Scaling

CSE 5539: Advanced Topics in Natural Language Processing

https://shocheen.github.io/courses/advanced-nlp-fall-2024

Logistics

- Project proposal: How did it go?
  - You'll receive (informal) feedback by Wednesday.

Logistics

- OSC Access: Any issues, questions?

Logistics

- Optional Assignment:
  - Feedback on your presentation (content / skills)
    - Your task: Write a self-review
    - My task: I will suggest improvements (if applicable)(
  - Feedback on the class
    - What you like, what you don’t, format, content, anything else: this will be anonymized.
    - I will implement things that can be immediately addressed.

Today’s goal

Bigger is better?

Part I: Scaling the model (parameter, data, compute)

Part II: Scaling at inference time

Stakeholder

Training Compute-Optimal Large Language Models

Jordan Hoffman et al., 2023

By Mona Gandhi

Motivation

- Size of large dense transformers → constantly rising
- Challenges:
  - overwhelming computational requirements
  - acquiring high-quality data

Need to understand **HOW** to scale these models?

Motivation

Kaplan et al [1]

→ Predictable relationship b/w model size and loss
→ Models should not be trained to their lowest point to be compute optimal
→ When computational budget - 10x, suggests model size - 5.5x and training tokens - 1.8x

Issues:

- Use fixed number of training tokens and learning rate for all models
- Does not consider the effect of training tokens

[1] Scaling laws for neural language models, Kaplan et al., 2020

Overview

Fewer params with same compute and better performance!

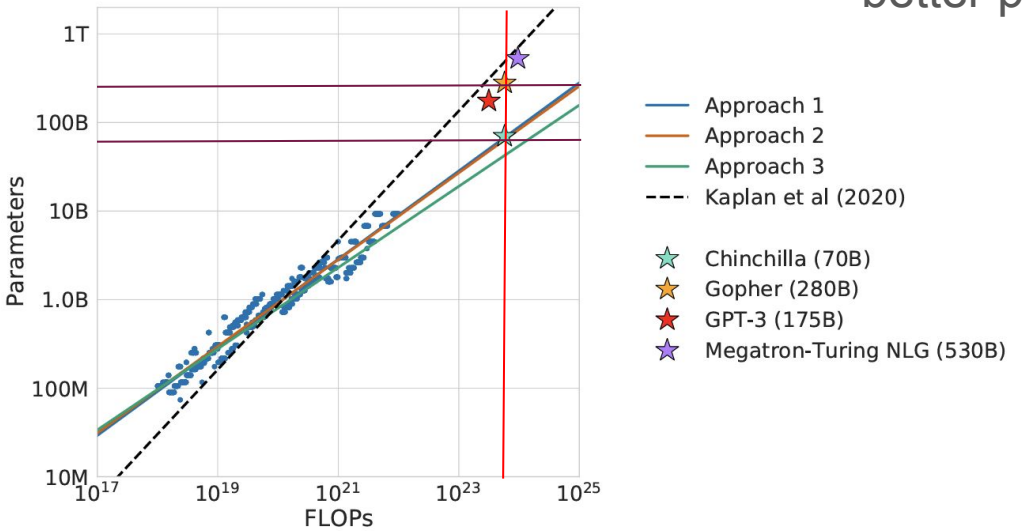


Figure 1 | **Overlaid predictions.** We overlay the predictions from our three different approaches, along with projections from Kaplan et al. (2020). We find that all three methods predict that current large models should be substantially smaller and therefore trained much longer than is currently done. 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).**

Methods

Given a fixed FLOP budget, how should one tradeoff model size and the number of training tokens?

- Start by training a range of models varying in both model size and training tokens
- And thus use the resulting training curves to fit an empirical estimator of how they should scale

Approach 1: Fix model sizes and vary number of training tokens

Vary the number of training tokens for a family of models (70M to 10B params), training each model for 4 different training sequences.

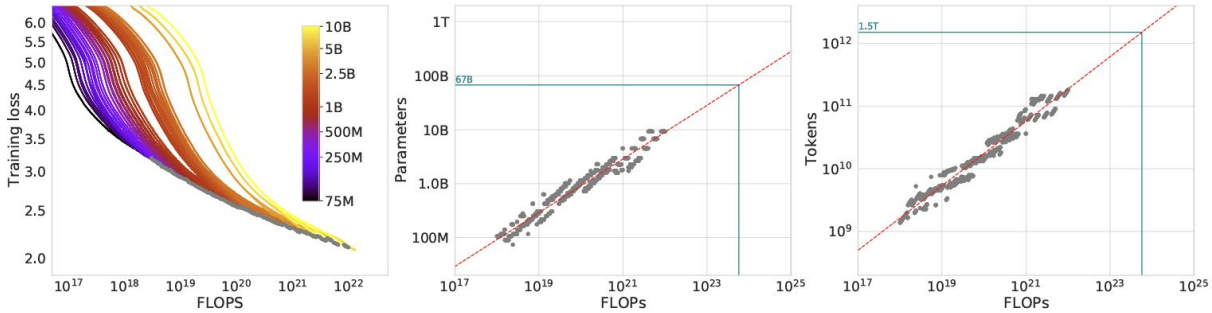


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Approach 2: IsoFLOP profiles

Vary the model size for a fixed set of 9 different training FLOPs (6×10^{18} to 3×10^{21}) and consider final training loss for each point.

Answers the question: *For a given FLOP budget, what is the optimal param count?*

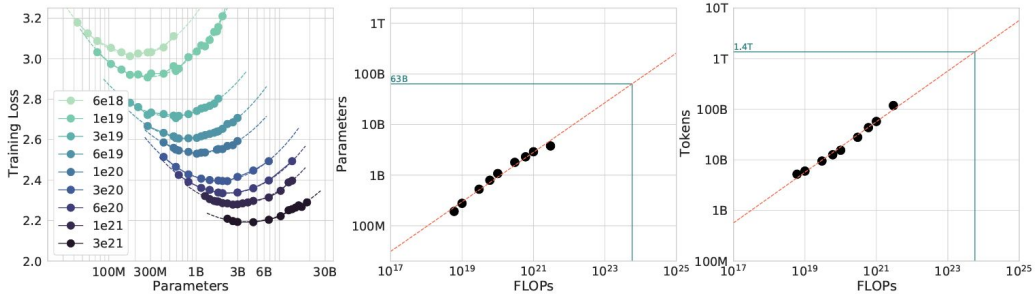


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**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*.

Approach 3: Fitting a parametric loss function

Model all final losses from Approach 1 and 2 as a parametric function of model param count and training tokens.

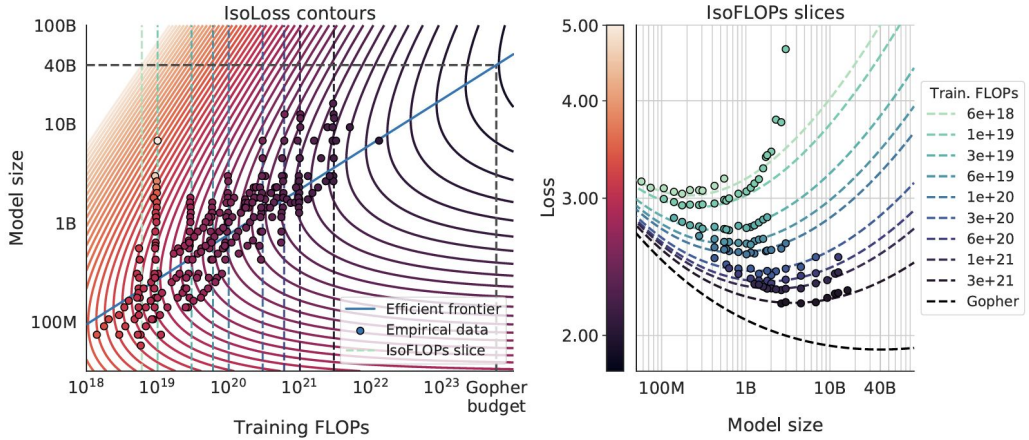


Figure 4 | **Parametric fit.** We fit a parametric modelling of the loss $\hat{L}(N, D)$ and display contour (**left**) and isoFLOP slices (**right**). For each isoFLOP slice, we include a corresponding dashed line in the left plot. In the left plot, we show the efficient frontier in blue, which is a line in log-log space. Specifically, the curve goes through each iso-loss contour at the point with the fewest FLOPs. We project the optimal model size given the *Gopher* FLOP budget to be 40B parameters.

