

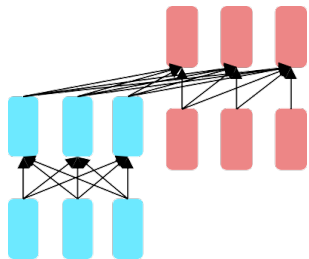
Transformer Language Models

CSCI 601-771 (NLP: Self-Supervised Models)

After Transformer ...

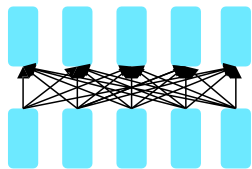


Topology variations



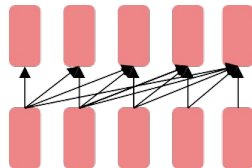
**Encoder-
Decoders**

Encodes an input sequence and generates an output sequence autoregressively, with the decoder also attending to the encoder's representations.



Encoders

Builds contextual representations of input tokens by allowing each token to attend to all input positions.



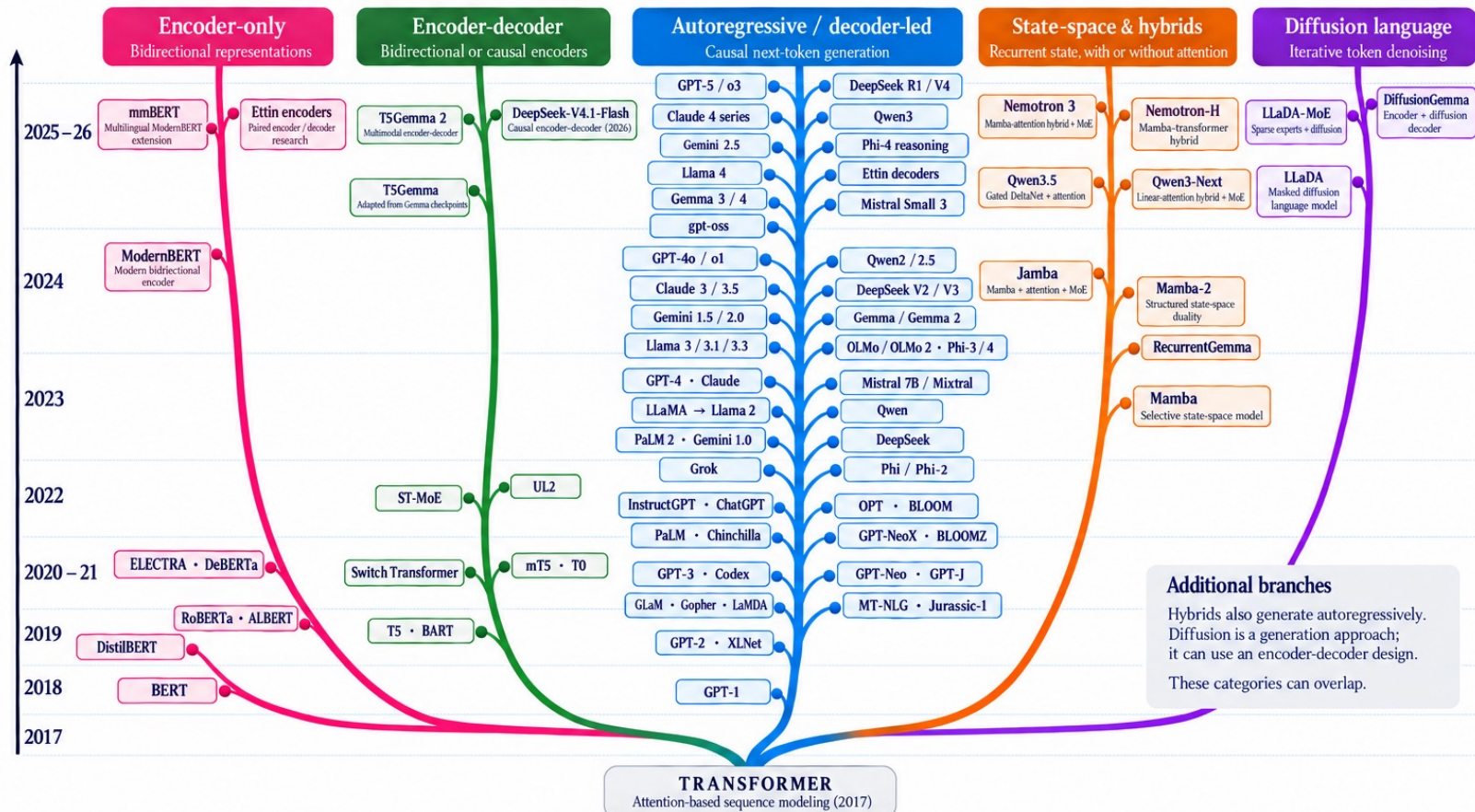
Decoders

Generates tokens autoregressively, with each position attending only to itself and earlier positions.

The Evolution of Language Models

Updated October 2026

Selected milestones • 2017 – 2026



Other architectural variations

- **Topology:** Decoder-only, Encoder-decoder, Encoder-only.
- **Sequence mixing in each layer:**
 - Example: MHA, MQA, GQA, etc.
 - Patterns: global, sliding-window, or interleaved local/global.
- **Positional encodings:** e.g., RoPE.
- **Feedforward layers:** Dense vs. MoE, Activations: GELU, SwiGLU, etc.
- **Normalization and block layout:** pre-norm vs. post-norm.
- **Dimensions:** Depth vs. width, head count and head dimension.
- **Tokenization:** Vocab size and algorithm (e.g., BPE), tokenizer-free models
- **Output scheme:** Next-token prediction, multi-token prediction, Diffusion LMs

Legend: **Already covered them;** **will cover next;** **will hopefully cover at some point.**



Normalization



Quiz: Pre-norm vs Post-norm

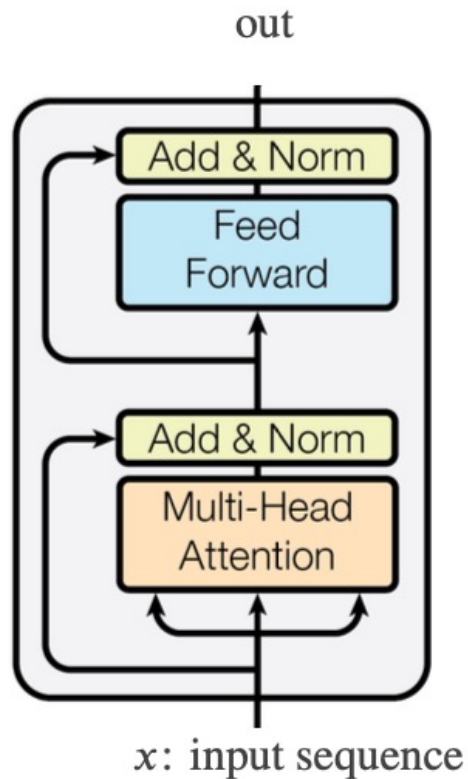
- Which is the original implementation?
- Which one gives gradients a direct identity path?

Post-norm

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

Pre-norm

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



Pre-norm vs Post-norm

- Pre-norm (right) is set up so that LayerNorm does not disrupt the residual stream (in gray)
- Both are valid architectures, but their optimization behavior differs.
- In practice, however, **Pre-norm is preferred over Post-norm.**

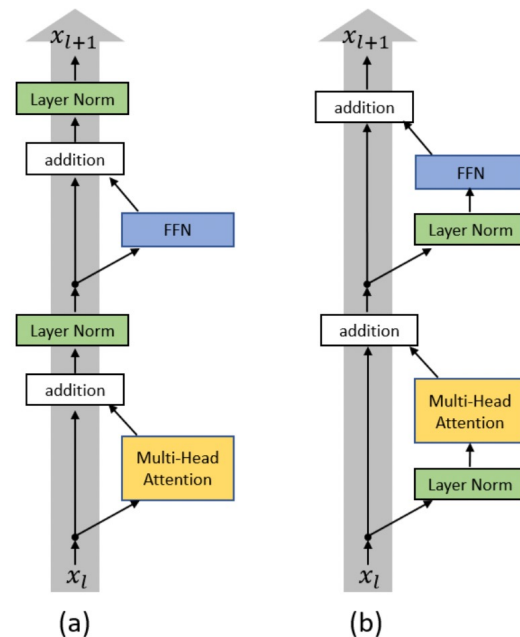
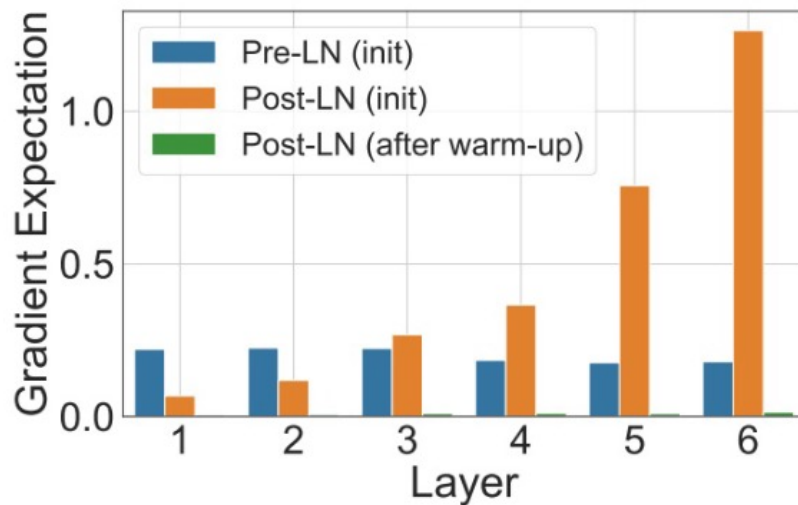


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

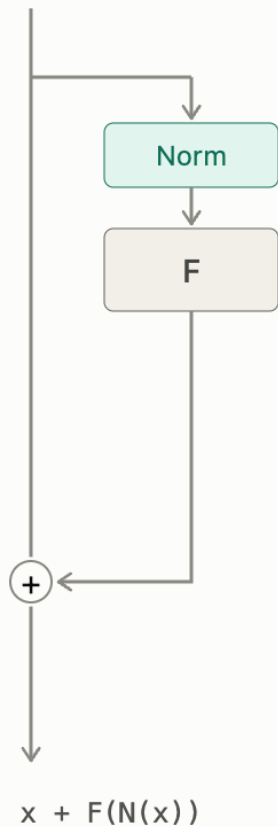
Pre-norm vs Post-norm: Empirical Result

More balanced gradients: Post-norm has large gradients near the output at initialization; pre-norm keeps them more uniform across layers.

