What was Revolutionized in the “Transformer Revolution”?

Stella Biderman
Executive Director @ EleutherAI

About us

EleutherAI is a non-profit research institute that specializes in large scale AI systems as well as making the field more accessible to researchers

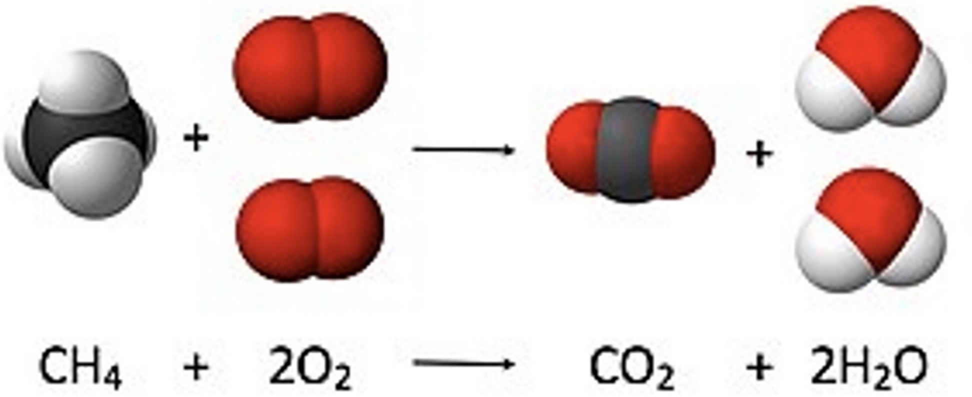
We do research on natural language processing and interpretability such as:

- Architecture design (parallel layers, RoPE, RWKV)
- Model Evaluation
- Learning dynamics
- Memorization

Created popular open source training (GPT-NeoX) and evaluation (lm-eval) libraries

Imagine an alternative world...

Chemistry equations are formulae that describe what *ratios* of reactants combine to gives the *corresponding amounts* of products.



$$CH_4 + 2O_2 \longrightarrow CO_2 + 2H_2O$$

Suppose we didn’t know that, and instead view them like baking instructions that “make X” for some amount X.

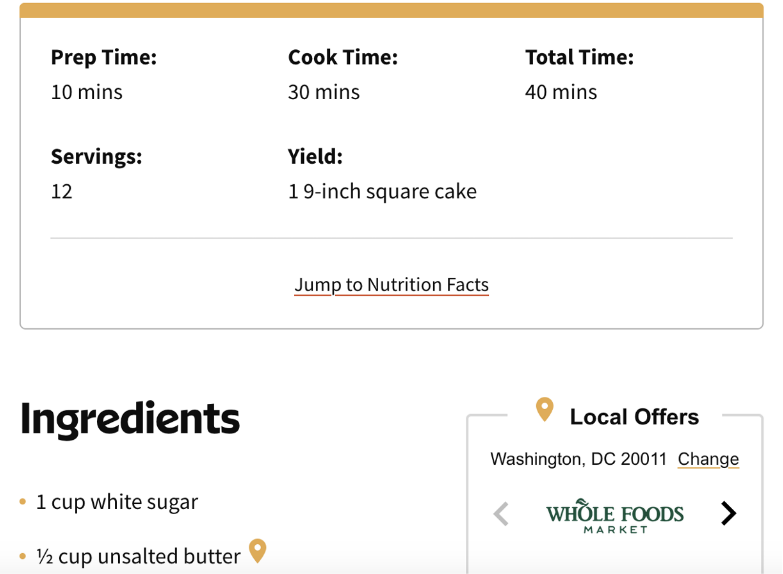
Imagine an alternative world...

Previous SOTA: Recipe 1 converts 5 A into 3 B

Discovery 1: Recipe 2 converts 3 A into 2 B

Discovery 2: Recipe 2 can also convert 3N A into 2N B *for any integer N*

If production of B goes through the roof, which discovery is more responsible?



Prep Time: 10 mins

Cook Time: 30 mins

Total Time: 40 mins

Servings: 12

Yield: 1 9-inch square cake

Jump to Nutrition Facts

Ingredients

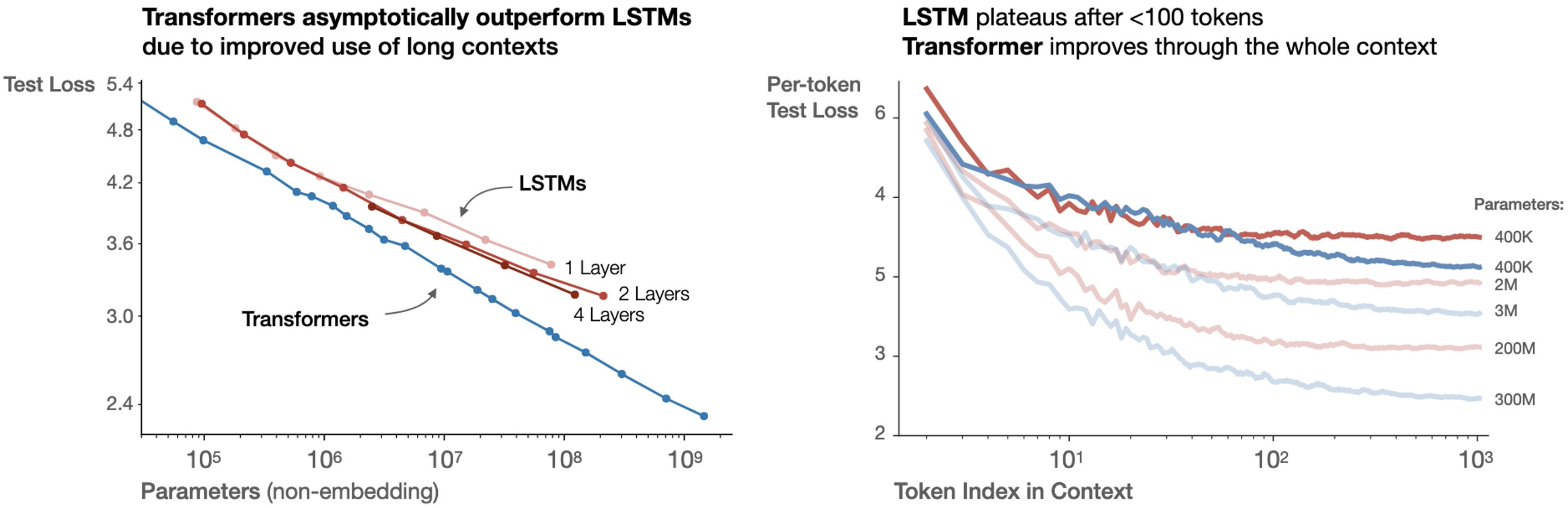
- 1 cup white sugar
- ½ cup unsalted butter

This is a simplified version of what happened in AI

What was “discovery 2”?

1. Converting unlabeled data into self-supervised data via next token prediction
2. Ability to train on the entire sequence simultaneously
3. Ability to distribute model training efficiently across hundreds of GPUs
4. Being willing to scale models up

Don’t transformers out preform RNNs?