Optimal model scaling

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

Here N → number of params, C → compute cost and D → size of training tokens

Optimal model scaling

Table 3 | **Estimated optimal training FLOPs and training tokens for various model sizes.** For various model sizes, we show the projections from Approach 1 of how many FLOPs and training tokens would be needed to train compute-optimal models. The estimates for Approach 2 & 3 are similar (shown in Section D.3)

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

Clearly suggests that given the training compute budget for many current LLMs, smaller models should have been trained on more tokens to achieve the most performant model.

Chinchilla

Optimal model size for Gopher → 40 to 70 B params

Compare Chinchilla to Gopher

- Both have the same number of FLOPs
- Differ in the size of the model and training tokens

Uses less compute for inference and storage compared to Gopher!

Chinchilla - Training as compared to Gopher

- Trained on same dataset as Gopher – MassiveText, slightly different subset distribution to account for increased number of training tokens.
- Uses AdamW instead of Adam in Gopher.
- Slightly modified SentencePiece Tokenizer, vocab is 94.15% similar to Gopher.

Model	Layers	Number Heads	Key/Value Size	d_{model}	Max LR	Batch Size
Gopher 280B	80	128	128	16,384	4×10^{-5}	3M $\rightarrow$ 6M
Chinchilla 70B	80	64	128	8,192	1×10^{-4}	1.5M $\rightarrow$ 3M

Table 4 | ***Chinchilla* architecture details.** We list the number of layers, the key/value size, the bottleneck activation size d_{model}, the maximum learning rate, and the training batch size (# tokens). The feed-forward size is always set to $4 \times d_{model}$. Note that we double the batch size midway through training for both *Chinchilla* and *Gopher*.

Chinchilla - Evaluation

	# Tasks	Examples
Language Modelling	20	WikiText-103, The Pile: PG-19, arXiv, FreeLaw, ...
Reading Comprehension	3	RACE-m, RACE-h, LAMBADA
Question Answering	3	Natural Questions, TriviaQA, TruthfulQA
Common Sense	5	HellaSwag, Winogrande, PIQA, SIQA, BoolQ
MMLU	57	High School Chemistry, Astronomy, Clinical Knowledge, ...
BIG-bench	62	Causal Judgement, Epistemic Reasoning, Temporal Sequences, ...

Table 5 | **All evaluation tasks.** We evaluate *Chinchilla* on a collection of language modelling along with downstream tasks. We evaluate on largely the same tasks as in Rae et al. (2021), to allow for direct comparison.

Chinchilla - Results

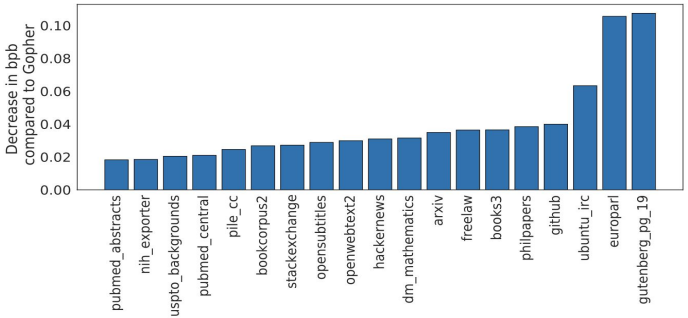


Figure 5 | **Pile Evaluation.** For the different evaluation sets in The Pile (Gao et al., 2020), we show the bits-per-byte (bpb) improvement (decrease) of *Chinchilla* compared to *Gopher*. On all subsets, *Chinchilla* outperforms *Gopher*.

Random	25.0%
Average human rater	34.5%
GPT-3 5-shot	43.9%
Gopher 5-shot	60.0%
***Chinchilla* 5-shot**	**67.6%**
Average human expert performance	*89.8%*
June 2022 Forecast	57.1%
June 2023 Forecast	63.4%

Table 6 | **Massive Multitask Language Understanding (MMLU).** We report the average 5-shot accuracy over 57 tasks with model and human accuracy comparisons taken from Hendrycks et al. (2020). We also include the average prediction for state of the art accuracy in June 2022/2023 made by 73 competitive human forecasters in Steinhardt (2021).

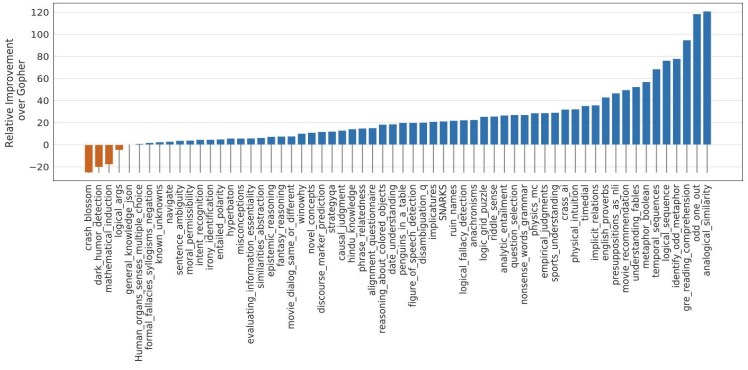


Figure 7 | **BIG-bench results compared to *Gopher*** *Chinchilla* out performs *Gopher* on all but four BIG-bench tasks considered. Full results are in Table A7.

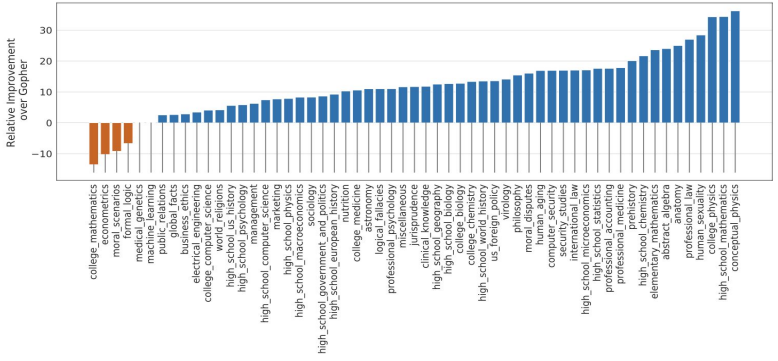


Figure 6 | **MMLU results compared to *Gopher*** We find that *Chinchilla* outperforms *Gopher* by 7.6% on average (see Table 6) in addition to performing better on 51/57 individual tasks, the same on 2/57, and worse on only 4/57 tasks.

Chinchilla - Results

	Chinchilla	*Gopher*	GPT-3	MT-NLG 530B
LAMBADA Zero-Shot	**77.4**	74.5	76.2	76.6
RACE-m Few-Shot	**86.8**	75.1	58.1	-
RACE-h Few-Shot	**82.3**	71.6	46.8	47.9

Table 7 | **Reading comprehension.** On RACE-h and RACE-m (Lai et al., 2017), *Chinchilla* considerably improves performance over *Gopher*. Note that GPT-3 and MT-NLG 530B use a different prompt format than we do on RACE-h/m, so results are not comparable to *Gopher* and *Chinchilla*. On LAMBADA (Paperno et al., 2016), *Chinchilla* outperforms both *Gopher* and MT-NLG 530B.

	Chinchilla	*Gopher*
All	78.3%	71.4%
Male	71.2%	68.0%
Female	79.6%	71.3%
Neutral	84.2%	75.0%

	Chinchilla	*Gopher*
Male *gotcha*	62.5%	59.2%
Male *not gotcha*	80.0%	76.7%
Female *gotcha*	76.7%	66.7%
Female *not gotcha*	82.5%	75.8%

Table 10 | **Winogender results. Left:** *Chinchilla* consistently resolves pronouns better than *Gopher*. **Right:** *Chinchilla* performs better on examples which contradict gender stereotypes (*gotcha* examples). However, difference in performance across groups suggests *Chinchilla* exhibits bias.

