When to Think Step by Step: Computing the Cost-Performance Trade-offs of Chain-of-Thought Prompting

Akshay Manglik
Computer Science
Columbia University
New York, United States
am5747@columbia.edu

Aman Choudhri
Statistics
Columbia University
New York, United States
ac4972@columbia.edu

***Abstract*—As large language models become more widely deployed, understanding the tradeoffs between inference optimizations and model scaling grows increasingly important. This paper introduces a novel methodology for evaluating the cost-performance characteristics of inference strategies like chain-of-thought (CoT) prompting across model scales and architectures. Through experiments across 26 models and 7 reasoning benchmarks, we find that CoT's benefits emerge differently when scaling model size versus training data - with stronger effects from increasing model size. We demonstrate that for many inference scenarios, CoT prompting is more compute-efficient than training larger models to achieve equivalent performance, although this effect is modulated by using a sufficiently large base model to generate enough high-quality tokens. Our analysis framework provides practical guidance for ML practitioners on when to employ inference optimizations versus scale up model training. We also identify limitations in current approaches to decomposing model capabilities, showing that code reasoning tasks play an outsized role in previous analyses. This work advances our understanding of how to optimally allocate compute between training and inference time for language model deployment.**

Code and results are available at our project page.

***Index Terms*—LLM, scaling laws, inference compute**

I. Introduction

A. Background and Motivation

Large language models have reshaped natural language processing, achieving state-of-the-art performance on tasks like text classification, machine translation, summarization, and question answering [42, 52]. Many of these advancements were driven by improvements in pre-training, such as increasing model size (N) and training data token counts (D), while also improving training data quality and recipes [5, 41, 29]. While post-training has improved instruction-following, alignment, and reasoning ability, much of model scaling research has been focused on pre-training, which drives the majority of costs of developing LLMs [25].

As demand for inference tokens grows, compute and cost constraints create a strong incentive for those deploying LLMs in practice to find smaller, sufficiently performant alternatives. In addition to 'free-lunch' systems improvements like Flash Attention [11] and vLLM [27] that allocate GPU memory more efficiently, researchers have turned to model quantization [23, 14, 36], model pruning [7], and distillation [21, 1, 2], as ways of reducing inference costs, with varying degrees of impact on model accuracy.

Another approach is to train smaller models on a larger number of training tokens, anticipating that a large volume of tokens will be generated at inference time [35, 15]. Essentially, this approach "trades" extra cost at training time for cheaper and less compute-intensive inference, amortizing the cost of training over the course of many inference requests.

Parallel to this push for smaller models has been a growing recognition that spending more compute at inference time, per request, can dramatically improve performance. There are a variety of strategies to allocate this extra compute via sampling and search methods, such as chain of thought, self-consistency, best-of-N, and Monte Carlo Tree Search (MCTS). For a more complete review of such "inference-time optimizations," please see [47].

The competition between inference cost incentivizing smaller and cheaper models and performance incentivizing larger, costly models or greater inference-time compute creates a need for detailed study of the tradeoffs of any given inference-time optimization. We attempt to address this need in this paper. Specifically, we introduce a novel methodology to study the cost-performance tradeoff of any given inference-time optimization, taking advantage of a recent observational technique to characterize the scaling performance of training compute [33]. We also analyze and interpret the limitations of this observational scaling technique, and we identify characteristics of emergent behavior of chain of thought at a variety of FLOP scales.

Our contributions are as follows:

1) We evaluate the effects of Chain of Thought Prompting on model performance across a variety of reasoning and knowledge benchmarks, noting disparate effects between increases in model size and training data levels.
2) We analyze the relative weight of training FLOPs and inference tokens in driving Standard Prompting and Chain of Thought Prompting accuracy, noting that train-

ing FLOPs is more important for Chain of Thought prompting than inference token count.
3) We identify
4) We identify how Chain of Thought allows a model to achieve benchmark performance of a larger mb propose a novel methodology for characterizing the cost-performance tradeoffs of LLM inference strategies across model scales and families.
5) We reproduce part of the Observational Scaling Laws analysis from Ruan et al, and demonstrate the importance of coding tasks for their analysis through benchmark ablation.
6) We document various evaluation approaches for the benchmarks and open-source our modified scripts.

II. Literature Review

A. Scaling Laws

Kaplan et al introduced neural scaling laws, which extrapolate out reliable log-linear predictions for model loss based on a parametric equation using model size (N) and training data levels (D) [24]. Hoffman et al developed the Chinchilla compute-optimal scaling laws, which scale the number of training tokens equally with model size [22]. Sardana et al incorporated the cost of inference into Chinchilla scaling laws and found that training smaller models for longer periods of time were superior.

Ruan et al extended the scaling law literature by formulating "Observational Scaling Laws." Ruan et al hypothesized that model families differ in their efficiency in converting training compute to benchmark performance; this is mediated by latent reasoning capabilities. Specifically, they measured these latent reasoning capabilities by decomposing benchmark performance into a low-dimensional space using PCA, and link the top three principal components to general reasoning, mathematical reasoning, and code reasoning. They generated model-family-specific curves that relate latent reasoning to compute, and model-family-agnostic curves that relate latent reasoning to benchmark performance. Additionally, they proposed a model-family-equivalent FLOPs metric, which scales a model family's performance to the compute efficiency of a reference family, and used this to predict emergent capabilities and post-training performance.