Transformers asymptotically outperform LSTMs due to improved use of long contexts

LSTM plateaus after <100 tokens
Transformer improves through the whole context

“Scaling Laws for Neural Language Models”
Kaplan et al. (2020)

RWKV matches transformers

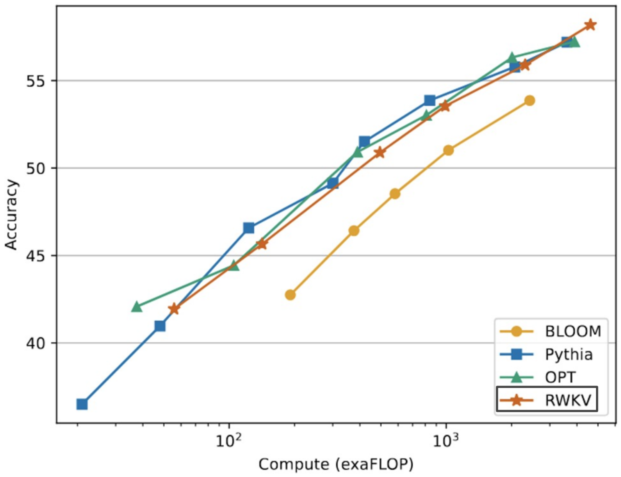
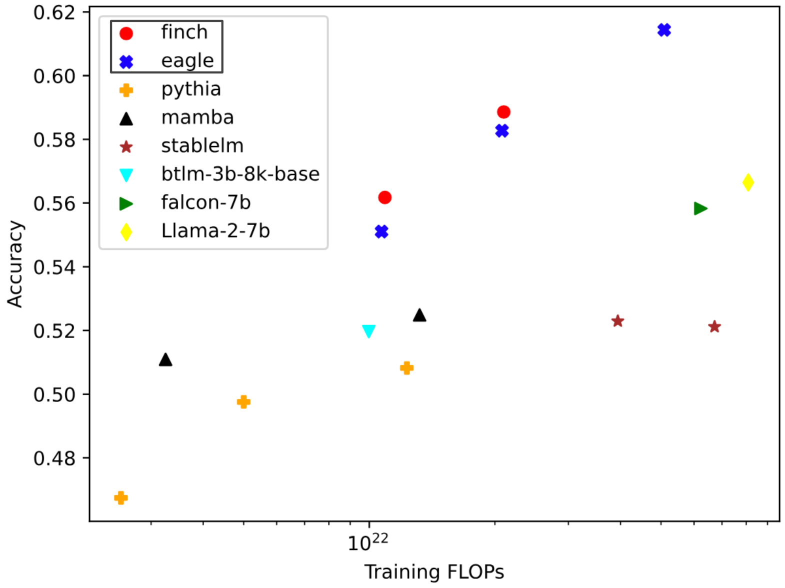
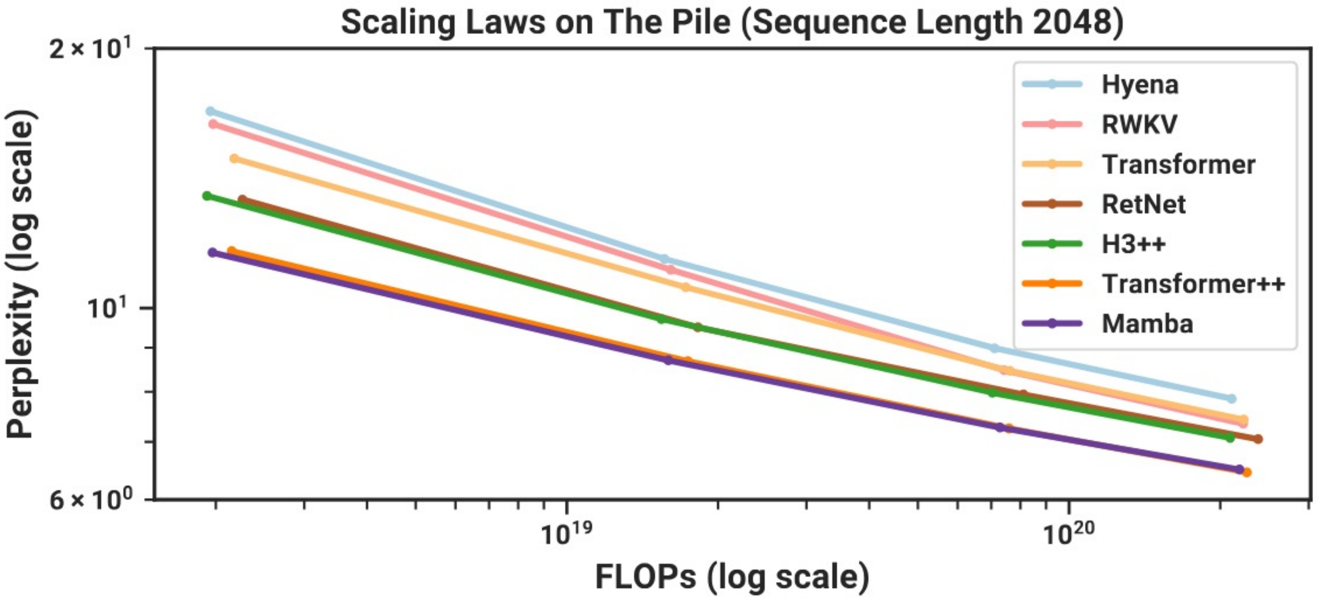


Figure 1: Average performance of RWKV models compared to transformers across twelve NLP tasks. For further details, see section 5.



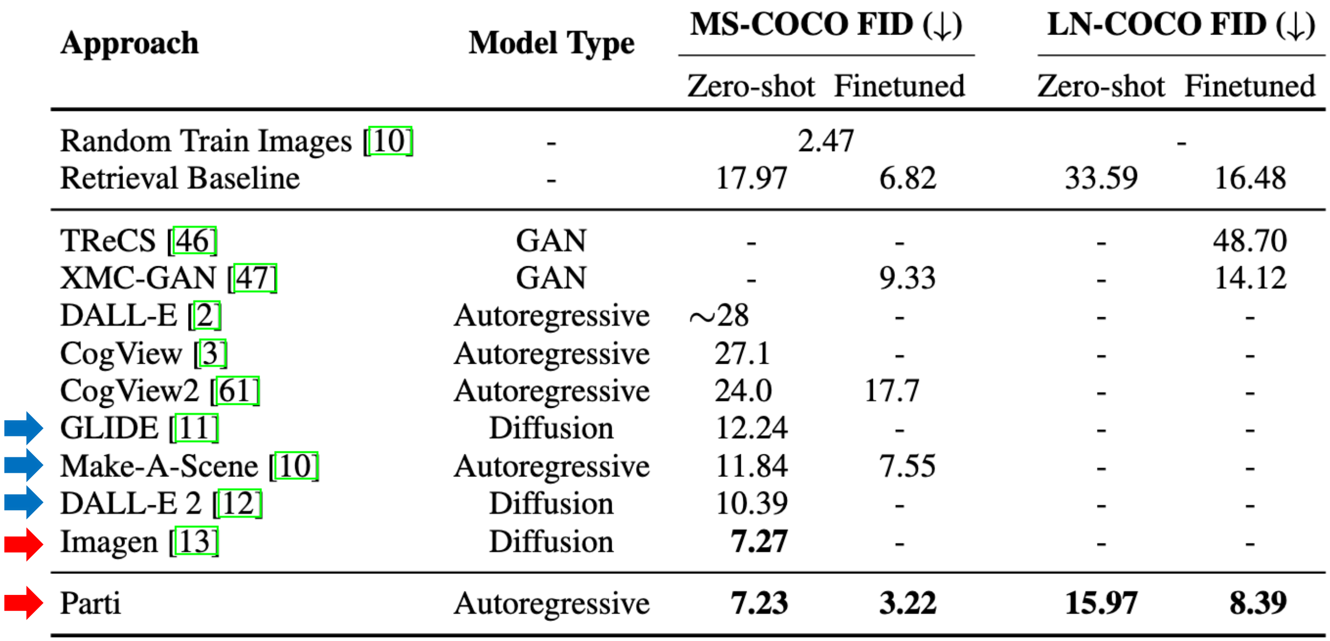
Two RWKV papers, Peng et al. (2023) and Peng et al. (2024)

So does Mamba



“Mamba: Linear-Time Sequence Modeling with Selective State Spaces” Gu and Dao (2024)

