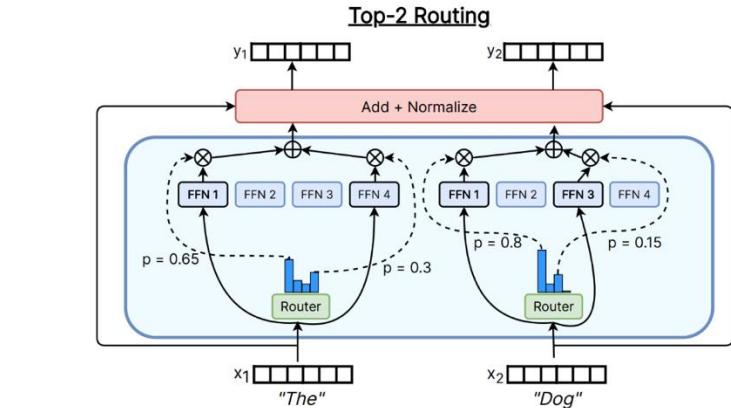
Tokens							
				T1	T2	T3	
				E1	3.13	0.14	0.74
				E2	0.51	-0.25	1.58
				E3	-1.32	1.97	0.1
				E4	2.25	2.61	0.02
				E5	-2.81	-0.68	-0.41

Global routing tokens should go to which experts

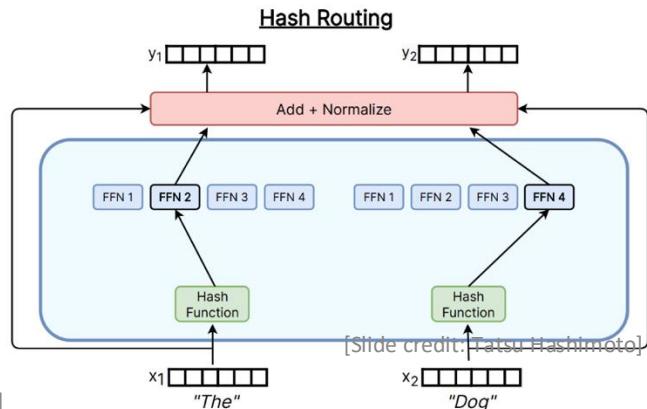
# Common routing variants

- Used in most MoEs
  - Switch Transformer ( $k=1$ )
  - Gshard ( $k=2$ ), Grok (2),
  - Mixtral (2), Qwen (4),
  - DBRX (4), DeepSeek (7)
- Common baseline

