Reasoning in Token Economies: Budget-Aware Evaluation of LLM Reasoning Strategies

Junlin Wang[1*] **, Siddhartha Jain** [3†] **, Ben Athiwaratkun** [4] **, Dejiao Zhang** [2] **, Baishakhi Ray** [2]**, Varun Kumar** [2]

[1]Duke University [2]AWS AI Labs [3]Nvidia [4]Together AI

Abstract

A diverse array of reasoning strategies has been proposed to elicit the capabilities of large language models. However, in this paper, we point out that traditional evaluations which focus solely on performance metrics miss a key factor: the increased effectiveness due to additional compute. By overlooking this aspect, a skewed view of strategy efficiency is often presented. This paper introduces a framework that incorporates the compute budget into the evaluation, providing a more informative comparison that takes into account both performance metrics and computational cost. In this budget-aware perspective, we find that complex reasoning strategies often don't surpass simpler baselines purely due to algorithmic ingenuity, but rather due to the larger computational resources allocated. When we provide a simple baseline like chain-of-thought self-consistency with comparable compute resources, it frequently outperforms reasoning strategies proposed in the literature. In this scale-aware perspective, we find that unlike self-consistency, certain strategies such as multi-agent debate or Reflexion can become worse if more compute budget is utilized.

1 Introduction

The arena of large language models (LLMs) such as GPT-4 (OpenAI, 2023; Touvron et al., 2023; Team, 2023; Jiang et al., 2023a) has seen a proliferation of diverse reasoning strategies. However, comparing these strategies fairly and comprehensively has proven to be a challenging task due to their varied computational requirements. For instance, strategies like tree of thoughts (ToT) necessitate branching out into multiple sequences and incorporating self-evaluation, making them more compute-intensive than others. Therefore, an evaluation framework that only accounts for performance metrics may miss crucial practical factors such as computational cost.

In this paper, we propose the inclusion of the compute budget into the performance measurement of different reasoning strategies. This budget-aware comparison yields a more balanced perspective on the effectiveness of reasoning strategies, accounting for both the quality of the output and the computational resources expended.

Our empirical research uncovers a significant correlation between the performance and the compute budget. We find that a straightforward baseline strategy, chain-of-thought reasoning coupled with self-consistency, can be remarkably competitive. When scaled to match the compute resources of more sophisticated methods such as Multi-Agent Debate (MAD) (Liang et al., 2023), Reflexion (Shinn et al., 2023), Plan and Solve (Wang et al., 2023), Least to Most Prompting (Zhou et al., 2022), Progressive Hint Prompting (Zheng et al., 2023), this baseline strategy often outperforms them in achieving the best trade-off between performance and budget. We further investigate the reasons behind the gap from simple CoT SC and other reasoning strategies by providing both empirical and theoretical evidence.

Then we scrutinize the influence of two specific types of budgets on performance: (1) the answer generation budget, and (2) the evaluation budget. The success of reasoning strategies that leverage self-evaluation is model/dataset-dependent and strongly correlated with calibration.

This work provides a robust framework for comparing a wide array of reasoning strategies and illuminates the significance of self-evaluation in these models. We hope this sets the stage for more focused research on efficient budget utilization and paves the way for the development of even more effective reasoning strategies.

[*]Work conducted during an internship at Amazon
[†]Work conducted while at Amazon

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 19916–19939

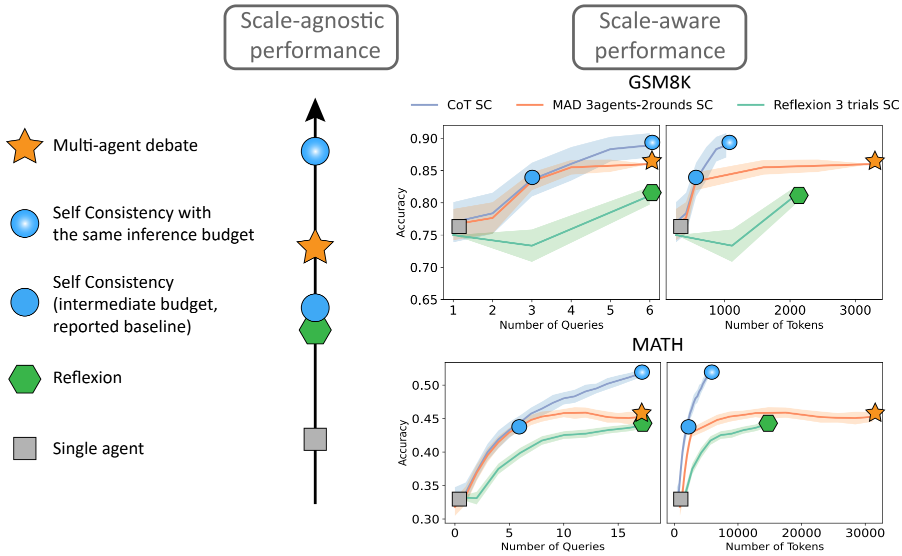


Figure 1: (1) Comparison of reasoning approaches multi-agent debate (MAD) against the SC baseline, considering both scale-agnostic and scale-aware evaluation, with published scores and our reproductions on the GSM8K and MATH dataset. The scale-aware evaluation furnishes more comprehensive insights into the influence of scale on reasoning strategies and offers a fairer method of comparison. (2) The scale-aware comparison between Reflexion and SC also illustrates the artifact of scale on performance. For both datasets, we show both budgets, the number of total tokens, and the number of queries. All results were obtained from GPT-3.5.

Concretely, our contributions are

- We introduce a budget-aware evaluation framework spanning three dimensions: queries, tokens, and monetary cost, advocating for the token-based metric as the most holistic. This metric adeptly captures both the latency and financial implications of computational tasks.
- We present a comprehensive evaluation of seven LLM reasoning strategies across five datasets using five models including GPT-4. Our analysis reveals that traditional evaluation metrics often overlook a critical aspect: the performance gains achievable through additional computational resources. This observation is strongly supported by CoT SC matches or even exceeds more complex strategies in effectiveness.
- We explore the dynamics of reasoning strategies, highlighting that MAD underperforms as diversity diminishes with each round. Conversely, Self-Consistency excels due to the independence of samples boosting diversity and its effectiveness in scenarios where the likelihood of being correct exceeds 50%.
- We conduct ablation studies on ToT and Reflexion by segregating the budget into answer generation and evaluation budgets. We found that self-evaluation is promising at increasing performance while being cost-effective but currently LLM can't self-evaluate well.

2 Related Work

2.1 Reasoning strategies for LLMs

An early work in the area was to prompt the language model to generate its Chain-of-Thought (CoT) (Wei et al., 2022) which led to significant improvements in the model's problem-solving abilities. Later work has involved prompting the language model to come up with plans for solving problems (Jiang et al., 2023b), using CoT and ask the model to critique and revise its solution (feedback) (Madaan et al., 2023; Scheurer et al., 2023; Chen et al., 2023a; Bai et al., 2022; Kim et al., 2023), generating multiple chain-of-thoughts and combining them using LLM (Yoran et al., 2023), setting up a tree search for chain-of-thought (*Tree of Thoughts* - ToT) (Yao et al., 2023), aggregating LLM generated feedback into guidelines that can improve future generation (Chen et al., 2023b), and using multiple LLMs as debating agents to refine a solution (Du et al., 2023). However, they are all evaluated on different datasets, and whether the baselines are computed or cost-matched is rarely considered. Notable exception is Huang et al.

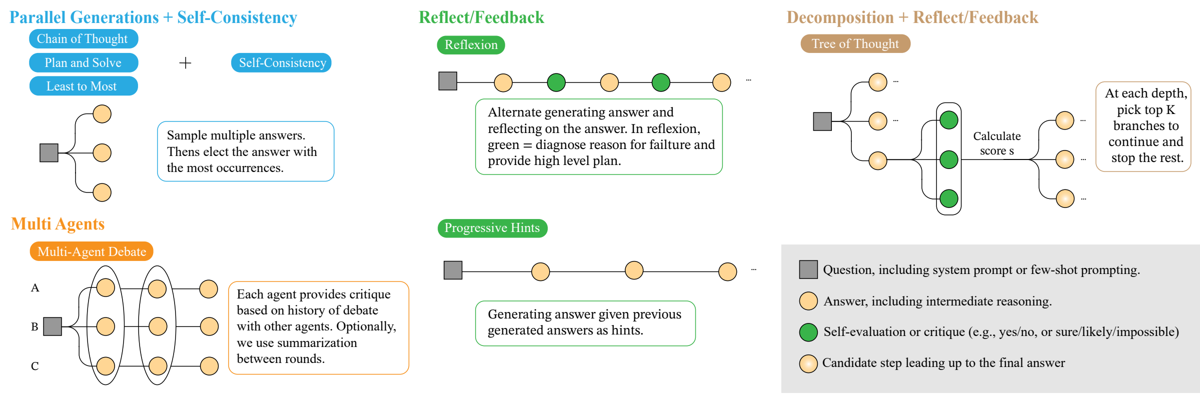


Figure 2: Overview of reasoning strategies. Green cell indicates question prompt, including system prompt and few-shot prompting. The orange cell indicates the answer. Blue cell indicates evaluation or critique.

(2023) where they found MAD is doing an unfair comparison to SC.

2.2 LLM output evaluation

There has been considerable work on evaluating the output of LLMs via ranker or self-evaluation. In Uesato et al. (2022); Yang et al. (2022), they train an evaluator for each step in a reasoning chain and rerank using the combined score. In Li et al. (2023), they weight the self-consistency by the trained verifier confidence. There has also been work recently on LLM to self-evaluate its own generations. In Bai et al. (2022), they use LLMs to do pairwise comparisons between generations achieving high accuracy. In Ling et al. (2023), self-consistency for every step is used to evaluate how to correct a deductive step is, but they failed at improving performance using that signal. Tian et al. (2023) examine multiple strategies for eliciting calibrated LLM self-evaluation. The self-refine (Madaan et al., 2023) approach uses LLMs to get detailed self-evaluation to improve the next generation. The Tree-of-Thoughts (Yao et al., 2023) paper uses LLM self-evaluation to rank which node to explore next. Our work conducts analysis on self-evaluation budget and access whether it is worth it to do self-evaluation.

3 Inference Budget of Reasoning Strategies

While the raw performance of different prompting or reasoning strategies for LLMs is a common topic, how different strategies perform when *budget-aware* is less well-studied. However, taking budget into account can be critical when using LLMs. In this section, we describe different usage scenarios that a user could be interested in and what budgetary metrics would be relevant to those scenarios.

3.1 Budget

We examine various budgetary metrics for LLMs. Given that the number of input and output tokens often feature prominently across these metrics, we designate them as n_I and n_O respectively.

API monetary cost is generally represented as $c = \alpha_1 \cdot n_I + \alpha_2 \cdot n_O$. Here, n_I and n_O correspond to the number of input and output tokens. The coefficients α_1 and α_2 are specific to the LLM API in use. It's worth noting that in scenarios involving parallel sampling of multiple outputs with a singular input, n_I is counted once.

