SFU NatLangLab

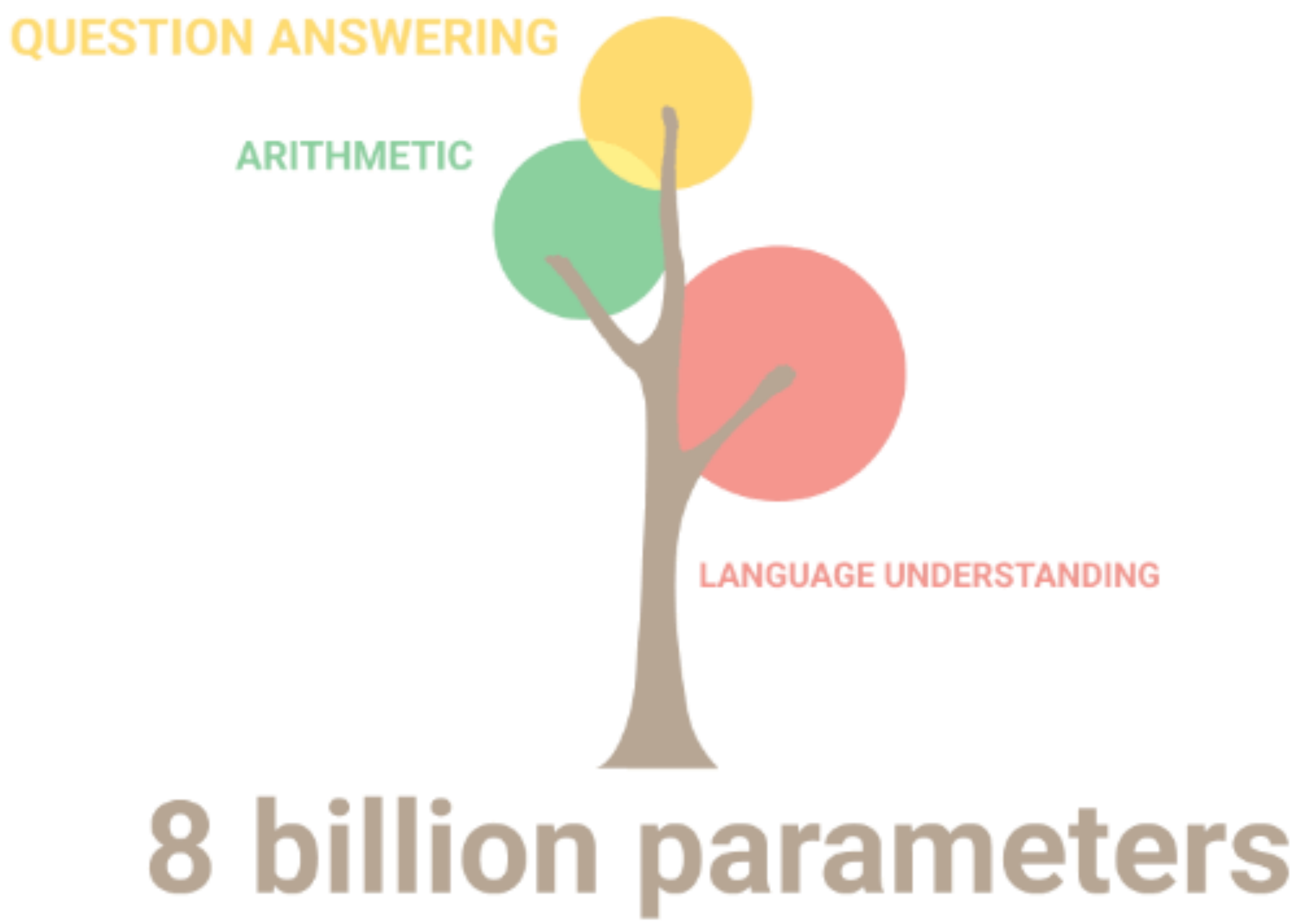
CMPT 413/713: Natural Language Processing

Scaling laws for LLMs

Spring 2024
2024-03-20

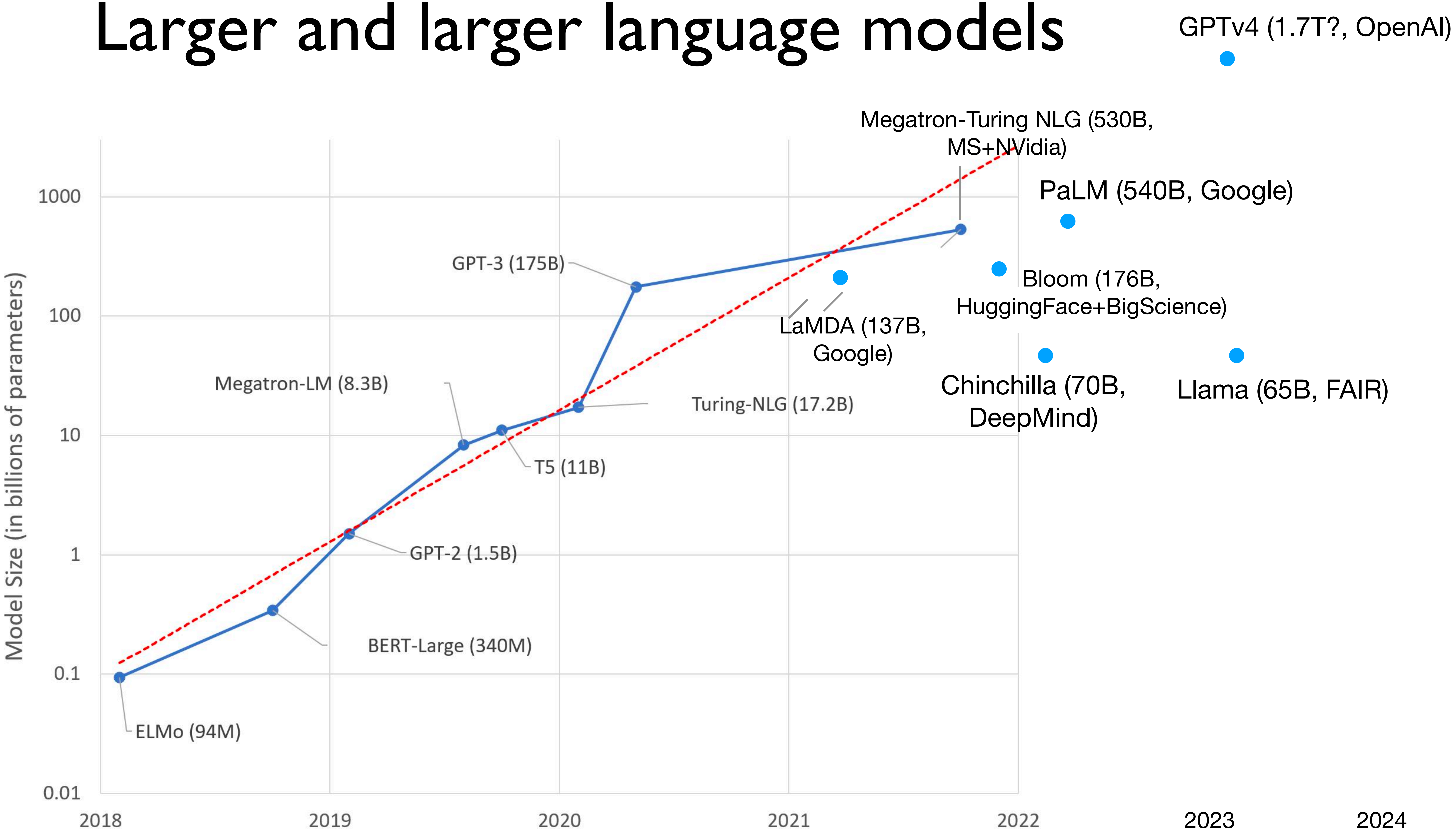
Slides adapted from Anoop Sarkar

New capabilities emerge at scale

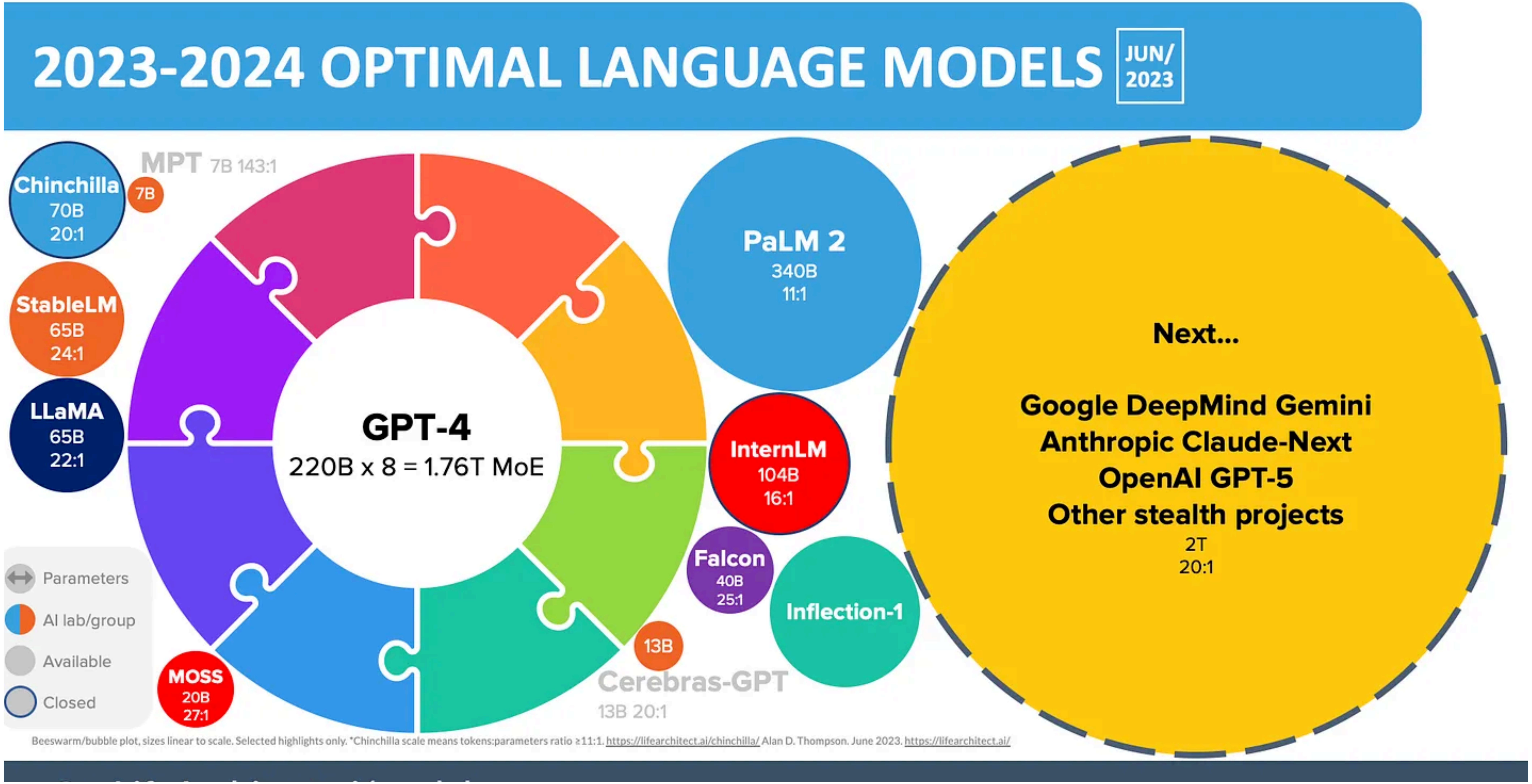


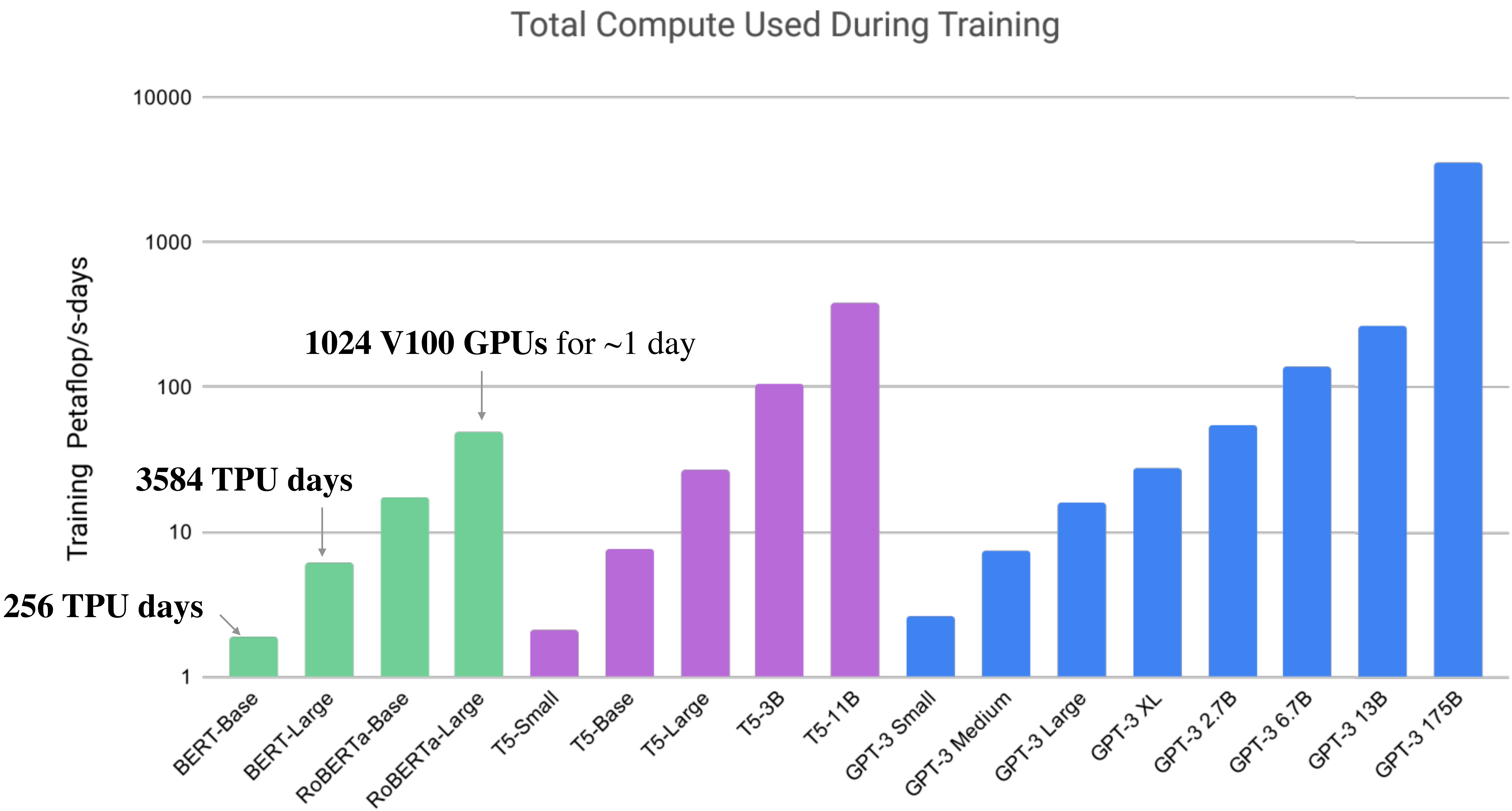
https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

Larger and larger language models



https://huggingface.co/blog/large-language-models


2023-2024 OPTIMAL LANGUAGE MODELS
JUN/ 2023
Chinchilla 70B 20:1
MPT 7B 143:1
7B
StableLM 65B 24:1
LLaMA 65B 22:1
GPT-4
220B x 8 = 1.76T MoE
PaLM 2 340B 11:1
InternLM 104B 16:1
Falcon 40B 25:1
Inflection-1
13B
Cerebras-GPT 13B 20:1
MOSS 20B 27:1
Next...
Google DeepMind Gemini
Anthropic Claude-Next
OpenAI GPT-5
Other stealth projects
2T
20:1
Parameters
AI lab/group
Available
Closed
Beeswarm/bubble plot, sizes linear to scale. Selected highlights only. *Chinchilla scale means tokens:parameters ratio ≥11:1. https://lifearchitect.ai/chinchilla/ Alan D. Thompson. June 2023. https://lifearchitect.ai/



Language Models are Few-Shot Learners (Brown et al. 2020)

How does LLM performance scale as we increase model and data size?

