Scaling Laws for LLMs

NLP: Fall 2023

Anoop Sarkar

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

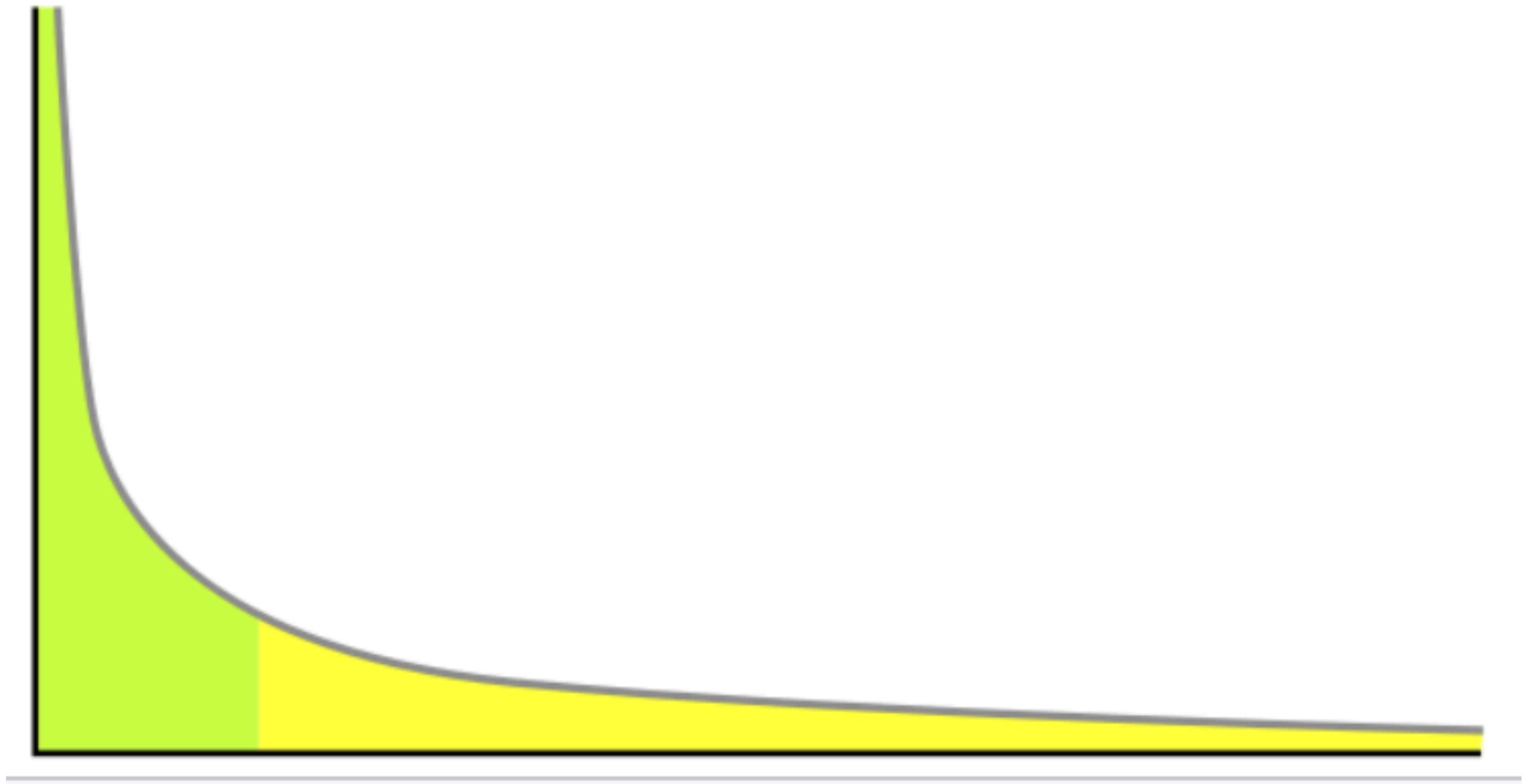
Dario Amodei
OpenAI
damodei@openai.com

https://arxiv.org/abs/2001.08361

Jan 2020

Scaling Laws for LLMs

Power laws



- A power law is a relation between two quantities: $f(x) = (a/x)^k$ e.g. model performance vs. model size.
- Number of model parameters N (excluding subword embeddings)
- Size of dataset D
- Amount of compute (MFLOPs) C
- N, D, C are dominant. Other choices in hyperparameters like width vs. depth are less relevant
- 1 PetaFLOP-day (PF-day) is 8.64×10^{19} FLOPS

https://openai.com/research/ai-and-compute

Operation	**Parameters**	**FLOPs per Token**
Embed	$(n_{\mathrm{vocab}} + n_{\mathrm{ctx}})\, d_{\mathrm{model}}$	$4d_{\mathrm{model}}$
Attention: QKV	$n_{\mathrm{layer}} d_{\mathrm{model}} 3 d_{\mathrm{attn}}$	$2 n_{\mathrm{layer}} d_{\mathrm{model}} 3 d_{\mathrm{attn}}$
Attention: Mask	—	$2 n_{\mathrm{layer}} n_{\mathrm{ctx}} d_{\mathrm{attn}}$
Attention: Project	$n_{\mathrm{layer}} d_{\mathrm{attn}} d_{\mathrm{model}}$	$2 n_{\mathrm{layer}} d_{\mathrm{attn}} d_{\mathrm{embd}}$
Feedforward	$n_{\mathrm{layer}} 2 d_{\mathrm{model}} d_{\mathrm{ff}}$	$2 n_{\mathrm{layer}} 2 d_{\mathrm{model}} d_{\mathrm{ff}}$
De-embed	—	$2 d_{\mathrm{model}} n_{\mathrm{vocab}}$
Total (Non-Embedding)	$N = 2 d_{\mathrm{model}} n_{\mathrm{layer}} \left(2 d_{\mathrm{attn}} + d_{\mathrm{ff}}\right)$	$C_{\mathrm{forward}} = 2N + 2 n_{\mathrm{layer}} n_{\mathrm{ctx}} d_{\mathrm{attn}}$

Table 1 Parameter counts and compute (forward pass) estimates for a Transformer model. Sub-leading terms such as nonlinearities, biases, and layer normalization are omitted.

Transformers asymptotically outperform LSTMs due to improved use of long contexts
Test Loss
5.4
4.8
4.2
3.6
3.0
2.4
LSTMs
1 Layer
2 Layers
4 Layers
Transformers
10^5
10^6
10^7
10^8
10^9
Parameters (non-embedding)
LSTM plateaus after <100 tokens
Transformer improves through the whole context
Per-token Test Loss
6
4
5
3
2
Parameters:
400K
400K
2M
3M
200M
300M
10^1
10^2
10^3
Token Index in Context