The same is true in image generation



Approach	Model Type	MS-COCO FID (↓)		LN-COCO FID (↓)	
		Zero-shot	Finetuned	Zero-shot	Finetuned
Random Train Images [10]	-	2.47		-	
Retrieval Baseline	-	17.97	6.82	33.59	16.48
TReCS [46]	GAN	-	-	-	48.70
XMC-GAN [47]	GAN	-	9.33	-	14.12
DALL-E [2]	Autoregressive	∼28	-	-	-
CogView [3]	Autoregressive	27.1	-	-	-
CogView2 [61]	Autoregressive	24.0	17.7	-	-
➡ GLIDE [11]	Diffusion	12.24	-	-	-
➡ Make-A-Scene [10]	Autoregressive	11.84	7.55	-	-
➡ DALL-E 2 [12]	Diffusion	10.39	-	-	-
➡ Imagen [13]	Diffusion	**7.27**	-	-	-
➡ Parti	Autoregressive	**7.23**	**3.22**	**15.97**	**8.39**

“Scaling Autoregressive Models for Content-Rich Text-to-Image Generation” Yu et al. (2022)

The bottom line

Are transformers the best architecture for a wide variety of domains, applications, and modalities? *Yes*

Can comparable advances be achieved in previous architectures? *Yes*

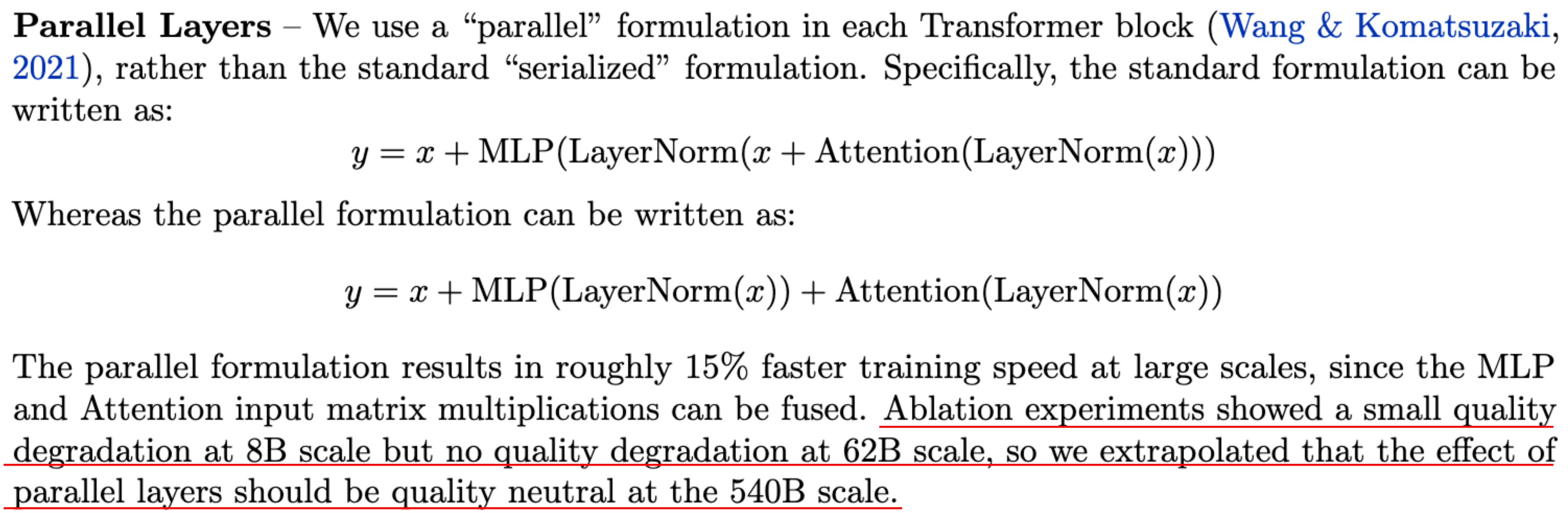
The most magic thing about a transformer is that it is the most efficient way to convert compute into useful work that we know of

A strange conclusion

In the long run (lots of data, lots of compute, lots of parameters) most architectural innovations designed to *improve performance* are irrelevant.

But architectural innovations designed to *improve throughput* are impactful.

“Worse” methods can even be better!



Parallel Layers – We use a “parallel” formulation in each Transformer block (Wang & Komatsuzaki, 2021), rather than the standard “serialized” formulation. Specifically, the standard formulation can be written as:

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

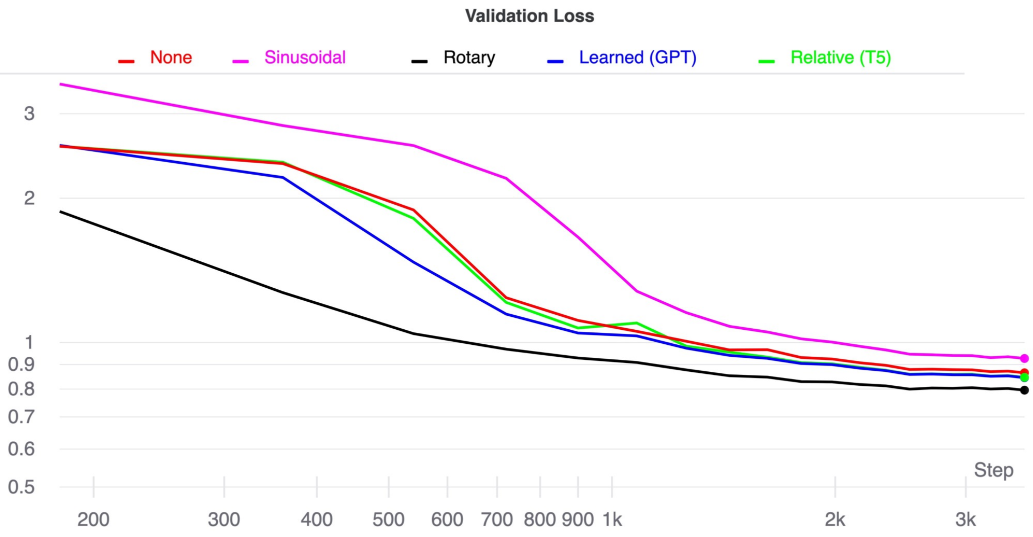
Whereas the parallel formulation can be written as:

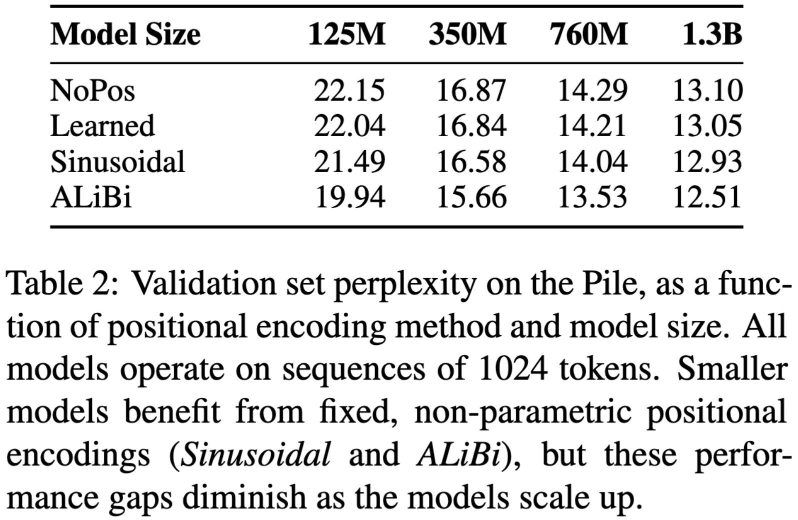
$$y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$$

The parallel formulation results in roughly 15% faster training speed at large scales, since the MLP and Attention input matrix multiplications can be fused. Ablation experiments showed a small quality degradation at 8B scale but no quality degradation at 62B scale, so we extrapolated that the effect of parallel layers should be quality neutral at the 540B scale.