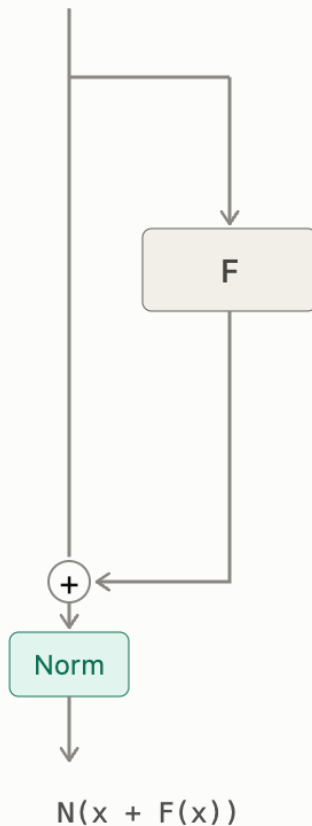


Other forms of LayerNorm

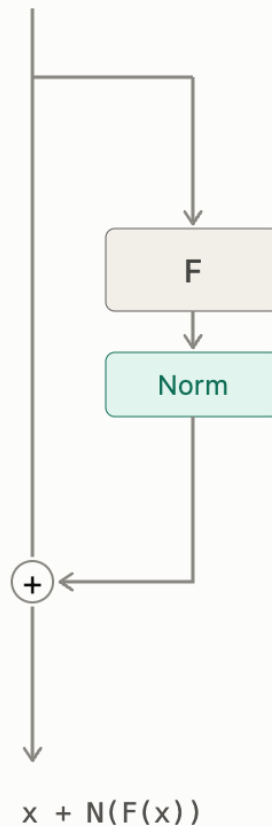
Pre-norm



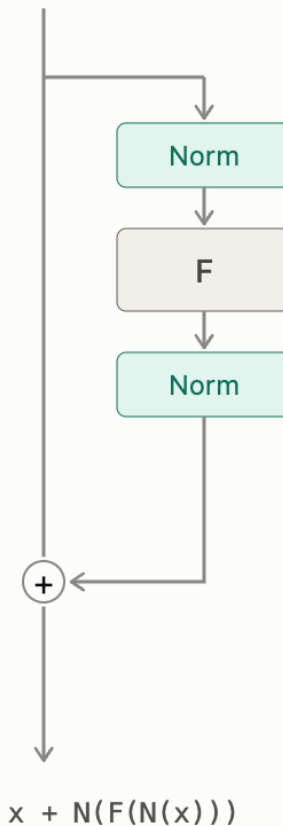
Post-norm



Output-norm



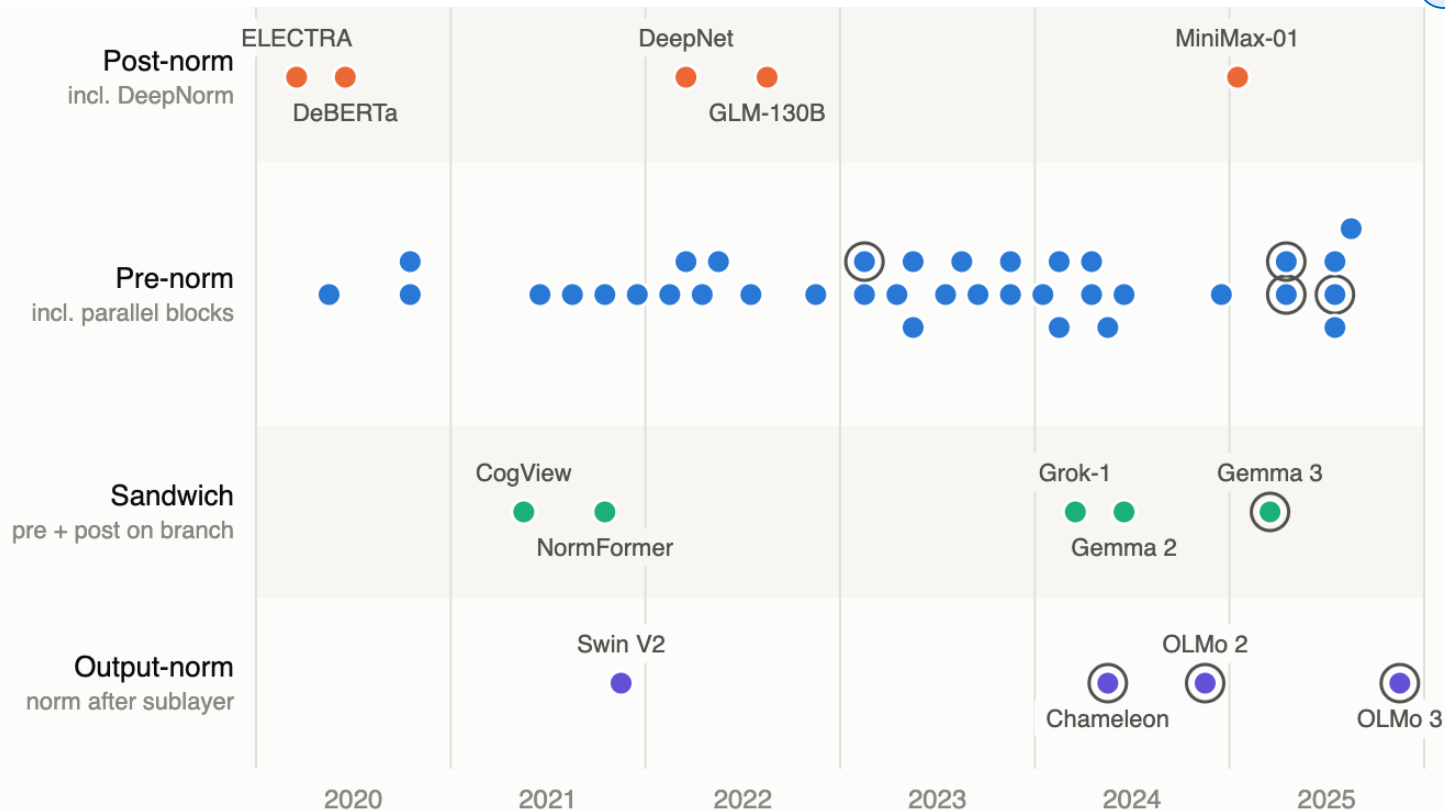
Sandwich



Who uses what?

● Post-norm ● Pre-norm ● Sandwich ● Output-norm ○ Ring = QK-norm

QK-norm also normalizes q and k tensors.



LayerNorm vs. RMSNorm

- Original transformer: **LayerNorm**
 - LayerNorm both centers activations and controls their scale.
 - What if we keep scale control **but skip centering**?

Stats computed
across **each hidden
dimensions**

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu(x)}{\sqrt{\text{Var}(x) + \epsilon}} + \beta.$$

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

$$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d} \sum_i x_i^2 + \epsilon}}.$$

Pulse Check: Normalization (1)

- How many learnable parameters do each normalization have?

- Q1:** LayerNorm:

- A1:** 2d (weight γ and bias β)

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu(x)}{\sqrt{\text{Var}(x) + \epsilon}} + \beta.$$

- Q2:** RMS Norm:

- A2:** d (weight γ)

$$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d} \sum_i x_i^2 + \epsilon}}.$$

Pulse Check: Normalization (2)

- How many FLOPS do each normalization have?

- **Q1:** LayerNorm:

- **A1:** 7d

- Variance: 3d (why?)
- Numerator subtract: d
- Element-wise scaling and division by denominator: 2d
- Sum bias term: d

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu(x)}{\sqrt{\text{Var}(x) + \epsilon}} + \beta.$$

- **Q2:** RMS Norm:

- **A2:** 4d

- Squaring each element (d)
- Summing squared elements (d)
- Element-wise gamma (d)
- Element-wise division (d)

$$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\frac{1}{d} \sum_i x_i^2 + \epsilon}}.$$

Why RMSNorm?

- **Modern explanation** – it's faster (and just as good).
 - **Fewer operations** (no mean calculation)
 - **Fewer parameters** (no bias term to store)
- But does floating point operations really determine speed? No!!
 - Non-matmul ops only make up 0.2% of FLOPS (in a specific study).
 - But GPUs achieve far lower throughput on non-matmul ops!!
 - (e.g., they may be memory-bound because you need to read/write)

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, performance) gains have been seen in papers.

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

approximately 5% higher throughput.

Performance slightly better.

[Slide credit: Tatsu Hashimoto]

Who uses what?

RMSNorm

no mean, no bias

mT5

Gopher

Chinchilla

LLaMA

Llama 2

Qwen

Mistral 7B

Yi

DeepSeek LLM

Mixtral

Gemma

Grok-1

Llama 3

Phi-3

DeepSeek-V2

Chameleon

Qwen2

Gemma 2

OLMo 2

DeepSeek-V3

MiniMax-01

Gemma 3

Qwen3

Llama 4

GLM-4.5

Kimi K2

SmolLM3

gpt-oss

OLMo 3

Trinity Large

Kimi K2.5

Step 3.5 Flash

Qwen3.5

GLM-5

MiniMax M2.5

Nanbeige 4.1

Ling 2.5

Sarvam 105B

Gemma 4

DeepSeek V4

LayerNorm

incl. bias-free

ELECTRA

GPT-3

DeBERTa

ViT

CogView

GPT-J

Jurassic-1

MT-NLG

NormFormer

Swin V2

GPT-NeoX-20B

DeepNet

PaLM

OPT

BLOOM

GLM-130B

Galactica

ViT-22B

Pythia

Falcon

MPT

Tiny Aya

2020

2021

2022

2023

2024

2025

2026 H1

LayerNorms: Recap

- Basically, most recent works use 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 (and less memory traffic), which saves on wall-clock time.



FFN layers



$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + \mathbf{b}_2$$

Is the Bias Term Necessary?

- FFN: The same nonlinear transformation independently to **each** token.
- Most modern transformers don't have bias terms.
 - Original Transformer:

$$\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2 \quad \begin{array}{l} \mathbf{W}_1 \in \mathbb{R}^{d \times d_{\text{ff}}}, \\ \mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d} \end{array}$$

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

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

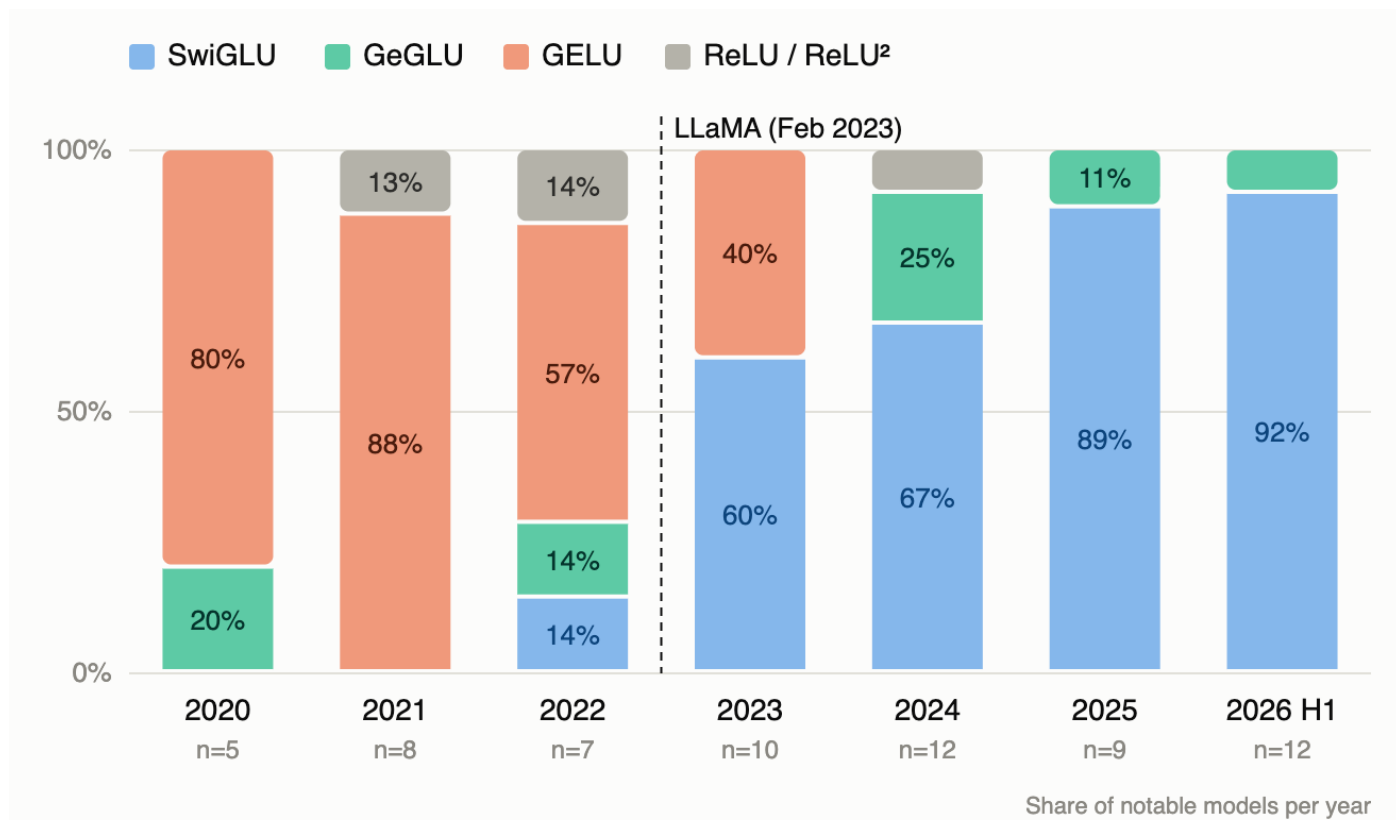
- Reasons:
 - Removing biases saves $d + d_{\text{ff}}$ params (a small reduction rel to total of $2d \times d_{\text{ff}}$)
 - Less memory (similar to RMSNorm).
 - PaLM paper specifically notes dropping it improves training stability.

Who uses bias term?

	With FFN bias	No FFN bias	...
2017	Transformer	—	
2018	GPT-1 BERT	—	
2019	GPT-2	T5	
2020	GPT-3	—	
2021	GPT-J	—	
2022	OPT BLOOM GPT-NeoX-20B	PaLM	
2023	Phi-2	LLaMA Llama 2 Falcon Mistral 7B Qwen	
2024	—	Llama 3 Gemma Mixtral OLMo DeepSeek-V3	
2025	gpt-oss	Llama 4 Qwen3 Gemma 3 Kimi K2 GLM-4.5	
2026	—	Kimi K2.5 Qwen3.5 GLM-5 MiniMax M2.5 Gemma 4	

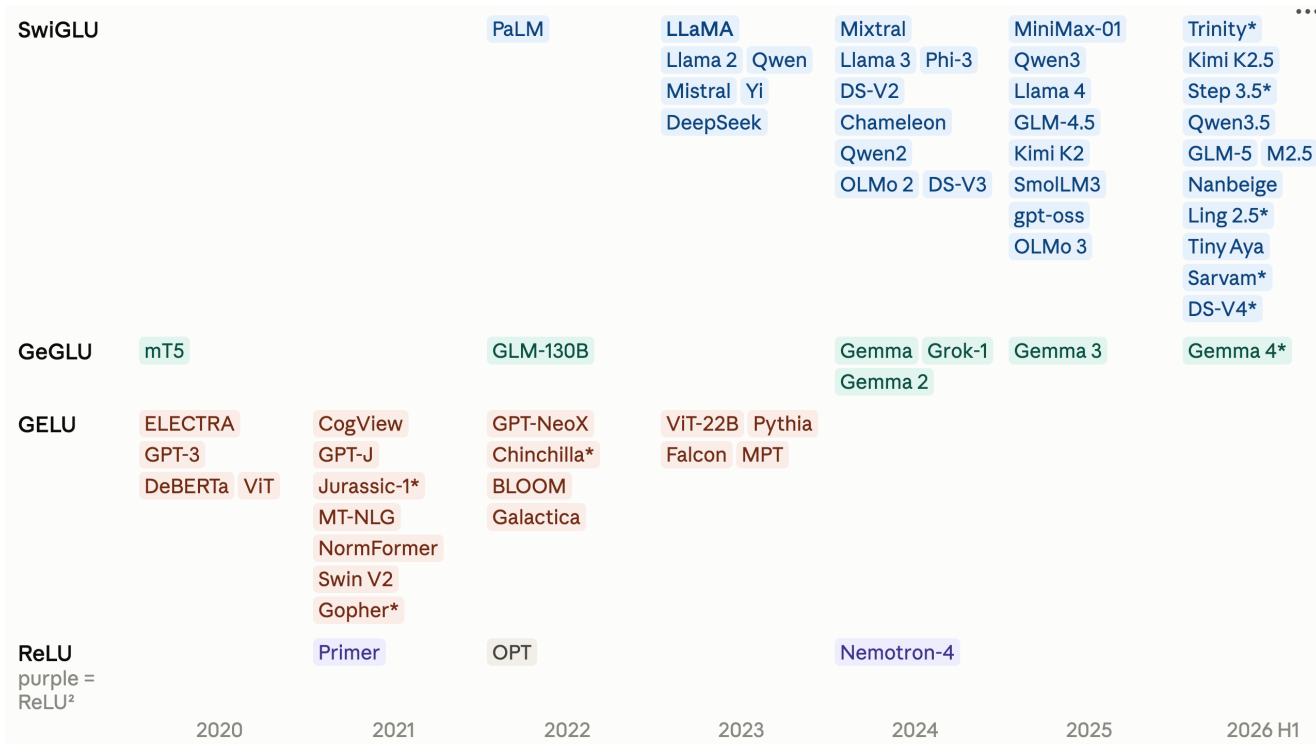
Which Activation fun should we use?