	Chinchilla	*Gopher*	GPT-3	MT-NLG 530B	Supervised SOTA
HellaSWAG	**80.8%**	79.2%	78.9%	80.2%	93.9%
PIQA	81.8%	81.8%	81.0%	**82.0%**	90.1%
Winogrande	**74.9%**	70.1%	70.2%	73.0%	91.3%
SIQA	**51.3%**	50.6%	-	-	83.2%
BoolQ	**83.7%**	79.3%	60.5%	78.2%	91.4%

Table 8 | **Zero-shot comparison on Common Sense benchmarks.** We show a comparison between *Chinchilla*, *Gopher*, and MT-NLG 530B on various Common Sense benchmarks. We see that *Chinchilla* matches or outperforms *Gopher* and GPT-3 on all tasks. On all but one *Chinchilla* outperforms the much larger MT-NLG 530B model.

	Method	*Chinchilla*	*Gopher*	GPT-3	SOTA (open book)
Natural Questions (dev)	0-shot	16.6%	10.1%	14.6%	
	5-shot	31.5%	24.5%	-	54.4%
	64-shot	35.5%	28.2%	29.9%	
TriviaQA (unfiltered, test)	0-shot	67.0%	52.8%	64.3 %	
	5-shot	73.2%	63.6%	-	-
	64-shot	72.3%	61.3%	71.2%	
TriviaQA (filtered, dev)	0-shot	55.4%	43.5%	-	
	5-shot	64.1%	57.0%	-	72.5%
	64-shot	64.6%	57.2%	-	

Table 9 | **Closed-book question answering.** For Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017), *Chinchilla* outperforms *Gopher* in all cases. On Natural Questions, *Chinchilla* outperforms GPT-3. On TriviaQA we show results on two different evaluation sets to allow for comparison to GPT-3 and to open book SOTA (FiD + Distillation (Izacard and Grave, 2020)).

Conclusion

- Trend so far → increase model size w/o increasing number of training tokens.
- Hypothesis: training larger and larger models is resulting into models that are substantially underperforming compared to what **could** be achieved with same compute power.
- Propose 3 predictive approaches → Gopher is over-sized!!
- Chinchilla (70B) outperforms Gopher (280B) and even larger models

Limitations

- Only have two comparable training runs at large scale, do not have intermediate results.
- Trained on less than one epoch of training data.
- Need of better, high quality training datasets.
  - May introduce ethical and privacy concerns as dataset size increases
- Chinchilla does suffer from bias and toxicity (but less affected than Gopher)

Need for dataset scaling!

DeepMind

Training Compute-Optimal Large Language Models

Reviewer

🔎: Hamoud Alhazmi

Recap of the main contributions

- Equal scaling of model size and training tokens:
  - The authors propose that for compute-optimal training, model size and training tokens should scale equally—doubling the model size should be matched by doubling the training tokens to achieve the best performance within a fixed compute budget.
- Chinchilla model:
  - A 70-billion-parameter model trained on 1.4 trillion tokens using the same compute budget as the larger Gopher (280 billion parameters).
  - Chinchilla outperforms Gopher and other larger models (e.g., GPT-3, Jurassic-1, Megatron-Turing), showing that bigger models are often over-sized and under-trained.
  - Chinchilla sets state-of-the-art performance on several benchmarks, including a 7% improvement over Gopher on the MMLU benchmark, with an average accuracy of 67.5%. This highlights the success of the compute-optimal scaling strategy.

🔎: Hamoud Alhazmi

Recap of the main contributions

- Extensive empirical study:
  - It is an extensive study, training over 400 models (ranging from 70 million to 16 billion parameters) on varying token counts. This analysis models the relationship between model size, training tokens, and performance, revealing that many large models are significantly undertrained for their size.
- Less compute for inference and fine-tuning:
  - The study shows that compute-optimal models like Chinchilla offer practical advantages. Despite being smaller, Chinchilla outperforms larger models and requires less compute for inference and fine-tuning, making it ideal for compute-limited applications.

🔎: Hamoud Alhazmi

Strengths

- **Clarity**: It is well-written, clearly explaining the methods used to explore the relationship between compute, model size, and data. It presents three approaches for estimating optimal trade-offs, with a well-motivated introduction and convincing empirical results.

- **Quality**: High-quality paper, with over 400 models trained to validate the hypothesis. It challenges existing assumptions and introduces Chinchilla, which outperforms larger models. Broad benchmark evaluations strengthen the conclusions.

- **Originality**: It advances compute-efficient language model training by proposing equal scaling of model size and data, challenging conventional approaches and setting a new direction for LLM design.

🔎: Hamoud Alhazmi

Strengths

- **Soundness**: The methodology is sound, with conclusions well-supported by extensive experimentation.

- **Broader Impact**: The paper has significant implications for LLM development, potentially shifting the focus toward more data-efficient and cost-effective model training.

🔎: Hamoud Alhazmi

Weaknesses

- **Power-law assumption**: One minor concern could be the reliance on the power-law assumption for scaling relationships, but the authors acknowledge this limitation and suggest potential future directions for refining these scaling laws.

- **Limited training runs at large scales**: Despite extensive experimentation with over 400 models, the paper provides only two large-scale training runs (Chinchilla and Gopher), limiting insight into how the scaling laws apply across intermediate model sizes and compute budgets.

- The paper briefly mentions **ethical concerns** like biases, toxicity, and privacy but lacks in-depth exploration. A more thorough analysis of these issues would have enhanced its impact on responsible AI discussions.

🔎: Hamoud Alhazmi

Review

- **Novelty (8/10)**: It offers new insights by challenging existing scaling laws and proposing compute-optimal training strategies, providing a fresh approach to model size and token scaling.
- **Correctness (9/10)**: It is backed by extensive experiments, with sound methodology and strong empirical evidence.
- **Clarity (7.5/10)**: Most of the parts are clearly explained.
- **Significance (10/10)**: Significant implications for future LLM development.
- **Recommendation**: Accept.

🔎: Hamoud Alhazmi

Archaeologist

Junjie Zhang

Gopher (2022)

Model	Layers	Number Heads	Key/Value Size	d_{model}	Max LR	Batch Size
44M	8	16	32	512	6×10^{-4}	0.25M
117M	12	12	64	768	6×10^{-4}	0.25M
417M	12	12	128	1,536	2×10^{-4}	0.25M
1.4B	24	16	128	2,048	2×10^{-4}	0.25M
7.1B	32	32	128	4,096	1.2×10^{-4}	2M
Gopher 280B	80	128	128	16,384	4×10^{-5}	3M $\rightarrow$ 6M

Table 1 | **Model architecture details.** For each model, we list the number of layers, the key/value size, the bottleneck activation size d_{model}, the maximum learning rate, and the batch size. The feed-forward size is always $4 \times d_{model}$.

Ref: Rae, Jack W., et al. "Scaling language models: Methods, analysis & insights from training gopher." *arXiv preprint arXiv:2112.11446* (2021).