“PaLM: Scaling Language Modeling with Pathways”
Chowdhery et al. (2022)

Details can depend on exact settings





Model Size	125M	350M	760M	1.3B
NoPos	22.15	16.87	14.29	13.10
Learned	22.04	16.84	14.21	13.05
Sinusoidal	21.49	16.58	14.04	12.93
ALiBi	19.94	15.66	13.53	12.51

Table 2: Validation set perplexity on the Pile, as a function of positional encoding method and model size. All models operate on sequences of 1024 tokens. Smaller models benefit from fixed, non-parametric positional encodings (*Sinusoidal* and *ALiBi*), but these performance gaps diminish as the models scale up.

Left: Biderman (2021) trains 350M models on Enwik8

Right: Haviv (2022) trains various sized models on the Pile

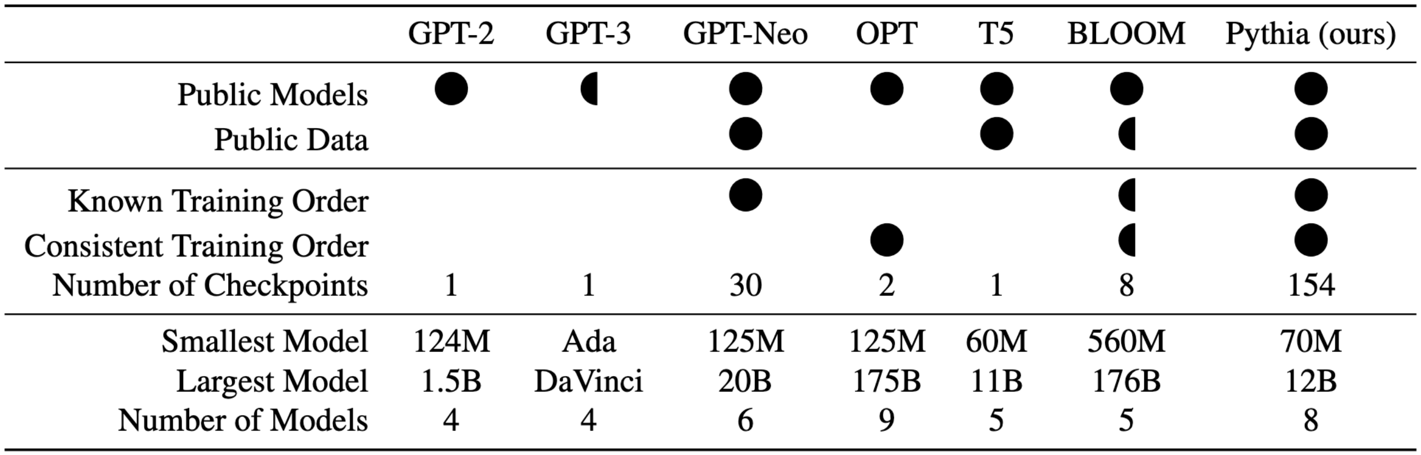
Changing research questions

Model design has become a lot less interesting, but other areas have new questions. For any form of analysis, we can ask the question “how do the results vary across scales?”

In particular, can we predict results for big models before we train them?

I am also very interested in how result vary *across training*

Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling (Biderman et al., 2023)



	GPT-2	GPT-3	GPT-Neo	OPT	T5	BLOOM	Pythia (ours)
Public Models	●	◖	●	●	●	●	●
Public Data			●		●	◖	●
Known Training Order			●			◖	●
Consistent Training Order				●		◖	●
Number of Checkpoints	1	1	30	2	1	8	154
Smallest Model	124M	Ada	125M	125M	60M	560M	70M
Largest Model	1.5B	DaVinci	20B	175B	11B	176B	12B
Number of Models	4	4	6	9	5	5	8

Table 2. Commonly used model suites and how they rate according to our requirements. Further information can be found in Appendix F.1.

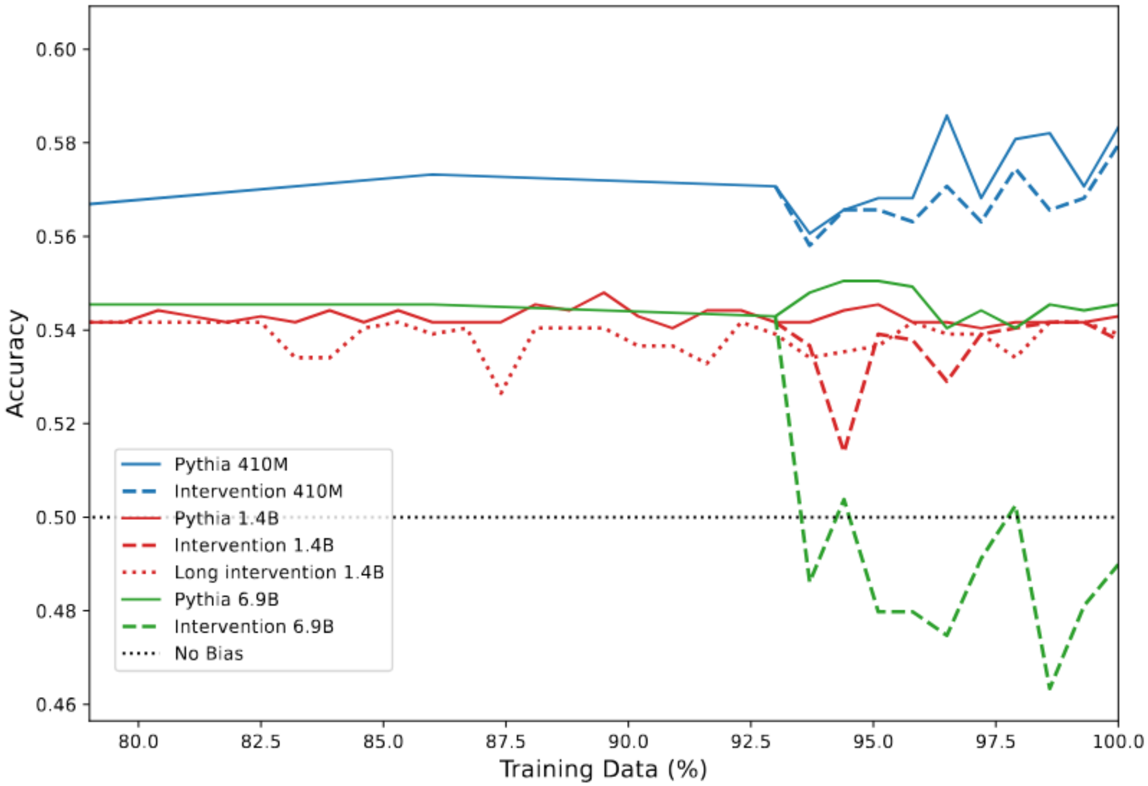
Open question: how do properties evolve over training

We know that models have a tendency to reproduce biases found in data

Let f(x) = y be the function where if x% of the doctors in a training dataset are men, the model will assume a doctor of an unspecified gender is a man y% of the time.

Interventionalist variant: how should I modify a dataset to deliberately end up with a rate of p? What about if I start with a partially trained model?

Open question: how do properties evolve over training



Retrain the last 1/7th with **all feminine pronouns.**

Changes measured gender bias only in the bigger model

Maybe the small models are too converged “in bias space”?