Total number of tokens a straightforward metric, is described by $t = n_I + n_O$. This becomes pertinent when $\alpha_1 = \alpha_2$, which is true for many LLM APIs and is also reflective of the compute cost. Its simplicity ensures it doesn't inherently favor any specific model or API provider.

Number of queries of planned API calls can be a rough proxy for the budget. Such numbers can be determined before inference, which gives us rough guidance before action. Note that in case we want to sample multiple outputs from the LLM, we count those as *separate* queries.

4 A Critical Evaluation in Budget-Aware Environments

This section explores key components that can make reasoning strategies successful from the budget-aware perspective. First, we show that the inference budget is often overlooked but is one of the primary indicators of the success of a reasoning strategy. We show that from the budget-aware

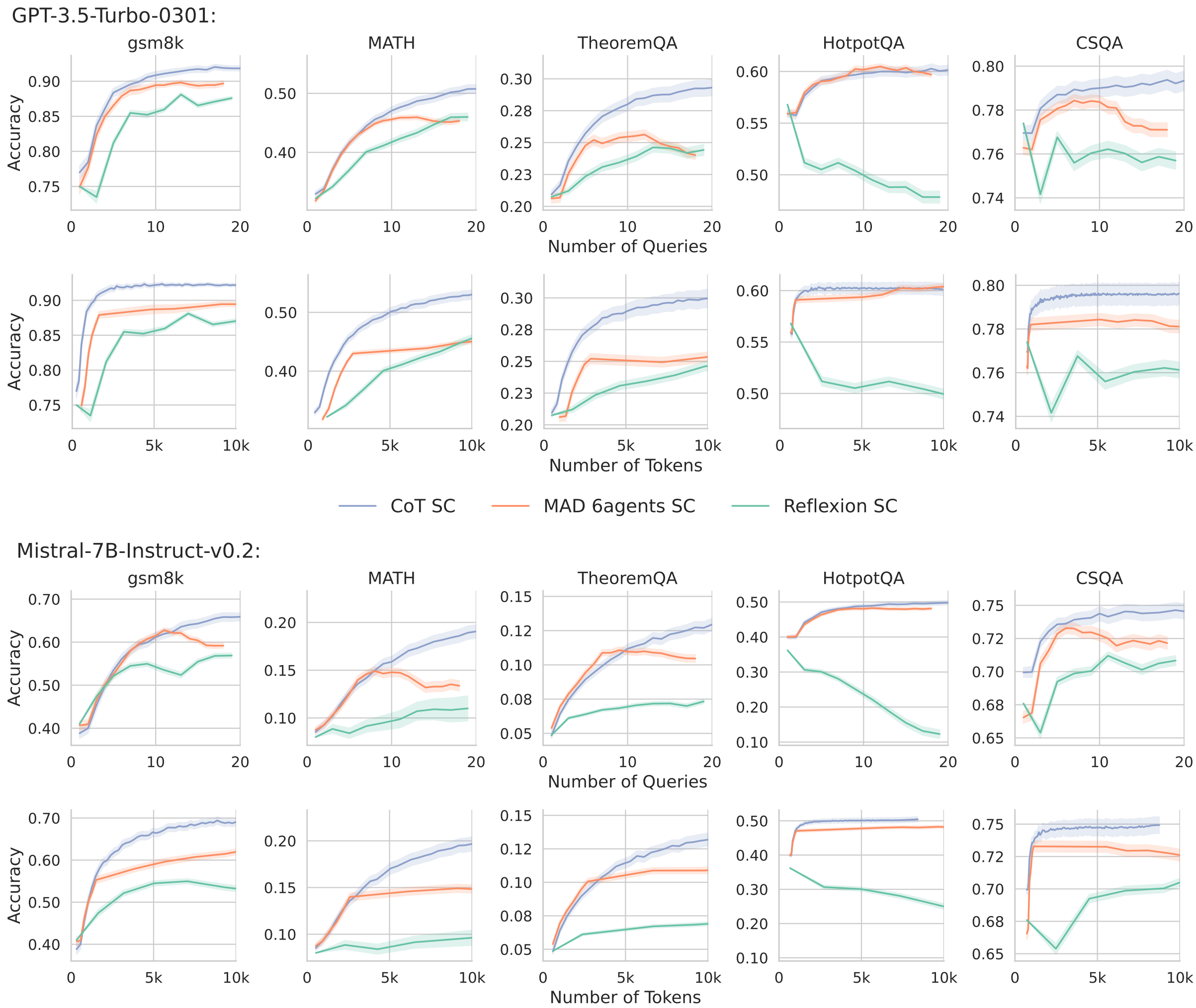


Figure 3: Performance@Number of Queries and Performance@Number of Tokens Plots for all 5 datasets. All three methods CoT SC, MAD, and Reflexion are plotted on two models (more models in Appendix G). All experiments here are run until at least 10k tokens or 20 queries. CoT with SC consistently beat other reasoning strategies across all 5 datasets with significantly less budget. The budget difference is even more drastic when counting the number of tokens. The MAD result is shown non-round-wise.

evaluation perspective, CoT (or variants of it like Plan and Solve, Least to Most) self-consistency, for instance, is a strong baseline that can outperform or match many proposed reasoning strategies in the literature given the same level of budget.

Experiement Setup We use existing reasoning strategies in literature to perform this study, namely Multi-Agent Debate (MAD) (Liang et al., 2023), Reflexion (Shinn et al., 2023), Plan and Solve (Wang et al., 2023), Least to Most Prompting (Zhou et al., 2022), Progressive Hint Prompting (Zheng et al., 2023), and Tree-of-Thoughts (Yao et al., 2023). We conducted our experiments across a diverse range of reasoning tasks, utilizing math reasoning datasets such as GSM8k (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and TheoremQA (Chen et al., 2023c), along with the commonsense reasoning task CSQA (Talmor et al., 2019), and multi-hop reasoning task HotpotQA (Yang et al., 2018) (see Appendix A.1). Additionally, we performed an in-depth analysis of the puzzle game Game24 (Yao et al., 2023). For models, we use Mistral-7B-Instruct, LLaMA-2-70b-chat, Mixtral-8x7B-Instruct, GPT-3.5, and GPT-4. (See Appendix A.1 for more details about model hyperparameters)

Metrics To provide a fair comparison, we control for the cost/budget of each reasoning strategy to have the same level of budget. Specifically, We demonstrate two budgetary metrics which built on what was discussed in Section 3.1:

1. **Performance@Number of Queries**: This

metric assesses accuracy across datasets for each reasoning strategy, controlling for the total number of queries allowed per sample question. Regardless of the specific actions taken within each query, all queries are treated equally. The performance is plotted as accuracy versus the number of queries, ranging from 1 to 20, allowing for a comparative analysis of how different strategies scale with additional computational steps.

2. **Performance@Number of Tokens**: This metric evaluates accuracy across datasets for each reasoning strategy, based on the total number of tokens allowed per sample question. We plot accuracy against the number of tokens, ranging from 0 to 10,000, for each query sample. This approach ensures the metric is not biased towards strategies with fewer but longer queries, providing a fair comparison of performance relative to computational resources used.

Each reasoning strategy may have different ways of increasing the budget depending on its design. For CoT SC, we increase the number of sampled CoT paths to increase queries/tokens. For MAD, we set the number of agents to six and increased the number of rounds up to three, maxing at 18 queries. For Reflxion, we increase the number of proposals of answers and reflections to increase the budget.

4.1 Inference budget unveils superiority of self-consistency baseline over MAD & Reflexion

We present that the observed improvements in performance for various reasoning methods may be strongly influenced by the use of a higher inference budget, rather than the intrinsic merit of the techniques themselves.

Results in Figure 3 elucidate the efficacy of reasoning techniques, including MAD and Reflexion, in contrast with the SC baseline. As illustrated in Figure 1 and 3, aligning the inference budgets reveals that the perceived advantages of novel strategies diminish. The SC baseline generally surpasses more complex methods with equal budgets on all datasets, except HotpotQA where it remains competitive. Reflexion consistently underperforms the other strategies, which we will explore further. Solely depending on scale-independent evaluations, as seen in previous studies, can yield incomplete or misleading results.

4.2 Plan and Solve, Least to Most, Progressive Hints performance gain primarily from increased budget

In this study, we assess the efficacy of the proposed budget-aware metrics across three additional reasoning strategies: Plan and Solve, Least to Most, and Progressive Hints, as depicted in Figure 4. Among these strategies, chain-of-thought self-consistency (CoT SC) emerges as a competitive approach in most scenarios. Plan and Solve, when coupled with self-consistency, surpasses CoT SC on HotpotQA. It's pertinent to recognize Plan and Solve and Least to Most as specialized iterations of the CoT approach. Specifically, Plan and Solve directs the LMs to strategize prior to resolving the query, whereas Least to Most deconstructs the question before answering. Thus, both strategies with SC can be conceptualized as nuanced versions of CoT SC. Conversely, Progressive Hints, which leverages sequential answers as cues for subsequent questions, exhibit the least effective performance. This comparative analysis underscores that the observed improvements in performance are primarily attributable to increased budget allocations rather than the inherent advantages of the methodologies. Evaluations on more datasets and types of budget as well as details about each strategy can be found in Appendix G.2.

4.3 Tree-of-Thoughts is competitive with a caveat

We evaluated the Tree-of-Thoughts strategy in a budget-aware manner on the logical game Game of 24. Notable discrepancies emerged in the behavior of the model when transitioning to GPT-4 and we modified our budget-aware metric slightly to further scrutinize GPT-4.

A strong model is needed to perform better than baseline In Figure 5, we show the performance of GPT-3.5 with the Tree-of-Thoughts reasoning strategy on Game of 24[1]. The performance of Tree-of-Thoughts lags that of a simple SC by a considerable margin. This is in stark contrast to the GPT-4 results with Tree-of-thoughts where CoT SC plateau very early and Tree-of-Thoughts beats it by a big margin, even when we account for the budget

[1] We used a modified thought evaluation prompt for GPT-3.5 that gave much better results than the default one

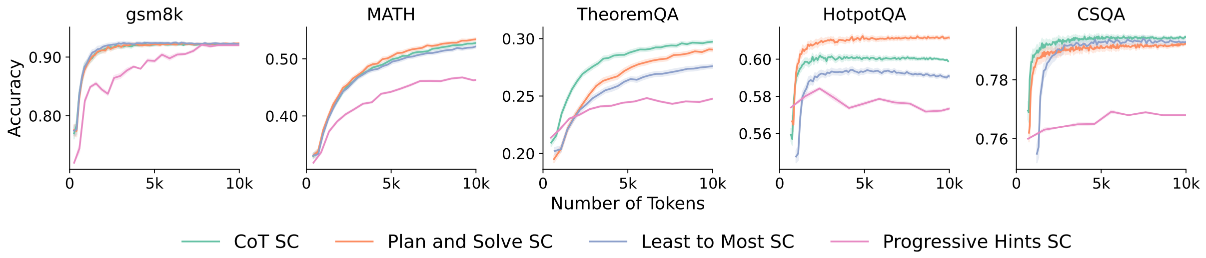


Figure 4: GPT-3.5-0301 Performance@Number of Tokens for all 5 datasets using three other strategies: Plan and Solve, Least to Most, Progressive Hints. More details at Appendix G.2.