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

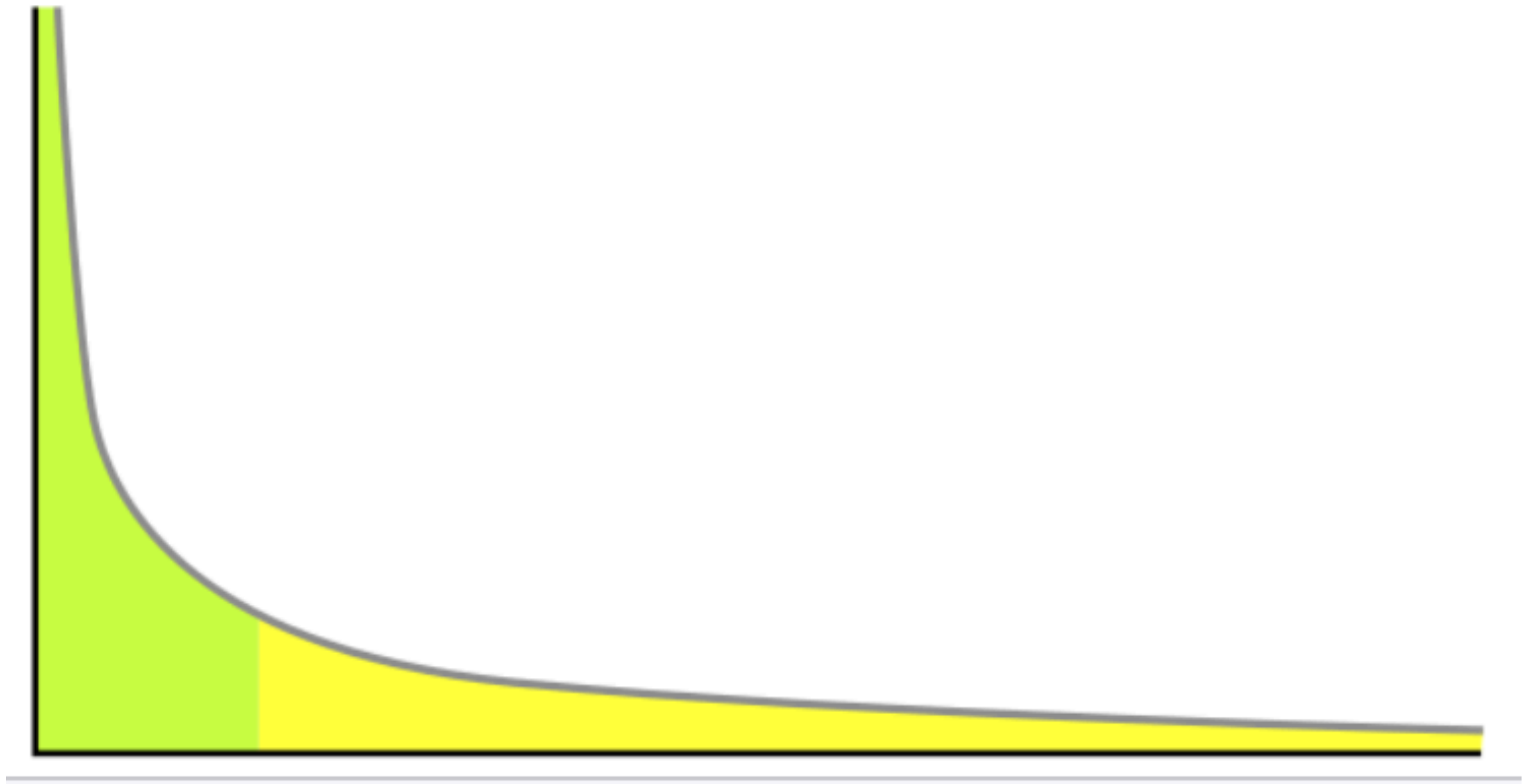
Dario Amodei
OpenAI
damodei@openai.com

https://arxiv.org/abs/2001.08361

Jan 2020

Scaling Laws for LLMs

Power laws



For LLMs, we are interested in how the test performance scales with relation to

- Model size: number of model parameters **N** (excluding subword embeddings)
- Data size: number of tokens trained on **D**
- Amount of compute (MFLOPs) **C** (1 PetaFLOP-day (PF-day) is 8.64×10^{19} FLOPS)

Findings

- Model performance scales as **power law** of model size and data size
- Power law: relation between two quantities where one quantity increases as a power of another
  - $f(x) = (a/x)^k$ e.g. model performance vs. model size
- N, D, C are dominant. Other choices in hyperparameters like width vs. depth are less relevant

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

Model size: computing the number of parameters

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

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.

Test performance

- Power law relationship to compute, dataset size, and number of parameters

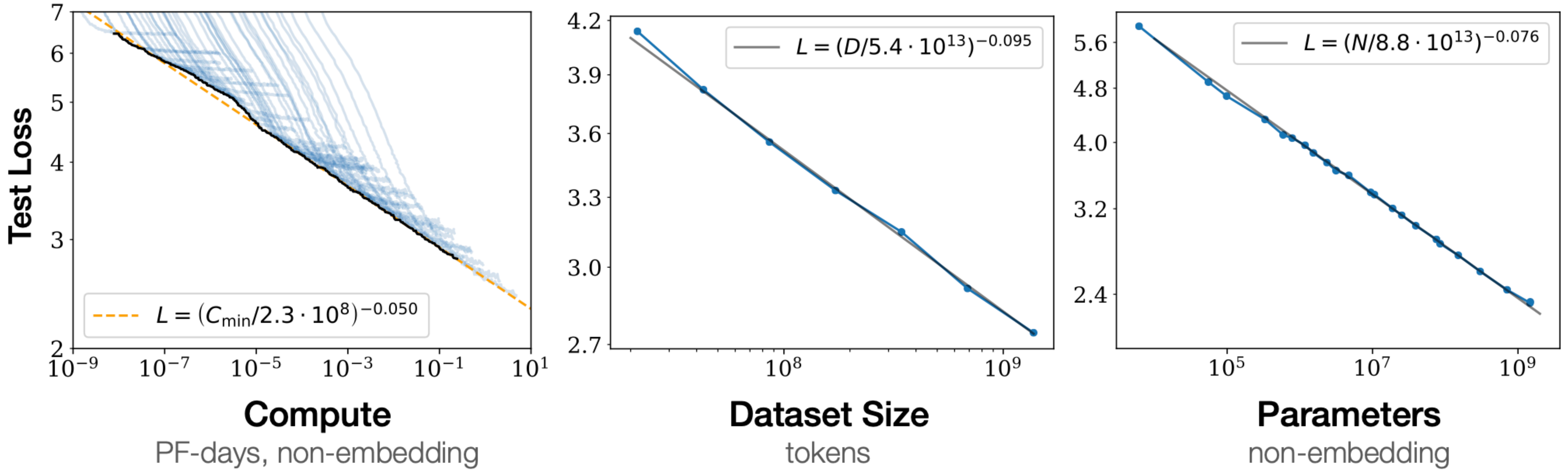
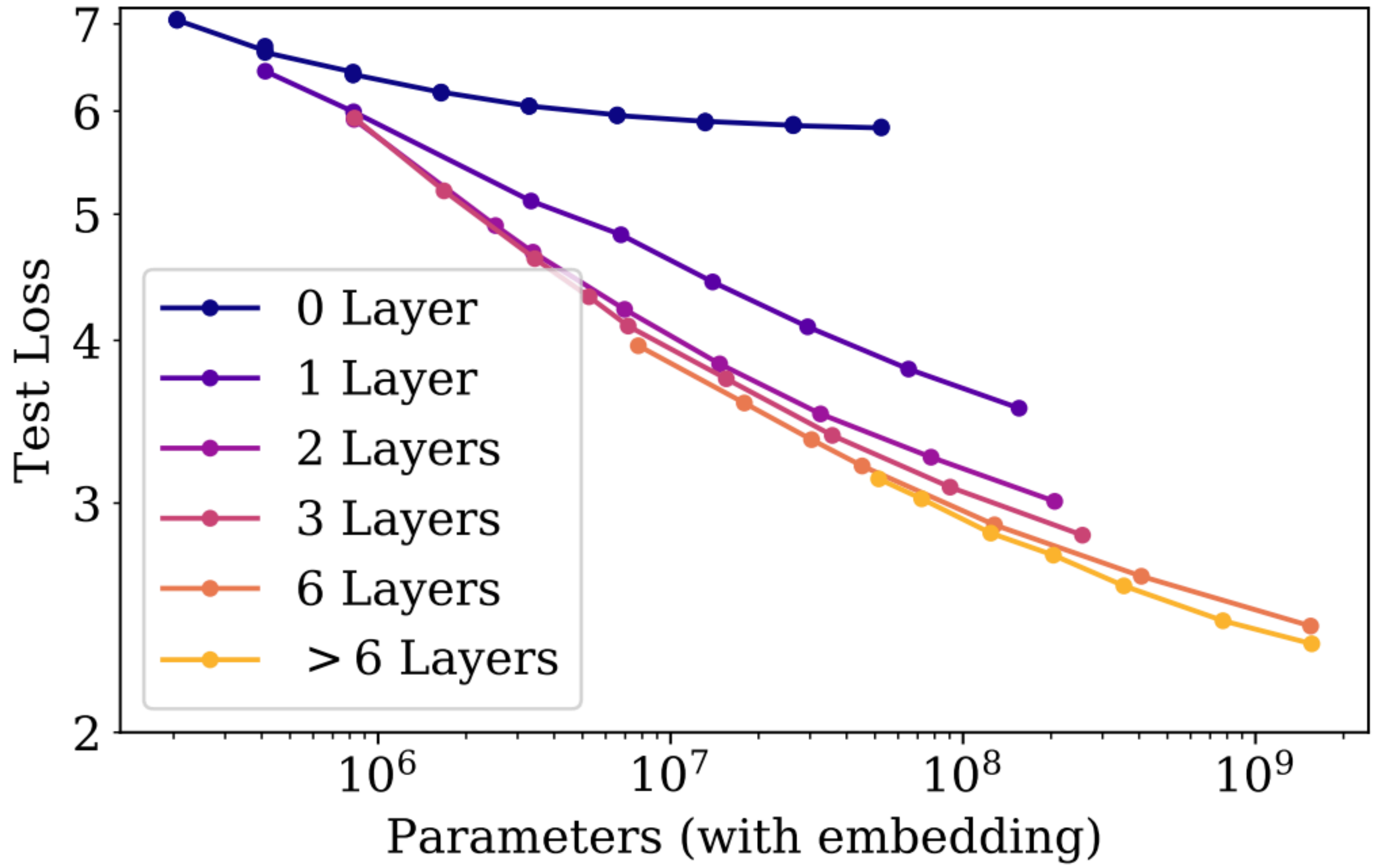
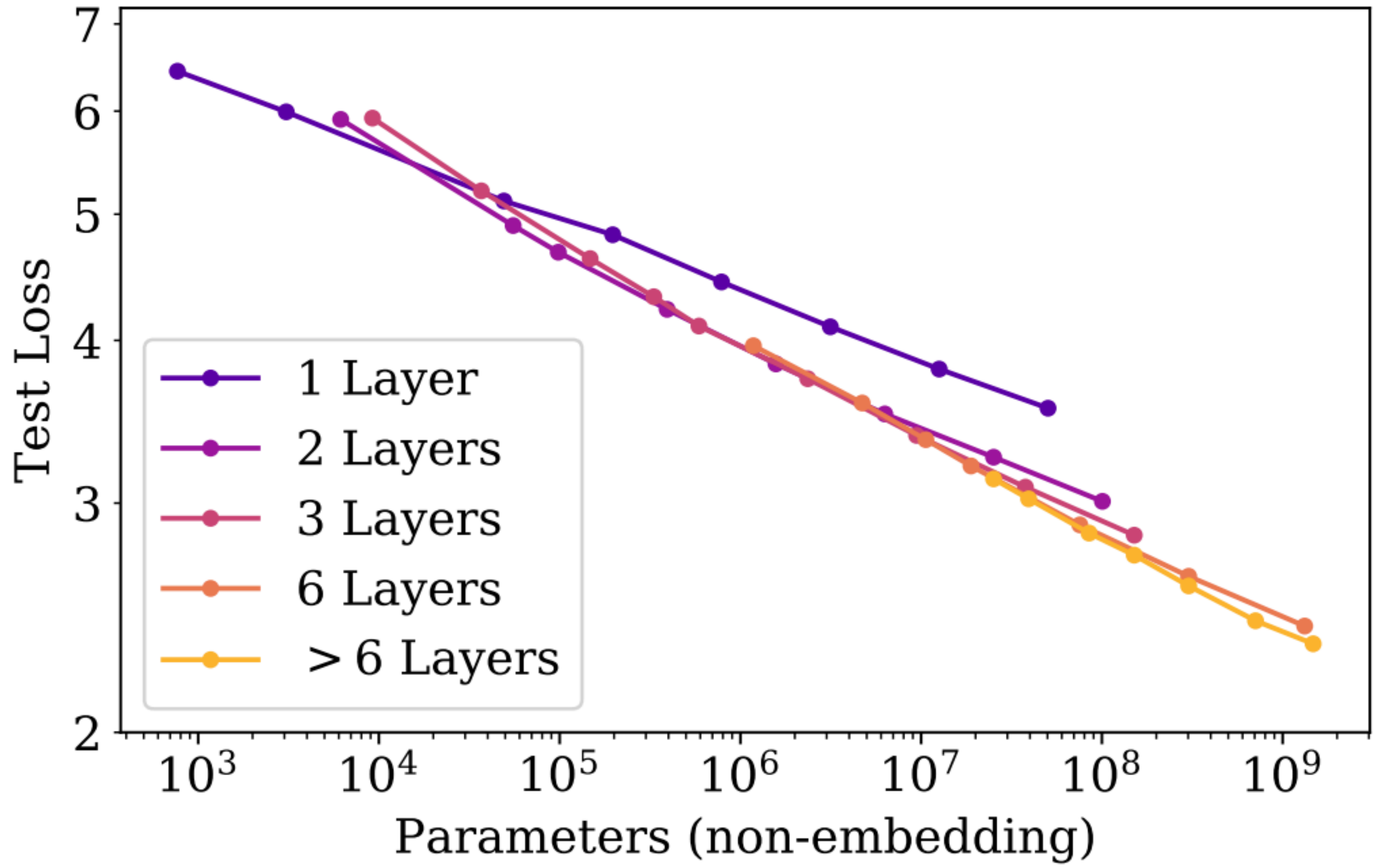


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when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

Excluding embeddings from parameter count

- Power law relationship not so clear when embeddings are included





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:

$$L(N) = (N_c/N)^{\alpha_N} \quad \alpha_N \approx 0.076, \quad N_c \approx 8.8 \times 10^{13} \text{ (non-embd params)}$$

- For models with limited dataset size:

$$L(D) = (D_c/D)^{\alpha_D} \quad \alpha_D \approx 0.095, \quad D_c \approx 5.4 \times 10^{13} \text{ (tokens)}$$