B. Inference Optimizations

We identify several categories of inference optimizations. We focus on optimizations that do not require further finetuning, although many optimizations utilize supervised finetuning, rejection finetuning, or RL-based post-training in order to further improve model reasoning at inference-time.

Sampling Several works have explored various sampling methods. Token-level sampling parameters adjust methods for selecting the next token from language model output distributions: these include top-k sampling, which only considers the top-k tokens with the highest probability; top-p, or nucleus, sampling, which considers the top tokens whose probability sums to p; and top-eta sampling, which weights probability by distribution entropy to improve generation in low entropy settings [47, 20]. Contrastive Decoding uses two LMs, one large and one small, to identify the best token; based on the idea that a log probability gap between a small, low-skill model and a larger, high-skill model indicates a token is especially promising, Contrastive Decoding predicts tokens based on the difference between the probability it is predicted by the large model and the probability it is predicted by the small model [30]. Task-level sampling methods repeatedly sample completions for a given task, to identify better answers. Token-level sampling complements task-level sampling methods by introducing diversity in generations, increasing the probability that additional generations cover the correct answer. Task-level sampling methods include Self-Consistency, which repeatedly samples chain of thought continuations for a task and outputs the most common answer, regardless of reasoning [45]. Chain of Thought Decoding notes that alternative tokens predicted by the model often contain chain of thought paths, even when not prompted for by the user [44]. They also identify that the final answer of a chain of thought path is more confident (having a large gap in probability between the first and second most probable tokens), and use that to create a metric for identifying chain of thought paths for decoding.

Search Search techniques attempt to refine intermediate model outputs during generation, by modeling the generation process as traversing over a tree or graph structure. Search techniques often include backtracking, and make use of sampling techniques like Monte Carlo Tree Search to estimate the value of potential continuations [51]. Another common search technique is Beam Search, which holds a "beam" of potential generations, adding to each potential sequence at each timestep and then winnowing them down to only keep the top-K using metrics like repetition and length [47].

Verifiers Verifier models, which identify whether an given output is correct or not, are often used to complement sampling and search, either by identifying correct solutions or providing intermediate reward to guide generations, e.g., through process reward models. Automatic verifier models are deterministic (e.g., automated proof solvers, code compilers), often used for math and coding tasks. Generative verifiers use an underlying generative AI model, such as a large language model, along with an output head that identifies if the answer is correct or outputs a reward. Learned verifiers are finetuned on verification data for greater accuracy [47, 37]. Verifiers can either be process-based or outcome-based, depending on whether they act on intermediate steps of a generation or only evaluate the final, full generation. Verifier-based methods include Best-of-N, which samples N generations and identifies whether any are correct, returning the correct one, and Weighted Majority Vote, which weights outputs by the probability they are considered correct (based on a reward model), and takes the most valued output based on those weights and frequencies [4]. Techniques like ReST-MCTS utilize process reward models to inform tree search and backtracking [51].

Prompting Prompting techniques append instructions or directives to the end of a prompt, in order to elicit desired

reasoning behavior from a language model. The most well-known prompting technique is Chain of Thought, which either directs the model to "think step by step" or provides the model with a few examples of reasoning processes [46, 26]. Both these techniques motivate the model to engage in an identifiable stepwise reasoning process, which often boosts generation quality and task accuracy. Chain of Thought has been noted to help primarily on math and symbolic reasoning prompts [38]. Other prompting techniques attempt to induce more complex reasoning behavior, often using few-shot demonstrations of the desired behavior; many exist, and descriptions of a few promising techniques follow. Analogical prompting drives the model to identify relevant analogies to the task using in-context learning, in order to condition the model on relevant portions of its weight space to the task at hand [49]. Least-to-Most Prompting motivates a model to break down a problem into constituent subproblems, and use the solution to the subproblems to inform the overall solution [54]. Complexity-Based Prompting prompts a language model with more complex few-shot reasoning examples (defined as having more reasoning steps); it also limits self-consistency to considering the top-K reasoning traces, sorted by number of reasoning steps [12]. Self-Endorsement breaks down model generations into constituent facts, has the model verify each fact, and selects the sample with the greatest number of verified facts or generates a new sample using only verified facts [43]. Step-Back Prompting prompts the model to identify a followup question to a task (e.g., what are the physics principles behind this question?), and then generates a chain-of-thought reasoning solution using the answer to that follow-up question [53].

III. Methodology

We consider how models perform with and without inference-time optimizations. Of the inference-time optimizations, we initially considered and implemented Chain of Thought, Beam Search, and Self-Consistency; we then narrowed our focus to only Chain of Thought, due to its importance in reasoning and limitations on compute.

To identify how reasoning ability improves and stratifies across model abilities, we evaluate a variety of model families, model sizes, and training token levels. Specifically, we evaluate the Gemma 2 {2B, 9B} [39], Llama 3.2 {1B, 8B} [15], Pythia {160M, 410M, 1B, 1.4B, 2.8B, 6.9B, 12B} [3], Qwen2.5 {0.5B, 1.5B, 3B, 7B, 14B} [48, 40], and OLMo {1B, 7B} [16] models. To understand the role of training token levels, following the guidance of [8] we evaluate OLMo 1B at checkpoints for {10B, 100B, 1T, 2T, and 3.05T} pre-training tokens, and OLMo 7B at checkpoints for {10B, 100B, 1T, 2T, and 2.75T} pre-training tokens.