Top-k

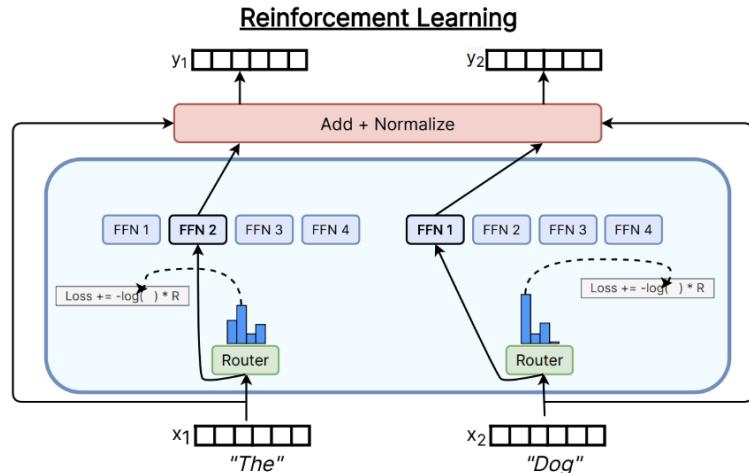


Hashing

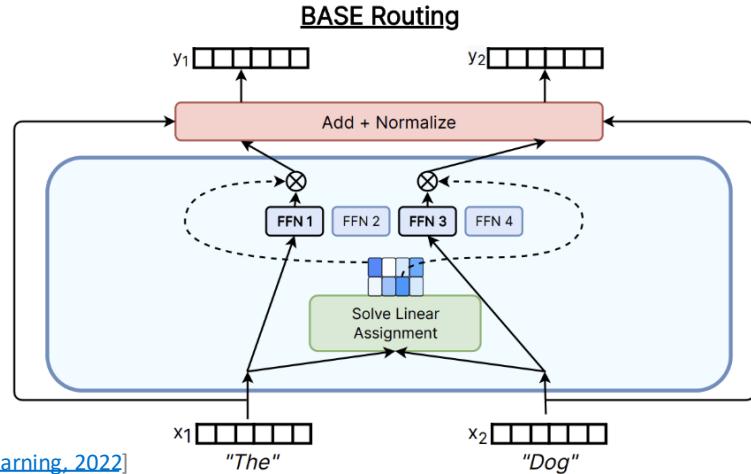


# Other routing variants

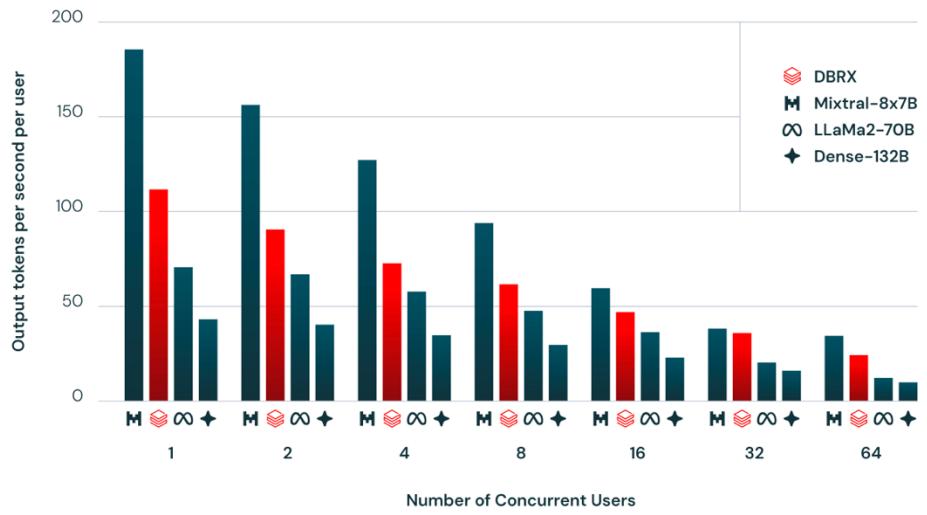
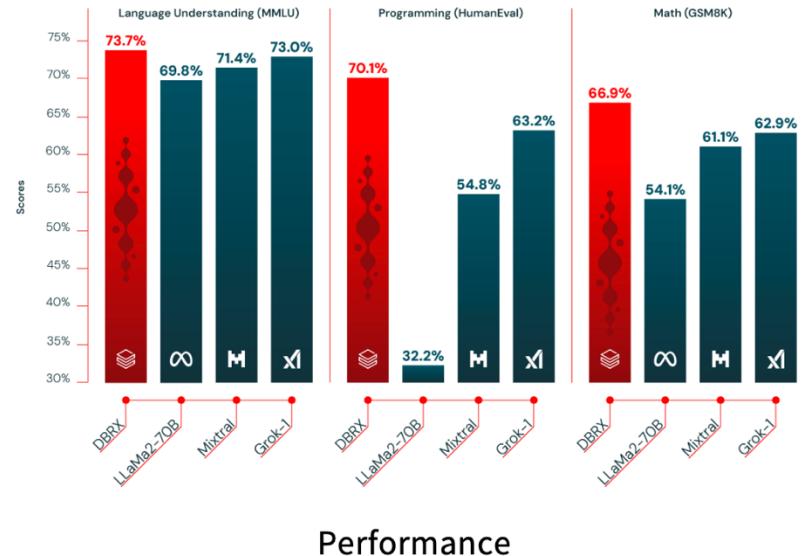
- RL to learn routes
  - Used in some of the earliest work Bengio 2013, not common now



- Solve a matching problem
  - Linear assignment for routing
  - Used in various papers like Clark '22



# Some recent MoE results



- MoEs are most of the highest-performance open models and are quite quick.