- There are many choices; but trending toward “gated” functions.



Which Activation fun should we use?

- There are many choices; but trending toward “gated” functions.



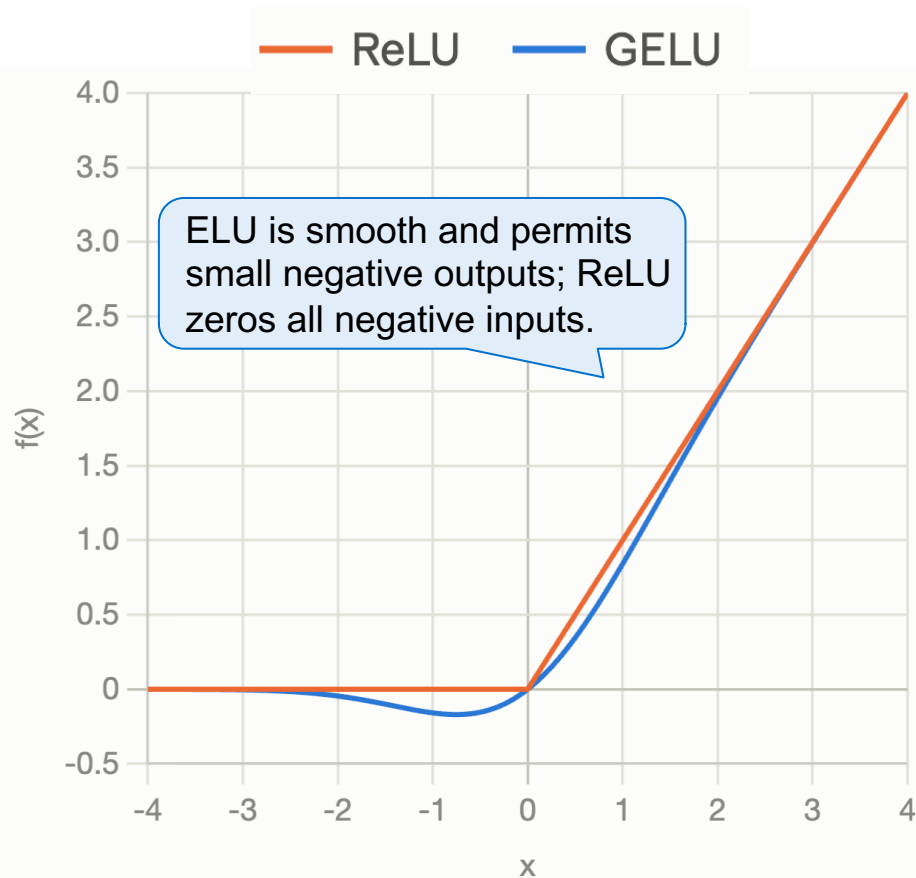
Activations: ReLU vs GELU

- **ReLU:**

- **Notable models:**
Original transformer,
T5, Chinchilla

- **GELU:**

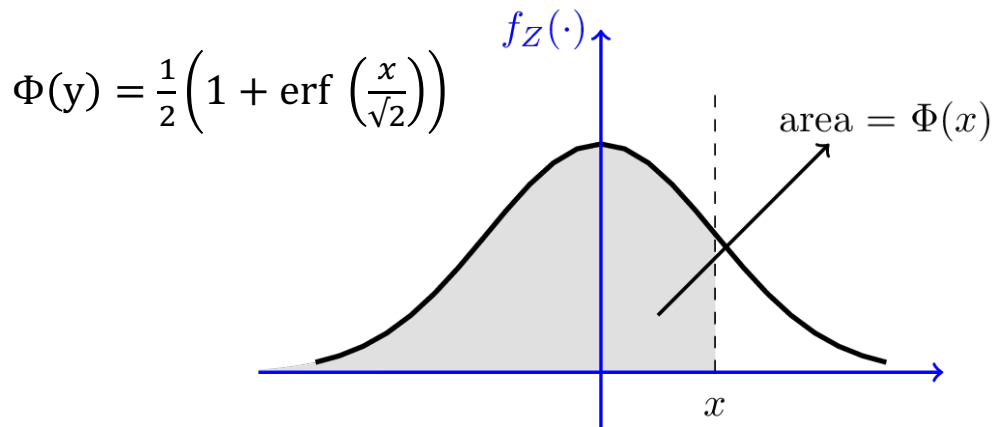
- **Notable models:**
GPT1/2/3, BERT,
RoBERTa.



GELU function, in details

$$\text{GELU}(\mathbf{y}) := \mathbf{y}\Phi(\mathbf{y})$$

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



Gated activations (*GLU)

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

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

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

- What if the first layer depended on two different views of the input?

$$f(\mathbf{x}\mathbf{W}_1) \rightarrow f(\mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V})$$

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

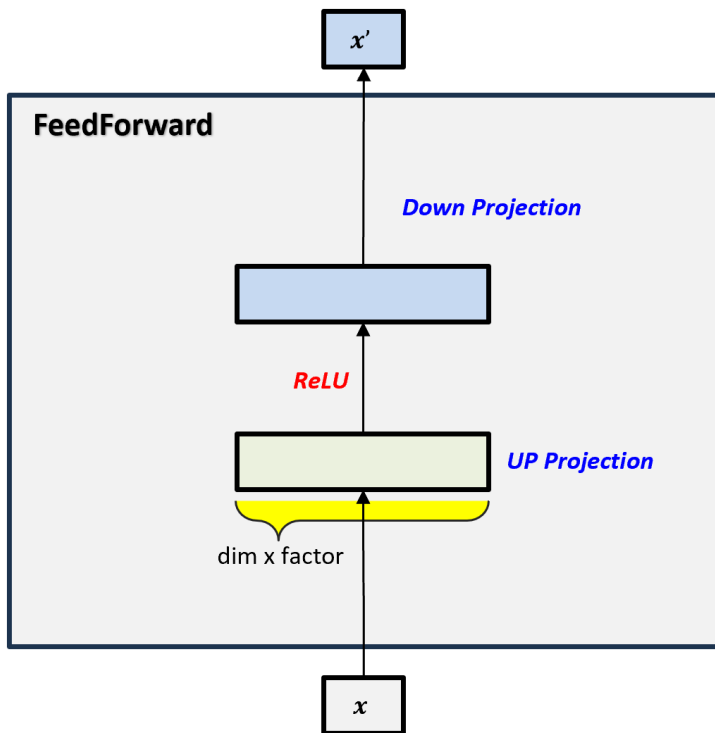
- The input is projected **twice**.
 - One projection, through the activation, acts as a **gate** (a soft volume knob), and it multiplies the other projection **elementwise**.
- The fully gated FFN is written in this form:

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

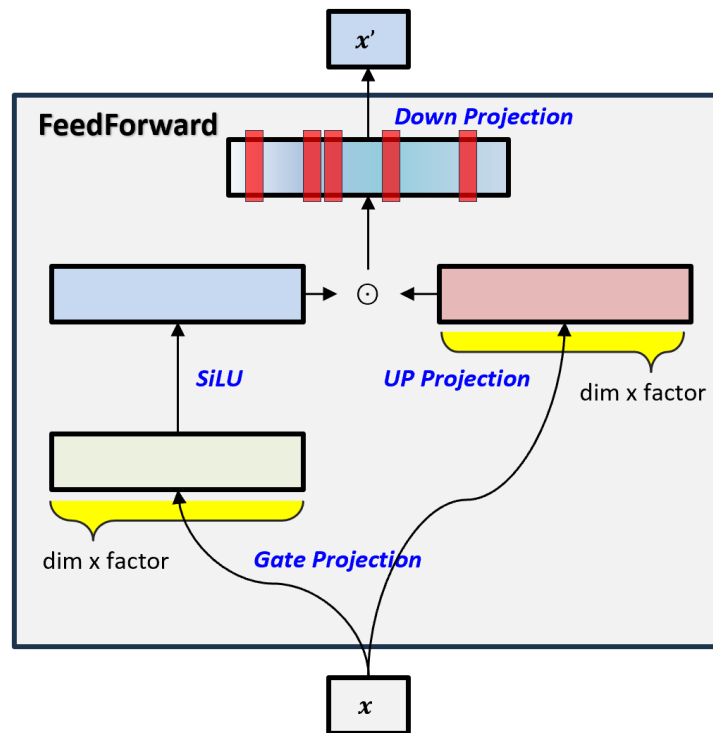
$$\begin{aligned}\mathbf{W}_1, \mathbf{V} &\in \mathbb{R}^{d \times d_{\text{ff}}}, \\ \mathbf{W}_2 &\in \mathbb{R}^{d_{\text{ff}} \times d}\end{aligned}$$

Gated activations (*GLU)

Original Transformer (2017)



Llama1 (2023)



Activations: Gated activations variants

- **GeGLU**

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

- **Notable models:** T5 v1.1, Gemma family (1 - 4)

- **SwiGLU** (or sometimes called: **SiLU**): swish function is $x \cdot \text{sigmoid}(x)$:

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

- **Notable models:** Widely used in recent LLMs

Do Gated Linear Units work?

- Yes, fairly consistently so.

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

Do gated linear units work?

- Yes, fairly consistently so.

Lower loss

Alas the gains of pre-training do not uniformly generalize to the end tasks. sigh ...

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

Recap: FFN layers

- Many variations (ReLU, GELU, *GLU) across models.
- **Gated activations (*GLU):** two projections; one projection gates the other elementwise.
 - **Cost:** three matrices instead of two, so we shrink d_{ff} maintain same FLOPS.
 - **Gains:** Modest gains at equal compute (Shazeer, 2020).
 - Default in nearly all LLMs since 2023; SwiGLU is the most common.
- **Bias term:** People more generally drop bias terms.



Architecture layout



Serial vs Parallel Layer

- The standard “serial” formulation: they compute attention, then the MLP
 - $h = x + \text{Attention}(\text{LayerNorm}_1(x))$
 - $y = h + \text{MLP}(\text{LayerNorm}_2(h))$
- The **parallel** formulation:
 - $y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$
- **Tradeoff:** the parallel MLP does not receive the block’s attention output.
- **Efficiency:**
 - Both branches read the **same** $\text{LN}(x)$, so one LayerNorm suffices.
 - The attention QKV and MLP up-projection can be fused into one matmul.

Parallel Layer: do they work?

- The evidence is sparse.
- PaLM claims that parallel formulation:
 - “~15% faster training speed at large scales”
 - “Ablation showed a small quality degradation at 8B but no quality degradation at 62B scale”
 - the paper didn’t ablate at 540B; it extrapolated from the 8B and 62B results.

Serial vs Parallel Layer: who uses what?

Parallel share

0/5

1/5

2/7

3/10

1/13

1/13

1/9

Serial

attn → MLP

ELECTRA

GPT-3

DeBERTa

ViT

mT5

Jurassic-1

MT-NLG

Swin V2

Gopher

Chinchilla

BLOOM

GLM-130B

Galactica

OPT

LLaMA

Llama 2

Mistral

DeepSeek

MPT

Qwen

Yi

Mixtral

Gemma

Jamba

Llama 3

DS-V2

Chameleon

Qwen2

Gemma 2

OLMo 2

Grok-1

Phi-3

DS-V3

MiniMax-01

Gemma 3

Qwen3

Llama 4

Nemotron-H

GLM-4.5

Kimi K2

SmolLM3

gpt-oss

Qwen3-Next

Kimi Linear

OLMo 3

Trinity

Kimi K2.5

Qwen3.5

GLM-5

Nanbeige

Gemma 4

DS-V4

M2.5

Parallel

attn + MLP

GPT-J

GPT-NeoX

PaLM

ViT-22B

Falcon

Pythia

Command R

Command A

Tiny Aya*

2020

2021

2022

2023

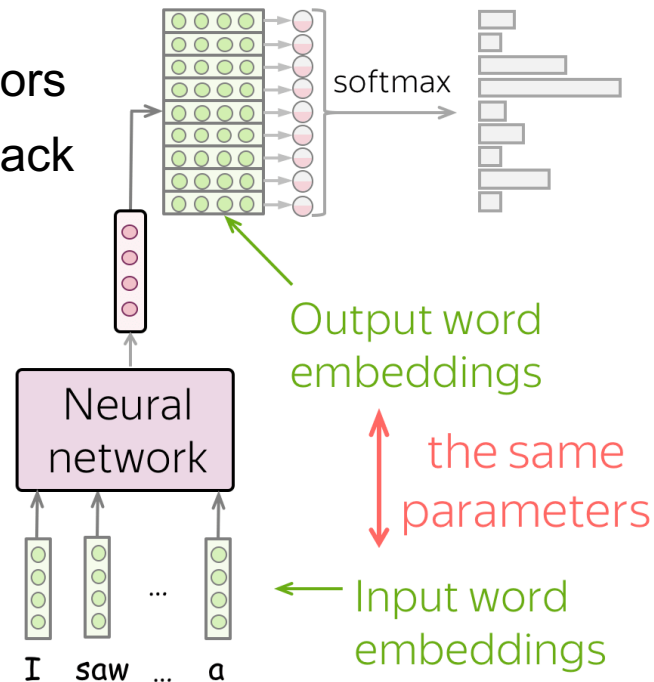
2024

2025

2026 H1

Embedding parameter tying

- Embed matrix ($\text{vocab} \times d$): turns token IDs into vectors
- Output projection ($d \times \text{vocab}$): turns hidden states back into logits over the vocabulary.
- **Problem:** Both are vocab-sized and often among the largest single tensors in the model.
 - Their dimensions match (after a **transpose**).
 - Do we need to keep separate matrices?