Following Ruan et al, we evaluate baseline and chain-of-thought inference on a variety of benchmarks that test subject-matter knowledge, commonsense reasoning, scientific reasoning, mathematical reasoning, and cross-linguistic reasoning. Specifically, we evaluate models on GSM8K [10], Arc-Challenge [9], Winogrande [34], XWinograd [32], HellaSwag [50], MMLU [19], and TruthfulQA [31], amounting to around 130,000 questions per model. Unlike Ruan et al, we do not consider HumanEval [6], due to difficulties with evaluating arbitrarily-generated code in a safe manner and compute limitations; we leave this as future work.

For conducting evaluations, we utilize Eleuther AI's open-source LM Evaluation Harness, which contains evaluation scripts for each of the benchmarks we evaluated [13].

GSM8K follows a "free-form generation" format, where models generate a number of tokens before a regex is used to identify the model's final numeric answer, if provided. This is done to enable Chain-of-Thought, which GSM8K tests by providing few-shot examples in each prompt. No other benchmark we evaluated provides few-shot examples. These other benchmarks follow a "log-likelihood" task format, where the model is given the question and a list of multiple choice answers, and the model answer is ascertained based on the logits of the four possible letter choices. This does not leave room for generating additional tokens of reasoning, which is necessary for all inference-time optimizations. Therefore, for evaluating Chain of Thought, we generate modified scripts for each benchmark, following guidance from the OLMES evaluation suite [17] on question framing and evaluation formats. These modified "free-form generation" evaluations permit models to generate rationales for answering a question, after which we use regexes to identify any answers given. We evaluate models using full precision (double precision) activations and weights.

These differing answer formats for baseline and chain of thought analysis make it difficult to compare answer quality and accuracy between models, as the free form generation format is significantly more difficult and harder to evaluate. (See Appendix for additional details.) We therefore run a second round of evaluations on both baseline and chain of thought tasks, using a unified hybrid format. For log-likelihood format question, we first generate up to 256 tokens of output, regardless of whether chain of thought is specified. Following this, we postpend the generation with the string "The answer is:" and evaluate the log-likelihood of all given answer choices, enabling a more reliable measurement of answer quality given a lack of consistency in language model answer formats. Additionally, to improve generation efficiency, we utilize the vLLM engine [28] for efficient memory management. We modify the LM Evaluation Harness scripts for vLLM integration to enable this custom answer format. We evaluate models by quantizing activations and weights using the brain floating point (bfloat16) format, due to similar benchmark performance to full precision activation and weights.

IV. Experimental Results

A. Log-Likelihood Form

In our first experimental section, we evaluate models using the common log-likelihood evaluation format. For each dataset, models are presented with a multiple choice question. Then the log-likelihood of each answer choice string ("(A)",

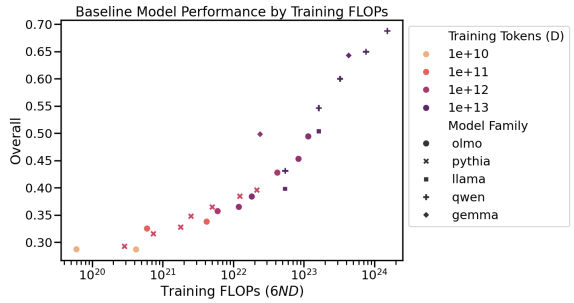

Fig. 1. Model Performance on Log-Likelihood Format. Accuracy scores across 7 tasks are averaged to yield one overall score. A tight correlation is observed between training FLOPs and performance, with significant performance growth occurring at 10^{23} training FLOPs.

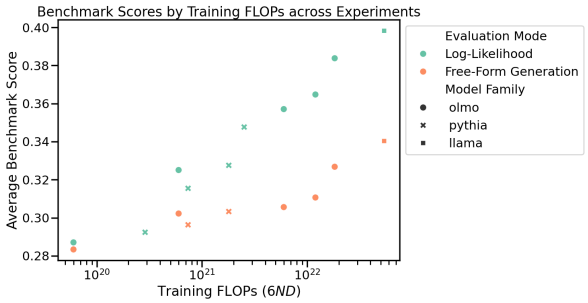

Fig. 2. Model Performance, Both Log-Likelihood and Hybrid Generation Tasks. Log-likelihood evaluation significantly outperforms free-form generation after around 10^{21} training FLOPs, reflecting the more difficult nature of the free-form task.

for example) is computed and the model is scored using the answer choice logits.

As this is a fairly inexpensive format, we were able to evaluate a diversity of models across scales from 160M to 14B parameters. The performance of each model is plotted in Figure 1, with the total compute used to train the model on the x-axis. To calculate this figure, we use the common approximation of training compute as $6ND$ FLOPs for N parameters and D training tokens.

Chain of Thought prompting requires that a model is able to output tokens to generate a "reasoning" trace. As such, it is unsuitable for use on the log-likelihood format.

B. Hybrid Generation Format

In the hybrid generation format, the models are similarly prompted with a multiple choice question, then they are allowed a budget of $T = 256$ tokens in which to output "reasoning tokens" that shift the conditional next-token distribution closer towards the correct answer. We then add "The answer is:" to the end of the model's response before measuring logits of all possible multiple choice answers to select a final answer.