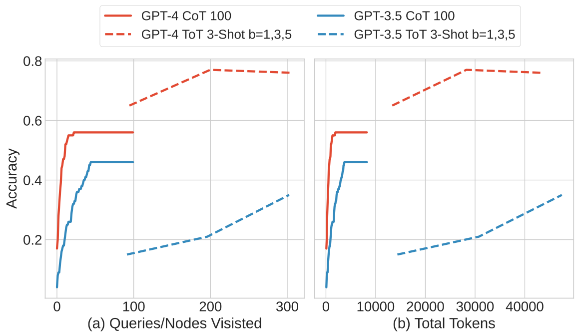


Figure 5: ToT vs. CoT SC for both GPT-3.5 and GPT-4 on Game of 24. The dotted lines represent the performance of ToT. For ToT, results for three settings are included. CoT SC results are on 100 samples.

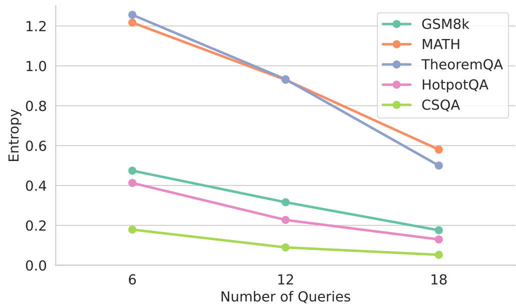


Figure 6: The diversity of the answers proposed by GPT-3.5 of MAD for each round goes down.

(query or token). However, Tree-of-Thoughts requires a significant budget commitment to deliver such a performance. On weaker models than GPT-4, it is still better to use CoT SC which outperforms ToT by a considerable margin (Figure 5).

5 What Makes Reasoning Strategies Work

We provide a detailed analysis of reasoning strategies. Specifically, we examine the causes of the performance gaps identified in our budget-aware evaluations among strategies such as MAD, Reflexion, and self-consistency in Section 5.1. In Section 5.2, we analyze the components of tree-of-thoughts. Finally Section 5.4 assesses the role of self-evaluation in the reasoning loop.

5.1 Reasoning strategies do not benefit equally from higher inference budget

The budget-aware perspective clearly guides which reasoning strategies are viable. A strategy is deemed effective only if it outperforms a baseline with an equivalent budget; otherwise, the additional cost isn't justified if the baseline achieves better results considering FLOPs, latency, monetary cost, or other relevant metrics. This raises the question of whether continuously increasing the budget can maximize capabilities.

As seen in Figure 3, we find that the CoT SC exhibits a smooth increase in scores with budget. However, such a trend does not always hold. For instance, with MAD, an augmented inference budget eventually experiences a performance plateau. For the MAD setting with 6 agents, the graph for MAD and CoT SC overlaps up to six queries. After six queries, the MAD strategy switches to the second round where the performance gain noticeably lessens compared to self-consistency. But, the amount of tokens required for each subsequent round increases drastically since previous conversations are encoded. The lowered performance may arise because subsequent rounds of MAD may incite a cascading effect of cumulative mistakes, or snowballed hallucinations Zhang et al. (2023).

5.1.1 Dependent sampling can hurt response diversity

Multi-agent debate conditions on the previous round's answers to sample new answers. We posit another reason MAD performs worse is due to reduced response diversity, hence more likely to tunnel on the wrong answer. To show this, we compared the entropy of the solutions generated at each

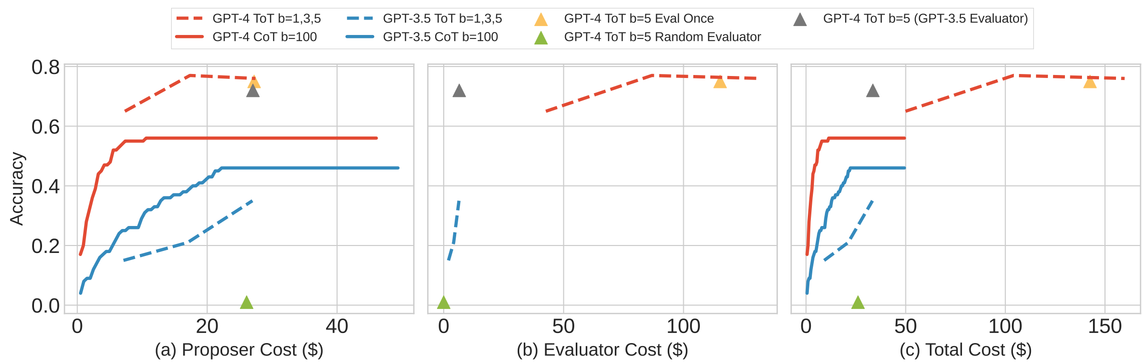


Figure 7: Thought proposer and thought evaluator budget on Game of 24.

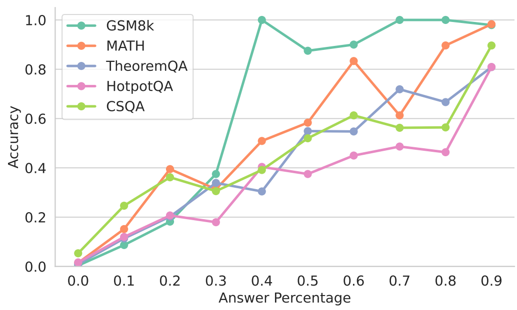


Figure 8: Calibration result binned by answer percentages. If an answer appears more times within all the samples, that answer is more likely to be correct.

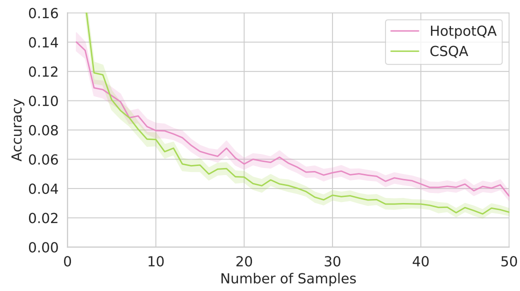


Figure 9: SC making things worse on QA problems. We selected a subset of problems where the correct answer is not the majority. For this subset, the performance decreases with more samples.

round for MAD vs. SC. As shown in Figure 6, the entropy consistently declines for MAD as each round suggesting exactly the kind of cascading effect we hypothesized. By contrast, CoT SC does not suffer such negative consequences and even increases its solution diversity since the responses are generated independently.

5.1.2 Effectiveness of independent sampling with chain-of-thought prompting

Next, we outline a framework that helps explain what makes self-consistency successful. We first empirically verified that the higher the occurrence of an answer, the more likely it is the correct answer (Figure 8). Self-consistency can capitalize on this and improve performance with more budget.

We model the answer generation process by LMs as a binomial distribution where each problem has an inherent probability p_i of being answered correctly. This analysis reveals several insights:

1. **Convergence**: The probability of a correct majority vote converges to 0 or 1 as the number of trials increases, depending on whether the probability of a correct answer p_i is less or greater than 0.5.
2. **Speed of Convergence**: Convergence is fast for extreme values of p_i (closer to 1 or 0), but slow if p_i is near 0.5.
3. **Distribution of Correctness**: By placing a prior on p_i (for instance, with a beta distribution), the aggregate score over the entire dataset converge to non-extreme values, resembling the behavior observed in our results.

That is, self-consistency performance increases smoothly over time is due to the artifact of a model consistently answering plausible answers that tend to be more correct than not. The alternative can happen otherwise (Figure 9). In Appendix C, we detail the analysis with extension to a multinomial setting with Dirichlet priors.

5.2 Complex strategies like Tree-of-thoughts scale with inference with caveats

In this section, we investigate the factors that contribute to the enhanced performance of the tree-of-thoughts strategy compared to CoT SC. ToT

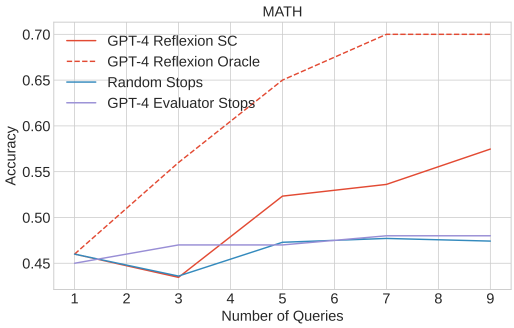


Figure 10: Ablation Study of the effect of evaluator on Reflexion with GPT-4.

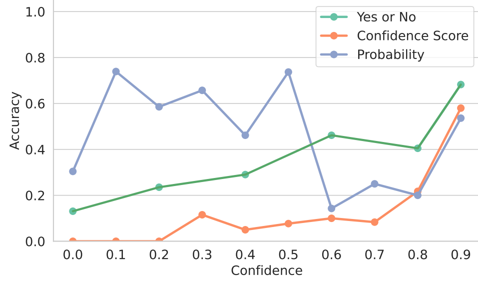


Figure 11: Calibration result for the math reasoning datasets. Three different self-evaluation methods are calibrated here.

mainly has two components: a proposer and a self-evaluator. The proposer proposes intermediate steps or answers and the evaluator decides whether to prune or continue on current branches. Hence we further divide the budget into the **proposer budget** and the **evaluator budget**. We aim to answer questions like how much of the performance can be attributed to self-evaluation ability.

For the ablation study, we compare four setups for tree-of-thoughts on the Game of 24: 1) The standard ToT where we use GPT-4 to evaluate the new thoughts; 2) The standard tree-of-thoughts strategy except we now do an evaluation only once as opposed to three times; 3) Using a weaker model (GPT-3.5) as the evaluator while using GPT-4 as the proposer; 4) Random evaluator, where we randomly select the subset of thoughts to prune.

Evaluator quality has a non-trivial impact As observed in Figure 7, a random evaluator leads to a very steep performance drop for ToT for both best@k as well as total accuracy. Results imply that an evaluator has a non-trivial impact. Evaluation is done only once per thought as opposed to multiple times also leads to performance drops.

Cost-efficiency of evaluator Using a weaker evaluator like GPT-3.5 allows us to maintain most of the performance cost-effectively. For instance, employing GPT-4 as the proposer and GPT-3.5 as the evaluator for 100 instances of Game24 costs \$33.53 with 72% accuracy. In contrast, using GPT-4 as both proposer and evaluator raises the cost nearly fivefold to \$159.87, with only a slight increase in accuracy to 76%.

More effective use of budget for proposer Employing a GPT-3.5 proposer with a GPT-4 evaluator resulted in markedly lower accuracy (38%) compared to using GPT-4 for both roles (76%). While this discussion does not extensively explore the role of the proposer, since most reasoning strategies involve one, we emphasize its importance. For further ablation results, refer to Appendix Table 3.