Embedding parameter tying

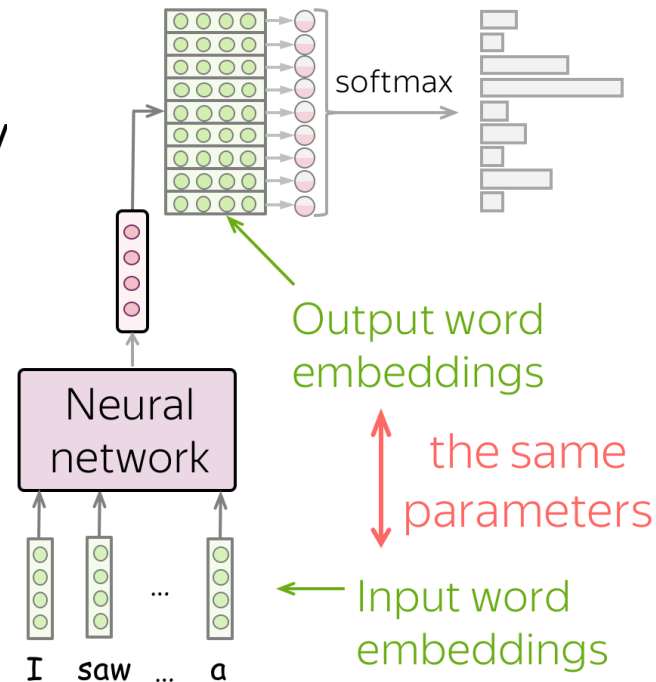
- The same weight matrix can be used for both the input embeddings and the output (projection) layer

```
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
```

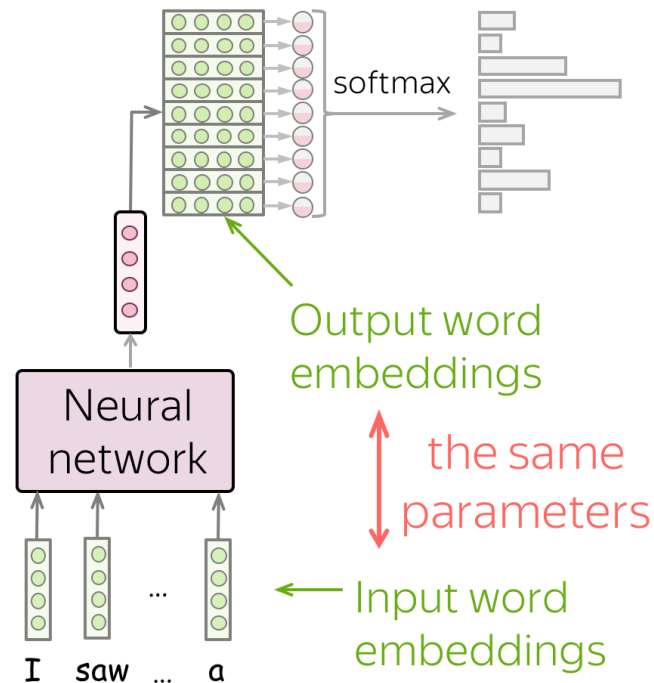
Because nn.Linear stores a linear layer's weight as (out_features, in_features).



Where did the transpose go? Hmm ...

Embedding parameter tying: justification

- **Intuition:** Input & output embeddings should live in the same space.
 - Intuitively, a token's meaning shouldn't depend on whether it's being *read* in or *predicted*.
- **Better embeddings for rare words:**
 - The output matrix gets a gradient for every vocab token at every softmax step.
 - The input emb only updates for tokens in the batch.
 - Tying lets rare words learn from that dense signal.
- **Memory efficiency:** Cuts one vocab $\times$ d matrix.
 - GPT-2 small: ~38M of 124M params (~30%).
 - Though the gains diminish for larger models (why?).



Who ties embedding parameters?



One caveat about this visualization: tying can vary within a family. There is often more incentive to tie embed/unembed matrices for **smaller** models.

Recap so far

- **Serial vs parallel layers:**

- Parallel layers have a compute win.
 - (even though their arithmetic work is not necessarily lower.)
- Empirical evidence is sparse.

- **Parameter tying:**

- Reduces model parameters and leads to (modest) gains.
- Unclear how much it matters at scale; many large LLMs (e.g., LLaMA) don't tie.

Architecture Hyperparams

There are many 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?
- **Note:** Ultimately the goal is to set these params in a way that we reach the lowest loss, given a set of constraints (e.g., total compute budget).

d_{ff} : How Wide Should the FFN Be?

- Feedforward – model dimension ratio:

$$\begin{aligned}\text{FFN}(\mathbf{x}) &= f(\mathbf{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + \mathbf{b}_2 \\ \mathbf{W}_1 &\in \mathbb{R}^{d \times d_{\text{ff}}}, \\ \mathbf{W}_2 &\in \mathbb{R}^{d_{\text{ff}} \times d}\end{aligned}$$

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

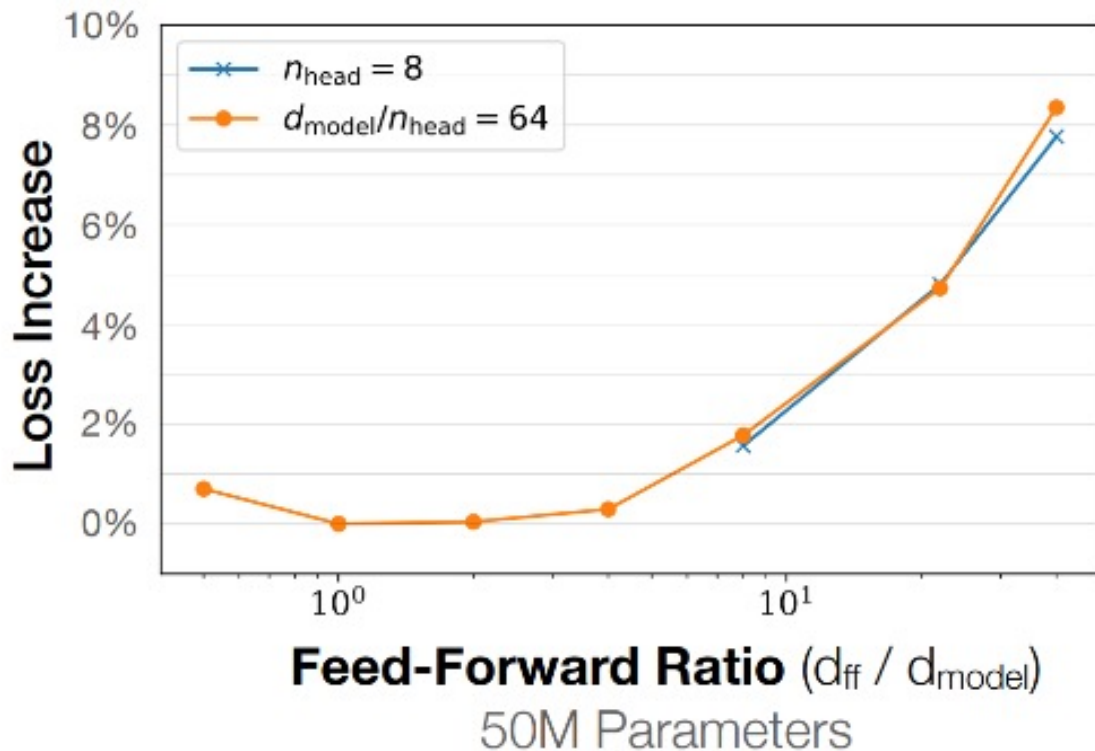
$$d_{\text{ff}} = 4d$$

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

[Slide credit: Tatsu Hashimoto]

d_{ff} : How Wide Should the FFN Be?

- Each point is a ~50M (non-embedding) param model. We vary the ratio $d_{\text{ff}}/d_{\text{model}}$.
- Empirically, there's a basin between 1-10 where this hyperparam is near-optimal.

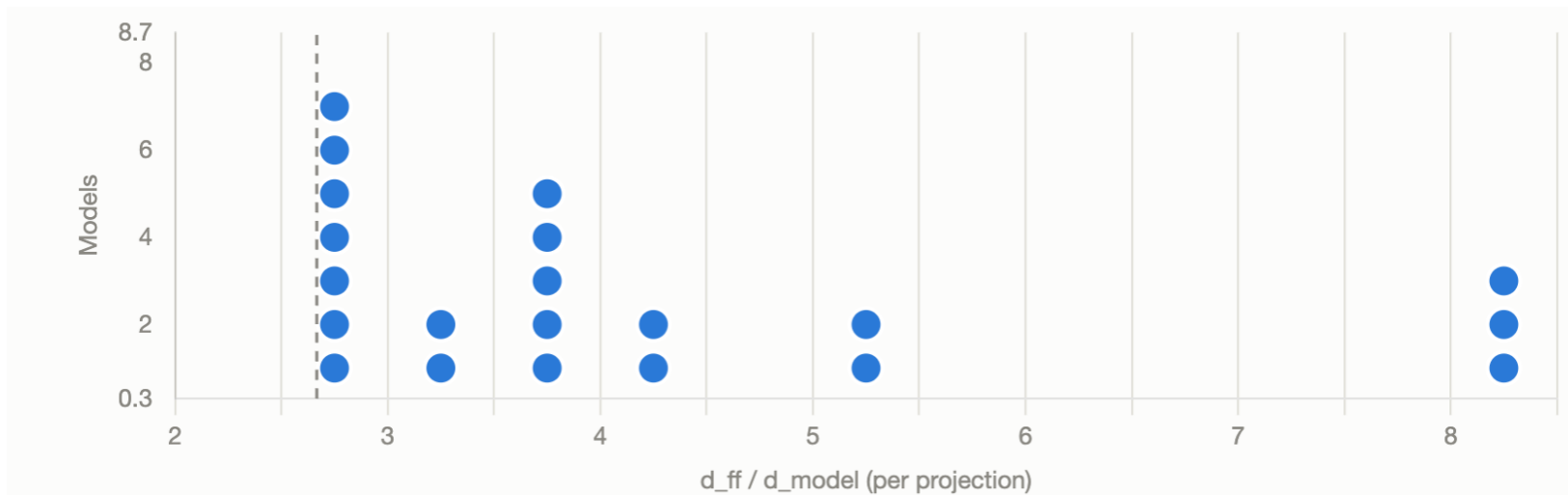


[Slide credit: Tatsu Hashimoto]

Setting d_{ff} for GLU/Gated variants

- Remember that GLU activations add extra matrix computation.
- One tradition is to set its d_{ff} in a way that it maintains that FFN param counts (hence, FLOPS).
 - Params in a non-gated FFN: $2d_{\text{ff}} \times d = 2 \times 4d \times d = 8d^2$
 - Params in a gated FFN: $3d_{\text{ff}} \times d$
 - $3d_{\text{ff}} \times d = 8d^2 \rightarrow d_{\text{ff}} = \frac{8}{3}d = 2.66\textcolor{red}{d}$

Setting d_{ff} for GLU/Gated variants



Notice that most models are within the range $\frac{d_{\text{ff}}}{d} \approx 2.66$

Head Dimension vs. Model Dimension

- Remember:

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

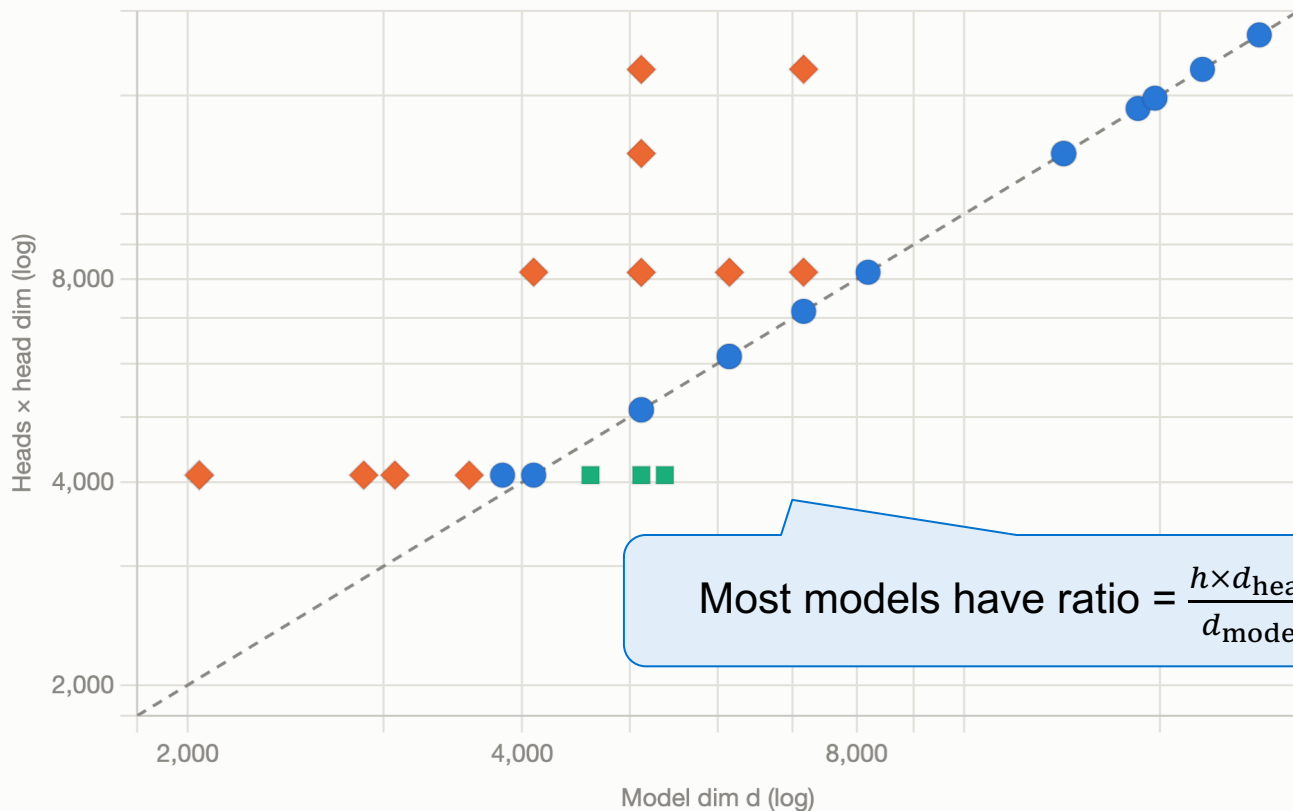
In practice, we use a reduced dimension for each head.

$$\mathbf{W}_i^q, \mathbf{W}_i^k, \mathbf{W}_i^v \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}, \quad \mathbf{W}^o \in \mathbb{R}^{(h \times d_{\text{head}}) \times d_{\text{model}}}$$

- The consensus: $h \times d_{\text{head}} = d_{\text{model}}$
- This doesn't have to be true since the matrix ($\mathbf{W}^o$) can take care of projection to model-dim.