# Some recent MoE results – Qwen

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

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

# Some recent MoE results – DeepSeek

- There's also some good recent ablation work on MoEs showing they're generally good.

Metric	# Shot	Dense	Hash Layer	Switch
# Total Params	N/A	0.2B	2.0B	2.0B
# Activated Params	N/A	0.2B	0.2B	0.2B
FLOPs per 2K Tokens	N/A	2.9T	2.9T	2.9T
# Training Tokens	N/A	100B	100B	100B
Pile (Loss)	N/A	2.060	1.932	1.881
HellaSwag (Acc.)	0-shot	38.8	46.2	49.1
PIQA (Acc.)	0-shot	66.8	68.4	70.5
ARC-easy (Acc.)	0-shot	41.0	45.3	45.9
ARC-challenge (Acc.)	0-shot	26.0	28.2	30.2
RACE-middle (Acc.)	5-shot	38.8	38.8	43.6
RACE-high (Acc.)	5-shot	29.0	30.0	30.9
HumanEval (Pass@1)	0-shot	0.0	1.2	2.4
MBPP (Pass@1)	3-shot	0.2	0.6	0.4
TriviaQA (EM)	5-shot	4.9	6.5	8.9
NaturalQuestions (EM)	5-shot	1.4	[14]	2.5

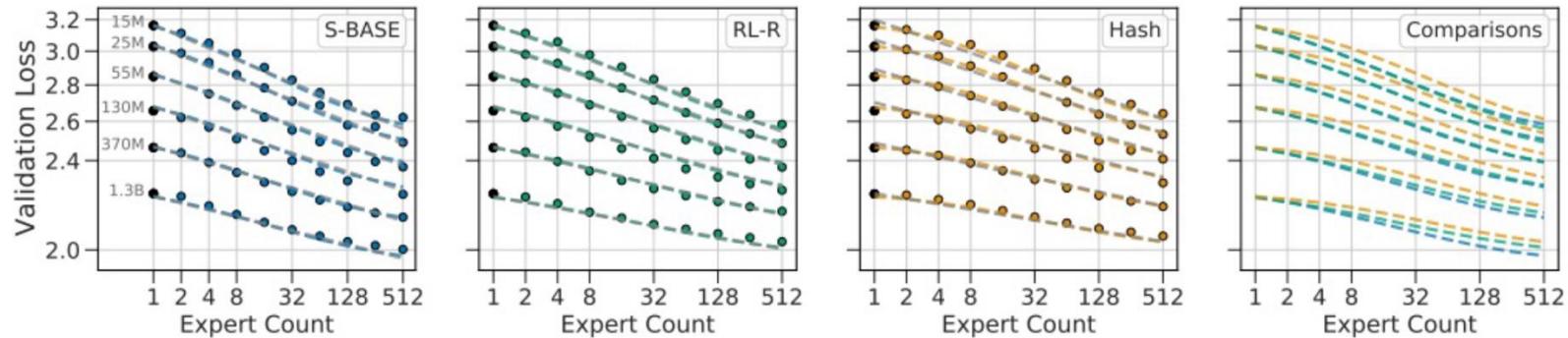
# How do we train MoEs?

- Major challenge: we need sparsity for training-time efficiency...
  - But sparse gating decisions are not differentiable!
- Solutions?
  1. Reinforcement learning to optimize gating policies
  2. Stochastic perturbations
  3. Heuristic 'balancing' losses.

Guess which one people use in practice?