5.3 Reflexion does not scale that well unless with a good evaluator

In an experiment shown in Figure 10, we compared various Reflexion configurations: standard, with oracle, with self-consistency, with a random evaluator, and with a GPT-4 evaluator. Results indicate that while Reflexion with an oracle significantly outperforms the self-consistency model, Reflexion with a GPT-4 evaluator lags behind the self-consistency version. This underscores the vast difference between an ideal and a practical evaluator, suggesting substantial potential for improvement in LLM self-evaluation capabilities.

5.4 Self-evaluation is a promising budget-efficient improvement but is currently lacking

The previous sections on Tree-of-Thoughts and Reflexion have highlighted the crucial role of a strong evaluator in enhancing performance. When self-evaluation capabilities are lacking, reasoning strategies struggle to scale effectively with increased inference. Self-evaluation usually involves very few tokens generated since evaluation is short. This can be potentially very cost-effective since prefilling is cheaper and faster. In this section, we investigate further how a self-evaluator can benefit the reasoning process in a budget-aware setting and demonstrate why there may still be a long way to go.

Self-evaluation ability We first benchmarked three types of evaluations: *Yes or No*: model answers Yes/No on answer correctness; *Score 1-10*:

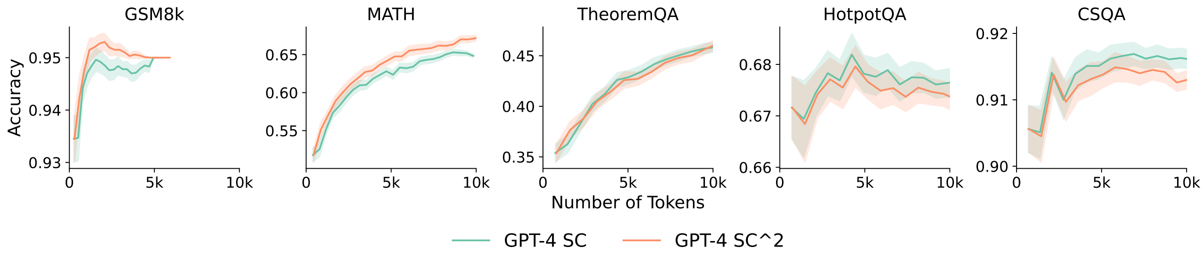


Figure 12: SC^2 with total tokens being the budget if caching is enabled.

Dataset	Correct Accuracy	Incorrect Accuracy	Total Accuracy
GSM8K	0.992	0.156	0.937
MATH	0.911	0.461	0.707
TheoremQA	0.945	0.232	0.547
HotpotQA	0.994	0.029	0.675
CSQA	0.987	0.06	0.901

Table 1: Self-evaluation accuracy on five datasets. Correct accuracy denotes self-evaluation accuracy for answers that turn out to be correct and vice versa. All numbers are obtained with GPT-4-0613.

model rates its confidence on a 1-10 scale; and *Probability between 0 to 1*: model rates its confidence on a 0 to 1 scale. Each method involves multiple evaluations, with the final confidence determined by averaging the scores. We found Yes or No to be the most calibrated as shown in Figure 11. More details and results in Appendix D.

Table 1 shows the self-evaluation accuracy for GPT-4 for multiple datasets. The self-evaluation accuracy turns out to be dependent on the dataset. On harder tasks like TheoremQA, GPT-4's accuracy is close to random. This means LLMs have a long way to go before they are reliable evaluators.

Self-Confident Self-Consistency (SC^2) As an investigation of using self-evaluation to improve reasoning procedure, we propose to weigh the SC by the confidence the model has in its answer, derived from self-evaluation. We call this score the *Self-Confident Self-Consistency* (SC^2) score. We showed that SC^2 beats self-consistency on GSM8k and MATH while fall behind on the other three as shown in Figure 12.[2] This shows that although theoretically, self-evaluation is promising (shown with oracle results in Figure 10), it is still lacking in practice due to low accuracy.

6 Conclusion

In this paper, we examined the performance of seven reasoning strategies on the often overlooked metric of budget. We used budget metrics of queries and tokens to reflect various ways LLMs are used (LLM APIs or self-host). We identified self-evaluation as an important aspect of many reasoning strategies and analyzed different prompting strategies to have the model evaluate its generations. We then evaluated self-evaluation and found that although self-evaluation could be promising at improving performance while being cost-effective, current LLMs are mostly incapable of doing that. With the current popularity of reasoning strategies, we think this more balanced budget-aware metric is beneficial for the community and helps set the correct trajectory for future LLM research.

7 Limitations

Our goal in the paper was to highlight the importance of different aspects of the generation budget for LLMs that are often ignored in the recent spate of reasoning strategies for LLMs. To that end, we chose some representative reasoning strategies and evaluated them on some common reasoning tasks. However, due to both monetary and time constraints, we could not include even more reasoning strategies or tasks. A more exhaustive evaluation might reveal additional nuances which would be interesting to explore.

[2]More details can be found at Appendix D.3.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. 2023a. Improving code generation by training with natural language feedback. *arXiv preprint arXiv:2303.16749*.

Liting Chen, Lu Wang, Hang Dong, Yali Du, Jie Yan, Fangkai Yang, Shuang Li, Pu Zhao, Si Qin, Saravan Rajmohan, et al. 2023b. Introspective tips: Large language model for in-context decision making. *arXiv preprint arXiv:2305.11598*.

Wenhu Chen, Ming Yin, Max Ku, Yixin Wan, Xueguang Ma, Jianyu Xu, Tony Xia, Xinyi Wang, and Pan Lu. 2023c. Theoremqa: A theorem-driven question answering dataset. *Conference on Empirical Methods in Natural Language Processing*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, D. Song, and J. Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS Datasets and Benchmarks*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv: 2310.01798*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *arXiv preprint arXiv: 2310.06825*.

Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2023b. Self-planning code generation with large language model. *arXiv preprint arXiv:2303.06689*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. 2023. Deductive verification of chain-of-thought reasoning. *arXiv preprint arXiv:2306.03872*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

William Merrill and Ashish Sabharwal. 2023. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. 2021. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35.

Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Angelica Chen, Kyunghyun Cho, and Ethan Perez. 2023. Training language models with language feedback at scale. *arXiv preprint arXiv:2303.16755*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Gemini Team. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv: 2312.11805*.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *arXiv preprint arXiv:2305.14975*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv: 2307.09288*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, R. Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *Annual Meeting of the Association for Computational Linguistics*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. *arXiv preprint arXiv:2205.12443*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought. *arXiv preprint arXiv:2304.13007*.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2023. How language model hallucinations can snowball. *CoRR*, abs/2305.13534.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv: 2304.09797*.

Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, D. Schuurmans, O. Bousquet, Quoc Le, and E. Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *International Conference on Learning Representations*.

A Model/Dataset Details

A.1 Datasets

Here we describe the datasets we used in our experiments.

GSM8K GSM8K consists of 8.5K grade school math problems. There are 7.5K examples in the training set and 1K in the testing set. Each problem is expressed in natural language and usually involves multi-hop reasoning.

MATH MATH dataset collects 12.5K (7.5K training, 5K testing) high-school level competitive math problems in natural languages. This dataset is considerably harder than GSM8K.

TheoremQA Theorem QA annotated 800 QA pairs covering over 300 theorems spanning across Math, EE&CS, Physics and Finance. We focus on math reasoning hence we only used the subset that covers math problems which contains 442 questions. This dataset is even harder than GSM8K since these questions are college-level and involve using theorems.

CSQA CSQA sourced commonsense reasoning questions from crowd workers based on ConceptNet. It has a total of 12,247 examples (9741, 1140,1140 for the size of train, dev, and test set respectively).

HotpotQA HotpotQA collects 113K question-answer pairs that require multi-hop reasoning. There are 7,405 pairs in the test set.

Game of 24 Game of 24 is a mathematical reasoning challenge, where the goal is to use 4 numbers and 4 arithmetic operations (+-*/) to obtain 24. (Yao et al., 2023) collects 100 problems from 4num.com which are ranked 901-1000 (it is ranked from easy to hard, so these 100 are relatively hard).

For each dataset above, we randomly sampled 100 samples from the test set for all of our experiments. For Game of 24, since there are exactly 100 problems, we just use the same 100 problems as in (Yao et al., 2023).

A.2 Model Hyperparameters

Since we want to maintain the diversity of reasoning processes, most of the results are obtained with a temperature of 1 for GPT-3.5 and GPT-4. In our preliminary study, we also tested with a temperature of 0.7 and 0.5 and observed the same conclusion. The GPT-3.5 version we used is 0301. The GPT-4 version we used is 0613.

For open-source models, we use a temperature of 1 as well.

A.3 Inference

For all the GPT models, we use OpenAI API. For all the open-source models, we either use Together Endpoint[3] or vllm[4] for inference.

B Additional Result for Budget-aware Performance Metrics

B.1 Budget Metrics on All Datasets

Budget metrics on all datasets are shown in Figure 3 and Appendix G.

B.2 Detailed description of Reasoning strategies

1. **Tree of thoughts** generates a search tree to search through possible chains of thought. It maintains a chain of thought. At each node in the tree, it generates a list of candidate thoughts to be added to the chain and does an evaluation to select the next thought to add. It concludes by generating an answer at a leaf node of the tree. The path in the tree from the root to the leaf node forms a single chain of thought, with each node corresponding to a single thought. If the answer is deemed incorrect (as per another evaluator), it backtracks to a previous node of the tree (unwinding the chain of thought along the way) and selects the next thought out of the candidate list of thoughts to add to the chain of thought.

B.3 Self-Evaluation with CoT

All of our self-evaluations are done without CoT. For both evaluation calibration and weighted confidence self-consistency, we only generated one token "yes" or "no" or one number. One may be interested in whether CoT can improve the self-evaluation performance and further boost the results. We tested this by extracting 160 CoT answers from 80 questions from GPT-3.5, where each question we extract 1 correct CoT answer and 1 incorrect CoT answer. We then compared the performance of direct evaluation versus CoT

[3] https://api.together.ai/playground/chat

[4] https://github.com/vllm-project/vllm

then evaluation. For GPT-3.5-turbo-0301, the accuracy increased from 50.625% to 54.375%. For GPT-4-0613, the accuracy increased from 78.75% to 79.375%. For GPT-4 the benefit from CoT is very mariginal and we concluded that it is not worth the extract cost from CoT. Hence we use the direct evaluation for all of our self-evaluations.

Figure 10 that investigates the Reflexion technique (Shinn et al., 2023) reveals a similar trend compared to the multi-agent debate with respect to inference scale. We find that Reflexion relies heavily on the oracle that helps the model determine when the correct answer is encountered and stops the generation early and returns that answer. This is in contrast to strategies like SC. We demonstrate the performance of Reflexion including baselines that have access to oracle and without. For direct comparison, it is more fair to compare strategies within the group with access to an oracle, or without. We find that in each group, inference scale is a strong prediction on the performance.