Head Dimension vs. Model Dimension

● Ratio ≈ 1 ◆ Ratio > 1.1 ■ Ratio < 0.95

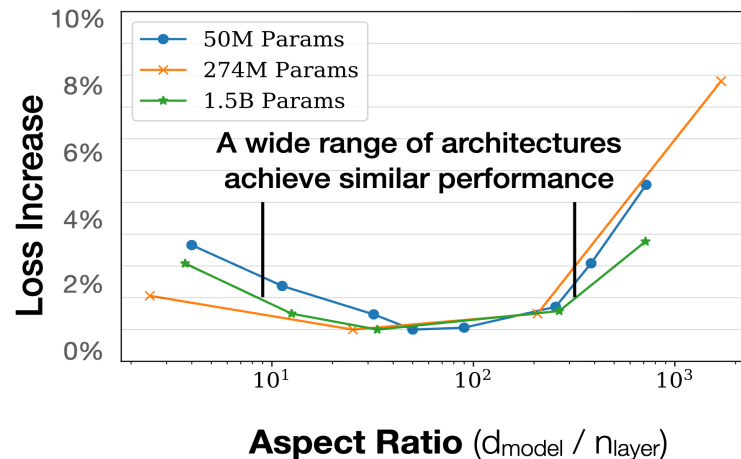


Aspect ratios (width vs depth)

- Wide range of ‘good’ values (100-200) with respect to loss.

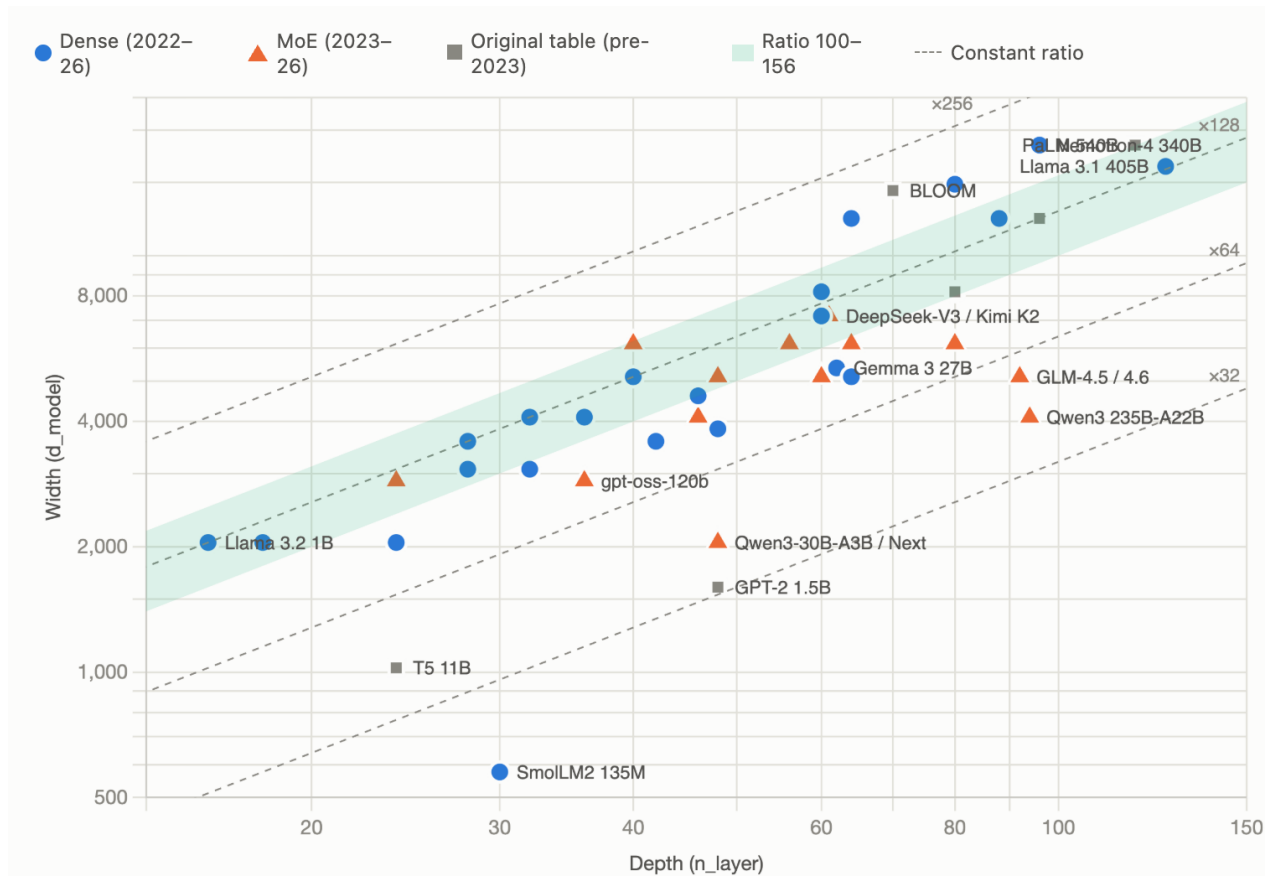
[Kaplan et al 2020]

- System-level considerations are not reflected here.
 - Extremely deep models are harder to parallelize.
 - Depth introduces sequential dependencies;
 - Width enables more within-layer parallelism
 - but can also increase communication.



Aspect ratios (width vs depth)

- Should my model be deep or wide? How deep and how wide?
- Note here width is the hidden dimension, **not** the context window width.
- Most models stick to ratio between 100-200.



Recap of architecture hyperparams

- FFN width (d_{ff}/d)
 - $4\times$ (standard) and $8/3\times$ (GLU) are parameter-matched defaults
 - Kaplan: flat basin at $\sim 1\text{--}4\times$ at fixed N (50M scale)
 - Modern GLU models span $\sim 2.5\text{--}8\times$ (median 3.5)
- Attention head dim:
 - Standard: $d_{\text{head}} \times h = d$
 - Standard convention; Kaplan finds weak sensitivity at fixed N
- Aspect ratio (d/n_{layers})
 - Wide range of ‘good’ values (100-200).
 - Systems constraints (in addition to quality constraints) influence the choice.

Pre-training data

LLM consume how much tokens?

Model	Release	Training tokens	Training data
BERT	2018	~131B	BooksCorpus (800M words) + English Wikipedia (2.5B words)
T5	2019	34B	C4 (cleaned Common Crawl)
GPT-3	2020	300B	Filtered Common Crawl, WebText2, Books1, Books2, Wikipedia
Gopher	2021	300B	MassiveText
Chinchilla	2022	1.4T	MassiveText
PaLM	2022	780B	Web pages, books, Wikipedia, conversations, GitHub code
BLOOM	2022	366B	ROOTS (46 natural + 13 programming languages)
Llama 2	2023	2T	Publicly available online data
Llama 3	2024	15T+	Publicly available online data
DeepSeek-V3	2024	14.8T	Multilingual web, with extra math and code
Qwen2.5	2024	18T	Multilingual web, code, math
OLMo 2 (13B)	2024	5.6T	OLMo-Mix-1124 + Dolmino-Mix-1124 (fully open)
Gemma 3 (27B)	2025	14T	Multilingual web, code, math, images
Qwen3	2025	~36T	119 languages; web, code, STEM, synthetic data
Kimi K2	2025	15.5T	Web, code, math, knowledge, plus rephrased data

Note that token counts depend on the tokenizer of each model.

They used to be in billions of tokens; now they're in trillions.

Where do we begin to collect data?

- Where do I find a very large dataset?
 - Crawling web is non-trivial (unless you're OpenAI with ton of resources).
 - The alternative is to look for websites that do the crawling for you.



- A non-profit that release a new crawl of the internet **every month**.
 - So far, there have been 100+ crawls from 2008.
 - In 2016, a crawl took 10-12 days on 100 machines.
 - **Not** a complete of the internet; they have some overlap but try to diversify.
 - Common Crawl follows links from previously crawled pages.

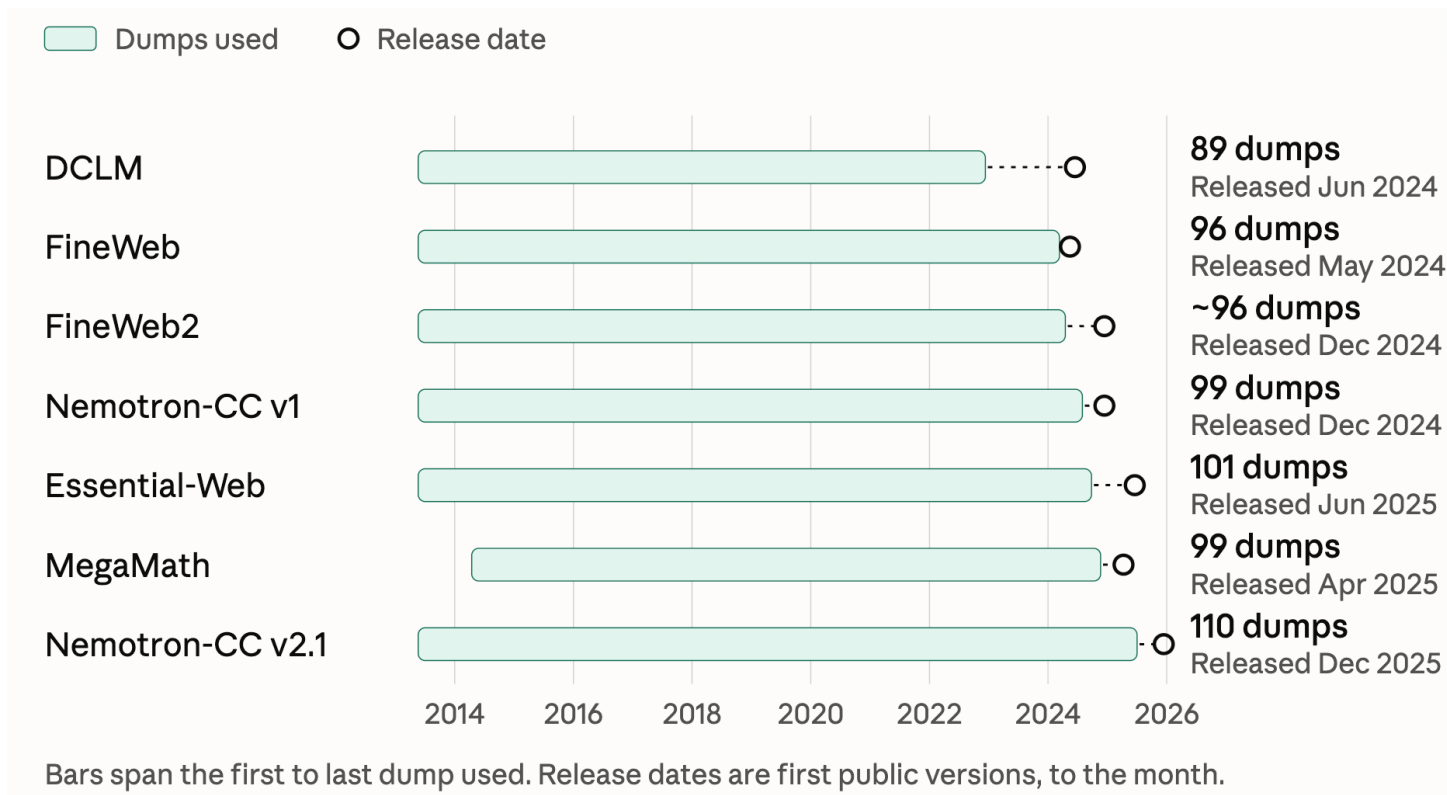


- CC is a common sources of pre-training data.
 - **WARC**: The raw HTTP responses, including full web pages.
 - **WAT**: The metadata summary from WARC files.
 - **WET**: The extracted plaintext from WARC files, stripping out HTML and other non-textual content.
- Also note, it respects robots.txt. So you'd find less diverse data in recent crawls.

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

How many dumps of CC do people use?

- Most efforts use many dumps to prepare their pre-training data.



CC: Examples

- These are some random documents that we fetched from CommonCrawl.
 - <https://github.com/danyaljj/sample-cc-docs/>
- What do you think?



CC is not clean. Is that a concern?



Cleaning data

- There is a lot of “junk” in CC data.
- It's common to remove a large chunk of it so obtain high-quality data.

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

Cleaning data

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 lemon is unknown, though

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.