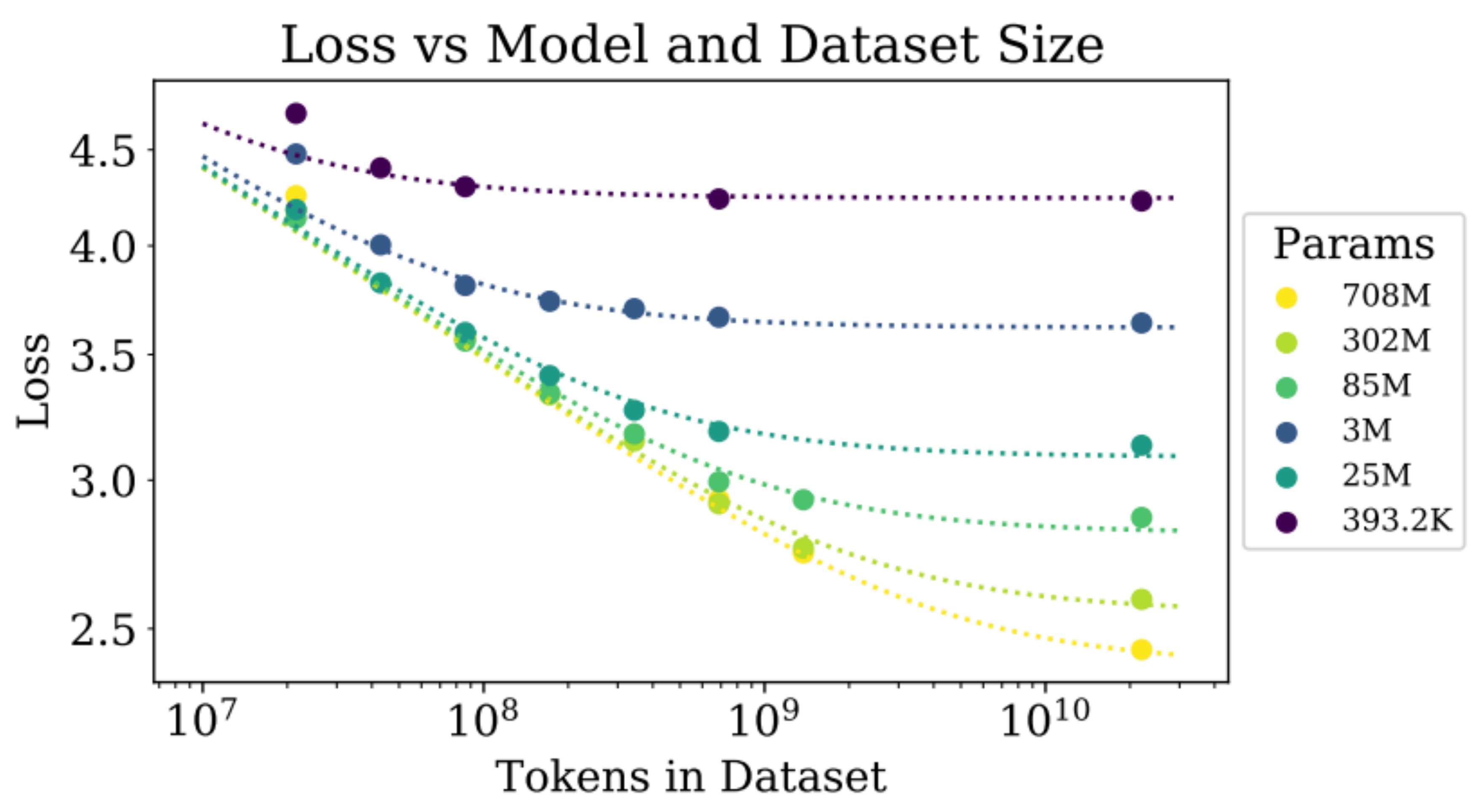
- For models trained with limited compute:

$$L(C) = (C_c^{min}/C_{min})^{\alpha_C^{min}} \quad \alpha_c^{min} \approx 0.050, \quad C_c^{min} \approx 3.1 \times 10^{8} \text{ (PF-days)}$$

Scaling laws for LLMs

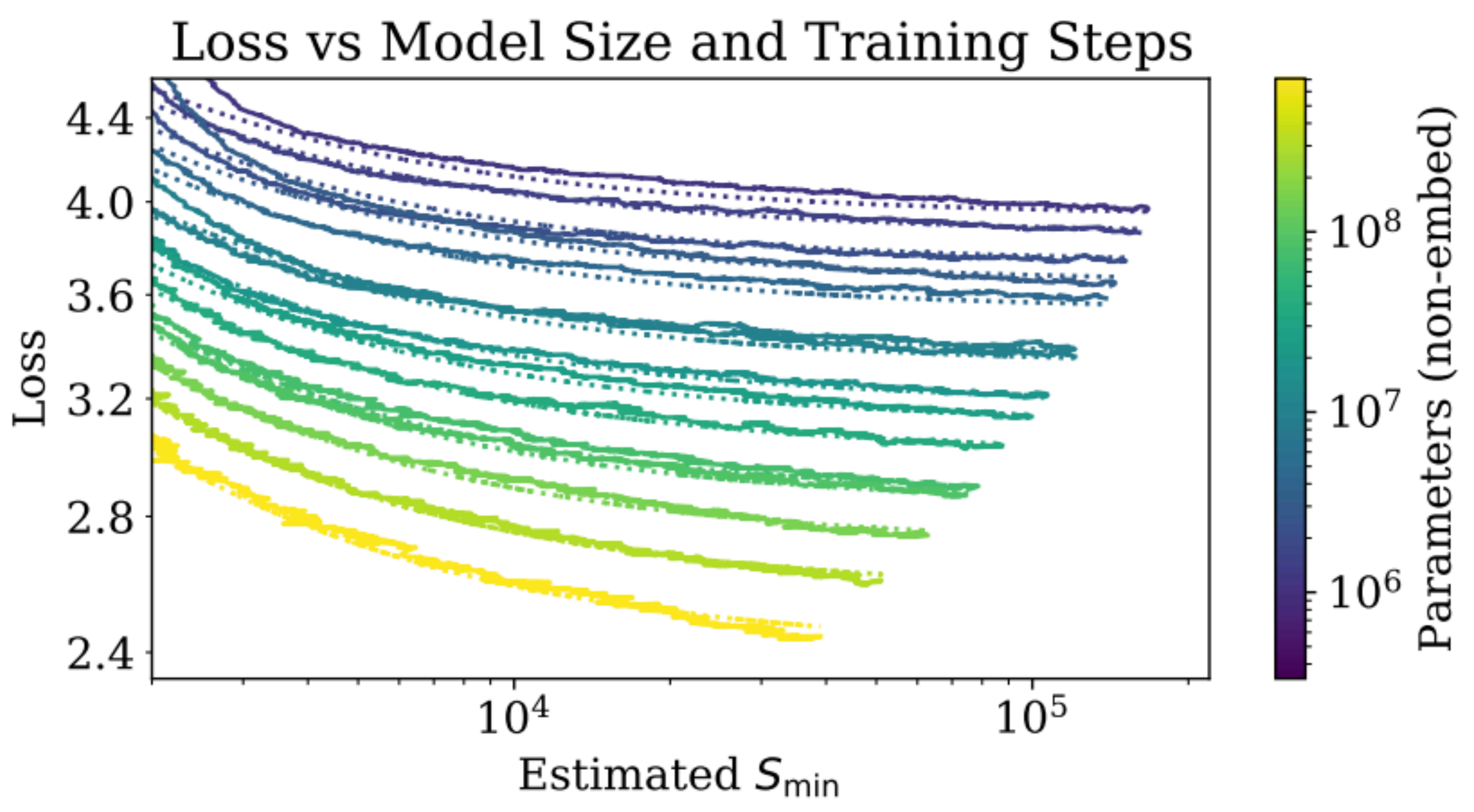
Test loss L as function
of model size N and dataset size D

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



Test loss L after transient period as function
of model size N and number of update steps S

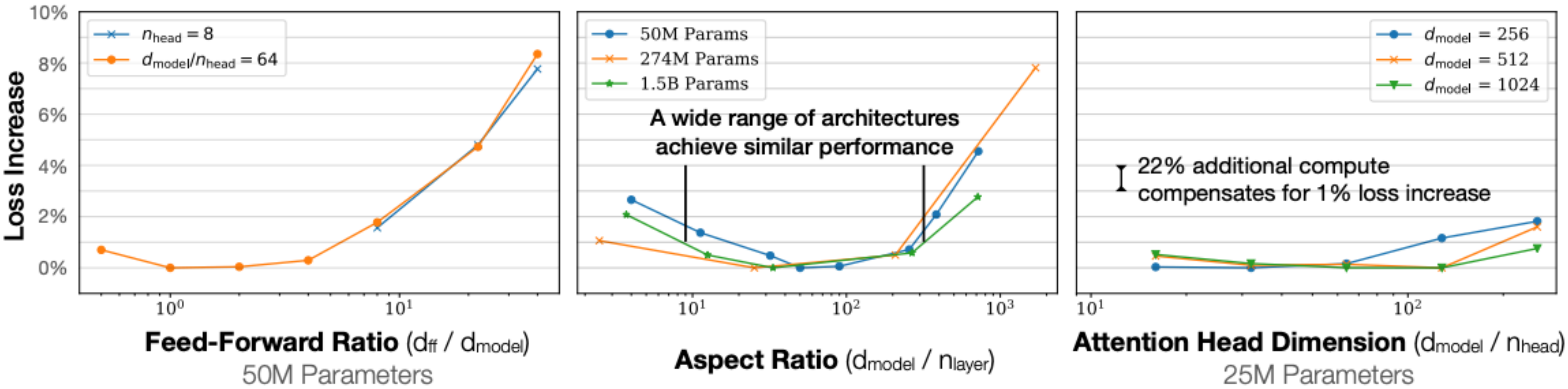
$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)}\right)^{\alpha_S}$$



N is number of model parameters (not including vocabulary and positional embeddings)
D is the number of tokens

Scaling laws for LLMs

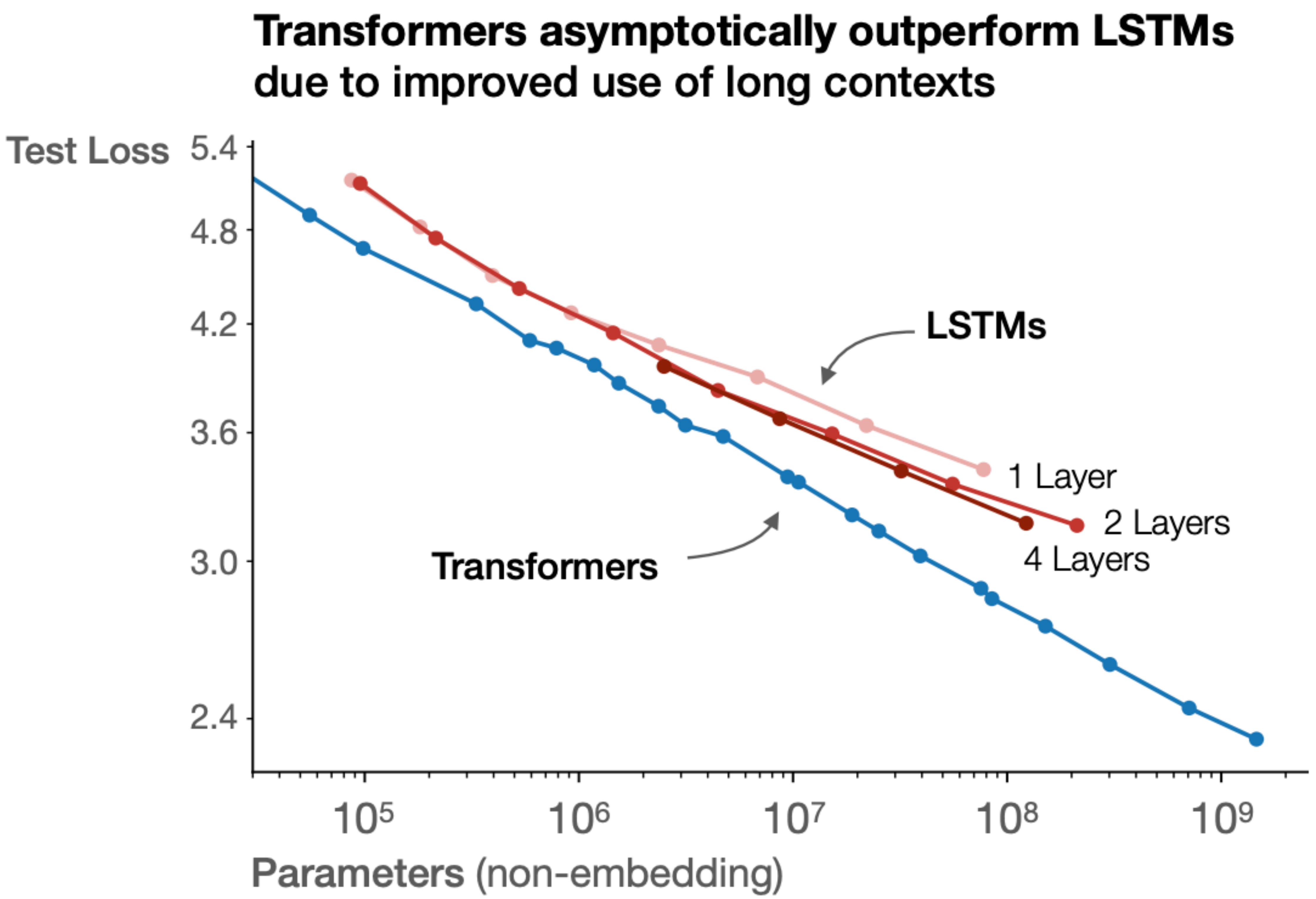
- Keeping model size N fixed, architecture shape doesn't matter that much

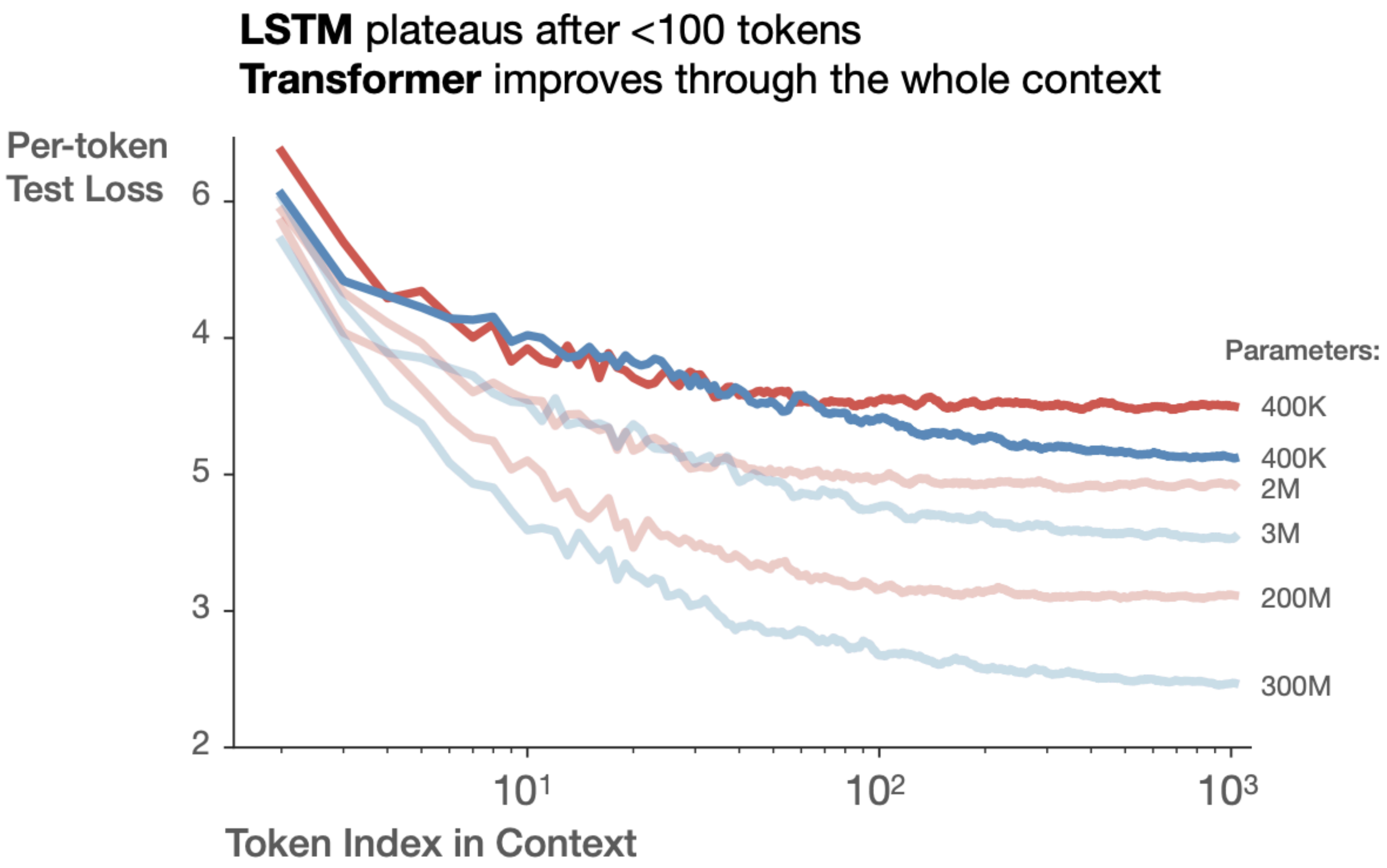


Scaling Laws for Neural Language Models, Kaplan et al, OpenAI, 2020

Comparing LSTM vs Transformers

- LSTM cannot take advantage of long context (>100 tokens)





Given a compute budget, what size model and amount of data should we train on?