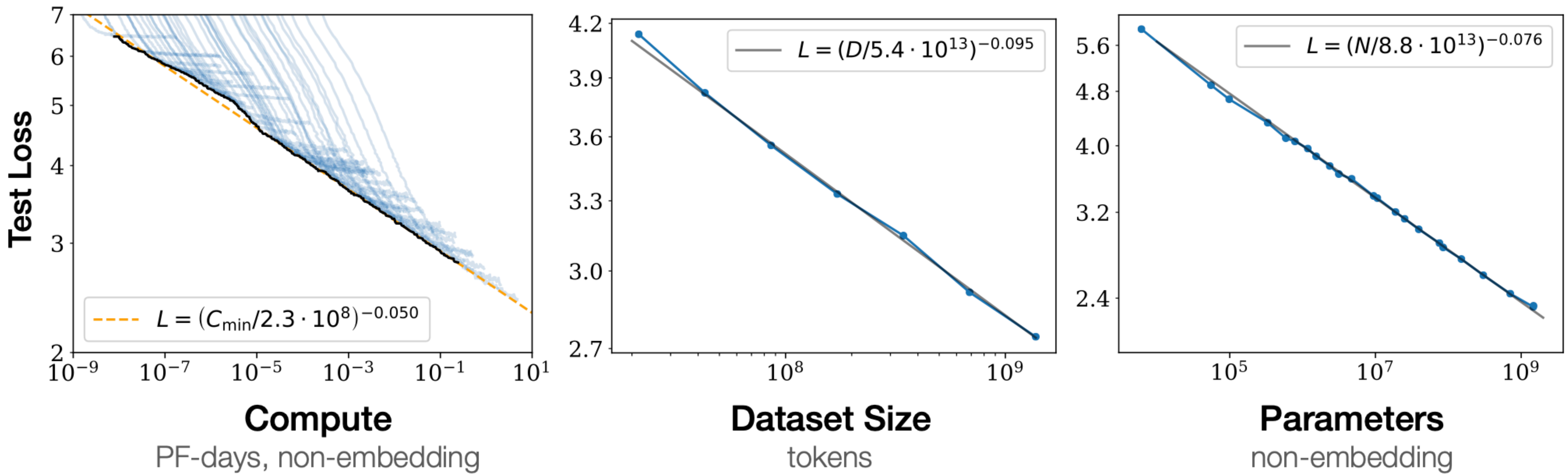


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

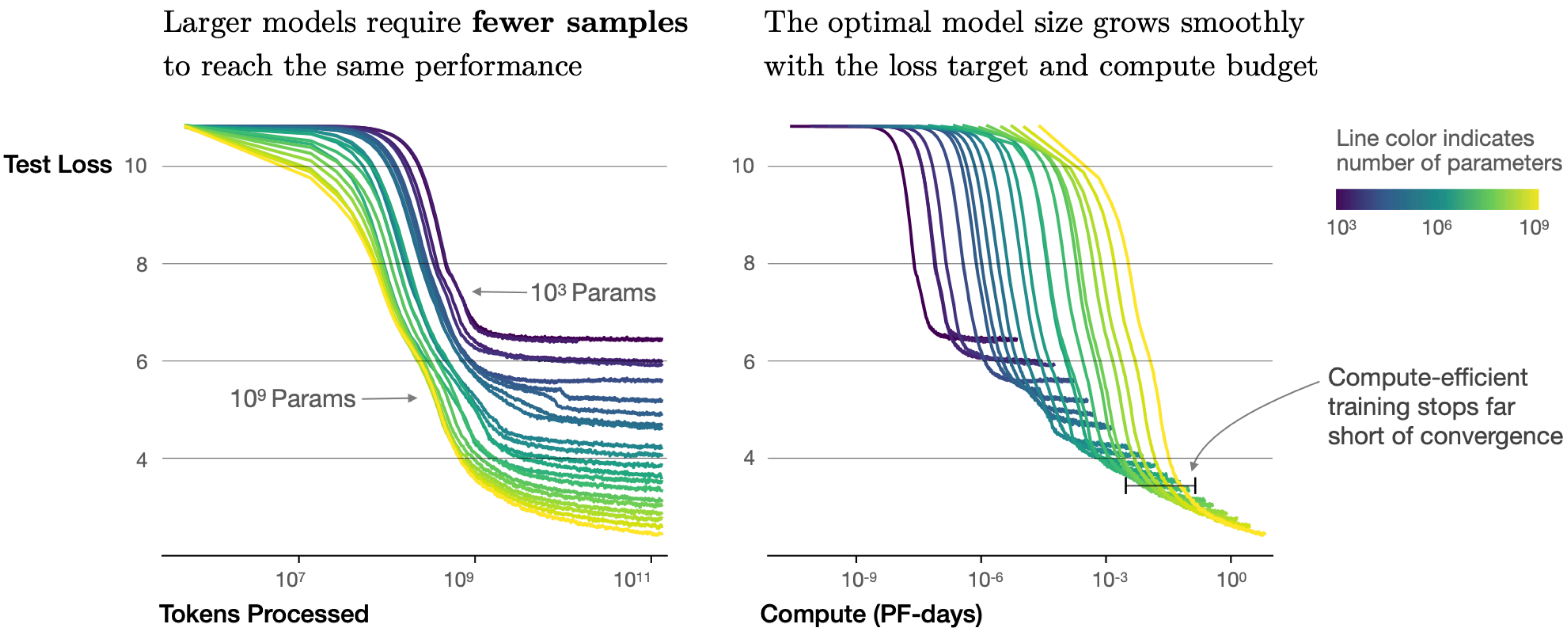


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

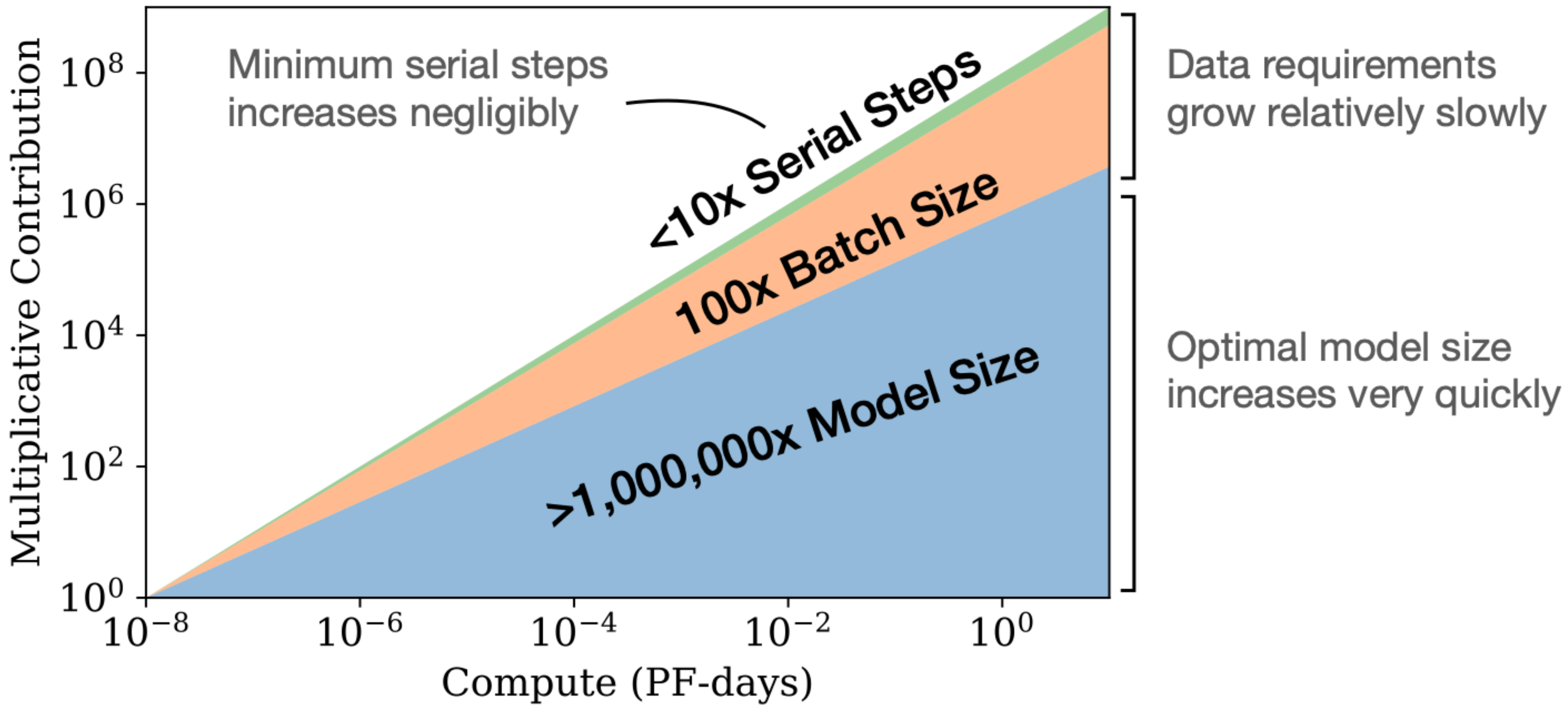
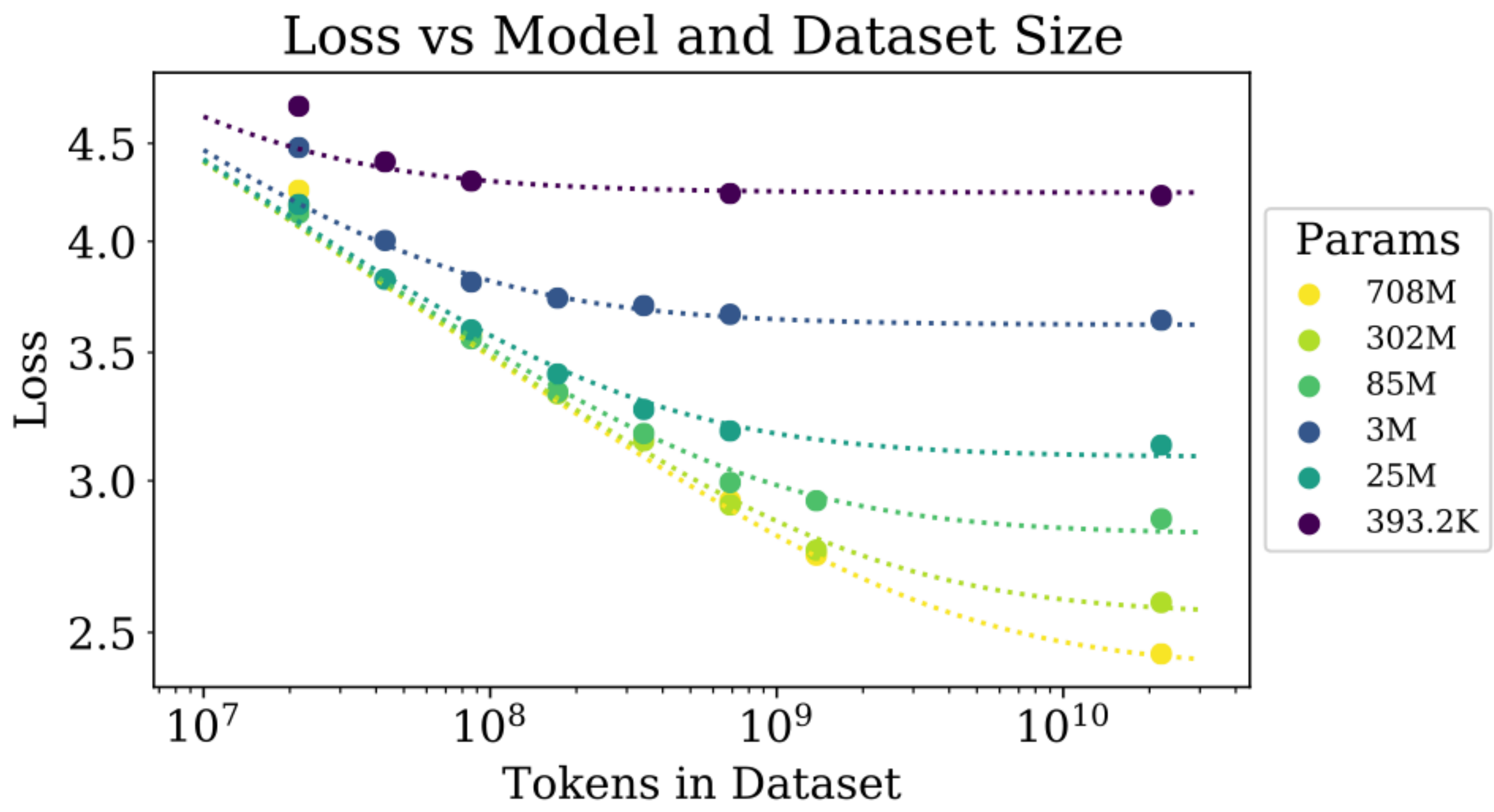
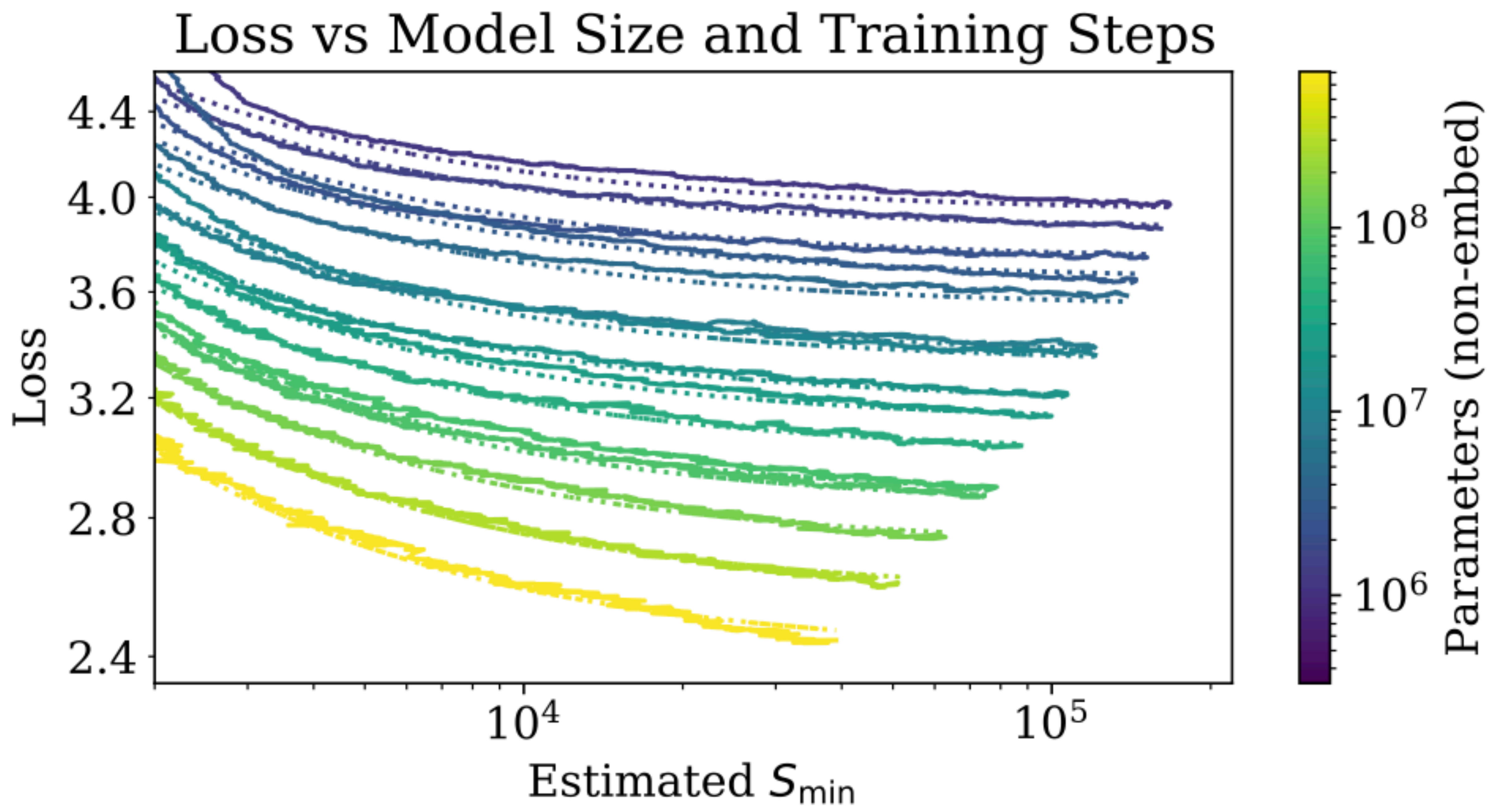