Lorem 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);  
  }  
}
```

Retain:

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

Remove any:

- “[Lorem ipsum](#)” text (dummy text), “terms of use”, etc.

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	19.24	80.88	71.36	26.98	39.82	27.65
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?



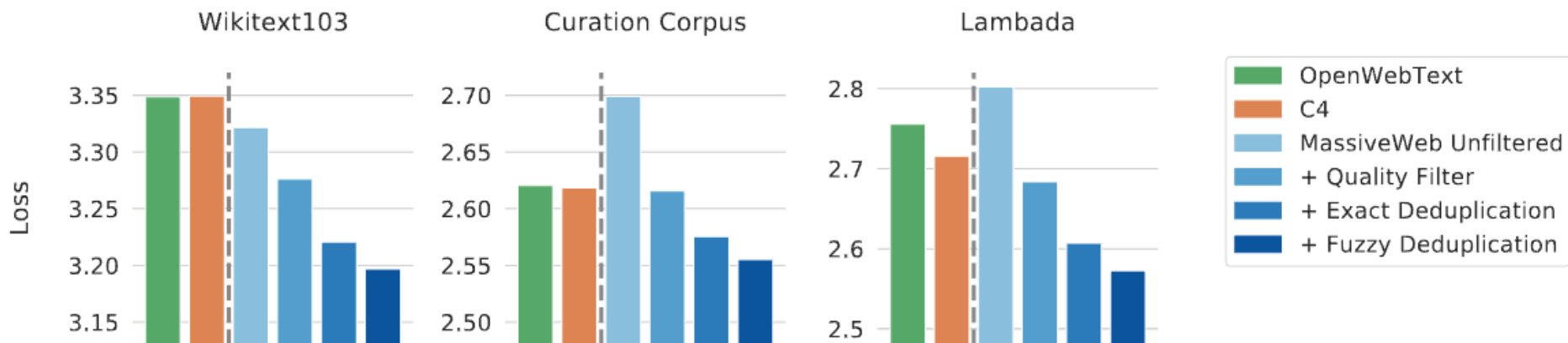
Pre-training Data Duplicates

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

Dataset	Example	Near-Duplicate Example
Wiki-40B	<code>\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. [...]</code>	<code>\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. [...]</code>
LM1B	<code>I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .</code>	<code>I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .</code>
C4	<code>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!</code>	<code>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!</code>

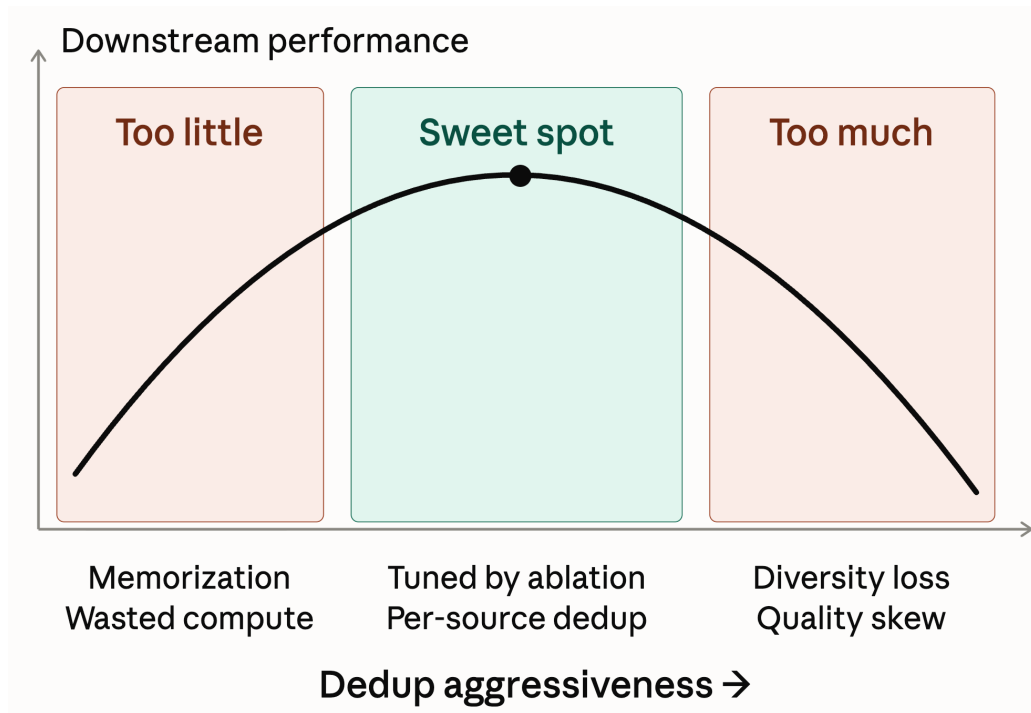
Deduplicating Data Improves LMs

- Perf of 1.4B param model (lower is better) with progressively more pre-processing.
- Applying a quality filter and de-duplication stages significantly improves quality.



Deduplicating Data: Caution

- Don't over-deduplicate: some repetition is signal!
- Facts that matter appear organically and often, in many phrasings
 - e.g., “Obama was US president from 2009 to 2017”
- Aggressive near-dup can collapse these, so the model may forget them



How do you scale data deduplication?

- Pre-training is huge. Naively deduplicating the data is going to take forever!!
- How do you deduplicate it?
 - Naively hashing each document (a good baseline)
 - Scaling it up requires advanced data structures.
 - SuffixArray
 - MinHash
 - BloomFilters
- There is no consensus on how dedup should be done (different works do it differently: different combination of algorithms and levels of aggressiveness.).

The simplest: hashing documents

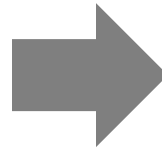
- Hash all documents, so each document receives one unique hash.
- **Efficiency:** This will be fast.
- **Granularity:**
 - This will be sensitive to small changes; any change in the document (e.g., one word change) would change its hash.
 - Also, we're deduplicating full documents.
- Different choices 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:
 - $O(m)$

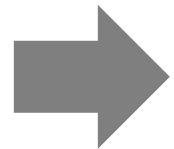
0	abaaba\$
1	baaba\$
2	aaba\$
3	aba\$
4	ba\$
6	a\$
7	\$

Sort suffixes
lexicographically



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

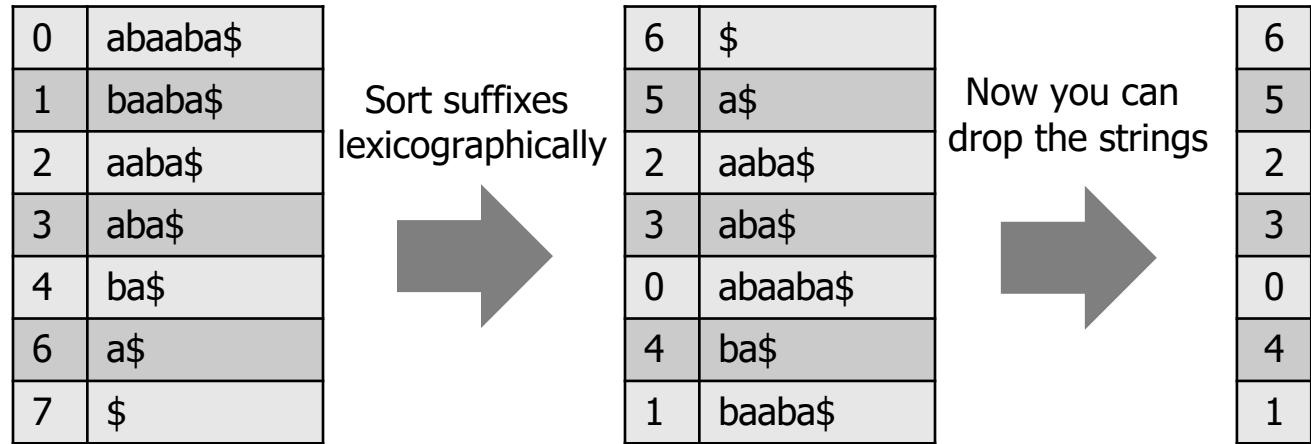
Now you can
drop the strings



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:
 - $O(m)$



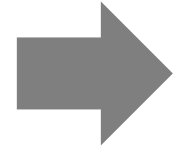
- You don't need the suffixes since, given their index, you can look them up from T .

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 ≥ 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: $O(n \log m)$
 - for $m = \text{len}(T)$ and $n = \text{len}(P)$
 - 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: [Soldaini at 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
	X	X	X	X	76B	00%	40.1	+0.0
	✓	X	X	X	66B	13%	41.0	+0.9
	X	✓	X	X	62B	18%	40.9	+0.8
	X	X	✓	X	51B	33%	41.4	+1.3
	X	X	X	✓	56B	26%	41.7	+1.6
	✓	✓	X	X	58B	24%	40.2	+0.1
	✓	X	✓	X	49B	36%	41.3	+1.3
	X	✓	✓	X	48B	37%	41.2	+1.2
	✓	✓	✓	X	45B	41%	41.7	+1.6

Individual
technique

Combined
techniques

What people do: Deduplicate over what unit?

- You can look for repeats across different units: documents, paragraphs, etc.
- Most works dedup on document-level.
- Some pipelines are very thorough (e.g., Llama3) and cover many units.

Unit	Catches	Used by
URL	Same page recrawled	Llama 3
Document	Copies, mirrors, templated pages, small edits	FineWeb Essential-Web FineWeb2 HPLT v2 DCLM Llama 3 Nemotron-CC TxT360 OLMo 3
Paragraph	Shared quotes and disclaimers	DCLM
Line	Menus, cookie banners, footers	Llama 3
Substring	Long repeated passages	Nemotron-CC OLMo 3

What people do: Which algorithm?

- Which algorithm you use depends on what unit of data you apply them to.
- MinHash LSH is near-universal for near-duplicates;
 - DCLM's Bloom filter is the main exception.
- Approximate matching happens almost only at the document level.
 - Finer units use exact matching, which is cheaper and less error-prone on short text.

Unit	Exact match	Approximate match
URL	URL hash Llama 3	—