Emergent and Predictable Memorization in Large Language Models (Biderman et al., 2023)

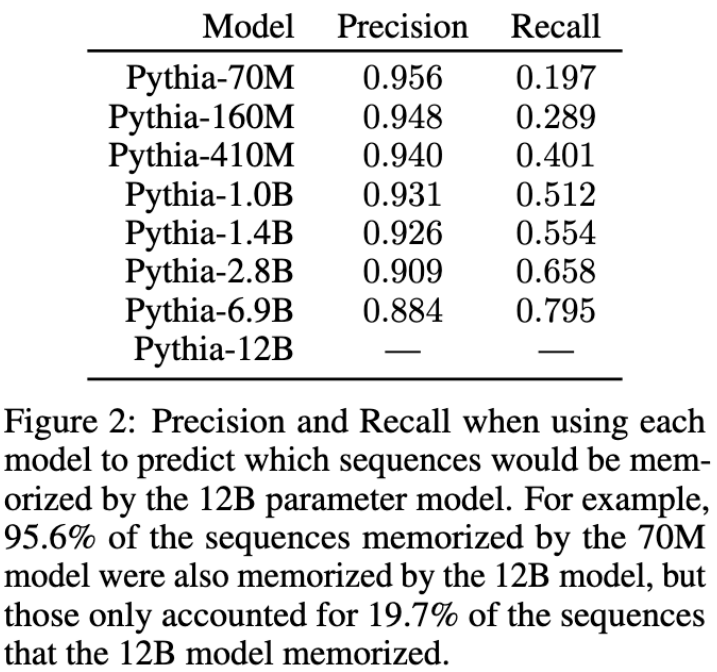
Not all memorization is bad! We want models to memorize

- 2+2=
- George Washington was the first president of the
- Smallpox was an infectious disease caused by

Memorization is bad *for certain types of data*

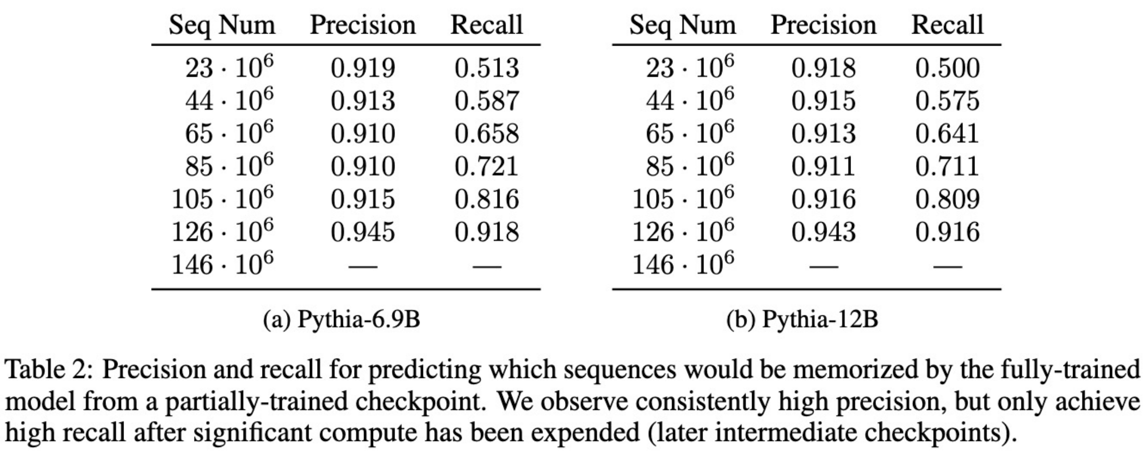
If we can predict what data will be memorized *before training a model* we can use that to inform the decision to train the model

Emergent and Predictable Memorization in Large Language Models (Biderman et al., 2023)



Model	Precision	Recall
Pythia-70M	0.956	0.197
Pythia-160M	0.948	0.289
Pythia-410M	0.940	0.401
Pythia-1.0B	0.931	0.512
Pythia-1.4B	0.926	0.554
Pythia-2.8B	0.909	0.658
Pythia-6.9B	0.884	0.795
Pythia-12B	—	—

Figure 2: Precision and Recall when using each model to predict which sequences would be memorized by the 12B parameter model. For example, 95.6% of the sequences memorized by the 70M model were also memorized by the 12B model, but those only accounted for 19.7% of the sequences that the 12B model memorized.



Seq Num	Precision	Recall
$23 \cdot 10^6$	0.919	0.513
$44 \cdot 10^6$	0.913	0.587
$65 \cdot 10^6$	0.910	0.658
$85 \cdot 10^6$	0.910	0.721
$105 \cdot 10^6$	0.915	0.816
$126 \cdot 10^6$	0.945	0.918
$146 \cdot 10^6$	—	—

(a) Pythia-6.9B

Seq Num	Precision	Recall
$23 \cdot 10^6$	0.918	0.500
$44 \cdot 10^6$	0.915	0.575
$65 \cdot 10^6$	0.913	0.641
$85 \cdot 10^6$	0.911	0.711
$105 \cdot 10^6$	0.916	0.809
$126 \cdot 10^6$	0.943	0.916
$146 \cdot 10^6$	—	—

(b) Pythia-12B

Table 2: Precision and recall for predicting which sequences would be memorized by the fully-trained model from a partially-trained checkpoint. We observe consistently high precision, but only achieve high recall after significant compute has been expended (later intermediate checkpoints).

Open question: does order matter in memorization?

My research say it doesn’t.

“Causal Estimation of Memorisation Profiles” (Lesci et al., 2024) says it does.

The primary disagreement is about statistical models and how we should measure the answer to this question. We already have the data and just need the right analysis.

LLM Circuit Analyses Are Consistent Across Training and Scale (Tigges et al., 2024)

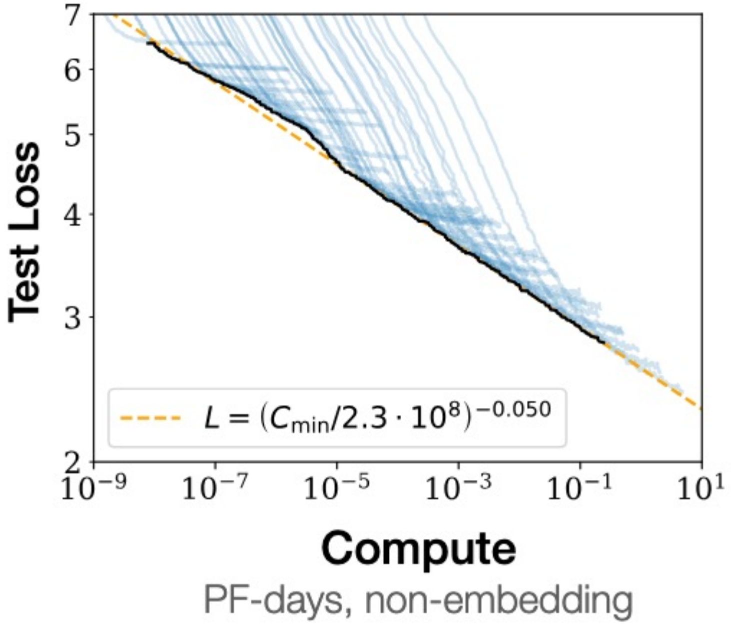
Investigates how stable circuit analysis is across training and scaling.

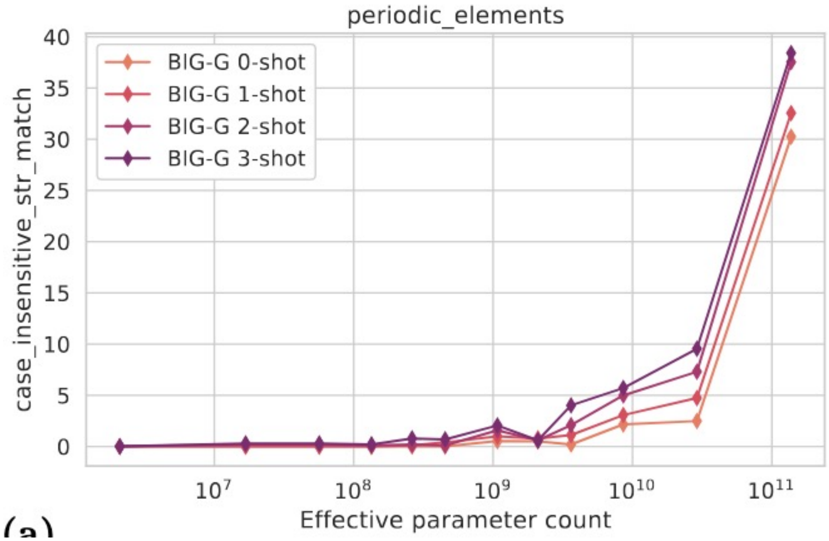
Result #1: Critical components emerge at similar token counts across scales

Result #2: Which neurons involved in the task changes over training.