Gopher (2022)

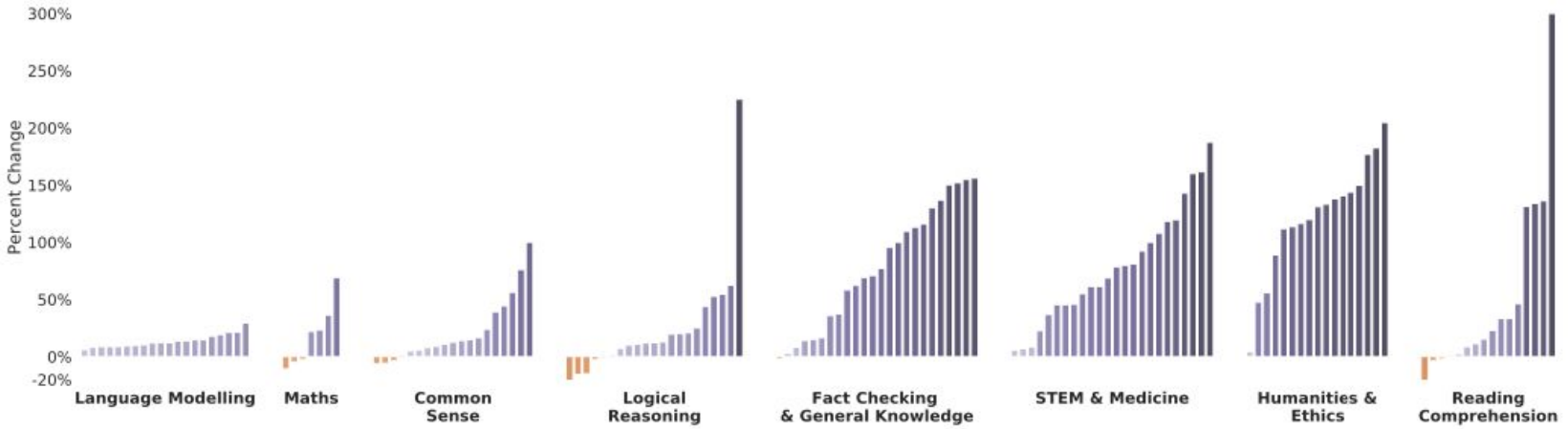


Figure 4 | **280B vs best performance up to 7.1B** across different tasks. We compare the performance of *Gopher* to the best performance of our smaller models up to 7.1B. In nearly every case, *Gopher* outperforms the best smaller model's performance. Small gains come from either scale not improving results substantially or the smaller models already being very performant. Language modelling improvements are in BPB and the rest are in terms of accuracy.

Scaling Laws for Neural Language Models (2020)

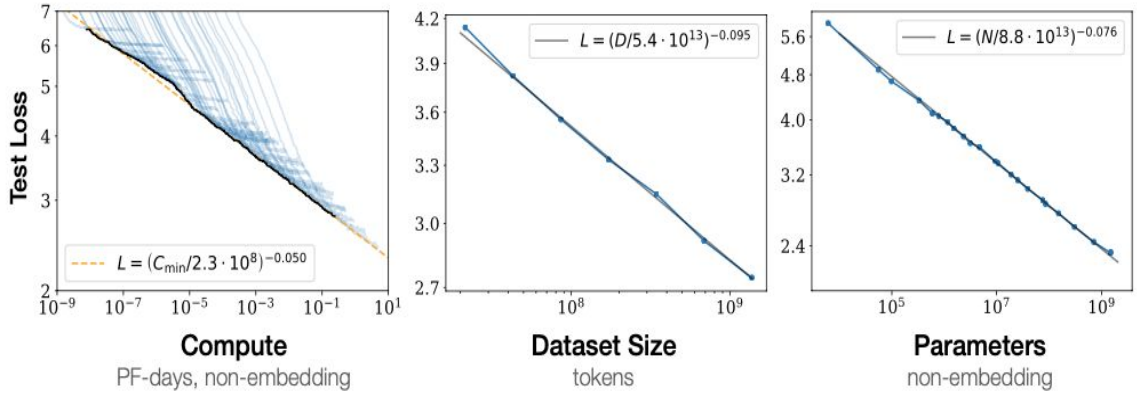


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when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

Smooth power laws: Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude. We observe no signs of deviation from these trends on the upper end, though performance must flatten out eventually before reaching zero loss.

Ref: Kaplan, Jared et al. "Scaling Laws for Neural Language Models." ArXiv abs/2001.08361 (2020): n. pag.

Scaling Laws for Neural Language Models (2020)

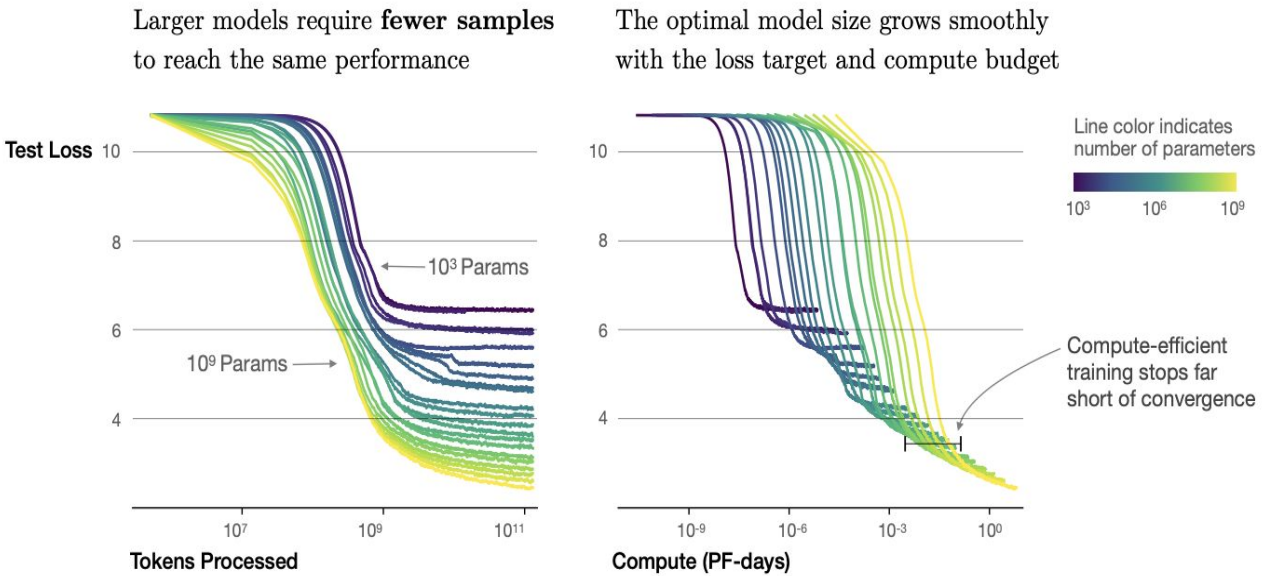


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps (Figure 2) and using fewer data points (Figure 4)

PF-day = $10_{15} \times 24 \times 3600 = 8.64 \times 10_{19}$ floating point operation

Scaling Laws for Neural Language Models (2020)

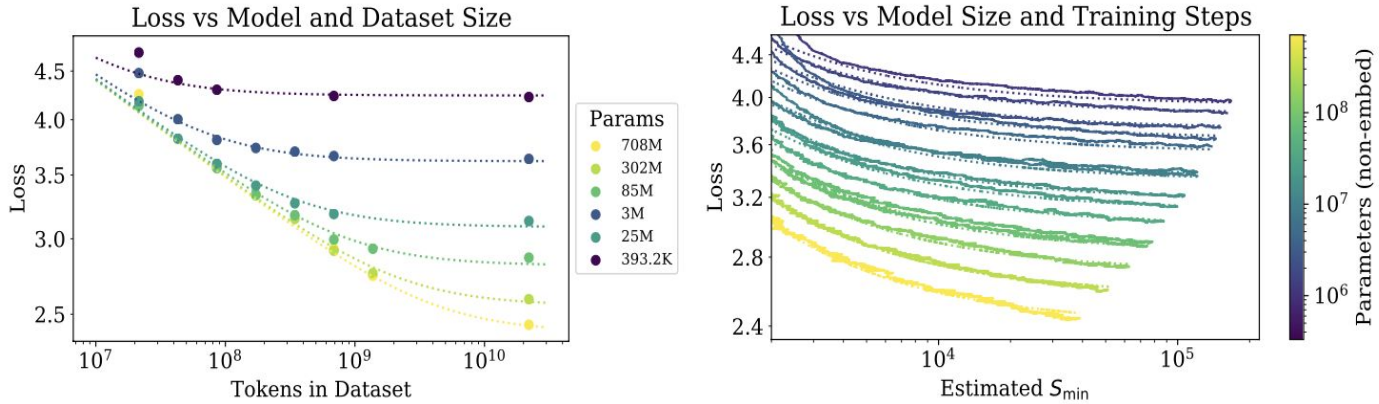


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of $S_{\min}$, the number of steps when training at large batch size (details in Section 5.1).

S_{min} – an estimate of the minimal number of training steps needed to reach a given value of the loss.