C Mathematical Framework for Self-Consistency

In many real-world reasoning tasks and decision-making processes, the use of SC has emerged as a powerful and often robust technique. Whether it's human experts forming a consensus or ensemble methods in machine learning, the idea of aggregating multiple opinions to reach a final decision has proven to be effective. The empirical success of SC in various domains, such as classification, regression, and human-driven decision-making, motivates a deeper examination into the underlying principles that make it work so well.

For instance, in complex reasoning tasks where individual models or experts might be uncertain, the wisdom of the crowd often leads to improved accuracy. SC can act as a regularization method, mitigating the effects of overfitting or biases that might be present in individual models. By combining multiple models or opinions, SC captures the common patterns among them, enhancing generalization to unseen data.

In this work, we seek to understand what makes SC an effective strategy, especially in the context of reasoning tasks. We aim to analyze the mathematical properties and probabilistic behavior that underlie this mechanism, considering various scenarios such as binary choices or multi-choice problems. Through rigorous analysis, simulations, and real-world datasets, we hope to derive insights that explain why SC often leads to consistent improvement and under what conditions it might fail.

The following section explores the mathematical explanation of SC, beginning with a simple binomial distribution model and gradually extending to more complex multinomial and Dirichlet distributions. By understanding the mathematical characteristics of these distributions, we hope to explain the empirical results observed in real-world reasoning tasks, thereby contributing to the ongoing efforts to harness the power of SC in a wide range of applications.

C.1 Self-Consistency Results on Reasoning Tasks

In our exploration of SC strategies applied to reasoning tasks, we conducted several experiments to analyze the effectiveness and behavior of different approaches. Figure 3 and Appendix G illustrate our findings, including the results for different tasks.

The convergence patterns and the improvement as the number of trials increases are shown for each task, highlighting the impact of SC.

These visualizations demonstrate the potential of SC in enhancing reasoning tasks, leading to more robust and accurate solutions. In this section, we will provide a theoretical framework that could explain the gains from SC. Note that we use Self-Consistency (SC) and Majority-Vote (MV) interchangeably.

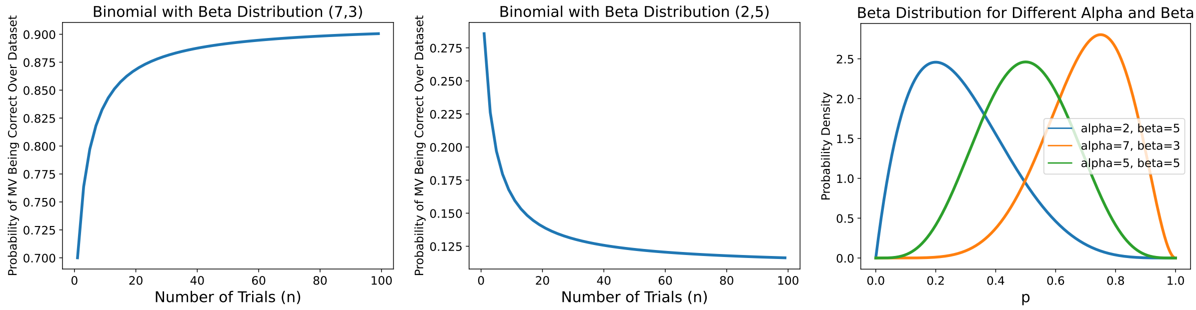
C.2 Binomial

We seek to analyze the behavior of parallel sampling with n trials with self-consistency or SC. In this setup, given a set of problems $\{x_i\}$, each problem's answer prediction (whether it is correct or not) can be modeled as a binomial distribution, assuming two choices (yes or no). Mathematically, the probability mass function for each problem's answer is given by:

$$f(X_i = k) = \binom{n}{k} p_i^k (1 - p_i)^{n-k}, \tag{1}$$

where X_i corresponds to the correct answer of the binomial distribution and p_i represents the probability of a correct answer for the i-th problem.

We can calculate the probability that SC yields the correct solution over n trials by calculating the probability that X_i yields a value that is at least



(a) (b) (c)

Figure 13: Convergence of self consistency under different beta distributions. Here, a Beta distribution that peaks at high p indicates that there are a lot of data examples where the model can solve with high probabilities, which leads to higher average self-consistency scores.

$n/2$. This is expressed as:

$$P(\text{MV correct}|x_i) = \sum_{k=\lceil n/2 \rceil}^{n} \binom{n}{k} p_i^k (1-p_i)^{n-k}. \tag{2}$$

By plotting the probability of MV being correct as a function of n, we observe that as n increases, $P(\text{MV correct}|x_i)$ either goes to 0 or 1, depending on whether $p_i > 0.5$ or $p_i < 0.5$ for this particular problem. This is evident in the synthetic experiment shown in Figure 14.

If p_i is extreme (closer to 1 or 0), then the convergence is fast, and the probability function can be described as:

$$\lim_{n\to\infty} P(\text{MV correct}|x_i) = \begin{cases} 1 & \text{if } p_i > 0.5, \\ 0 & \text{if } p_i < 0.5. \end{cases} \tag{3}$$

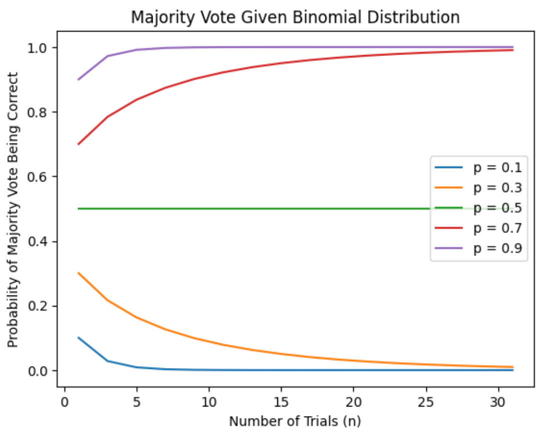


Figure 14: Probability of self consistency being correct for a given problem with varying p.

On the other hand, if p_i is close to 0.5, the convergence is slow, reflecting the uncertainty associated with an answer that is nearly equally likely to be correct or incorrect.

Over the set of all problems we consider, we place a beta distribution over p_i and integrate $P(\text{MV correct}|x_i)$ over the set of all problems to obtain $P(\text{MV correct})$. This can be expressed mathematically as:

$$P(\text{MV correct}) = \int_0^1 P(\text{MV correct}|p_i) \cdot f(p_i|\alpha,\beta)\, dp_i, \tag{4}$$

where $f(p_i|\alpha,\beta)$ is the probability density function of the beta distribution with parameters α and β.

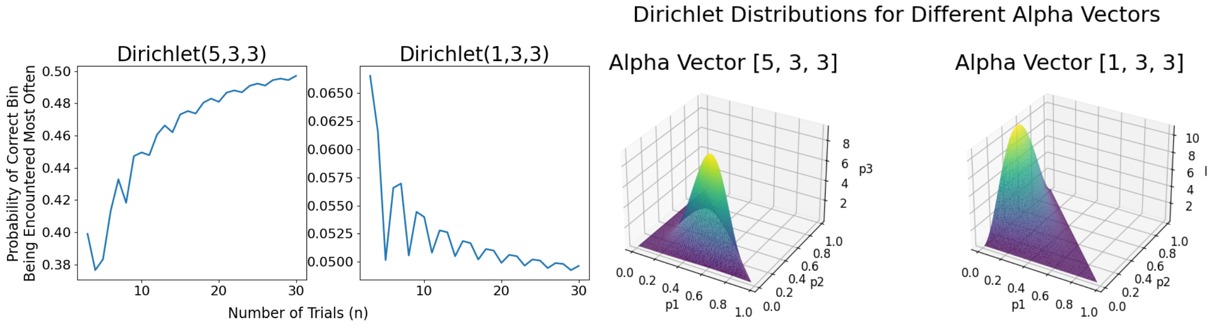
If we select a beta distribution where the mode peaks beyond 0.5, then we find that $P(\text{MV correct})$ increases as a function of n, albeit to a value less than 1 as you can see in Figure 13. This behavior explains our observation in real datasets directly.

This also implies that for datasets where majority vote leads to consistent improvement, the distribution of p_i needs to be peaked greater than 0.5. There would also exist a set of problems where self-consistency leads to lowered performance, specifically for the set of problems where $p_i < 0.5$.

By carefully selecting the parameters of the beta distribution, we can control the characteristics of the majority voting process and gain insights into the behavior of parallel sampling across various datasets. This mathematical framework provides a powerful tool for understanding and optimizing the majority vote process in practical applications.

C.3 Generalization to multinomial

We can further generalize this setup by considering each problem as being modeled by a multinomial distribution with K choices. In this more generalized scenario, the distribution of probabilities over problems can also be modeled by a Dirichlet distribution.



(a) (b)

Figure 15: Convergence of self consistency under different Dirichlet distributions with $K = 3$

Let $p = (p_1, p_2, \ldots, p_K)$ be the probabilities associated with the K choices, and let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_K)$ be the parameters of the corresponding Dirichlet distribution. The probability of obtaining a correct majority vote for a given problem is then:

$$P(\text{MV correct}|p) = \sum_{k=\lceil n/2 \rceil}^{n} \text{multinomial}(k; n, p), \tag{5}$$

where the sum is taken over all combinations of k votes that would result in a majority for the correct choice.

The overall probability of obtaining a correct majority vote, integrating over all problems, can be expressed as:

$$P(\text{MV correct}) = \int P(\text{MV correct}|p) \cdot f(p|\alpha)\, dp, \tag{6}$$

where $f(p|\alpha)$ is the probability density function, which can be modeled by the Dirichlet distribution.

Following a similar simulation to the binary case, we find that the conclusions hold (see Figure 15). Specifically, if the mode of the Dirichlet distribution is biased towards the correct choices, the probability of the majority vote being correct increases with n, and the set of problems where self-consistency leads to lowered performance can be characterized by the subset where the correct choice probabilities are below certain thresholds.

This generalization to multinomial and Dirichlet distributions adds complexity but also additional flexibility in modeling the majority voting process, making it applicable to a broader range of practical scenarios.

Method	Correct Accuracy	Incorrect Accuracy	Total Accuracy
Yes or NO	0.911	0.461	0.707
Score 1-10	0.995	0.149	0.613
Probability 0.0-1.0	0.886	0.115	0.537

Table 2: Self-evaluation accuracy on MATH with three methods

D Self-Evaluation

D.1 Self-Evaluation Method

Given an answer, there are multiple ways we can prompt the LLM to evaluate that answer. Here we examine 3 possibilities for self-evaluation

1. **Binary**[5] - we ask the model to output Yes/No as to whether the answer is correct. We do this multiple times and take the fraction of times the model answers Yes as the confidence of the model in the answer.

2. **Numerical confidence** - we ask the model to output a score between 1 and 10 to indicate its confidence in the answer. We do this multiple times and take the average as the confidence of the model in the answer.

3. **Confidence probability** - similar to the previous strategy except now we prompt the model to output a confidence between 0.0 and 1.0 and average it.

The evaluation result is shown in Figure 11 and Table 2. The binary Yes or No is the most well calibrated.

