CSE 493s/599s Lecture 6. Parameter Efficient Fine-tuning

Sewoong Oh

Lecture notes

- These lecture notes are based on other courses in LLMs, including
  - CSE493S/599S at UW by Ludwig Schmidt: https://mlfoundations.github.io/advancedml-sp23/
  - EE-628 at EPFL by Volkan Cevher: https://www.epfl.ch/labs/lions/teaching/ee-628-training-large-language-models/ee-628-slides-2025/
  - ECE381V Generative Models at UT Austin by Sujay Sanghavi
  - and various papers and blogs cited at the end of the slide deck

Outline

- Tokenizers
- Language models
- Architecture
    - Transformers
    - Mixture-of-experts
- Inference
    - Speculative decoding
    - In-context learning
    - Chain-of-thought prompting
    - Test-time compute
- Post-training
    - Parameter Efficient fine-tuning
    - Alignment

Parameter Efficient Fine-Tuning (PEFT)

Fine-tuning

- When given a small data to adopt to a new task domain, there are two ways to use that data
  - **In-context learning**: put the examples in the prompt
  - **Fine-tuning**: optimize (part of) model weights
- Supervised Fine-Tuning (SFT) is a common practice
  - **to adapt** a given base LM to the target domain of interest, given labeled fine-tuning samples.
    - e.g., sentiment analysis, or named-entity classification.
  - It is critical to run Supervised Fine-Tuning (called mid-training) before you run reinforcement learning for **alignment** (called post-training).
- With the increasing scale of LLMs, oftentimes full-scale fine-tuning of the model weights is **prohibitively expensive**.

- With the increasing scale of LLMs, oftentimes full-scale fine-tuning of the model weights is **prohibitively expensive**.
  - For running **inference** on 1B model, you need ~3GB of memory (depends on the sequence length)
    - 1 copy of model weights, activations, KV cache,
  - For running **SFT**, you need ~18GB of memory
    - 1 copy of model weights, gradient, optimizer state (mean, variance), activations

inference memory usage

misc.

model weight W

SFT memory usage

misc.

model weight W

gradient $\nabla_W \ell(f_W(x), y)$

past gradient mean

past gradient variance

- Adam optimizer for SFT

$$W_{t+1} \leftarrow W_t - \frac{\gamma}{\sqrt{v_t} + \epsilon} m_t$$

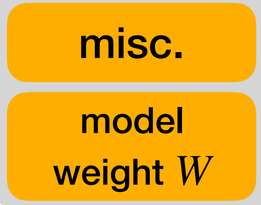
$$g_t$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

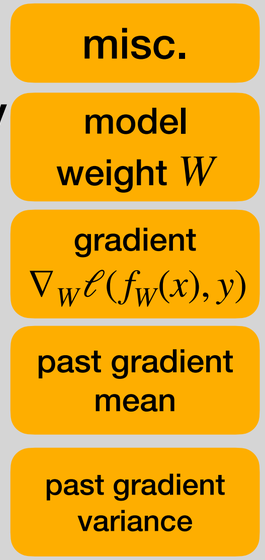
$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- With the increasing scale of LLMs, oftentimes full-scale fine-tuning of the model weights is **prohibitively expensive**.
  - For running **inference** on 1B model, you need ~3GB of memory (depends on the sequence length)
    - 1 copy of model weights, activations, KV cache,
  - For running **SFT**, you need ~18GB of memory
    - 1 copy of model weights, gradient, optimizer state (mean, variance), activations

inference memory usage



SFT memory usage



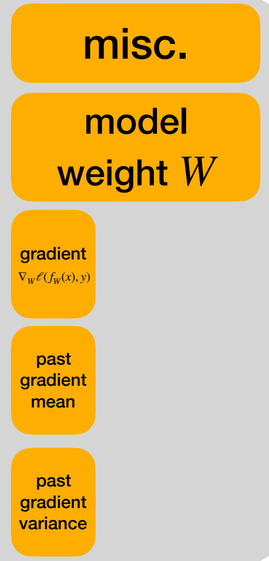
- Adam optimizer for SFT

$$W_{t+1} \leftarrow W_t - \frac{\gamma}{\sqrt{v_t} + \epsilon} m_t$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

PEFT



- **Parameter Efficient Fine-Tuning** (PEFT) corresponds to a family of approaches that freezes most of the parameters of the original pertained network and only trains a small subset of parameters called **adapters**.

Low Rank Adaptation (LoRA)

- Given a layer of pretained weight matrix W, fine-tuning results in an updated weight matrix

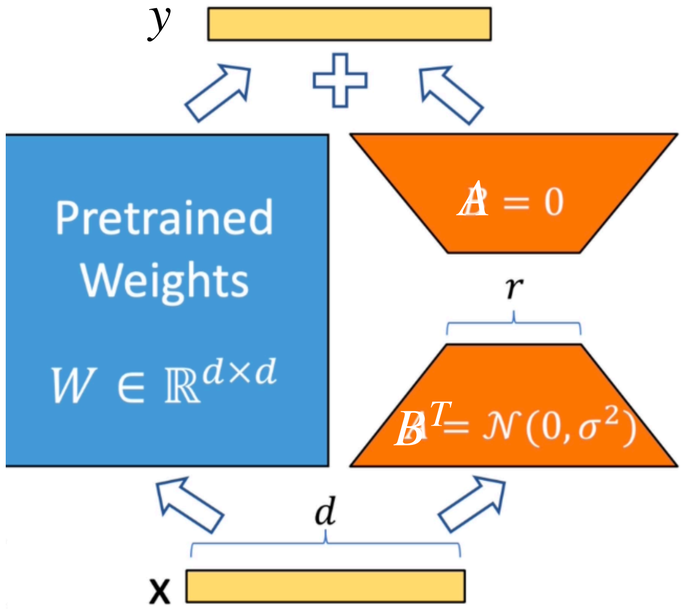
$$W' \leftarrow \underbrace{W}_{\text{pre-trained weight}} + \underbrace{\Delta W}_{\text{fine-tuned update}} .$$

- Instead of an unrestricted, full-rank update ΔW, LoRA parametrizes the update to be low-rank using a bi-linear form:

$$\Delta W = AB^T \text{ such that}$$

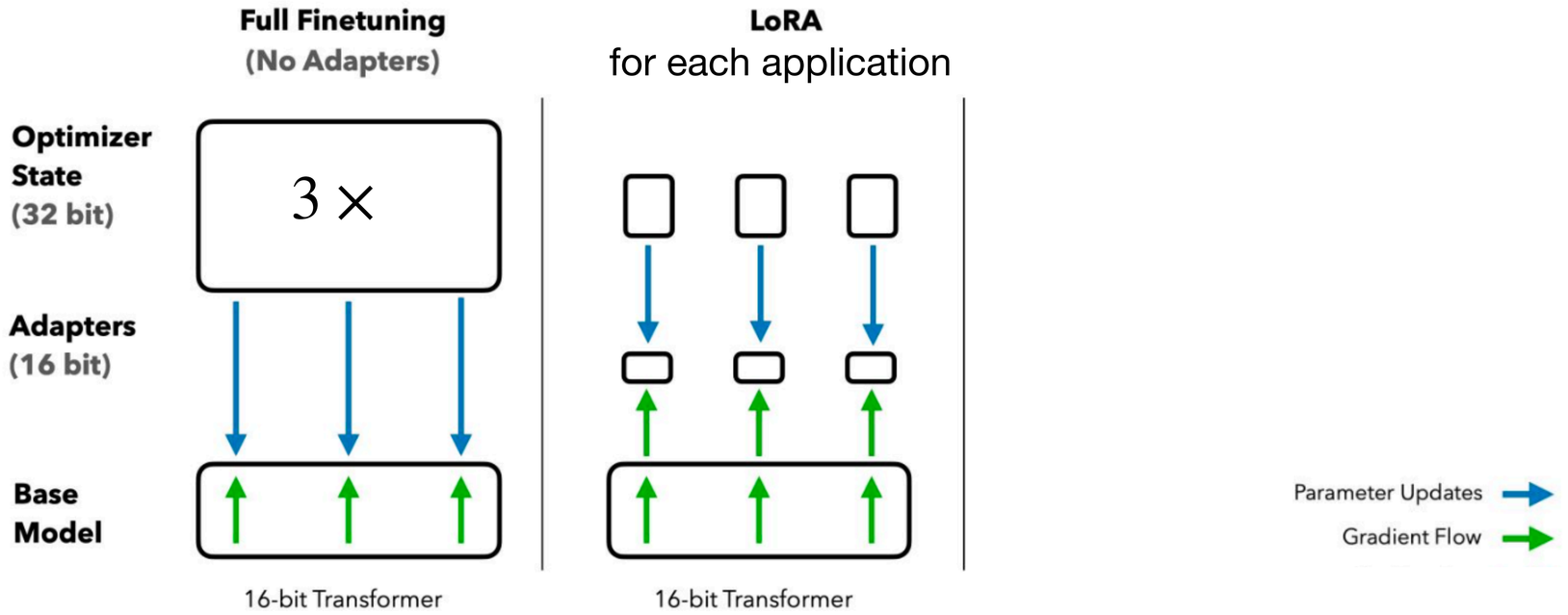
$$W' \leftarrow \underbrace{W}_{\text{this is frozen}} + \underbrace{AB^T}_{\text{this is optimized}} ,$$

where $A \in \mathbb{R}^{d_{\text{out}} \times r}, B \in \mathbb{R}^{d_{\text{in}} \times r}$ are trainable parameters during fine-tuning.