# How do we train MoEs?

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



(REINFORCE baseline approach, Clark et al 2020)

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

# Stochastic approximation

- From Shazeer et al 2017 – routing decisions are stochastic with gaussian perturbations.
  - This naturally leads to experts that are a bit more robust.
  - The softmax means that the model learns how to rank K experts

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))$$

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

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

# Stochastic approximation

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

```
router_weights = mtf.Variable(shape=[d_model, num_experts])

# router_logits shape: [num_cores, tokens_per_core, numExperts]
router_logits = mtf.einsum([inputs, router_weights], reduced_dim=d_model)

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

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

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

# Load balancing losses

- A key issue regarding systems efficiency: using the experts evenly.
- Define an auxiliary loss and add it the total model loss during training.

Given  $N$  experts indexed by  $i = 1$  to  $N$  and a batch  $\mathcal{B}$  with  $T$  tokens, the auxiliary loss is computed as the scaled dot-product between vectors  $f$  and  $P$ ,

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

So if an expert gets triggered or get assigned higher probability, downweight their share

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

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

and  $P_i$  is the fraction of the router probability allocated for expert  $i$ ,<sup>2</sup>

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

[Slide credit: Tatsu Hashimoto]

# Recent Extensions of Load Balancing

- Per-expert balancing – same as the switch transformer

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

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

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

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

$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^D f'_i P'_i, \quad (15)$$

$$f'_i = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j, \quad (16)$$

$$P'_i = \sum_{j \in \mathcal{E}_i} P_j, \quad (17)$$

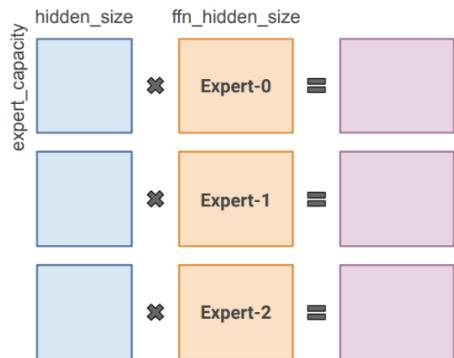
[Slide credit: Tatsu Hashimoto]

# Training MoEs – the systems side

- MoE routing allows for parallelism, but also some complexities
- Modern libraries like MegaBlocks (used in many open MoEs) use smarter sparse MMs