This format is notably more difficult for the models, and requires far more forward passes. As such, we were only able to evaluate a subset of the models. The average benchmark scores, contrasted with those from the log-likelihood evaluation mode, are presented in Figure 2. At around 10^{21} training FLOPs, the performance from log-likelihoood evaluation rapidly diverges from that of free-form generation. The two do remain correlated, however.

C. Chain of Thought Prompting

The inference-time optimization we evaluate is Chain of Thought prompting [46]. For this strategy, we keep the same generation budget of $T = 256$ tokens. But before prompting the model for its response, we insert the tokens: "Let's think step by step." The performance results are displayed in Figure 3.

Concordant with the original Chain of Thought paper, we observe differential benefits across model sizes and train tokens. We examine this further in Figure 4, plotting the difference between performance with and without CoT.

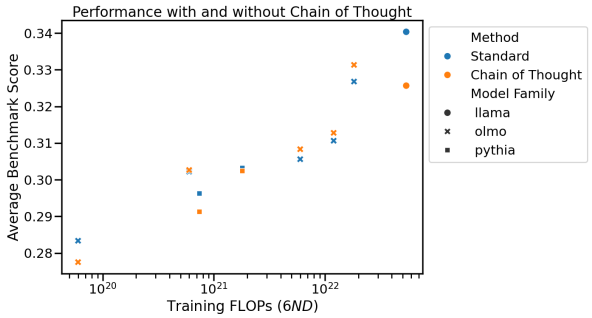

Fig. 3. Model Performance with and without Chain of Thought.

Interestingly, we observe different rates of growth in the benefit of Chain of Thought across model families. In the Pythia family, where train tokens were fixed and model size increased, we see a relatively steep growth rate. By contrast, the performance improvement slope within the OLMO family is far shallower. For this experiment, the OLMO models were all intermediate checkpoints of the same 1B-parameter model, meaning the model size was fixed and the train tokens varied.

Data across a larger collection of models is necessary to

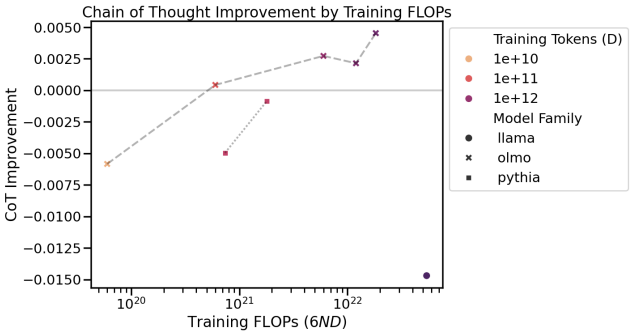

Fig. 4. Average Benchmark Performance Improvement Using Chain of Thought. Higher is better, with the line $y = 0$ representing equivalent performance with or without Chain of Thought. Dotted lines are drawn between models within the same family.

provide more evidence for this apparent trend. But to our knowledge, this is a novel result.

Contrary to expectations, the 1B LLAMA-3.2 model suffered a substantial performance hit when using Chain of Thought prompting. This may be due to the model hitting the $T = 256$ generated tokens ceiling and being unable to produce a sufficiently helpful answer rationale within budget. Table I supports this theory: LLAMA-3.2 on average generates far more tokens for the same prompt relative to the other models we evaluated, when using Chain of Thought.

Model	Standard	CoT	Token Ratio
Llama-3.2-1B	36.68	136.07	24.65
OLMo-1B, 3T checkpoint	20.90	30.10	94.69
OLMo-1B, 2T checkpoint	14.92	47.10	6.18
OLMo-1B, 1T checkpoint	17.15	74.54	47.49
OLMo-1B, 0.1T checkpoint	15.99	39.91	12.22
OLMo-1B, 0.01T checkpoint	16.79	50.56	8.64
Pythia-1B	23.77	64.77	7.94
Pythia-410M	23.77	18.73	1.63
Mean	21.25	57.72	8.20

TABLE I
TOKENS GENERATED USING CHAIN OF THOUGHT VERSUS STANDARD PROMPTING

D. Latent Reasoning Decomposition

Following Ruan et al, we decompose our benchmark performance into a low-dimensional space using PCA. Figure 5 shows the relative weights of each benchmark from our latent reasoning vectors. Only the first principal component is meaningful, and it primarily averages over MMLU (subject-matter knowledge) and GSM8K (mathematical reasoning). We attribute this partially to using a small number of models and smaller-sized models, which limits the variation in benchmark performance crucial for this analysis.

However, our analysis also does not include code reasoning (HumanEval), which could explain the poor PC differentiation as well. This led us to reproduce the Ruan et al PC loadings, sans HumanEval, in Figure 6. We also find that the Ruan et al PC loadings are highly dependent on the presence of code reasoning; without including a code benchmark, there is only one meaningful PC with substantially weaker coefficients, which averages over all benchmarks in similar proportions to our PC loadings.