D.2 Self-evaluation is correlated with problem difficulty

To get an understanding of whether models found it easier to evaluate answers to easier problems, we computed the following metric for a 100 problem subset of the GSM8K dataset. For each problem i, let a_{ij} be the jth answer. We had 20 sampled answers per problem. We computed the fraction c_i of answers that were correct. Our assumption was that c_i indicates the difficulty of the problem – the higher the value, the easier the problem. For each answer a_{ij}, we obtained the binary self-evaluation confidence as described in the beginning of this section (we sampled the evaluation 5 times). We then computed the correlation ρ_i between the self-evaluation confidence for the answers a_{ij} and the binary vector indicating whether the answers were correct or not. We then computed the correlation between ρ_i and c_i. We obtained a correlation of 0.347 with a p-value of 0.00026 – a clear indication that an increase in the problem difficulty results in the self-evaluation becoming more noisy. We repeated this experiment for MATH and TheoremQA and obtained correlations of 0.31 and 0.42 with p-values of 0.02 and 0.0025 respectively.

[5] We also investigate a variant where we ask the model to think step by step before evaluating. While we see a small increase in performance for such a strategy, it also necessitates a big increase in the token budget. Further analysis is in the Supplement.

D.3 Self-Confident Self-Consistency (SC^2) Details

We take the answer which has the highest SC score as the predicted answer. Formally the definition is

$$SC^2_a = \sum_{a_i=a} \text{confidence}(a_i) \tag{7}$$

where $\text{confidence}(a_i) = \frac{\sum_{v_j} \mathbb{I}(v_j=\text{Yes})}{m}$ where m denotes the number of Binary evaluations v_j sampled. We apply this strategy to the MATH, TheoremQA (integer answer subset), TheoremQA (random subset), and HotpotQA datasets. SC^2 is consistently on par or better than a simple majority vote. The results are in Figure 16. SC^2 achieves non-trivial gain for math reasoning tasks but the overall costs increase quite a bit. This prompts us to inquire whether the achieved performance boost justifies the additional costs incurred. However, if we have the option to cache, then during self-evaluation, previous questions and answers can be cached and don't need to be encoded again. This can save a lot of budget and the new results would look like Figure 12. We see non-trivial gains for the math reasoning datasets. However, for TheoremQA we see markedly smaller gains. We hypothesize that the reason for this is that TheoremQA is a harder dataset for the model. As we showed in the previous section, self-evaluation ability decreases as problem difficulty increases. GPT-4 shows a self-evaluation ability of no better than random for TheoremQA and thus we observe very small improvement. Overall, a budget-aware comparison of reasoning methods is a healthy approach to compare among vastly different methods.

D.3.1 Budget-efficiency

The strategy requires only a handful of extra tokens (m additional tokens per answer corresponding to the Yes/No) to execute (Figure 17). However, it does require more encoded tokens (We can sample

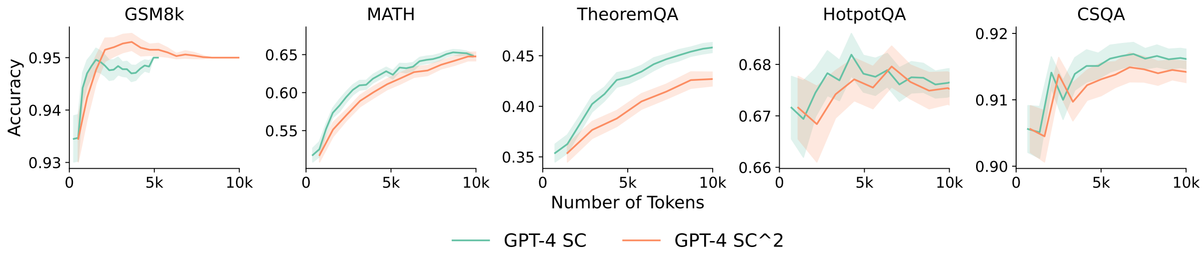


Figure 16: SC^2 with total tokens being the budget. There are sizable improvements in using our method SC^2 on math reasoning tasks.

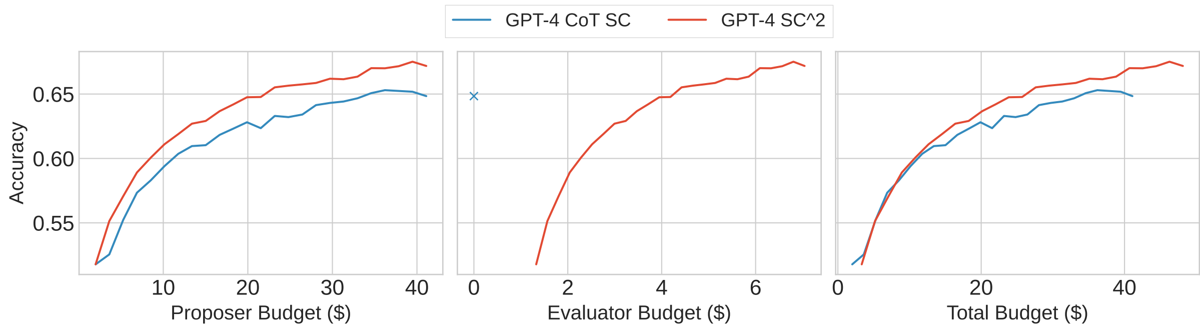


Figure 17: Separate proposer budget and evaluation budget on the dataset of MATH.

Method	Top1	Best out of all	Total Accuracy
ToT b=5 (GPT-4,GPT-4)	0.74	0.76	0.4
ToT b=3 (GPT-4,GPT-4)	0.77	0.77	0.49
ToT b=1 (GPT-4,GPT-4)	0.65	0.65	0.65
ToT eval once (GPT-4,GPT-4)	0.73	0.75	0.352
CoT 100 times (GPT-4)	0.17	0.56	0.0756
ToT Random Eval (GPT-4)	0.0	0.04	0.008
ToT b=5 (GPT-3.5,GPT-3.5)	0.25	0.35	0.11
CoT 100 times (GPT-3.5)	0.04	0.46	0.0252
ToT b=5 (GPT-4,GPT-3.5)	0.68	0.72	0.302
ToT b=5 (GPT-3.5,GPT-4)	0.3	0.38	0.156

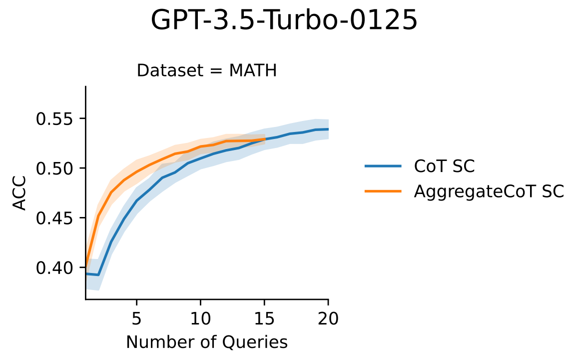
Table 3: Various results on Game of 24. ToT refers to Tree-of-Thoughts. For ToT, the first model name in the parenthesis refers to the model used to *generate* the candidate thoughts, while the second model name refers to the model used to *evaluate* the candidate thoughts.

all of the m additional tokens as part of a single query). Thus if one is self-hosting the model, this strategy has only marginal additional cost.

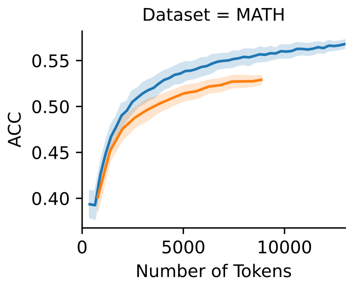
D.3.2 Query vs Token budget

While we have discussed both query and token budget in this paper, token budget has some notable advantages as a metric.

Theoretical aspects Equivalence in the number of queries can be arbitrarily far from the equivalence in the amount of compute. Merrill and Sabharwal (2023) and Pérez et al. (2021) both show that the expressive power of transformers can be greatly enhanced by generating intermediate steps in the computation (colloquially called chain of thought). Merrill and Sabharwal (2023) shows that without any bound in the number of steps, an encoder-decoder architecture with only one encoder and three decoder layers can simulate a Turing Machine and thus a single query to such a Transformer can perform computations with arbitrarily large amount of compute. Pérez et al. (2021) shows that even for decoder-only transformers, allowing for polynomial-sized chains of thought makes it powerful enough to do, in a single query, any computation a Turing Machine can do in polynomial time. While the number of queries metric fails to capture this, by contrast, the number of tokens metric which is novel to our paper, does capture this aspect as it by definition includes the length of the generated thought as part of the compute.



(a) Query Budget



(b) Token Budget

Figure 18: We evaluated this custom reasoning strategy on MATH with GPT-3.5-Turbo-0125 for 15 queries, so in theory it should generate 15*4=60 responses. Here is the result based on the number of queries metric (we name the custom reasoning strategy AggregateCoT). We can find that it never outperforms Chain-of-Thought Self-Consistency with same amount of tokens. The "improvement" previously was an unfair comparision because the custome reasoning strategy will use much more tokens per query.

Practical aspects The above is not just a theoretical consideration. In Figure 18a we demonstrate, a custom reasoning strategy that at first glance, seems to outperform self-consistency – based on the number of queries metric. However, when we properly take the holistic compute budget into account via the number of tokens metric, we can see that self-consistency is more token-efficient (Figure 18b). That is, the number of tokens as a metric of budget captures the nuances of resources required for LLM reasoning more properly.

E More Ablation Results for Tree of Thought

In Table 3 you can see more ToT results on the task of Game of 24. Most of the results are shown in the Figure 7. The table mainly shows the ablation for when using GPT-3.5 as the proposer and GPT-4 as the evaluator. We see that the performance is better than using a GPT-3.5 as the evaluator but far below the performance of using GPT-4 as the proposer.

F Terms and Licenses

GSM8K, MATH, TheoremQA, CSQA are under the MIT license. HotpotQA is under the CC BY-SA 4.0 License. All the datasets and models are used for their intended use.

G Results From More Models

G.1 MAD & Reflexion

Here we extend the results to a variety of models: GPT-3.5-Tubo-0125, Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a), LLaMA-2-70b-chat(Touvron et al., 2023), and Mixtral-8x7B-Instruct-v0.1. Overall, we find similar trends that self-consistency is extremely competitive compared to multi-agent debate and reflexion, when evaluated in a budget-aware manner.

We observed that it is very consistent that CoT with self-consistency beat other reasoning strategies across models with various sizes/training procedures. Multi-agent debate and Reflexion often decrease performances with more budget. This is not surprising considering our analysis in Section 5. Note that for LLaMA-2-70b-chat, we can't run Mad and Reflexion to the same amount of budget as CoT with self-consistency due to the context limit of around 4k. But the trend stays similar.