- Frozen
- Learned Adapter

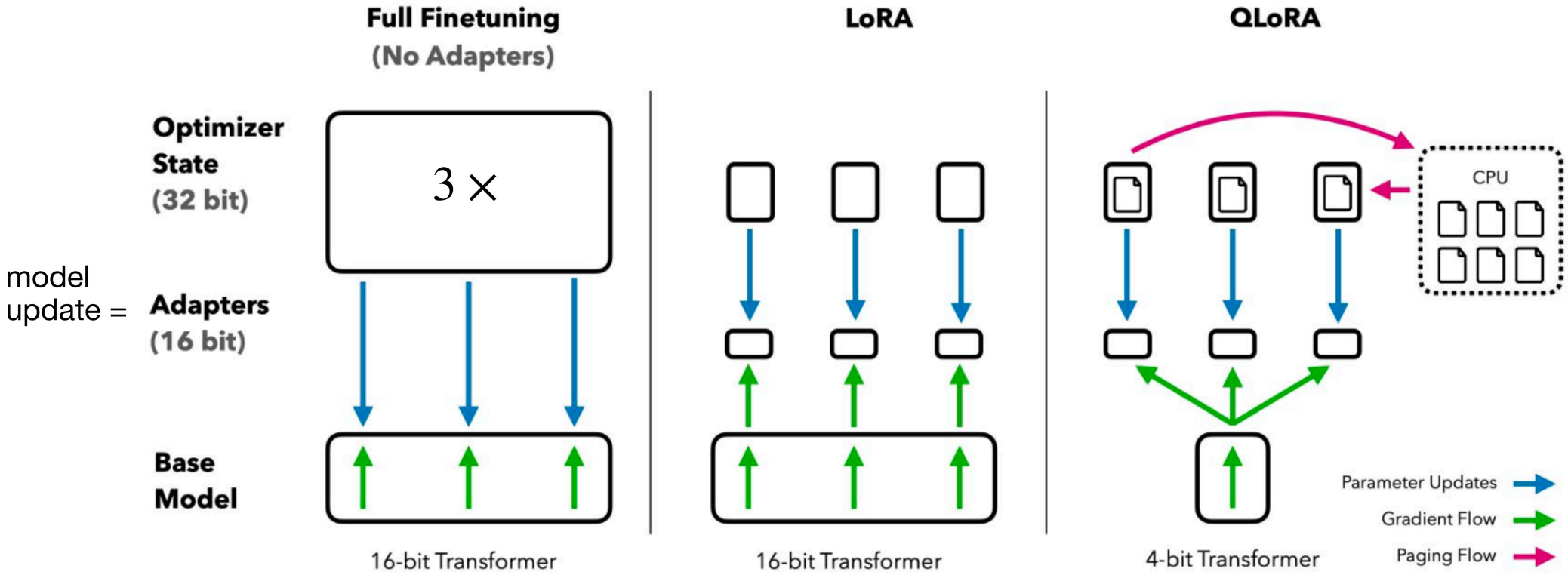
- LoRA Fine-tuning has the following advantages:
  - LoRA requires significantly fewer trainable parameters, requiring reduced **memory** usage: $r \cdot (d_{\text{in}} + d_{\text{out}}) \ll d_{\text{in}} \times d_{\text{out}}$.
  - LoRA avoids making multiple copies of the full parameter, since the base model is frozen.
  - LoRA updates can be applied selectively to specific layers, further reducing **computational** overhead.
  - LoRA updates are **modular**, and can be plugged in and out at inference time.



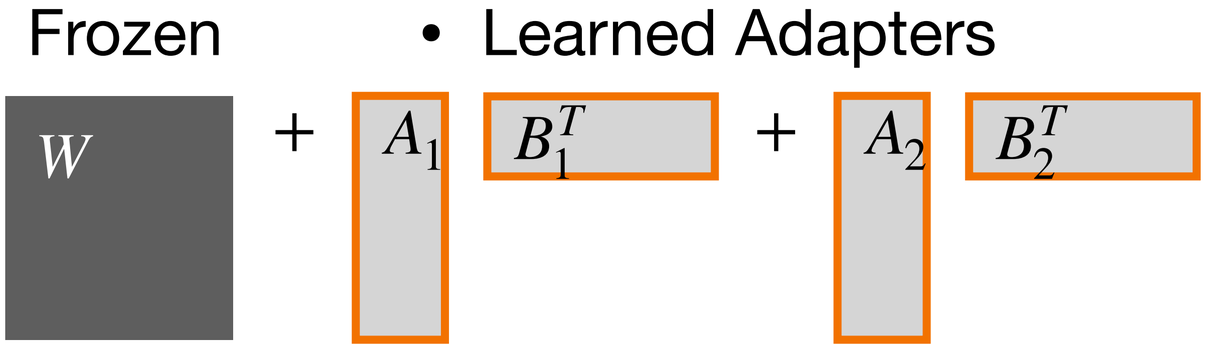
What is the memory bottleneck?

- **QLoRA** [Dettmers et al. 2023] significantly reduces the memory requirement further, by quantizing the frozen base model weights to 4bit precision, while maintaining the fine-tuned performance.

- Average memory requirement for fine-tuning 65B LLM reduces from 780GB of memory to 48GB, which enables fine-tuning on a single GPU.

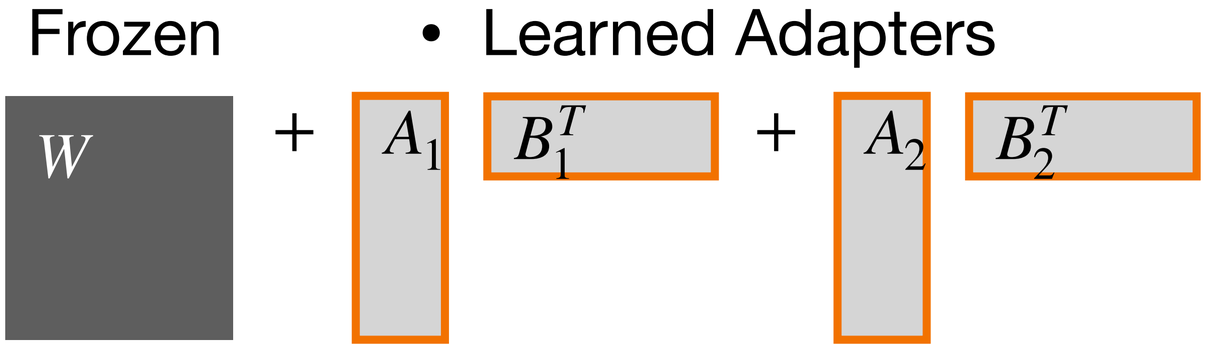


- One down side, is that when we train multiple LoRA adapters on different tasks, say code and math, and **merge** them to get both skills, the **merged adapter** use twice the memory.



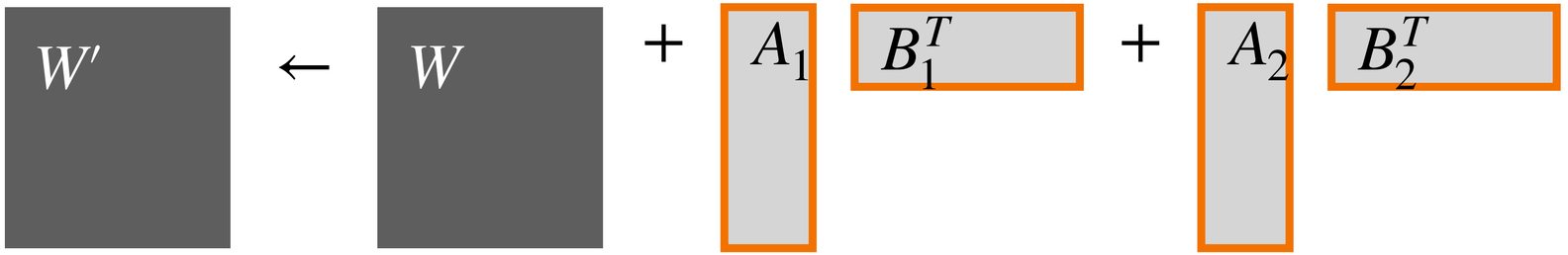
- How do we avoid that?

- One down side, is that when we train multiple LoRA adapters on different tasks, say code and math, and **merge** them to get both skills, the **merged adapter** use twice the memory.

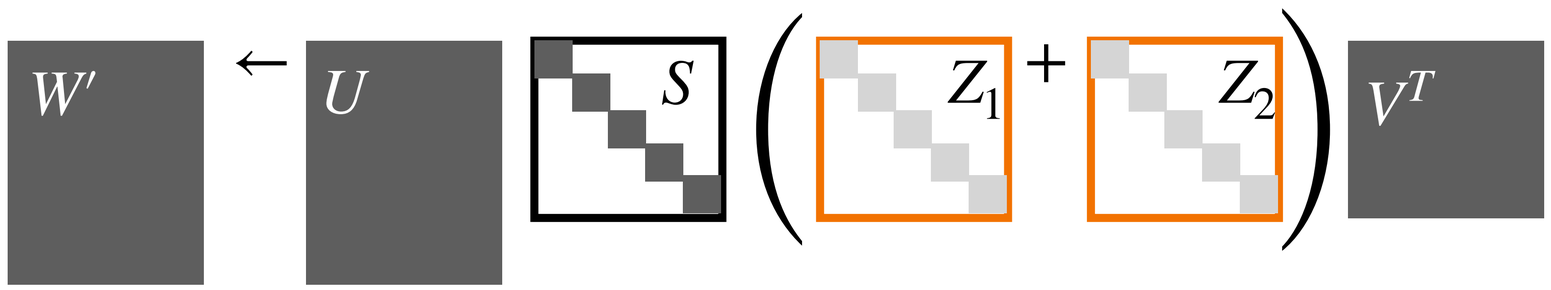


- This parameter increase when merging can be avoided, if the subspaces spanned by A_1 and A_2 are the same and the subspaces spanned by B_1 and B_2 are the same. But, how do we enforce that?
  - **SVF**: Use the subspace of the **SVD(W)**, and train a **diagonal** Z, initialized at I

- One down side, is that when we train multiple LoRA adapters on different tasks, say code and math, and merge them to get both skills, the **merged adapter** use twice the memory.

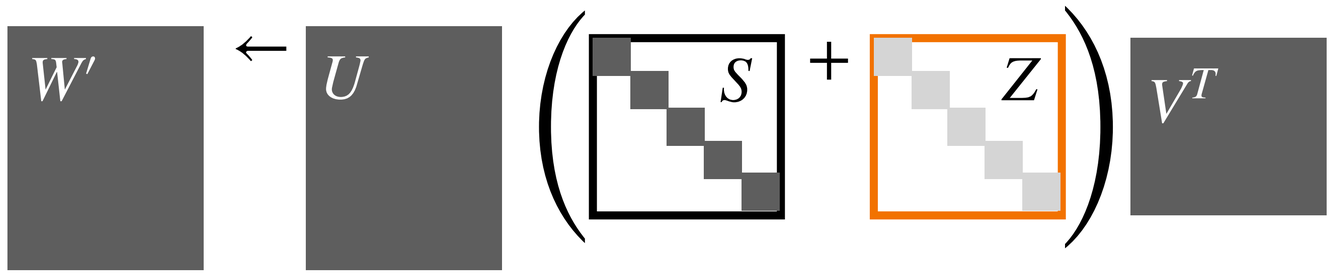


- This parameter increase when merging can be avoided, if the subspaces spanned by A_1 and A_2 are the same and the subspaces spanned by B_1 and B_2 are the same. But, how do we enforce that?
  - SVF: Use the subspace of the SVD(W), and train a diagonal Z, initialized at I
  - When merged, the number of parameters stay the same.

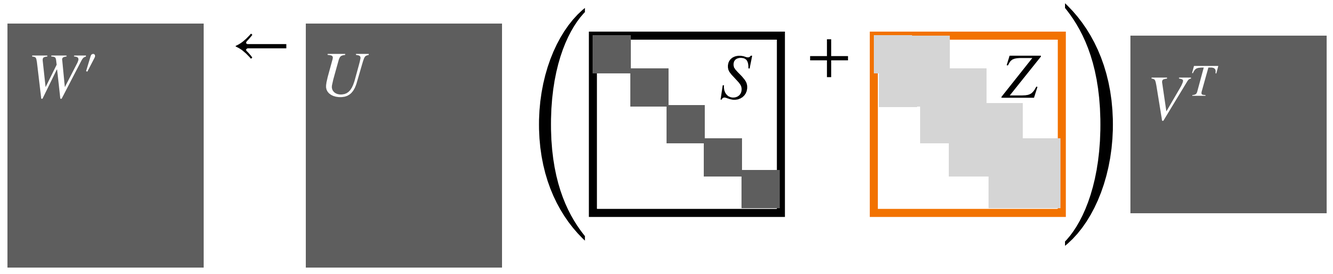


- How do we get a larger size adapter for better SFT performance?

- One can traverse the parameter size vs. downstream accuracy trade-off keeping the SVD subspace frozen:
  - Plain:



  - Banded:



  - Random:

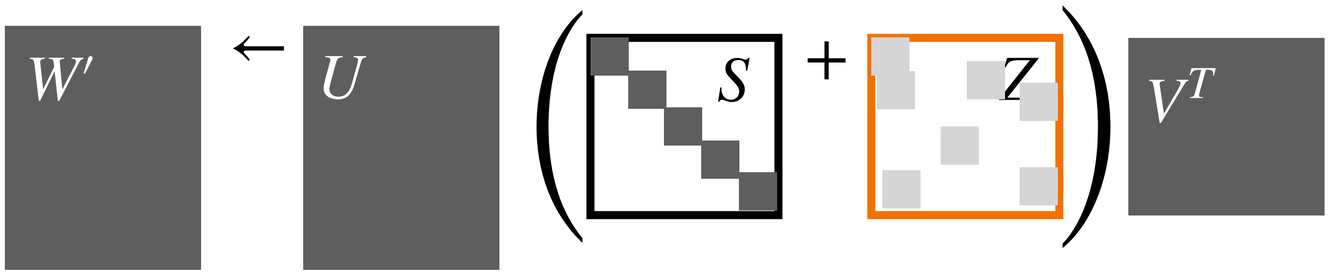
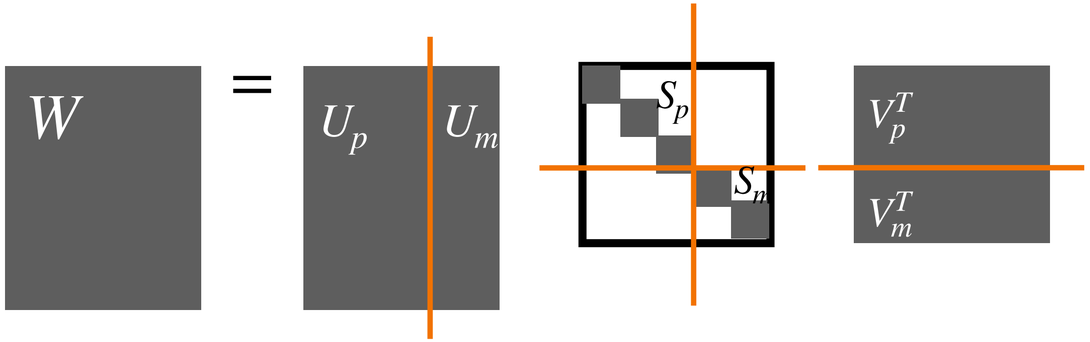


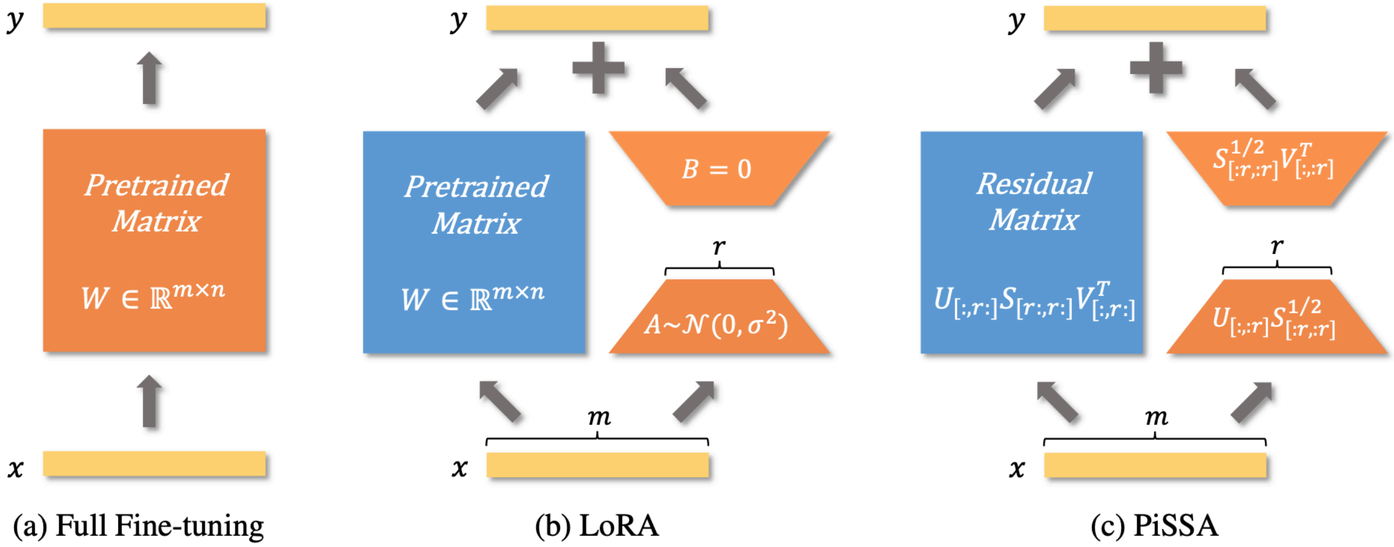
Table 5: Results on fine-tuning Gemma-2B with SVFT using different M parameterizations.

Structure	#Params	GSM-8K	MATH
Plain(diagonal)	0.2M	40.34	14.38
Banded	3.3M	46.47	**16.04**
	6.4M	47.84	15.68
Random	3.3M	47.76	15.98
	6.4M	**50.03**	15.56

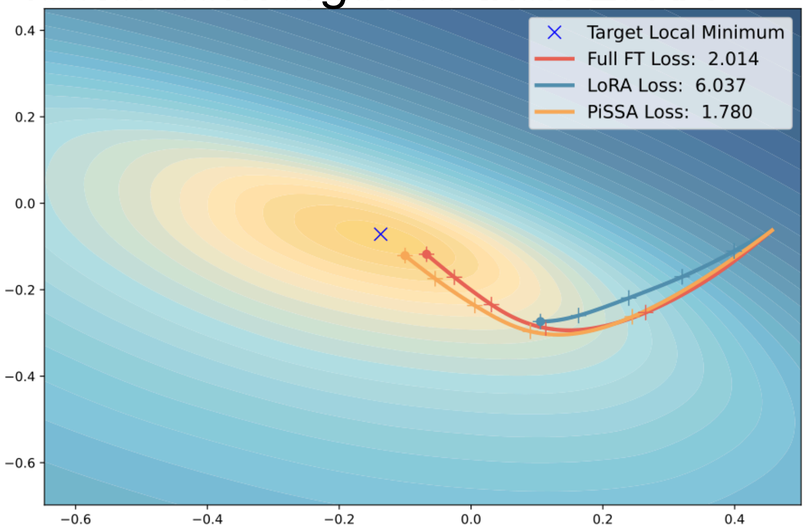
- PISSA (Principal Singular values and Singular vectors Adaptation) [Meng et al. 2024]
  - freezes only the **minor components**, and
  - initializes LoRA with the **principal components**



$$= \underbrace{U_p S_p V_p^T}_{\text{large singular values}} + \underbrace{U_m S_m V_m^T}_{\text{small singular values}}$$



Better initialization leads to training along PCA directions and faster convergence than LoRA



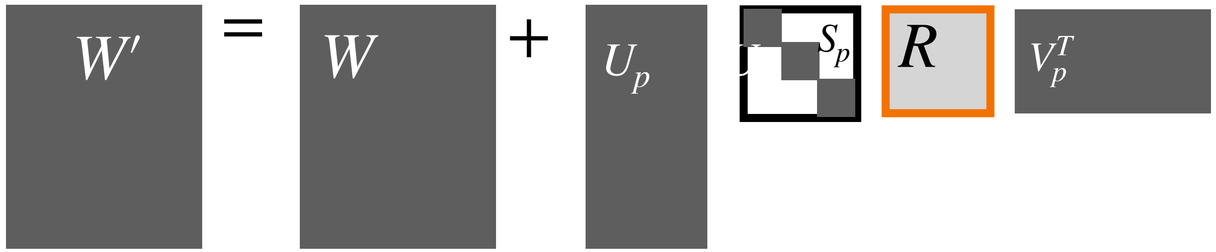
- MiLoRA [Wang et al. 2025] on the other hand
  - freezes only the **principal components**, and
  - initializes LoRA with the **minor components**
- When **the pretrained tasks are aligned with the target task**, the principal components are well-aligned and training the minor components are more effective

PEFT	GSM8K	MATH	Avg.
LoRA	56.6	10.8	33.7
PiSSA	51.3	10.4	30.8
MiLoRA	**58.6**	**11.6**	**35.1**

Table 9: Math reasoning evaluation results for LLaMA2- 7B

- Theoretical analysis of [Li et al. 2025] suggests that initializing with the PCA subspace gives faster convergence.

- LoRA-XS achieves even further parameter reduction than LoRA [Balazy et.al. 2024]



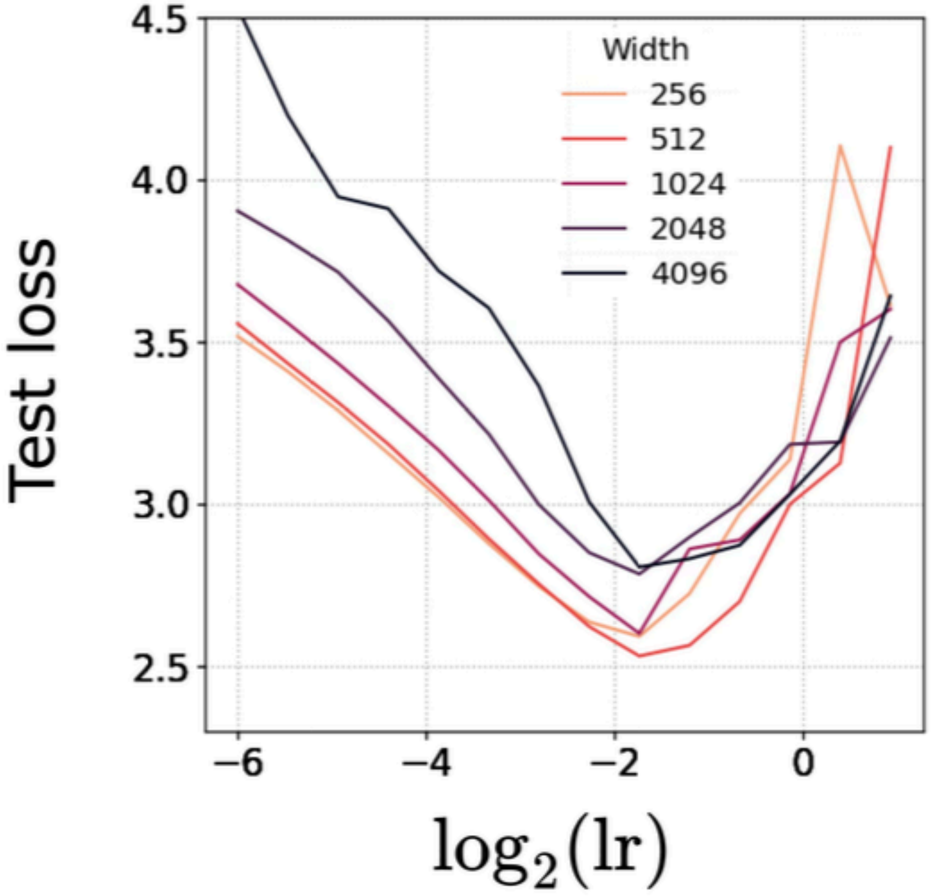
$$W' = W + U_p S_p R V_p^T$$

Model	Method	Rank	# Trainable Parameters	GSM8K	MATH
Mistral (7B)	Full FT	-	7242M	67.02	18.60
	LoRA	64	168M	67.70	19.68
	LoRA-XS	64	0.92M	68.01	17.86
		32	0.23M	63.23	15.88
		16	0.057M	57.92	14.44
Gemma (7B)	Full FT	-	8538M	71.34	22.74
	LoRA	64	200M	74.90	31.28
	LoRA-XS	64	0.80M	74.22	27.62
		32	0.20M	71.72	27.32
		16	0.050M	68.46	26.38

Scaling Laws in LLMs

- **More compute** in pre-training, through larger **models**, more training **data**, and longer **training**, can improve performance
    - if optimal hyperparameters are chosen
- On the other hand, suboptimal combinations of choices can lead to performance degradation with scale
- We need guidance for hyper parameter choices when scaling up the model training
    - for optimal performance , and
    - for predicting that performance

    $\Longrightarrow$ **Scaling Law**



- **Definition**. **Neural Scaling Laws** describe how neural network performance changes as key factors are scaled up or down.
- We consider the following four factors:
  - Size of the model N: number of parameters
  - Size of the training dataset D: number of samples or tokens
  - Compute C: measured in FLOPs (FLoating-point OPerations)
  - Generalization performance L: test loss after training

- In the middle regime of dataset size D, it is conjectured that some power-law governs the scaling

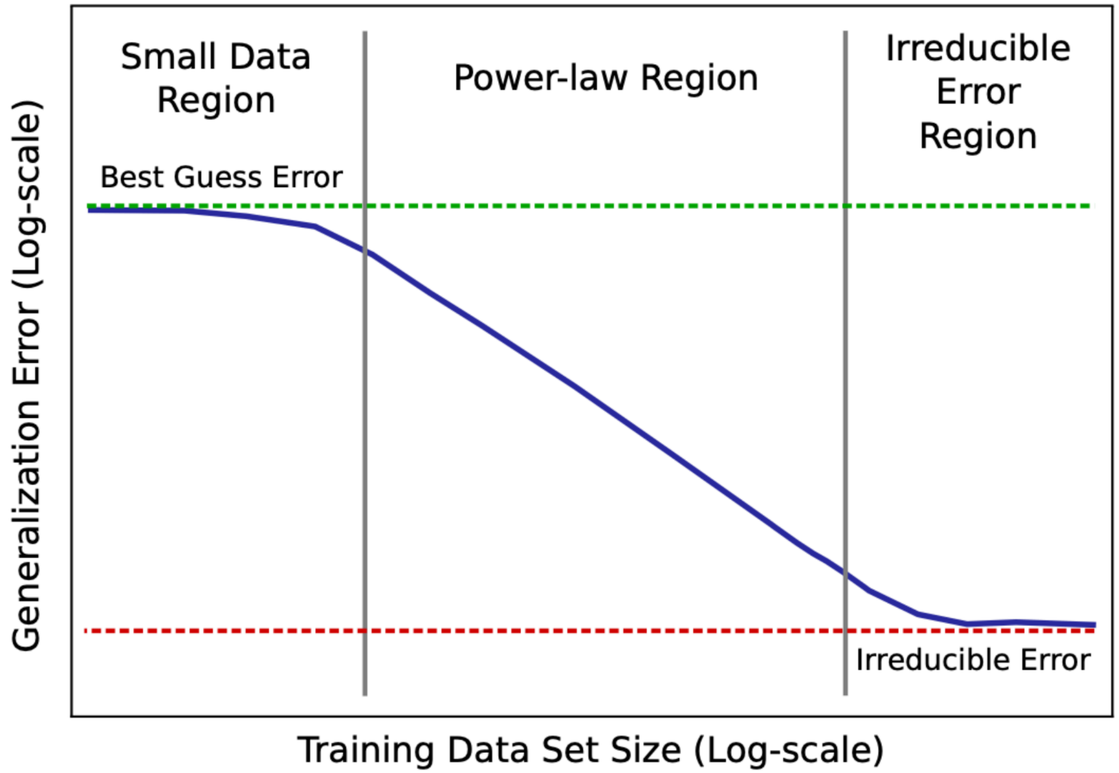


Figure 6: Sketch of power-law learning curves

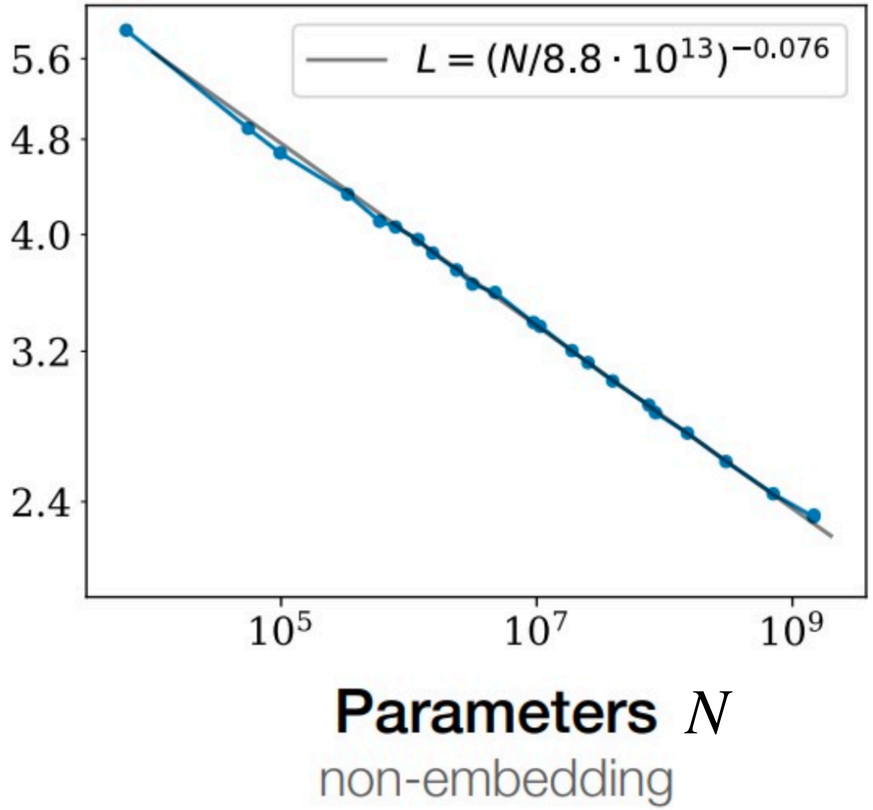
- How do you check for such “power laws”?

- Precisely, Kaplan's (empirical) scaling law [Kaplan et al. 2020] predicts that

  - when **the number of parameters** N is limited, for sufficiently large datasets,

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \text{ where } \alpha_N \sim 0.076, \; N_c \sim 8.8 \times 10^{13} \text{ (parameters)}$$

test loss $L(N)$



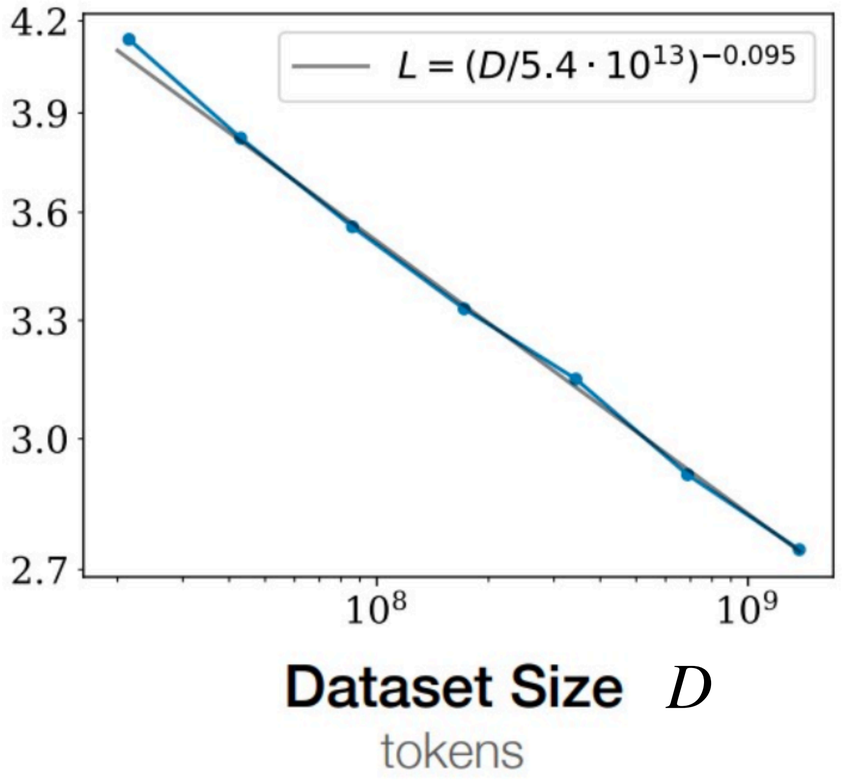
each point on the curve is achieved by a different data size D_N hyperparameter tuned for each of model size N

- Precisely, Kaplan's (empirical) scaling law [Kaplan et al. 2020] predicts that

  - when **the dataset size** D is limited, for sufficiently large models,

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \text{ where } \alpha_N \sim 0.095, \; N_c \sim 5.4 \times 10^{13} \text{ (samples)}$$

test loss $L(D)$

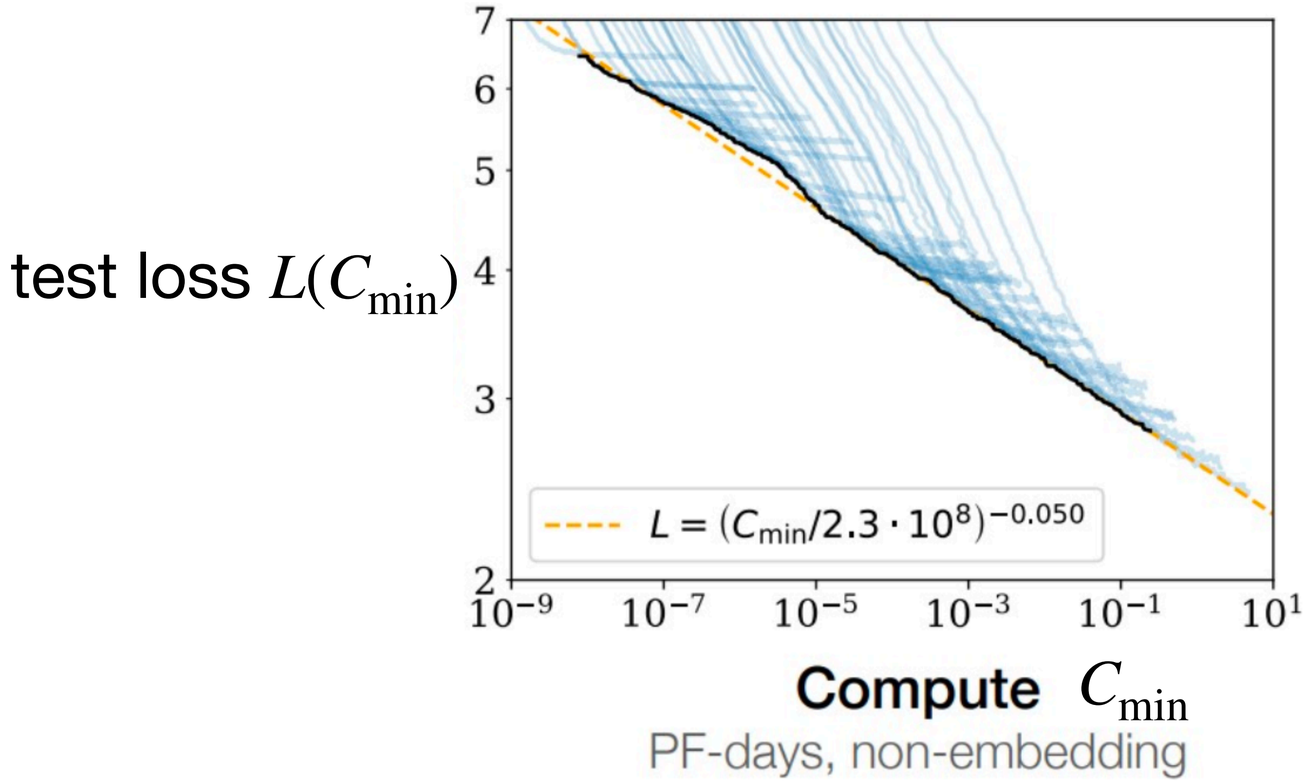


each point on the curve is achieved by a model size N_D hyperparameter tuned for each of dataset size D

- Precisely, Kaplan's (empirical) scaling law [Kaplan et al. 2020] predicts that

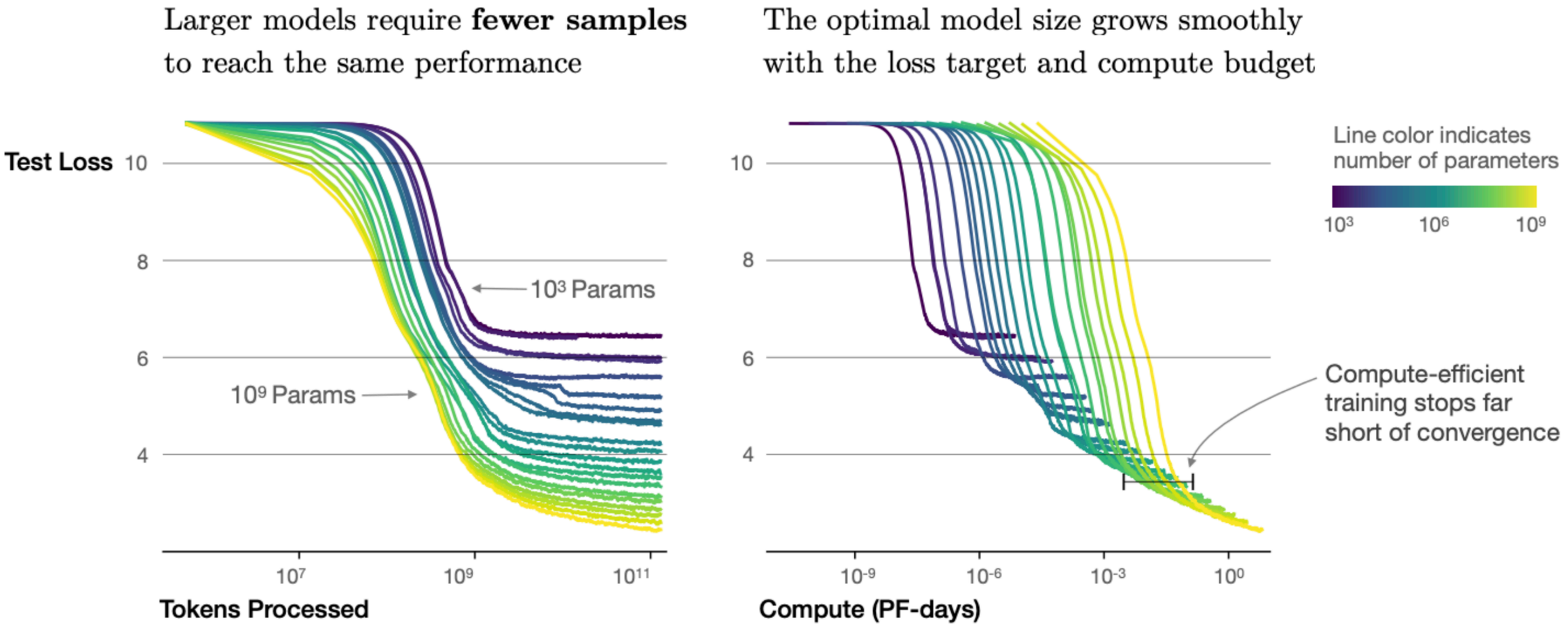
  - when **the compute** C is limited, for sufficiently large dataset, the minimum compute $C_{\min}$ needed to achieve a target test loss scales as

$$L(C_{\min}) = \left(\frac{C_{\min,c}}{C_{\min}}\right)^{\alpha_{C_{\min}}}, \text{ where } \alpha_{C_{\min}} \sim 0.050, \; C_{\min,c} \sim 2.3 \times 10^8 \text{ (PF-days)}$$



- the scaling law is for the pareto frontier (in black solid curve)
- each blue curve is for a fixed model size and changing the dataset size

1 PF-day = 10^{15} FLOPs/second x 24 hours x 3600 seconds/hour



- For optimal performance, all three factors (dataset, compute, model size) need to scale together
- Larger models need fewer samples to achieve the same loss
- Large models are compute-optimal when undertrained
- Train on **larger model with fewer samples** (than training smaller model to convergence)

- Kaplan's scaling law suggests that for fixed compute budget, we should prioritize larger model size

- However, it fails to predict compute optimal scaling in large compute regime, because there is not enough data to train all the parameters

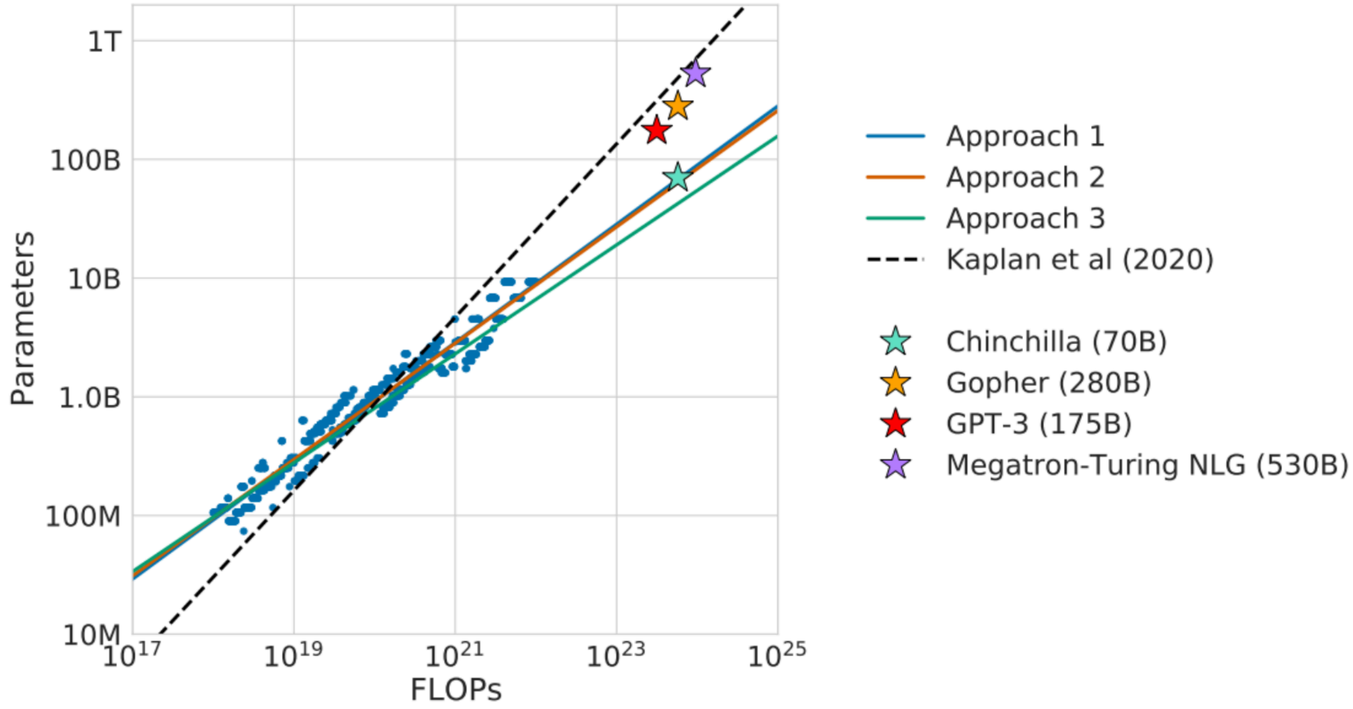


Figure 1 | **Overlaid predictions.** We overlay the predictions from our three different approaches, along with projections from Kaplan et al. (2020). We find that all three methods predict that current large models should be substantially smaller and therefore trained much longer than is currently done. In Figure A3, we show the results with the predicted optimal tokens plotted against the optimal number of parameters for fixed FLOP budgets. ***Chinchilla* outperforms *Gopher* and the other large models (see Section 4.2).**

- Chinchila [Hoffman et al. 2022] scaling law suggests that model size N and data size D should scale together, at the same rate

Table 2 | **Estimated parameter and data scaling with increased training compute.** The listed values are the exponents, a and b, on the relationship $N_{opt} \propto C^a$ and $D_{opt} \propto C^b$. Our analysis suggests a near equal scaling in parameters and data with increasing compute which is in clear contrast to previous work on the scaling of large models. The 10^{th} and 90^{th} percentiles are estimated via bootstrapping data (80% of the dataset is sampled 100 times) and are shown in parenthesis.

Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

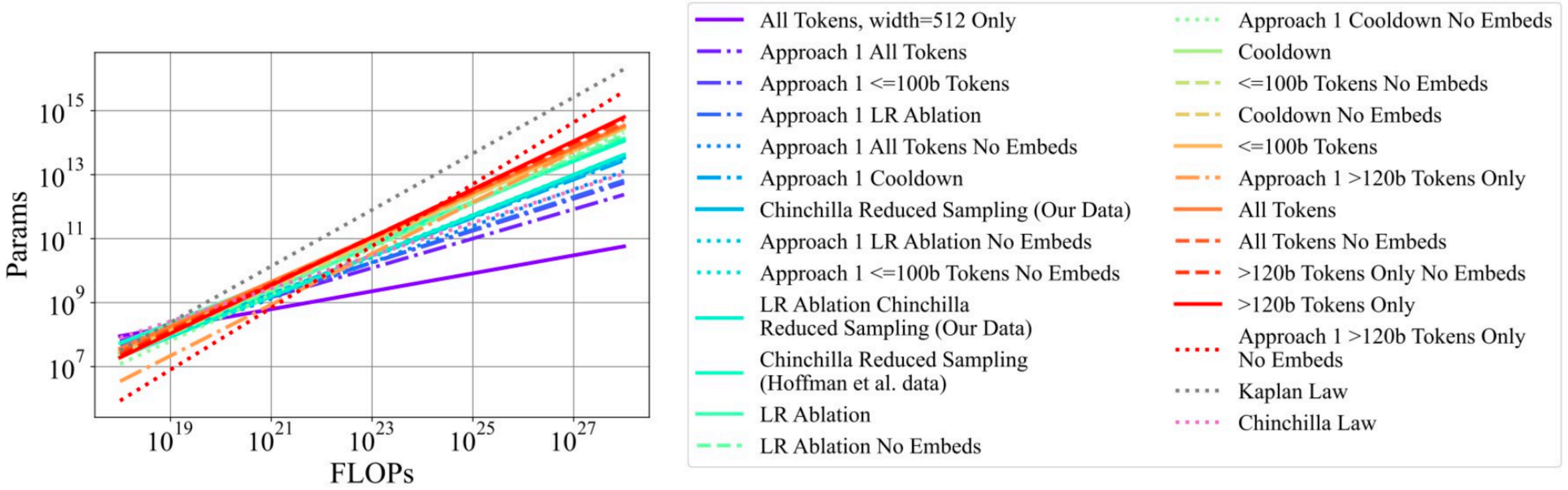
- it was empirically discovered that to scale up compute, N and D need to increase together: 20 training tokens is optimal per parameter

Optimal number of tokens for various sized models

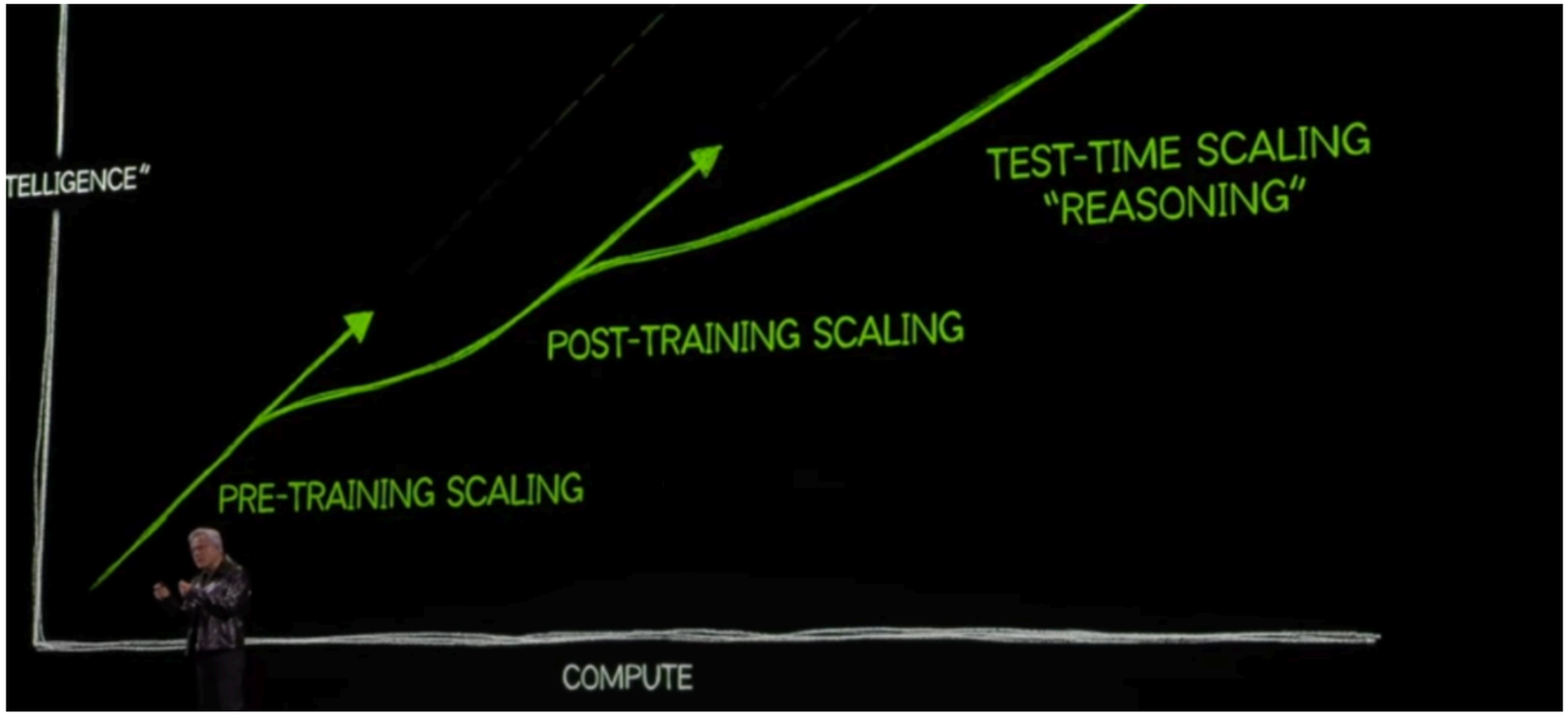
Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion

- Later experiments [McLeish et al. 2025] show that scaling law slope is sensitive to many small experimental design choices, including
  - parameter counting rule: whether to count the embedding layer or not
  - width and depth ratio
  - learning rate scheduler

- Kaplan and Chinchilla extract different scaling laws but neither of them are wrong



- Beyond pretraining scaling laws, there is increasing interest in post-training scaling and test-time scaling. Techniques to improve scaling include
  - Post training: fine-tuning, quantization, pruning, distillation, etc.
  - Test time: best of N sampling, MCMC sampling, thinking time, etc.



Outline

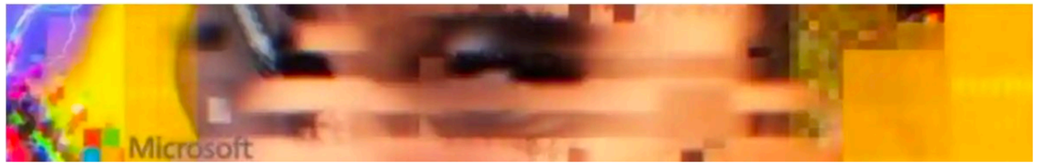
- Tokenizers
- Language models
- Architecture
    - Transformers
    - Mixture-of-experts
- Inference
    - Speculative decoding
    - In-context learning
    - Chain-of-thought prompting
    - Test-time compute
- Post-training
    - Parameter Efficient fine-tuning
    - Alignment

Alignment

- AI model responses can be misaligned with what we want them to do, which can sometimes cause real harm.

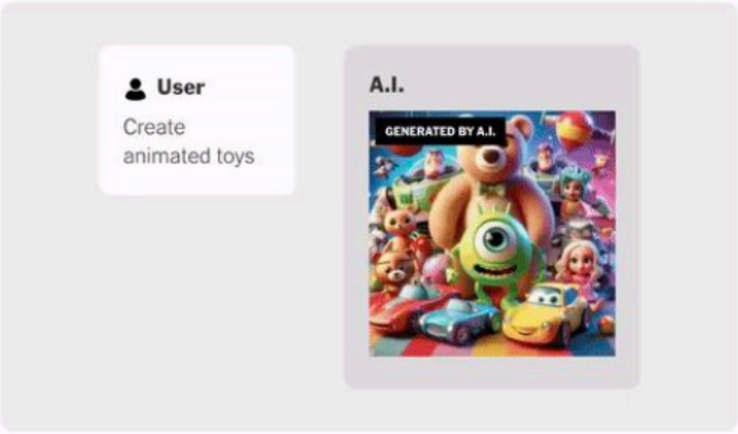
Microsoft 'deeply sorry' for racist and sexist tweets by AI chatbot

Company finally apologises after 'Tay' quickly learned to produce offensive posts, forcing the tech giant to shut it down after just 16 hours



The New York Times

Artificial Intelligence › | A.I. Faces Quiz | How the A.I. Race Began | Key Figures in the Field | One Year of ChatGPT



We Asked A.I. to Create the Joker. It Generated a Copyrighted Image.

By Stuart A. Thompson Jan. 25, 2024

The Washington Post
Democracy Dies in Darkness

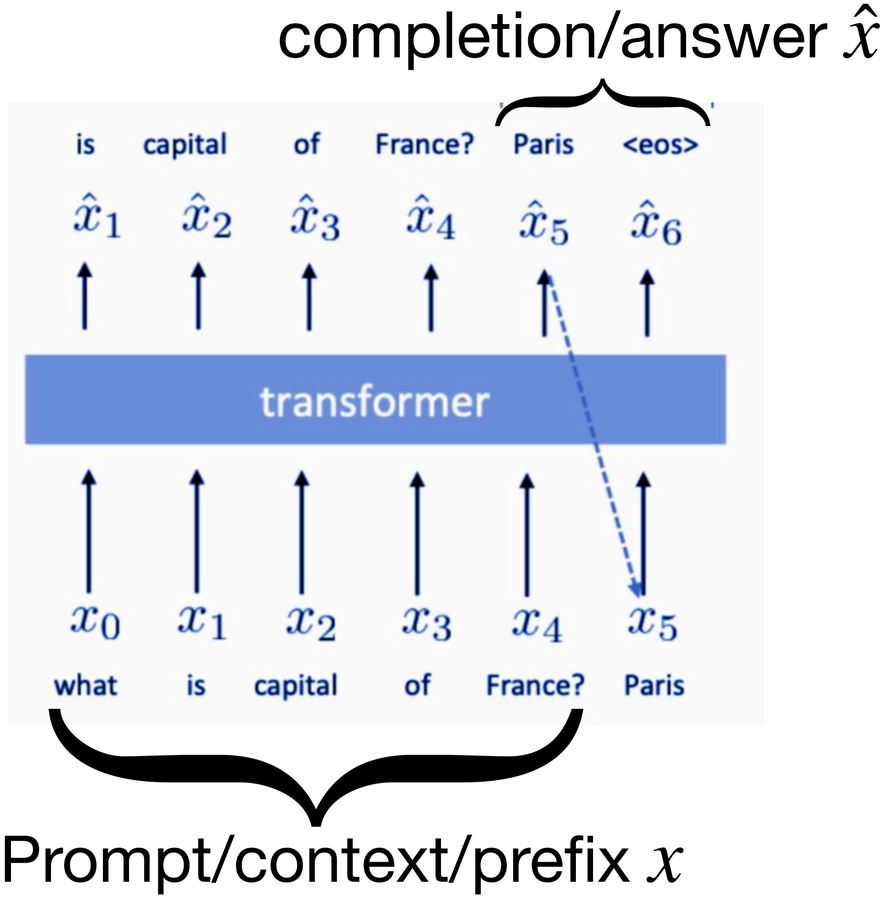
Chatbots' inaccurate, misleading responses about U.S. elections threaten to keep voters from polls

By Garance Burke | AP

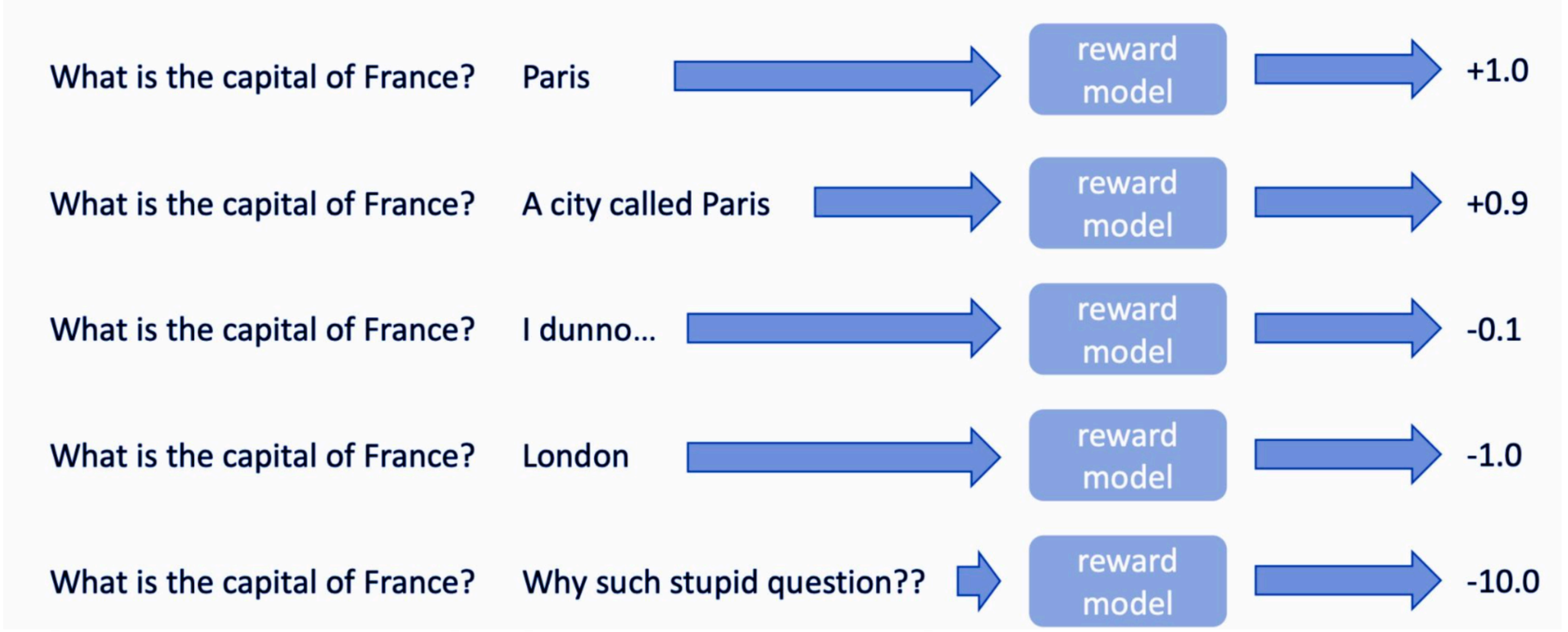
- Scaling pretaining does not address the challenges in **alignment** with human values and intent. We need post-training based on **RLHF (Reinforcement Learning with Human Feedback)**.

- We do not cover reinforcement learning in this class in any depths, but we will learn as much as we need along the way. There are other amazing classes by Kevin Jamieson and Natasha Jacques.

- Given a prompt/context/prefix x and its completion $\hat{x}$, a **reward model** assigns a scalar value on the quality of the completion: $r(x, \hat{x}) \in \mathbb{R}$

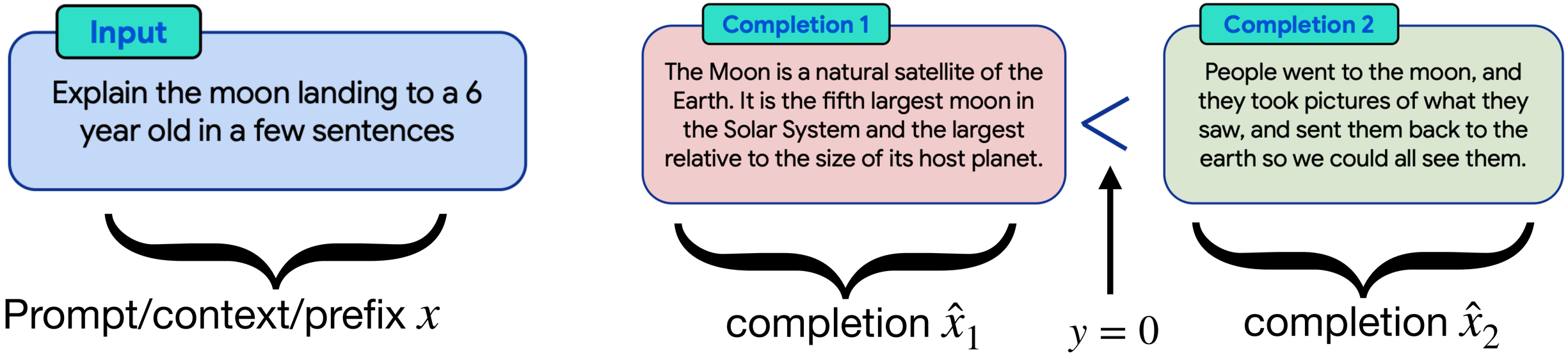


- consider a neural network function $r(x, \hat{x})$, that we want to train on **human labelled preference**, such that it provides useful “reward”



- However, it is challenging to collect reliable reward, so instead we collect data consisting of pairwise comparisons on which completion is preferred. This does not fit questions where exact solutions exist, like math and coding, where we use RLVF (Reinforcement Learning with Verifiable Feedback) instead.

- Each sample $(x, \hat{x}_1, \hat{x}_2, y)$ consists of prompt x, two different completions $\hat{x}_1$ and $\hat{x}_2$, and preference label $y = 1$ if $\hat{x}_1$ is preferred over $\hat{x}_2$, which we write as $\hat{x}_1 \succ \hat{x}_2$, and $y = 0$, otherwise.



- To model such preferences so that we can learn the model, we borrow mathematical foundations of **choice models,** in particular, **Random Utility Models (RUMs)**.
- Under RUM, each option has corresponding utility, and when we make a choice, we observe a randomly perturbed utility and choose the one that has maximum observed utility. In the context of LLM post-training, the completions are our options to choose from and utility of an option is the reward of a completion.
  - **Random Utility Model**
    - hidden true rewards of the two completions: $r^*(x, \hat{x}_1)$ and $r^*(x, \hat{x}_2)$
    - observed rewards: $r^*(x, \hat{x}_1) + z_1$ and $r^*(x, \hat{x}_2) + z_2$
    - preference: $\mathbb{P}(\hat{x}_1 \succ \hat{x}_2) = \mathbb{P}(r^*(x, \hat{x}_1) + z_1 > r^*(x, \hat{x}_2) + z_2)$
  - Different choices of the noise gives different models. When the noise z_i's follow independent **Gumbel distribution**, the resulting distribution of the preference simplifies to a logistic distribution, which is called **Bradley-Terry model**.

- **Bradley-Terry (BT) model** or Bradley-Terry-Luce (BTL) model
  - hidden true rewards of the two completions: $r^*(x, \hat{x}_1)$ and $r^*(x, \hat{x}_2)$
  - observed rewards: $r^*(x, \hat{x}_1) + z_1$ and $r^*(x, \hat{x}_2) + z_2$
  - **preference**:

$$\begin{aligned}\mathbb{P}(\hat{x}_1 \succ \hat{x}_2) &= \mathbb{P}(r^*(x, \hat{x}_1) + z_1 \geq r^*(x, \hat{x}_2) + z_2) \\ &= \frac{1}{1 + \exp\{-(r^*(x, \hat{x}_1) - r^*(x, \hat{x}_2))\}}\end{aligned}$$

- Learning a NN reward function $r(\cdot, \cdot)$ given data $\mathcal{D} = \{(x_i, \hat{x}_{i,\text{win}}, \hat{x}_{i,\text{lose}})\}$:

$$\max_r \sum_{(x_i, \hat{x}_{i,\text{win}}, \hat{x}_{i,\text{lose}})} \log \frac{1}{1 + \exp\{-(r(x_i, \hat{x}_{i,\text{win}}) - r(x_i, \hat{x}_{i,\text{lose}}))\}}$$

Sources

- Other courses in LLMs that the lecture slides are based on
  - CSE493S/599S at UW by Ludwig Schmidt: https://mlfoundations.github.io/advancedml-sp23/
  - EE-628 at EPFL by Volkan Cevher: https://www.epfl.ch/labs/lions/teaching/ee-628-training-large-language-models/ee-628-slides-2025/
  - https://sharif-llm.ir/assets/lectures/Chain-of-Thought-Prompting.pdf
  - https://www.cs.princeton.edu/courses/archive/fall22/cos597G/lectures/lec09.pdf
- Useful blog posts
  - https://azizbelaweid.substack.com/p/complete-summary-of-absolute-relative
  - https://blog.dust.tt/speculative-sampling-llms-writing-a-lot-faster-using-other-llms/
  - https://gordicaleksa.medium.com/eli5-flash-attention-5c44017022ad
  - https://medium.com/@dillipprasad60/qlora-explained-a-deep-dive-into-parametric-efficient-fine-tuning-in-large-language-models-llms-c1a4794b1766
- Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd ed. draft). draft, third edition, 2023.
- Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”, In International Conference on Learning Representations, 2013.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation”, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
- Ofir Press, Noah A. Smith1,3 Mike Lewis2 , “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”, In International Conference on Learning Representations, 2022
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need”, In Neural Information Processing Systems, 2017
- Beitong Zhou, Cheng Cheng, Guijun Ma, and Yong Zhang. “Remaining useful life prediction of lithium-ion battery based on attention mechanism with positional encoding”, In IOP Conference Series: Materials Science and Engineering, 2020.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks." In International Conference on Machine Learning, 2013

- Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In Neural Computation, 9(8):1735–1780, 11 1997.
- Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, In ACL 2014
- Andrey Andreyevich Markov. “Essai d’une recherche statistique sur le texte du roman. ‘Eugene Onegin’ illustrant la liaison des epreuve en chain”. In: Izvistia Imperatorskoi Akademii Nauk (Bulletin de l’Académie Impériale des Sciences de St.-Pétersbourg). 6th ser, 7:153–162, 1913.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi, “The Curious Case of Neural Text Degeneration”, In *International Conference on Learning Representations*, 2020
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre and John Jumper, “Accelerating Large Language Model Decoding with Speculative Sampling”In, *ACL-findings*, 2024
- Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", In *International Conference on Machine Learning*, 2015
- Shibani Santurkar∗ MIT shibani@mit.edu Dimitris Tsipras∗ MIT tsipras@mit.edu Andrew Ilyas∗ MIT ailyas@mit.edu Aleksander Madry, “How Does Batch Normalization Help Optimization?”, In *Advances in Neural Information Processing Systems* 31 (NeurIPS 2018)
- Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton, “Layer Normalization “, In 2016
- Tianyu Gao, Adam Fisch, Danqi Chen, “Making Pre-trained Language Models Better Few-shot Learners”, In *ACL*, 2021
- Sewon Min1,2 Xinxi Lyu1 Ari Holtzman1 Mikel Artetxe2 Mike Lewis2 Hannaneh Hajishirzi1,3 Luke Zettlemoyer, “rethinking the role of demonstrations what makes in conte…”
- Hila Gonen1,2 Srini Iyer2 Terra Blevins1 Noah A. Smith1,3 Luke Zettlemoyer1, “Demystifying Prompts in Language Models via Perplexity Estimation”
- E Akyürek, B Wang, Y Kim, J Andreas , “In-context language learning: Architectures and algorithms”, 2024
- What learning algorithm is in-context learning? Investigations with linear models Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, Denny Zhou, 2022
- Ziqian Lin, Kangwook Lee, “Dual Operating Modes of In-Context Learning”, 2024
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, In NeurIPS 2022
- Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, Yusuke Iwasawa, “Large Language Models are Zero-Shot Reasoners”, In NeurIPS 2022
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, Denny Zhou, “Self-Consistency Improves Chain of Thought Reasoning in Language Models”, In ICLR 2023

- Shunyu Yao · Jeffrey Zhao · Dian Yu · Nan Du · Izhak Shafran · Karthik Narasimhan, Yuan Cao, “ReAct: Synergizing Reasoning and Acting in Language Models”, In ICLR 2025
- Satyapriya Krishna1, Kalpesh Krishna2, Anhad Mohananey†2, Steven Schwarcz2, Adam Stambler2, Shyam Upadhyay2, Manaal Faruqui*3 , “Fact, Fetch, and Reason: A Unified Evaluation of Retrieval-Augmented Generation“
- Salaheddin Alzubi, Creston Brooks, Purva Chiniya, Edoardo Contente, Chiara von Gerlach, Lucas Irwin, Yihan Jiang, Arda Kaz, Windsor Nguyen, Sewoong Oh, Himanshu Tyagi, Pramod Viswanath, “Open Deep Search: Democratizing Search with Open-source Reasoning Agents“, https://arxiv.org/abs/2503.20201
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, Heng Ji, “Executable Code Actions Elicit Better LLM Agents”, In *ICML* 2024
- Peter Shaw, Jakob Uszkoreit Ashish Vaswani, “Self-Attention with Relative Position Representations”, 2018
- Ofir Press1,2 Noah A. Smith1,3 Mike Lewis2, “TRAIN SHORT, TEST LONG: ATTENTION WITH LINEAR BIASES ENABLES INPUT LENGTH EXTRAPOLATION”, 2022
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs”, In NeurIPS 2023
- Fanxu Meng1,2, Zhaohui Wang1, Muhan Zhang1,2∗, “PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models” In *NeurIPS* 2024
- Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, Yun Chen, “MiLoRA: Harnessing Minor Singular Components for Parameter-Efficient LLM Finetuning”, In *NAACL* 2025
- Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, Jacek Tabor, "LoRA-XS: Low-Rank Adaptation with Extremely Small Number of Parameters”
- Bingcong Li, Liang Zhang, Aryan Mokhtari, Niao He, “On the Crucial Role of Initialization for Matrix Factorization.”, In *ICLR* 2025
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. “Scaling laws for neural language models”, 2020.
- 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, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022.

- Sean McLeish, John Kirchenbauer, David Yu Miller, Siddharth Singh, Abhinav Bhatele, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. “Gemstones: A model suite for multi-faceted scaling laws”, 2025