Document	Content hash Essential-Web Nemotron-CC OLMo 3	MinHash LSH FineWeb Essential-Web FineWeb2 Llama 3 Nemotron-CC TxT360 OLMo 3 Bloom filter DCLM
Paragraph	—	Bloom filter DCLM
Line	Line hash + count Llama 3	—
Substring	Suffix array Nemotron-CC OLMo 3	—

What people do:

Which population gets dedup?

- Shards are arbitrary slices made for parallel computing dedup algorithms.
- Dedup on dumps or shards reduces its aggressiveness.
 - In FineWeb, global dedup cut 36T tokens to 4T and trained worse, because cross-year repeats turned out to be useful.

Population	URL	Document	Paragraph	Line	Substring	...
Each dump	—	FineWeb Essential-Web	—	—	—	
Each language across all dumps	—	FineWeb2	—	—	—	
Each shard or bucket	—	DCLM OLMo 3 (32 shards)	DCLM	Llama 3 (30M docs) OLMo 3 (56 shards)		
Entire corpus	Llama 3	Essential-Web OLMo 3 Llama 3 Nemotron-CC TxT360	—	—	—	

What people do:

What do you do with the duplicates?

- Most efforts simply keep one copy.
- Newer pipelines dedup maintain some degree of duplicates somewhere in the pipeline.

Approach	What happens	Used by
Keep one copy	Other copies removed	FineWeb Essential-Web HPLT v2 DCLM Llama 3 Nemotron-CC
Keep cross-dump repeats	Dedup per dump only	FineWeb Essential-Web
Upsample by count	Repeat by cluster size; huge clusters once	FineWeb2
Upsample by quality	Repeat the best documents	OLMo 3

“Upsample by *”: once duplicates are removed, deliberately put back some controlled repetition.

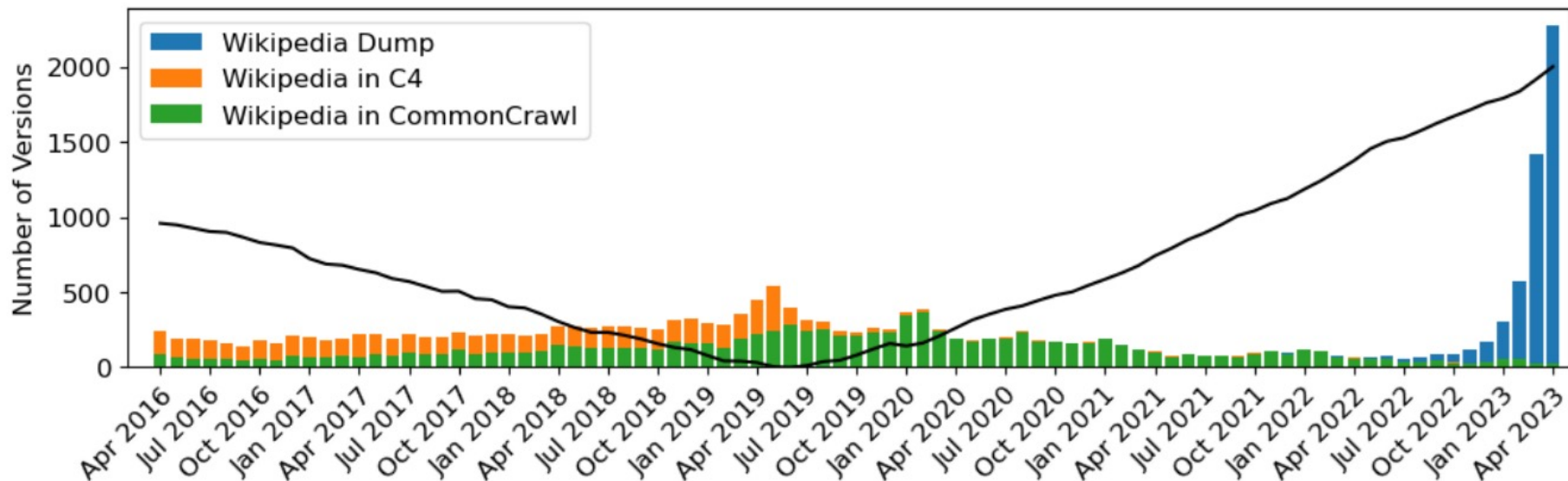


Should I worry about old data
in my pre-training?



Prevalence of stale data: RedPejamas

- Individual dumps of CC are incomplete picture of the internet.
- Therefore, developers fetch multiple dumps. But older dumps contain more older content!
- Old/stale data in pre-training can skew your model toward them.



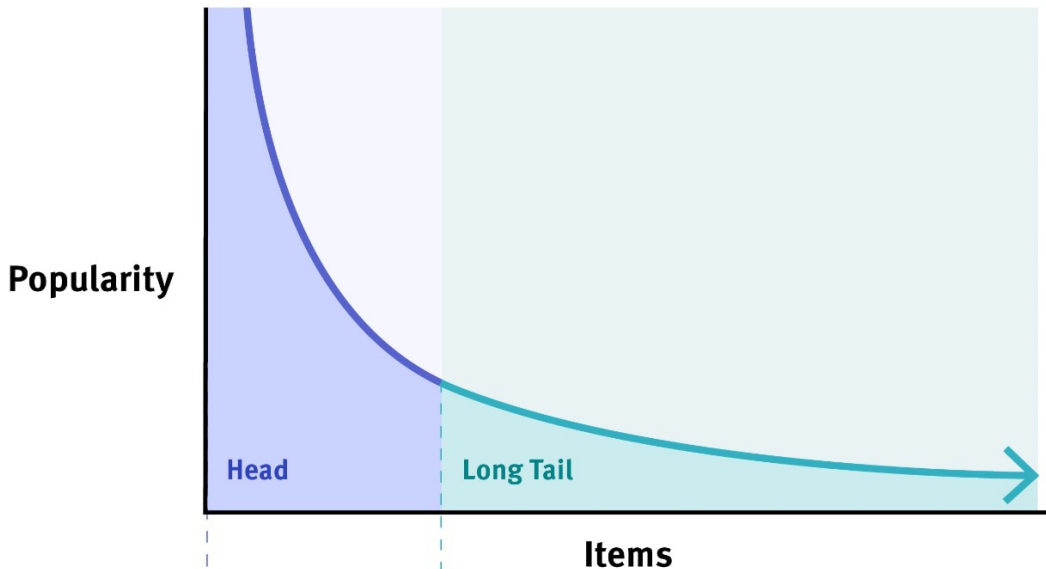


Should I worry about skew of
the data mixtures in my pre-training?



Data mixtures (and the long tail)

- 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.



Data mixtures (and the long tail)

- 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.
 - Overrepresented domains would dominate
 - (e.g., your model would be a muchx100 better at English than Azeri).
- Therefore, you need to balance the data!

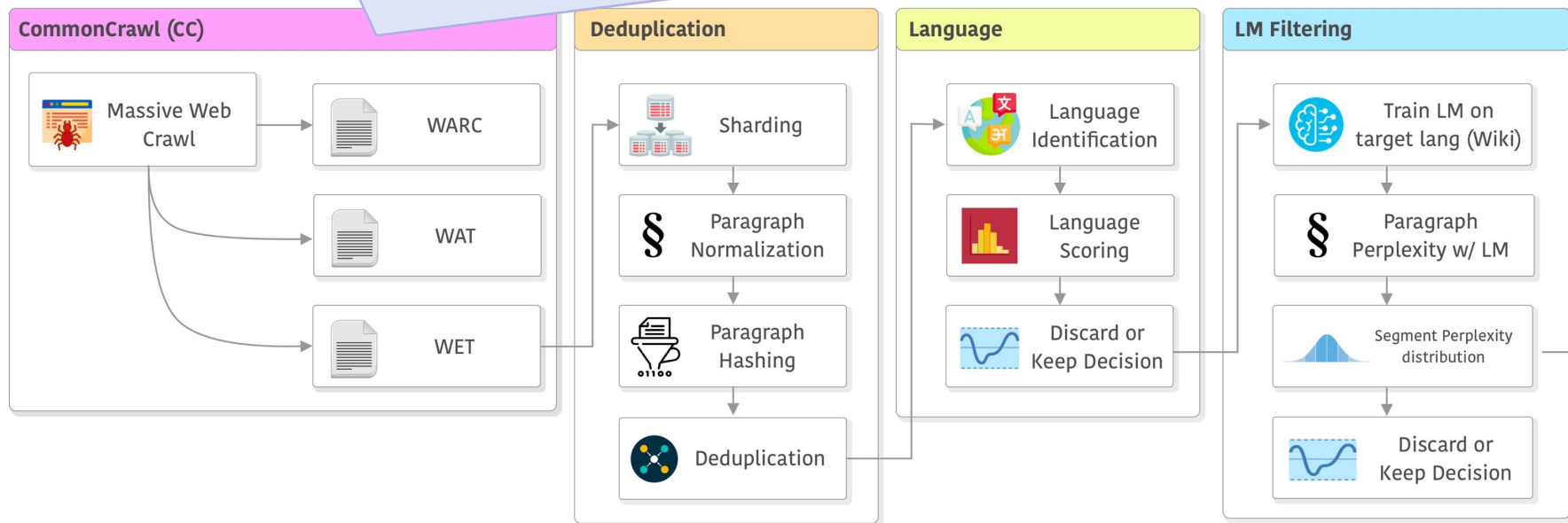
Language filtering

- Many models limit their data to English.
- Danger in English-only filtering:
 - accidentally filtering out dialect of English.
 - Ill-defined for code-switching (e.g., English + Chinese).
- To go beyond English, you need to identify langs. How do people identify languages?
 - A popular choice is [fastText](#) which supports 176 langs.
 - [GlottLID](#) is a similar classifier but identifies many more (1665) languages

Dataset / model	Year	LID	Scope
Dolma*, FineWeb, DCLM	2024	lid.176	English
Llama 3	2024	lid.176	8% non-English
FineWeb2	2025	GlottLID	1,000+ languages
Apertus	2025	GlottLID	1,811 languages

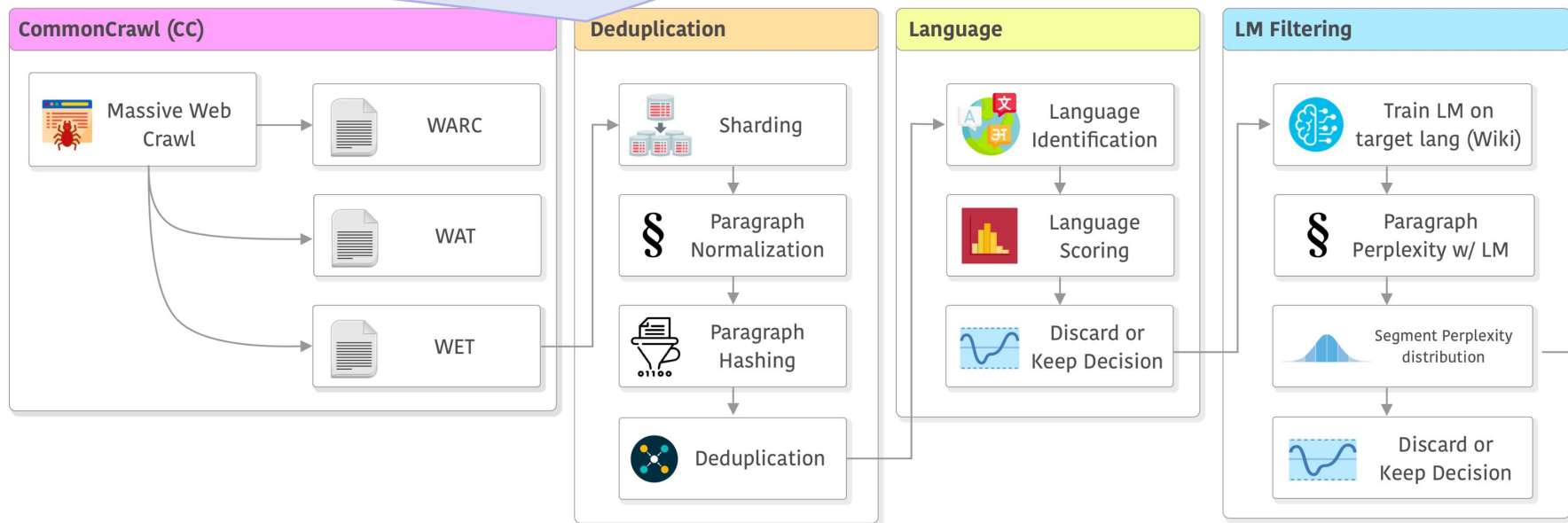
LLaMA 1's Data Pipeline

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.



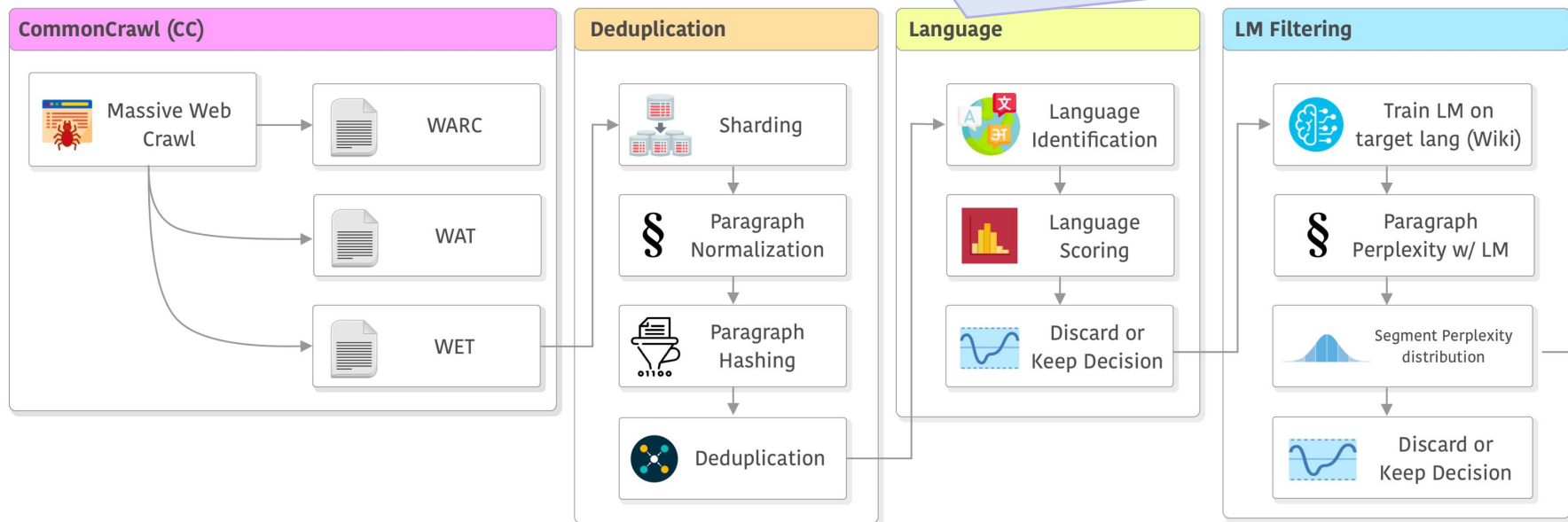
LLaMA 1's Data Pipeline

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

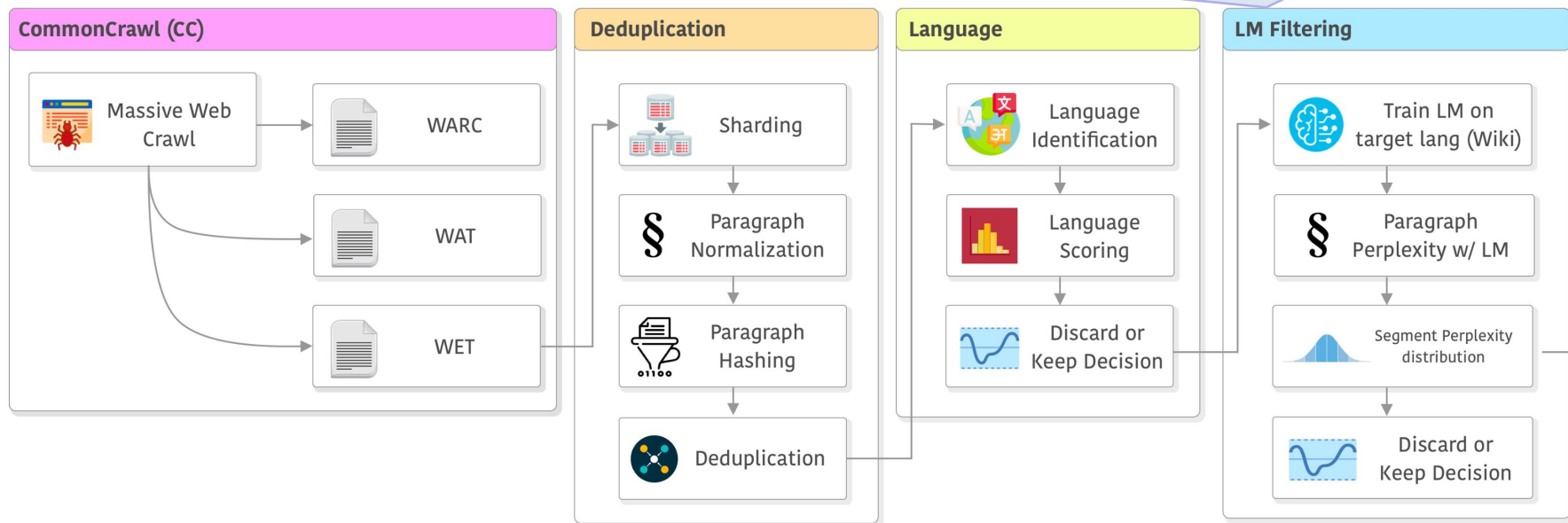
Perform language identification and decide whether to keep or discard languages. The order of when you do this in the pipeline can impact the language discrimination quality.



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



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.
- **Cleaning data** can save tons of compute and even give you perf gains.

The actual pre-training



How should we select the
right hyperparams?

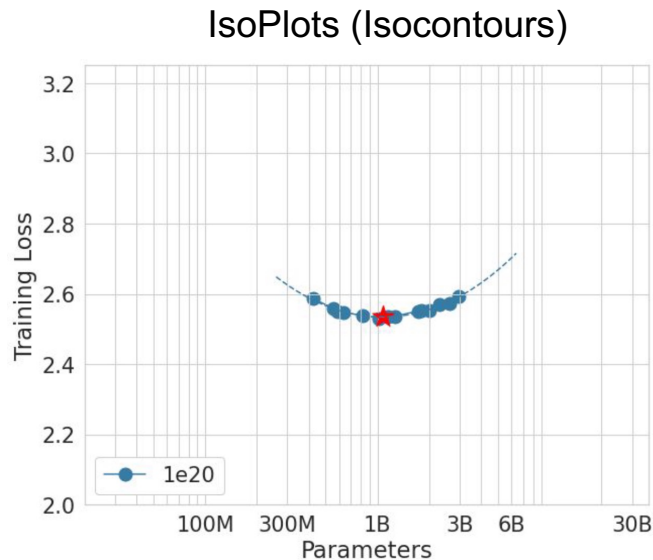


Q: What would you do?

- Zuckerberg gave you a \$500M budget for training Llama-10.
- You set aside \$10M for finding the best architecture at smaller scale, assuming that your ultimate model will be much larger.
- This way, you pick your parameters with rigorous experiments at small scale:
 - Optimal training params: Learning rate, warmup, weight decay, etc.
 - Architecture configs by scaling (each x50) the optimal values at small scale.
- Q: What you like (or don't) about this recipe?
- Optimal depth/width, lr, batch size, weight decay, init, and residual scaling are **not scale-invariant**.

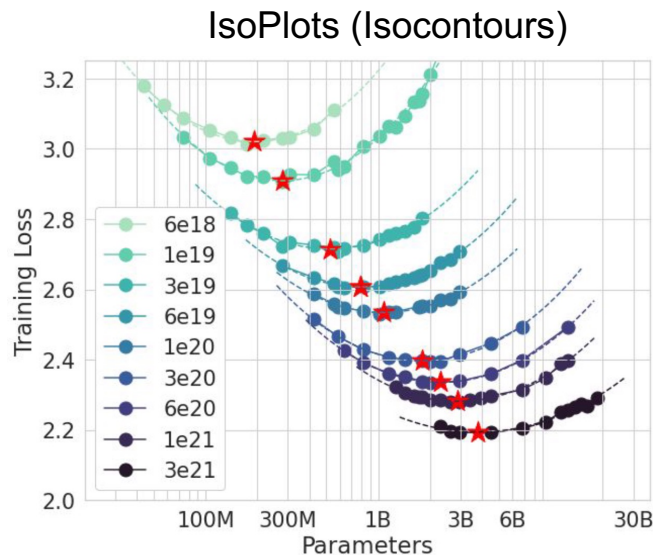
IsoPlots: Tradeoffs at a smaller scale

- The performance of your model depends on a complex combination of many factors.
- Goal: find the best combinations, for a fixed compute.
