Lecture 8: Language Model

Berkeley CS294-158 Deep Unsupervised Learning
Spring 2024

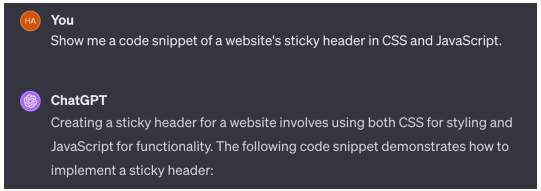
Hao Liu

Outline

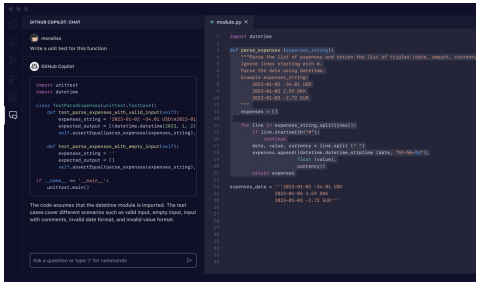
- Methodologies behind large language models
  - Basics
  - Bottlenecks and solutions
  - Scaling laws
  - Capabilities
- How to train your large language models
  - Sharding

Successes of machine and deep learning

- Language model is everywhere
- Secret sauce: enormous compute power

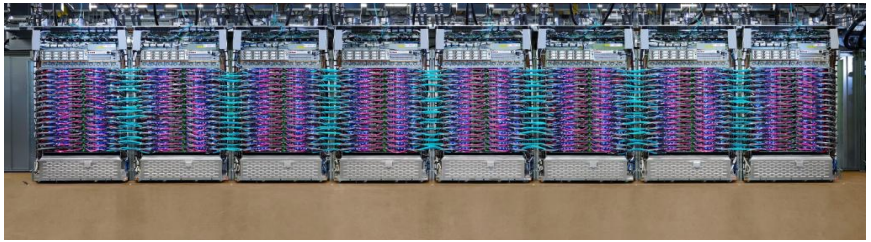


Chatbot



Copilot

Sora



TPU datacenter

Language model

- A language model is a probability distribution over sequences:
  - Likelihood distribution $p_\theta(y)$
  - Sequence of tokens $y = (y_1, \ldots, y_T)$
  - Parameters: θ

Language model

- Typical language models are autoregressive:

$$p_\theta(y) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{<t})$$

- Model is parameterized by a transformer

Autoregressive

- Autoregressive factorization
  - Next token prediction
  - Predicting future
- Every bit becomes supervision
- Natural for conversational AI

Modeling likelihood

- Maximum likelihood: make observed data likely under the model

$$\arg\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathrm{y} \in \mathcal{D}} \log p_\theta(\mathrm{y})$$

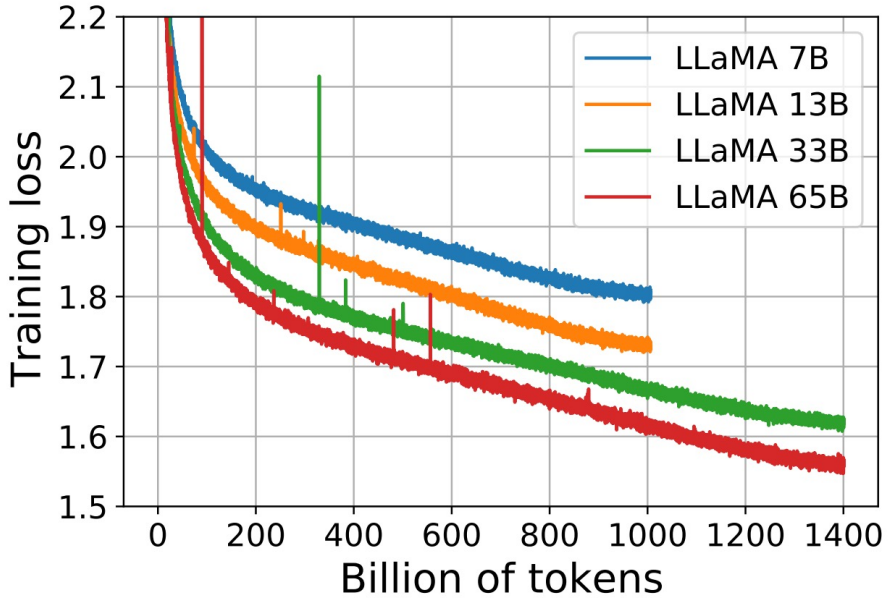
- D: 1.5 trillions of tokens in Llama
- θ: 7 to 70 billions of parameters

Learning

- **Pretraining**: large-scale unsupervised learning
- **Finetuning**: specializing the model for specific tasks
- **Learning from feedback**: improving the model with feedback on its outputs, such as human feedback and debug message

Why unsupervised learning?

- Much more data available
- Much better generalization



Scaling compute

- We want to model data distribution better by adding more compute.
- "*The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin*." The Bitter Lesson, Richard Sutton 2019

Compute

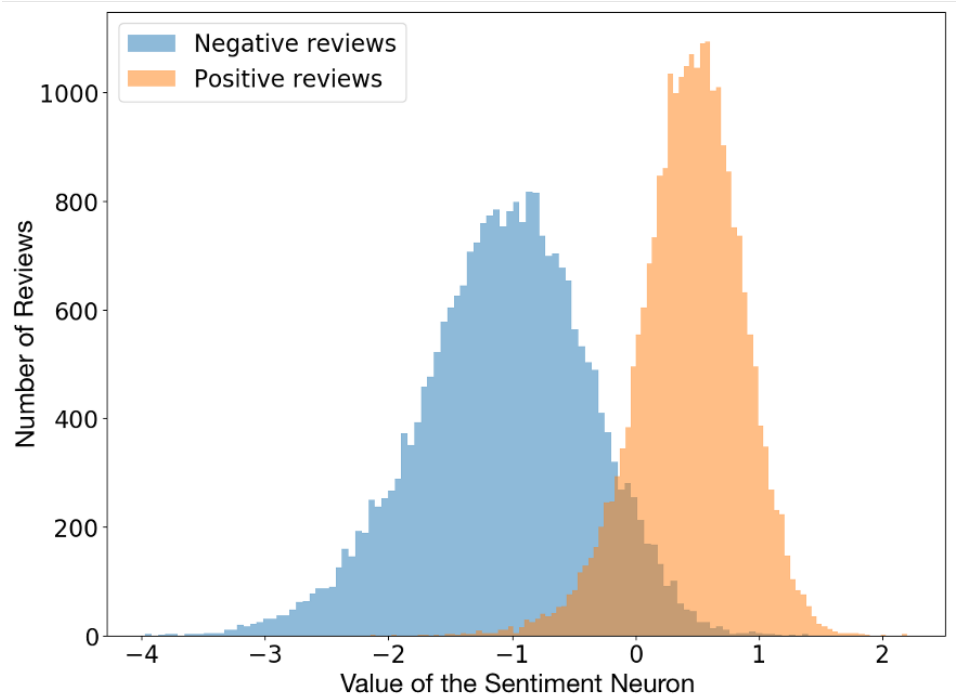
- Compute is forward and backward passes using our model on token sequences.
- Mostly many matrix multiplications (matmul)
  - Unit: floating point operations (FLOPs)
- Compute cost: $C = 6ND(1+s/6d)$
  - N: parameters, D: data size, s: context size, d: hidden dimension
- Adding compute by using more tokens, larger context, larger model

Token

- Byte-based: most general but too long
- Character-based: each word requires too many tokens
- Word-based: `dog' and `dogs' should share meaning
- Subword-based:
  - Byte-pair encoding: replace top appearing pair with a new token
  - Repeat until reach a given vocab size

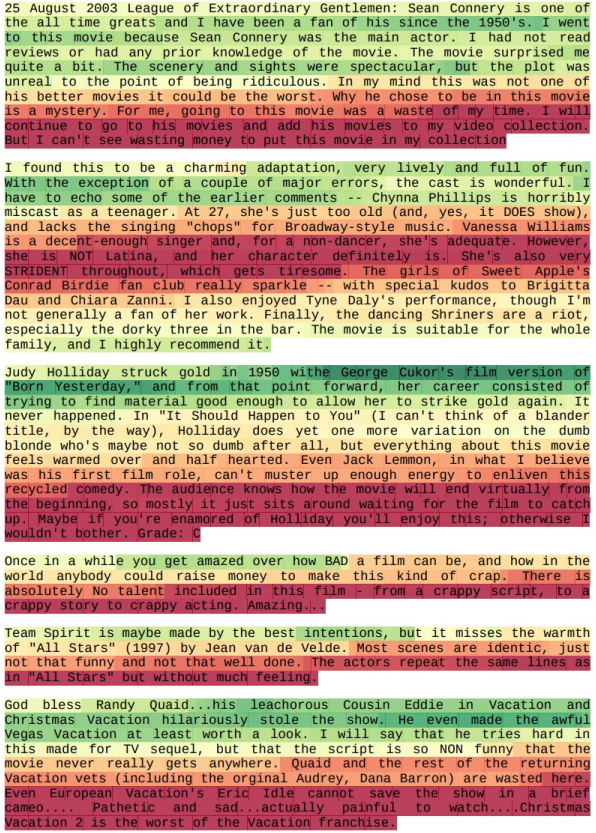
Train LSTM with more compute

- **LSTM** learns a sentiment neuron after training to predict the next word on a large amount of Amazon reviews.



Radford, Alec, Rafal Jozefowicz, and Ilya Sutskever. "Learning to generate reviews and discovering sentiment." arXiv 2017.

Train LSTM with more compute



- Visualizing the value of the sentiment cell as it processes six randomly selected high contrast IMDB reviews. Red indicates negative sentiment while green indicates positive sentiment. Best seen in color.

Radford, Alec, Rafal Jozefowicz, and Ilya Sutskever. "Learning to generate reviews and discovering sentiment." arXiv 2017.

Train LSTM with more compute

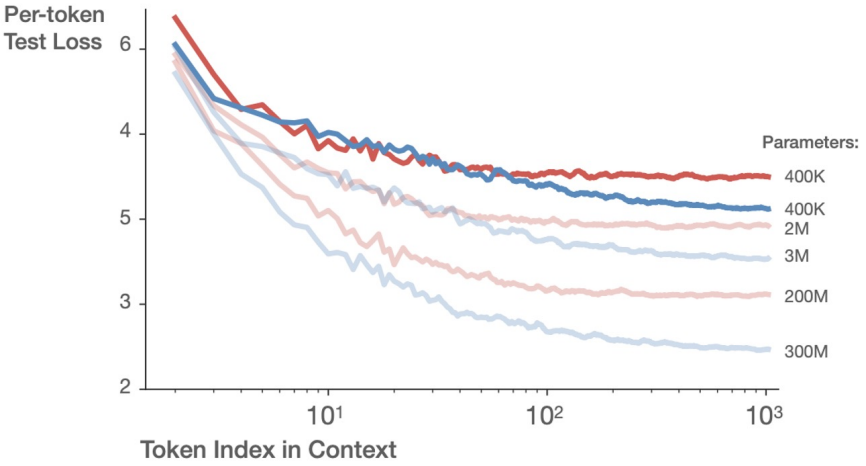
- **LSTM** learns a sentiment neuron after training to predict the next word on a large amount of Amazon reviews.

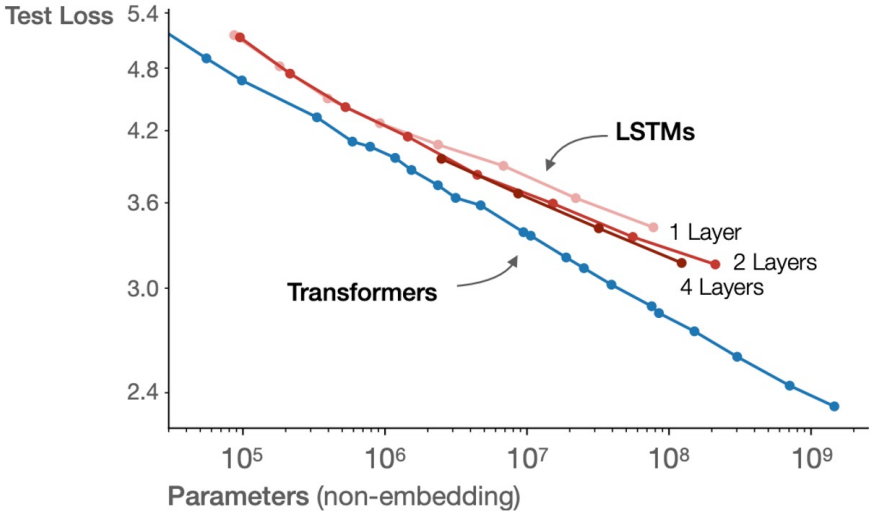
Sentiment fixed to positive	Sentiment fixed to negative
Just what I was looking for. Nice fitted pants, exactly matched seam to color contrast with other pants I own. Highly recommended and also very happy!	The package received was blank and has no barcode. A waste of time and money.
This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord.	Great little item. Hard to put on the crib without some kind of embellishment. My guess is just like the screw kind of attachment I had.
Best hammock ever! Stays in place and holds it's shape. Comfy (I love the deep neon pictures on it), and looks so cute.	They didn't fit either. Straight high sticks at the end. On par with other buds I have. Lesson learned to avoid.
Dixie is getting her Doolittle newsletter we'll see another new one coming out next year. Great stuff. And, here's the contents - information that we hardly know about or forget.	great product but no seller. couldn't ascertain a cause. Broken product. I am a prolific consumer of this company all the time.
I love this weapons look . Like I said beautiful !!! I recommend it to all. Would suggest this to many roleplayers , And I stronge to get them for every one I know. A must watch for any man who love Chess!	Like the cover, Fits good. . However, an annoying rear piece like garbage should be out of this one. I bought this hoping it would help with a huge pull down my back & the black just doesn't stay. Scrap off everytime I use it.... Very disappointed.

Radford, Alec, Rafal Jozefowicz, and Ilya Sutskever. "Learning to generate reviews and discovering sentiment." arXiv 2017.

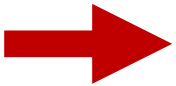
Scaling LSTM is difficult

- **Attention** allows scalably improving models. Transformers asymptotically outperform LSTMs due to improved use of context.





"Better usage of context"

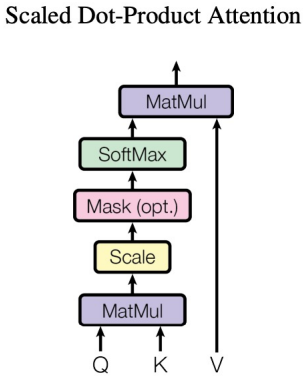


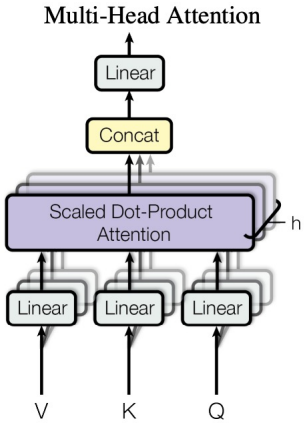
"Better model performance"

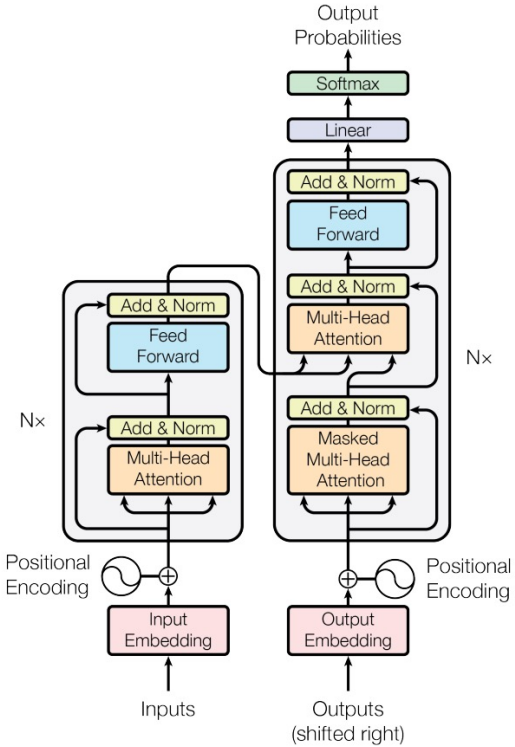
Scaling laws for neural language models. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020)

Transformer architecture

- Attention allows attending to past tokens without forgetting.
- Big FFNs, highly scalable



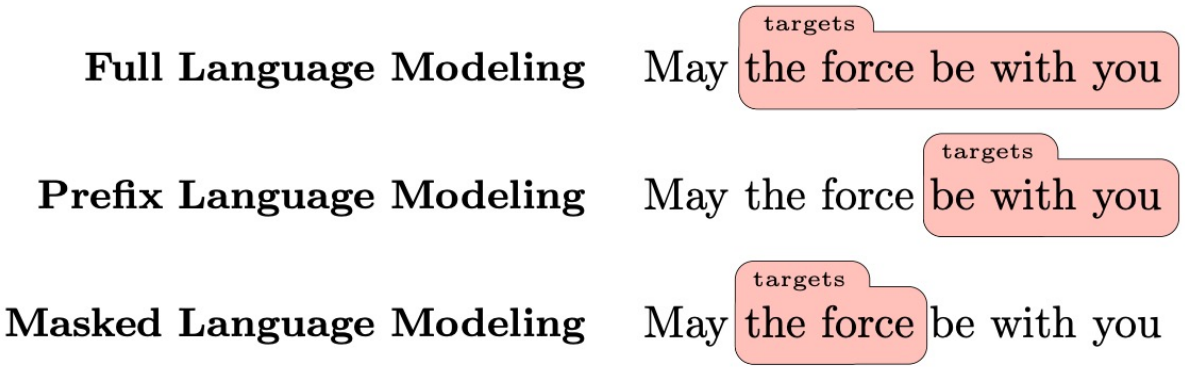




Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Pretraining objective

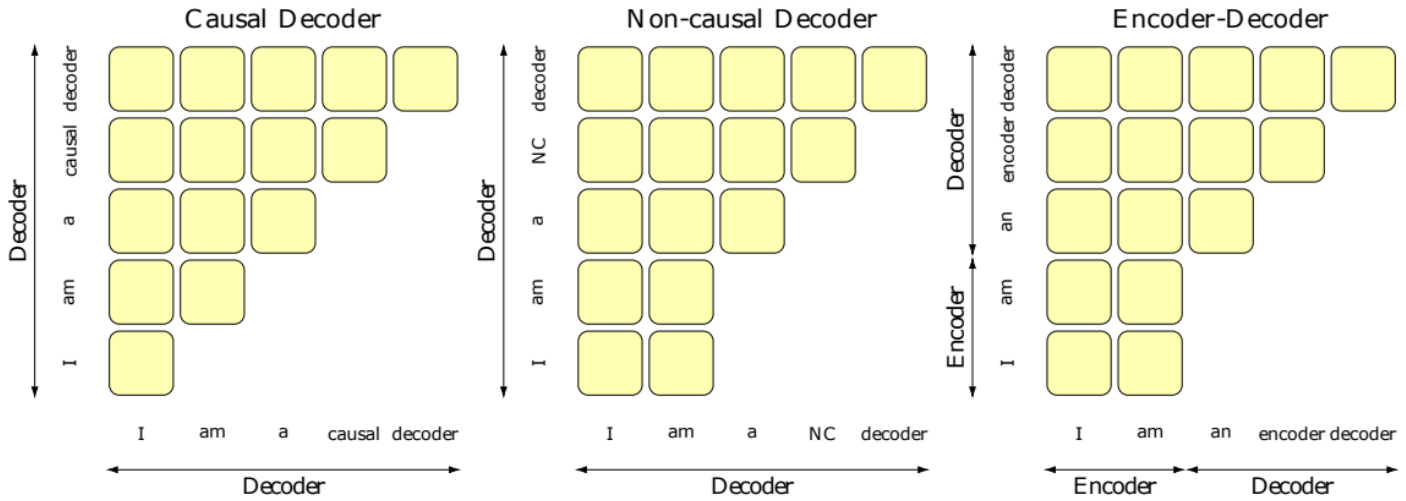
- Full autoregressive: GPT, Llama
- Prefix autoregressive: T5
- Masked: BERT, ELMO



Wang, Thomas, et al. "What language model architecture and pretraining objective works best for zero-shot generalization?." (2022)

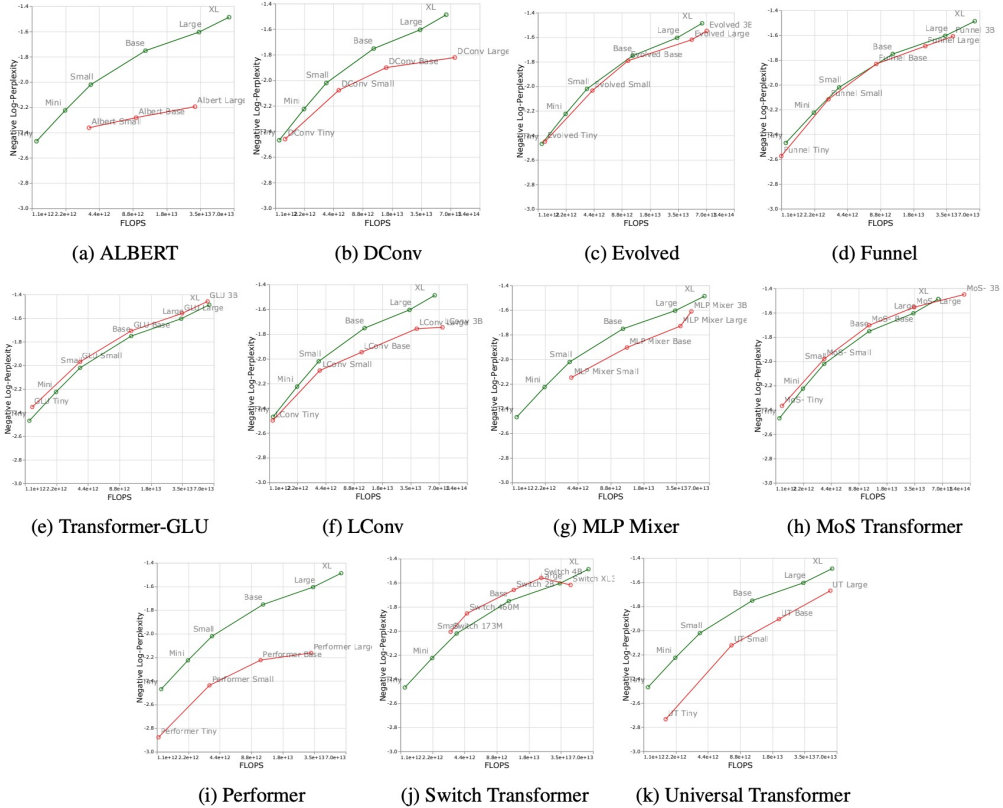
Pretraining objective

- T5, BERT, GPT. All with different masking
- Causal decoder (GPT) used most frequently



Wang, Thomas, et al. "What language model architecture and pretraining objective works best for zero-shot generalization?." (2022)

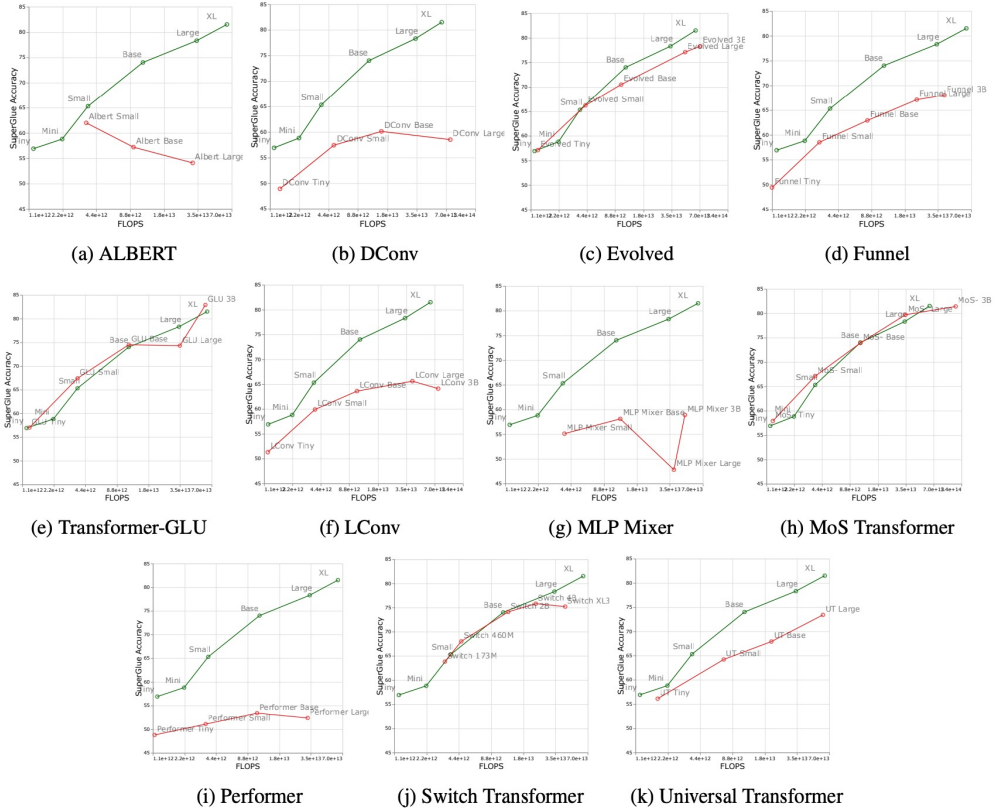
Model architecture



- Upstream Negative Log-Perplexity of vanilla Transformer compared to other models.
- Transformer outperforms other models.

Tay, Y., Dehghani, M., Abnar, S., Chung, H. W., Fedus, W., Rao, J., ... & Metzler, D. Scaling laws vs model architectures: How does inductive bias influence scaling? (2022).

Model architecture



(a) ALBERT (b) DConv (c) Evolved (d) Funnel

(e) Transformer-GLU (f) LConv (g) MLP Mixer (h) MoS Transformer

(i) Performer (j) Switch Transformer (k) Universal Transformer

- Downstream accuracy of vanilla Transformer compared to other models.
- Transformer outperforms other models.

Tay, Y., Dehghani, M., Abnar, S., Chung, H. W., Fedus, W., Rao, J., ... & Metzler, D. Scaling laws vs model architectures: How does inductive bias influence scaling? (2022).

Compute cost

- Parameter count is N, token count is D, what is the compute cost?
  - C = 6ND(1+s/6d): Compute increases with more data and larger model
  - LLaMA (s << 6d): C = 6ND = 6 ∗ 7 billion ∗ 2 trillion = 8.4 × 10^22 FLOPs

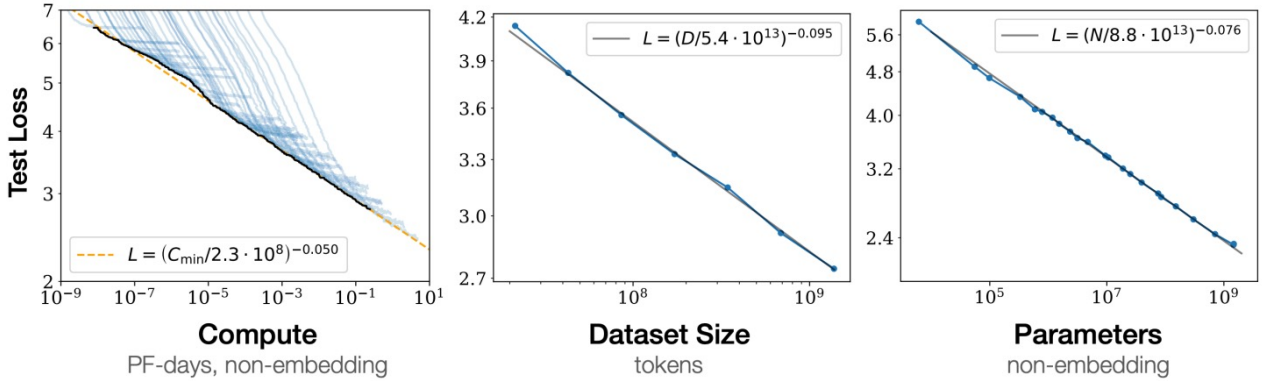
- Shall I allocate more compute to data or model?

$$N_{opt}(C), D_{opt}(C) = \underset{N,D \text{ s.t. } \mathrm{FLOPs}(N,D)=C}{\operatorname{argmin}} L(N, D)$$

"Scaling laws for neural language models." (2020)

Optimal token and parameter

- Empirical performance has a power-law relationship with each individual factor.
  - Train model of different sizes and number of tokens (light blue lines)
  - Pick minimal loss (black line)
  - Run linear regression on log – log
- Follow power-law: $N_{opt} \propto C^a, D_{opt} \propto C^b$



"Scaling laws for neural language models." (2020)

Allocate compute

- Empirical performance: $N_{opt} \propto C^a, D_{opt} \propto C^b$
- We have a + b = 1 because C = 6ND

Allocate more compute to parameters (a) or tokens (b)?

Allocate compute

- Coefficients of model scaling and data scaling vary with training data distribution

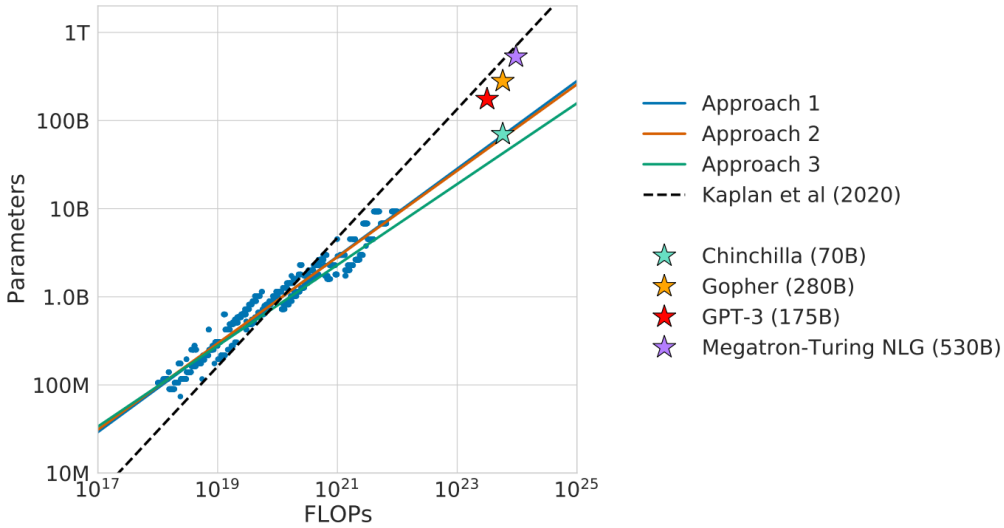
Approach	Coeff. a where $N_{\text{opt}}(M_{\text{opt}}) \propto C^a$	Coeff. b where $D_{\text{opt}} \propto C^b$
OpenAI (OpenWebText2)	0.73	0.27
Chinchilla (MassiveText)	0.49	0.51
Ours (Early Data)	0.450	0.550
Ours (Current Data)	0.524	0.476
Ours (OpenWebText2)	0.578	0.422

- OpenAI (2020) gives more compute to parameters
- DeepMind (2022) gives more compute to tokens
- Best practice: train different small models and data sizes, and fit the constants yourself.