E. Cost-Performance Profiling

Finally, we model the relationship between inference cost and benchmark performance. Figure 7 demonstrates the relationship between tokens generated and average benchmark performance, for both standard prompted models and chain of thought prompted models. Performance of standard and chain of thought prompting can be thought of as having two influences: increased training FLOPs, and increased inference tokens generated. We decompose the performance of Standard and Chain of Thought prompting into these two elements using fitted IsoFLOP curves to standard and chain of thought prompt data, respectively. We fit training FLOPs and inference token data for each prompt group using the equation

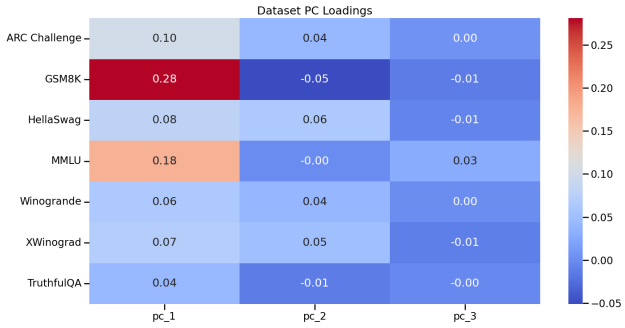

Fig. 5. PC Loadings from Decomposing Benchmark Performance into Low-Dimensional Space.

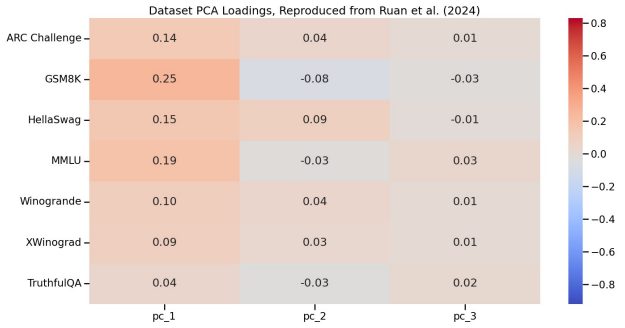

Fig. 6. PC Loadings from Ablating HumanEval (Code) from Ruan et al Data [33]. We use the same visualization scale as the original graph from the Ruan paper.

$benchmark = \alpha \log(FLOPs) + \beta * tokens + \delta$. We find optimal fit for Standard Prompting for $\alpha = 0.00639, \beta = 0.000684, \delta = -0.0215]$, and for Chain of Thought Prompting for $\alpha = 0.00680, \beta = 0.0000798, \delta = -0.0320$.

The slope of these curves indicates that Standard Prompting's performance is due to increased inference tokens generated: for a given FLOP level (red dotted line), benchmark score increases with more inference tokens generated. Conversely, Chain of Thought Prompting performance improvements are largely due to *increases in training FLOPs*, which are associated with more tokens generated. For a given FLOP level (black dotted line) for Chain of Thought, benchmark score only increases slightly with more inference tokens generated, but the IsoFLOP curves are more spaced out, indicated a larger gain from higher training FLOPs.

Following [33], we calculate a "Train-Equivalent FLOPs" metric for each model. Within each model family, we fit a linear regression of average benchmark performance on log-train-compute. This regression allows us to back out the estimated number of additional FLOPs it *would* take to train a given model to reach the same performance as a Chain of Thought variant.

In Figure 8, we examine the relationship between additional inference tokens generated per request (as a ratio to standard inference tokens) when using Chain of Thought (the

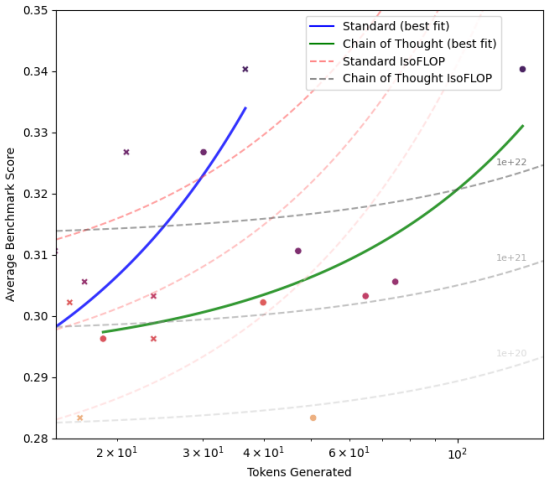

Fig. 7. Average Benchmark Score versus Inference Tokens Generated, for Standard Prompting and Chain of Thought Prompting. IsoFLOP curves are plotted for varying training FLOP levels. X's refer to Standard Prompting, and dots refer to Chain of Thought prompting, with points colored darker for larger training FLOPs.

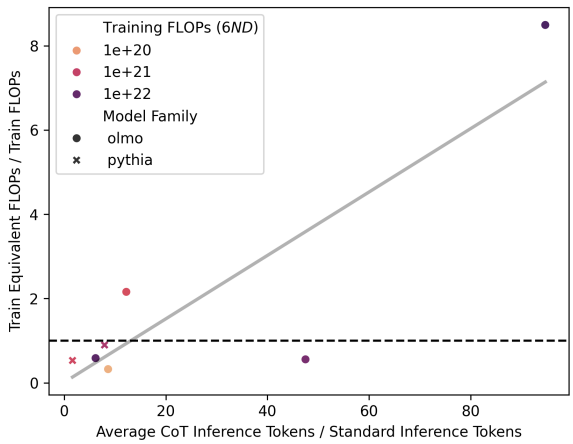

Fig. 8. Train-Equivalent Compute Ratio versus Average Inference Cost with CoT. As the cost of inference with CoT grows (a higher ratio of CoT tokens to standard prompting tokens), there may be weak positive correlation with larger improvements in train-equivalent FLOPs.

"real cost") versus the additional train compute that *would have been* necessary to achieve the same performance (the "alternative cost"), as a ratio to original train FLOPs. We find that increasing the number of chain of thought inference tokens, relative to original inference tokens, increases the train equivalent FLOPs: essentially, performing longer chain of thoughts allows you to achieve the results of a larger base model, controlling for model size.

We further develop this comparison between cost at inference time and cost at train time in Figure 9. Following the framing of [35], we fix a performance threshold and study the optimal allocation of compute across train and inference based on the volume of inference requests one expects a model will fulfill. Specifically, we focus on the Chain of Thought performance of the final OLMo-1B checkpoint as our threshold.

The standard approximation for inference FLOP costs in large language models is $2ND_{\text{inf}}$ FLOPs, where D_{inf} is the number of inference tokens generated for a given request. Using our estimated average CoT "generation ratios" from Table I, we can therefore model the inference tokens generated from one request using CoT as $\tilde{D}_{\text{inf}} = D_{\text{inf}}\beta_{\text{CoT}}$. With this approximation, we can calculate the lifetime expected FLOP cost of a model as the sum of its train FLOPs and the FLOPs per request times the expected number of lifetime requests. The results from this calculation applied to the final checkpoint of the OLMo-1B model are displayed in Figure 9. The figure dislays a large region, up to roughly 10 trillion lifetime token generations, wherein Chain of Thought prompting is a more cost-effective allocation of compute to achieve the same performance.

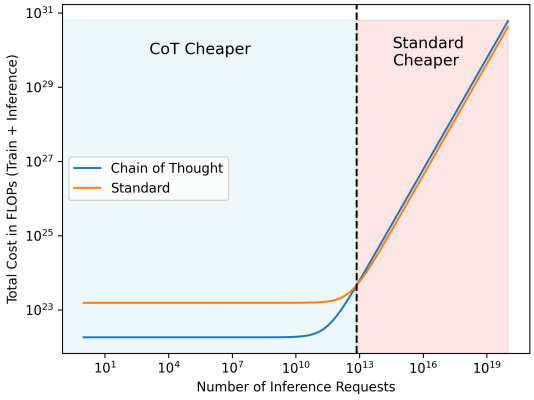

Fig. 9. Cost Optimality Regimes for Inference Strategies, by Expected Inference Volume (for OLMo-1B). Inference cost was determined using the model size and token generation performance from the final checkpoint of OLMo-1B.

V. Conclusion

In this paper, we propose and test a novel methodology to evaluate the performance-cost tradeoffs of large language model inference optimizations across model scales and families, extending previous work in [33, 35].

We adapt the log-likelihood task format to become suitable for use with Chain of Thought prompting by including rationale generation ("hybrid generation"). We find that this format adjustment worsens baseline benchmark performance across the board, with pronounced effects for larger models, despite the same approach for determining the final answer (log-likelihood of multiple choice answers). This suggests that rationales may add noise to the model's answer, resulting in less accurate answers. This effect may be limited to 1.4B and smaller models we examine for hybrid generation.

With this adapted task format, we evaluate the efficacy of Chain of Thought prompt strategies across 8 models, 1.4B and smaller, on 7 benchmark datasets. We reproduce a well-known finding that Chain of Thought prompting is effective primarily only for large train FLOP models. We also find support for a novel scaling behavior for CoT using intermediate model checkpoints from the OLMo model. Namely: that the benefits from the Chain of Thought optimization emerge far more slowly when scaling train compute by using more train tokens rather than by increasing model size.

We break down the observed inference scaling behavior for chain of thought and standard prompting by plotting IsoFLOP curves; we find that much of the inference scaling behavior for Chain of Thought is due to greater *training FLOPs*, which is associated with more inference tokens generated. In contrast, for Standard Prompting, generating more tokens is associated with a steeper benchmark score, and cannot be as well explained by simplying increasing the FLOPs of the underlying base model.

However, we also find that increasing Chain of Thought tokens allows a model to match the performance of a larger base model, even when controlling for model size. This supports the literature consensus that Chain of Thought can improve model reasoning and allow for the use of shallower, cheaper base models.

Based on this tradeoff between inference tokens, base model size, and overall cost, we leverage our extension of the Observational Scaling Law framework to create a general method for identifying compute optimality regimes for inference-time optimizations based on expected lifetime inference demand. In a regime with few expected inference tokens (perhaps because one expects to quickly cycle to newer language models), it can be more cost-effective to spend extra compute at inference-time. Because log-train-compute correlates linearly with average model performance on downstream benchmark tasks, we can compute a "Train-Equivalent FLOP" metric that directly links an inference-optimized model to the additional compute necessary to train a base model to the same performance. With this, we can directly and analytically characterize the optimal regime in which it is more cost-effective to use an inference optimization rather than spend more train compute.

Finally, we evaluate the Ruan et al Observational Scaling Laws with our benchmark data. We find that Observational Scaling Laws are strongly dependent on the presence of a code benchmark, which our data lacks; we confirm this by ablating the code benchmark from the original Ruan et al data, and we find that only one general reasoning PC remains, which closely matches our own results. This may imply that observational scaling law method is weaker and more dependent on benchmark variety than initially apparent.

Opportunities for future work include adding additional benchmarks, including coding benchmarks; training with more models and larger model sizes, especially $>$12B; and investigating a variety of inference-time optimizations, including search methods.