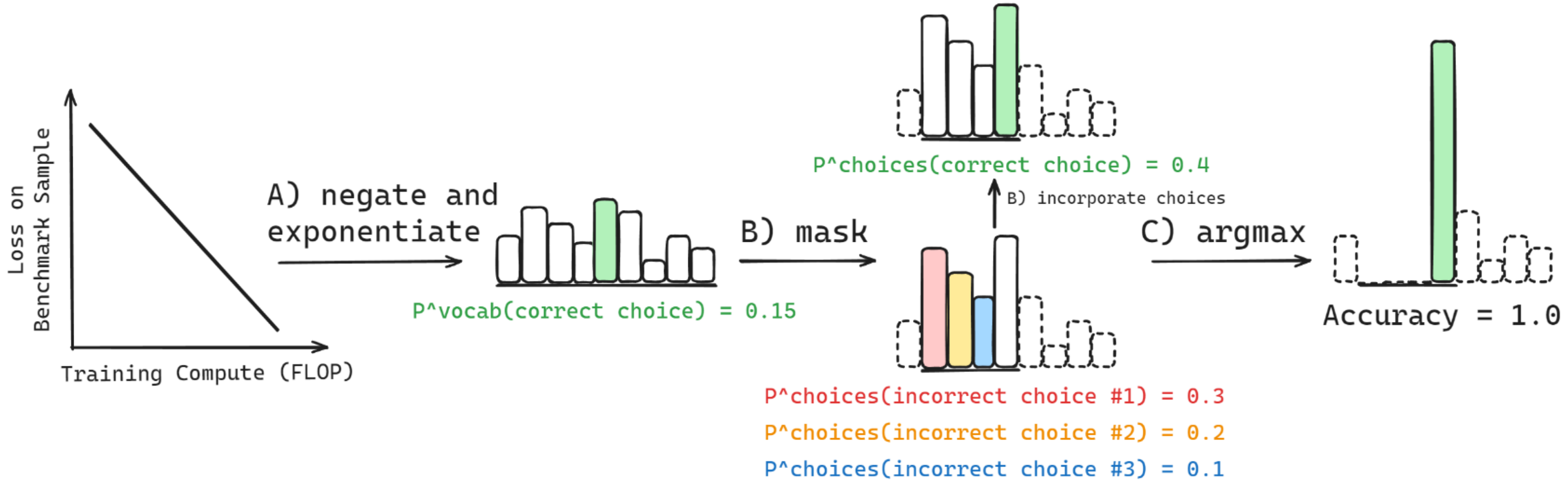
Result #3: Despite #2, the algorithm the model learns is stable over training.

Predicting Downstream Capabilities (Schaeffer et al., 2024)



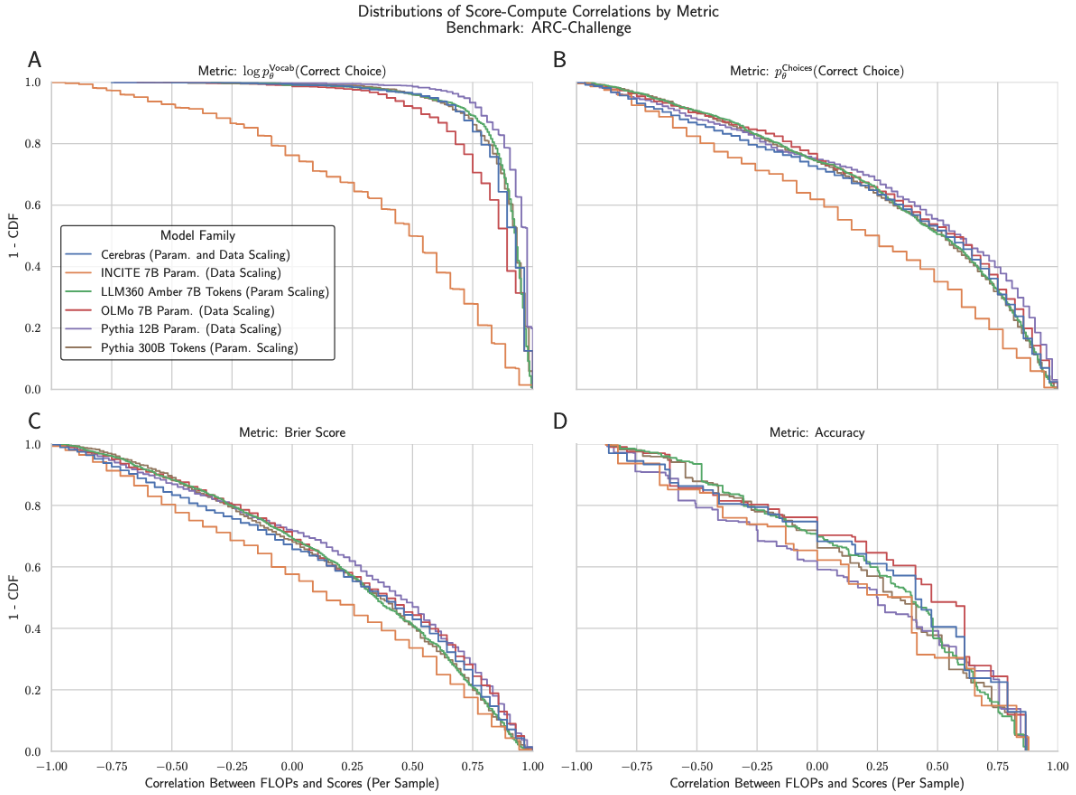


Predicting Downstream Capabilities (Schaeffer et al., 2024)


Benchmark Sample Loss on
Training Compute (FLOP)
A) negate and exponentiate
P^vocab(correct choice) = 0.15
B) mask
P^choices(correct choice) = 0.4
B) incorporate choices
P^choices(incorrect choice #1) = 0.3
P^choices(incorrect choice #2) = 0.2
P^choices(incorrect choice #3) = 0.1
C) argmax
Accuracy = 1.0

Builds on Schaeffer et al. (2023)

Predicting Downstream Capabilities (Schaeffer et al., 2024)



Answer: everywhere!

Schaeffer et al. (2023)’s suggestion that continuous metrics solve the problem isn’t universal