- Approach:
 1. Fix a compute budget FLOP
 2. Train a few models and vary their size
 3. Fit a parabola and find the minimum



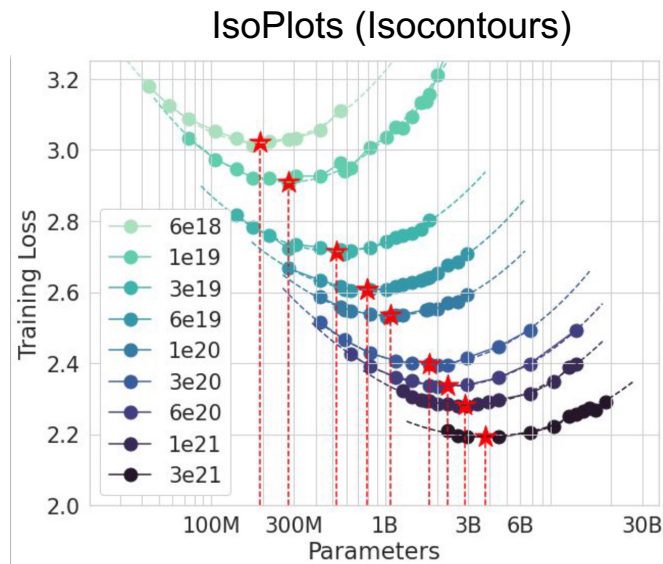
IsoPlots: Tradeoffs at a smaller scale

- The performance of your model depends on a complex combination of many factors.
- Goal: find the best combinations, for a fixed compute.
- Approach:
 1. Fix a compute budget FLOP
 2. Train a few models and vary their size
 3. Fit a parabola and find the minimum
 4. Repeat 1-3 for various FLOP budgets



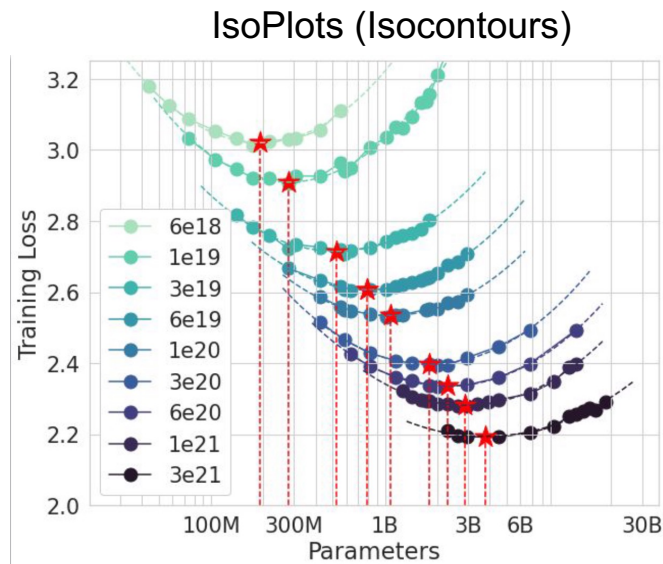
IsoPlots: Tradeoffs at a smaller scale

- The performance of your model depends on a complex combination of many factors.
- Goal: find the best combinations, for a fixed compute.
- Approach:
 1. Fix a compute budget FLOP
 2. Train a few models and vary their size
 3. Fit a parabola and find the minimum
 4. Repeat 1-3 for various FLOP budgets
- It's good to change various parameter (e.g., training data, size, or other hyperparams) and see how it's quality (loss) changes.



Predictive models for parameters

- Overall, IsoPlots show how loss depends on three axes: model size (parameters N), dataset size (tokens D), and compute budget C .
- They're a tool for reasoning about *how to spend your training budget efficiently*.
- But it also allows one to see track effective hyperparams (LR, batch size, etc.) changes with N , C , or D .
- **Lesson:**
 - Don't overtune your hyperparameters at small scales and expect to use them at large
 - Instead develop predictive metrics based on parameters:



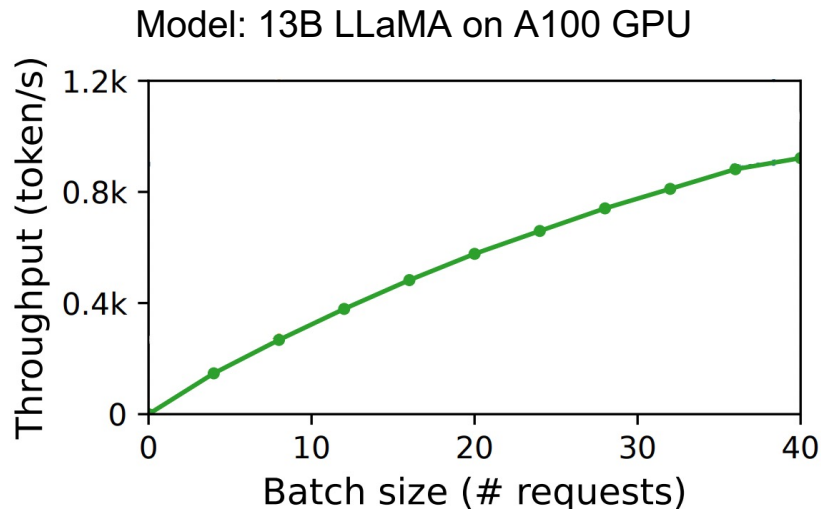


How should I
train the model?



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).
- If you can afford larger batch (larger GPU), it's generally worth it.



Q: Agreed or not

- “Use the largest batch size that you can, at all costs” – agreed or not?
- To allocate more batches of data within your memory, you may need to truncate each instance shorter.
- The other issue is about the quality of your estimated gradients:
 - Small batch sizes will result in **noisy gradients estimates**.
 - Therefore, the model may not be able to converge to the optimal performances.
 - A large batch size while giving very accurate gradient estimations will tend to make **less use of each training token**
 - Slower convergence and potentially wasting compute.

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

Their seq length:
2,048 tokens

Note:
tokens per batch =
batch size x seq len

[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

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

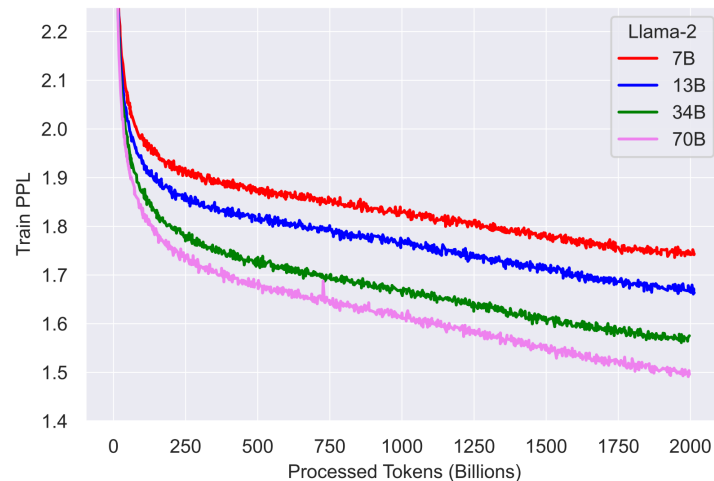
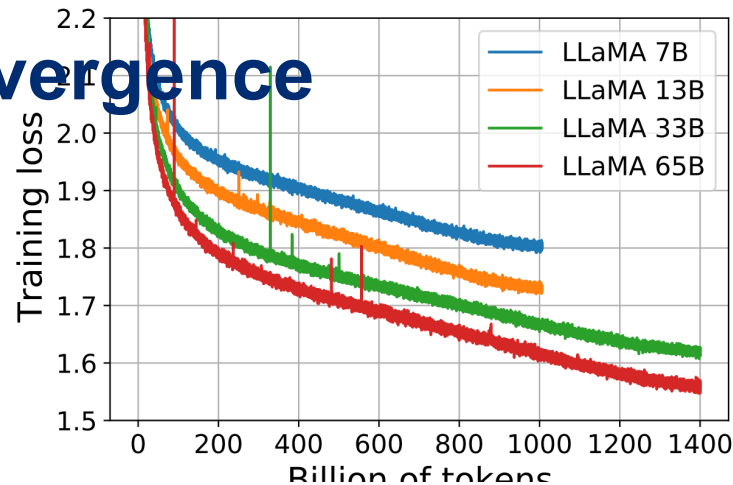
* Most of the times papers just don't discuss dropout. On open models, this closely matches not doing dropout. This may not be true of closed models.

[Slide credit: Tatsu Hashimoto]

- <https://arxiv.org/pdf/2509.01440>

Monitoring the 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.



Monitoring the convergence with end tasks

- Some works have also monitored **end task** performance during pre-training.
- Use **likelihood** of the correct answer rather than accuracy
 - (you don't even need to consider the incorrect answers)
- Very similar to the "cloze mmlu" trend where you use the probability of the full answer instead of A, B, C, D.
 - Not discrete metrics (e.g., Accuracy)

Monitoring the convergence with end tasks

- These two typically correlate, but not always.

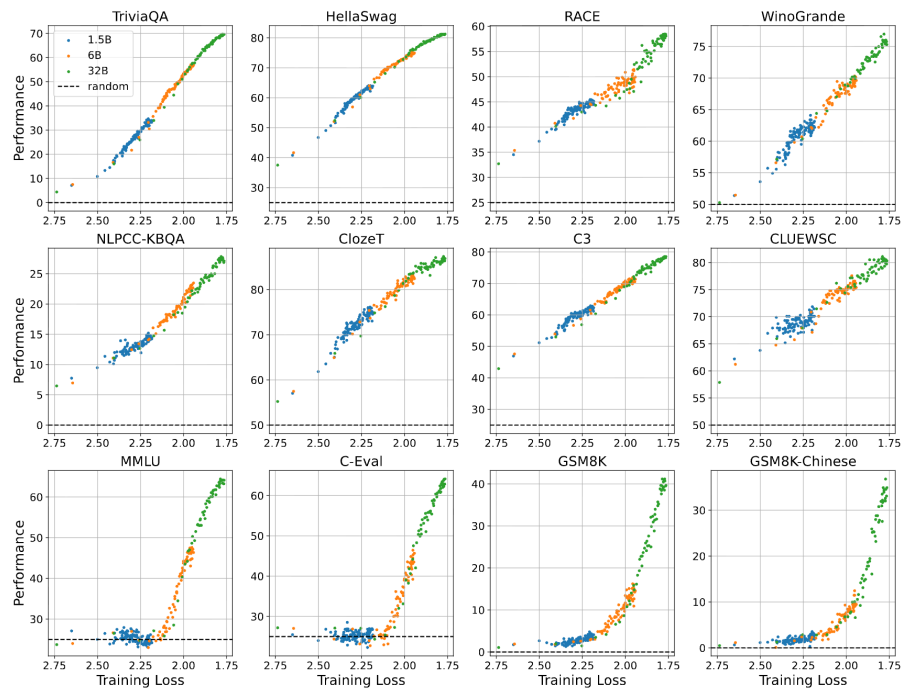


Figure 1: **The performance-vs-loss curves of 1.5B, 6B, and 32B models.** Each data point is the loss (x -axis) and performance (y -axis) of the intermediate checkpoint of one of the three models. We mark the results of random guess in black dashed lines.

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.

- Lots of cool MOE + RL stuff: <https://cursor.com/resources/Composer2.pdf>

- Packing docs during pre-training: Inter-document masking (also called **sample packing with attention masking**) is widely used in modern LLM pretraining and finetuning.
- <https://claude.ai/share/9dcd5054-556f-4929-a04a-717f04b25e9a>

Karpathy Zero to Hero

<https://www.youtube.com/playlist?list=PLAqhlrjkxbuWI23v9cThsA9GvCAUhRvKZ>

Nate's slides

- https://docs.google.com/presentation/d/1oWoIrRXzLONu_oSXRG3lJdRRwIO3kkKmg37mLQtSfi4/edit?usp=sharing
- Some of these will go to the efficiency deck.
- Some others (like small models) will go here.