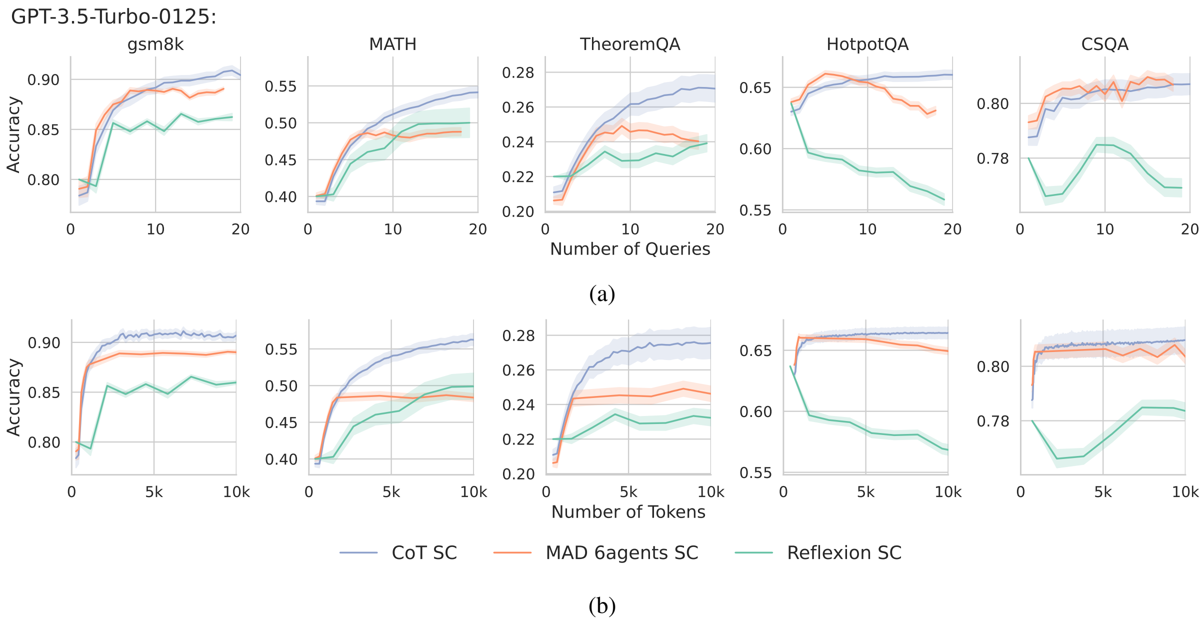
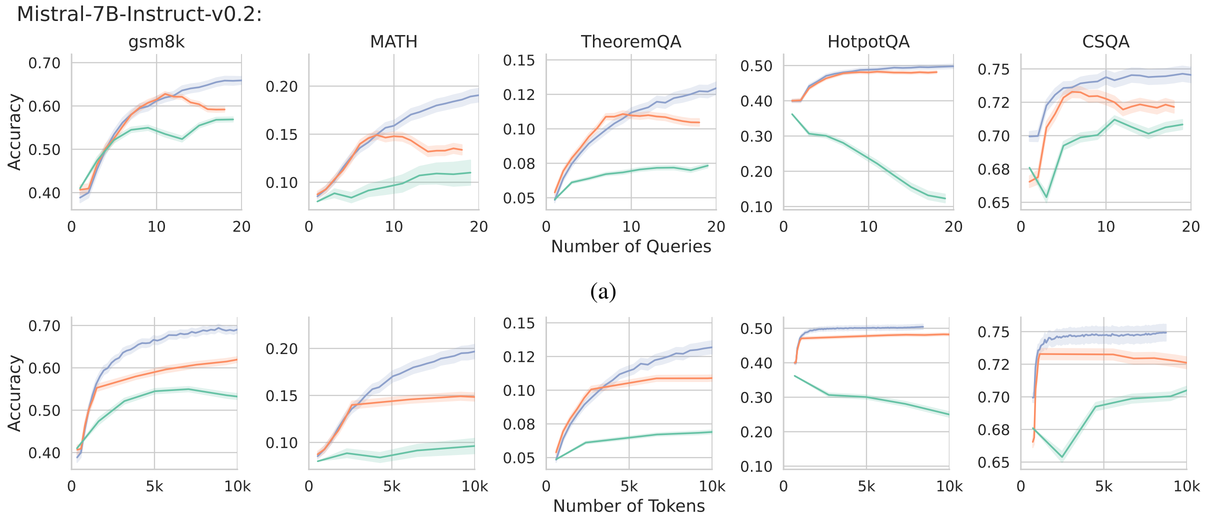
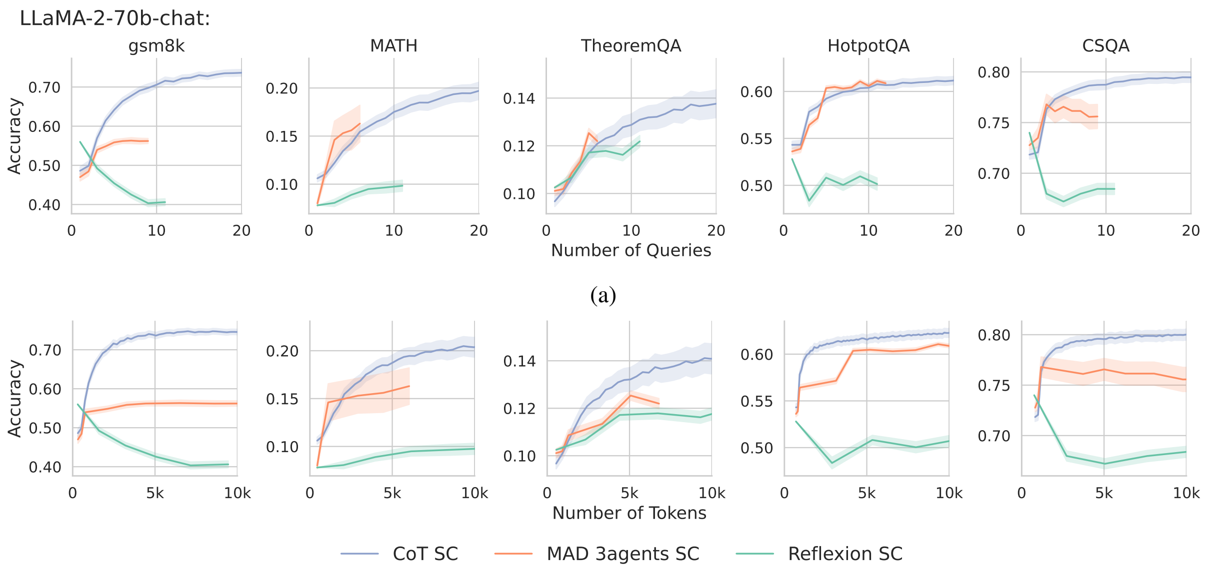


Figure 19: GPT-3.5-0125: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.



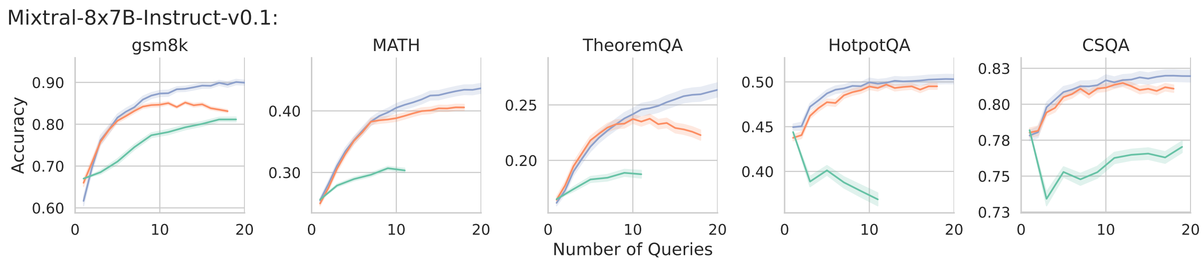
(b)

Figure 20: Mistral-7B-Instruct-v0.2: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.

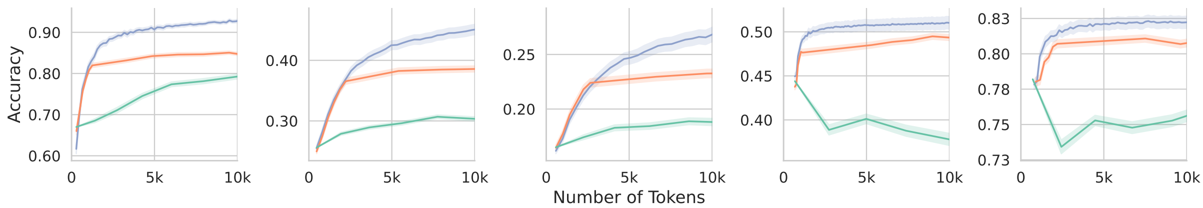


(b)

Figure 21: LLaMA-2-70b-chat: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.



(a)



(b)

Figure 22: Mixtral-8x7B-Instruct-v0.1: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.

G.2 Three Other Reasoning Strategies

In this section, we will evaluate on three other reasoning strategies in the self-consistency family: Plan and Solve (Wang et al., 2023), Least to Most Prompting (Zhou et al., 2022), and Progressive Hint Prompting (Zheng et al., 2023).

Plan and Solve It asks LLMs to do some planning before solving a question. It is like an extension to CoT. To plot the accuracy vs. budget figure, we increase the number of sampled CoTs to increase the budget.

Least to Most Prompting This strategy prompts the model to decompose a question first and then answer each subquestion before aggregating them to the final answer. To plot the accuracy vs. budget figure, we also increase the number of sampled CoTs to increase the budget.

Progressive Hint Prompting This strategy uses previous answers as hints to generate next answer. To plot the accuracy vs. budget figure, we increase the number of rounds of regeneration of answer given hints.

All three new strategies here can be integrated with self-consistency seamlessly, since they are mostly just variants of chain-of-thought. Based on the plots, it seems that normal self-consistency is still very competitive, but different prompting styles can make a big difference. For some models and some datasets, a strategy other than CoT converges to a higher performance. This is strong evidence that self-consistency is a really budget-effective strategy.