Acknowledgment

Thanks to Nikhil Sardana and Jacob Portes of Mosaic Research for their detailed guidance on language model evaluation and inference-aware scaling laws. A further thanks to Kaoutar El Maghroui for her mentorship throughout Columbia's COMS6998: High-Performance Machine Learning course. We would also like to thank Google and the TPU Research Cloud for providing free TPU credits and resources.

References

[1] Marah Abdin et al. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. Aug. 2024. DOI: 10.48550/arXiv.2404.14219. arXiv: 2404.14219 [cs]. (Visited on 12/21/2024).

[2] Marah Abdin et al. *Phi-4 Technical Report*. Dec. 2024. DOI: 10.48550/arXiv.2412.08905. arXiv: 2412.08905 [cs]. (Visited on 12/21/2024).

[3] Stella Biderman et al. "Pythia: A suite for analyzing large language models across training and scaling". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 2397–2430.

[4] Bradley Brown et al. *Large Language Monkeys: Scaling Inference Compute with Repeated Sampling*. Sept. 2024. arXiv: 2407.21787 [cs]. (Visited on 10/23/2024).

[5] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. eprint: arXiv:2005.14165.

[6] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

[7] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. *A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations*. 2024. arXiv: 2308.06767 [cs.LG]. URL: https://arxiv.org/abs/2308.06767.

[8] Leshem Choshen, Yang Zhang, and Jacob Andreas. *A Hitchhiker's Guide to Scaling Law Estimation*. 2024. eprint: arXiv:2410.11840.

[9] Peter Clark et al. *Think You Have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. Mar. 2018. DOI: 10.48550/arXiv.1803.05457. arXiv: 1803.05457 [cs]. (Visited on 12/21/2024).

[10] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. Nov. 2021. DOI: 10.48550/arXiv.2110.14168. arXiv: 2110.14168 [cs]. (Visited on 12/21/2024).

[11] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. June 2022. DOI: 10.48550/arXiv.2205.14135. arXiv: 2205.14135 [cs]. (Visited on 12/21/2024).

[12] Yao Fu et al. *Complexity-Based Prompting for Multi-Step Reasoning*. 2022. arXiv: 2210.00720.

[13] Leo Gao et al. *A framework for few-shot language model evaluation*. Version v0.4.3. July 2024. DOI: 10.5281/zenodo.12608602. URL: https://zenodo.org/records/12608602.

[14] Amir Gholami et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. June 2021. DOI: 10.48550/arXiv.2103.13630. arXiv: 2103.13630 [cs]. (Visited on 12/21/2024).

[15] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. Nov. 2024. DOI: 10.48550/arXiv.2407.21783. arXiv: 2407.21783 [cs]. (Visited on 12/21/2024).

[16] Dirk Groeneveld et al. "OLMo: Accelerating the Science of Language Models". In: *Preprint* (2024).

[17] Yuling Gu et al. *OLMES: A Standard for Language Model Evaluations*. June 2024. DOI: 10.48550/arXiv.2406.08446. arXiv: 2406.08446. (Visited on 11/28/2024).

[18] Sylvain Gugger et al. *Accelerate: Training and inference at scale made simple, efficient and adaptable.* https://github.com/huggingface/accelerate. 2022.

[19] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. Jan. 2021. DOI: 10.48550/arXiv.2009.03300. arXiv: 2009.03300. (Visited on 11/28/2024).

[20] John Hewitt, Christopher D. Manning, and Percy Liang. *Truncation Sampling as Language Model Desmoothing*. 2022. arXiv: 2210.15191.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. Mar. 2015. DOI: 10.48550/arXiv.1503.02531. arXiv: 1503.02531 [stat]. (Visited on 12/21/2024).

[22] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. Mar. 2022. arXiv: 2203.15556 [cs]. (Visited on 10/23/2024).

[23] Itay Hubara et al. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". In: *Journal of Machine Learning Research* 18.187 (2018), pp. 1–30. ISSN: 1533-7928. (Visited on 12/21/2024).

[24] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. Jan. 2020. DOI: 10.48550/arXiv.2001.08361. arXiv: 2001.08361. (Visited on 11/16/2024).

[25] Apoorv Khandelwal et al. $100Kor100Days : Trade-offswhenPre-TrainingwithAcademicResources$. 2024. eprint: arXiv:2410.23261.

[26] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2022. arXiv: 2205.11916.

[27] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. Sept. 2023. DOI: 10.48550/arXiv.2309.06180. arXiv: 2309.06180 [cs]. (Visited on 12/21/2024).

[28] Woosuk Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.

[29] Jeffrey Li et al. *DataComp-LM: In search of the next generation of training sets for language models*. 2024. eprint: arXiv:2406.11794.

[30] Xiang Lisa Li et al. *Contrastive Decoding: Open-ended Text Generation as Optimization*. 2022. arXiv: 2210.15097.

[31] Stephanie Lin, Jacob Hilton, and Owain Evans. *TruthfulQA: Measuring How Models Mimic Human Falsehoods*. May 2022. DOI: 10.48550/arXiv.2109.07958. arXiv: 2109.07958 [cs]. (Visited on 12/21/2024).