"DeepSeek LLM Scaling Open-Source Language Models with Longtermism." (2024)

Chinchilla scaling

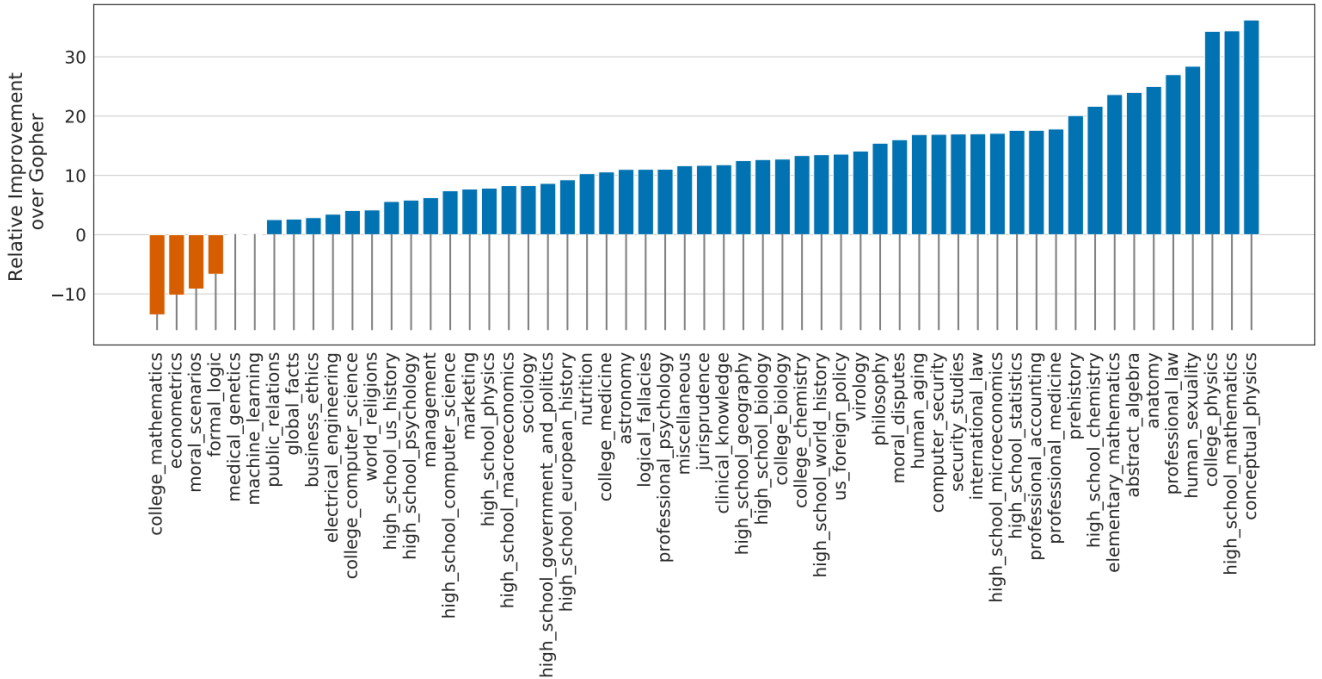
- Chinchilla scaling law: a = 0.49, b=0.51
- Implication: more compute efficient than other models



"Training Compute-Optimal Large Language Models." (2022).

Chinchilla scaling

- Chinchilla outperforms Gropher significantly by allocating compute better



"Training Compute-Optimal Large Language Models." (2022).

Determine tokens

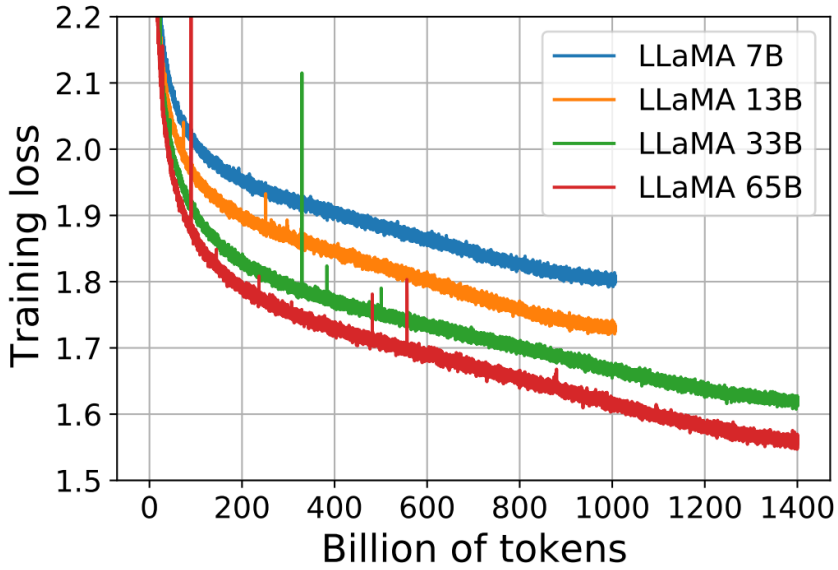
- Estimated optimal training tokens: about 20 times number of parameters
- It's best to double the estimate: tokens = 40x parameters

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

"Training Compute-Optimal Large Language Models." (2022).

Inference optimal

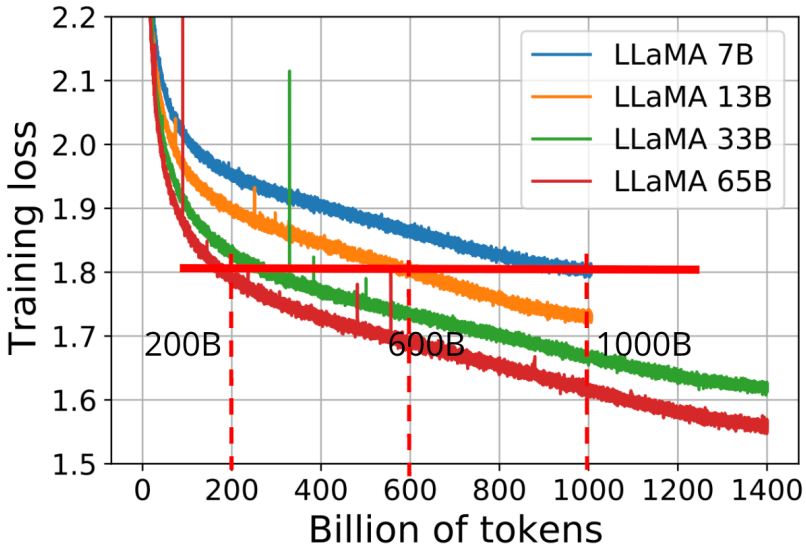
- Train small model and more tokens to achieve better inference
  - E.g. Llama 7b is trained on 7 times of Chinchilla optimal

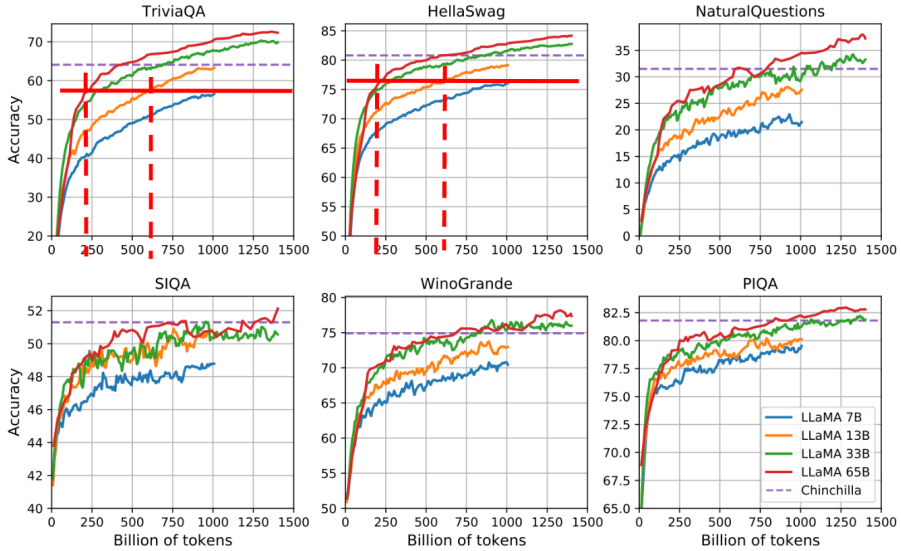


Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." (2023).

Loss predicts performance

- Loss determines downstream performance. Both large and small models have the same performance if they have the same loss.

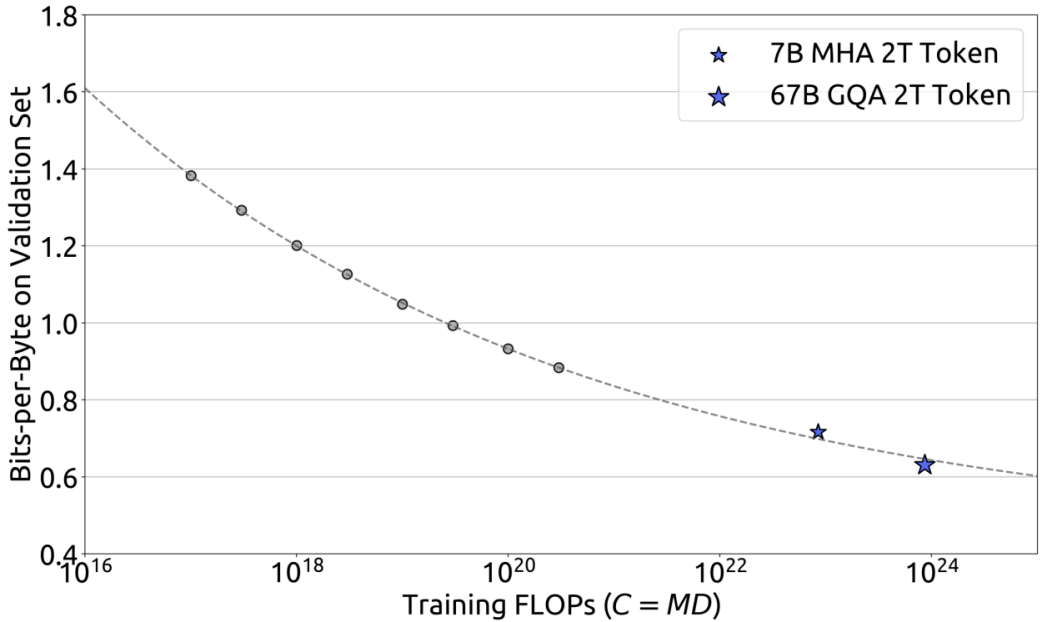




Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." (2023).

Scaling law prediction

- Scaling law can predict performance at larger scale
  - Predicting performance of larger models and larger datasets

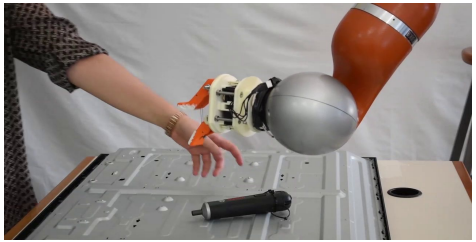


"DeepSeek LLM Scaling Open-Source Language Models with Longtermism." (2024)

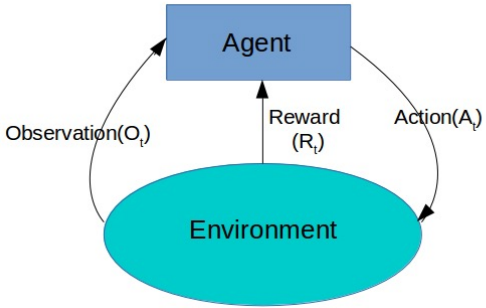
Large context

- Major bottleneck: cannot even fit complex, long sequences into Transformers. Needed for agent, world modeling, codebase, and hyperlinked web.

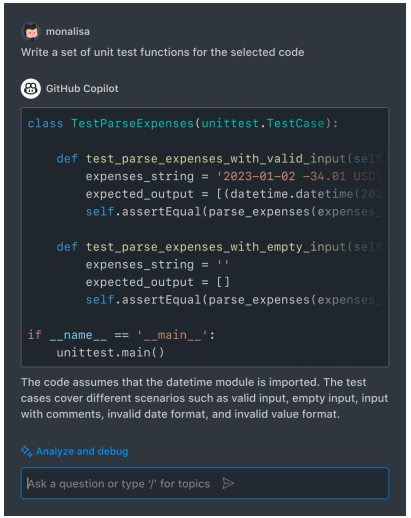
World



Agent

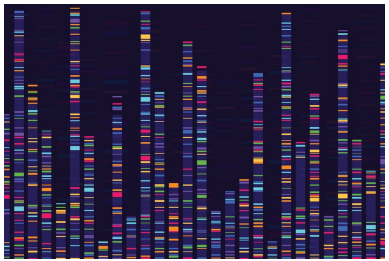


Codebase



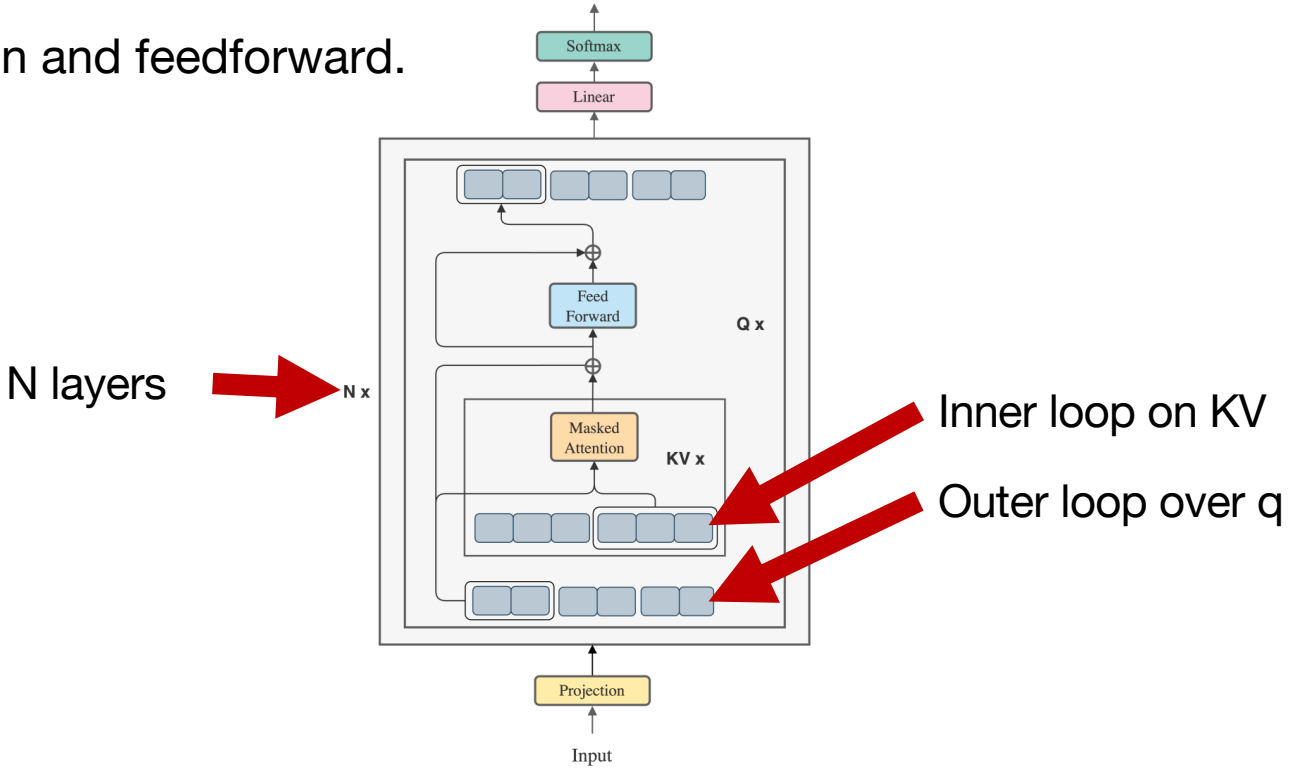
Hyperlinked web

Genome



Blockwise parallel transformers

- Reorganize the computation of attention and feedforward.
- Exact attention and feedforward.



Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.

Analysis of memory cost

- Standard attention + standard FFN
  - Peak of attention: O(s**2)
  - Peak of FFN: 8bsh

→ O(s**2)

- Memory efficient attention + standard FFN
  - Peak of attention: max(4bch, 2bsh) = 2bsh
  - Peak of FFN: 8bsh

→ 8bsh

- BPT
  - Peak of attention: 2bsh
  - Peak of FFN: max(8bch, 2bsh) = 2bsh because s >> c

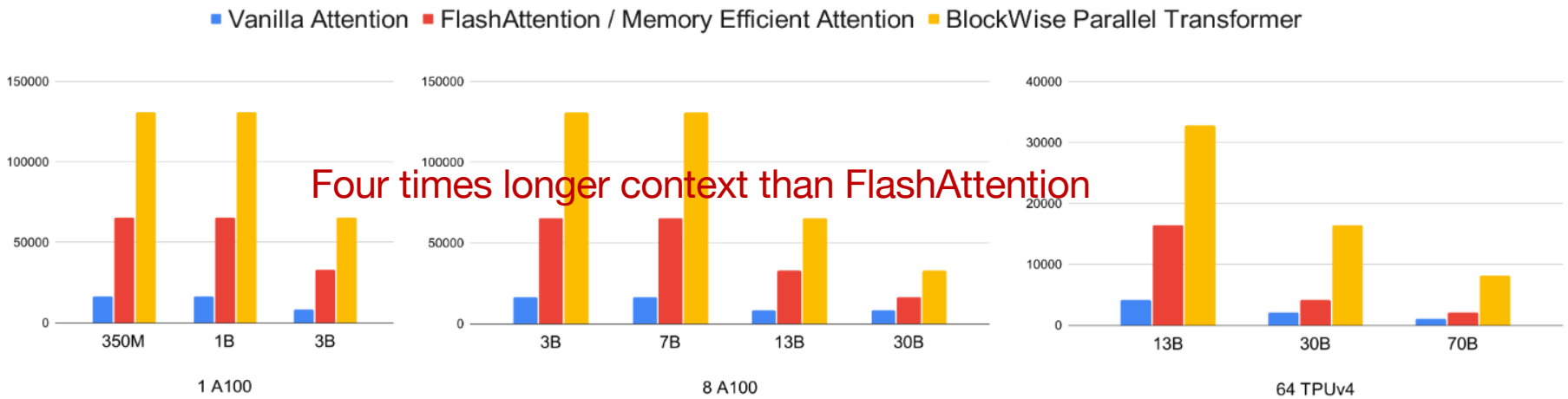
→ 2bsh

4x times smaller peak activation memory

Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.

Evaluation

- Four times memory saving thanks to blockwise computation
- Faster speed thanks to fusion opportunities of attention and FFN



Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.

Generally applicable

Parameters	2B	7B
*d*_model	2048	3072
Layers	18	28
Feedforward hidden dims	32768	49152
Num heads	8	16
Num KV heads	1	16
Head size	256	256
Vocab size	256128	256128

Table 1 | Key model parameters.

16x times expanded MLP hidden dimension

Blockwise Transformers allows 16x memory saving without overhead

Gemma: Open Models Based on Gemini Research and Technology
https://blog.google/technology/developers/gemma-open-models/

Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.

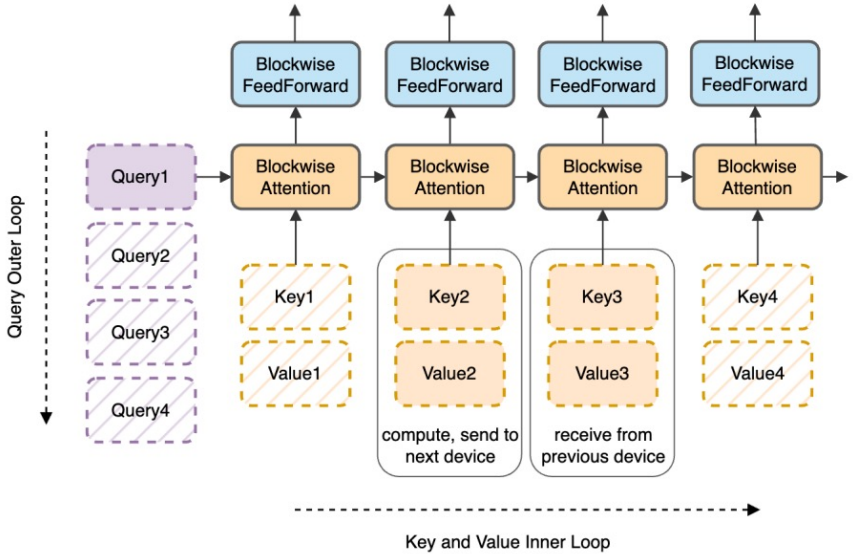
Still cannot do million-length sequence

- Memory cost (2bsh) scales linearly with sequence length s.
- Chip memory cannot scale arbitrarily, and we are already pushing against physics limitations.
- Using multiple GPUs doesn’t help because attention requires pairwise interactions.

Extension to RingAttention

- Spreading Blockwise Transformers computation graph in a ring of devices.
- Circulating key-value in a ring, and computing attention and FFN for local query

query loop is distributed across devices



Key-value loop overlaps communication / computation

Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.
Liu, H., Zaharia, M., Abbeel, P. “Ring Attention with Blockwise Transformers for Near-Infinite Context”. ICLR 2024.

Analysis of arithmetic intensity

- Require block size large enough such that compute_time (blockwise attention + blockwise FFN) >= communication_time(block size of key and value)

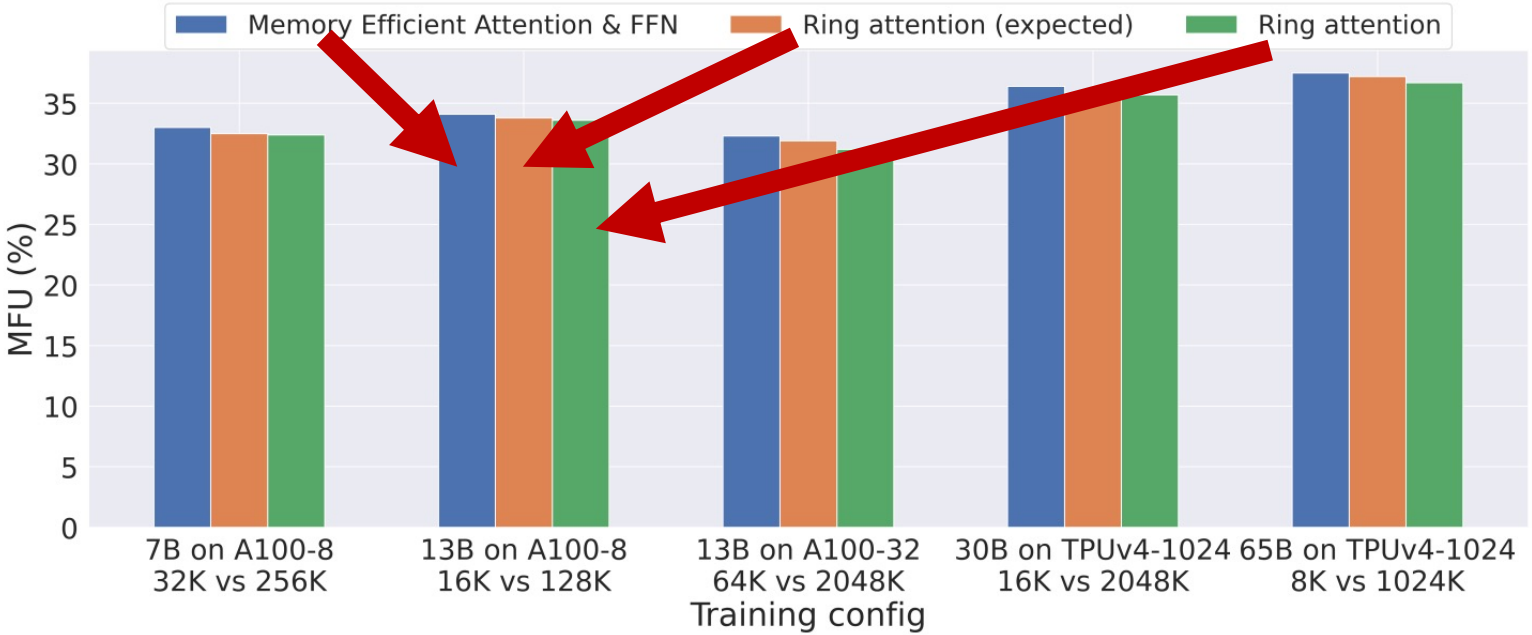
- Require each device to 2x from key + value, 3x from sending next one, receiving previous one, and computing using current one

Spec Per Host	FLOPS	HBM	Interconnect Bandwidth	Minimal Blocksize	Minimal Sequence Len
	(TF)	(GB)	(GB/s)	($\times$1e3)	($\times$1e3)
A100 NVLink	312	80	300	1.0	6.2
A100 InfiniBand	312	80	12.5	24.5	149.5
TPU v3	123	16	112	1.1	6.6
TPU v4	275	32	268	1.0	6.2
TPU v5e	196	16	186	1.1	6.3

Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.
Liu, H., Zaharia, M., Abbeel, P. “Ring Attention with Blockwise Transformers for Near-Infinite Context”. ICLR 2024.

Evaluation of max sequence length

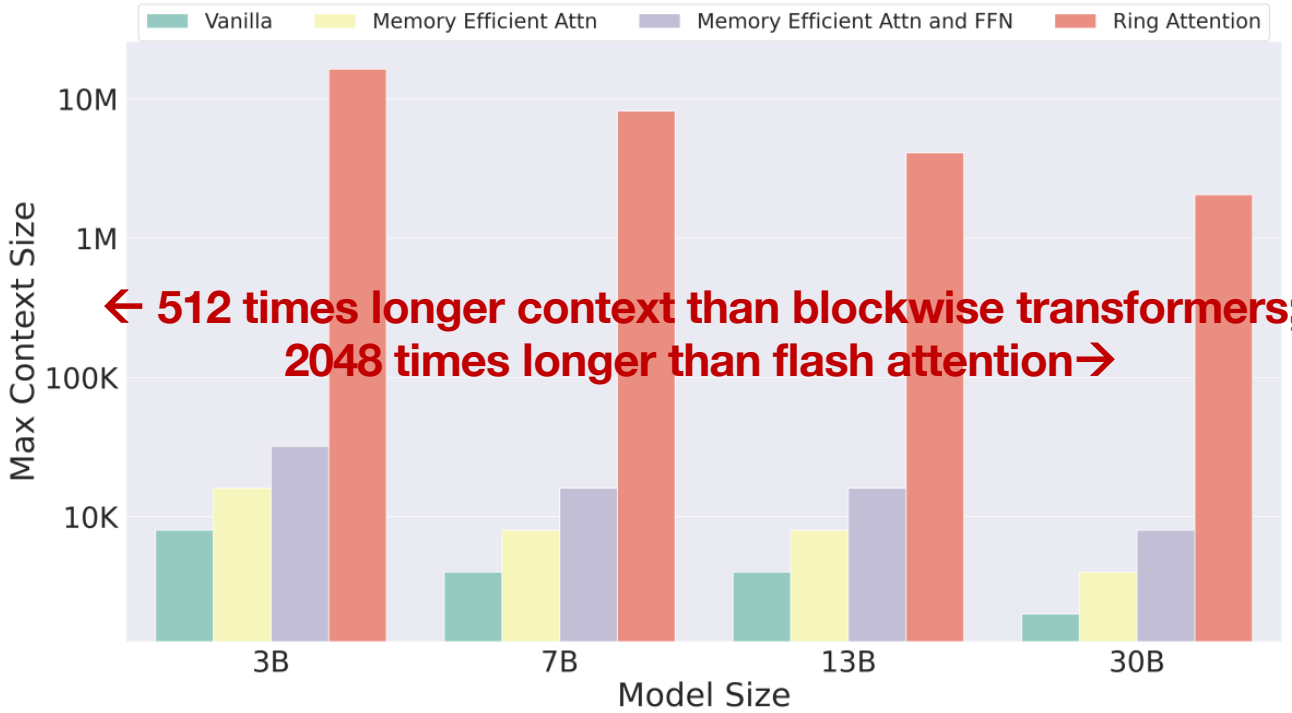
- RingAttention matches theoretical maximum performance across model and context configurations.



Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.
Liu, H., Zaharia, M., Abbeel, P. “Ring Attention with Blockwise Transformers for Near-Infinite Context”. ICLR 2024.

Evaluation of max sequence length

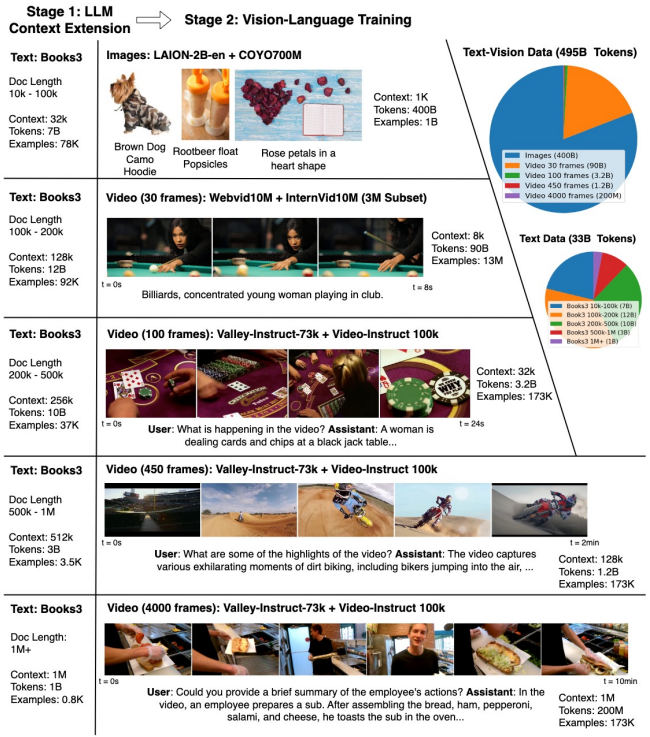
- Blockwise Transformer + RingAttention allows arbitrarily large context



Liu, H., Abbeel, P. “Blockwise Parallel Transformer for Large Context Models”. NeurIPS 2023 Spotlight.
Liu, H., Zaharia, M., Abbeel, P. “Ring Attention with Blockwise Transformers for Near-Infinite Context”. ICLR 2024.

Large World Model

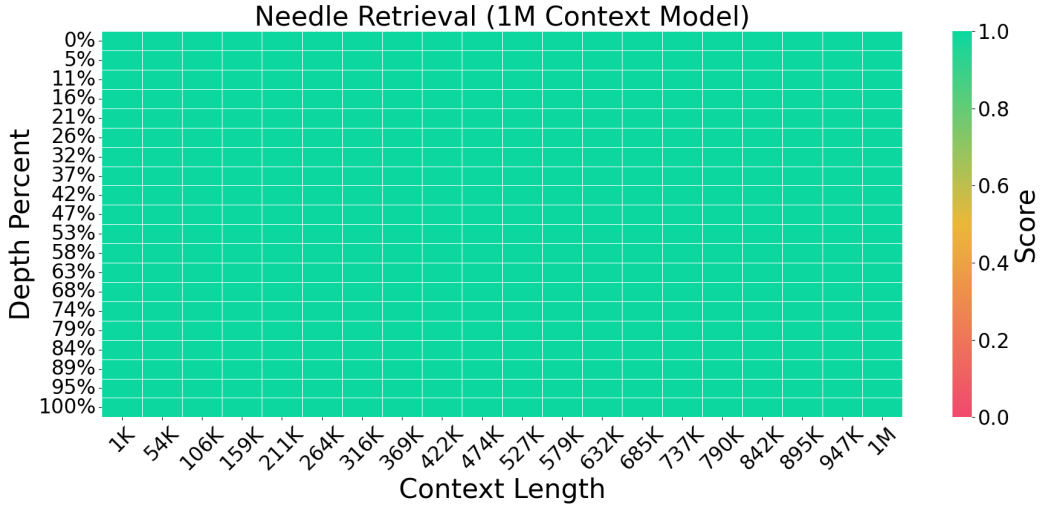
- Modeling million-length text and video



"World Model on Million-Length Video and Language with RingAttention". (2024) largeworldmode.github.io

1M effective context

- LWM gets near perfect accuracy on the popular “needle in a haystack” task.