Figure 3 As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. We illustrate this for a billion-fold increase in compute. For optimally compute-efficient training, most of the increase should go towards increased model size. A relatively small increase in data is needed to avoid reuse. Of the increase in data, most can be used to increase parallelism through larger batch sizes, with only a very small increase in serial training time required.

Power laws for test loss

- Let $L(\cdot)$ represent the test loss dependent on either parameters N, or dataset size D or compute C
- For models with limited number of parameters: $L(N) = (N_c/N)^{\alpha_N}; \alpha_N \approx 0.076, N_c \approx 8.8 \times 10^{13}$(non-embd params)
- For models with limited dataset size: $L(D) = (D_c/D)^{\alpha_D}; \alpha_D \approx 0.095, D_c \approx 5.4 \times 10^{13}$(tokens)
- For models trained with limited compute: $L(C) = (C_c^{min}/C_{min})^{\alpha_C^{min}}; \alpha_c^{min} \approx 0.050, C_c^{min} \approx 3.1 \times 10^{8}$(PF-days)



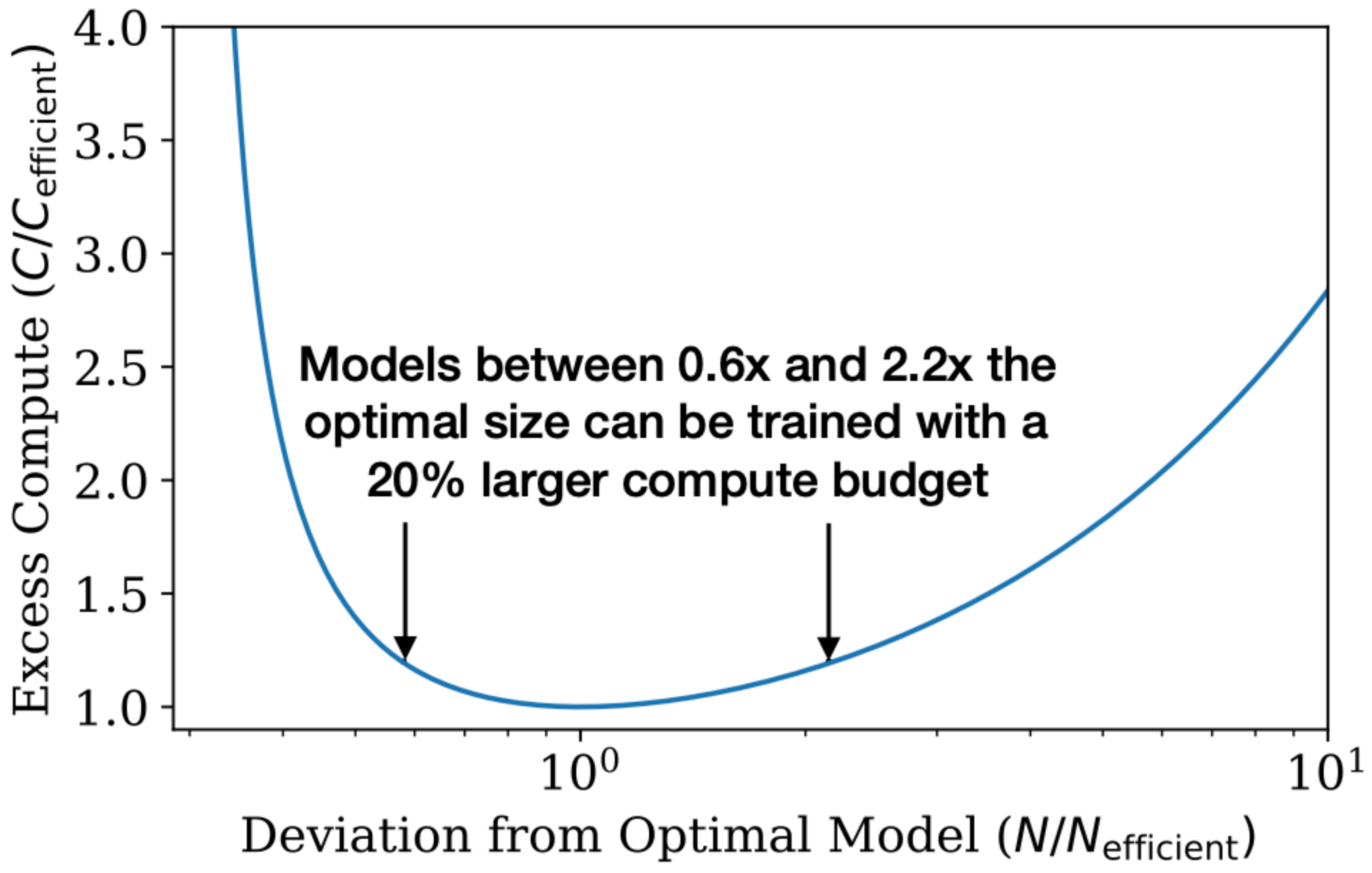


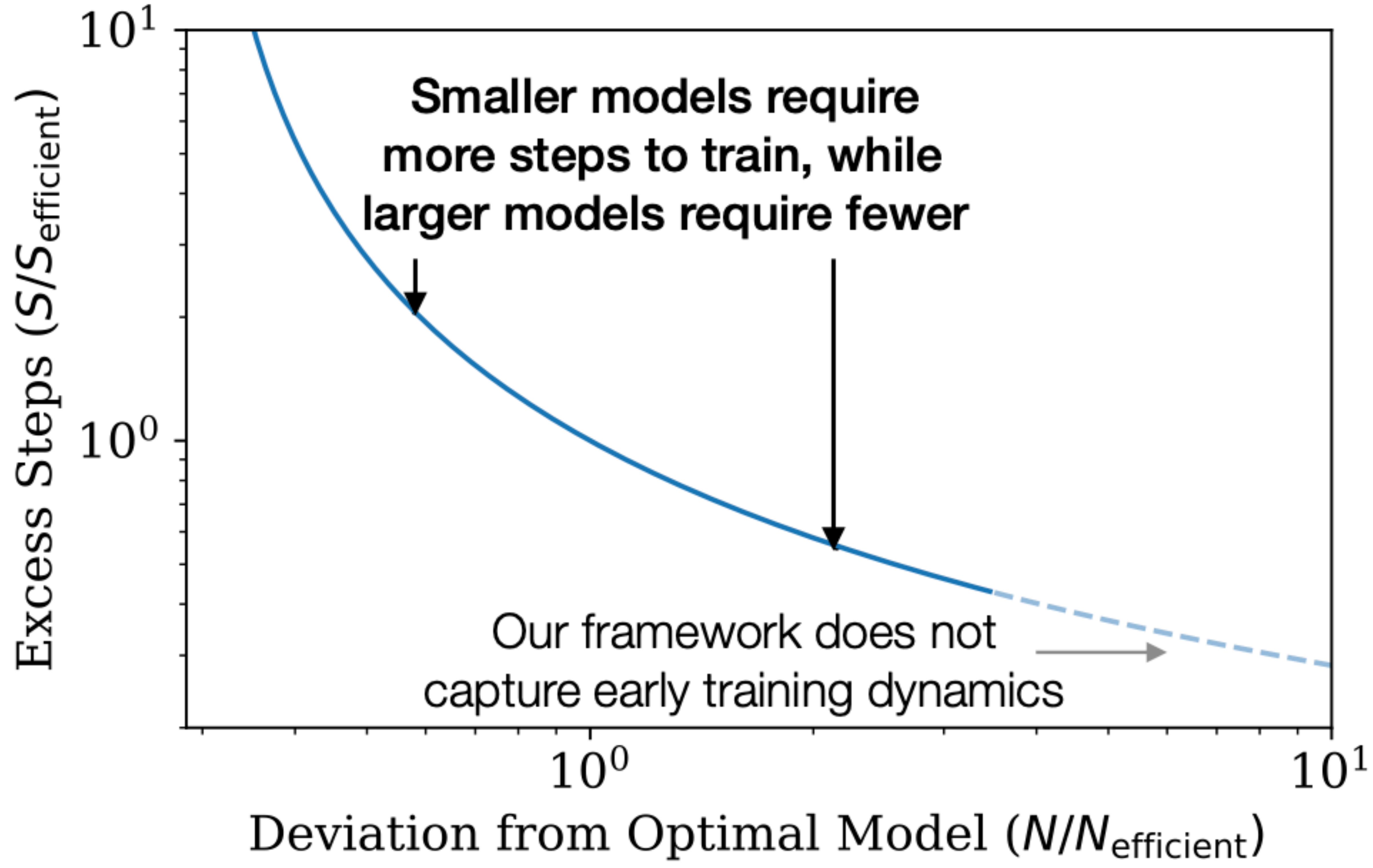
$$L(N,D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

$$L(N,S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\text{min}}(S)}\right)^{\alpha_S}$$

S = parameter update steps

Optimal Allocation of Compute Budget





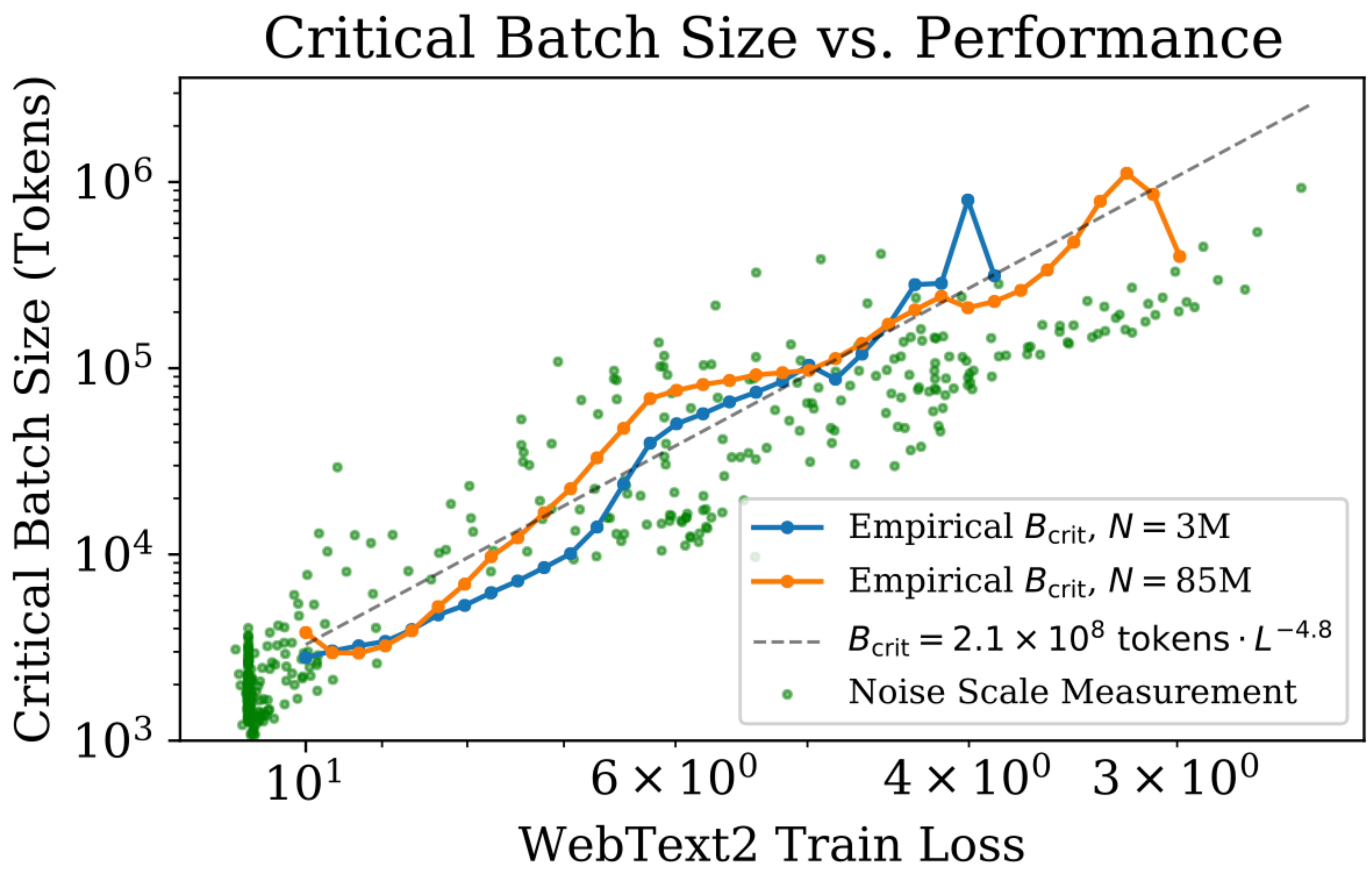


Figure 10 The critical batch size B_{crit} follows a power law in the loss as performance increase, and does not depend directly on the model size. We find that the critical batch size approximately doubles for every 13% decrease in loss. B_{crit} is measured empirically from the data shown in Figure 18, but it is also roughly predicted by the gradient noise scale, as in [MKAT18]. arXiv:1812.06162

Lessons from scaling LLMs

- Number of model parameters N
  Size of dataset D
- Amount of compute (MFLOPs) C

- Performance depends strongly on scale, weakly on model shape
- Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two
- Performance improves predictably as long as we scale up N and D in tandem
- Training curves follow predictable power-laws whose parameters are roughly independent of the model size

Lessons from scaling LLMs

- Transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set.
- Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps and using fewer data points
- The ideal batch size for training these models is roughly a power of the loss only, and continues to be determinable by measuring the gradient noise scale

Training Compute-Optimal Large Language Models

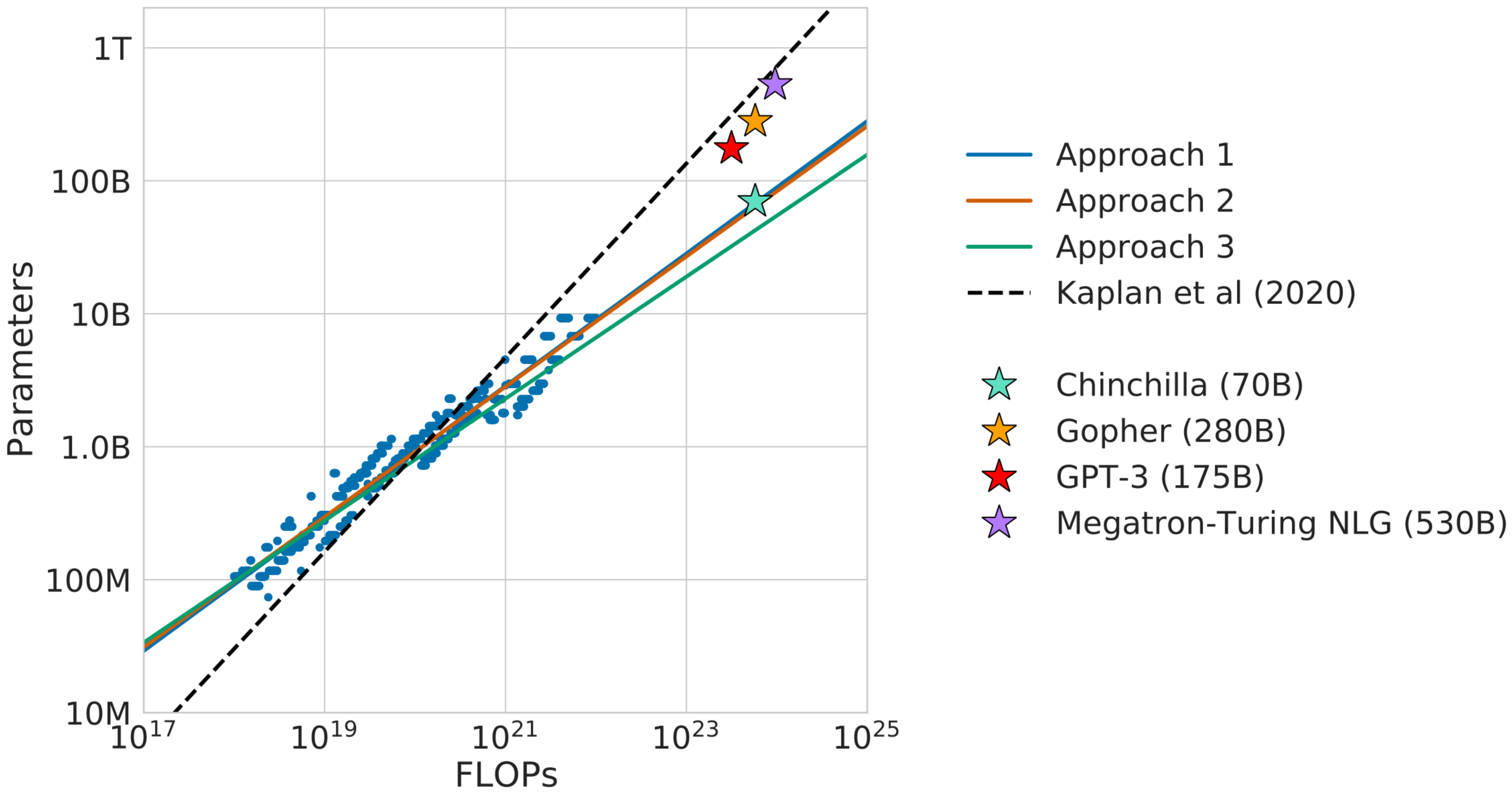
Jordan Hoffmann⋆, Sebastian Borgeaud⋆, Arthur Mensch⋆, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre⋆