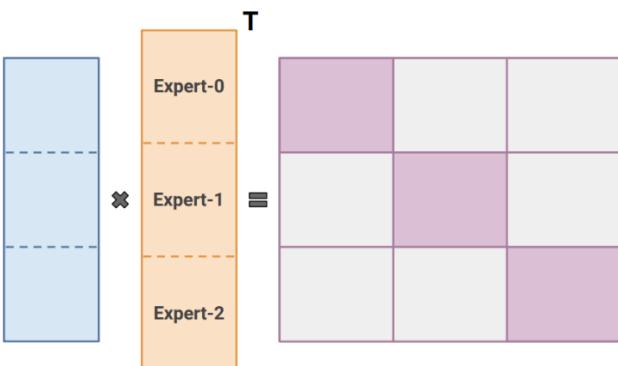
## (A) Batched Matrix Multiplication

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



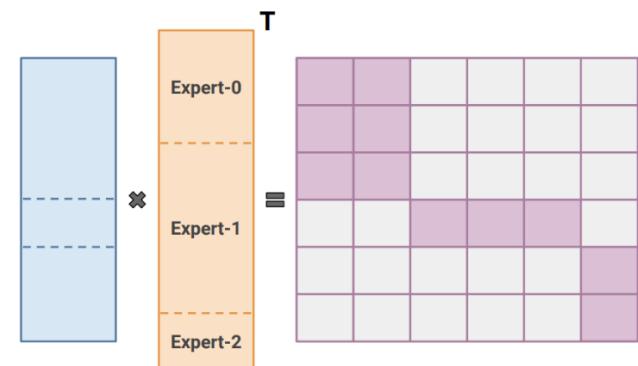
## (B) Block Diagonal Matrix Multiplication

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



## (C) Block Sparse Matrix Multiplication

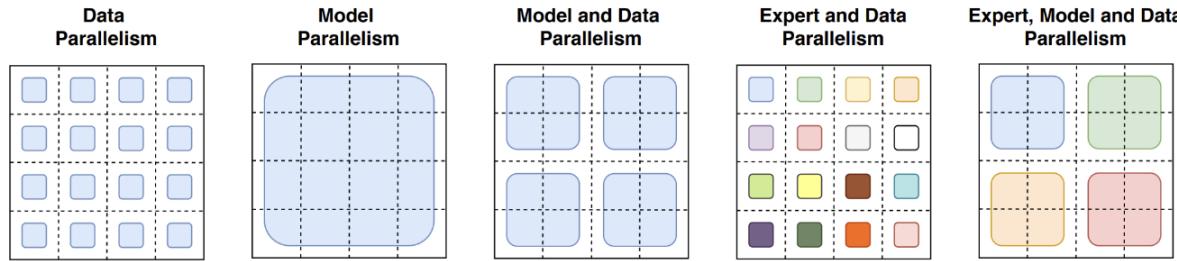
We can enable load imbalanced routing and variable sized experts by expressing expert computation as block sparse matrix multiplication.



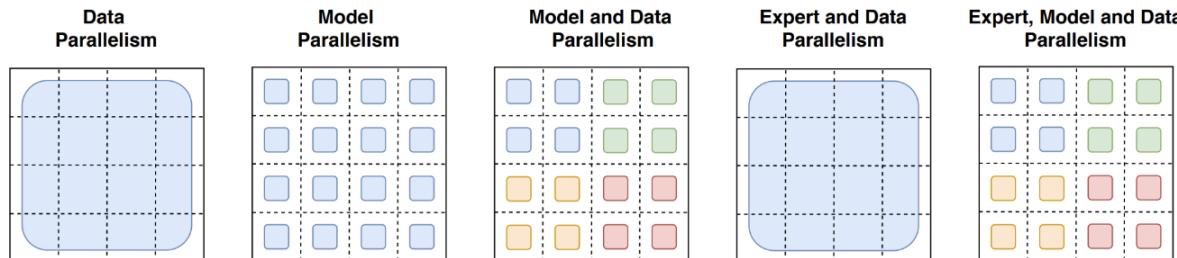
# Training MoEs – the systems side

- Enables additional kinds of parallelism

How the *model weights* are split over cores

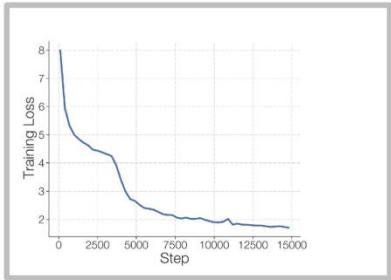


How the *data* is split over cores



[Slide credit: Tatsu Hashimoto]

# Side issue – stability



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

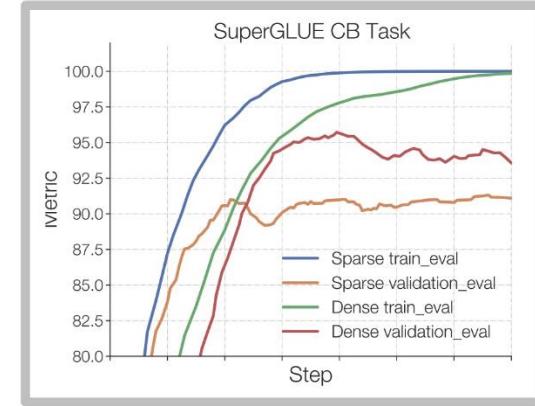
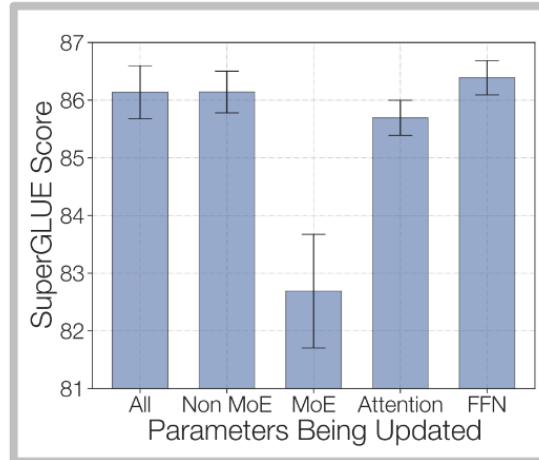
[Zoph et al 2022]

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

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

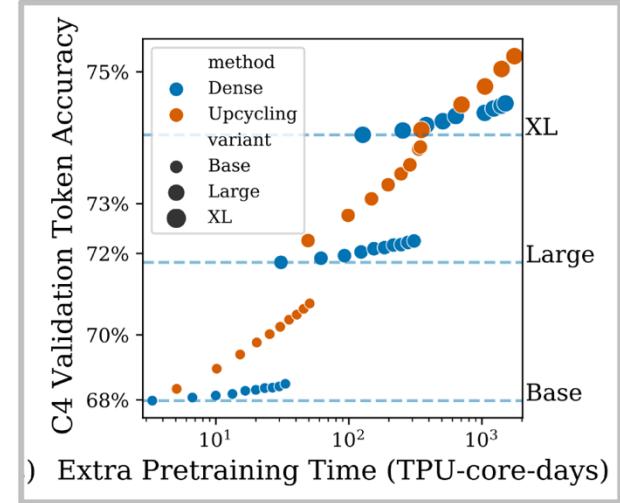
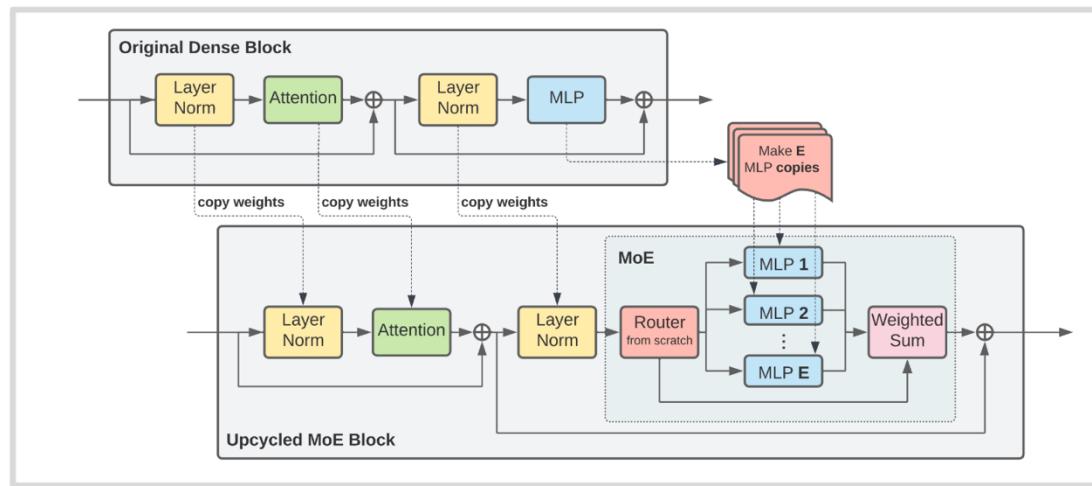
# Issues with MoEs — fine-tuning

- Sparse MoEs can overfit on smaller fine-tuning data
- Zoph et al solution – finetune non-MoE MLPs
- DeepSeek solution – use lots of data 1.4M SFT