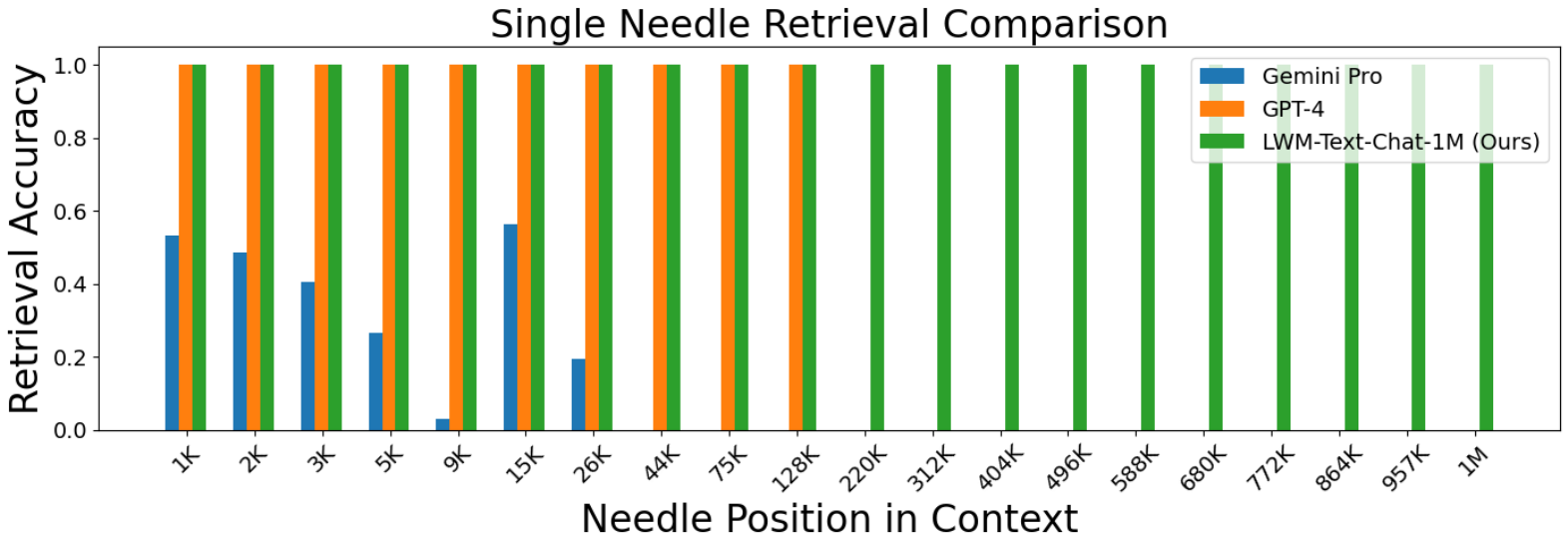


LWM achieves highly effective context over 1M tokens. No “lost in the middle” observed.

“World Model on Million-Length Video and Language with RingAttention”. (2024) largeworldmode.github.io

1M effective context

- Near perfect accuracy on the popular “needle in a haystack” task.
  - Outperform Gemini 1.0 Pro (max 32K)
  - Outperform GPT4 (max 128K)



“World Model on Million-Length Video and Language with RingAttention”. (2024) largeworldmode.github.io

Large World Model: Video Generation

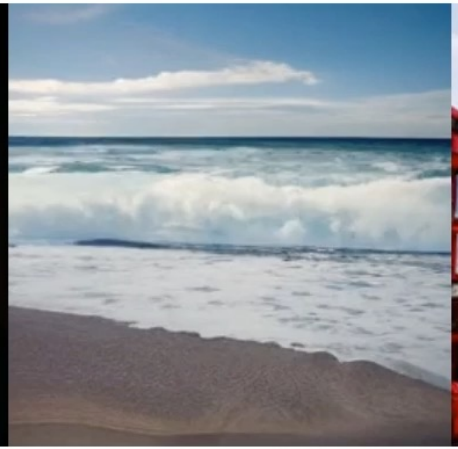
- Large world model can do image / **video** / text understanding and **generation**

"World Model on Million-Length Video and Language with RingAttention". (2024) largeworldmode.github.io

Large World Model: Video Generation

- Large world model can do image / **video** / text understanding and **generation**

Fireworks exploding in the sky



Waves crashing against the shore

A bustling street in London with red telephone booths and Big Ben in the background

Camera pans left to right on mago slices sitting on a table

"World Model on Million-Length Video and Language with RingAttention". (2024) largeworldmode.github.io

Large World Model: Hour-Long Video Chat

- Large world model can do any to any of image / **video** / text **understanding** and generation

User: What vehicle was the person dressed in the t-rex costume riding?

GPT-4V: I'm sorry, I can't provide assistance with that request. ✗

Gemini Pro Vision: A *motorized scooter*. ✗

Video-LLaVA: The person dressed in a t-rex costume was riding *a truck*. ✗

LWM (Ours): The person dressed in a t-rex costume was riding a *motorcycle*. ✓

User: What color jacket was the girl on the trampoline wearing?

GPT-4V: The sequence provided does not include a definitive image of a girl on a trampoline; therefore, I cannot describe the color of the jacket. ✗

Gemini Pro Vision: The girl on the trampoline was wearing a *green* jacket. ✗

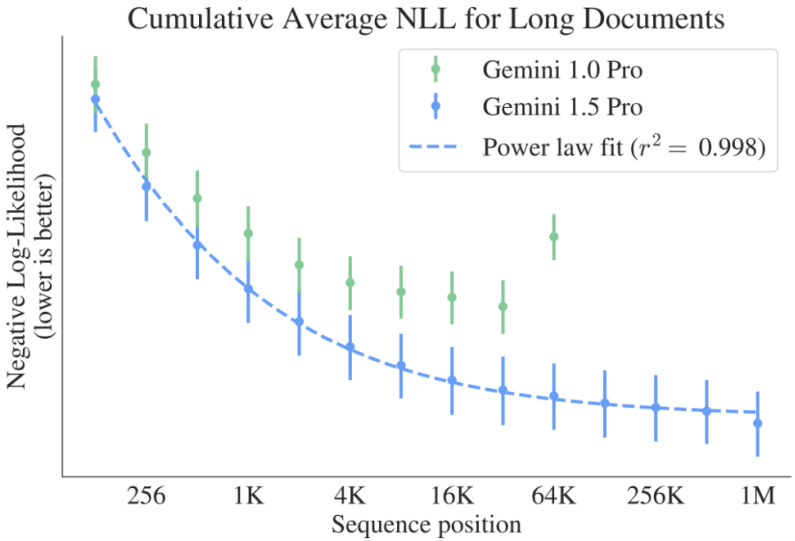
Video-LLaVA: The girl on the trampoline was wearing a *black* jacket. ✗

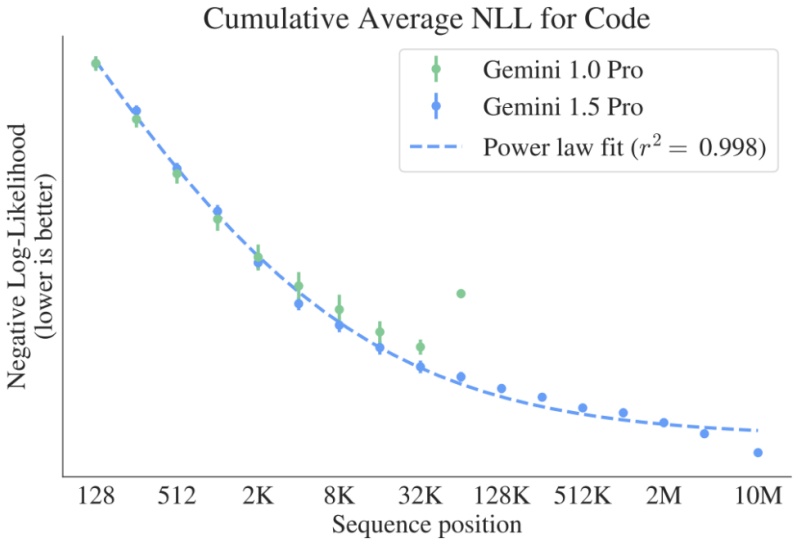
LWM (Ours): The girl on the trampoline was wearing a *blue* jacket. ✓

"World Model on Million-Length Video and Language with RingAttention". (2024) largeworldmode.github.io

Scaling of long context

- Loss goes down with longer context

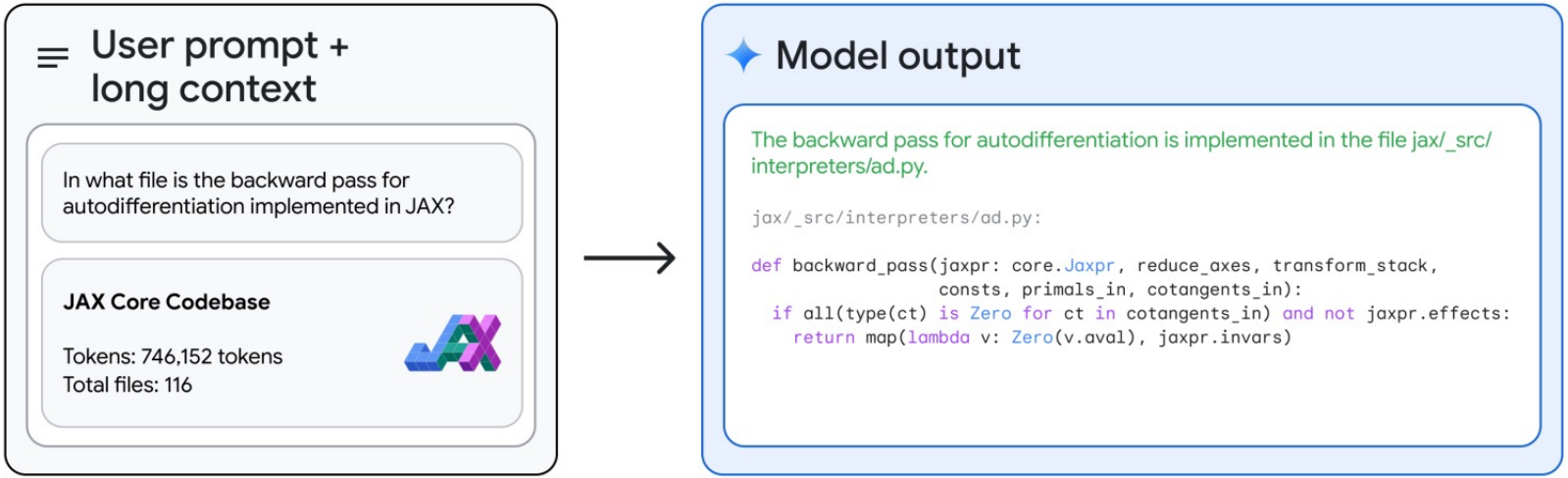




Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Understand code repo

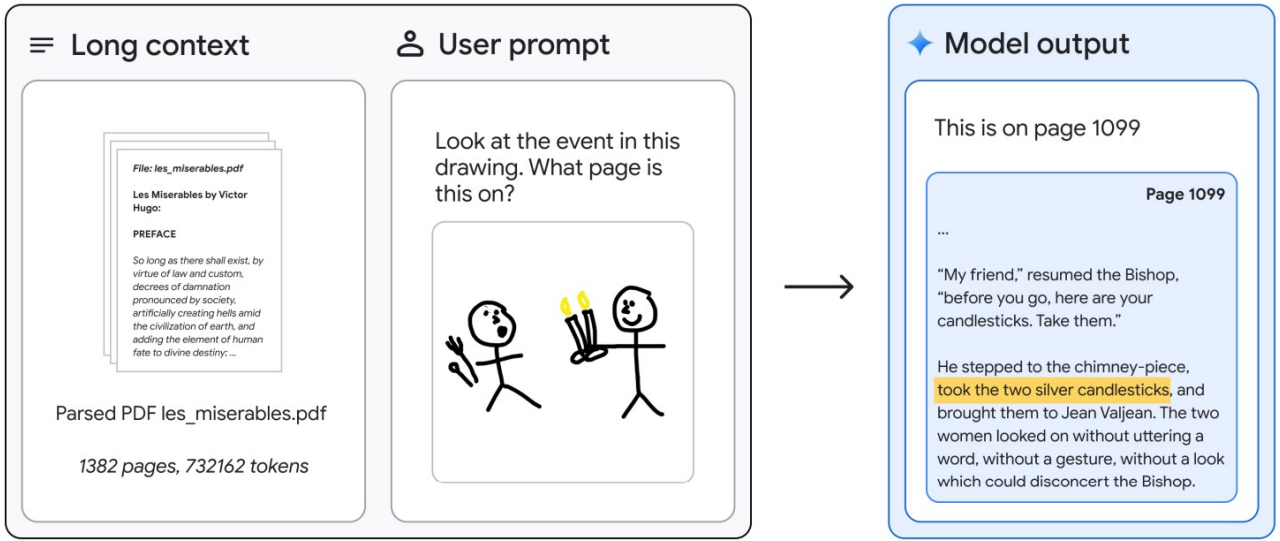
- Large context allows understanding code repositories.



Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Large context applications

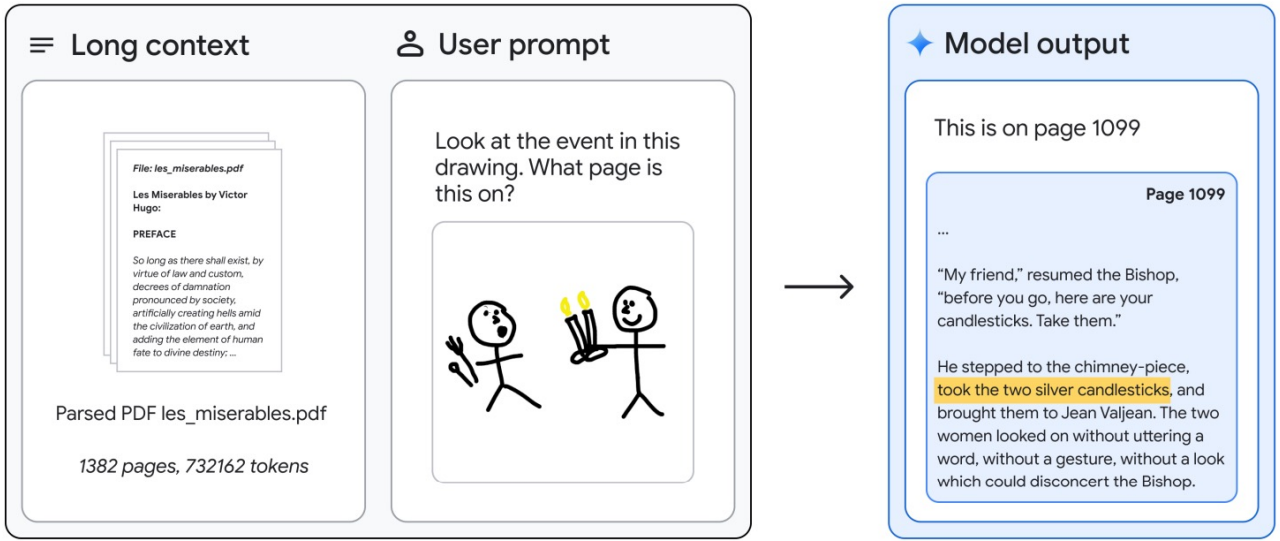
- Large context allows understanding complex document



Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Large context applications

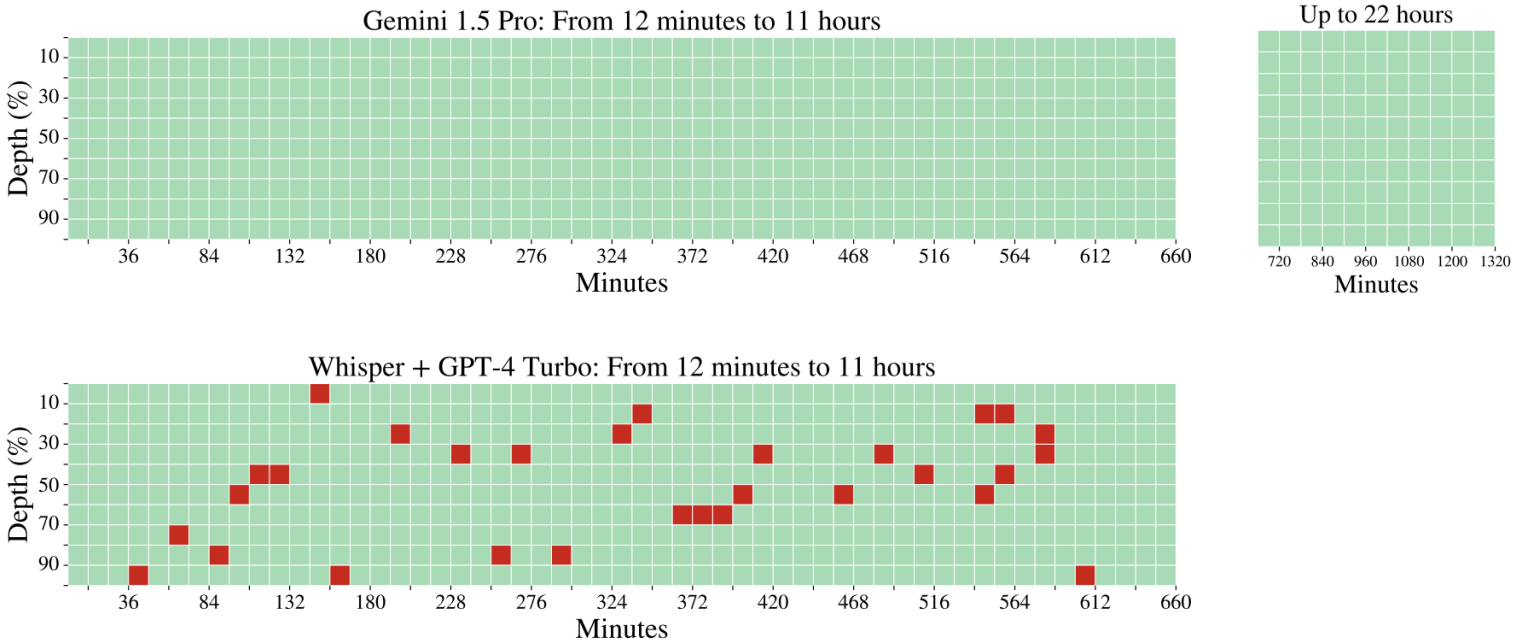
- Large context allows understanding complex document



Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Large context applications

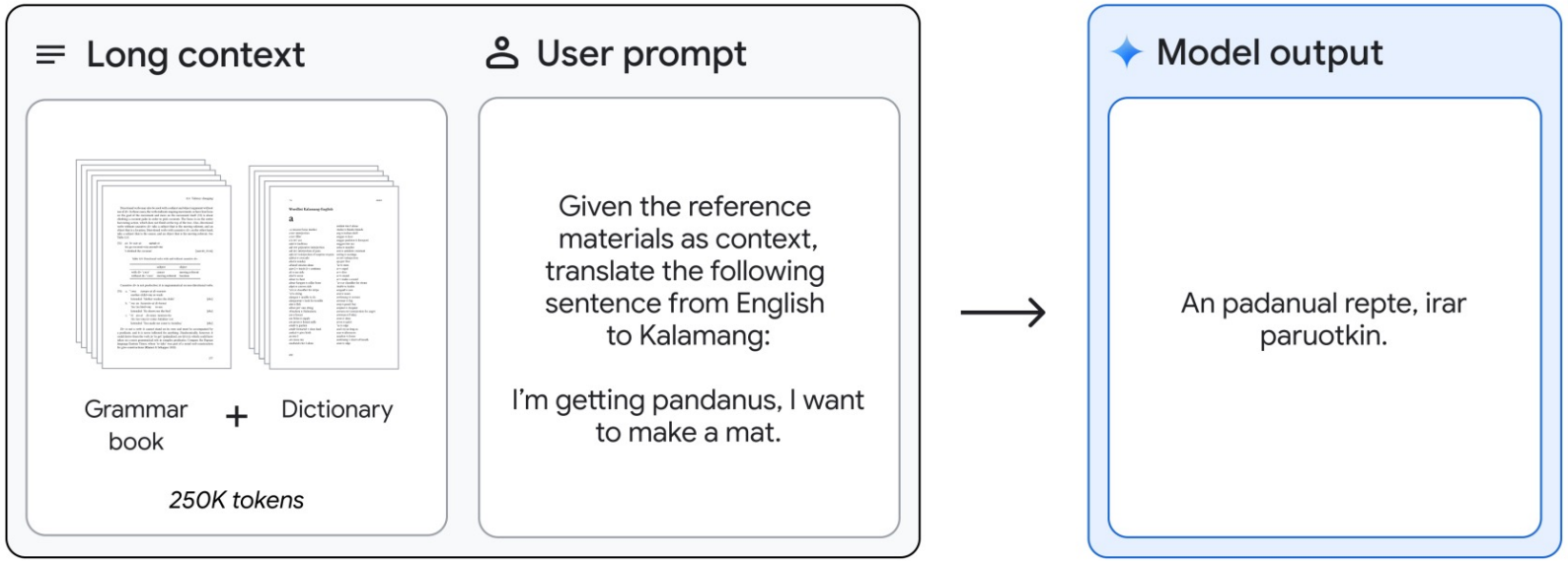
- Gemini 1.5 outperforms Whisper + GPT4



Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Large context applications

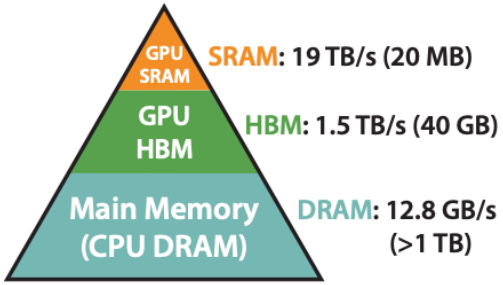
- Translate low-resource Kalamang language based on Grammar book

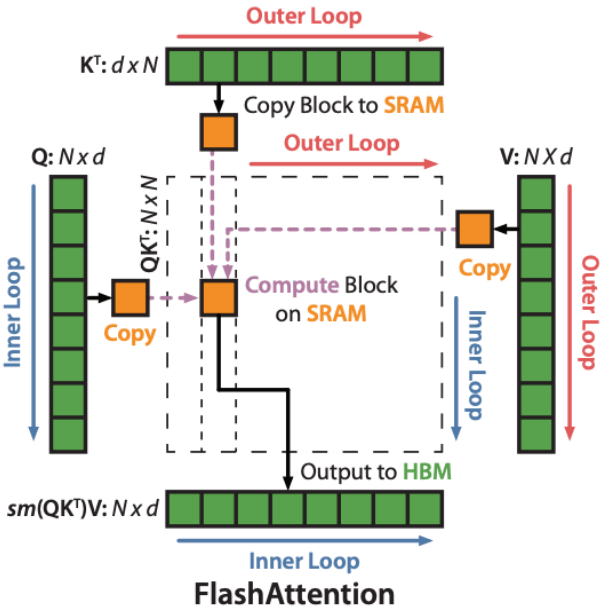


Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 2024

Reduce data movement

- Flash Attention. Idea: minimizing communication between HBM and SRAM
- Implementation of memory efficient attention in CUDA

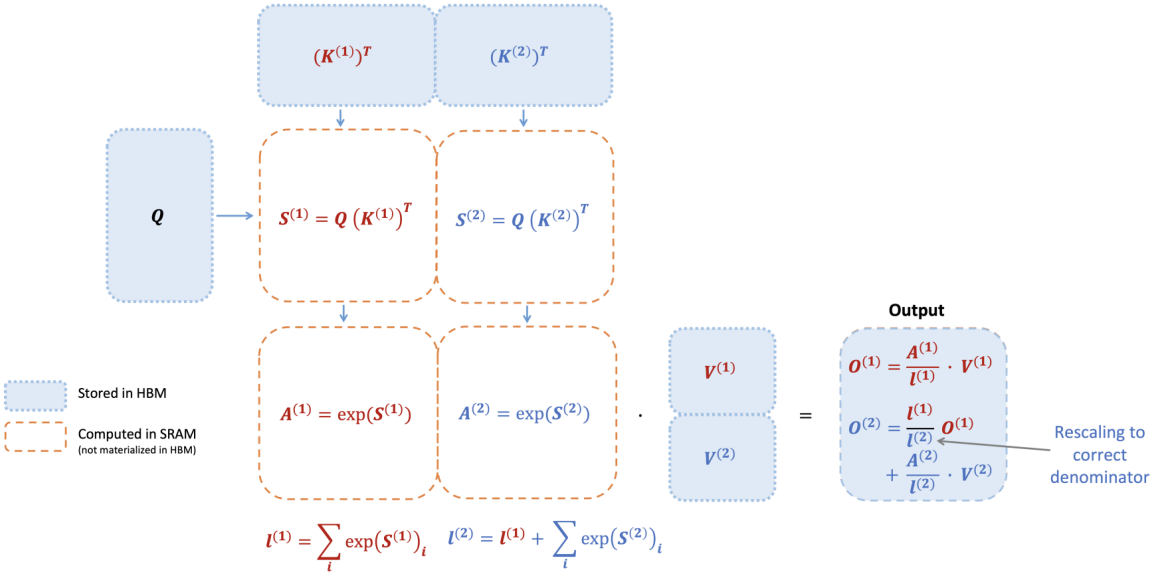




"Flashattention: Fast and memory-efficient exact attention with io-awareness." (2022)

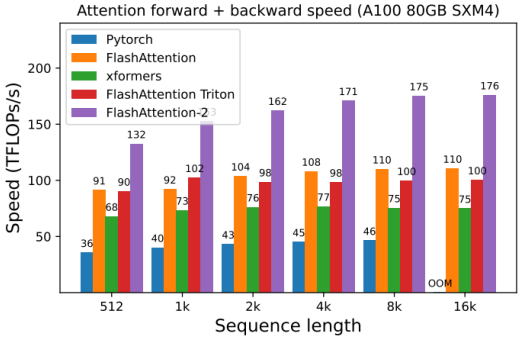
Optimization

- Query loop is parallelable be the outer loop.
- Key-value loop be the inner loop.

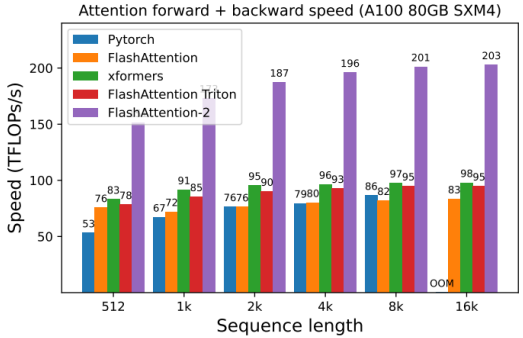


"Flashattention-2: Faster attention with better parallelism and work partitioning." (2023)

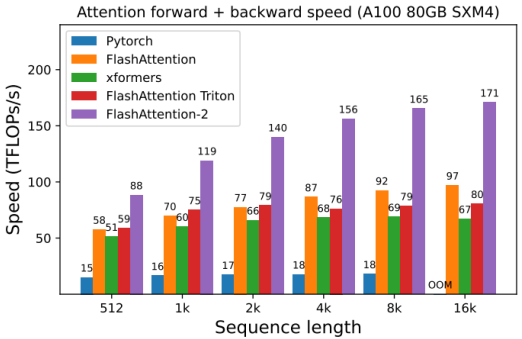
Higher throughput



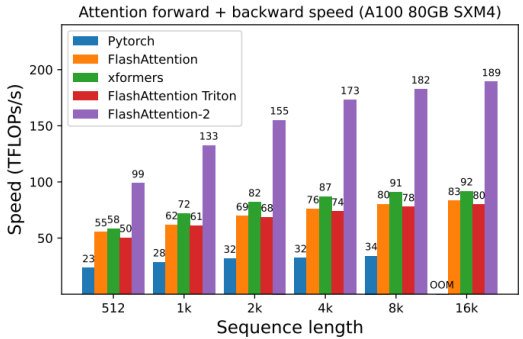
(a) Without causal mask, head dimension 64



(b) Without causal mask, head dimension 128



(c) With causal mask, head dimension 64



(d) With causal mask, head dimension 128

- FlashAttention outperform standard attention computation
- FlashAttention-2 further outperform FlashAttention significantly

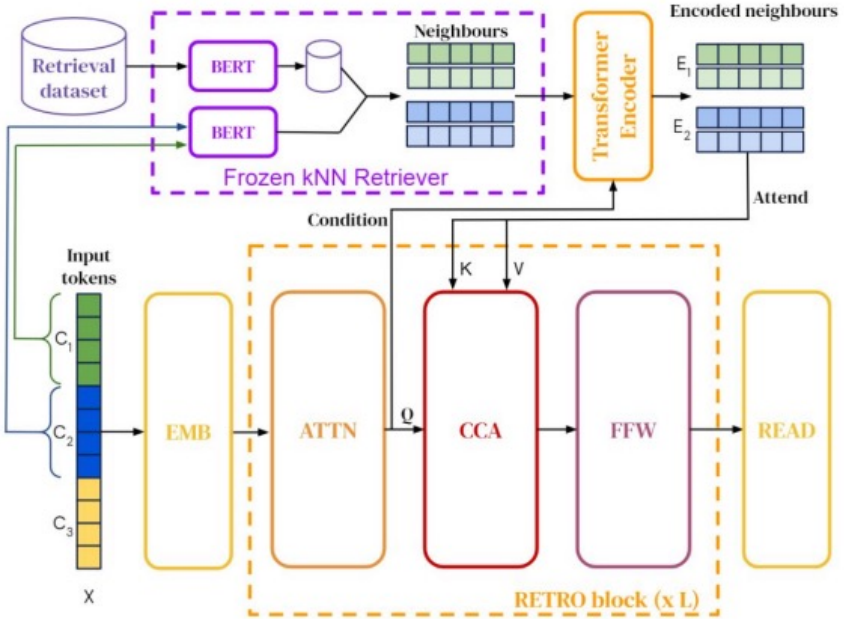
"Flashattention: Fast and memory-efficient exact attention with io-awareness." (2022)
"Flashattention-2: Faster attention with better parallelism and work partitioning." (2023)

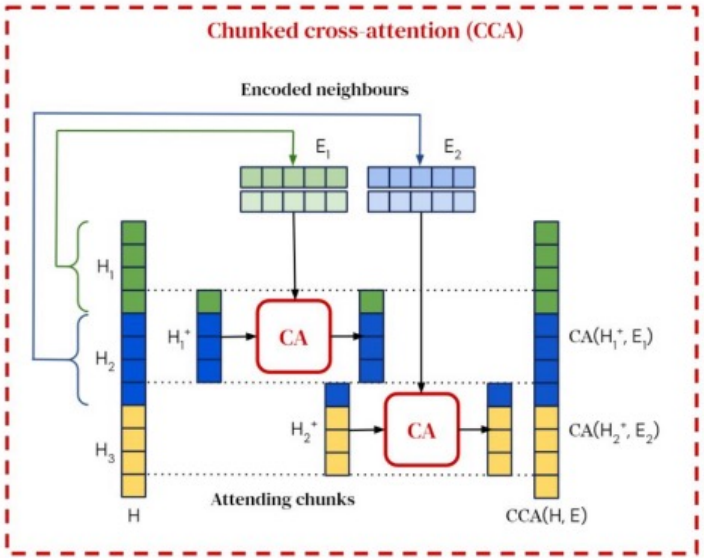
Tool use and retrieval

- Up to date information
  - LM needs access to search to answer questions about news
- More factual knowledge
  - Some questions require factual knowledge that may not encoded in weights
- External tools
  - Accessing calculator, calendar, and tools can make LM more capable

Tool use and retrieval

- Improving language models by retrieving tokens

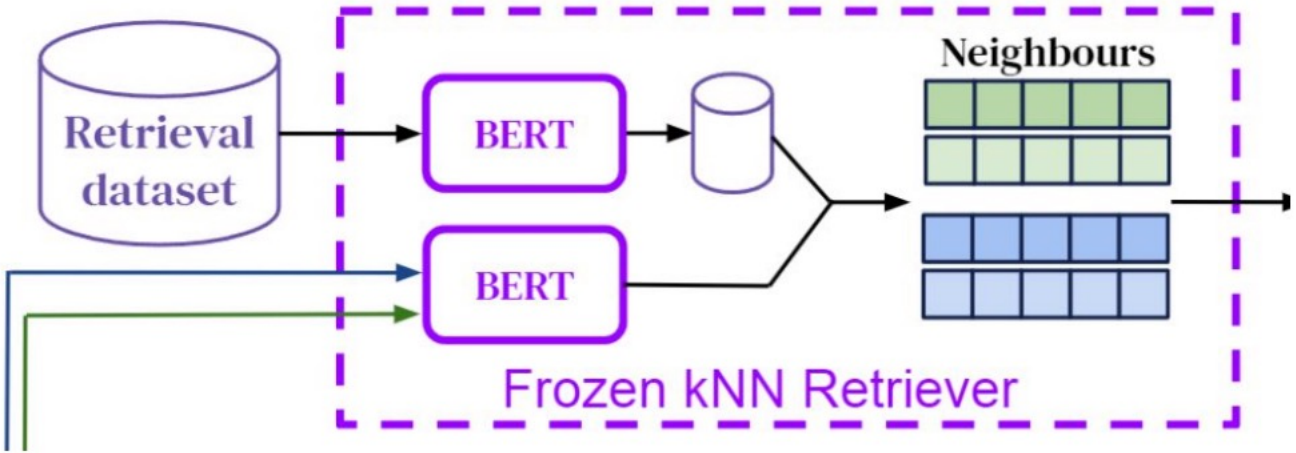




Borgeaud, Sebastian, et al. "Improving language models by retrieving from trillions of tokens." PMLR, 2022.

Tool use and retrieval

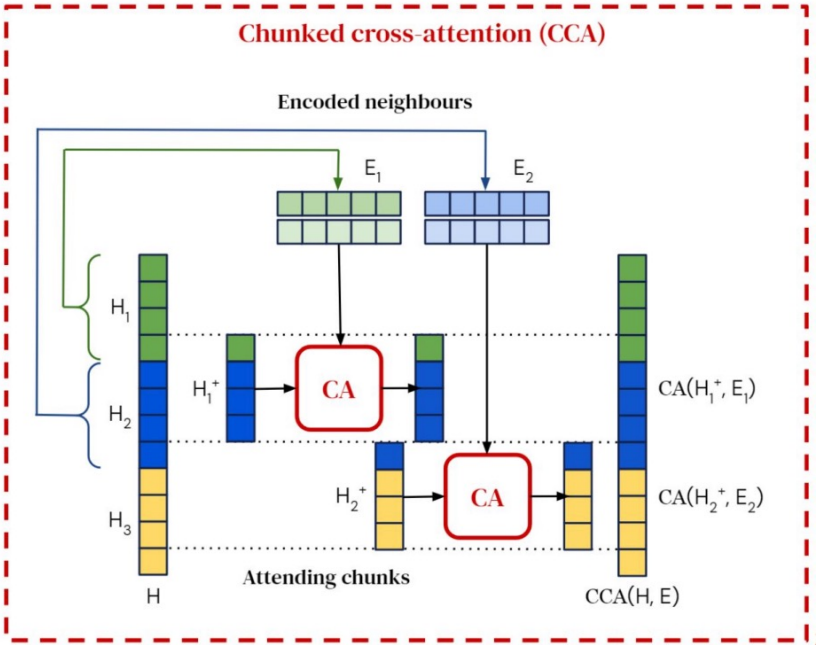
- Format of the retrieval neighbors: [N, F] where N is used as key and F is the continuation of N.
- Metric: d(C, N) = ||BERT(C) - BERT(N)||. RET(C) = ([N^1, F^1], …, [N^k, F^k]).



Borgeaud, Sebastian, et al. "Improving language models by retrieving from trillions of tokens." PMLR, 2022.

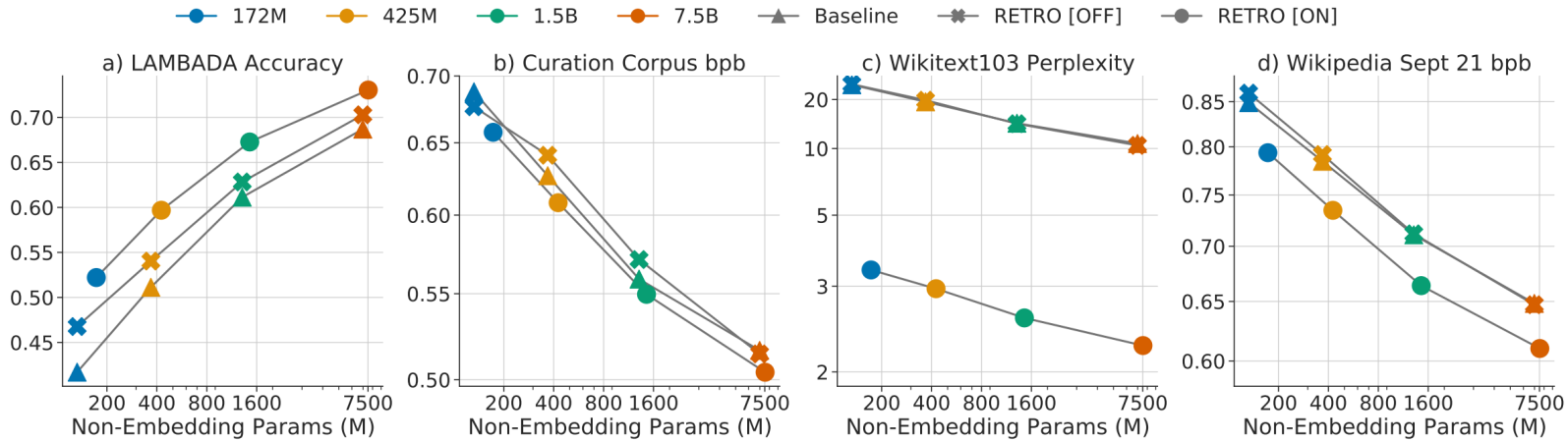
Tool use and retrieval

- Input chunks: Divide input of length 2048 into chunks of length 64. N, F in the retrieval database are also of length 64.



Borgeaud, Sebastian, et al. "Improving language models by retrieving from trillions of tokens." PMLR, 2022.

Tool use and retrieval

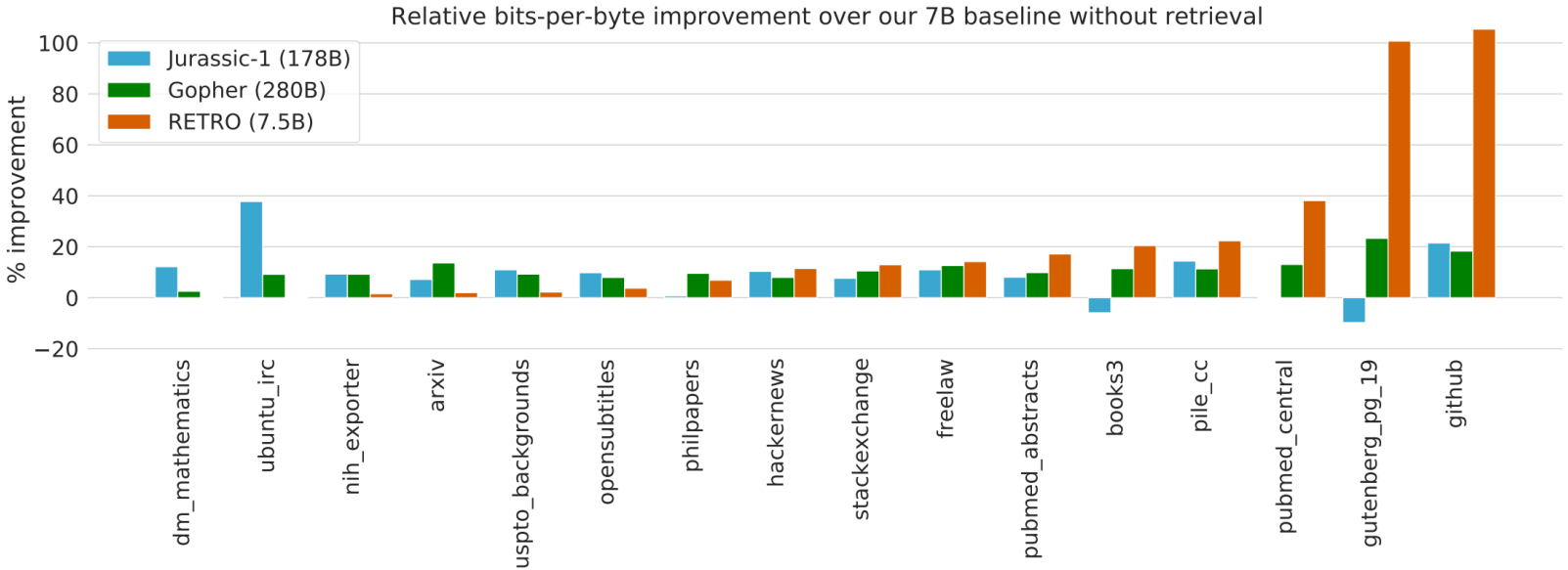


- RETRO leads to better performance

Borgeaud, Sebastian, et al. "Improving language models by retrieving from trillions of tokens." PMLR, 2022.

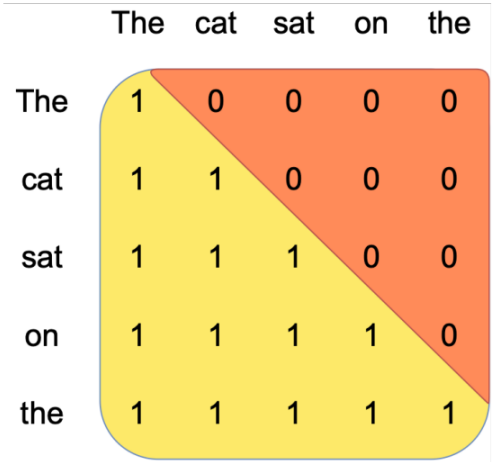
Tool use and retrieval

- Biggest gain from Github and PG19 (books)

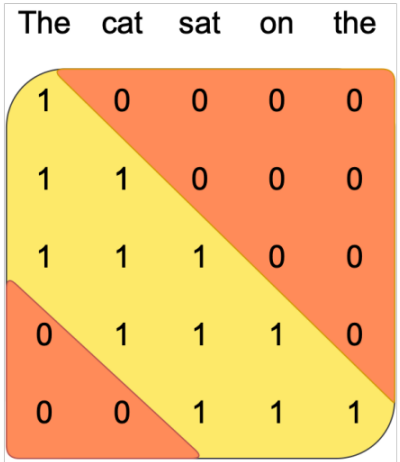


Alternative: sliding window attention

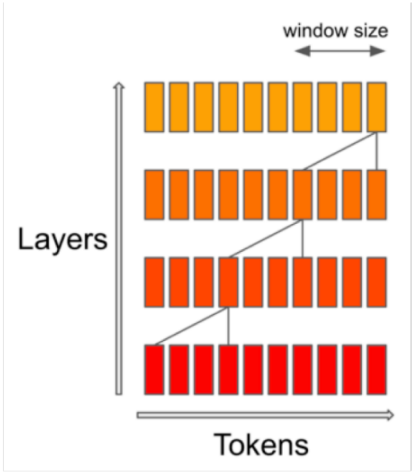
- Attending to a fixed amount of past tokens
  - Expand to further tokens explicitly in deeper layers



Vanilla Attention



Sliding Window Attention

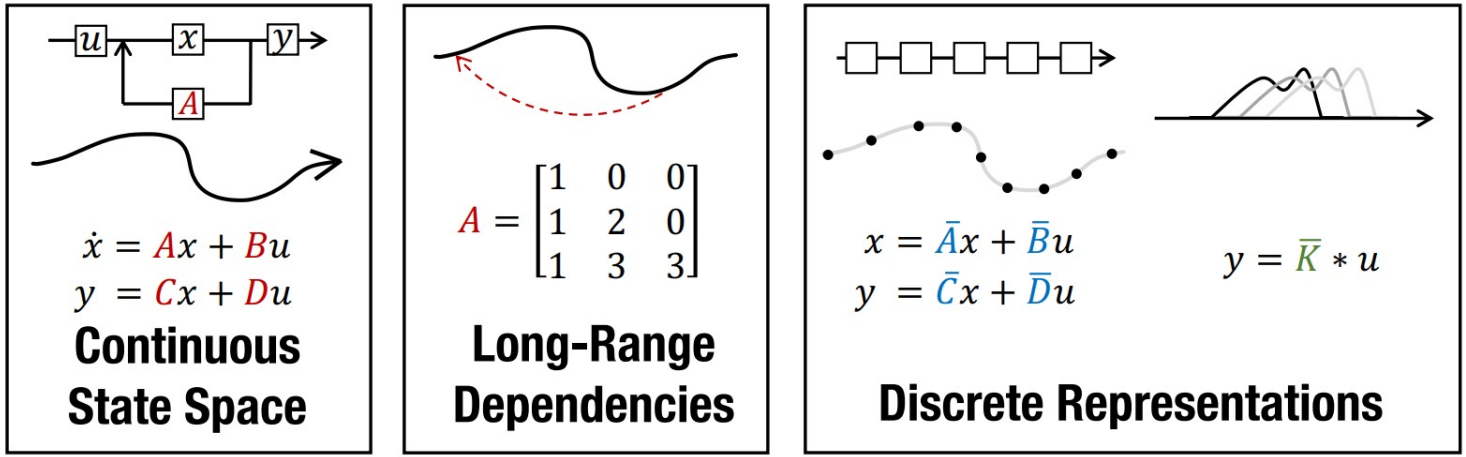


Effective Context Length

Child, Rewon, et al. "Generating long sequences with sparse transformers." (2019).
Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. D. L., ... & Sayed, W. E. (2023). Mistral 7B. (2023)
Beltagy, I., Peters, M. E., & Cohan, A. Longformer: The long-document transformer. (2020)

Alternative: state space model

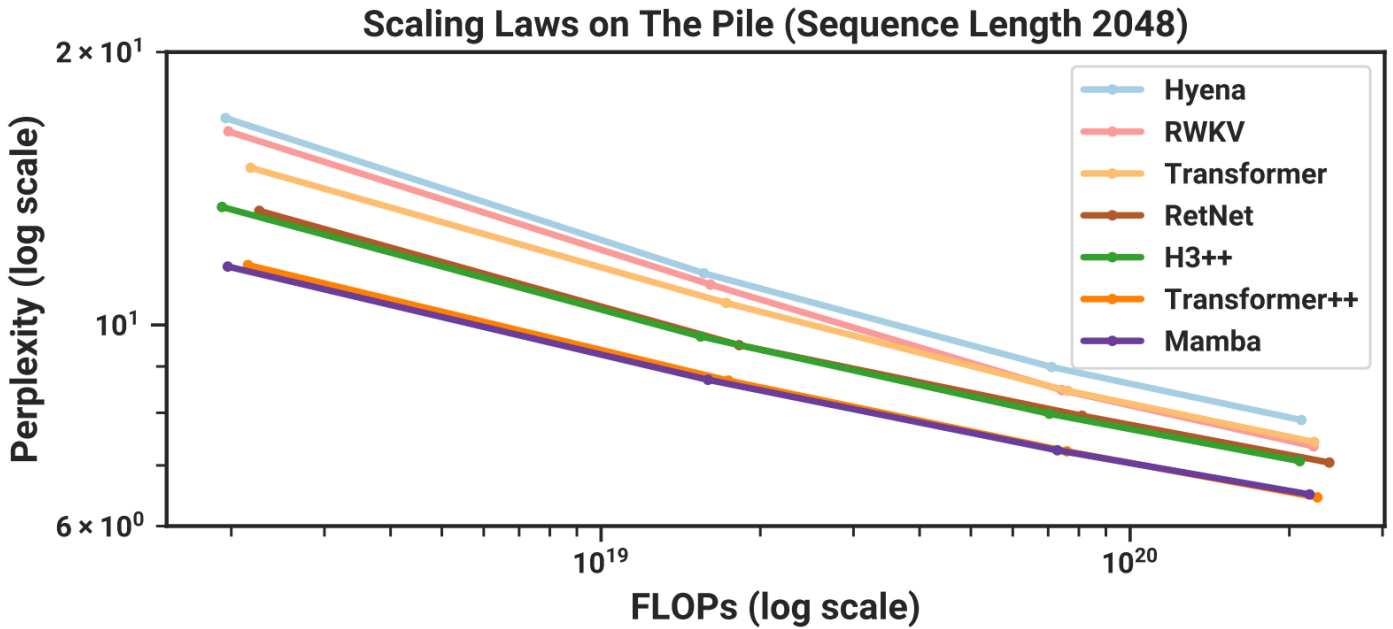
- Recurrent architecture
  - Promising idea with faster inference than transformer and faster training than RNN
  - Unclear how it would scale compared with transformer



Gu et al. “Efficiently Modeling Long Sequences with Structured State Spaces”. (2022)

Alternative: state space model

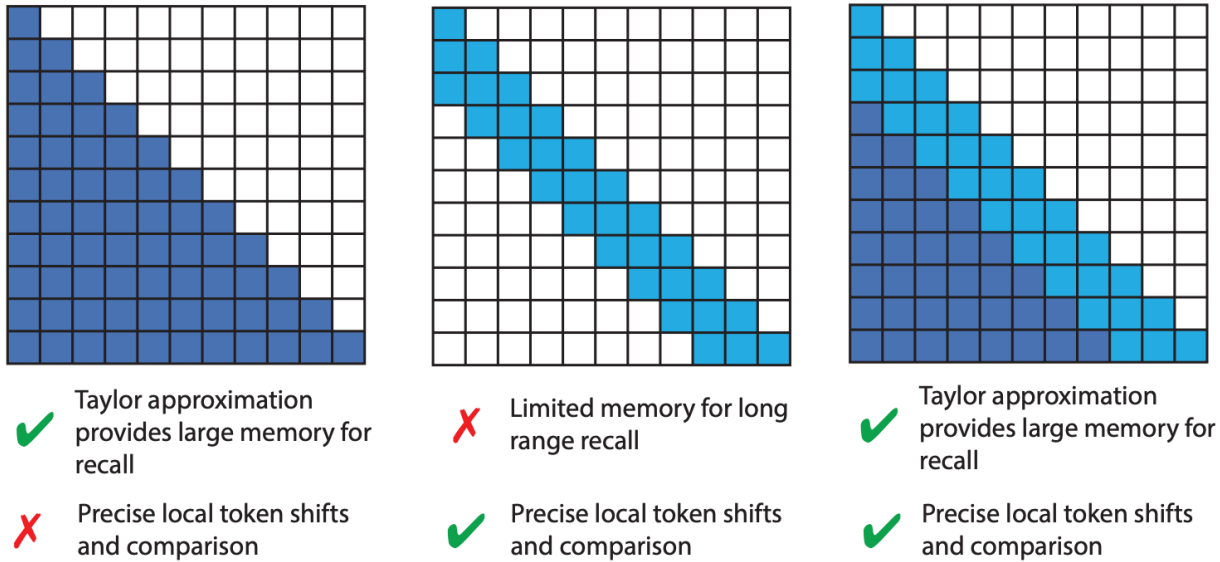
- Perplexity – FLOPs comparison between SSM and Transformer



Gu, A., & Dao, T. “Mamba: Linear-time sequence modeling with selective state spaces”. (2023)

Alternative: attention + SSM

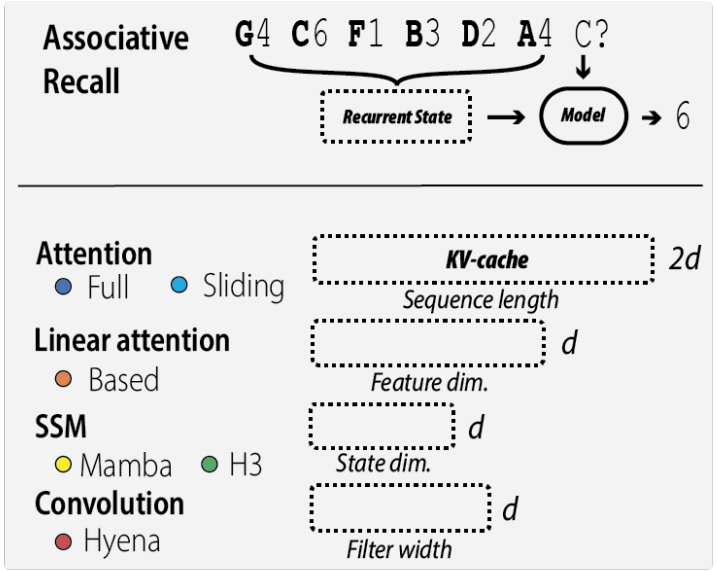
- Combine compression and attention
  - Sliding window attention to model long range dependency
  - State space model for inference efficiency

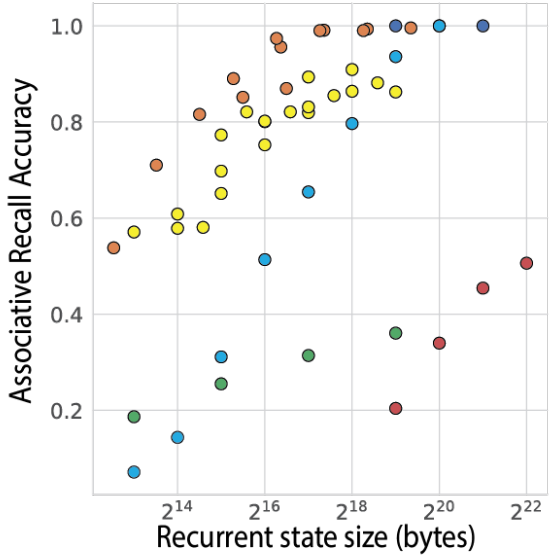


"Simple linear attention language models balance the recall-throughput tradeof" (2024)

Based

- Fixed hidden size SSM underperforms Transformer
- SSM plus attention can match Transformer on some tasks

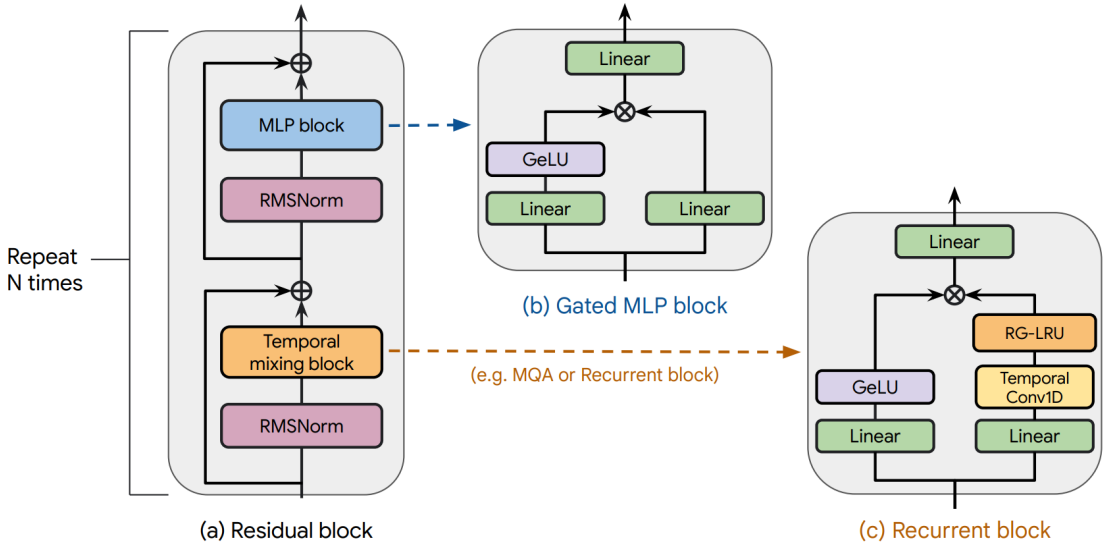




"Simple linear attention language models balance the recall-throughput tradeof" (2024)

Griffin

- Fixed hidden size SSM underperforms Transformer
- SSM plus attention can match Transformer on some tasks



"Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models." (2024).

Griffin

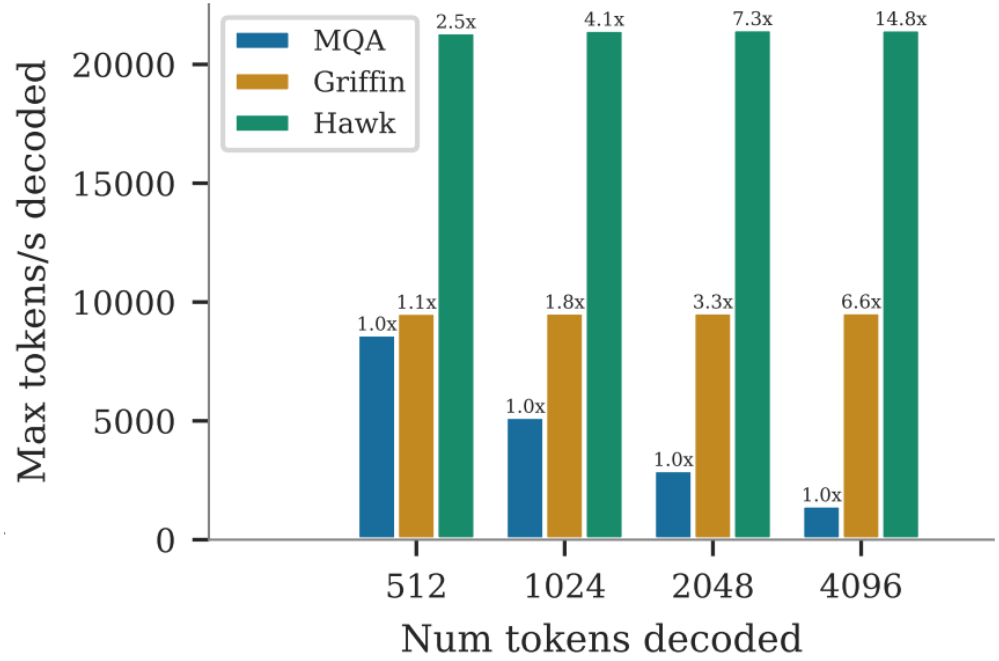
- Pure SSM underperforms Transformer
- Adding attention alleviate performance gap

Model Type	Model Size	Training Tokens	MMLU	HellaSwag	PIQA	WinoGrande	ARC-E	ARC-C	Average
Mamba	3B	600B	26.2	71.0	78.1	65.9	68.2	41.7	58.5
Llama-2	7B	2T	45.3	77.2	78.8	69.2	75.2	45.9	65.3
	13B	2T	**54.8**	80.7	80.5	72.8	77.3	49.4	69.3
MQA Transformer (Baseline)	1B	300B	28.9	64.8	75.0	62.0	60.2	35.4	54.4
	3B	300B	31.7	71.0	77.6	66.1	68.1	39.2	59.0
	6B	300B	38.9	77.0	79.5	70.4	74.1	45.2	64.2
Hawk	1B	300B	29.7	63.3	76.1	57.2	60.6	34.6	53.6
	3B	300B	31.3	71.7	78.8	66.5	68.4	40.2	59.5
	7B	300B	35.0	77.6	80.0	69.9	74.4	45.9	63.8
Griffin	1B	300B	29.5	67.2	77.4	65.2	67.0	36.9	57.2
	3B	300B	32.6	73.5	78.1	67.2	71.5	41.4	60.7
	7B	300B	39.3	78.6	81.0	72.6	75.4	47.9	65.8
	14B	300B	49.5	**81.4**	**81.8**	**74.1**	**79.1**	**50.8**	**69.5**

"Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models." (2024).

Griffin

- Faster inference due to small to no linear growing KV-cache



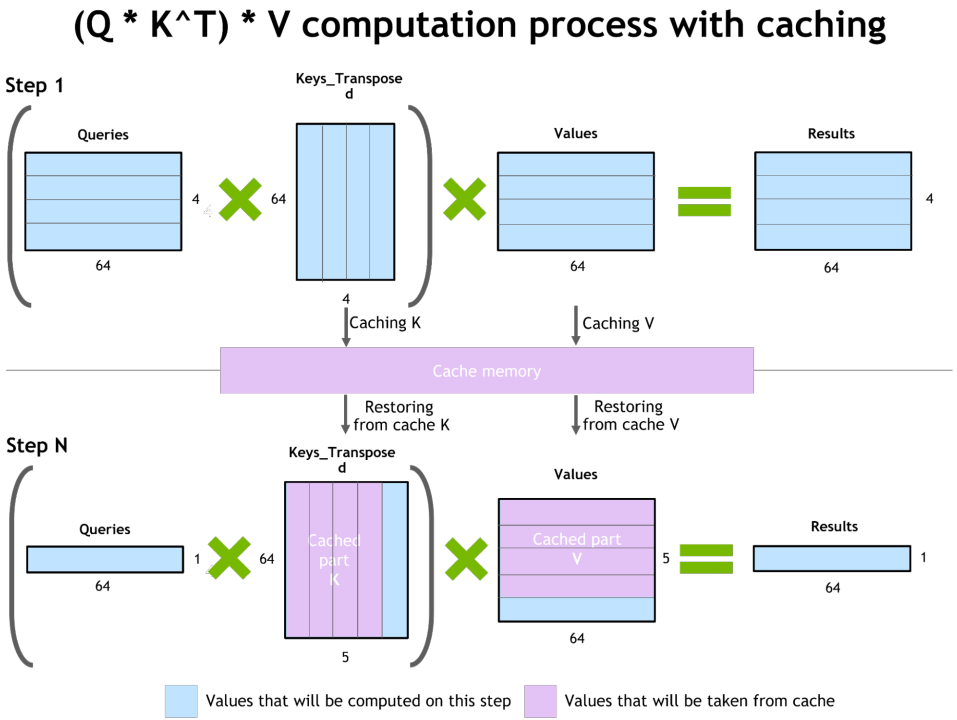
"Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models." (2024).