Scaling Laws for Neural Language Models (2020)

We have observed consistent scalings of language model log-likelihood loss with **non-embedding parameter count N, dataset size D, and optimized training computation C_{min},** as encapsulated in Equations (1.5) and (1.6). Conversely, we find **very weak dependence** on many architectural and optimization hyperparameters. Since scalings with N, D, C_{min} are power-laws, there are diminishing returns with increasing scale.

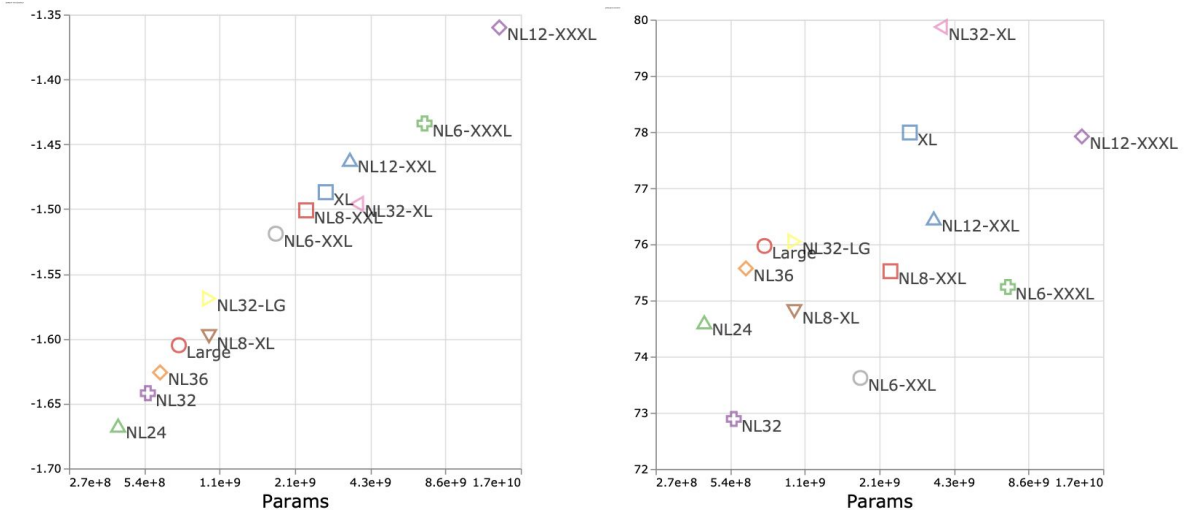
Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers.(2020)

We find that scaling laws may differ in **upstream and downstream setups.** Specifically, contrary to Kaplan et al. (2020), we find that downstream performance *strongly* depends on shape and not only on model size. Hence, pretraining performance may not necessarily transfer to downstream applications. (Figure 1).

Our findings show that pre-training perplexity can often be a **deceiving indicator of down-stream quality** and therefore model building based on upstream perplexity can be challenging. Scaling laws can differ substantially when considering metrics on actual downstream fine- tuning. (Figure 1)

Ref: Tay, Yi et al. "**Scale** Efficiently: Insights from Pre-training and Fine-tuning Transformers." *ArXiv* abs/2109.10686 (2021): n. pag.

Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers.(2020)



(a) Pre-training scaling

(b) Fine-tuning scaling

Figure 1: **The predictability and unpredictability of pre-training versus fine-tuning.** While the upstream pre-training performance measured by negative log-perplexity scales with model size quite independently from the model shape, the downstream performance (SuperGlue (avg) score) does not. This indicates that the shape of models plays an important role on how it performs on the target task and the performance is not merely a function of parameter size.

Visionary

AU24 CSE 5539 Presentation

Training Compute-Optimal Large Language Models

J. Hoffmann, S. Borgeaud, A. Mensch et al.

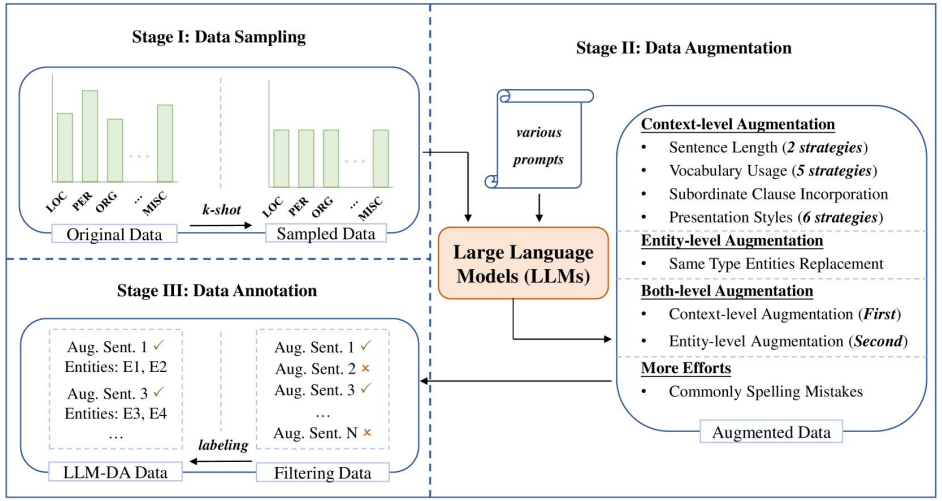
NeurIPS 2022

Paper: arxiv.org/abs/2203.15556

Sriram Sai Ganesh

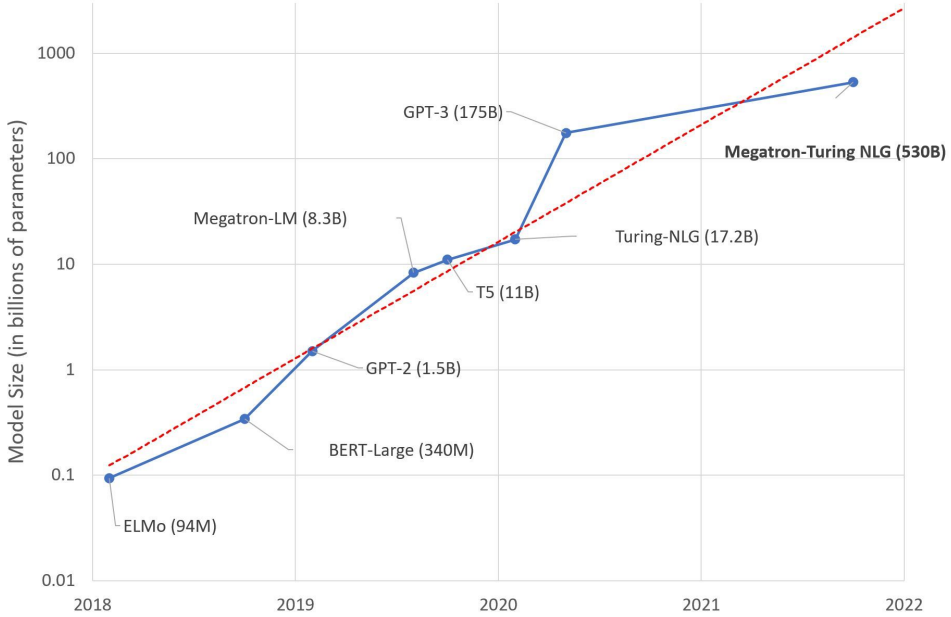
Data-Centric Training 📚

- Compute-optimal => lower FLOPs for identical performance.
- Focus moves to high-quality data for training
- Data Augmentation Using LLMs – https://arxiv.org/pdf/2403.02990
- Towards solving a related problem – How can we tell if something is AI-generated?