Large models are more sample-efficient than small models

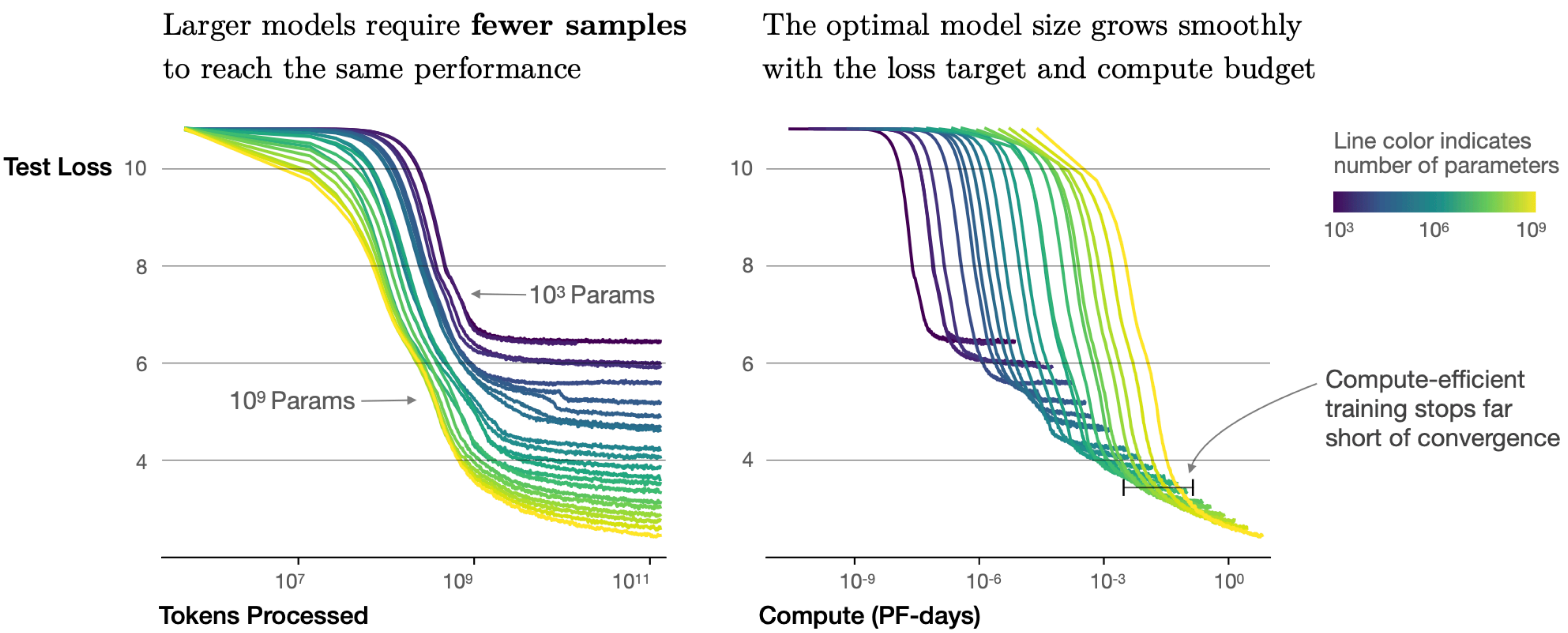
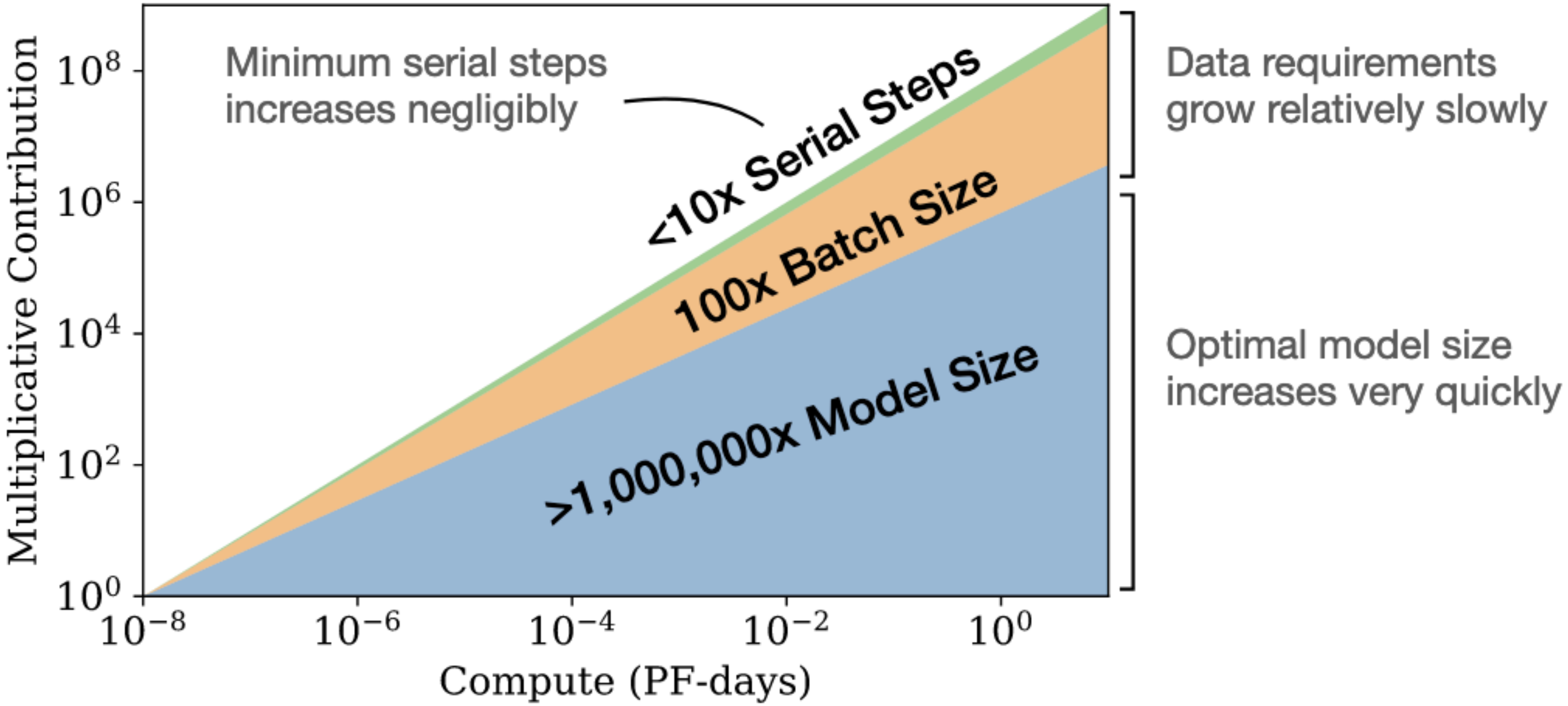


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

How to allocate increasing compute?

For compute-efficient training

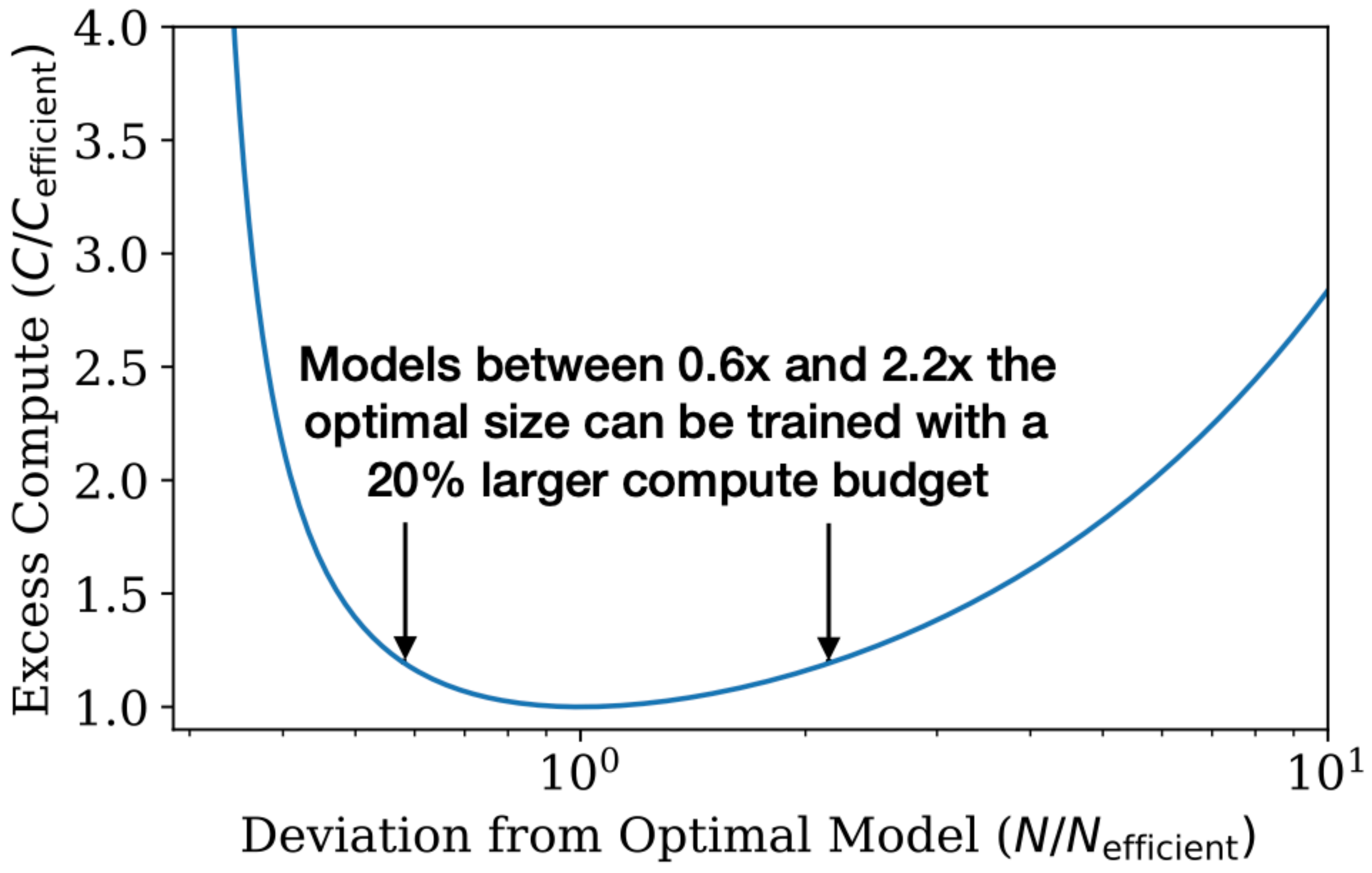
- Increase **model size** more than data (increase data sublinearly).
- Increase **batch size** as data size increases

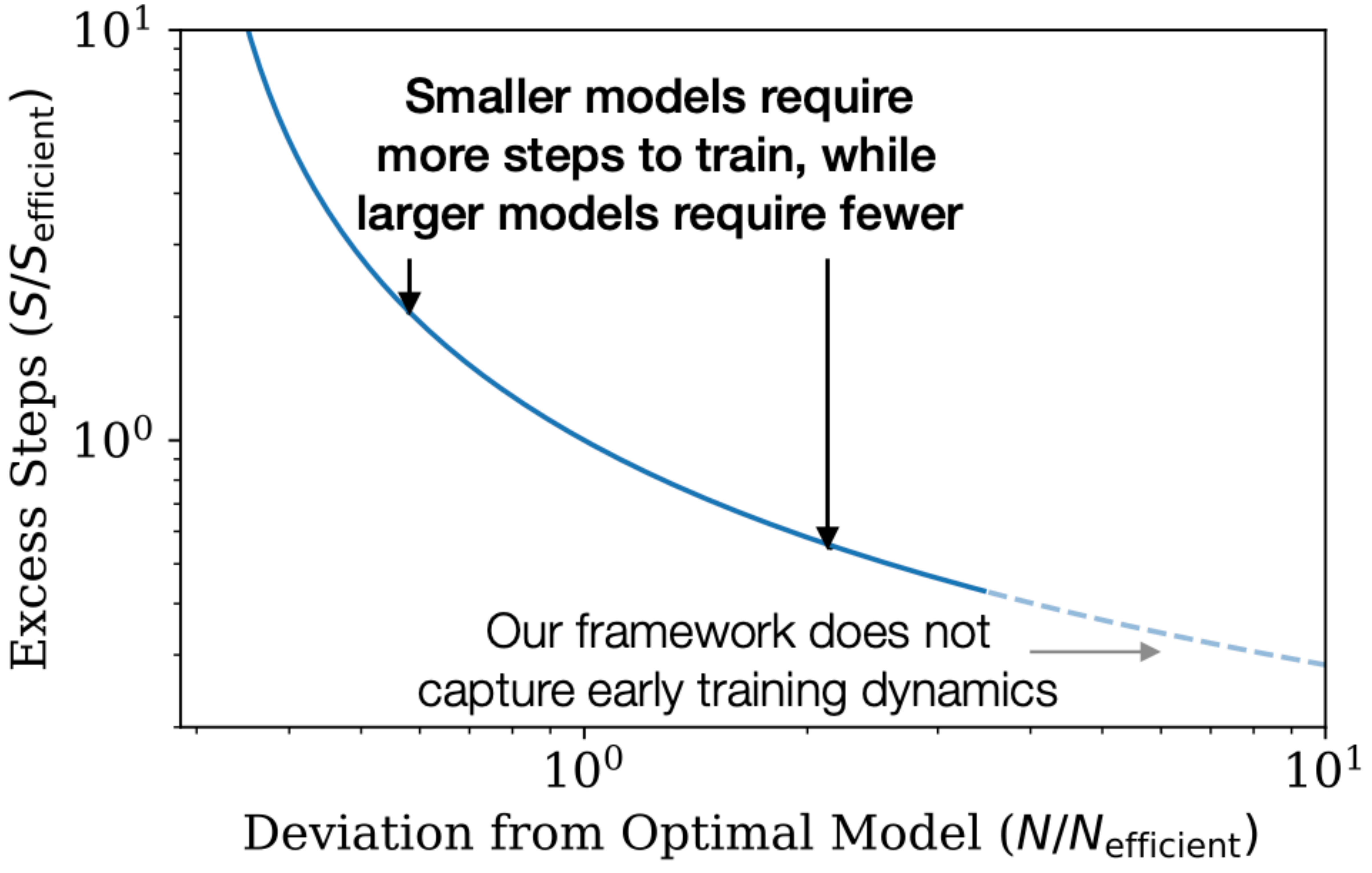


Billion-fold (10^9) increase in compute time

Optimal Allocation of Compute Budget

Training at fixed batch size (should increase batch size with more data)





Models larger than the optimal-size can train faster (with less steps)