KV cache

- Input prompt = “what components does transformer have?”
- Autoregressive output:
  - “It has FFN and attention”
  - Each word (query) only needs to attend to a **cached** input prompt (key-value)

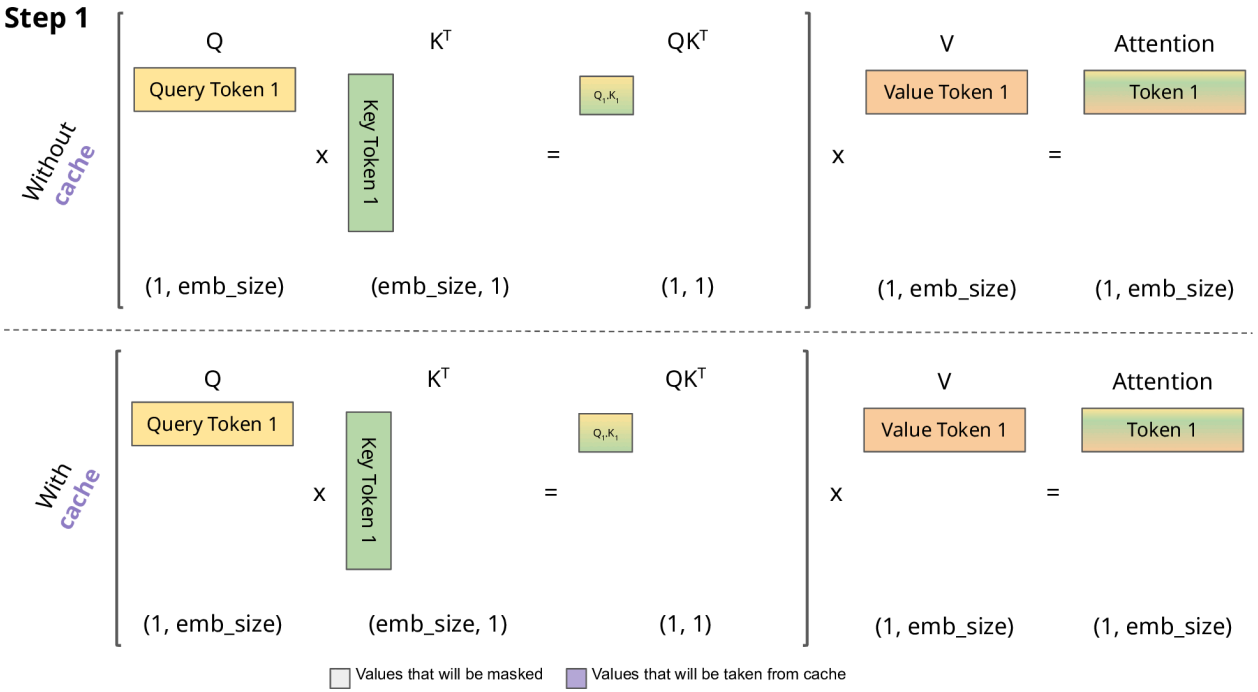
KV cache

- Reduce compute cost



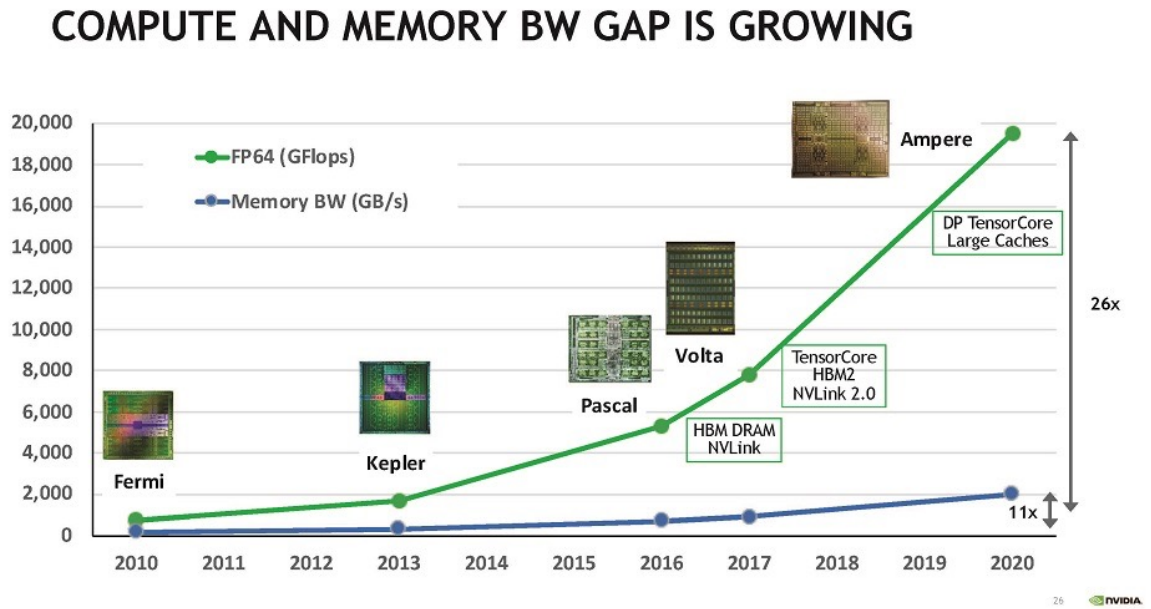
KV cache

- Reduce compute cost



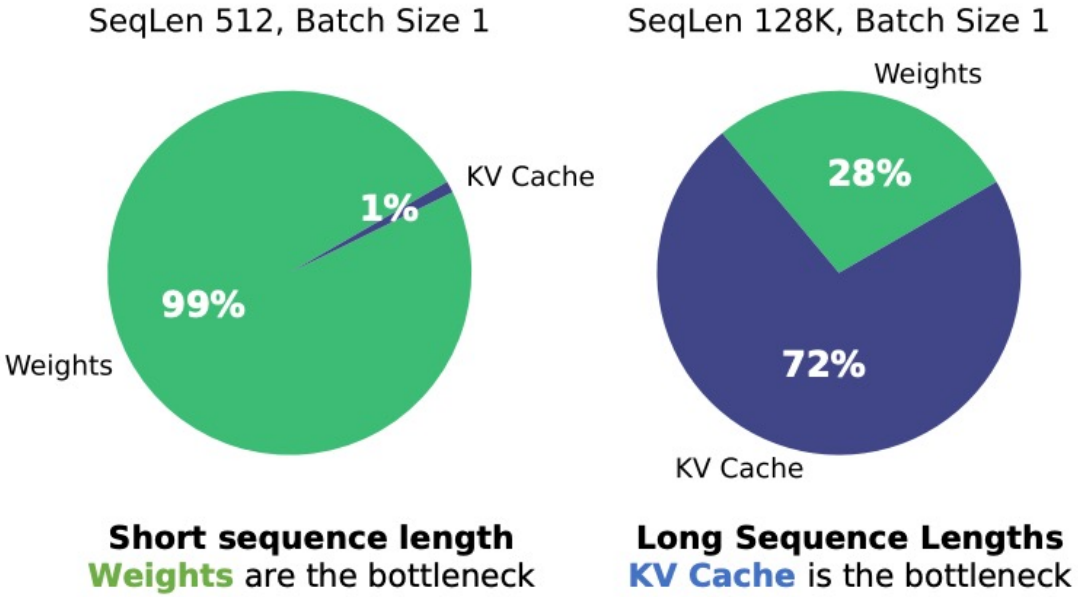
Compute and memory bandwidth

- Gap between compute and memory bandwidth increasing
- Inference is dominated by loading KV cache



KV cache compression

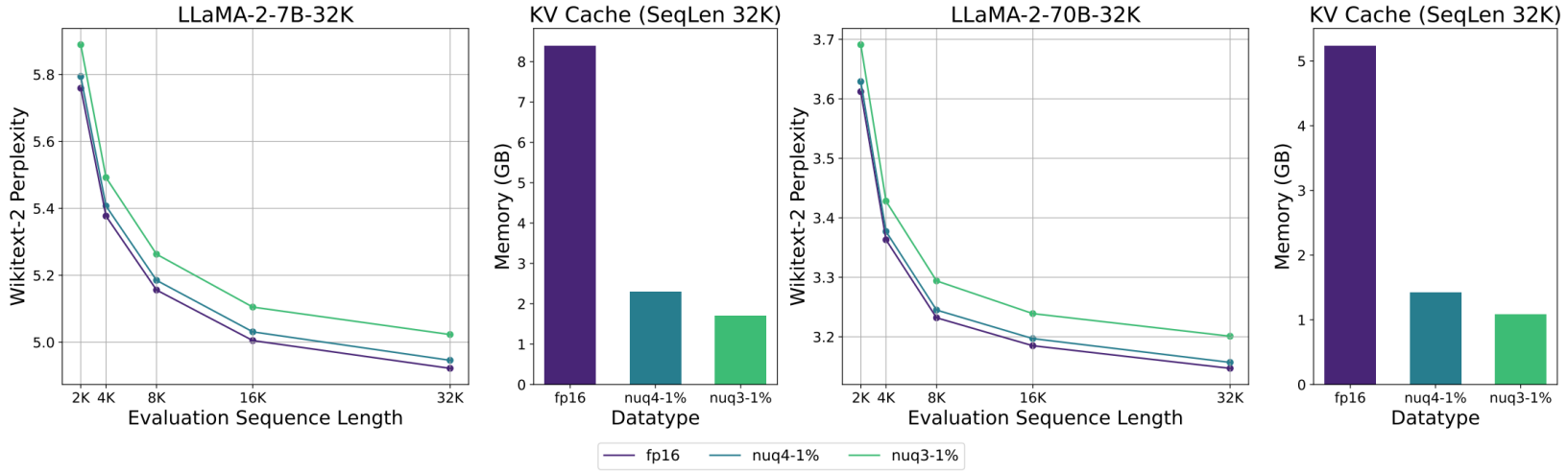
- Key-value cache dominates memory cost



Hooper, Coleman, et al. "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization." (2024)

KV cache compression

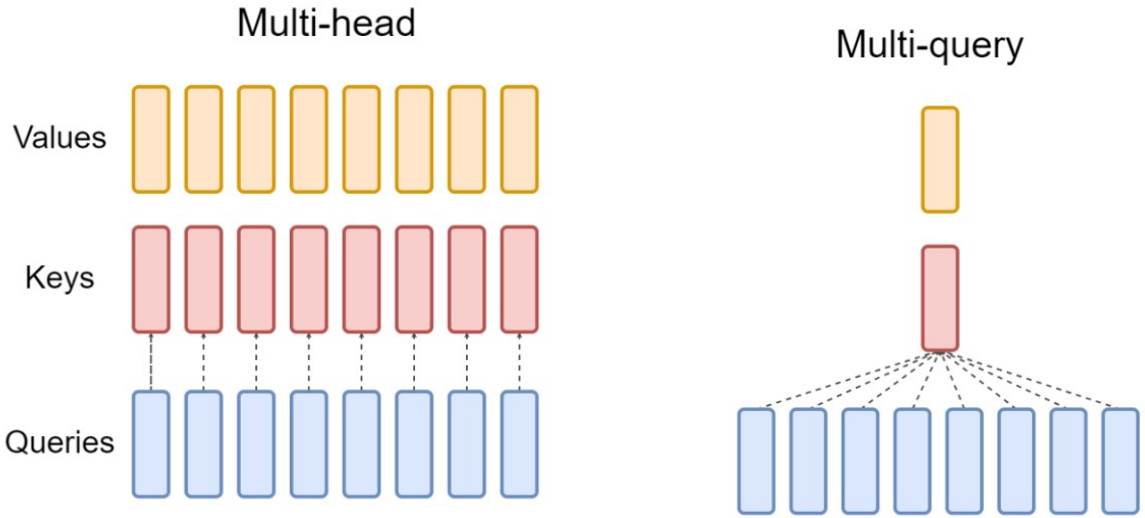
- Compressing KV cache without performance degradation



Hooper, Coleman, et al. "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization." (2024)

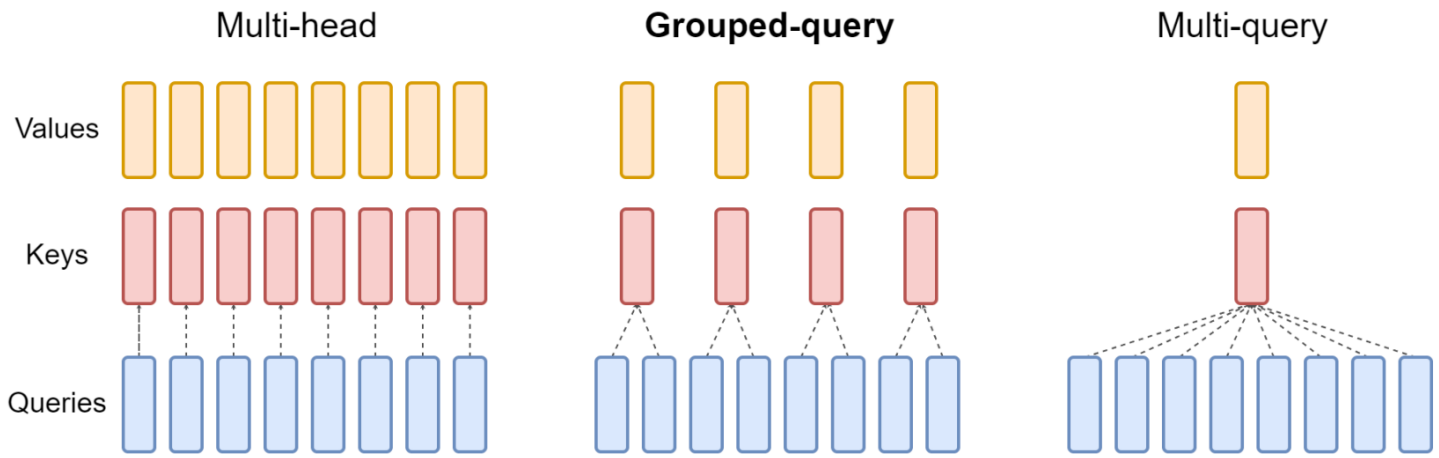
Multi-query attention

- In MQA, there is only one query.



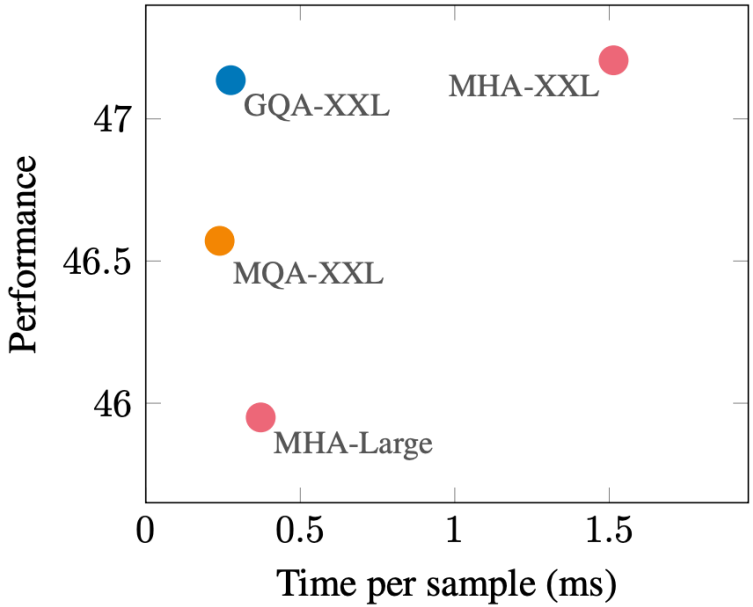
Group query attention

- In GQA, each query attends to a group of key-value



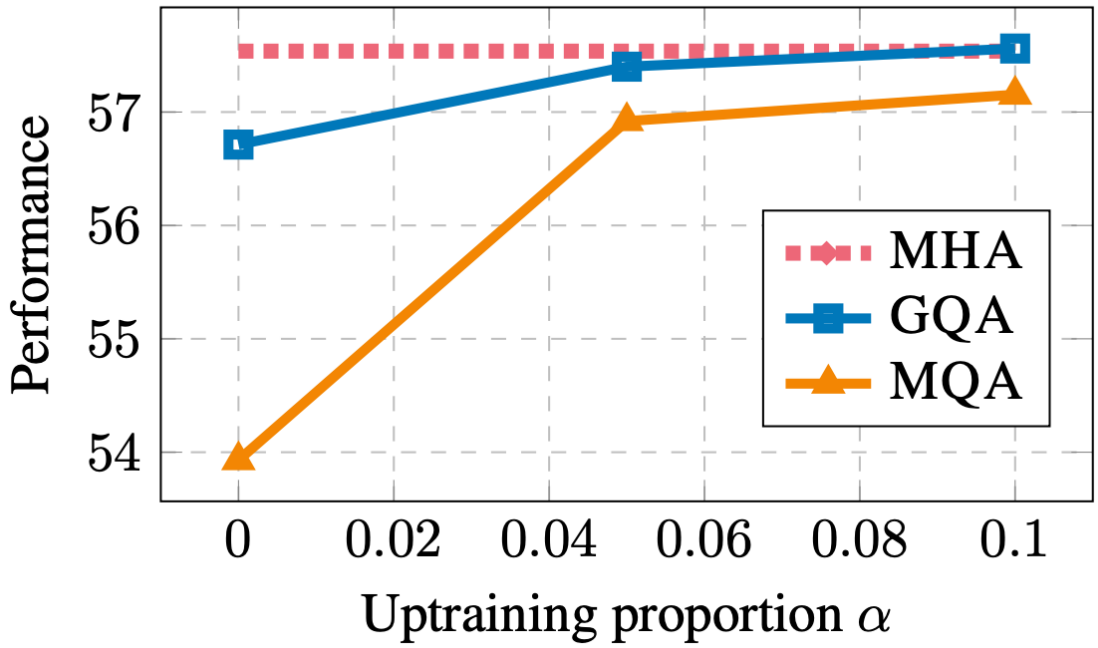
Trade off

- GQA trade off inference speed and performance



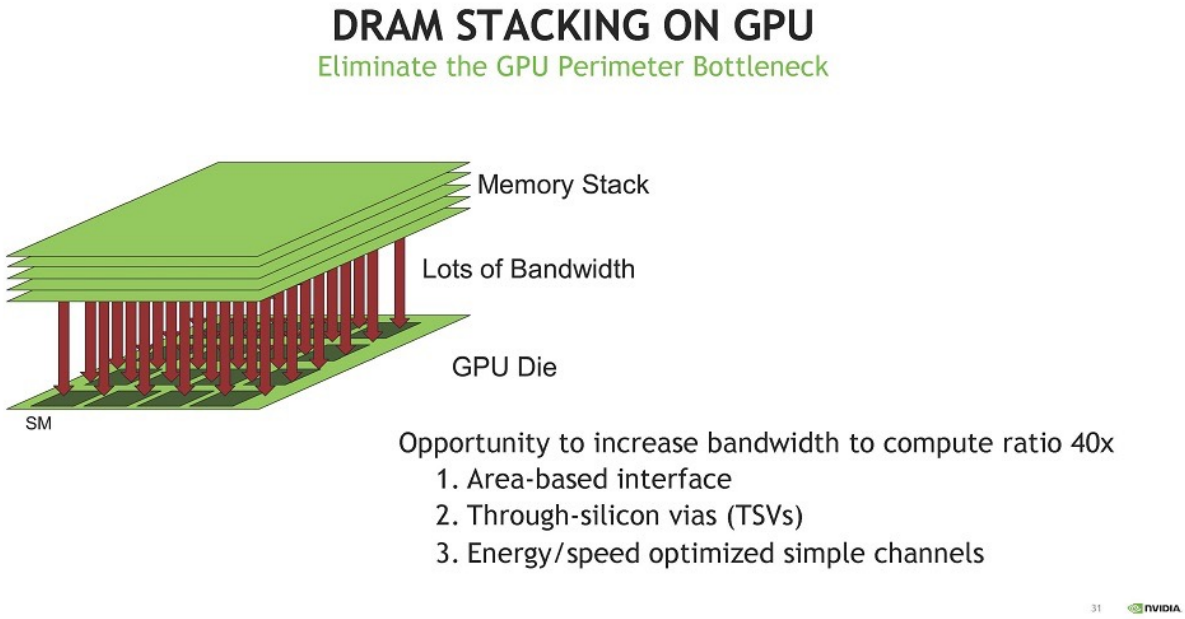
Group query attention

- Continue training to convert MHA to GQA to MQA



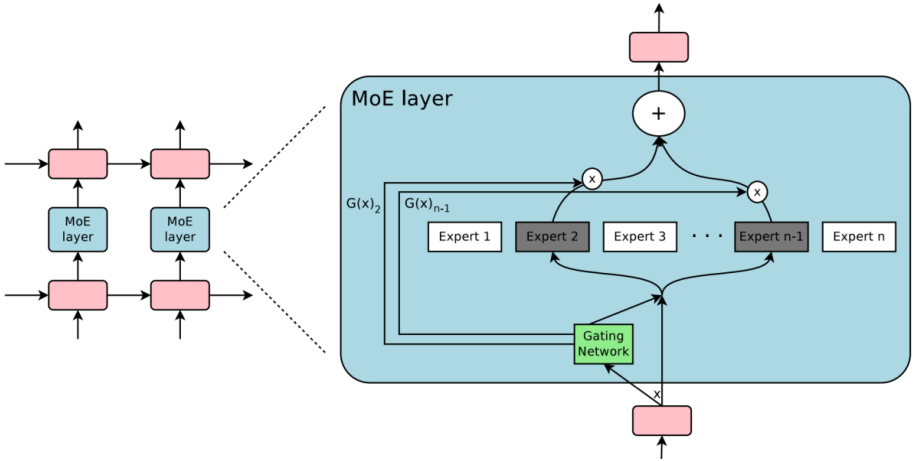
DRAM stacking on GPU

- 3D-stacks HBM memory directly on top of the processing cores
- Both compute and memory bandwidth scale with GPU die area



Mixture of experts

- Select a subset of FFNs to execute.
  - Decouple compute from parameters.



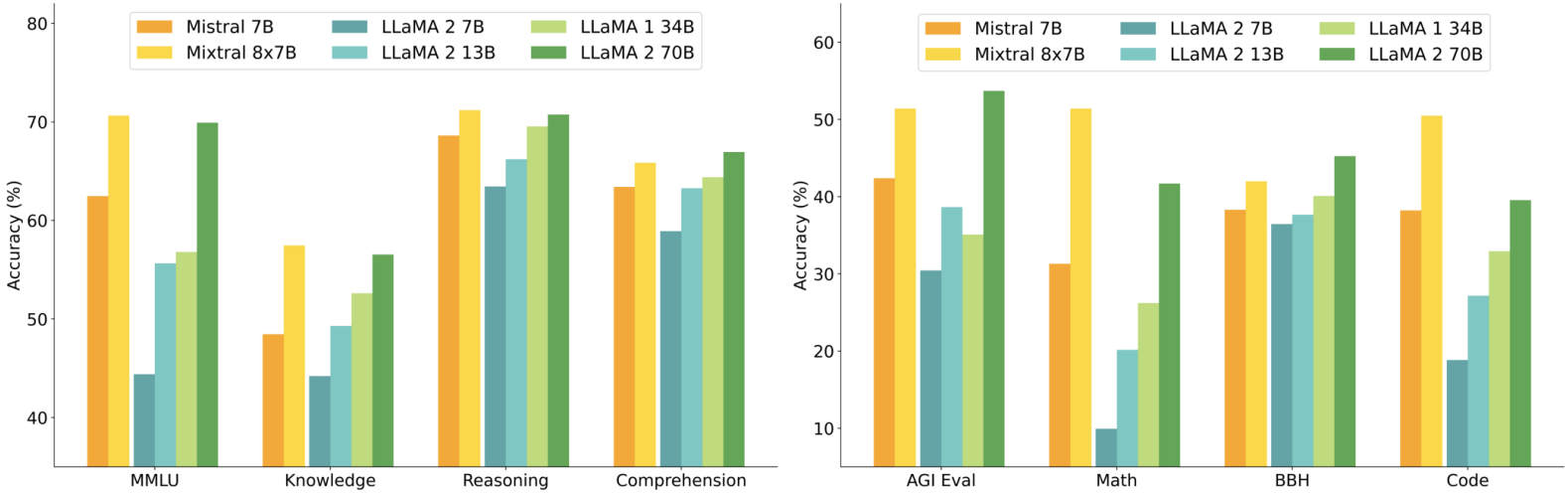
$$g(x) = \text{softmax}(\text{keep\_top\_k}(f_{\text{gating}}(x), k))$$

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

Shazeer, Noam, et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." (2017).

Mixtral 8x7B

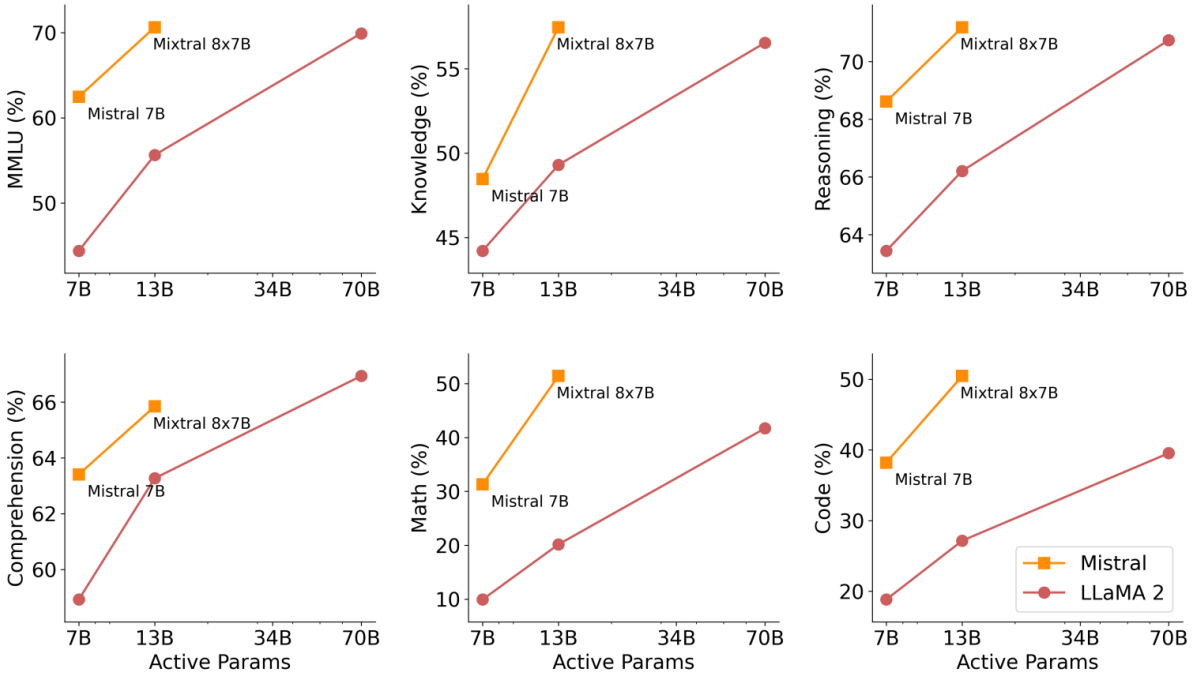
- Better performance



Mistral AI. "Mixtral of experts." (2024).

Mixtral 8x7B

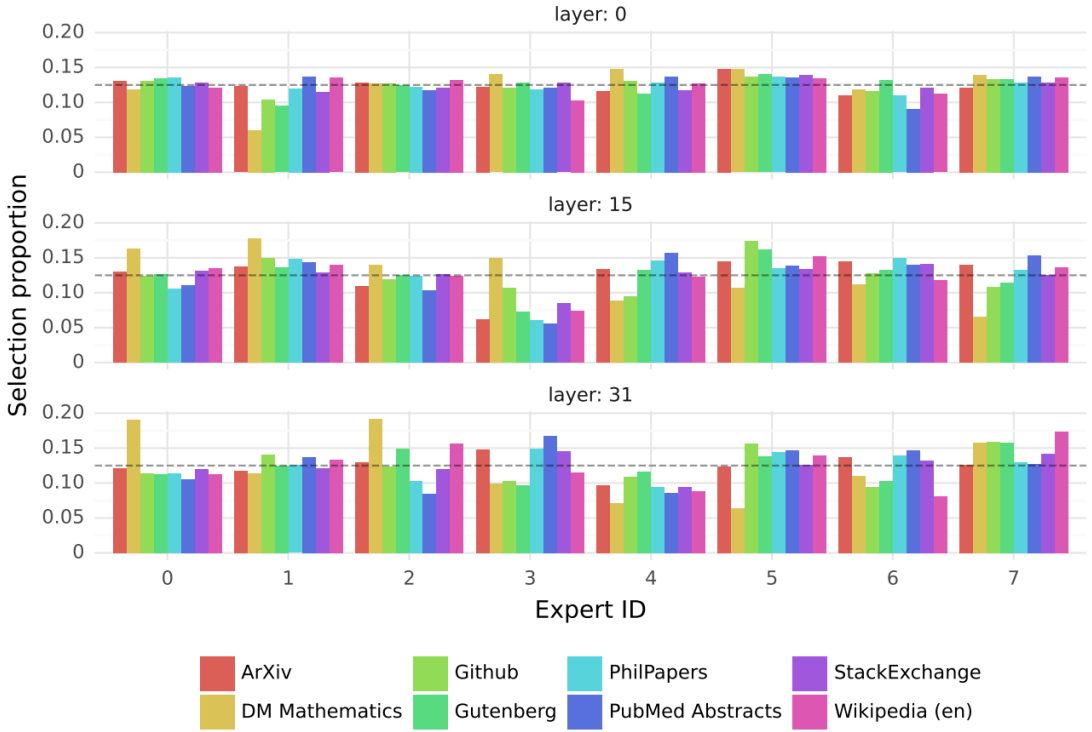
- Same active params, better performance



Mistral AI. "Mixtral of experts." (2024).

Mixtral 8x7B

- Expert selection may not be interpretable



Mistral AI. "Mixtral of experts." (2024).

Reasoning

- Simple arithmetic problems

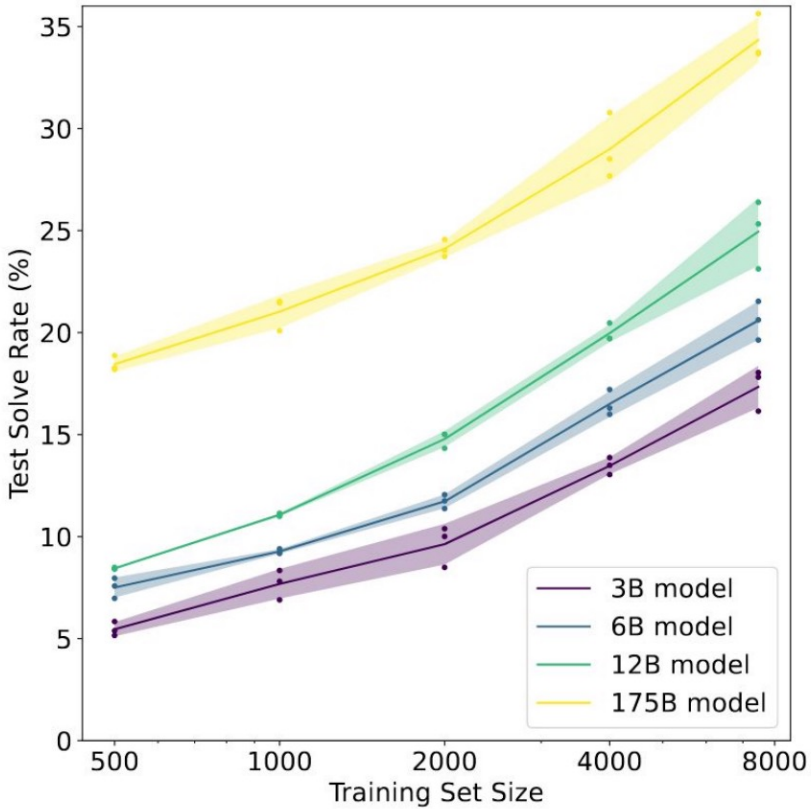
Problem: Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?
Solution: Beth bakes 4 2 dozen batches of cookies for a total of 4*2 = <<4*2=8>>8 dozen cookies
There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of 12*8 = <<12*8=96>>96 cookies
She splits the 96 cookies equally amongst 16 people so they each eat 96/16 = <<96/16=6>>6 cookies
Final Answer: 6

Problem: Mrs. Lim milks her cows twice a day. Yesterday morning, she got 68 gallons of milk and in the evening, she got 82 gallons. This morning, she got 18 gallons fewer than she had yesterday morning. After selling some gallons of milk in the afternoon, Mrs. Lim has only 24 gallons left. How much was her revenue for the milk if each gallon costs $3.50?
Mrs. Lim got 68 gallons - 18 gallons = <<68-18=50>>50 gallons this morning.
So she was able to get a total of 68 gallons + 82 gallons + 50 gallons = <<68+82+50=200>>200 gallons.
She was able to sell 200 gallons - 24 gallons = <<200-24=176>>176 gallons.
Thus, her total revenue for the milk is $3.50/gallon x 176 gallons = $<<3.50*176=616>>616.
Final Answer: 616

Problem: Tina buys 3 12-packs of soda for a party. Including Tina, 6 people are at the party. Half of the people at the party have 3 sodas each, 2 of the people have 4, and 1 person has 5. How many sodas are left over when the party is over?
Solution: Tina buys 3 12-packs of soda, for 3*12= <<3*12=36>>36 sodas
6 people attend the party, so half of them is 6/2= <<6/2=3>>3 people
Each of those people drinks 3 sodas, so they drink 3*3=<<3*3=9>>9 sodas
Two people drink 4 sodas, which means they drink 2*4=<<4*2=8>>8 sodas
With one person drinking 5, that brings the total drank to 5+9+8+3= <<5+9+8+3=25>>25 sodas
As Tina started off with 36 sodas, that means there are 36-25=<<36-25=11>>11 sodas left
Final Answer: 11

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., ... & Schulman, J. Training verifiers to solve math word problems. (2021)

Finetuning requires lots of data



- Not scalable to rely on humans to curate reasoning data
- to achieve > 80%, needs 100 times more fine-tuning data for 175B model

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., ... & Schulman, J. Training verifiers to solve math word problems. (2021)

Scratchpad

- Scratchpad for finetuning on step by step reasoning

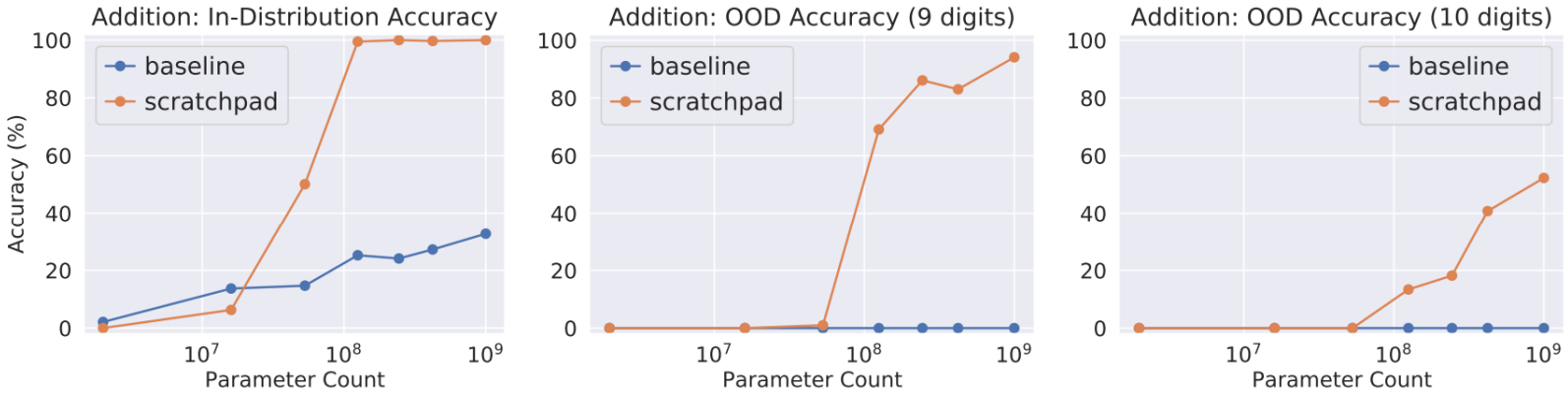
```
Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 ,  C: 0
2 + 5 , 6 C: 1  # added 9 + 7 = 6 carry 1
, 8 6 C: 0  # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6
```

Nye, Maxwell, et al. "Show your work: Scratchpads for intermediate computation with language models." (2021)

Scratchpad

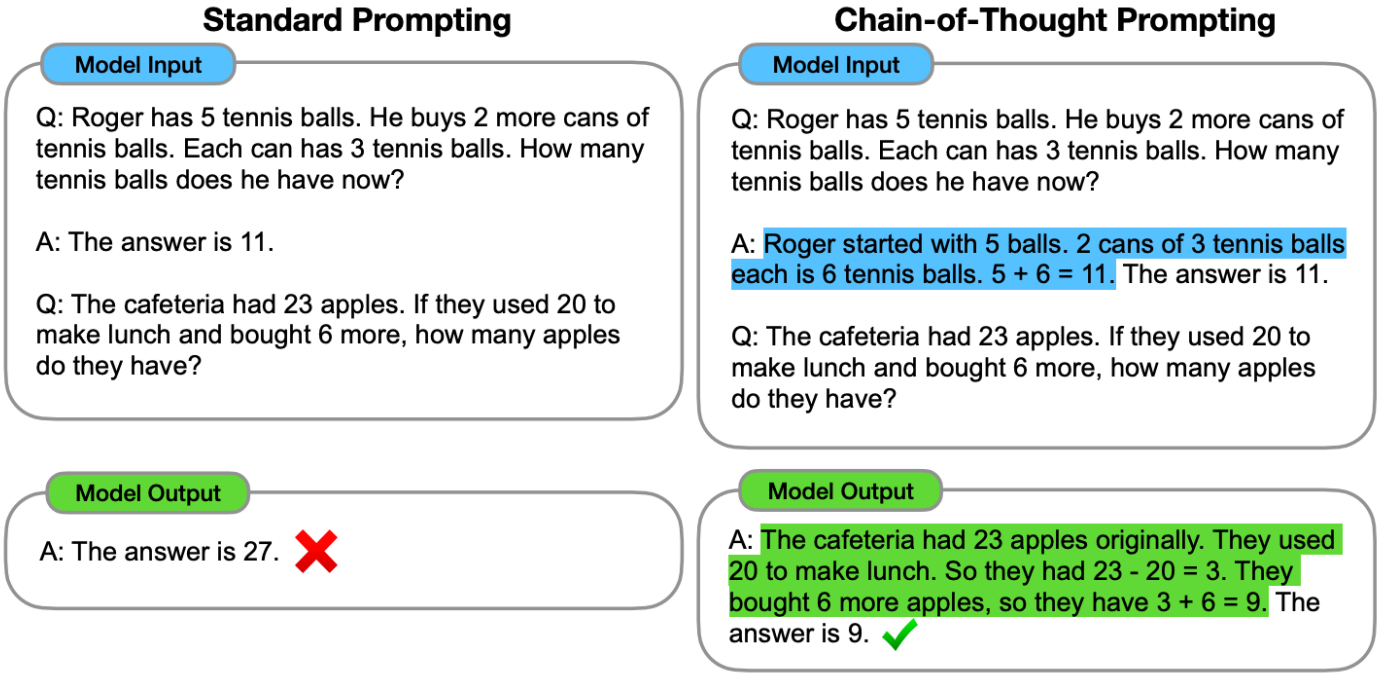
- Scratchpad for finetuning on step by step reasoning
  - Improved reasoning



Nye, Maxwell, et al. "Show your work: Scratchpads for intermediate computation with language models." (2021)

CoT

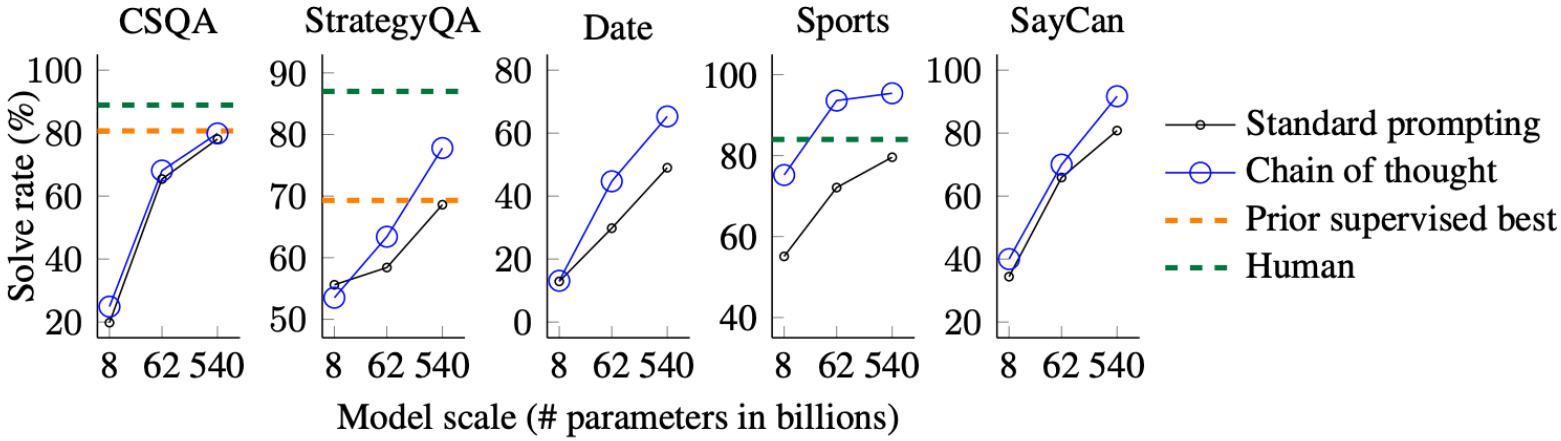
- Chain-of-thought prompting



Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." (2022)

CoT

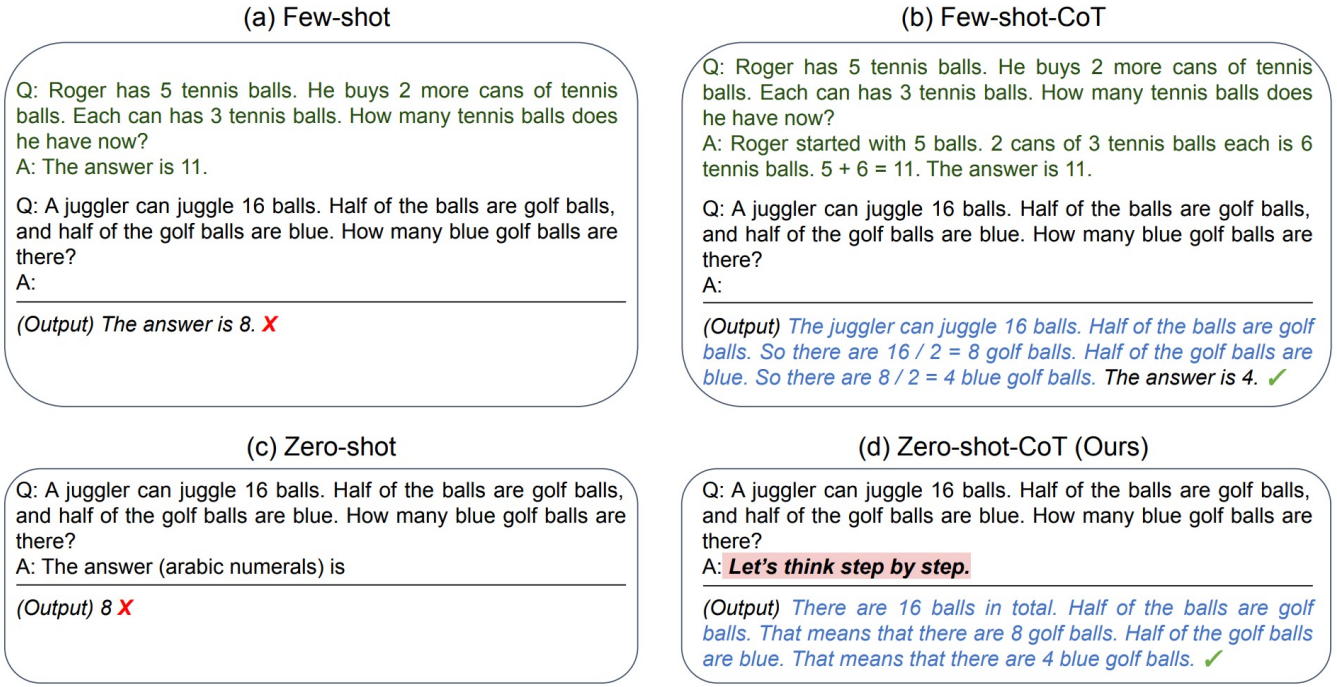
- Chain-of-thought prompting
  - Improved reasoning



Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." (2022)

Zero-shot CoT

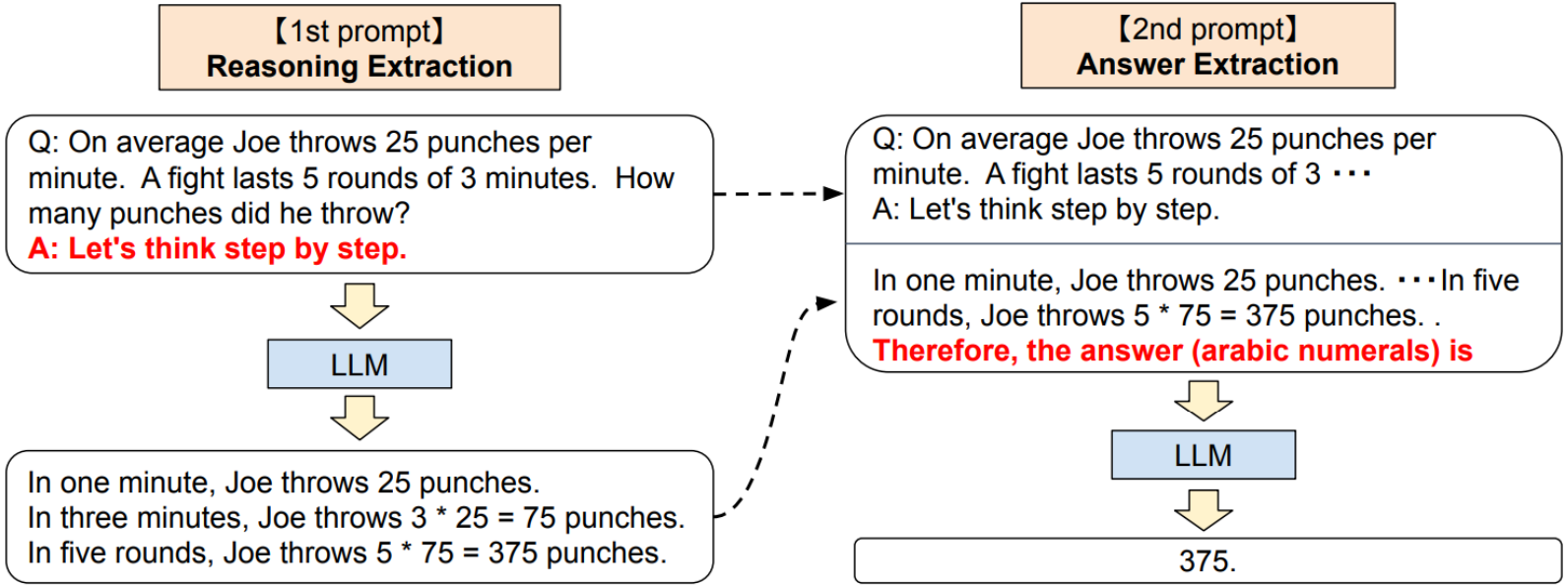
- Chain-of-thought prompting without manual examples



Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. Large language models are zero-shot reasoners. (2022).

Zero-shot CoT

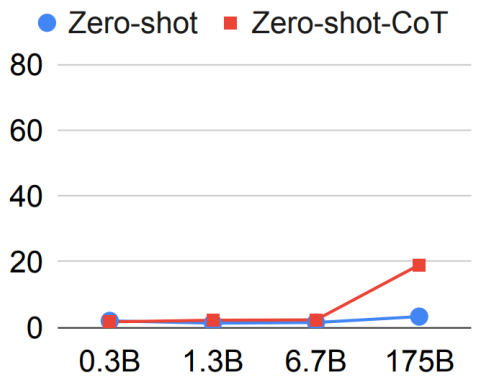
- Chain-of-thought prompting without manual examples



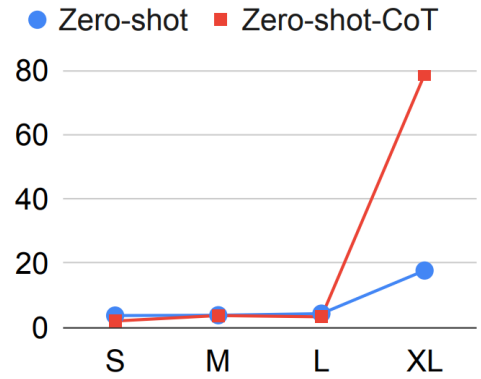
Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. Large language models are zero-shot reasoners. (2022).

Zero-shot CoT

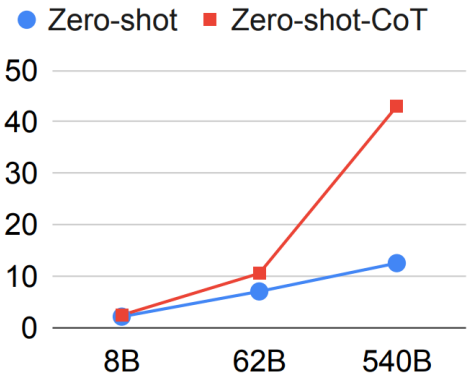
- Chain-of-thought prompting without manual examples



(a) MultiArith on Original GPT-3



(b) MultiArith on Instruct GPT-3

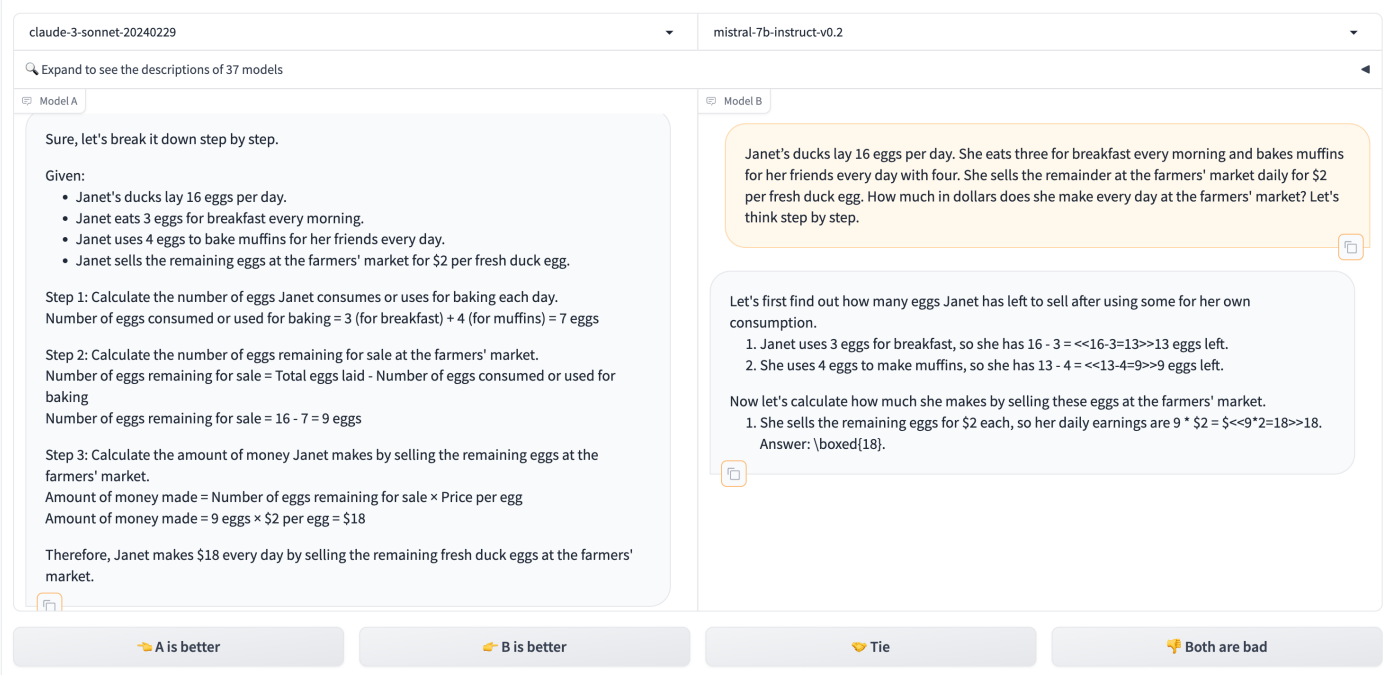


(c) GMS8K on PaLM

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. Large language models are zero-shot reasoners. (2022).

Zero-shot CoT

- Chain-of-thought prompting without manual examples



Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. Large language models are zero-shot reasoners. (2022).

Process feedback

- Let's verify step by step

The denominator of a fraction is 7 less than 3 times the numerator. If the fraction is equivalent to $2/5$, what is the numerator of the fraction? (Answer: $\boxed{14}$)

☹ 😐 😎 Let's call the numerator x.

☹ 😐 😎 So the denominator is 3x-7.

☹ 😐 😎 We know that x/(3x-7) = 2/5.

☹ 😐 😎 So 5x = 2(3x-7).

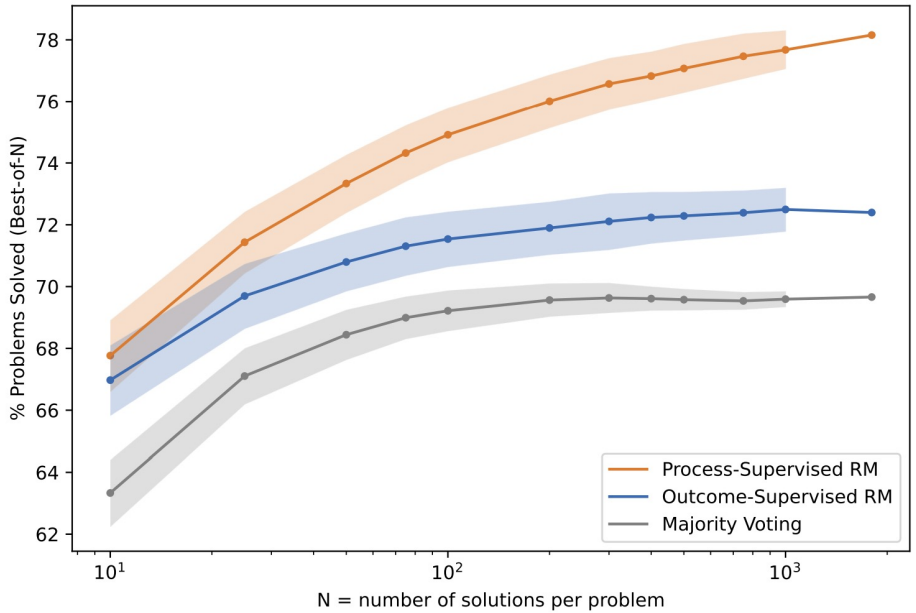
☹ 😐 😎 5x = 6x - 14.

☹ 😐 😎 So x = 7.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., ... & Cobbe, K. Let's Verify Step by Step. (2023).

Process feedback

	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	**78.2**	69.6

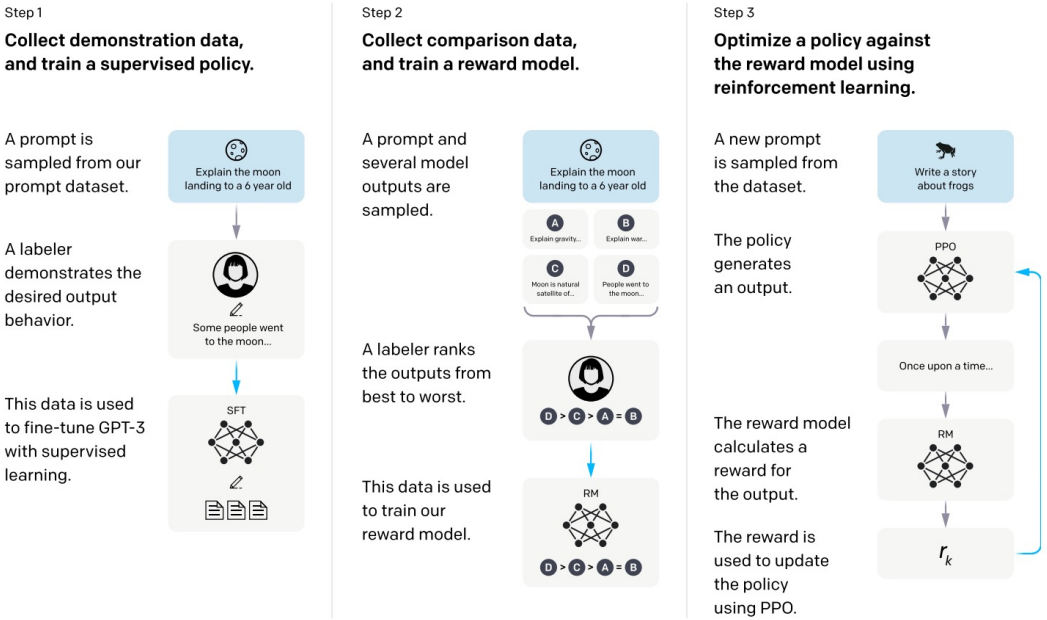


- Process supervision does not saturate early
- However, process supervision is very expensive

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., ... & Cobbe, K. Let's Verify Step by Step. (2023).

RLHF

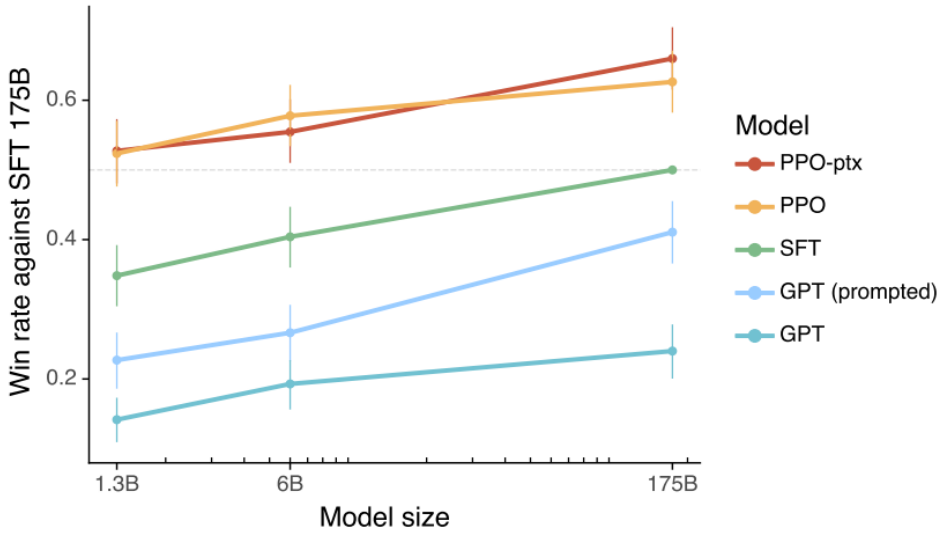
- Collecting demonstration for supervised finetuning
- Train a reward model for reinforcement learning



Ouyang, Long, et al. "Training language models to follow instructions with human feedback." (2022)

RLHF

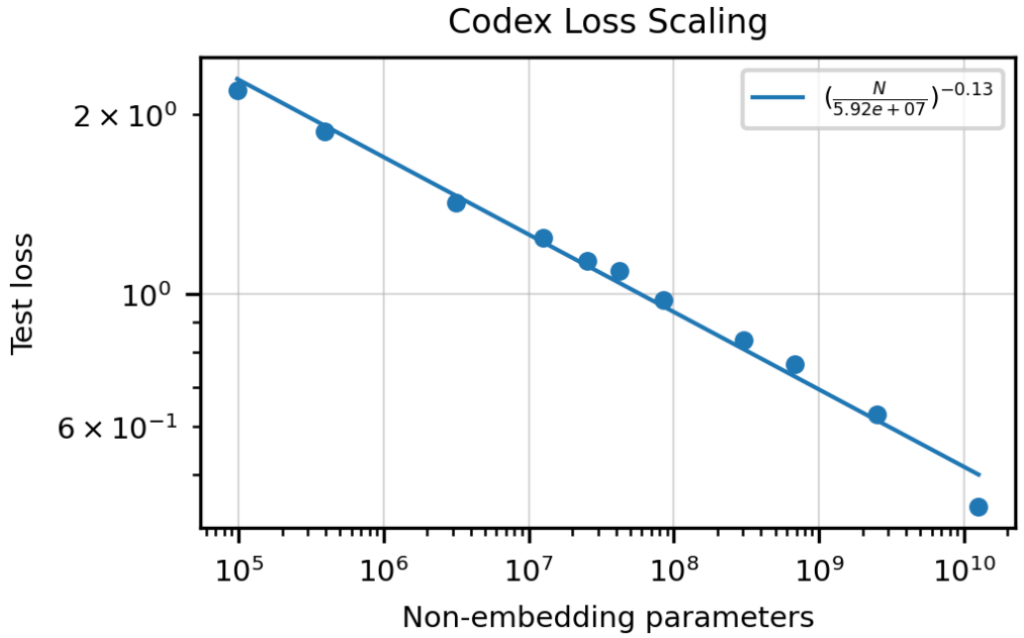
- RLHF outperforms SFT and scales well to larger model size



Ouyang, Long, et al. "Training language models to follow instructions with human feedback." (2022)

Code generation loss scaling

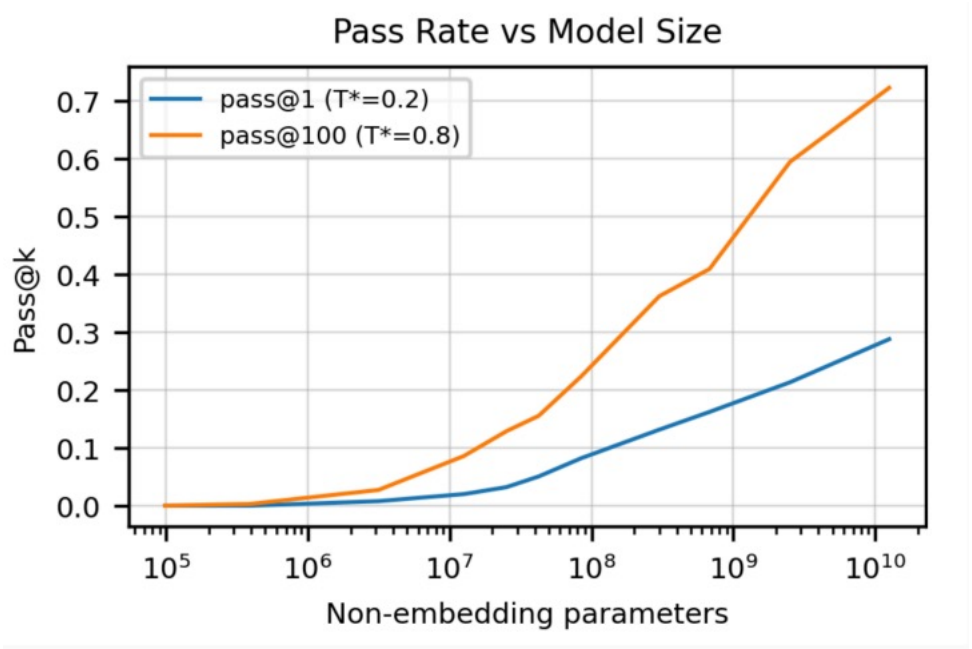
- Scaling works for code synthesis too
- Test loss shows power-law w.r.t model parameters



Chen, Mark, et al. "Evaluating large language models trained on code." (2021).

Code generation accuracy scaling

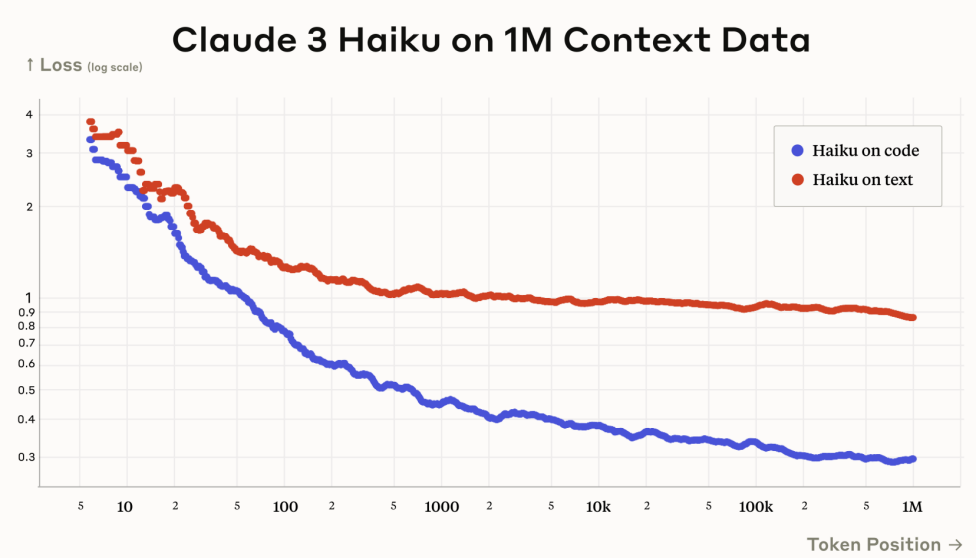
- Pass@1 and pass@100 both increase with larger model



Chen, Mark, et al. "Evaluating large language models trained on code." (2021).

Large context code loss

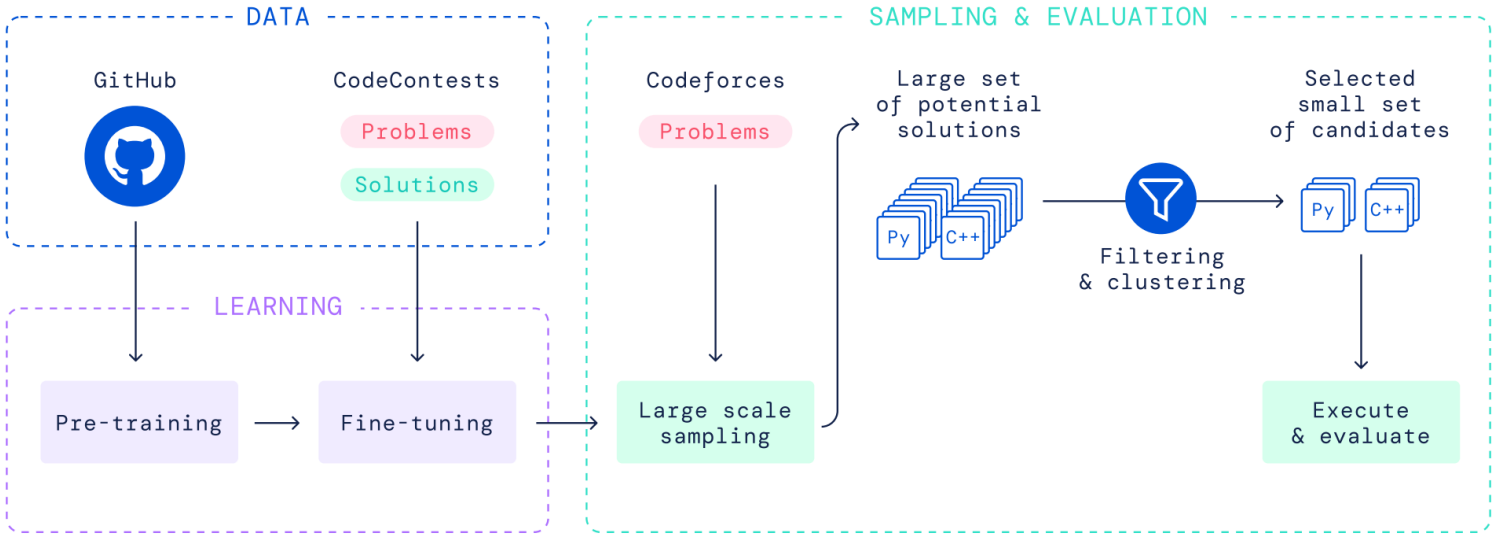
- Power-law on the context size axis too
- Loss on code are generally lower compared with on text



The Claude 3 Model Family: Opus, Sonnet, Haiku. 2024

AlphaCode

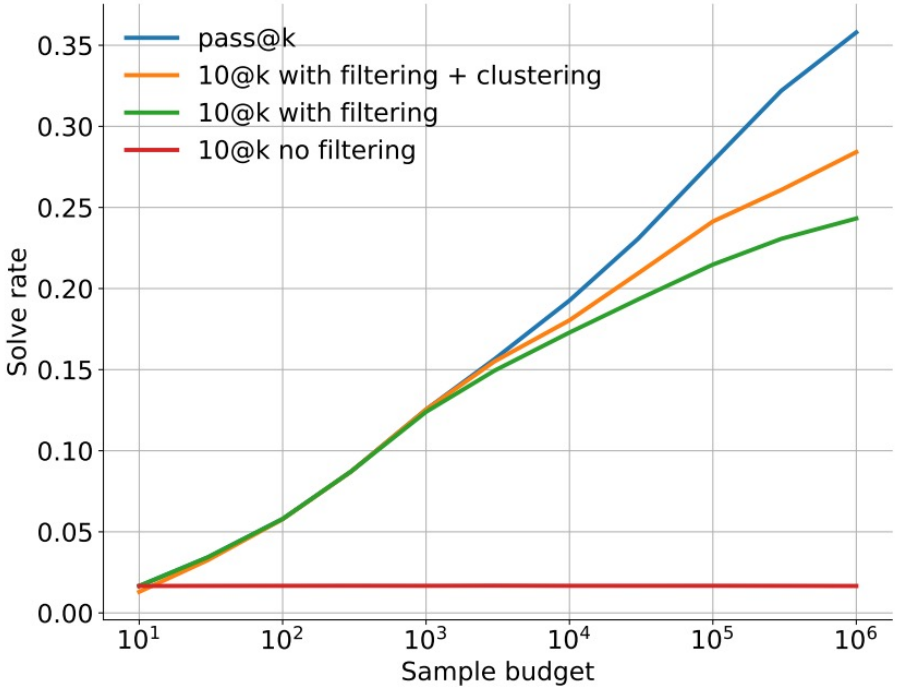
- Inference optimization by large scale sampling and filtering



"Competition-Level Code Generation with AlphaCode" (2022)
"AlphaCode 2 Technical Report" (2023)

AlphaCode

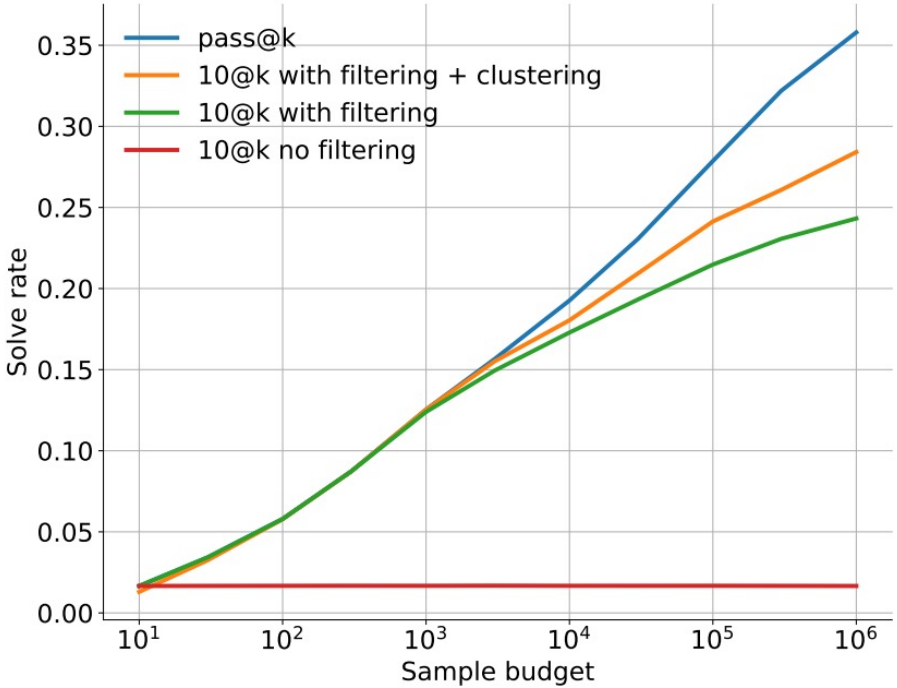
- Inference optimization by large scale sampling and filtering



"Competition-Level Code Generation with AlphaCode" (2022)
"AlphaCode 2 Technical Report" (2023)

AlphaCode

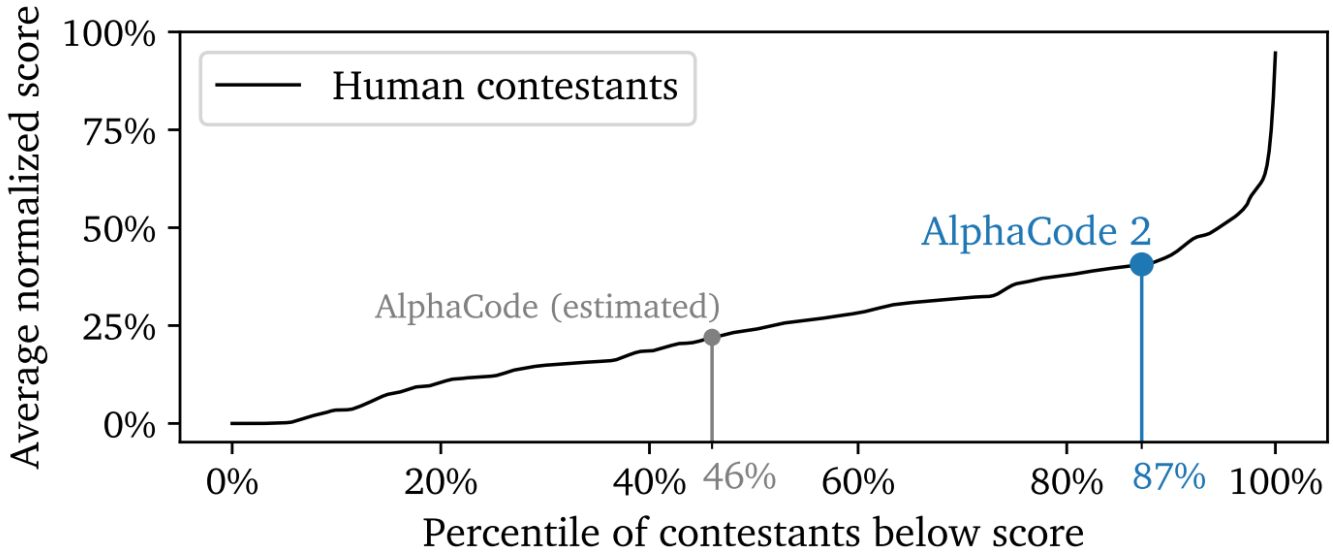
- Inference optimization by large scale sampling and filtering



"Competition-Level Code Generation with AlphaCode" (2022)
"AlphaCode 2 Technical Report" (2023)

AlphaCode2

- AlphaCode2 replaces base model with Gemini pro
- A good pretrained model plus inference time sampling



"Competition-Level Code Generation with AlphaCode" (2022)
"AlphaCode 2 Technical Report" (2023)

Pretraining data

- Most data is from common crawl

Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3†	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
FreeLaw	51.15 GiB	6.12%	1.5	76.73 GiB	15.06 KiB
Stack Exchange	32.20 GiB	5.13%	2.0	64.39 GiB	2.16 KiB
USPTO Backgrounds	22.90 GiB	3.65%	2.0	45.81 GiB	4.08 KiB
PubMed Abstracts	19.26 GiB	3.07%	2.0	38.53 GiB	1.30 KiB
Gutenberg (PG-19)†	10.88 GiB	2.17%	2.5	27.19 GiB	398.73 KiB
OpenSubtitles†	12.98 GiB	1.55%	1.5	19.47 GiB	30.48 KiB
Wikipedia (en)†	6.38 GiB	1.53%	3.0	19.13 GiB	1.11 KiB
DM Mathematics†	7.75 GiB	1.24%	2.0	15.49 GiB	8.00 KiB
Ubuntu IRC	5.52 GiB	0.88%	2.0	11.03 GiB	545.48 KiB
BookCorpus2	6.30 GiB	0.75%	1.5	9.45 GiB	369.87 KiB
EuroParl†	4.59 GiB	0.73%	2.0	9.17 GiB	68.87 KiB
HackerNews	3.90 GiB	0.62%	2.0	7.80 GiB	4.92 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails†	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB
The Pile	**825.18 GiB**			**1254.20 GiB**	**5.91 KiB**

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., ... & Leahy, C. The pile: An 800gb dataset of diverse text for language modeling. (2020).

Filtering data

- Llama's dataset is calibrated toward Wikipedia
- Outperforming GPT3 and other models

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

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

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

"LLaMA: Open and Efficient Foundation Language Models" (2023).

RedPajam

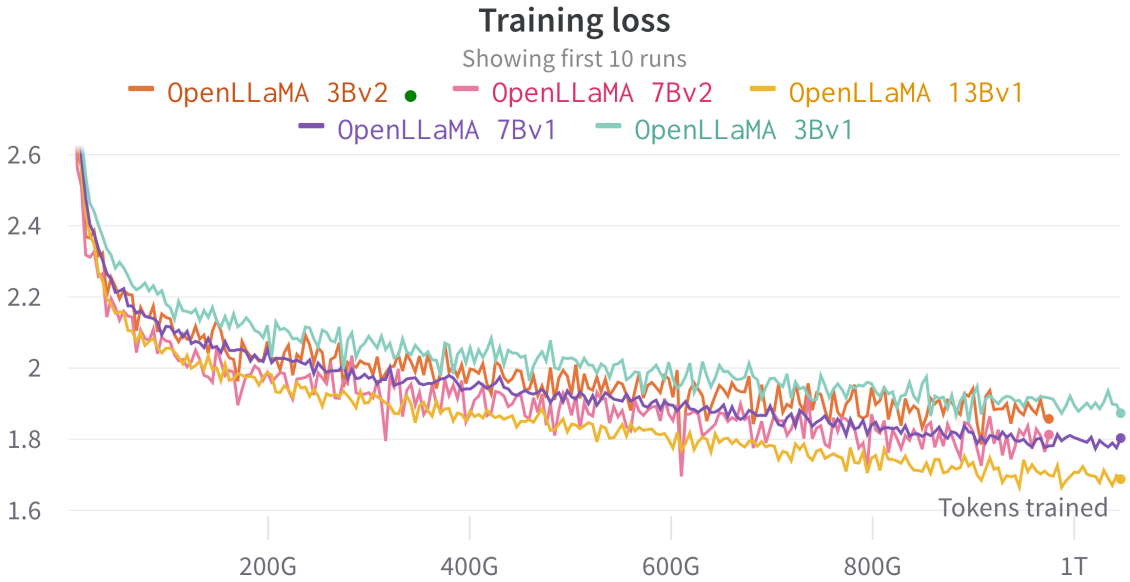
- RedPajam: an open-source dataset

	RedPajama	LLaMA*
CommonCrawl	878 billion	852 billion
C4	175 billion	190 billion
Github	59 billion	100 billion
Books	26 billion	25 billion
ArXiv	28 billion	33 billion
Wikipedia	24 billion	25 billion
StackExchange	20 billion	27 billion
Total	1.2 trillion	1.25 trillion

https://www.together.ai/blog/redpajama

OpenLLaMA

- OpenLLaMA matches LLaMA performance

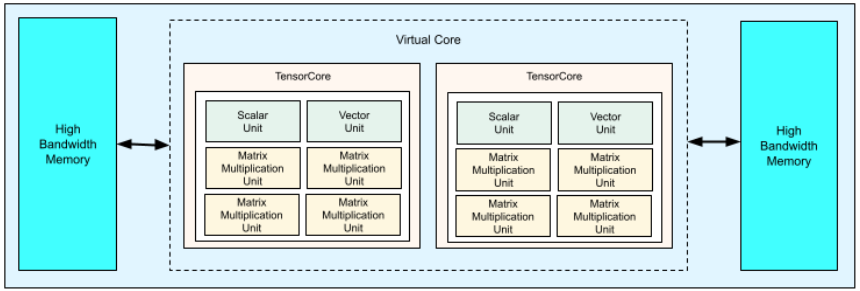


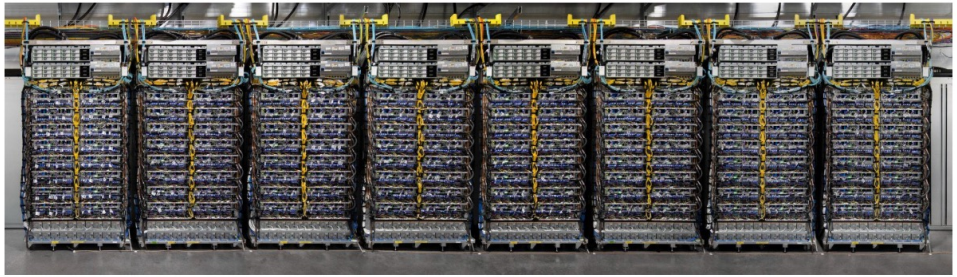
"OpenLLaMA: An Open Reproduction of LLaMA" (2024)

TPU / GPU

- Flops, HBM size and bandwidth, Interconnect

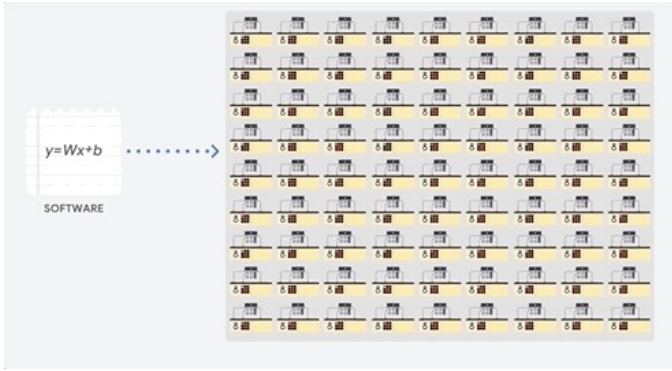
Key specifications	v4 Pod values
Peak compute per chip	275 teraflops (bf16 or int8)
HBM2 capacity and bandwidth	32 GiB, 1200 GBps
Measured min/mean/max power	90/170/192 W
TPU Pod size	4096 chips
Interconnect topology	3D mesh
Peak compute per Pod	1.1 exaflops (bf16 or int8)
All-reduce bandwidth per Pod	1.1 PB/s
Bisection bandwidth per Pod	24 TB/s



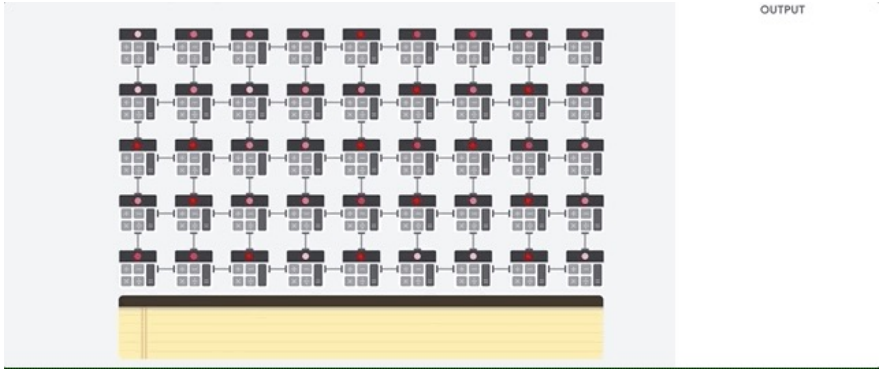


GEMM

- High-computational throughput on neural network calculations
- GPU compute unit is smaller but more, TPU has large compute units



GPU, many 8x4x8 ALU

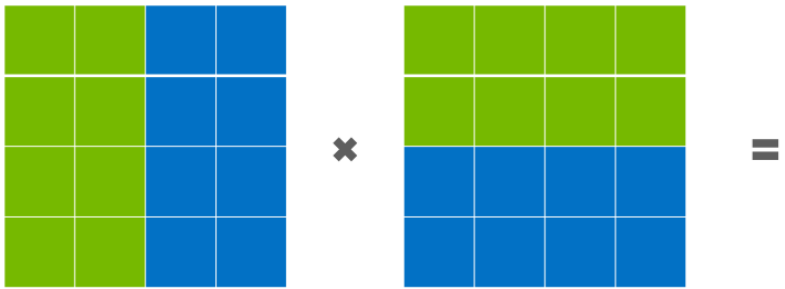


TPU, systolic array 8x128x128

https://cloud.google.com/tpu/docs/

Matmul sharding

- Simple case: sharded matmul Y = XA
- Row sharding on A, column sharding on X



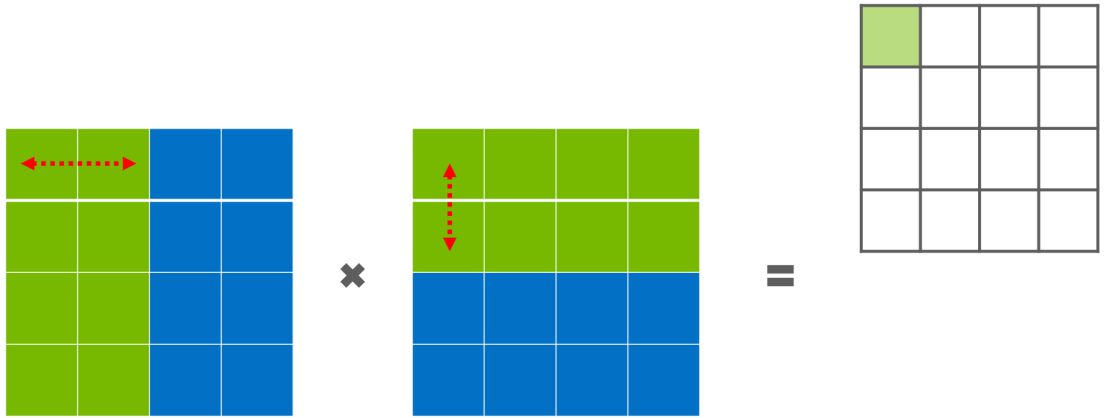
$$X = [X_1, X_2] \qquad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Shoeybi M, Patwary M, Puri R, LeGresley P, Casper J, Catanzaro B. Megatron-lm: Training multi-billion parameter language models using model parallelism. (2019)

Matmul sharding

- Row sharding on A, column sharding on X



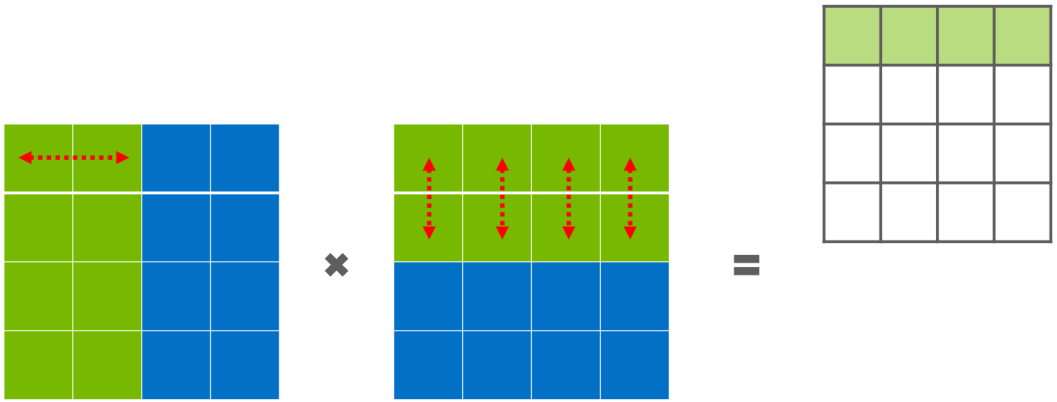
$$X = [X_1, X_2]$$

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Matmul sharding

- Row sharding on A, column sharding on X

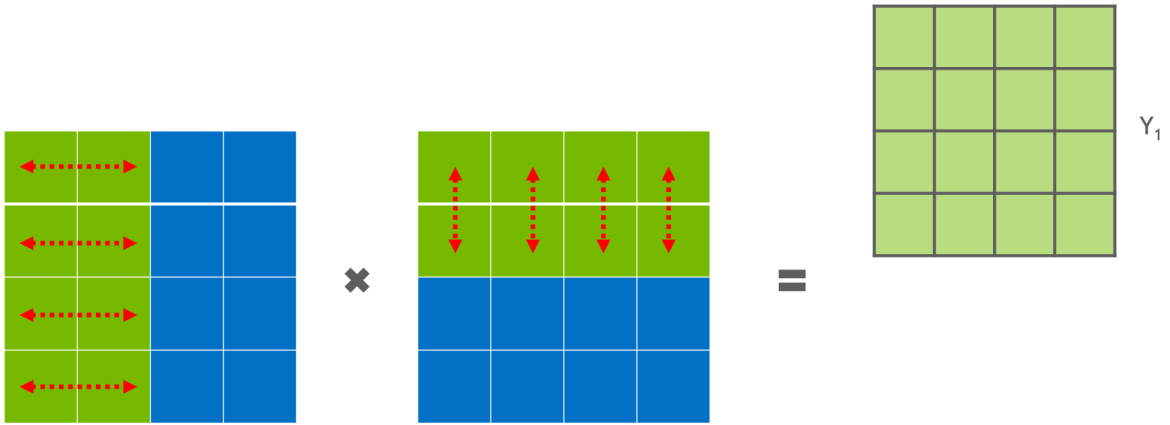


$$X = [X_1, X_2] \qquad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Matmul sharding

- Row sharding on A, column sharding on X



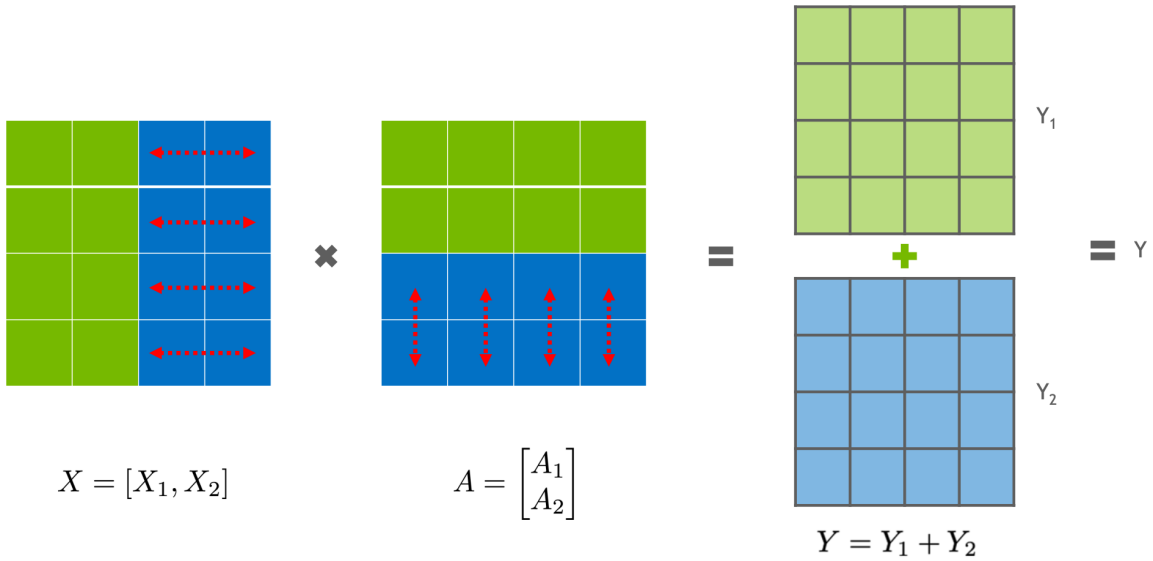
$X = [X_1, X_2]$

$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$

$Y = Y_1 + Y_2$

Matmul sharding

- Row sharding on A, column sharding on X



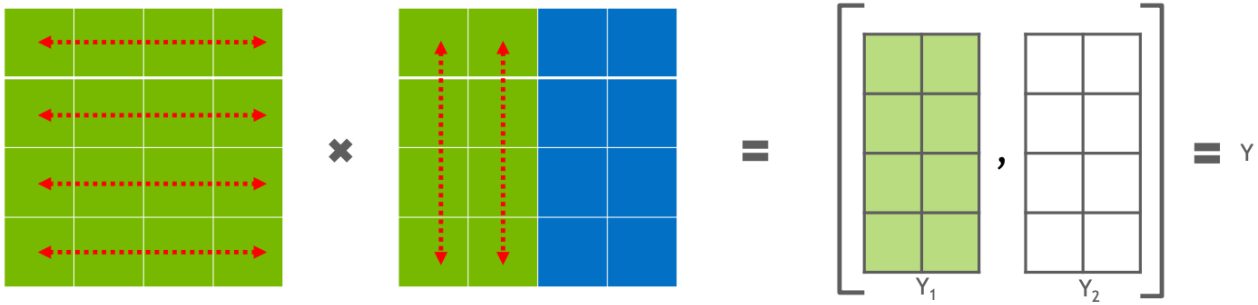
$X = [X_1, X_2]$

$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$

$Y = Y_1 + Y_2$

Matmul sharding

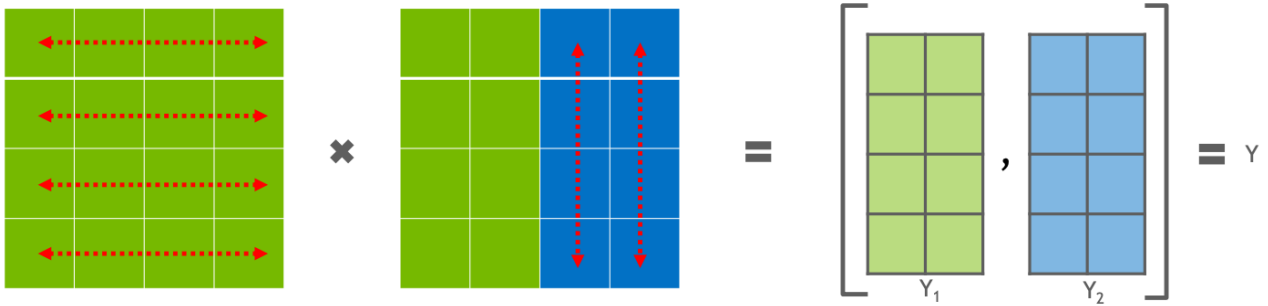
- Column sharding on A
- Replicate X



X $A = [A_1, A_2]$ $Y = [Y_1, Y_2]$

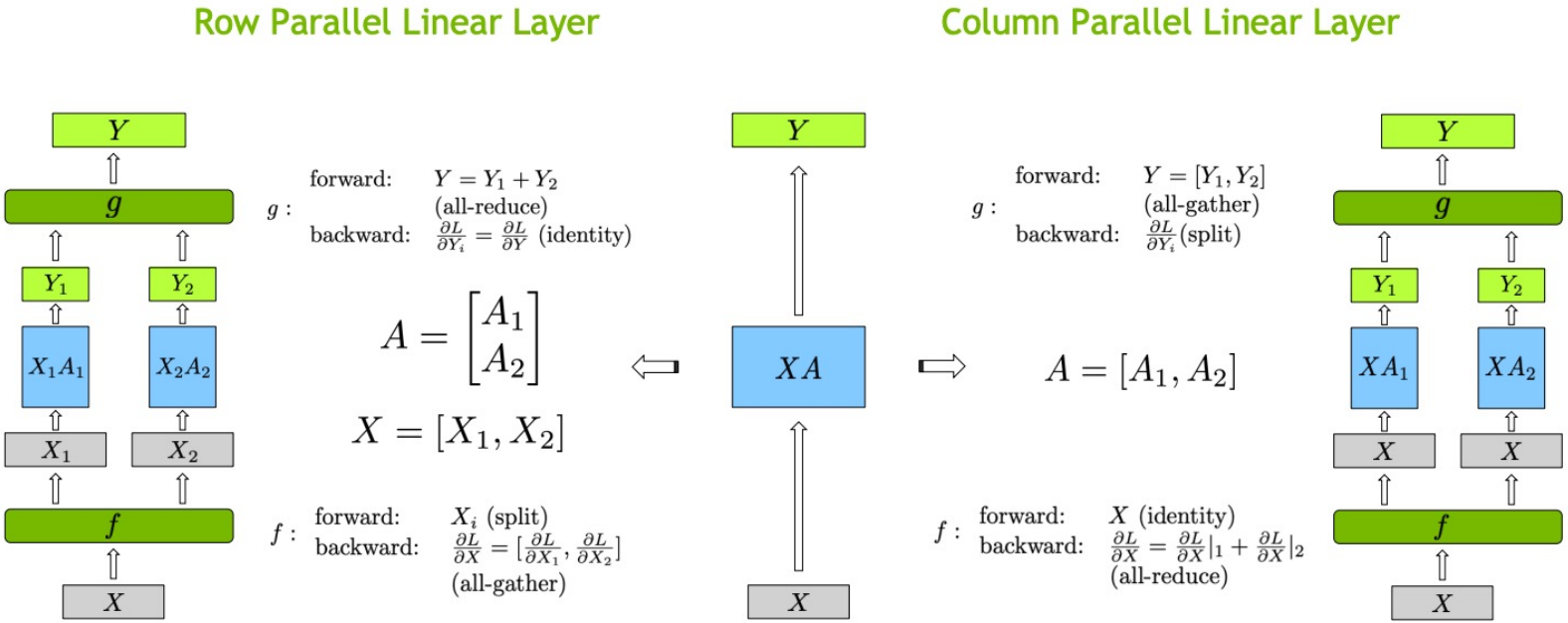
Matmul sharding

- Column sharding on A
- Replicate X



X $\quad A = [A_1, A_2]$ $\quad Y = [Y_1, Y_2]$

Matmul sharding



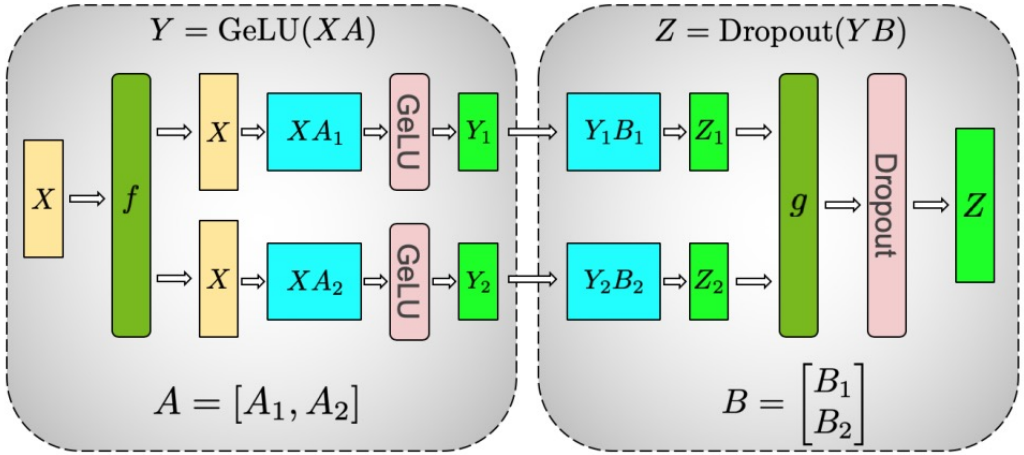
MLP sharding

- Y = GELU(XA)B
- Approach 1: split X column-wise and A row-wise
  - GeLU of sums != sum of GeLUs $X = [X_1, X_2]$ $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ $Y = \mathrm{GeLU}(X_1 A_1 + X_2 A_2)$
  - Requires synchronization before GeLU
- Approach 2: split A column-wise
  - No sharding on contraction dimension of X