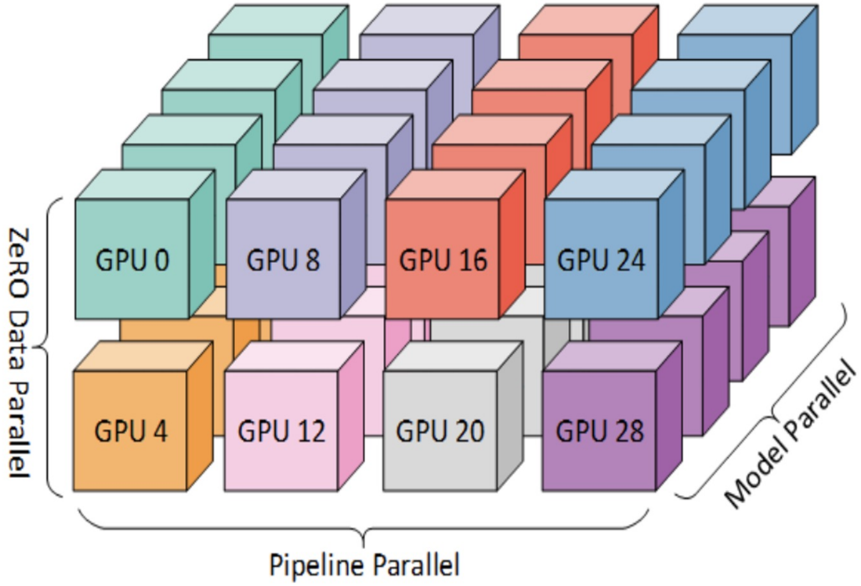
Back-Up

How we parallelize large scale models

Pipeline parallelism

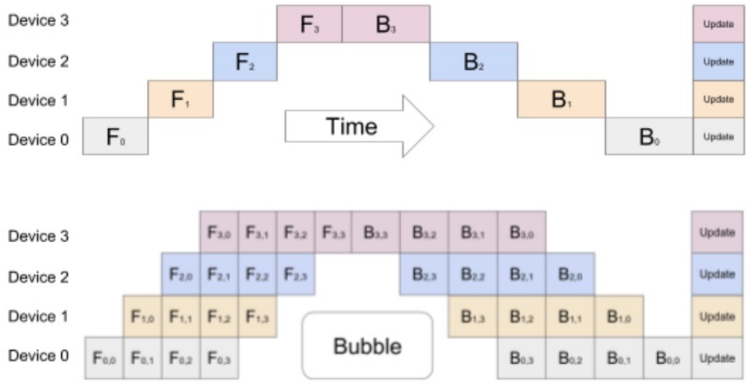
Tensor parallelism

Data parallelism



Pipeline parallelism

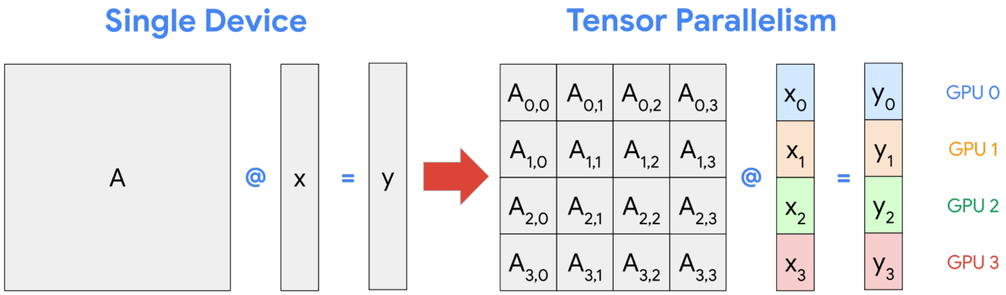
Pipeline parallelism: Splitting different layers across different workers


Device 3
F_3
B_3
Update
Device 2
F_2
B_2
Update
Device 1
F_1
B_1
Update
Device 0
F_0
Time
B_0
Update
Device 3
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
Update
Device 2
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
Update
Device 1
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
Update
Device 0
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$
Bubble
$B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$
Update

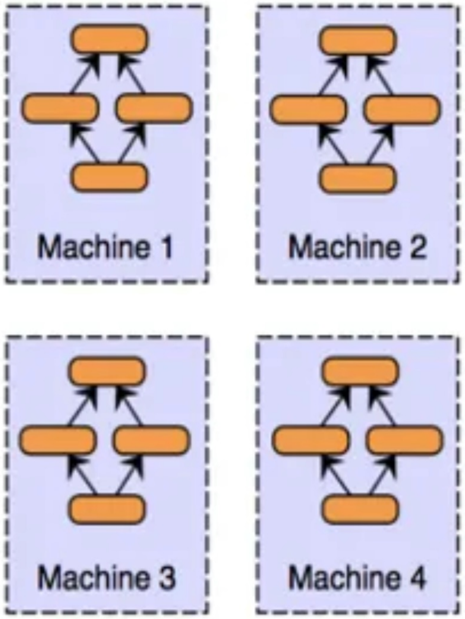
***Top:** The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. **Bottom:** GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.*

Tensor parallelism

Tensor parallelism: breaking a single layer among multiple workers


Single Device
A
@
x
=
y
Tensor Parallelism
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$ $A_{0,3}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$ $A_{1,3}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$ $A_{2,3}$
$A_{3,0}$ $A_{3,1}$ $A_{3,2}$ $A_{3,3}$
@
x_0
x_1
x_2
x_3
=
y_0
y_1
y_2
y_3
GPU 0
GPU 1
GPU 2
GPU 3

Data parallelism



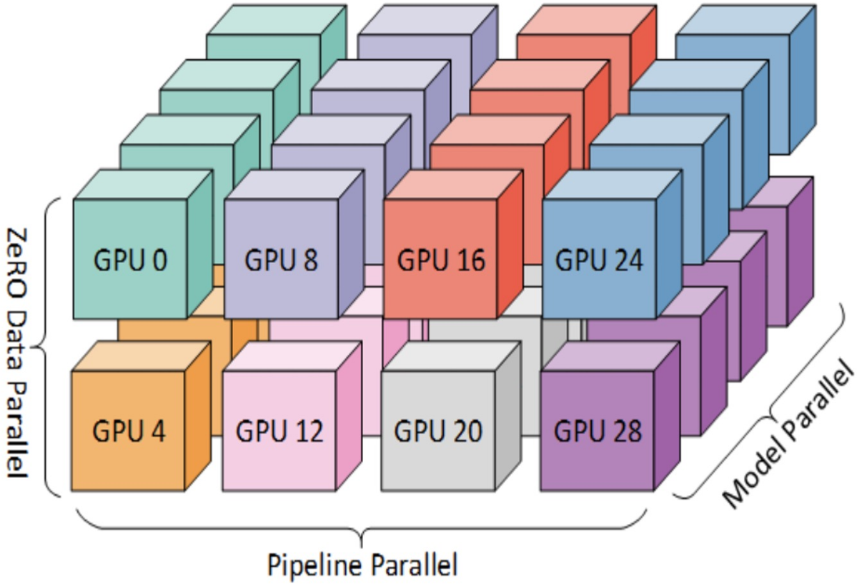
Data parallelism: Replicating the entire model pipeline with different input data

How we parallelize large scale models

Pipeline parallelism: Splitting different layers across different workers

Tensor parallelism: breaking a single layer among multiple workers

Data parallelism: Replicating the entire model pipeline with different input data



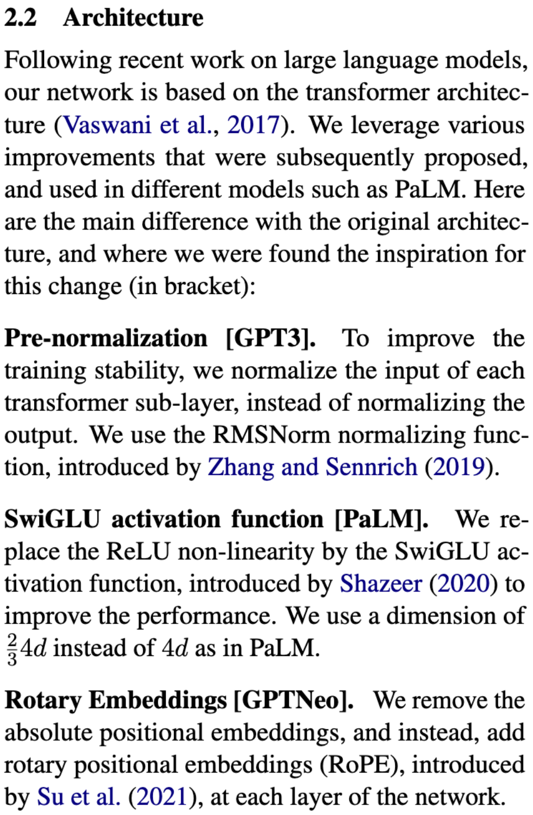
Are you irrelevant in the design of today’s models?

Myth: academic and small-scale researchers are irrelevant in modern model-design research

Reality: a substantial amount of innovation in model design

Caveat: What matters in model design has changed substantially, and most academics are not being educated in what’s important (HPC)

Where did LLaMA’s innovations come from?