https://arxiv.org/abs/2203.15556

Train longer on more tokens

Lessons from training Chinchilla

- From GPT3: large models should not be trained to lowest possible loss to be compute optimal
- Question: **Given a fixed FLOPs budget how should one trade off model size and number of training tokens?**
- Pre-training loss L(N, D) for N parameters and D training tokens. Find the optimal N and D values for a given compute budget.
- Empirical study on training 400 models from 70M to 16B parameters, trained on 5B to 400B tokens.
- Answer: **Train smaller models for (a lot) more training steps**.



1T
100B
10B
1.0B
100M
10M
Parameters
10^{17}
10^{19}
10^{21}
10^{23}
10^{25}
FLOPs
Approach 1
Approach 2
Approach 3
Kaplan et al (2020)
Chinchilla (70B)
Gopher (280B)
GPT-3 (175B)
Megatron-Turing NLG (530B)

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

The GPT3 paper

Language Models are Few-Shot Learners

Tom B. Brown* **Benjamin Mann*** **Nick Ryder*** **Melanie Subbiah***

Jared Kaplan† **Prafulla Dhariwal** **Arvind Neelakantan** **Pranav Shyam**

Girish Sastry **Amanda Askell** **Sandhini Agarwal** **Ariel Herbert-Voss**

Gretchen Krueger **Tom Henighan** **Rewon Child** **Aditya Ramesh**

Daniel M. Ziegler **Jeffrey Wu** **Clemens Winter**

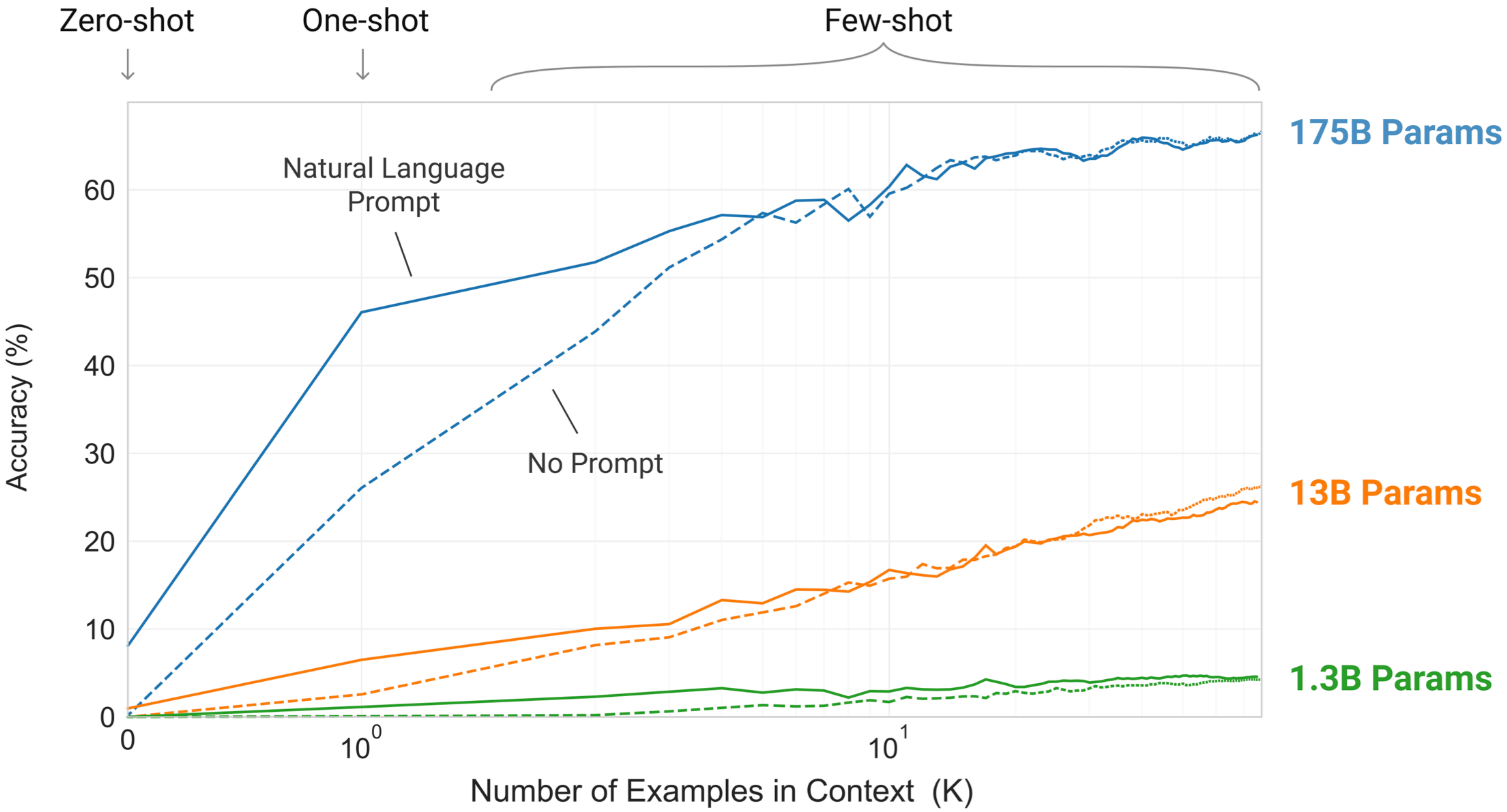
Christopher Hesse **Mark Chen** **Eric Sigler** **Mateusz Litwin** **Scott Gray**

Benjamin Chess **Jack Clark** **Christopher Berner**

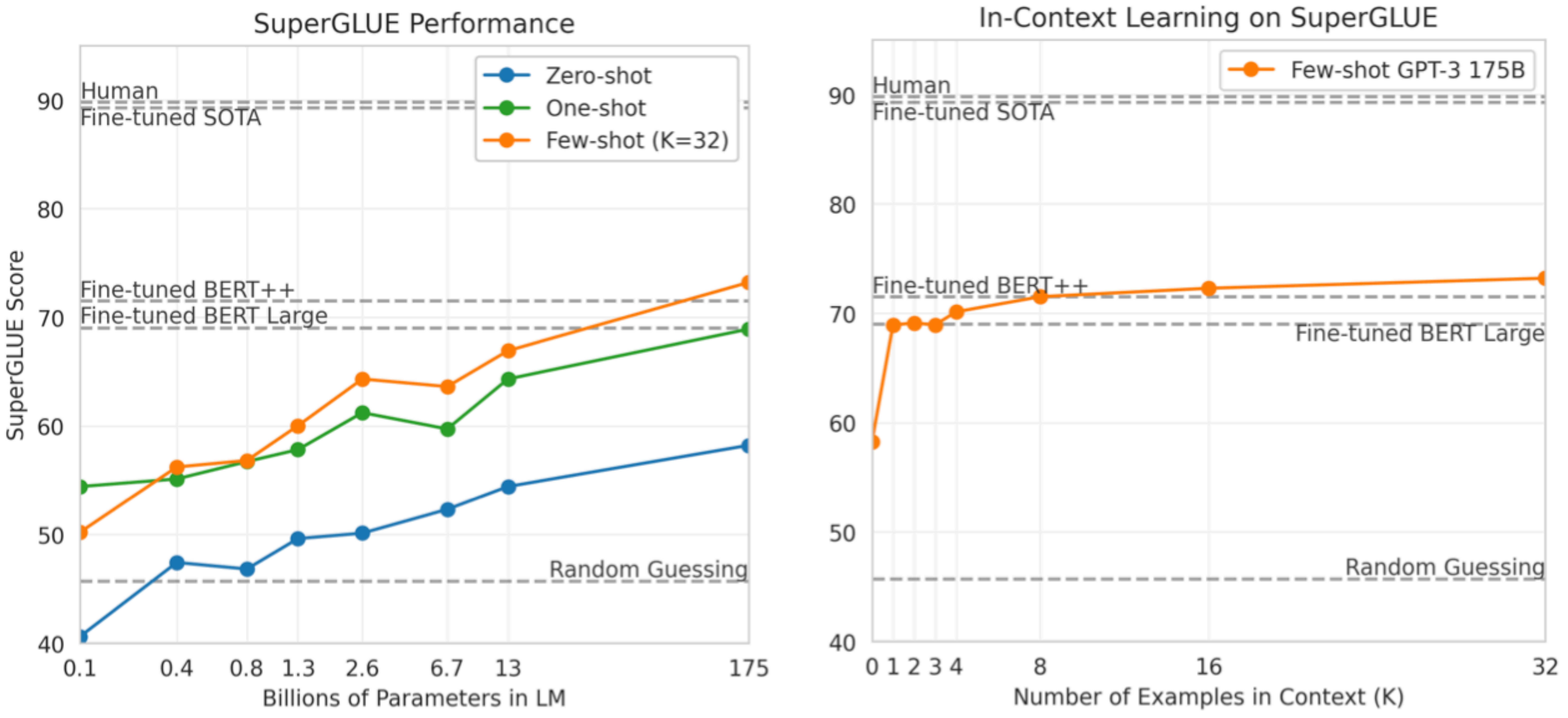
Sam McCandlish **Alec Radford** **Ilya Sutskever** **Dario Amodei**

https://arxiv.org/abs/2005.14165

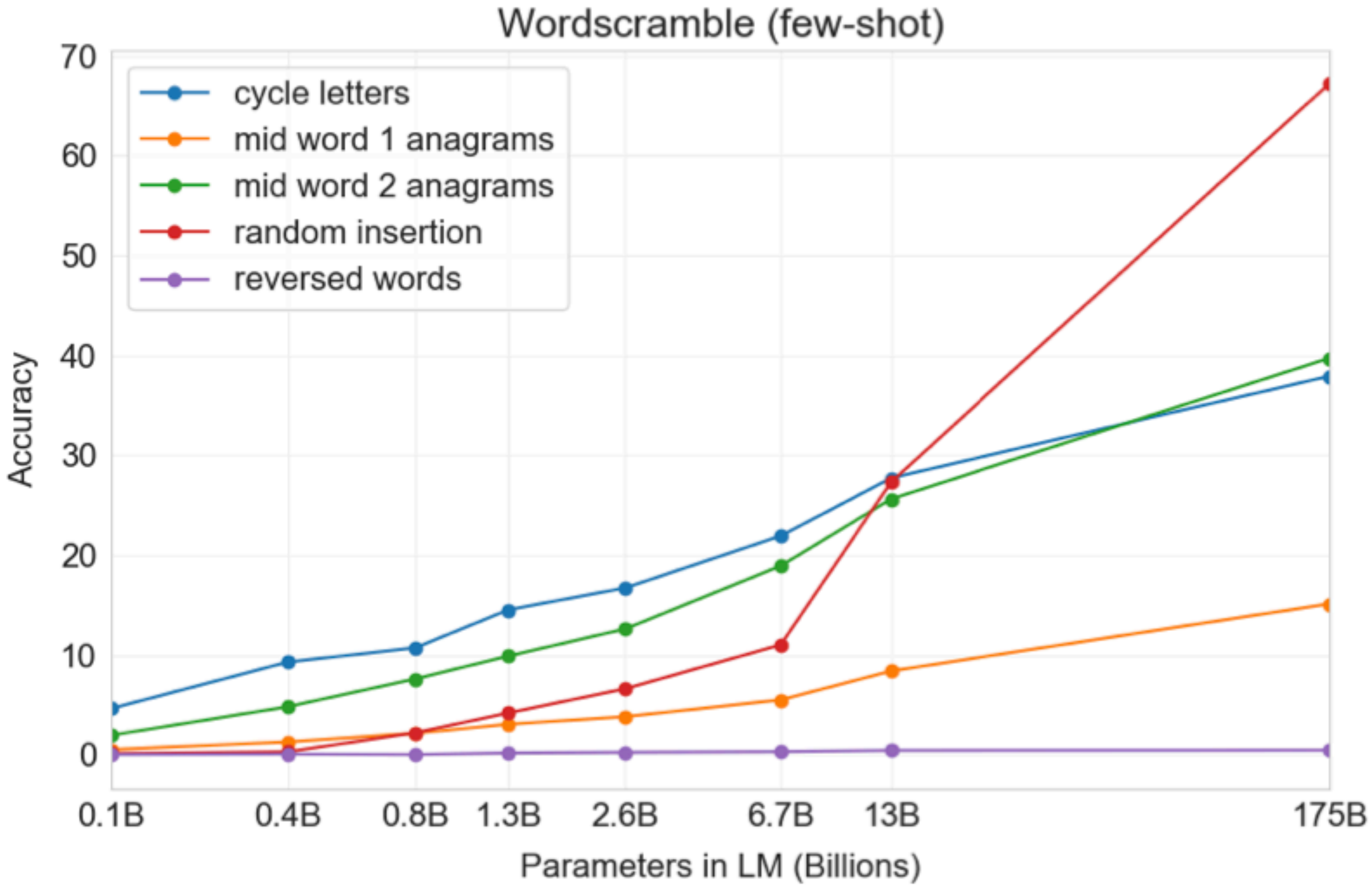
NeurIPS 2020, Vancouver, BC

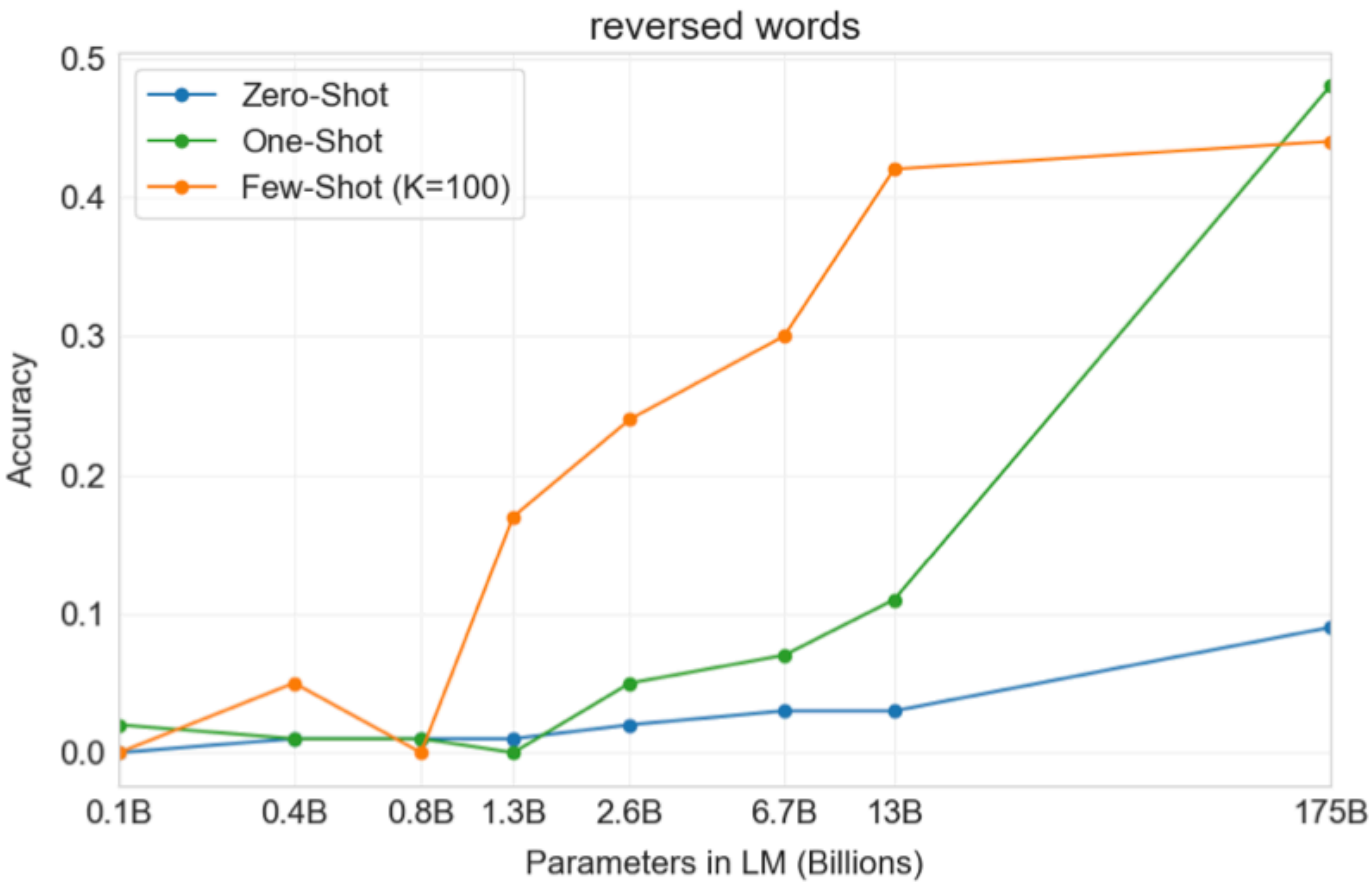


Zero-shot
One-shot
Few-shot
175B Params
13B Params
1.3B Params
Natural Language
Prompt
No Prompt
Accuracy (%)
60
50
40
30
20
10
0
0
10^0
10^1
Number of Examples in Context (K)



Performance on SuperGLUE increases with number of examples in context. We find the difference in performance between the BERT-Large and BERT++ to be roughly equivalent to the difference between GPT-3 with one example per context versus eight examples per context.


Wordscramble (few-shot)
cycle letters
mid word 1 anagrams
mid word 2 anagrams
random insertion
reversed words
Accuracy
0
10
20
30
40
50
60
70
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)



reversed words
Zero-Shot
One-Shot
Few-Shot (K=100)
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)


mid word 1 anagrams
Zero-Shot
One-Shot
Few-Shot (K=100)
Accuracy
0
2
4
6
8
10
12
14
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)
random insertion
Zero-Shot
One-Shot
Few-Shot (K=100)
Accuracy
0
10
20
30
40
50
60
70
0.1B
0.4B
0.8B
1.3B
2.6B
6.7B
13B
175B
Parameters in LM (Billions)

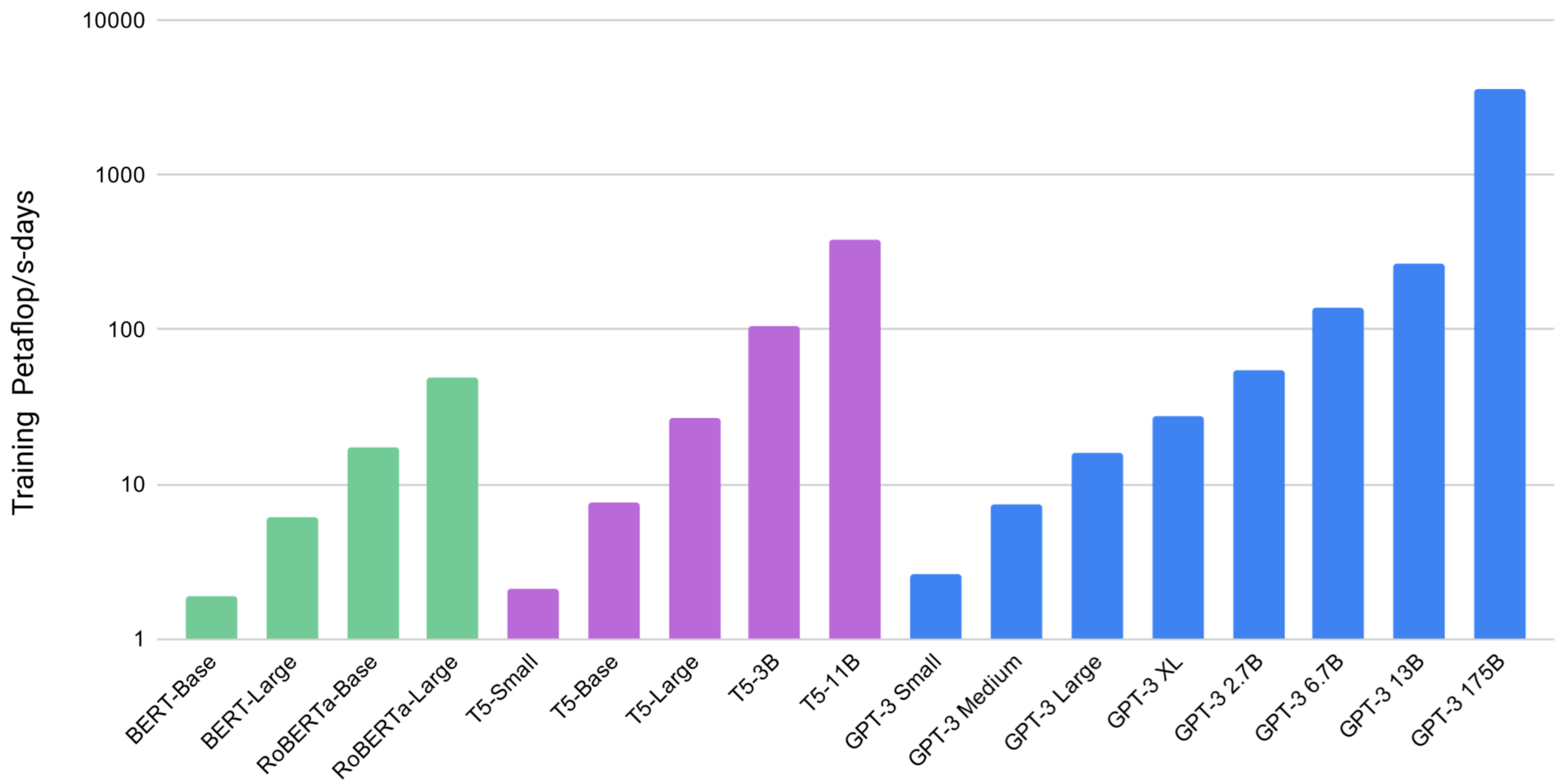


Figure 7.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in the Appendix.

GLaM: Efficient Scaling of Language Models with Mixture-of-Experts

Nan Du [*1] **Yanping Huang** [*1] **Andrew M. Dai** [*1] **Simon Tong** [1] **Dmitry Lepikhin** [1] **Yuanzhong Xu** [1]
Maxim Krikun [1] **Yanqi Zhou** [1] **Adams Wei Yu** [1] **Orhan Firat** [1] **Barret Zoph** [1] **Liam Fedus** [1] **Maarten Bosma** [1]
Zongwei Zhou [1] **Tao Wang** [1] **Yu Emma Wang** [1] **Kellie Webster** [1] **Marie Pellat** [1] **Kevin Robinson** [1]
Kathleen Meier-Hellstern [1] **Toju Duke** [1] **Lucas Dixon** [1] **Kun Zhang** [1] **Quoc V Le** [1] **Yonghui Wu** [1]
Zhifeng Chen [1] **Claire Cui** [1]

https://arxiv.org/abs/2112.06905

Mixture of Experts (MoE) for LLMs

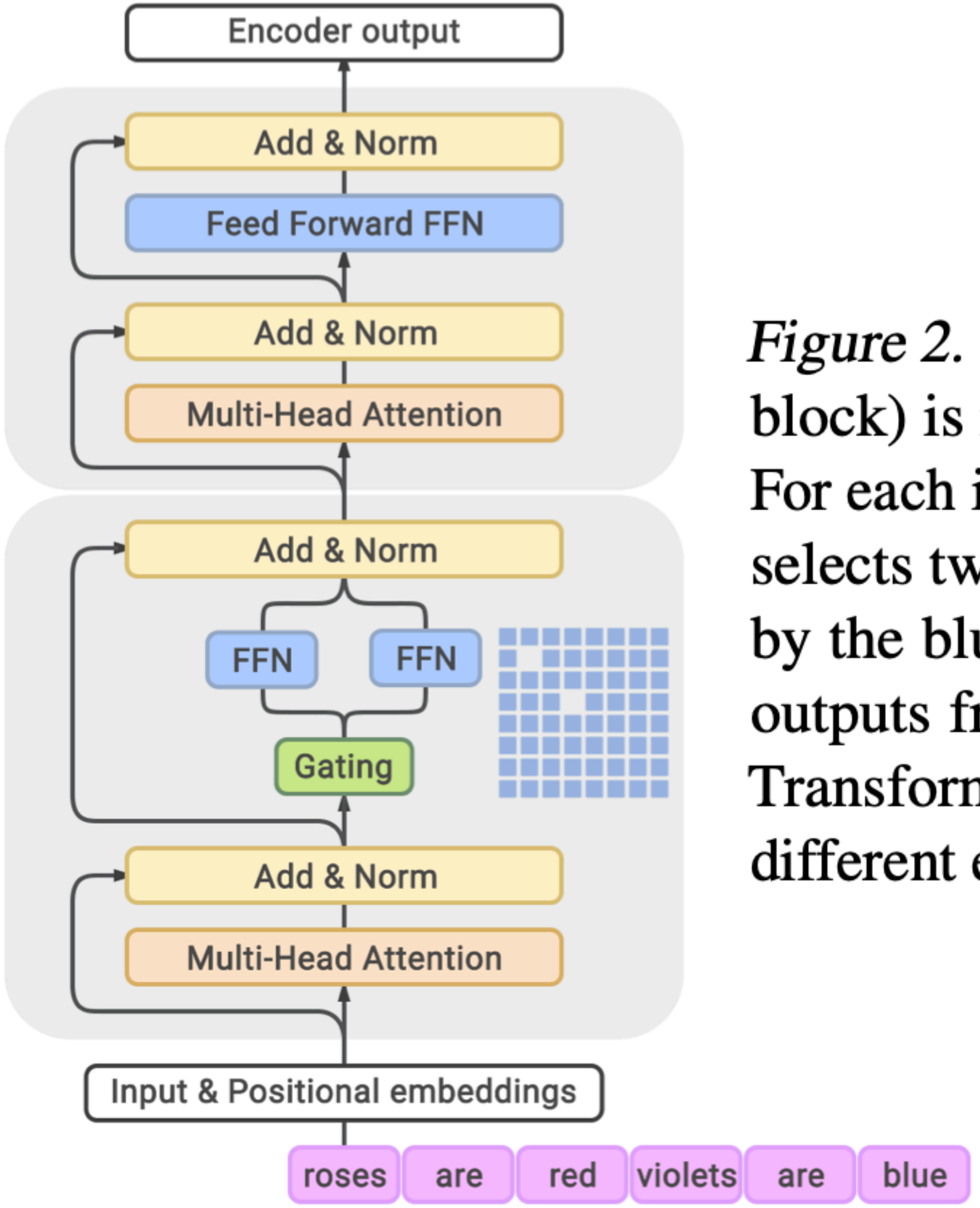
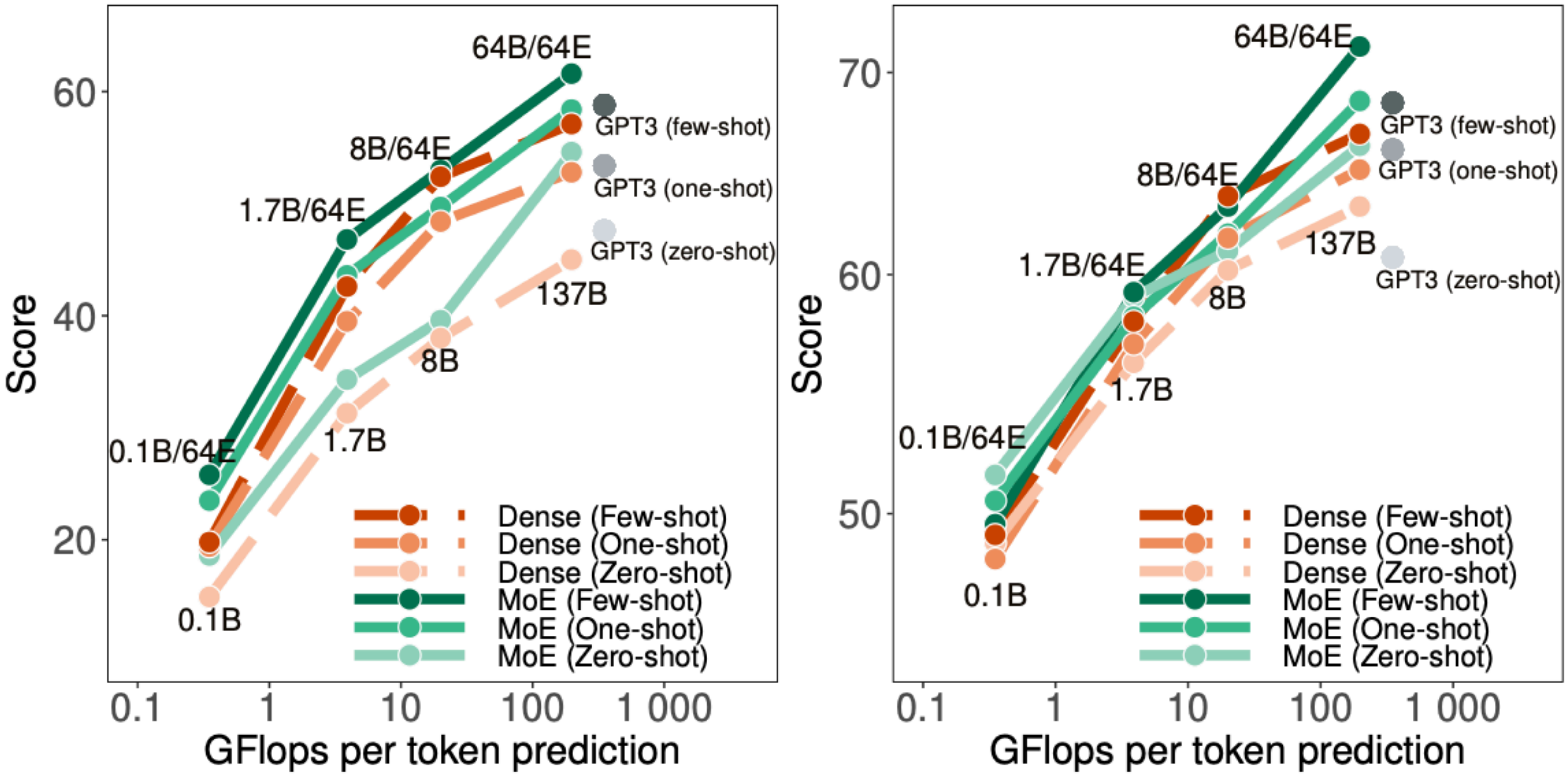


Figure 2. GLaM model architecture. Each MoE layer (the bottom block) is interleaved with a Transformer layer (the upper block). For each input token, *e.g.*, 'roses', the *Gating* module dynamically selects two most relevant experts out of 64, which is represented by the blue grid in the MoE layer. The weighted average of the outputs from these two experts will then be passed to the upper Transformer layer. For the next token in the input sequence, two different experts will be selected.

Mixture of Experts (MoE) for LLMs

Better effective FLOPs per token prediction in causal LMs



PaLM: Scaling Language Modeling with Pathways

Aakanksha Chowdhery[*] Sharan Narang[*] Jacob Devlin[*]
Maarten Bosma Gaurav Mishra Adam Roberts Paul Barham
Hyung Won Chung Charles Sutton Sebastian Gehrmann Parker Schuh Kensen Shi
Sasha Tsvyashchenko Joshua Maynez Abhishek Rao[†] Parker Barnes Yi Tay
Noam Shazeer[‡] Vinodkumar Prabhakaran Emily Reif Nan Du Ben Hutchinson
Reiner Pope James Bradbury Jacob Austin Michael Isard Guy Gur-Ari
Pengcheng Yin Toju Duke Anselm Levskaya Sanjay Ghemawat Sunipa Dev
Henryk Michalewski Xavier Garcia Vedant Misra Kevin Robinson Liam Fedus
Denny Zhou Daphne Ippolito David Luan[‡] Hyeontaek Lim Barret Zoph
Alexander Spiridonov Ryan Sepassi David Dohan Shivani Agrawal Mark Omernick
Andrew M. Dai Thanumalayan Sankaranarayana Pillai Marie Pellat Aitor Lewkowycz
Erica Moreira Rewon Child Oleksandr Polozov[†] Katherine Lee Zongwei Zhou
Xuezhi Wang Brennan Saeta Mark Diaz Orhan Firat Michele Catasta[†] Jason Wei
Kathy Meier-Hellstern Douglas Eck Jeff Dean Slav Petrov Noah Fiedel

Google Research

https://arxiv.org/abs/2204.02311

PaLM: model architecture

- **SwiGLU Activation** – We use SwiGLU activations $(\text{Swish}(xW) \cdot xV)$ for the MLP intermediate activations because they have been shown to significantly increase quality compared to standard ReLU, GeLU, or Swish activations (Shazeer, 2020). Note that this does require three matrix multiplications in the MLP rather than two, but Shazeer (2020) demonstrated an improvement in quality in compute-equivalent experiments (i.e., where the standard ReLU variant had proportionally larger dimensions).

- **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: model architecture

- **Multi-Query Attention** – The standard Transformer formulation uses k attention heads, where the input vector for each timestep is linearly projected into "query", "key", and "value" tensors of shape $[k, h]$, where h is the attention head size. Here, the key/value projections are shared for each head, i.e. "key" and "value" are projected to $[1, h]$, but "query" is still projected to shape $[k, h]$. We have found that this has a neutral effect on model quality and training speed (Shazeer, 2019), but results in a significant cost savings at autoregressive decoding time. This is because standard multi-headed attention has low efficiency on accelerator hardware during auto-regressive decoding, because the key/value tensors are not shared between examples, and only a single token is decoded at a time.

- **RoPE Embeddings** – We use RoPE embeddings (Su et al., 2021) rather than absolute or relative position embeddings, since RoPE embeddings have been shown to have better performance on long sequence lengths.

- **Shared Input-Output Embeddings** – We share the input and output embedding matrices, which is done frequently (but not universally) in past work.

PaLM: model architecture

- **No Biases** – No biases were used in any of the dense kernels or layer norms. We found this to result in increased training stability for large models.

- **Vocabulary** – We use a SentencePiece (Kudo & Richardson, 2018a) vocabulary with 256k tokens, which was chosen to support the large number of languages in the training corpus without excess tokenization. The vocabulary was generated from the training data, which we found improves training efficiency. The vocabulary is completely lossless and reversible, which means that whitespace is completely preserved in the vocabulary (especially important for code) and out-of-vocabulary Unicode characters are split into UTF-8 bytes, with a vocabulary token for each byte. Numbers are always split into individual digit tokens (e.g., "123.5 $\rightarrow$ 1 2 3 . 5").

PaLM: model hyperparameters

Model	Layers	# of Heads	d_{model}	# of Parameters (in billions)	Batch Size
PaLM 8B	32	16	4096	8.63	$256 \rightarrow 512$
PaLM 62B	64	32	8192	62.50	$512 \rightarrow 1024$
PaLM 540B	118	48	18432	540.35	$512 \rightarrow 1024 \rightarrow 2048$

Table 1: Model architecture details. We list the number of layers, d_{model}, the number of attention heads and attention head size. The feed-forward size d_{ff} is always $4 \times d_{\text{model}}$ and attention head size is always 256.

PaLM: training data

Total dataset size = 780 billion tokens	
Data source	Proportion of data
Social media conversations (multilingual)	50%
Filtered webpages (multilingual)	27%
Books (English)	13%
GitHub (code)	5%
Wikipedia (multilingual)	4%
News (English)	1%

Table 2: Proportion of data from each source in the training dataset. The multilingual corpus contains text from over 100 languages, with the distribution given in Appendix Table 29.

PaLM: Pathways data parallelism

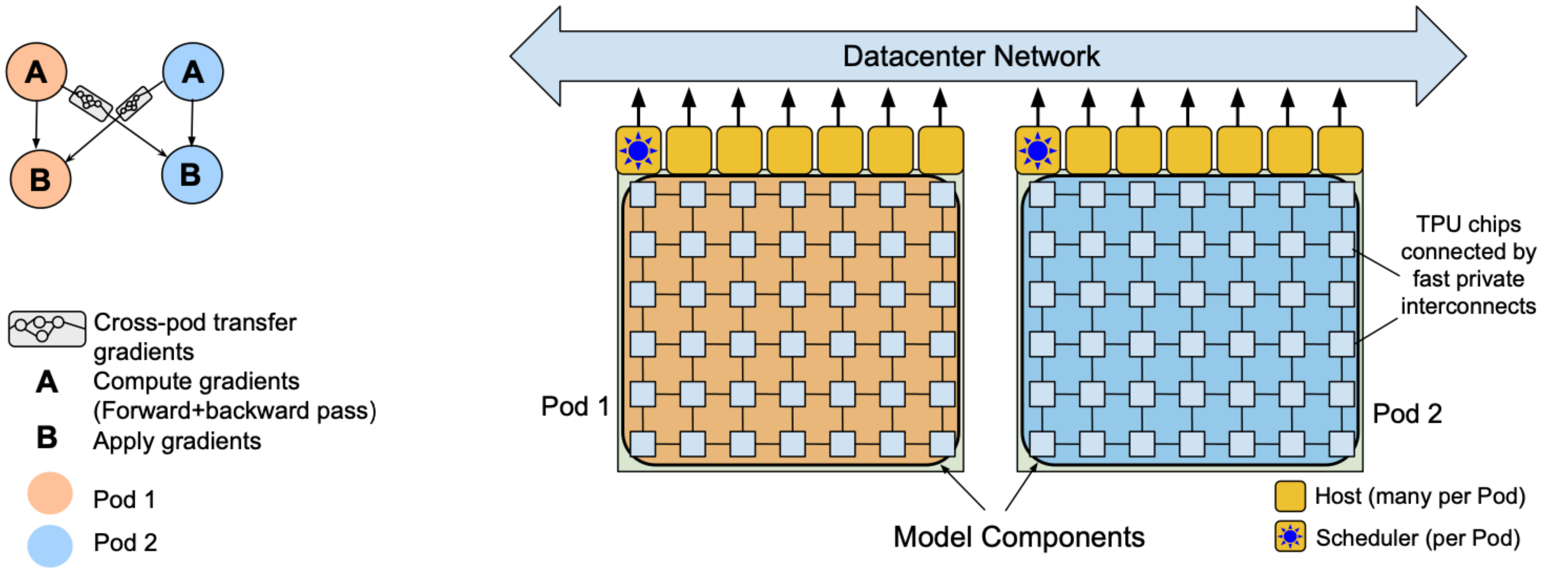


Figure 2: The Pathways system (Barham et al., 2022) scales training across two TPU v4 pods using two-way data parallelism at the pod level.