    $A = [A_1, A_2]$ $[Y_1, Y_2] = [\mathrm{GeLU}(X A_1), \mathrm{GeLU}(X A_2)]$
  - No synchronization necessary

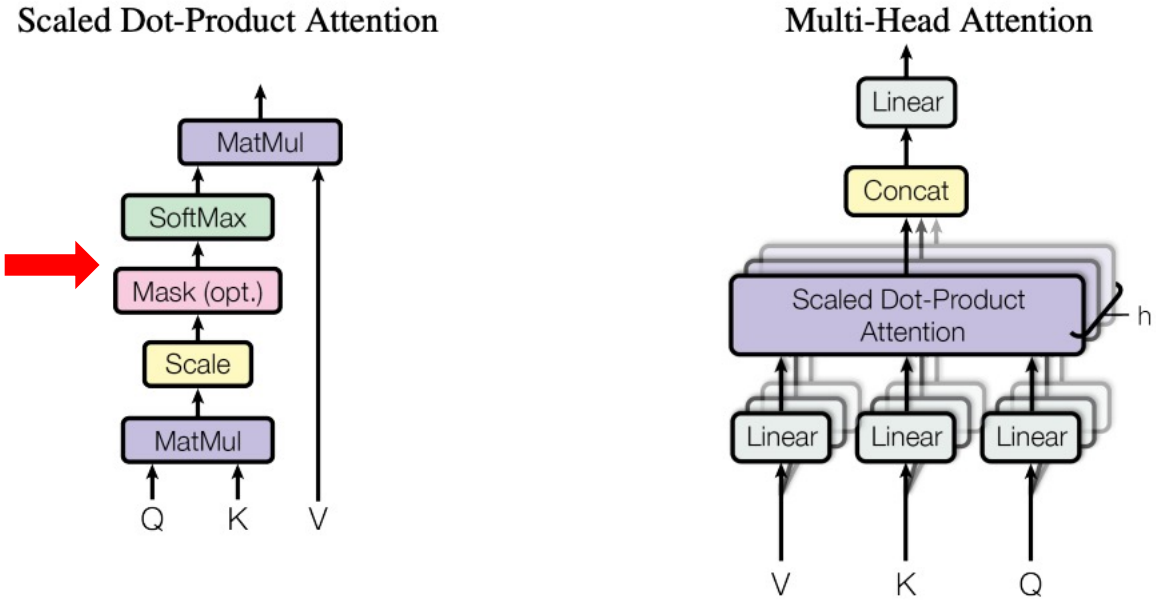
MLP sharding

- **f** and **g** are conjugate, **f** is **identity** operator in the forward pass and **all-reduce** in the backward pass while **g** is **all-reduce** in forward and **identity** in backward.



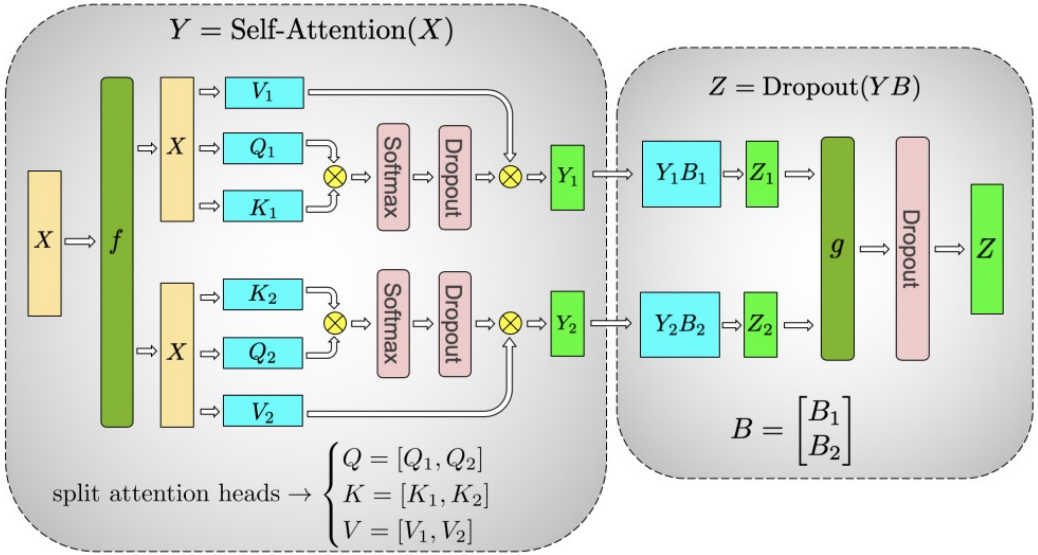
Attention sharding

- Nonlinearity in attention, so column sharding.



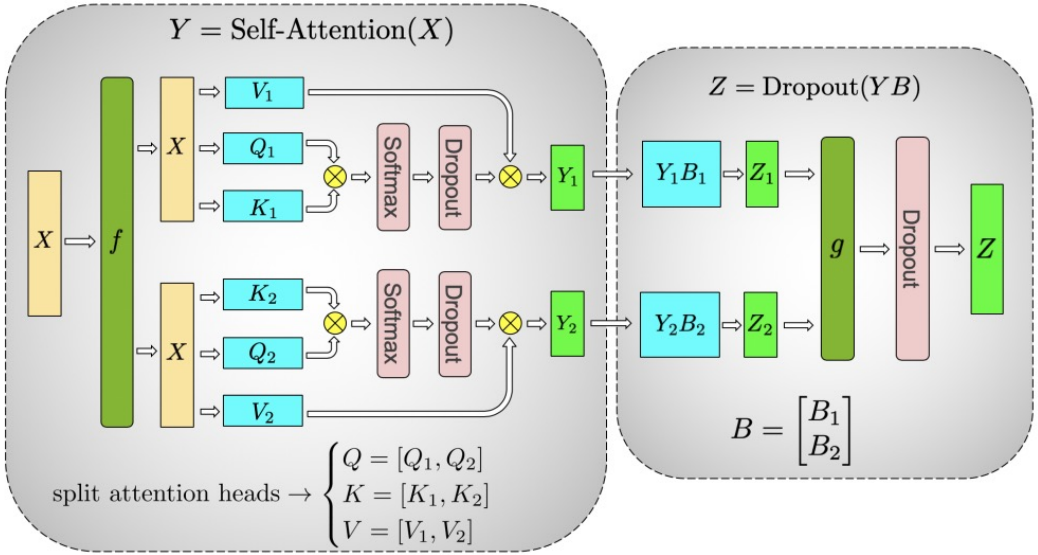
Attention sharding

- **f** and **g** are conjugate, **f** is **identity** operator in the forward pass and **all-reduce** in the backward pass while **g** is **all-reduce** in forward and **identity** in backward.



Attention sharding

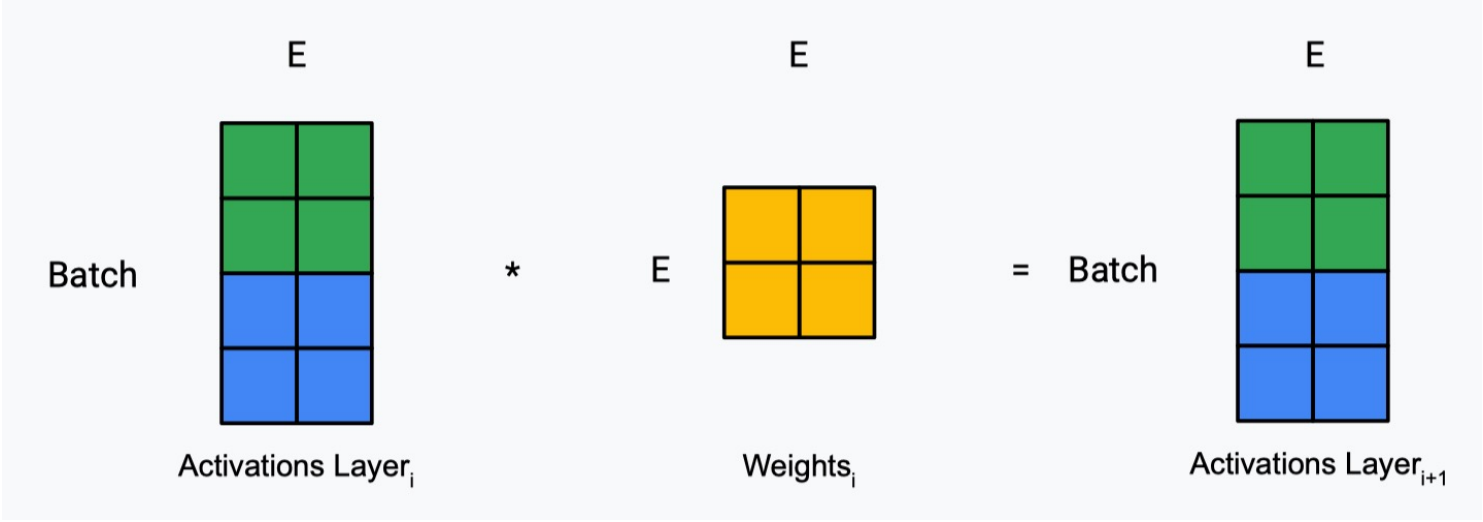
- **f** and **g** are conjugate, **f** is **identity** operator in the forward pass and **all-reduce** in the backward pass while **g** is **all-reduce** in forward and **identity** in backward.



Data parallelism

- Activations need to be sharded – they're much bigger that weights. We can either shard them by rows or columns.

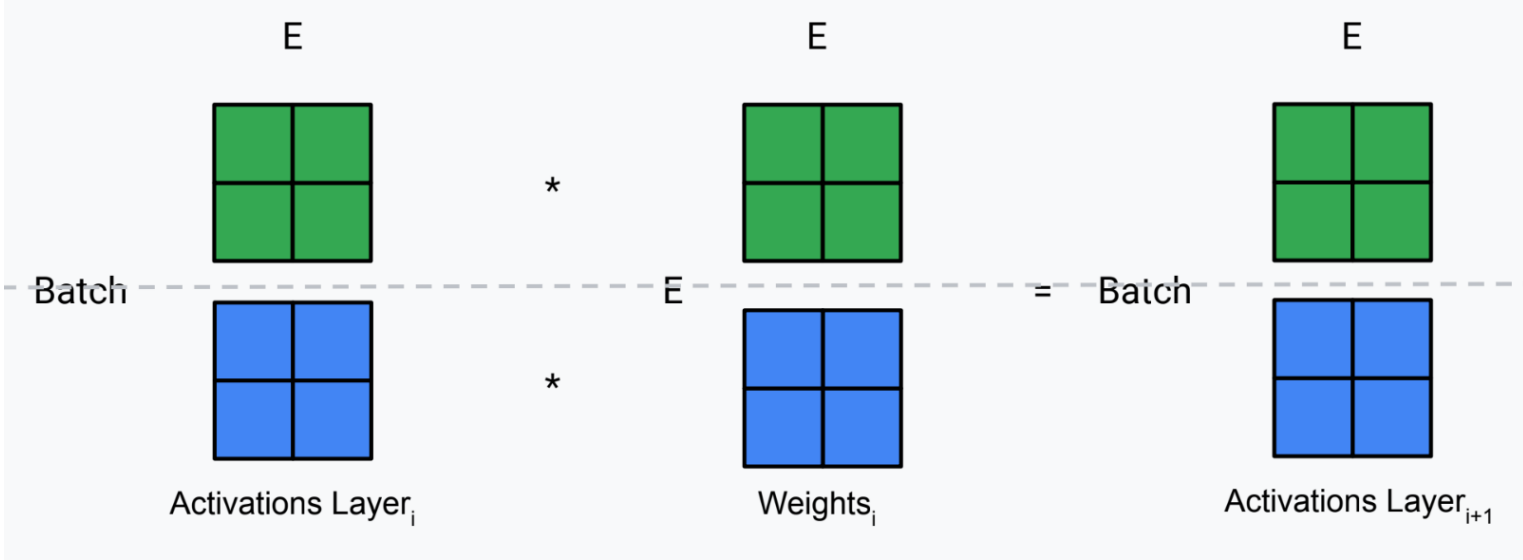
- It keeps working with a linear speedup as long as you scale batch linearly with number of TPUs



Rafi W. "Sharding Techniques Single Slice Sharding For Dense LLMs"

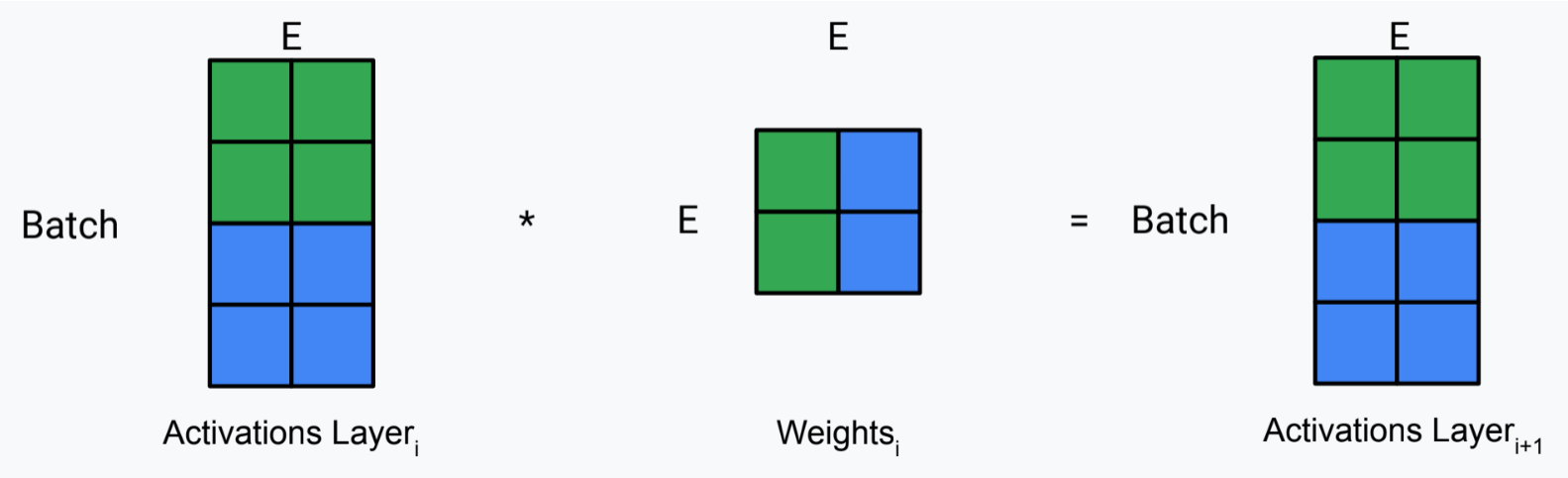
Data parallelism

- At 4 bytes per parameter we need 700GB for 175B GPT3
- But A100 has only 80GB.



Fully sharded data parallelism

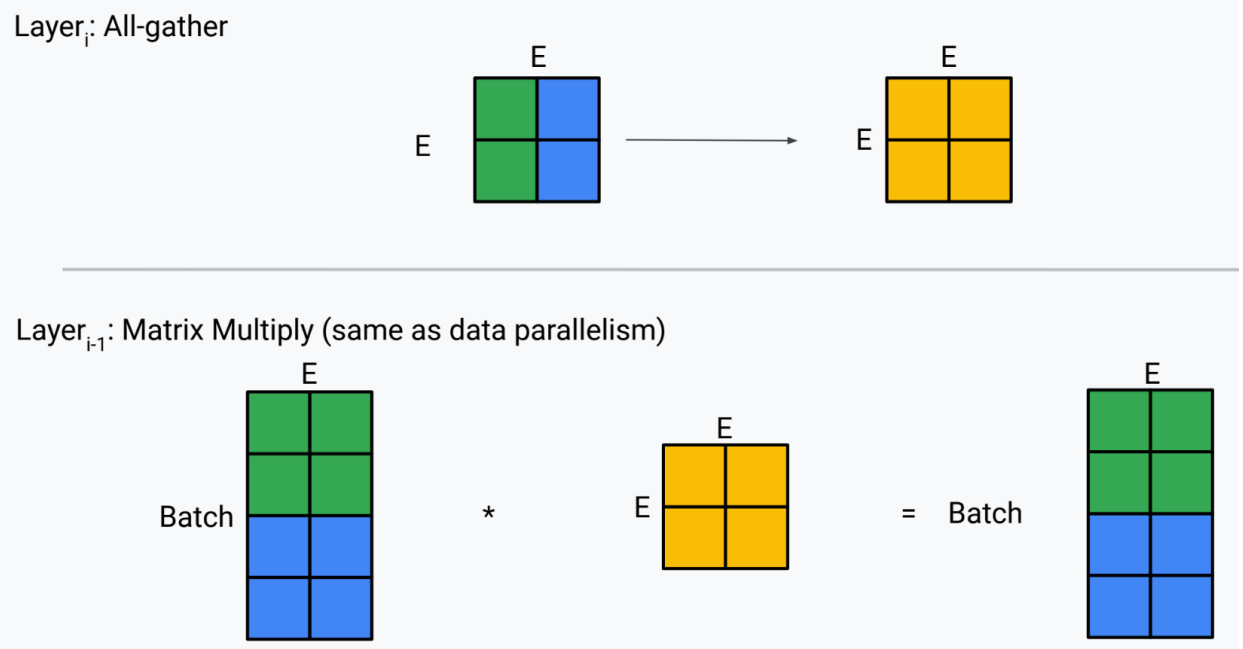
- All-gathering the next layer while doing the arithmetic on this layer
- Requires storing only 1 layer at a time (next slide)



Rafi W. “Sharding Techniques Single Slice Sharding For Dense LLMs”

Fully sharded data parallelism

- All-gathering the next layer while doing the arithmetic on this layer
- Requires storing only 1 layer at a time

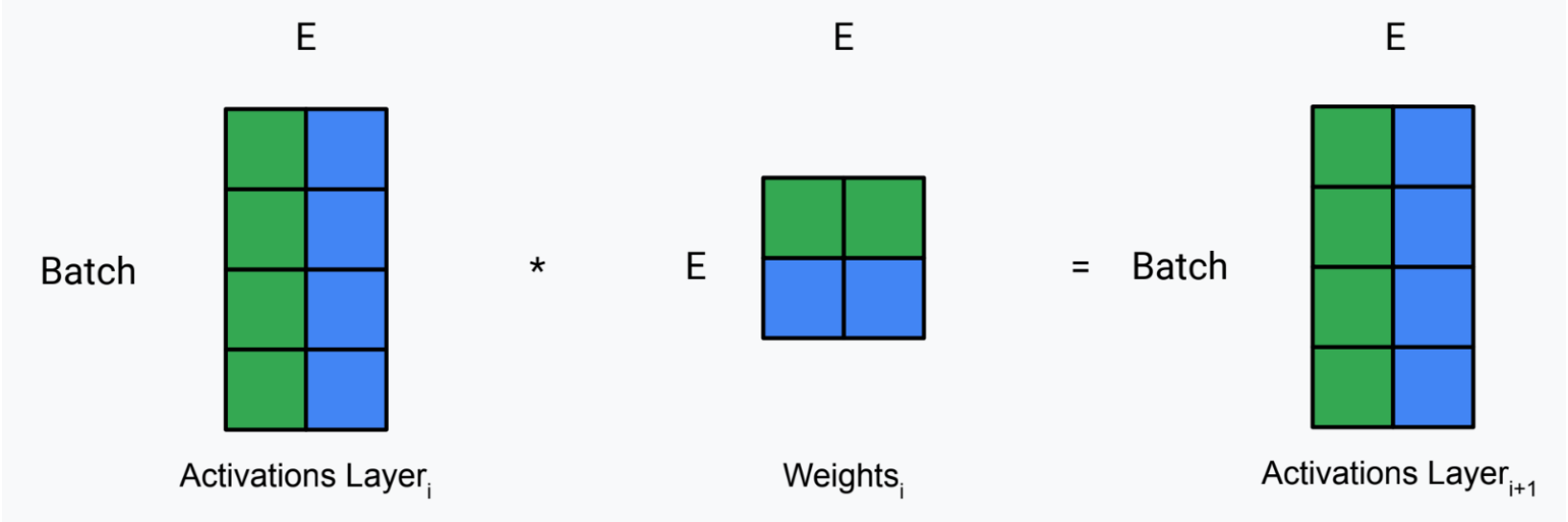


Fully sharded data parallelism

- The matrix multiply takes: FLOPs = 2 * BATCH_PER_CHIP * E2
- The all gather requires reading ~2 * E2 bytes (assuming bfloat16)
- Arithmetic intensity (FLOPs/bytes) = 2 * BATCH_PER_CHIP * E2 / (2 * E2 bytes) = BATCH_PER_CHIP FLOPs / byte.
- On TPU, ICI arithmetic intensity is 1018 FLOPs/byte, so we need our BATCH_PER_CHIP to be comfortably larger than 1018.

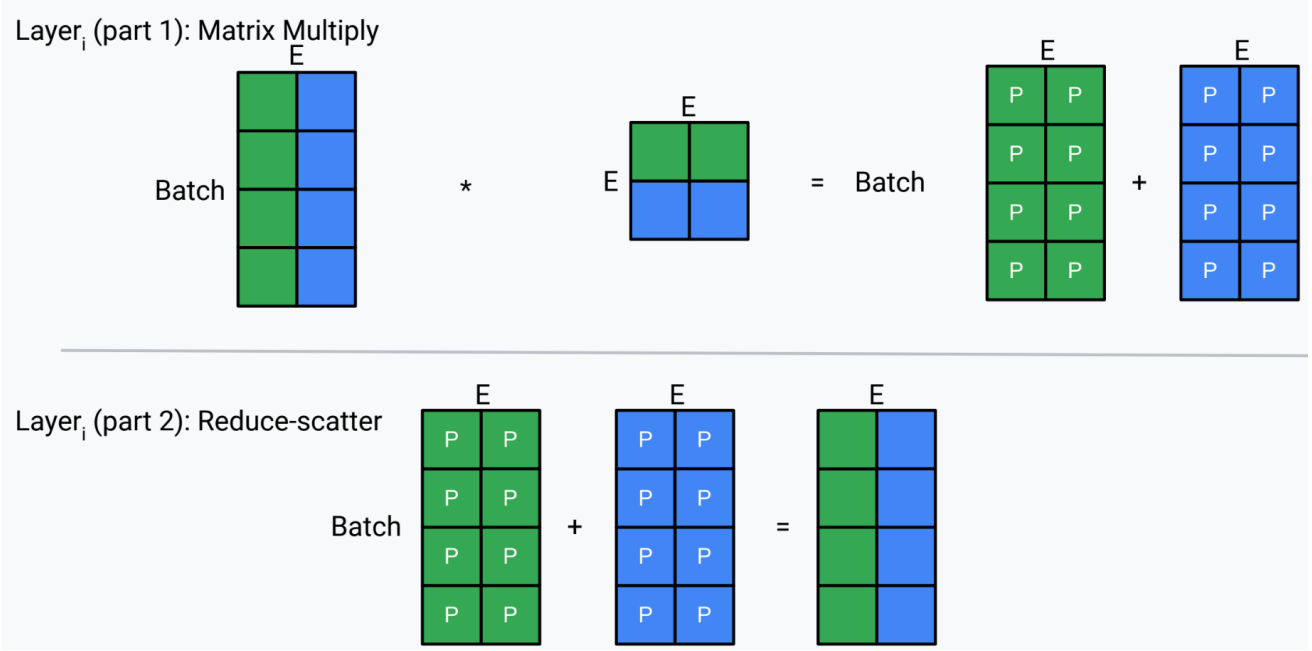
Tensor parallelism

- Shard the activation dimension



Tensor parallelism

- Shard the activation dimension

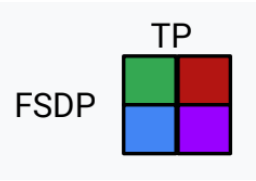


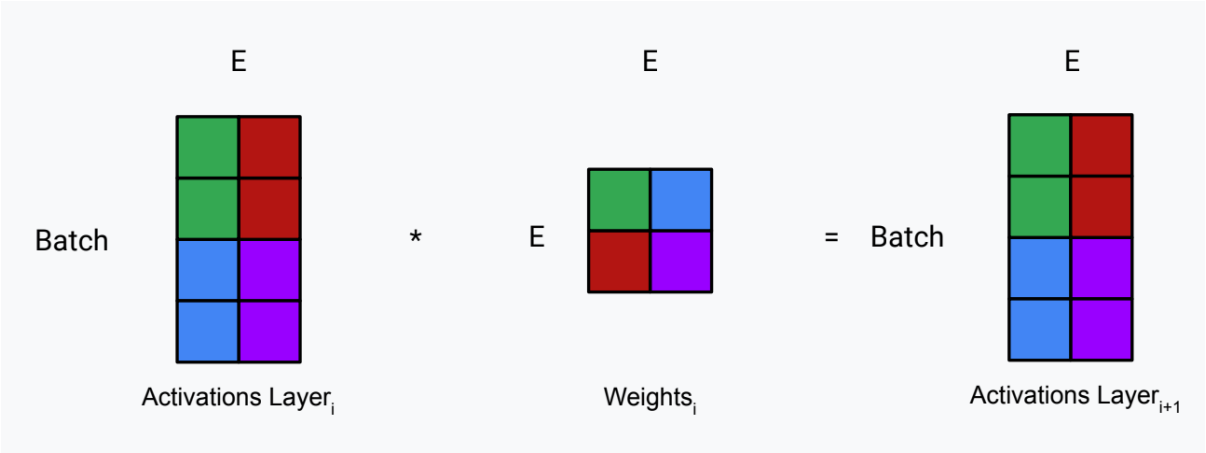
Tensor parallelism

- The matrix multiply takes (per chip): FLOPs = 2 * BATCH * E2 / (NUM_TP_SHARDS)
- The reduce scatter requires ~2 * BATCH * E bytes (assuming bfloat16)
- Arithmetic intensity (FLOPs/bytes) = E / NUM_TP_SHARDS.
- TPU ICI arithmetic intensity is 1018 FLOPs/byte, so we need E to be comfortably larger 1018 * NUM_TP_SHARDS.

FSDP with TP

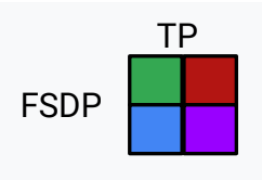
- While processing layer i, all-gather layer weights for layeri+1 for FSDP
- Then it just becomes TP

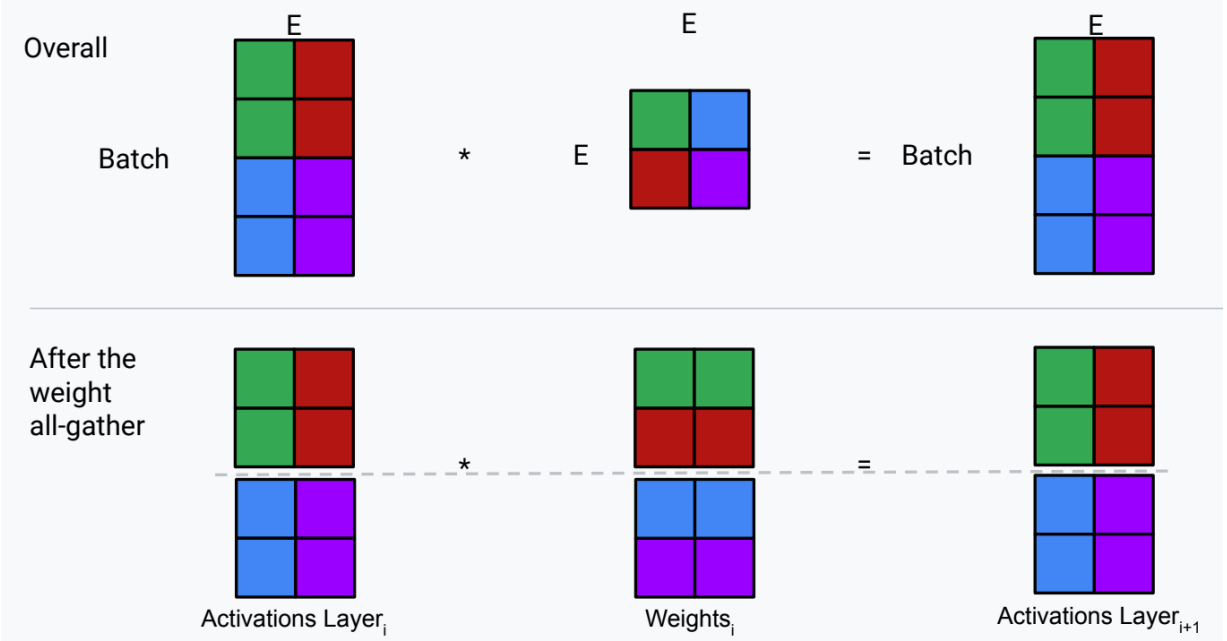




FSDP with TP

- While processing layer i, all-gather layer weights for layeri+1 for FSDP
- Then it just becomes TP





FSDP with TP

- Given NUM_TP_SHARDS and NUM_FSDP_SHARD
- When both:
  - E > 2 * 1018 * NUM_TP_SHARDS
  - NUM_TP_SHARD * BATCH_PER_CHIP > 2 * 1018
- 256 TPU: 16 for TP and 16 for FSDP
  - E > ~32k
  - BATCH_PER_CHIP > 125

Practical guideline

- Scaling model size and data size
  - Use FSDP as much as possible
  - Use TP/DP/SP to reduce batch size
- Scaling context size
  - Use Blockwise Transformers to reduce memory cost
  - Use RingAttention to increase context arbitrarily

Open problems

- Large language model → large world model
  - Predict world realistically, including text, vision, action.
  - Video conversation
- Consistent reasoning
  - With million-length “tape” but models stop step-by-step thinking after one hundred steps.
- Training and inference efficiency
  - Exponentially more efficient architecture or training paradigm