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

# Other training methods — Upcycling



- Can we use a pre-trained LM to initialize a MoE?

# Upcycling example - MiniCPM

- Uses the MiniCPM model (topk=2, 8 experts,  $\sim 4B$  active params).

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

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

- Simple MoE, shows gains from the base model with  $\sim 520B$  tokens for training

# Upcycling example – Qwen MoE

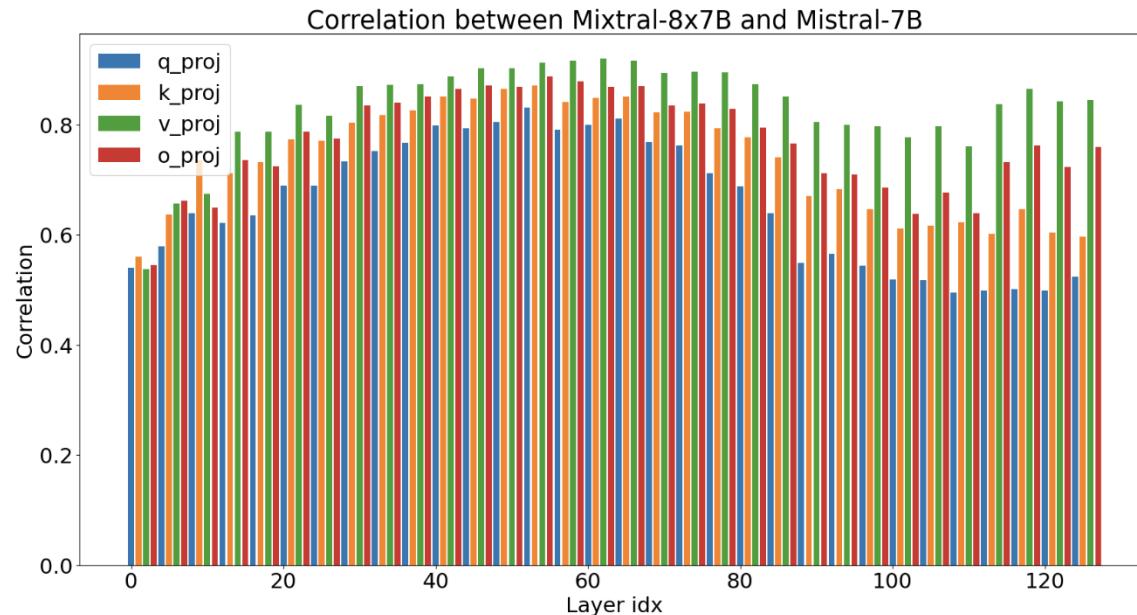
- Qwen MoE – Initialized from the Qwen 1.8B model top-k=4, 60 experts w/ 4 shared.

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

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

# Upcycling example (?) Mixtral

- Some people think Mixtral may also be upcycled



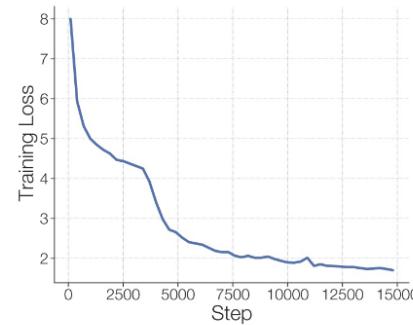
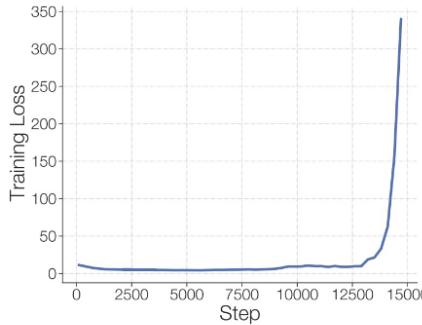
- but since Mixtral is only open weights (no open training code) we don't really know ..

[Source: [https://twitter.com/tianle\\_cai/status/1734188749117153684](https://twitter.com/tianle_cai/status/1734188749117153684)] [Slide credit: Tatsu Hashimoto]

# Why haven't MoEs been more popular?

- Training objectives are somewhat heuristic (and sometimes unstable):

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



[Zoph et al 2022]