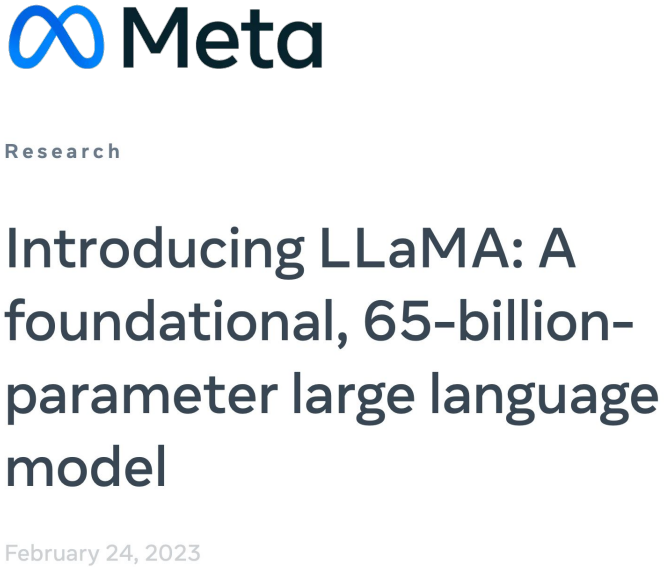
Big model go vroom 🏎️ 📈

- Larger models, emergent abilities.
  - New SoTA on a variety of task benchmarks, in succession.
- ‘Golden’ datasets
  - Often a well-kept secret.
- Arms-race like situation*
- No signs of slowing down*



Democratizing Frontier Research 🗳️

- Resources for SoTA model development out-of-reach for most NLP researchers.
  - GPT-4 – ~36M V100 hours;
- Models currently available may not be fair towards all.
  - Bias considerations
- Towards sustainability.



ai.meta.com/blog/large-language-model-llama-meta-ai/

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Stakeholder

Shantanu

Motivation

Motivation is to explore how “scaling test-time compute” for large language models (LLMs) can improve their performance, especially on complex tasks, without the need to increase model size

High-Level Summary of Paper

- Paper introduces adaptive strategies (iterative revisions and search-based methods with reward models) to allocate test-time compute based on task difficulty, optimizing performance without increasing model size.
- Given a fixed generation and compute budget, smaller models can achieve results comparable to much larger models on challenging tasks through these strategies
- In addition, it examines through experimental design the different scenarios where a tradeoff between pretraining compute and test-time compute is advantageous

Key Contributions

1. Explores Novel Adaptive method to allocate test-time compute based on task difficulty
2. Process Reward Models (PRMs) predicts the reward or correctness of each step during test-time inference. ORM evaluates only the final outcome.
3. Test-time compute can often replace the need to train large LLM models. Highly dependent on the task.
4. Search vs. Revision Strategies (Combination?): Paper evaluates two approaches: Searching to find the best answer and Correcting the answer if it is incorrect

Experiment Setup

The Proposed Framework

- Two Knobs for LLM Distribution Changes
  - **Proposer**: LLM generates an initial distribution of responses. This proposal distribution can be improved by iterative revisions or optimization techniques such as reinforcement learning
  - **Verifier**: The verifier scores and selects the best response from multiple proposals. It can evaluate final answers (as in best-of-N methods) or assess correctness at intermediate steps (using PRM).

Optimization for Scaling Equation

$$\theta^*_{q,a^*(q)}(N) = \text{argmax}_\theta \left(\mathbb{E}_{y \sim \text{Target}(\theta, N, q)} \left[\mathbb{1}_{y = y^*(q)} \right] \right), \quad (1)$$

where $y^*(q)$ denotes the ground-truth correct response for q, and $\theta^*_{q,y^*(q)}(N)$ represents the test-time compute-optimal scaling strategy for the problem q with compute budget N.

Assessing the Difficulty of a Question (MATH Dataset)

- Categorize each question into one of five difficulty levels based on the model's pass@1 rate (the likelihood of generating the correct answer in the first attempt).
- Difficulties were generated using 2048 samples per question.
- Proposes a Model-predicted difficulty (using verifier predictions) and Oracle Difficulty (ground-truth answer-based) methods for difficulty estimation

1st Experiment: Scaling Test-Time Compute via Verifiers

Comparing PRM Search Methods

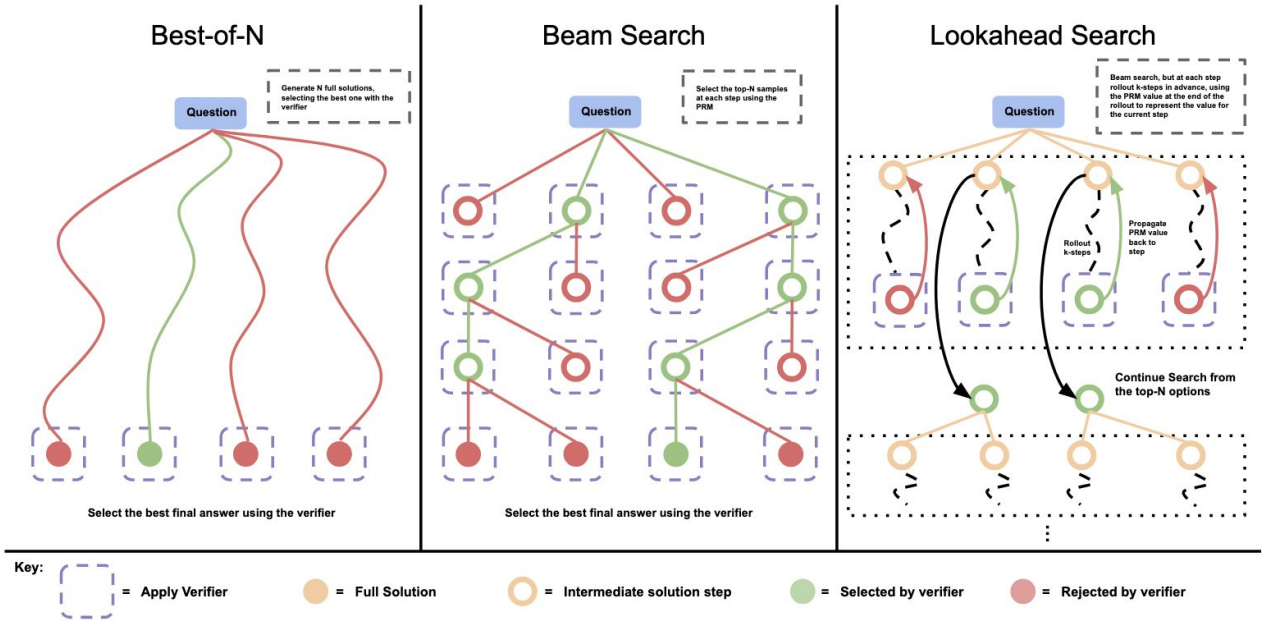


Figure 2 | ***Comparing different PRM search methods.*** **Left:** Best-of-N samples N full answers and then selects the best answer according to the PRM final score. **Center:** Beam search samples N candidates at each step, and selects the top M according to the PRM to continue the search from. **Right:** lookahead-search extends each step in beam-search to utilize a k-step lookahead while assessing which steps to retain and continue the search from. Thus lookahead-search needs more compute.

Results

- “Smaller generation budgets, beam search significantly out-performs best-of-N. However, as the budget is scaled up, these improvements greatly diminish, with Beam search often underperforming the best-of-N baseline.”
- “Lookahead-search generally underperforms other methods at the same generation budget, likely due to the additional computation inducted by simulating the lookahead rollouts”

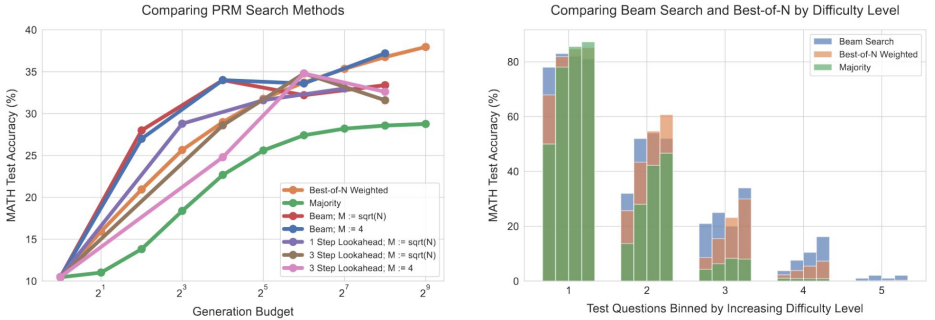


Figure 3 | **Left:** ***Comparing different methods for conducting search against PRM verifiers***. We see that at low generation budgets, beam search performs best, but as we scale the budget further the improvements diminish, falling below the best-of-N baseline. Lookahead-search generally underperforms other methods at the same generation budget. **Right:** ***Comparing beam search and best-of-N binned by difficulty level***. The four bars in each difficulty bin correspond to increasing test-time compute budgets (4, 16, 64, and 256 generations). On the easier problems (bins 1 and 2), beam search shows signs of over-optimization with higher budgets, whereas best-of-N does not. On the medium difficulty problems (bins 3 and 4), we see beam search demonstrating consistent improvements over best-of-N.

2nd Experiment: Refining the Proposal Distribution

Different Refinement Approaches

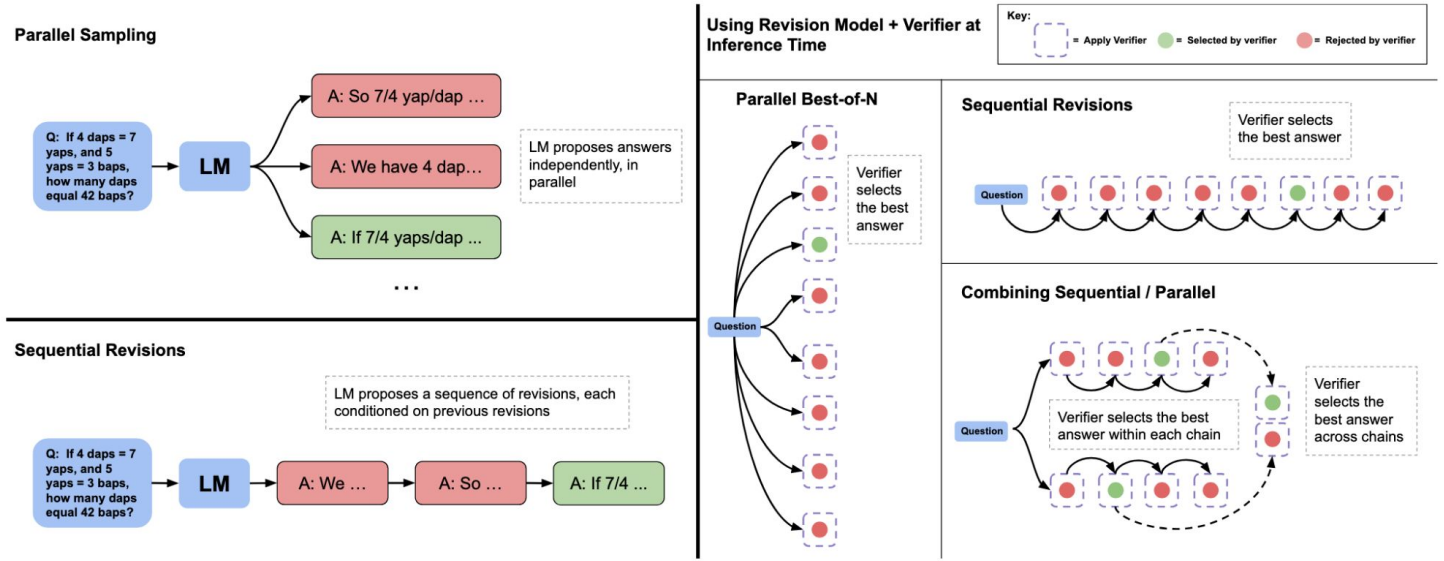


Figure 5 | ***Parallel sampling (e.g., Best-of-N) verses sequential revisions.*** **Left:** Parallel sampling generates N answers independently in parallel, whereas sequential revisions generates each one in sequence conditioned on previous attempts. **Right:** In both the sequential and parallel cases, we can use the verifier to determine the best-of-N answers (e.g. by applying best-of-N weighted). We can also allocate some of our budget to parallel and some to sequential, effectively enabling a combination of the two sampling strategies. In this case, we use the verifier to first select the best answer within each sequential chain and then select the best answer across chains.

Results

- “We find that there exists a tradeoff between sequential (e.g. revisions) and parallel (e.g. standard best-of-N) test-time computation, and the ideal ratio of sequential to parallel test-time compute depends critially on both the compute budget and the specific question at hand.”
- “Specifically, easier questions benefit from purely sequential test-time compute, whereas harder questions often perform best with some ideal ratio of sequential to parallel compute.”

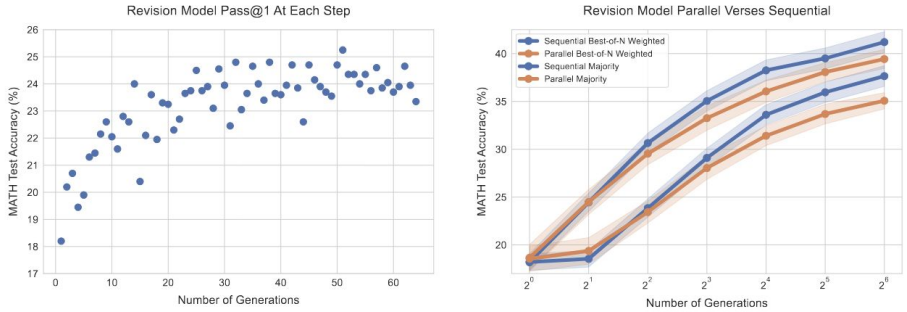


Figure 6 | **Left:** ***Our revision model's pass@1 at each revision step.*** Pass@1 gradually improves after each revision step, even improving beyond the 4 revision steps that it was trained for. We estimate pass@1 at each step by by averaging over the performance of 4 revision trajectories of length 64 for each question in the test-set. **Right:** ***Sequential vs parallel sampling from the revision model.*** Comparing performance when generating N initial answers in parallel from our revision model, verses generating N revisions sequentially, with the model. When using both the verifier and majority voting to select the answer, we see that generating answers sequentially with the revision model narrowly outperforms generating them in parallel.

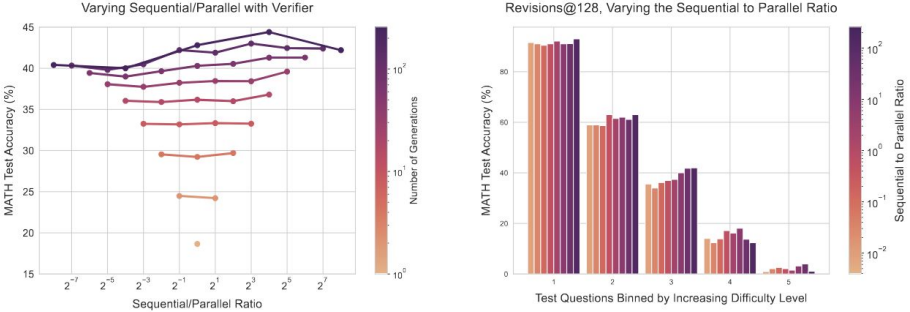


Figure 7 | **Left:** ***Varying the ratio of the generation budget allocated sequential revisions to verses parallel samples.*** Each line represents a fixed generation budget as the ratio is changed. We use the verifier for answer selection. We see that while increased sequential revisions tends to outperform more parallel compute, at higher generation budgets there is an ideal ratio that strikes a balance between the two extremes. **Right:** ***Varying the sequential to parallel ratio for a generation budget of 128 across difficulty bins.*** Using verifier-based selection, we see that the easier questions attain the best performance with full sequential compute. On the harder questions, there is an ideal ratio of sequential to parallel test-time compute.