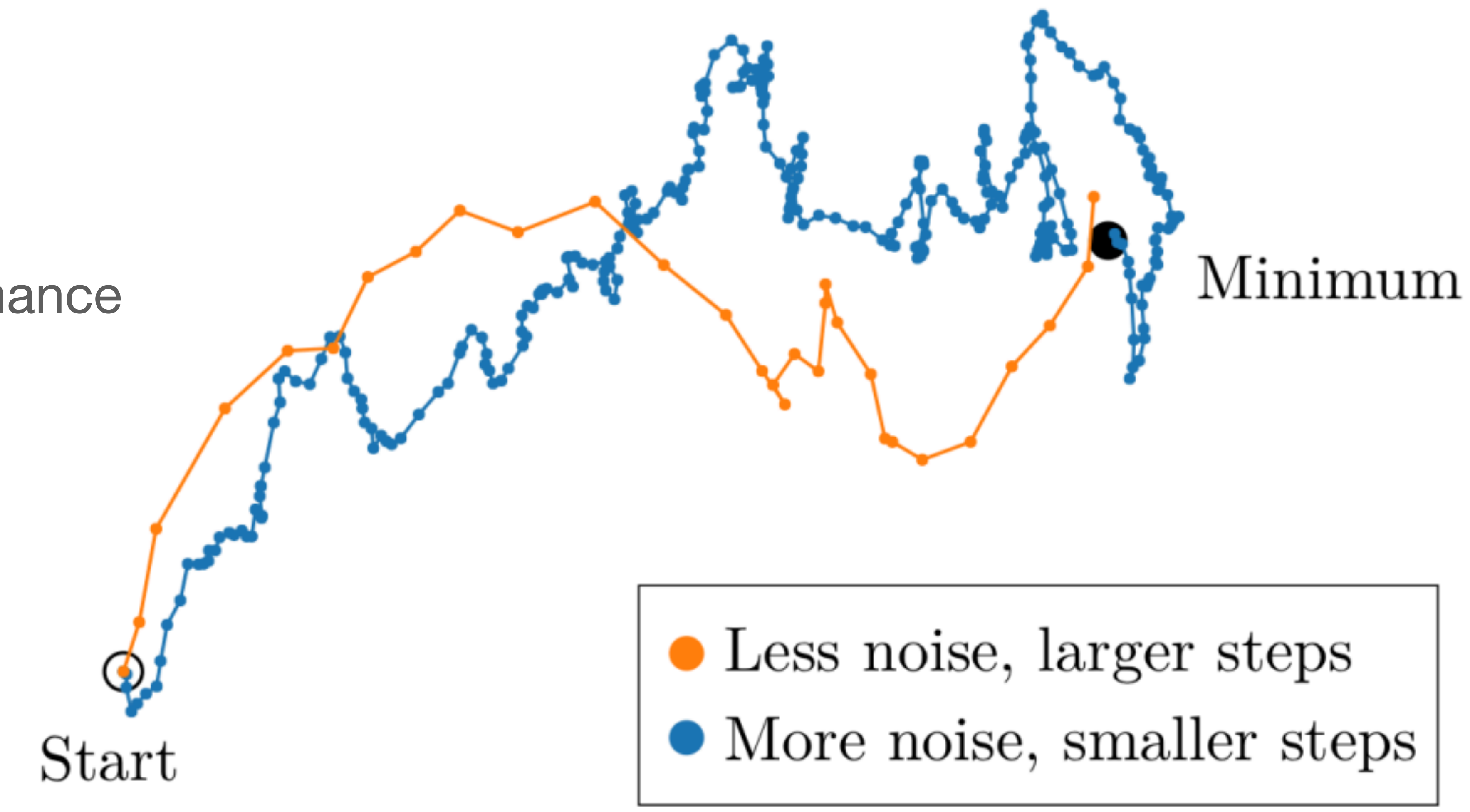
Critical batch size

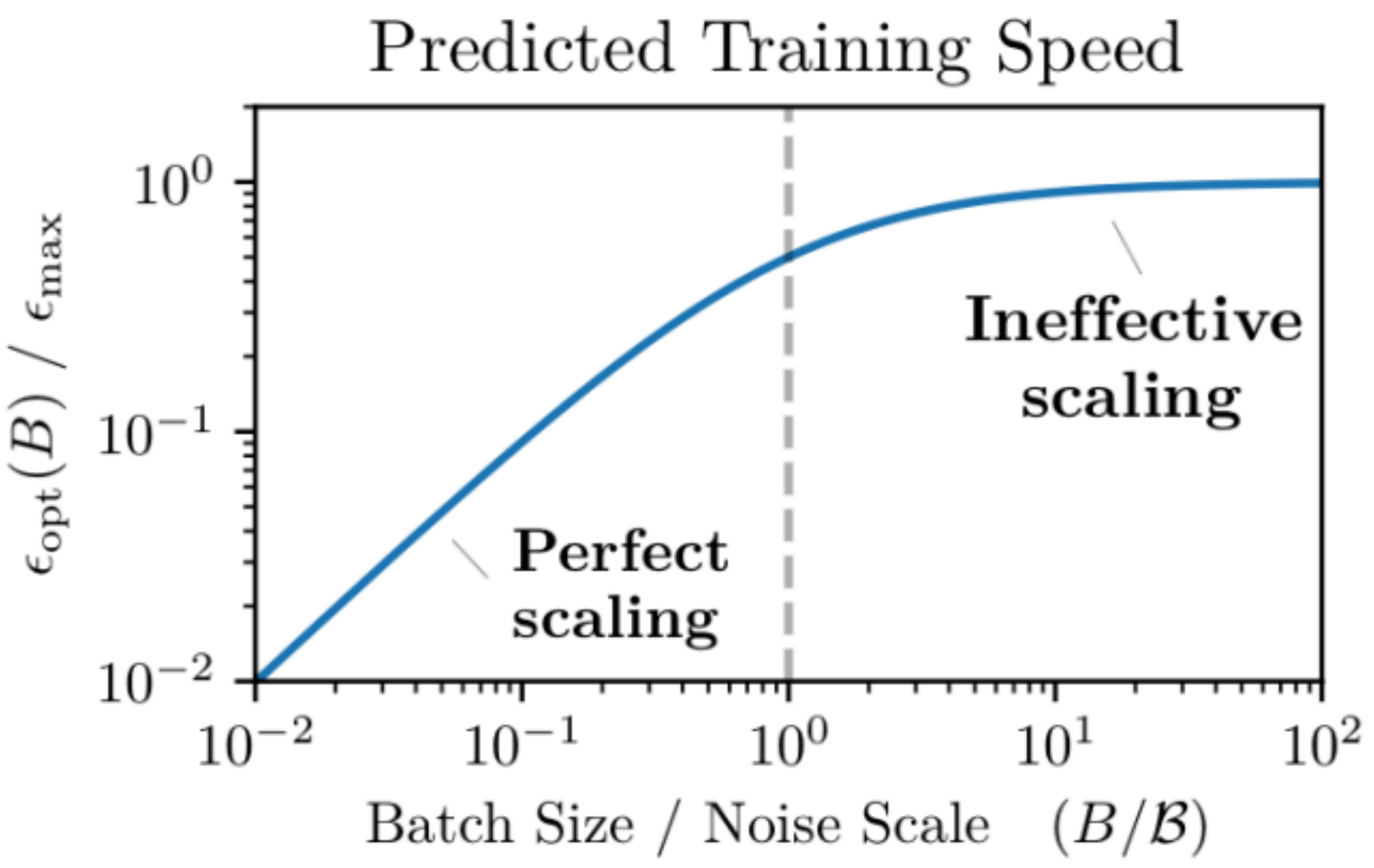
For compute efficient training, train with $B_{\text{crit}} = \frac{E_{\min}}{S_{\min}}$

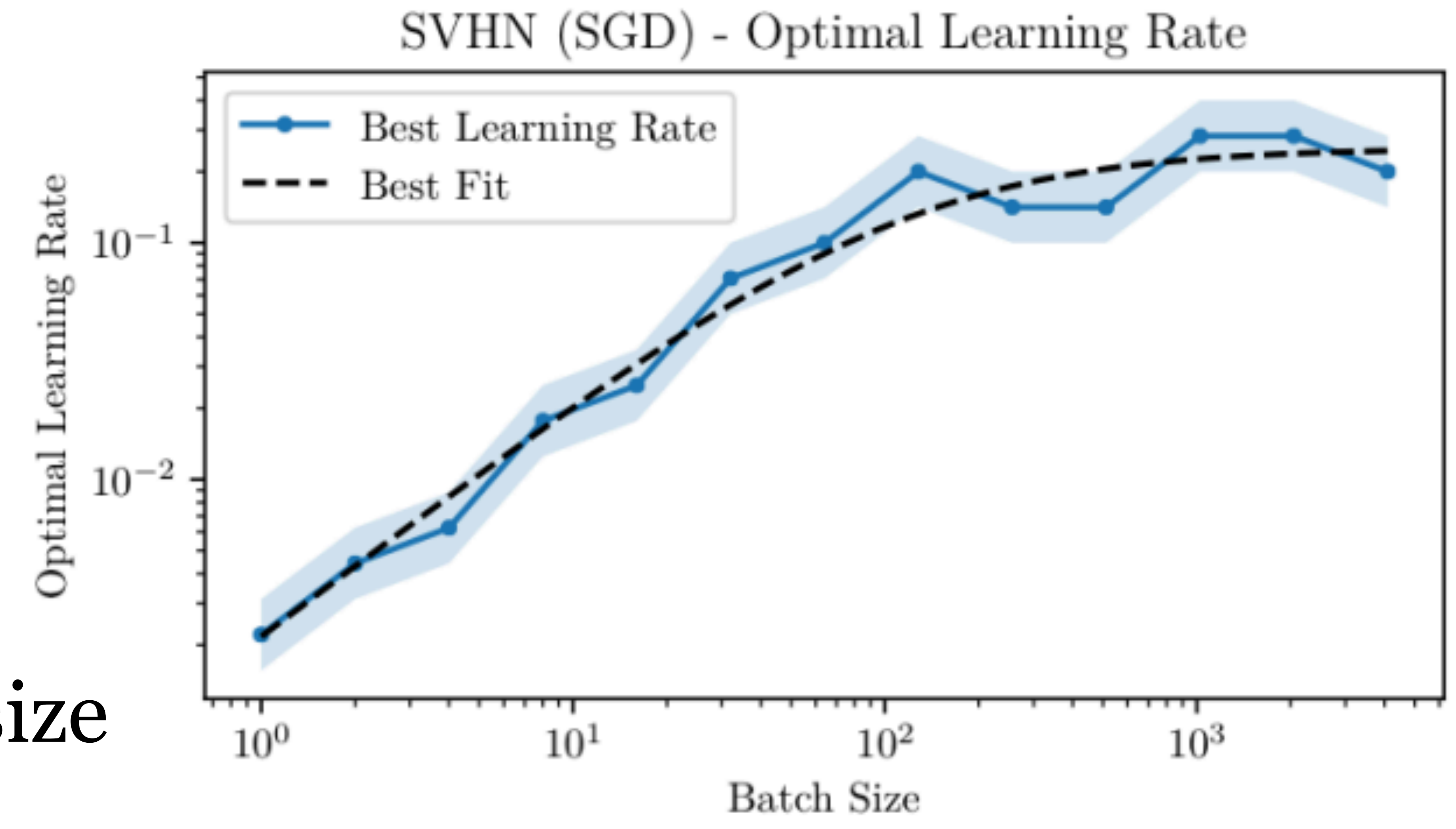
Number of training examples to reach a given performance

Number of training steps

- Larger B: more stable gradient, less training steps
- Critical batch size: above which scaling efficiency decreases significantly







- Optimal learning rate scales linearly with batch size

An Empirical Model of Large-Batch Training [MacCandlish, et al. 2018] - arXiv:1812.061

Critical batch size as function of test loss

$$B_{\mathrm{crit}}(L) \equiv \frac{E_{\mathrm{min}}}{S_{\mathrm{min}}}$$

$$B_{\mathrm{crit}}(L) \approx \frac{B_*}{L^{1/\alpha_B}}$$

$B_* \sim 2 \cdot 10^8$ tokens

$\alpha_B \sim 0.21$

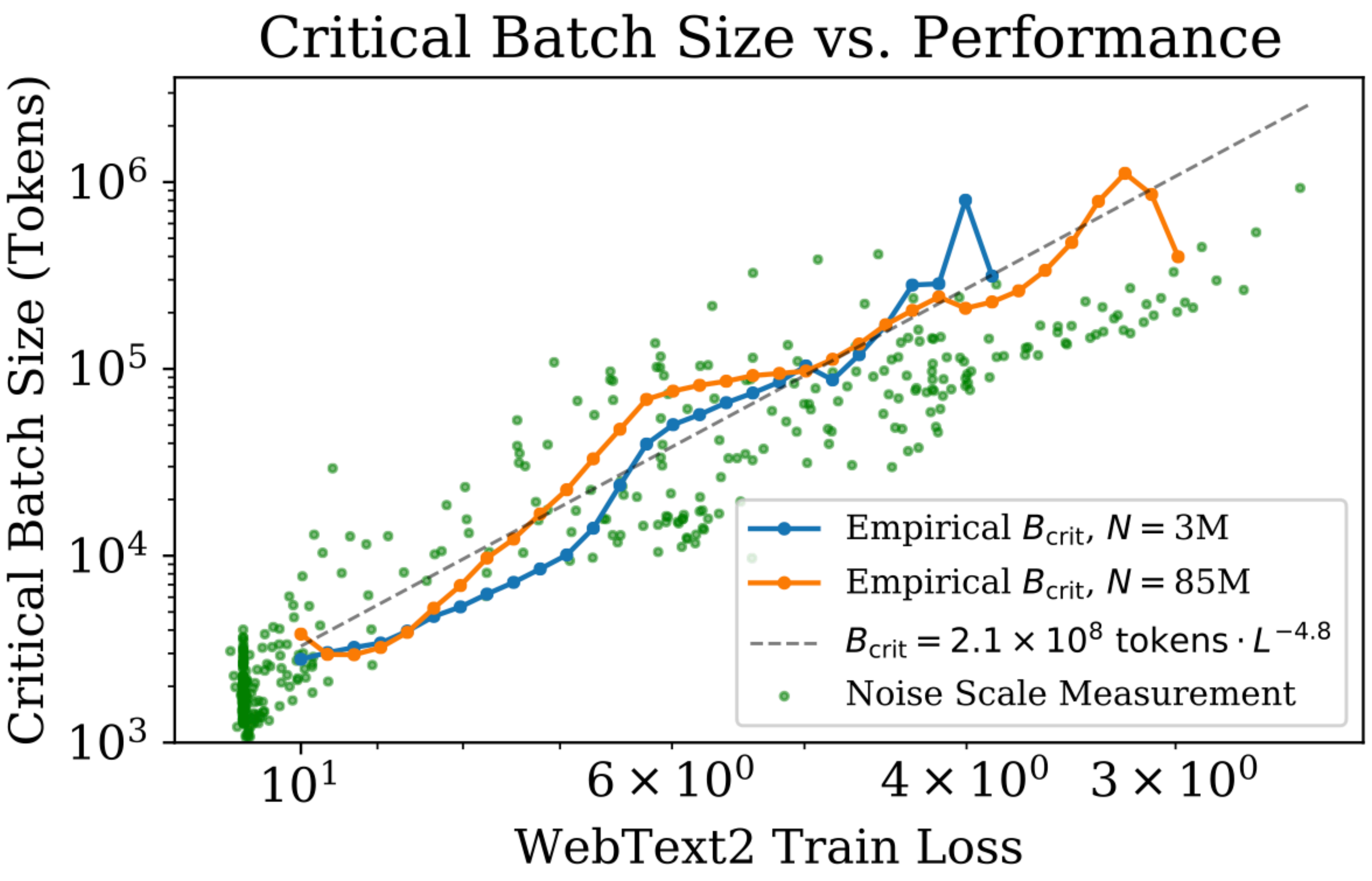


Figure 10 The critical batch size B_{crit} follows a power law in the loss as performance increase, and does not depend directly on the model size. We find that the critical batch size approximately doubles for every 13% decrease in loss. B_{crit} is measured empirically from the data shown in Figure 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
- 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

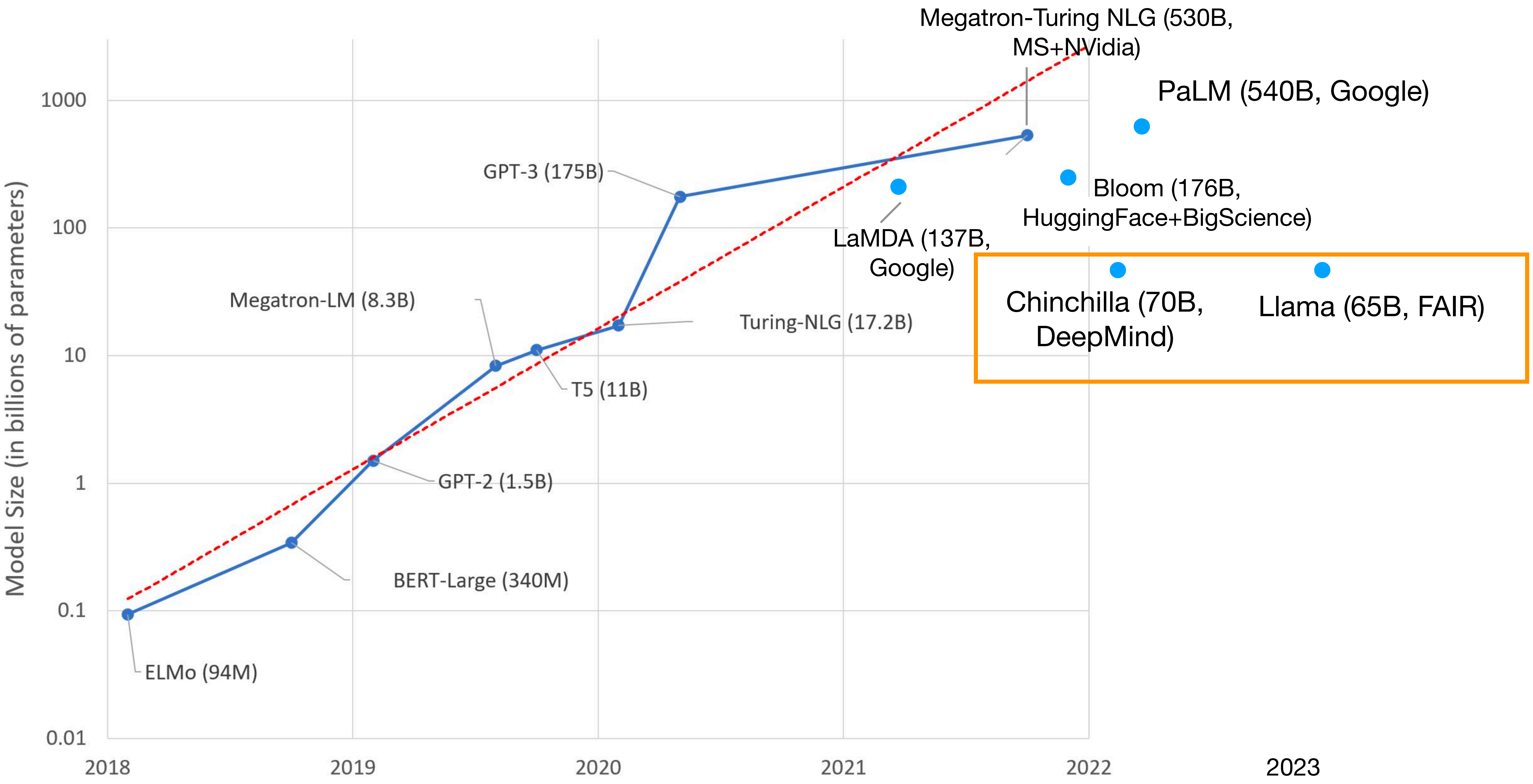
Lessons from scaling LLMs

- 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
- If no constraints on data and model size, with given compute budget C

$$N \propto C^{\alpha_C^{\min}/\alpha_N} \qquad B \propto C^{\alpha_C^{\min}/\alpha_B} \qquad S \propto C^{\alpha_C^{\min}/\alpha_S} \qquad D = B \cdot S$$

Is larger models always better?
Can we train high-performance smaller models with more data?

Is bigger always better?



https://huggingface.co/blog/large-language-models

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*

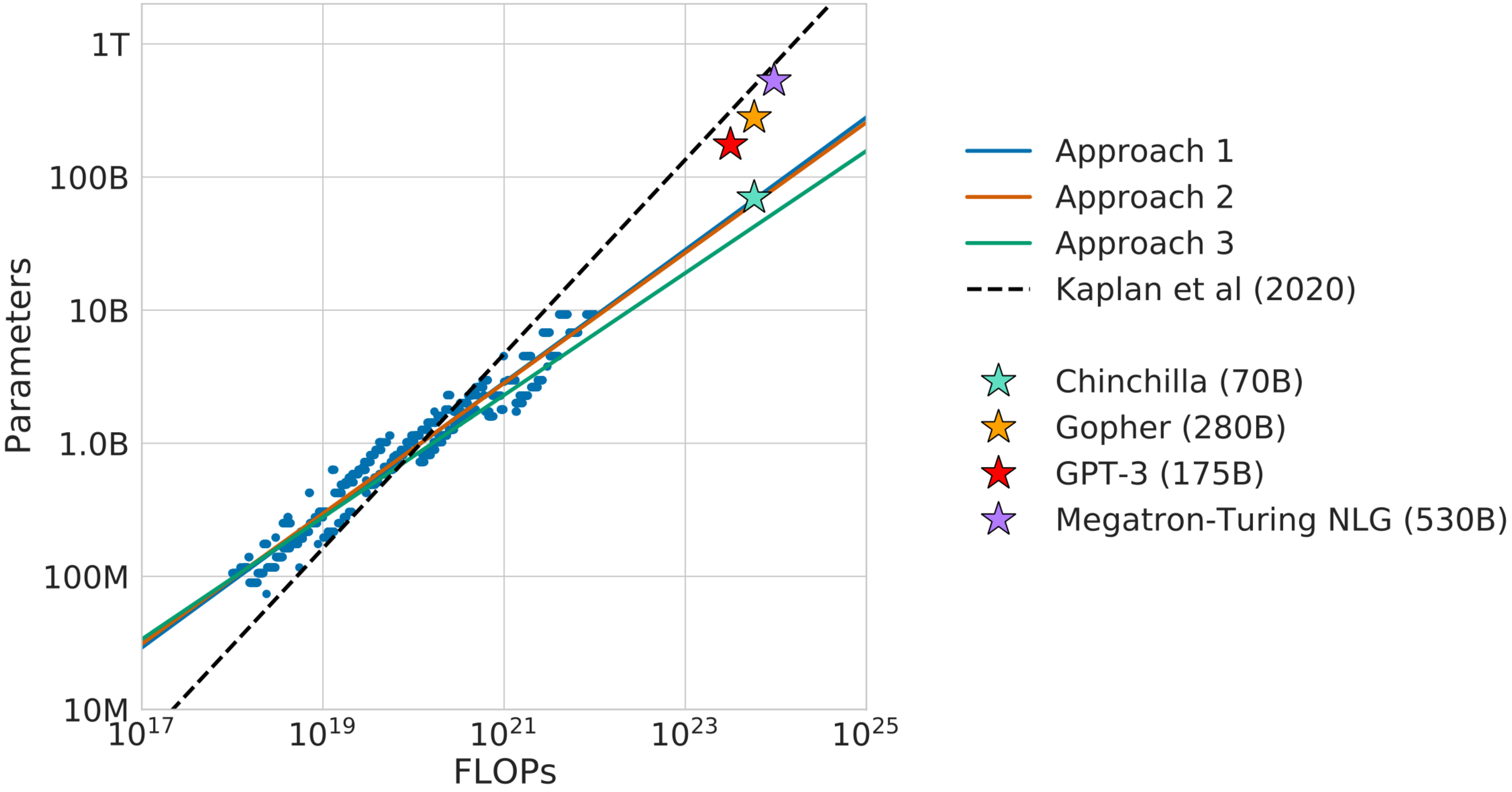
https://arxiv.org/abs/2203.15556

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**.

- **Better to scale model size and number of tokens linearly!**



- For different model sizes, choose number of training tokens to keep FLOPs constant

IsoFLOP

Model size

Data size

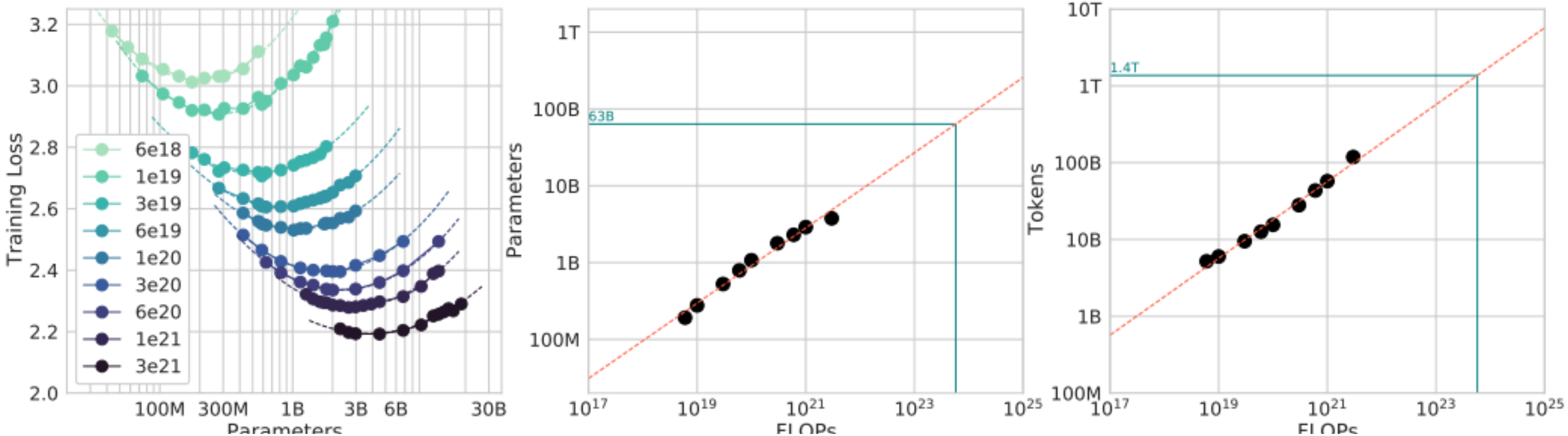


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

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

LLaMA: Open and Efficient Foundation Language Models

Hugo Touvron[*], Thibaut Lavril[*], Gautier Izacard[*], Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave[*], Guillaume Lample[*]

Meta AI

https://arxiv.org/abs/2302.13971

LLaMA

- 65B model trained on 1.4T tokens for ~21 days on 2048 A100 GPU with 80GB RAM.

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

LLaMA: Open and Efficient Foundation Language Models [Touvron et al. FAIR, 2023]

LLaMA

Architecture

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

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

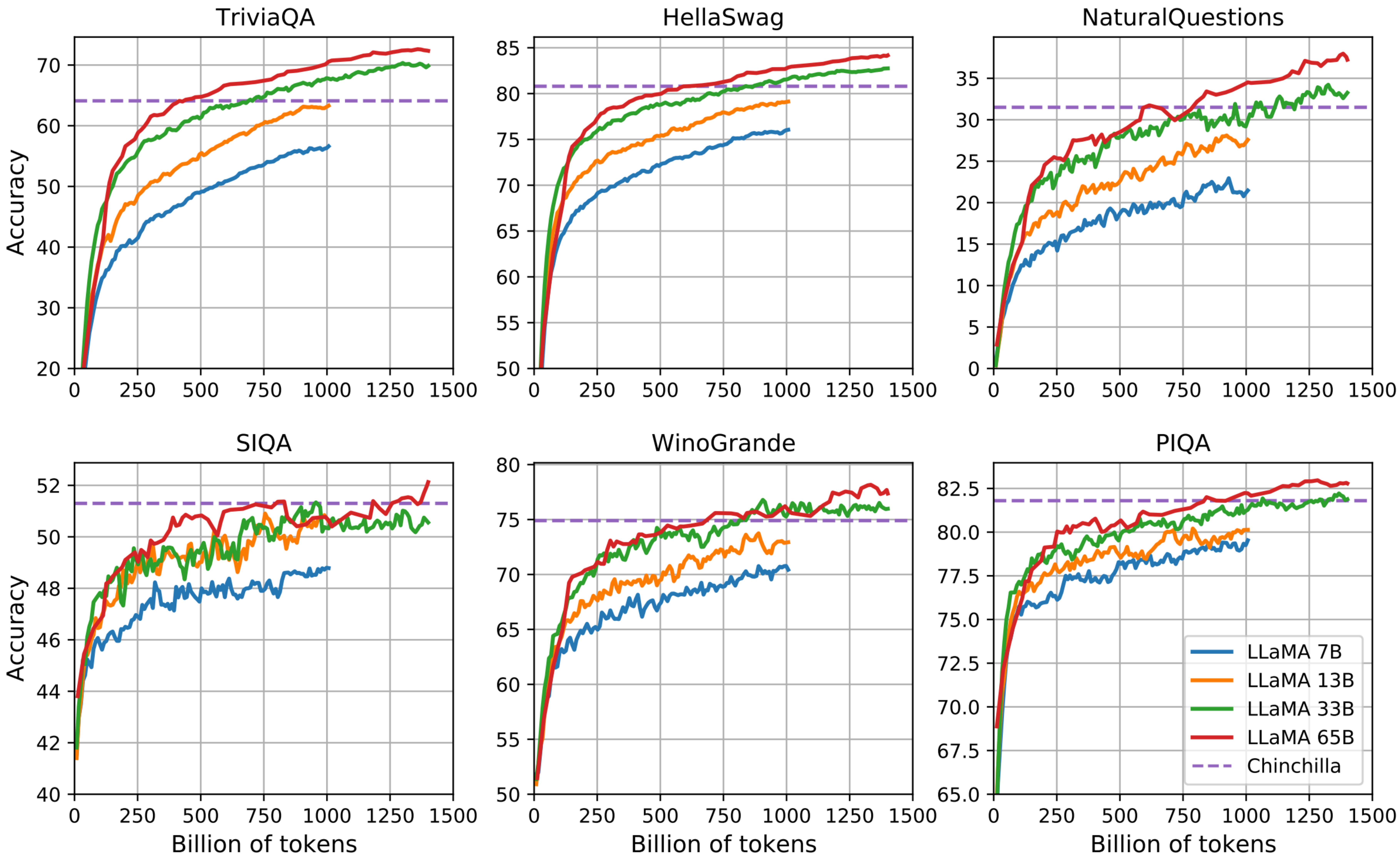
Rotary Embeddings [GPTNeo]. We remove the absolute positional embeddings, and instead, add rotary positional embeddings (RoPE), introduced by Su et al. (2021), at each layer of the network.

Training data

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

LLaMA: Open and Efficient Foundation Language Models [Touvron et al. FAIR, 2023]

LLaMA



<u>*LLaMA: Open and Efficient Foundation Language Models*</u> *[Touvron et al. FAIR, 2023]*

LLaMA

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	**88.0**	82.3	-	83.4	**81.1**	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	**82.8**	**52.3**	**84.2**	77.0	78.9	56.0	**60.2**

LLaMA: Open and Efficient Foundation Language Models [Touvron et al. FAIR, 2023]

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

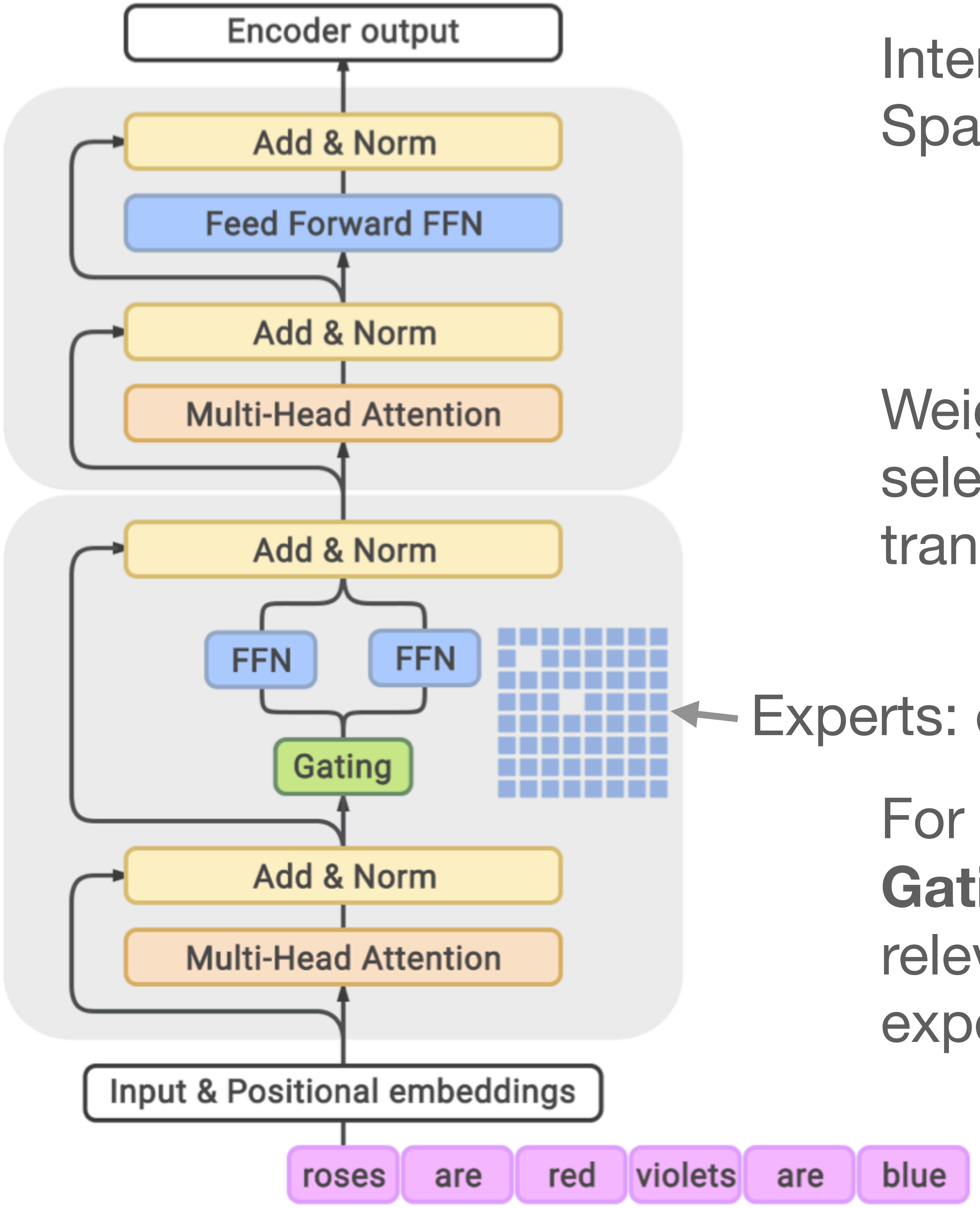
Nan Du [*1] **Yanping Huang** [*1] **Andrew M. Dai** [*1] **Simon Tong** [1] **Dmitry Lepikhin** [1] **Yuanzhong Xu** [1]
Maxim Krikun [1] **Yanqi Zhou** [1] **Adams Wei Yu** [1] **Orhan Firat** [1] **Barret Zoph** [1] **Liam Fedus** [1] **Maarten Bosma** [1]
Zongwei Zhou [1] **Tao Wang** [1] **Yu Emma 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

Transformer

MoE



Interleaved transformer and MoE layers
Sparse activation of experts

Weighted average of outputs from selected experts is passed to the transformer layer

Experts: each expert is a FFN

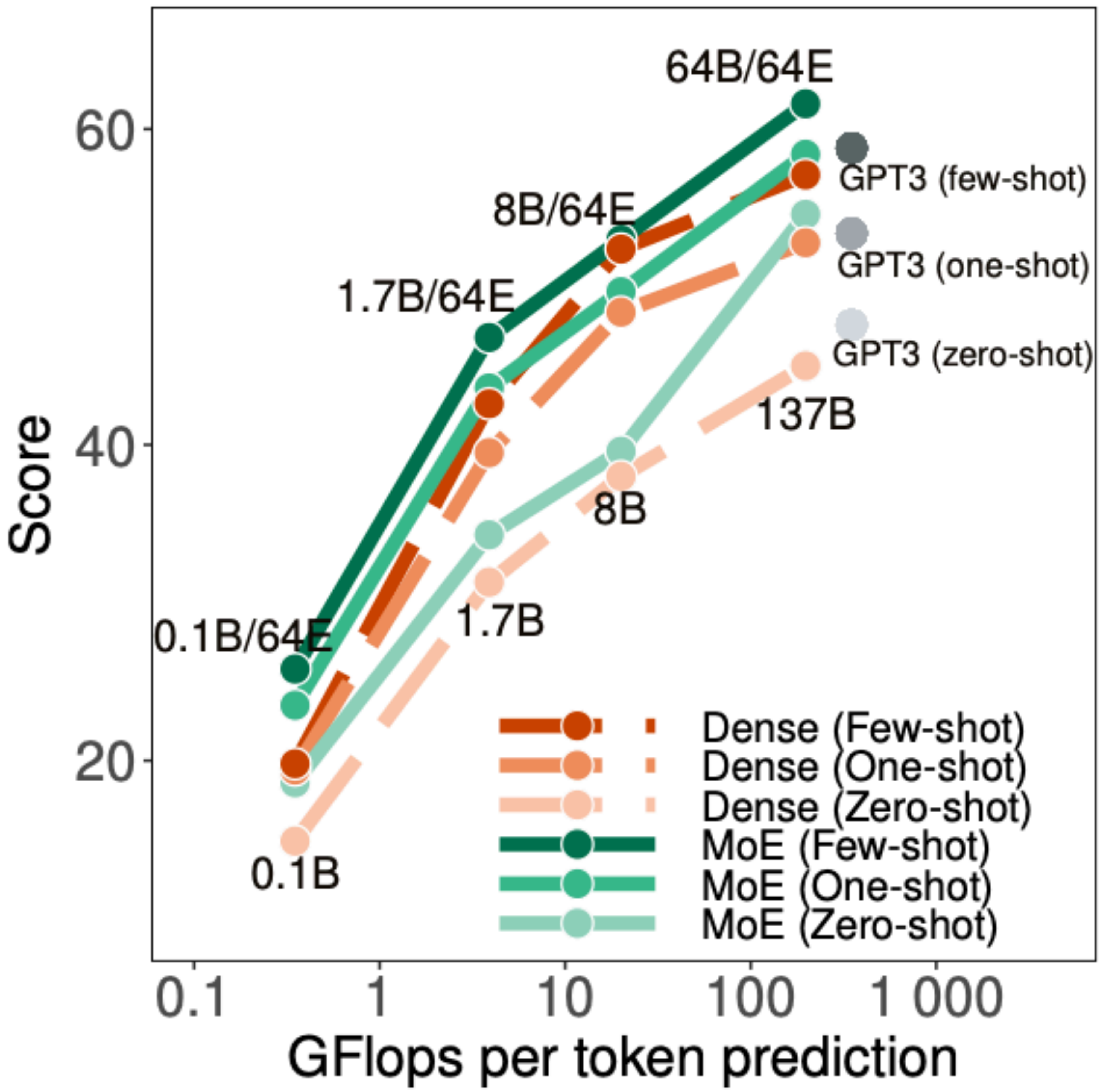
For each input token (e.g. 'roses'), the **Gating** module selects the two most relevant experts out of 64. Two different experts selected for each token.

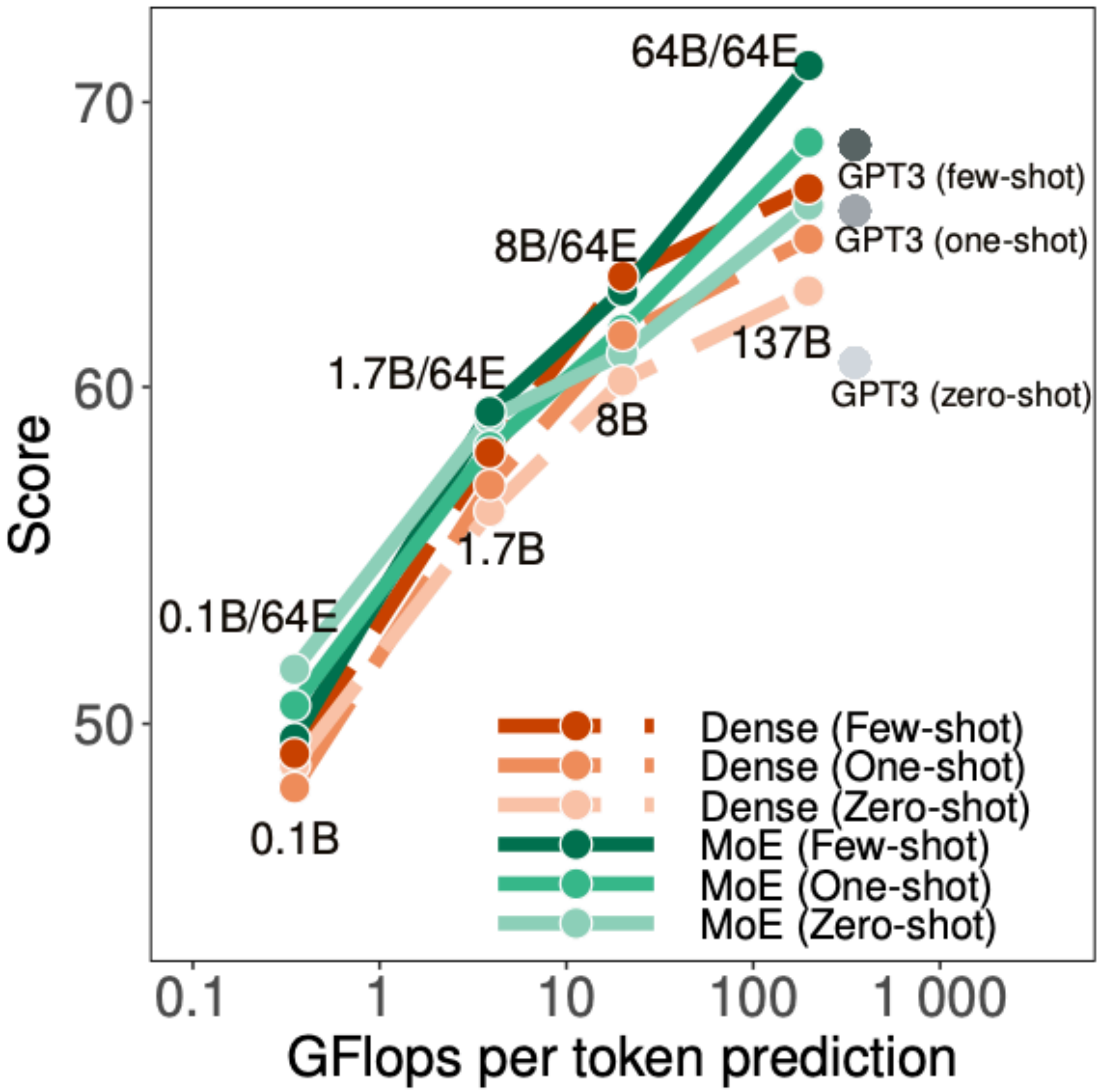
Larger models with less activated parameters per input token
More performant with similar amount of compute

Model Name	Model Type	n_{params}	$n_{\text{act-params}}$
BERT	Dense Encoder-only	340M	340M
T5	Dense Encoder-decoder	13B	13B
GPT-3	Dense Decoder-only	175B	175B
Jurassic-1	Dense Decoder-only	178B	178B
Gopher	Dense Decoder-only	280B	280B
Megatron-530B	Dense Decoder-only	530B	530B
GShard-M4	MoE Encoder-decoder	600B	1.5B
Switch-C	MoE Encoder-decoder	1.5T	1.5B
GLaM (64B/64E)	MoE Decoder-only	1.2T	96.6B

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

PaLM: Scaling Language Modeling with Pathways, Chowdhery et al, Google, 2022

Architecture

- SwiGLU activation: $\text{Swish}(xW) \otimes xV$
- Parallel layers
  - Serial: $y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x)))$
  - Parallel: $y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$
    - 15% faster training speed (degradation for small models 8B, but no degradation at 62B)
- Attention: Shared key-value across heads, query is still separately projected per head
- RoPE (rotary position) embeddings
- Shared input-output embeddings
- No biases: increased training stability
- Vocabulary: SentencePiece with 256k tokens

Training data

- 780 billion tokens of natural language + source code from github

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

- Trained on two TPU v4 pods
  - Each pod had 3072 TPU chips attached to 768 hosts (total 6144 chips)
  - Each pod had full copy of model parameters
- Model + data parallelism, no pipeline parallelism
  - 12-way model parallelism, 256-way data sharing

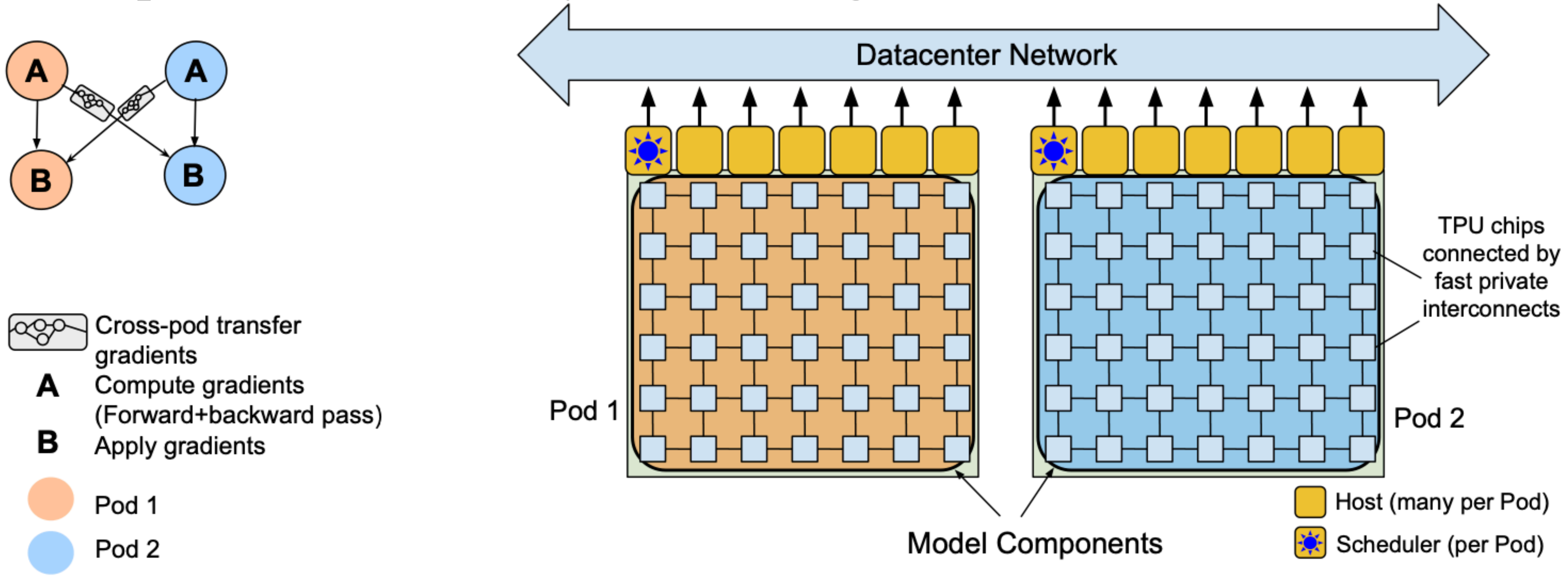
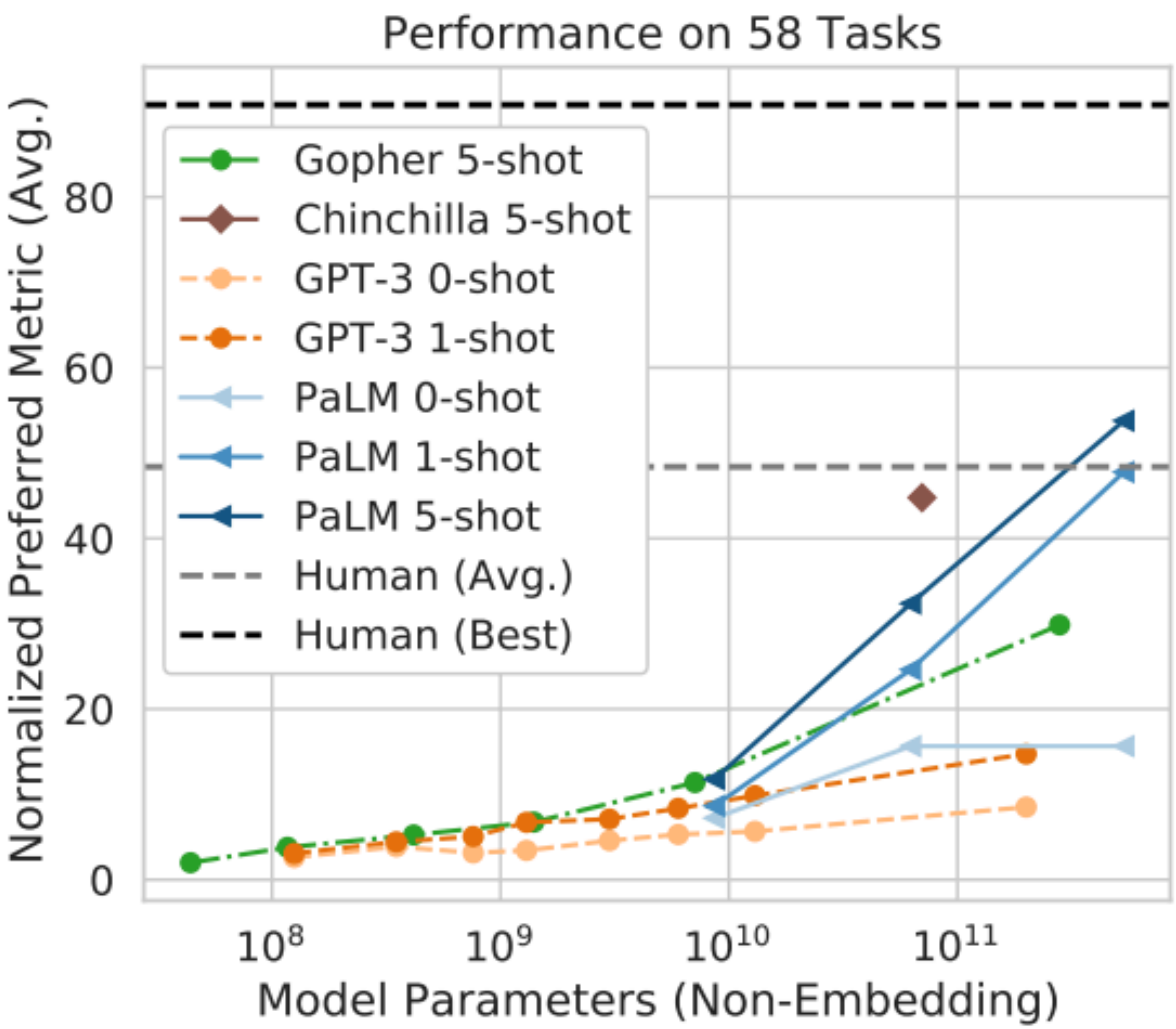
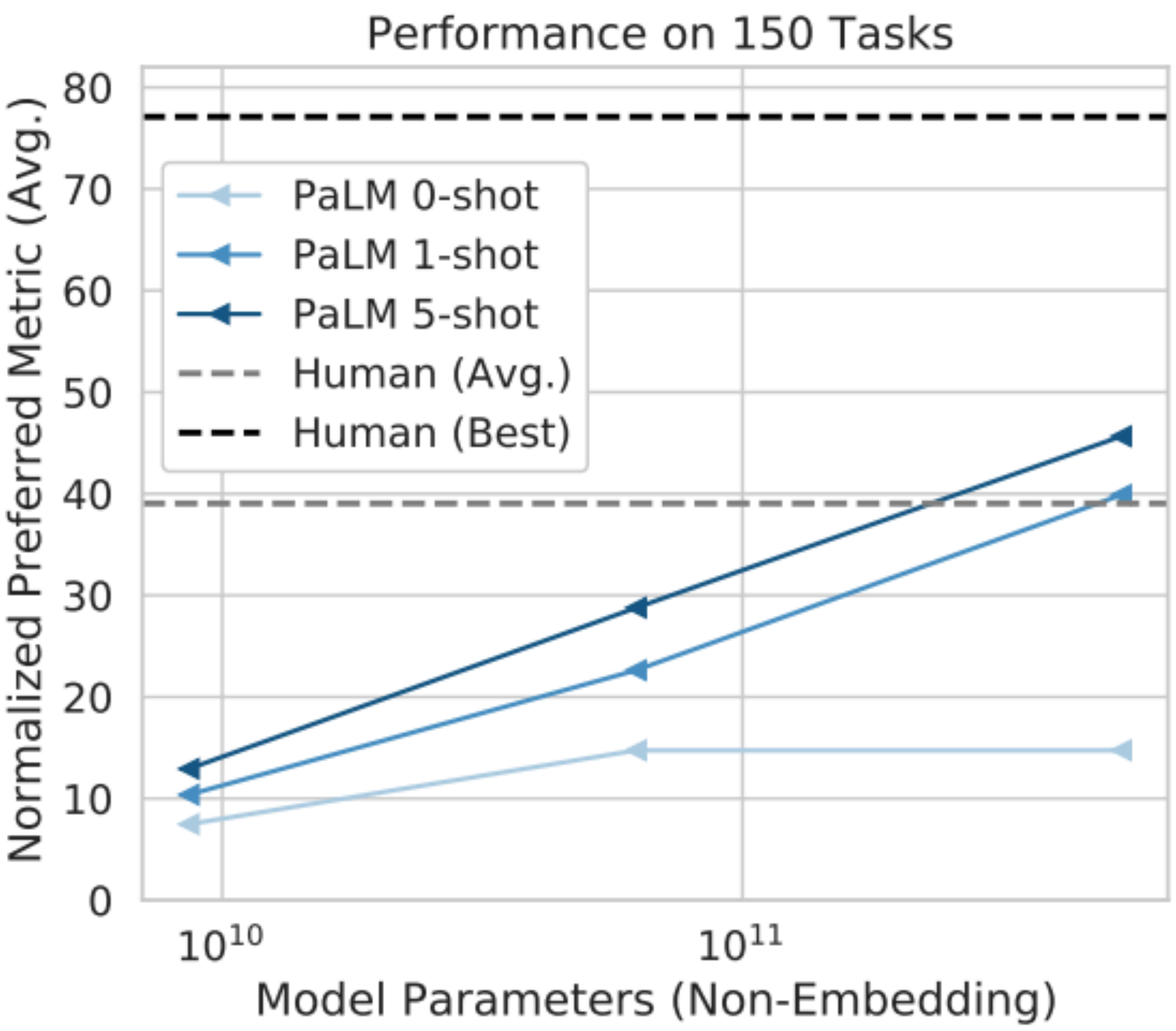


Figure 2: The Pathways system (Barham et al., 2022) scales training across two TPU v4 pods using two-way data parallelism at the pod level.

Chinchilla: 70B
GPT-3: 175B
Gopher: 280B
PaLM: 540B

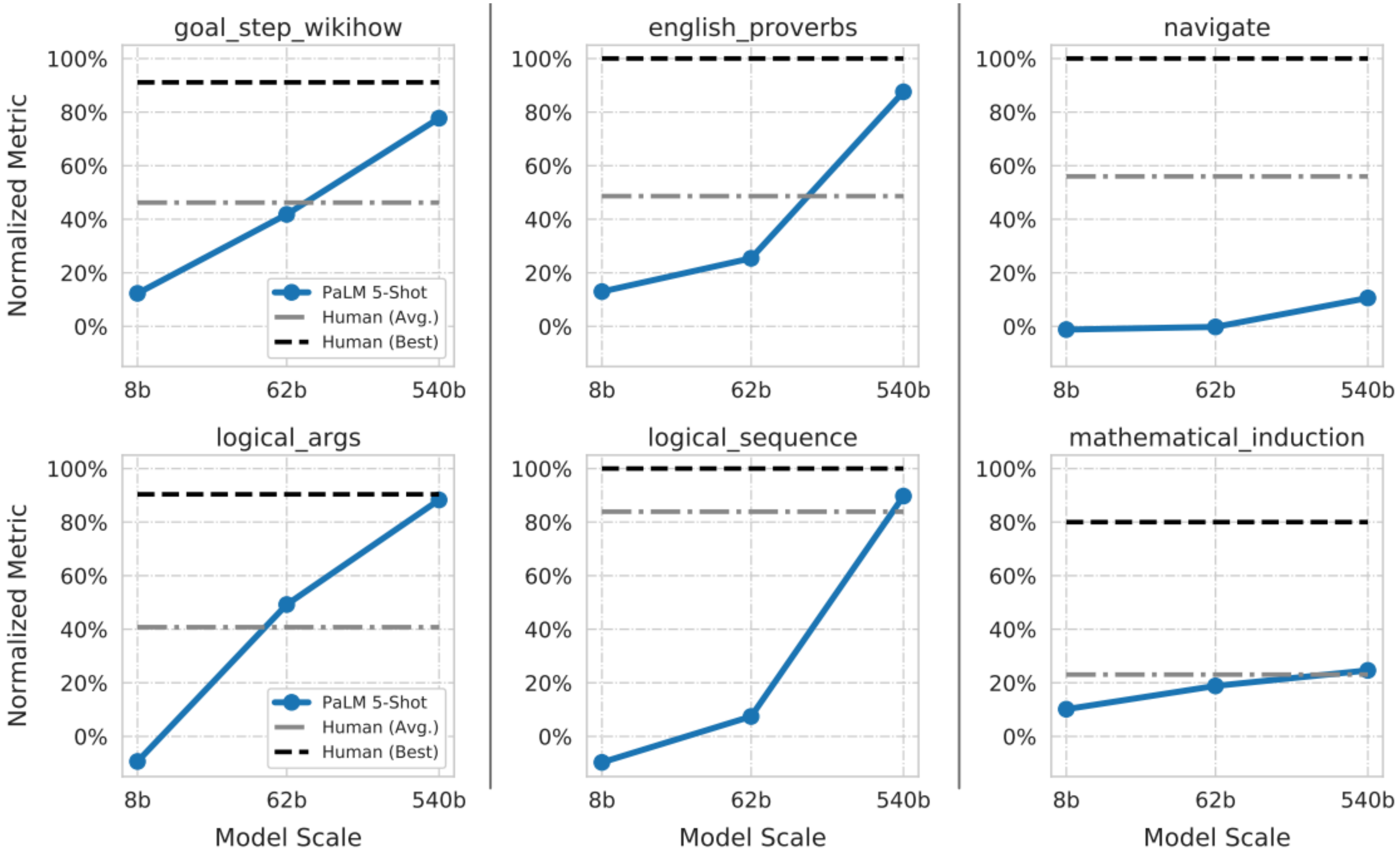
PaLM





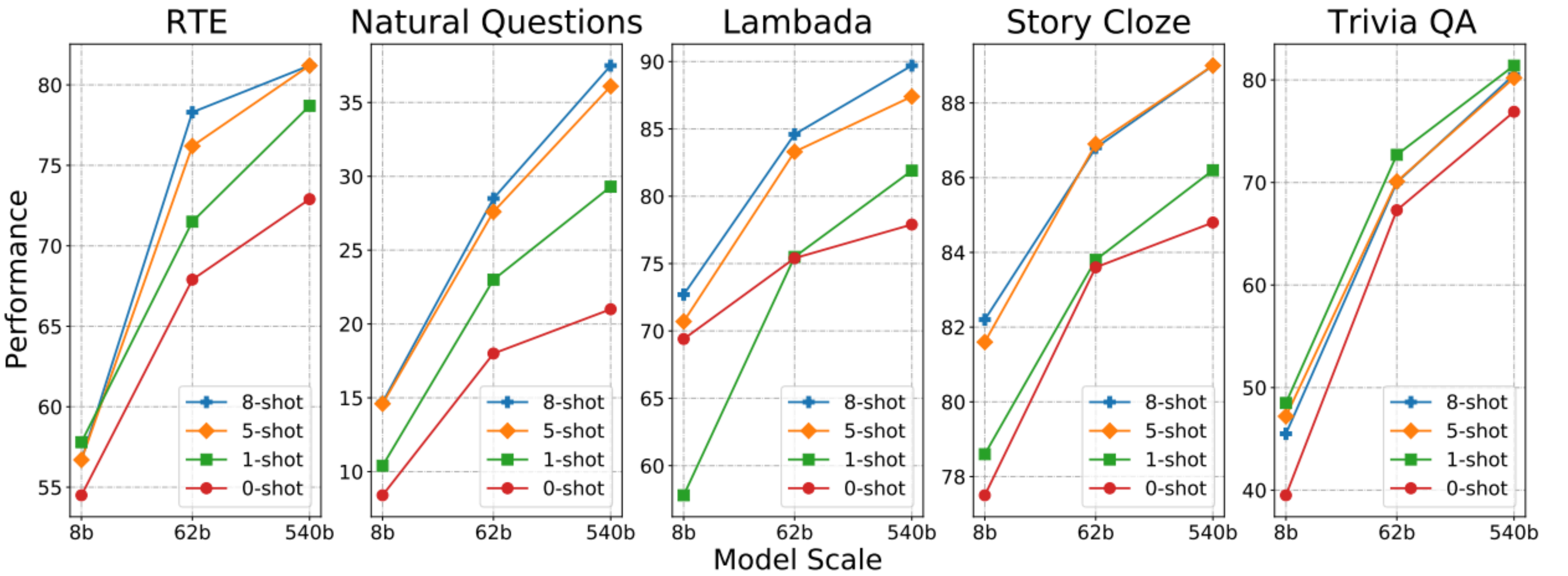
PaLM: Scaling Language Modeling with Pathways, Chowdhery et al, Google, 2022

PaLM



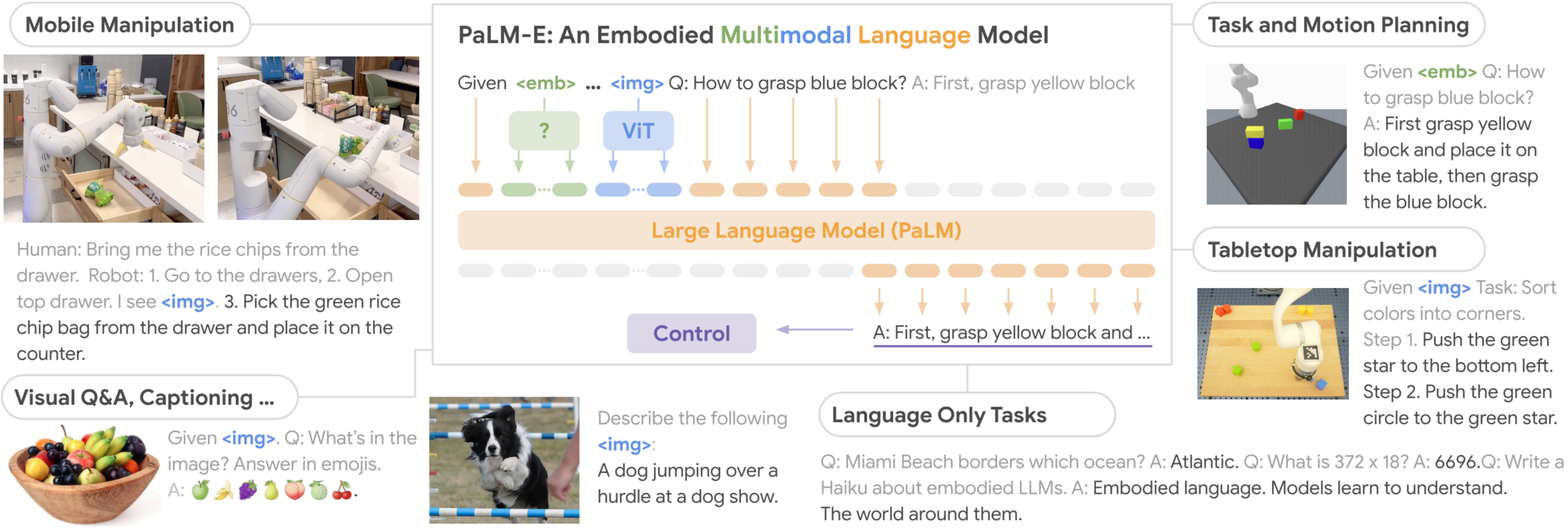
PaLM: Scaling Language Modeling with Pathways, Chowdhery et al, Google, 2022

PaLM



PaLM: Scaling Language Modeling with Pathways, Chowdhery et al, Google, 2022

Toward multimodal agents



PaLM-E: An Embodied Multimodal Language Model [Dreiss et al, Google, 2023]
https://palm-e.github.io/