[32] Niklas Muennighoff et al. *Crosslingual Generalization through Multitask Finetuning*. May 2023. DOI: 10.48550/arXiv.2211.01786. arXiv: 2211.01786 [cs]. (Visited on 12/21/2024).

[33] Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. *Observational Scaling Laws and the Predictability of Language Model Performance*. Oct. 2024. arXiv: 2405.10938 [cs]. (Visited on 10/25/2024).

[34] Keisuke Sakaguchi et al. *WinoGrande: An Adversarial Winograd Schema Challenge at Scale*. Nov. 2019. DOI: 10.48550/arXiv.1907.10641. arXiv: 1907.10641 [cs]. (Visited on 12/21/2024).

[35] Nikhil Sardana et al. *Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws*. July 2024. arXiv: 2401.00448. (Visited on 10/14/2024).

[36] Ao Shen, Zhiquan Lai, and Dongsheng Li. "Exploring Quantization Techniques for Large-Scale Language Models: Methods, Challenges and Future Directions". In: *Proceedings of the 2024 9th International Conference on Cyber Security and Information Engineering*. ICCSIE '24. New York, NY, USA: Association for Computing Machinery, Dec. 2024, pp. 783–790. ISBN: 9798400718137. DOI: 10.1145/3689236.3695383. (Visited on 12/20/2024).

[37] Charlie Snell et al. *Scaling LLM Test-Time Compute Optimally Can Be More Effective than Scaling Model Parameters*. Aug. 2024. arXiv: 2408.03314 [cs]. (Visited on 09/19/2024).

[38] Zayne Sprague et al. *To CoT or not to CoT? Chain-of-thought helps mainly on math and symbolic reasoning*. 2024. arXiv: 2409.12183.

[39] Gemma Team. "Gemma". In: (2024). DOI: 10.34740/KAGGLE/M/3301. URL: https://www.kaggle.com/m/3301.

[40] Qwen Team. *Qwen2.5: A Party of Foundation Models*. Sept. 2024. URL: https://qwenlm.github.io/blog/qwen2.5/.

[41] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. Feb. 2023. arXiv: 2302.13971. (Visited on 10/23/2024).

[42] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[43] Ante Wang et al. *Fine-Grained Self-Endorsement Improves Factuality and Reasoning*. 2024. arXiv: 2402.15631.

[44] Xuezhi Wang and Denny Zhou. *Chain-of-Thought Reasoning Without Prompting*. 2024. arXiv: 2402.10200.

[45] Xuezhi Wang et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2022. arXiv: 2203.11171.

[46] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Jan. 2023. DOI: 10.48550/arXiv.2201.11903. arXiv: 2201.11903 [cs]. (Visited on 12/13/2024).

[47] Sean Welleck et al. *From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models*. Nov. 2024. DOI: 10.48550/arXiv.2406.16838. arXiv: 2406.16838 [cs]. (Visited on 12/21/2024).

[48] An Yang et al. "Qwen2 Technical Report". In: *arXiv preprint arXiv:2407.10671* (2024).

[49] Michihiro Yasunaga et al. *Large Language Models as Analogical Reasoners*. 2023. arXiv: 2310.01714.

[50] Rowan Zellers et al. *HellaSwag: Can a Machine Really Finish Your Sentence?* May 2019. DOI: 10.48550/arXiv.1905.07830. arXiv: 1905.07830 [cs]. (Visited on 12/21/2024).

[51] Dan Zhang et al. *ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search*. 2024. arXiv: 2406.03816.

[52] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2024. arXiv: 2303.18223 [cs.CL]. URL: https://arxiv.org/abs/2303.18223.

[53] Huaixiu Steven Zheng et al. *Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models*. 2023. arXiv: 2310.06117.

[54] Denny Zhou et al. *Least-to-Most Prompting Enables Complex Reasoning in Large Language Models*. 2022. arXiv: 2205.10625.

VI. Appendix

A. Evaluation Hardware

For evaluation, we used a variety of accelerators, including RTX 4090, V100, A100 PCIe, H100 SXM, and TPU v4-8, using Columbia's Terremoto computing cluster, Google Cloud Platform, TPU Research Cloud, and third-party compute providers (Hyperbolic AI). We explored evaluations on single-node multi-GPU, multi-node multi-GPU, single-node multi-TPU, and multi-node multi-TPU settings, although the vast majority of evaluations were done on single-node multi-GPU settings. In addition to vLLM, we used the Accelerate package [18] to enable multi-GPU training.

B. Free-Form Generation Performance

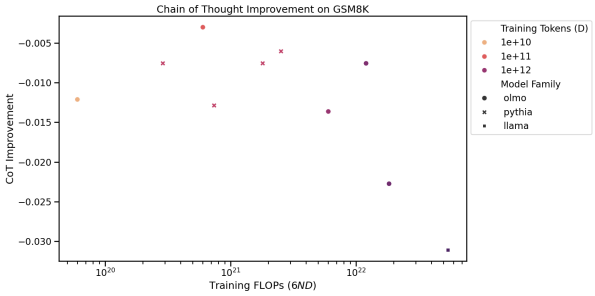

Fig. 10. Model Performance on Chain of Thought with Free-Form Generation Format. Accuracy scores across 7 tasks are averaged to yield one overall score, which is subtracted from baseline performance with log-likelihood format. Across the board worse performance, even with the benefits of CoT, indicates the comparative difficulty of the free-form generation format compared to log-likelihood.