2.2 Architecture

Following recent work on large language models, our network is based on the transformer architecture (Vaswani et al., 2017). We leverage various improvements that were subsequently proposed, and used in different models such as PaLM. Here are the main difference with the original architecture, and where we were found the inspiration for this change (in bracket):

Pre-normalization [GPT3]. To improve the training stability, we normalize the input of each transformer sub-layer, instead of normalizing the output. We use the RMSNorm normalizing function, introduced by Zhang and Sennrich (2019).

SwiGLU activation function [PaLM]. We replace the ReLU non-linearity by the SwiGLU activation function, introduced by Shazeer (2020) to improve the performance. We use a dimension of $\frac{2}{3}4d$ instead of $4d$ as in PaLM.

Rotary Embeddings [GPTNeo]. We remove the absolute positional embeddings, and instead, add rotary positional embeddings (RoPE), introduced by Su et al. (2021), at each layer of the network.

Inspirations: OpenAI, Google, and EleutherAI

Invention: Two pre-LLMs and a “GPU-poor” Chinese start-up

Pathways Language Model (PaLM)

SwiGLU Activation – Noam Shazeer

Parallel Layers – **EleutherAI (non-profit)**

Multi-Query Attention – Noam Shazeer

RoPE Embeddings – **Zhuiyi Technology Co (small start-up)**

Shared Input-Output Embeddings – Several independent inventors

No biases – Google (novel in this paper)

Vocabulary – Google's novel variant on **EleutherAI (non-profit)**

Some Innovations by the “GPU Poor”

Parallel Layers – EleutherAI

RoPE Embeddings – Zhuiyi Technology Co

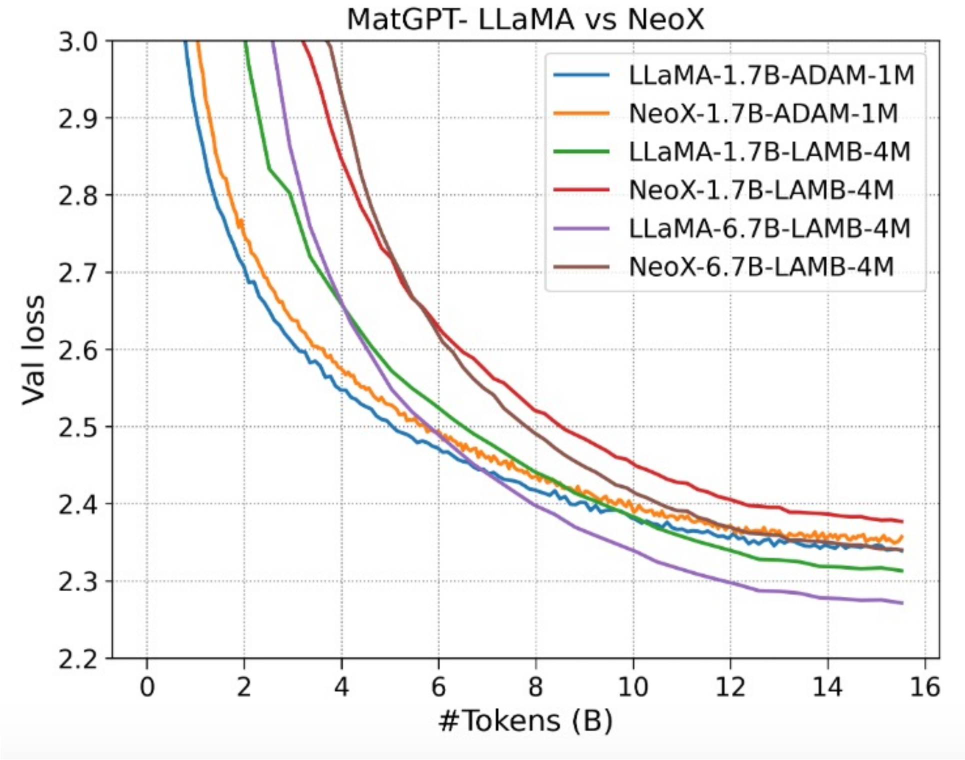
RoPE Extension – Simultaneous (Nous + Indie devs + EleutherAI) and Meta

Alibi Embeddings – University of Washington

Flash Attention – Stanford University

The most popular open source library for training LLMs
at scale is EleutherAI’s, not NVIDIA’s or Meta’s

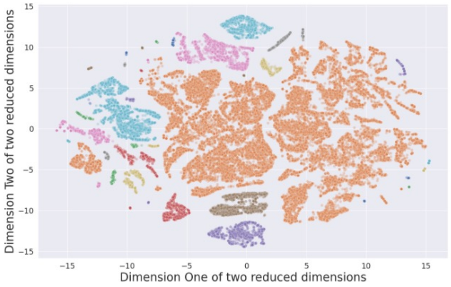
LLaMA vs GPT-NeoX architectures



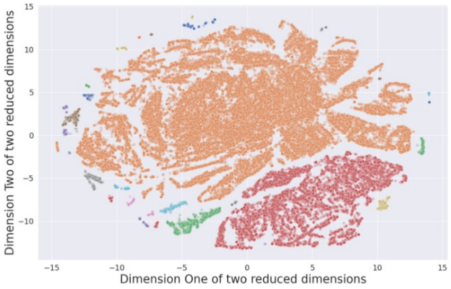
It’s very unclear what model architecture is best. Some studies have shown that LLaMA-style is superior to GPT-NeoX-style architectures when looking at loss…

Yin et al. (2024)

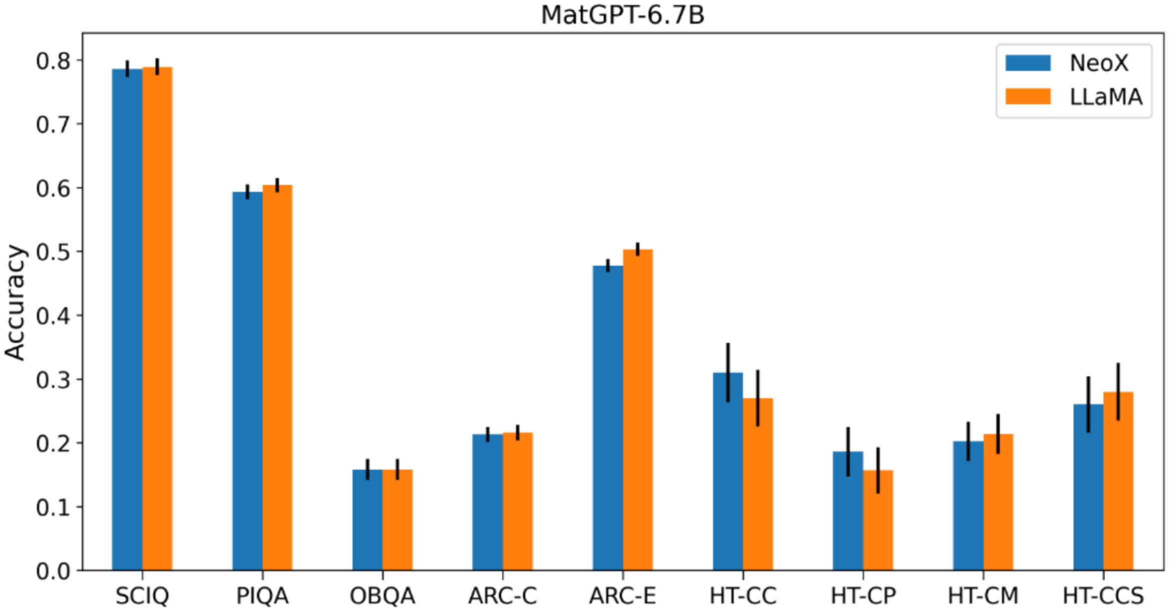
LLaMA vs GPT-NeoX architectures



(e) MatGPT-NeoX 6.7B embedding clustering; Hidden size=4096



(f) MatGPT-LLaMA 6.7B embedding clustering; Hidden size=4096



But those gains **go away completely** when looking at downstream perf