Bringing it all Together

Conclusion:

- For easy and medium tasks, test-time compute often does not require the need for additional pretraining. However, for more challenging tasks or when the inference requirements are high, increasing pretraining is generally more effective at boosting model performance
- Pretraining is often necessary to improve results on more difficult problems that fall outside of the model's trained abilities

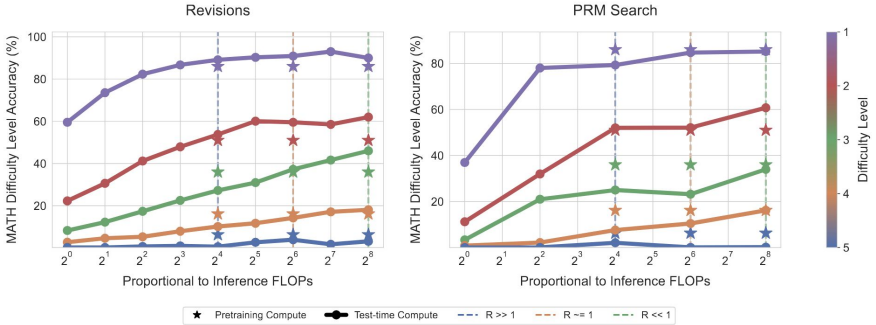


Figure 9 | ***Tradeoff between pretraining and test-time compute in a FLOPs-matched evaluation.*** Each line represents the performance of scaling test-time compute with our compute-optimal policy in each oracle difficulty bin. We plot the results for revisions on the left and search on the right. The stars represent the greedy pass@1 performance of a base model pretrained with ~ 14 times more parameters. We plot test-time compute budget on the x-axis, and place the stars at three different locations along the x-axis, each corresponding to the FLOPs equivalent point of comparison between scaling parameters and scaling test-time compute for three different inference compute loads (e.g. $R = \frac{D_{\text{inference}}}{D_{\text{pretrain}}}$). If the star is below the line, this implies that it is more effective to use test-time compute than to scale model parameters, and if the star is above the line this implies that scaling parameters is more effective. We see that on the easy questions or in settings with a lower inference load (e.g. $R << 1$), test-time compute can generally outperform scaling model parameters. However, on the harder questions or in settings with a higher inference load (e.g. $R >> 1$), pretraining is a more effective way to improve performance.

Reviewer

Adam Ryan

Strengths

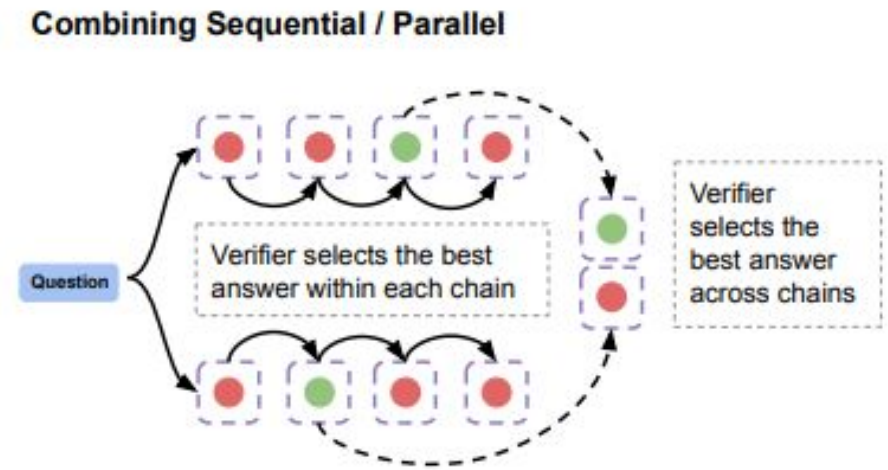
- Method for generating sequential revision data using high-temperature batch generation and subsequent manual chain construction.
- Use of a difficulty-based scaling ratio between sequential and parallel sampling. (at the same time is a weakness)
- Comprehensive evaluation of various approaches, including search strategies and verifiers.

Weaknesses

- Heavy reliance on accurate difficulty assessment; misjudgment impacts results significantly. Flawed separation method
- Limited explanation on how problem difficulty is determined compared to detailed search algorithm descriptions.
- Evaluation limited to the MATH dataset.
- Limited/insufficient depth on verifier training methods.
- Lack of clarity/Not clear to read. Way too many fancy words
- Focuses more on explaining search algorithms than highlighting novel contributions.
- Some questionable choices

Example

- When combining techniques, why apply verifier twice instead of once for all?



Review

- Novelty 4.0/5
- Correctness 4.5/5
- Clarity 2.0/5
- Significance 4.0/5
- Recommendation: Tentative Accept

Archaeologist

Zephyr Jiang

Which work inspires this one

- Chain-of-thought prompting
  - Self-refine LLM
- Scaling law for training
- Best-of-n sampling
  - Searching (Beam-search/DFS/BFS/MCTS) on inference

Concurrent work/Which work is inspired by this one

- OpenAI's o1 model
- Training LLM to self-correct
- My class project for this course - application of LLM self-correct

Visionary

Roozbeh Nahavandi

Future Directions

Scalability and Generalization

- **How would optimal test-time compute do across different tasks?**

  The question is whether the same approach can be applied effectively to various problem types, or if adjustments are needed. Do strategies that work in one area, like mathematics, generalize well to others, such as Code Generation & Understanding or Logical Reasoning?

Future Directions

Scalability and Generalization

- **How would optimal test-time compute do across different tasks?**

  The question is whether the same approach can be applied effectively to various problem types, or if adjustments are needed. Do strategies that work in one area, like mathematics, generalize well to others, such as Code Generation & Understanding or Logical Reasoning?

- **How would compute-optimal scaling and process-based verifier models impact larger/smaller models?**

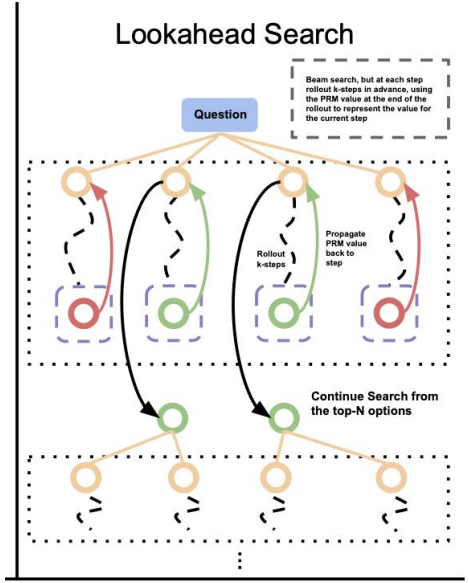
  Would applying these strategies to larger/smaller models lead to significant improvements in accuracy and efficiency, or would the gains be marginal due to the models' inherent capabilities? Could these techniques further optimize computational resources in large-scale models, especially when dealing with complex tasks?

Future Directions

Optimization of Test-Time Compute Strategies

- **How would other search algorithms like Monte Carlo Tree Search affect test-time compute optimization?**

  Search algorithms like MCTS, beam search, or lookahead search could explore multiple response paths at test-time. The question is whether these methods can enhance output refinement and accuracy without significantly increasing computational costs.



Future Directions

Optimization of Test-Time Compute Strategies

- **How would other search algorithms like Monte Carlo Tree Search affect test-time compute optimization?**

  Search algorithms like MCTS, beam search, or lookahead search could explore multiple response paths at test-time. The question is whether these methods can enhance output refinement and accuracy without significantly increasing computational costs.

- **How would chain-of-thought reasoning compare to their approach?**

  Would it provide better accuracy or consistency during test-time compute? How can we implement chain-of-thought reasoning alongside the compute-optimal strategies in this paper to further enhance performance?

Future Directions

Optimization of Test-Time Compute Strategies

- **How would other search algorithms like Monte Carlo Tree Search affect test-time compute optimization?**

  Search algorithms like MCTS, beam search, or lookahead search could explore multiple response paths at test-time. The question is whether these methods can enhance output refinement and accuracy without significantly increasing computational costs.

- **How would chain-of-thought reasoning compare to their approach?**

  Would it provide better accuracy or consistency during test-time compute? How can we implement chain-of-thought reasoning alongside the compute-optimal strategies in this paper to further enhance performance?

- **How can we prevent the language model from exploiting the reward model?**

  Models may exploit weaknesses in the reward model to generate low-quality outputs. Can integrating strategies like chain-of-thought reasoning or adaptive feedback help mitigate this and ensure better-quality results?

Future Directions

Adaptive and Dynamic Compute Allocation

- **How can scaling with dynamic task difficulty prediction optimize test-time compute?**

  This paper estimates task difficulty using model-specific metrics, but future models could predict task difficulty in real-time. The question is whether dynamically adjusting compute allocation based on these predictions can lead to more efficient use of resources, allowing the model to focus computational power only on challenging tasks while optimizing overall performance.

Thank You!