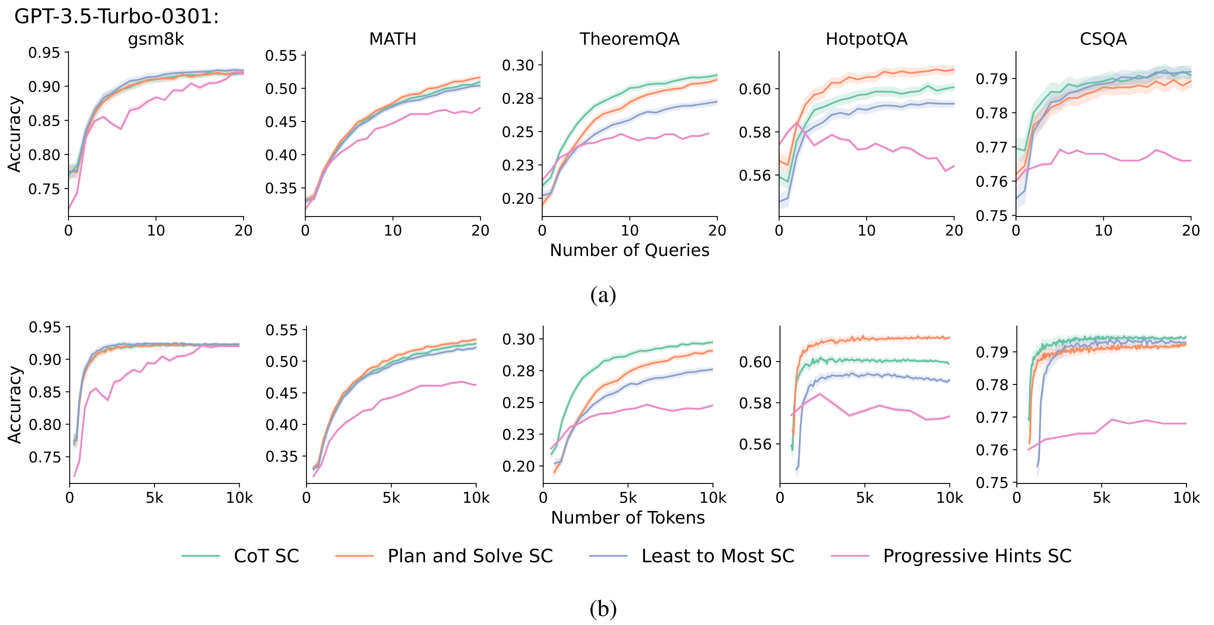


Figure 23: GPT-3.5-0301: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.

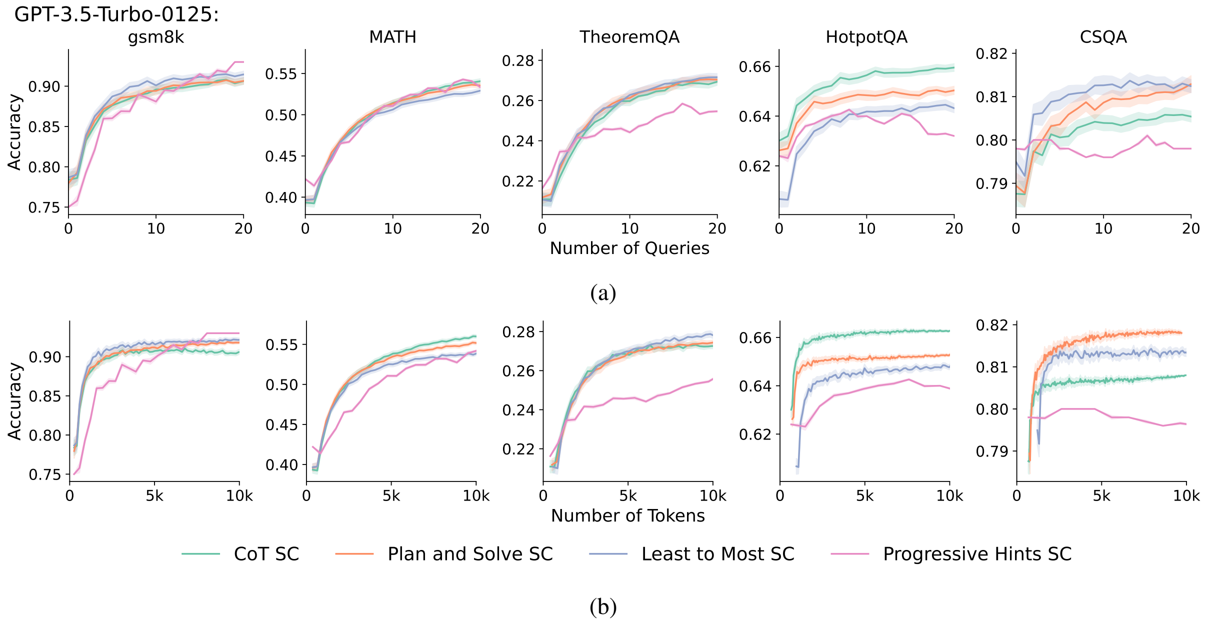


Figure 24: GPT-3.5-0125: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.

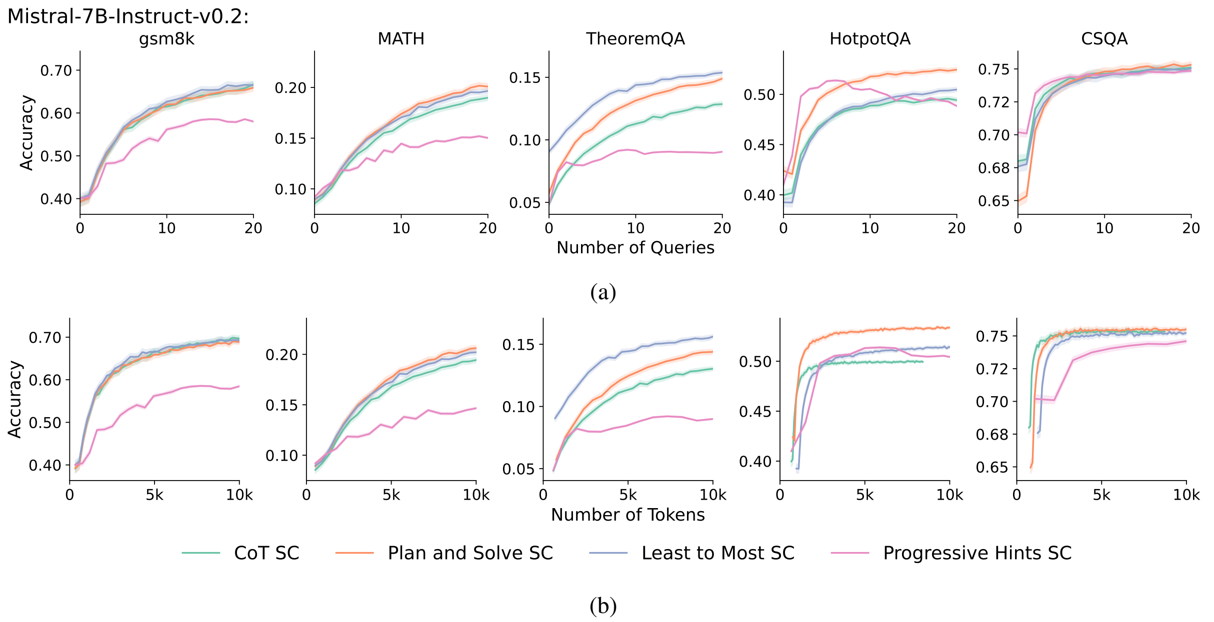
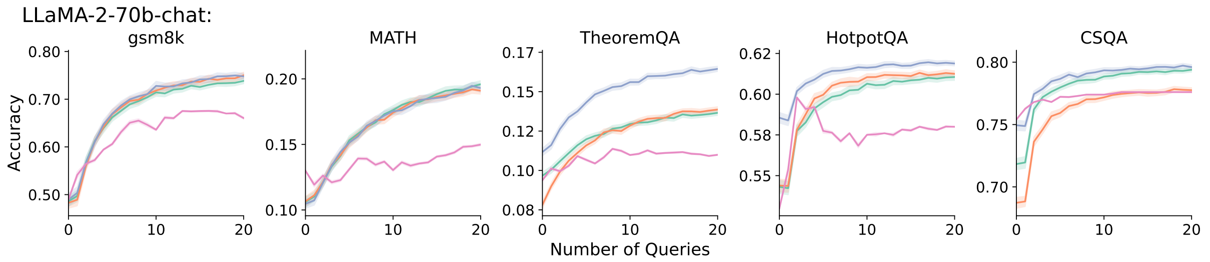
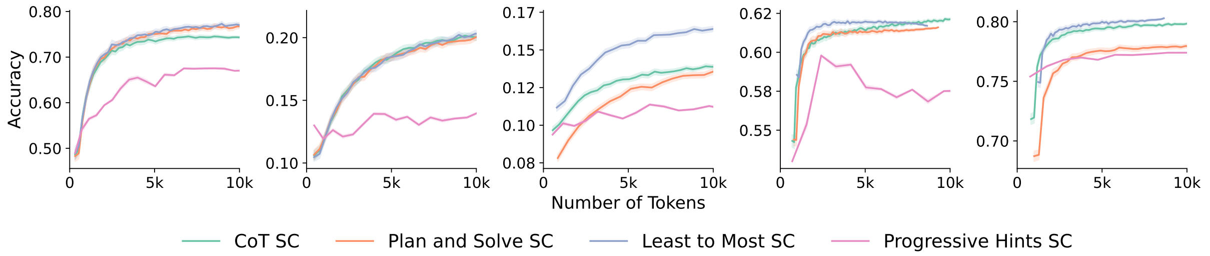


Figure 25: Mistral-7B-Instruct-v0.2: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.

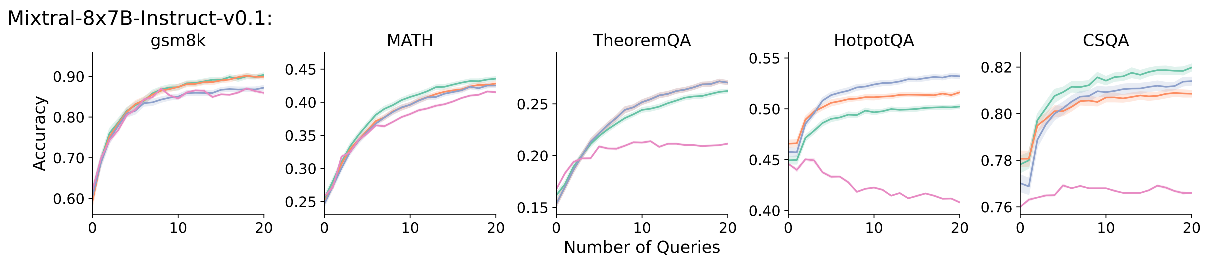


(a)

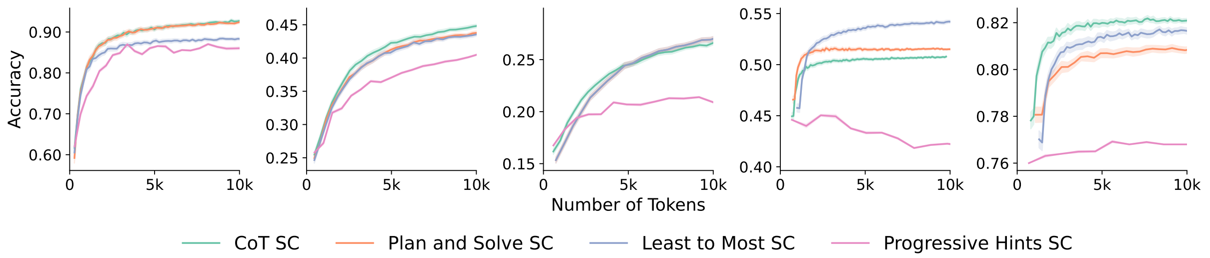


(b)

Figure 26: LLaMA-2-70b-chat: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.



(a)



(b)

Figure 27: Mixtral-8x7B-Instruct-v0.1: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets.