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

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have ushered in a diverse array of reasoning strategies, each with unique computational requirements. Traditional evaluations that focus solely on performance metrics miss a key factor: the increased effectiveness due to scale. 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 process, providing a more informative comparison that takes into account both performance metrics and computational cost. Our scale-aware investigation reveals a strong correlation between performance and compute budget, showing that simple strategies like Self-Consistency (SC) can outperform more complex methods when scale is considered. We further explore the impact of two specific types of budgets: answer generation and evaluation, highlighting the significant role of self-evaluation in performance enhancement for certain reasoning strategies. We also propose Self-Confidence-weighted Self-Consistency (SC^2) as a new baseline and identify a correlation between model calibration and success in self-evaluation-based strategies. These findings open doors for more efficient budget utilization and may spur the development of more robust and cost-effective reasoning strategies and LLM applications.

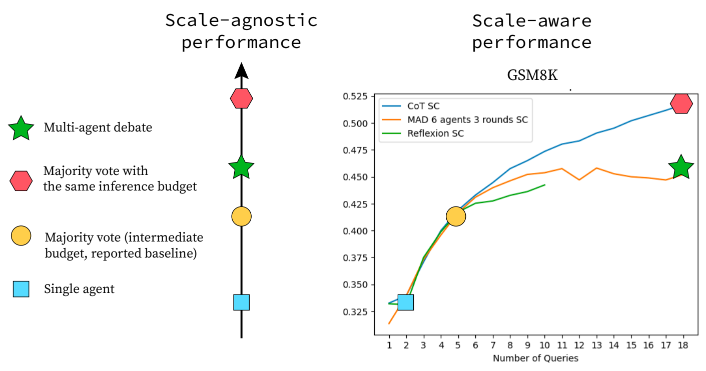
1 Introduction

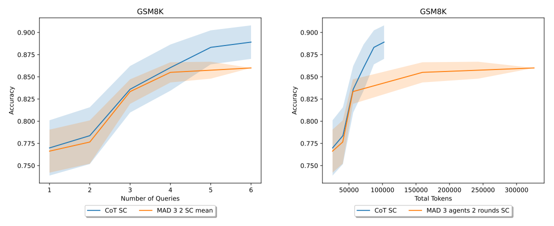
The arena of large language models (LLMs) such as GPT-4 (OpenAI, 2023) has seen a proliferation of diverse reasoning strategies. However, comparing these strategies fairly and comprehensively has proven to be a challenging task due to their varied computational requirements. For instance, strategies like the 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.

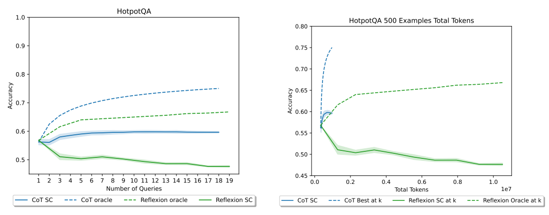
Our empirical research uncovers a significant correlation between the performance and the compute budget. We find that a straightforward baseline strategy, the chain of thought reasoning coupled with Self-Consistency, can be remarkably competitive. When scaled to match the compute resources of more sophisticated methods such as multi-agent debate and self-reflection, this baseline strategy often outperforms them in achieving the best trade-off between performance and budget.

We further scrutinize the influence of two specific types of budgets on performance: (1) the answer generation budget, and (2) the evaluation budget. Our findings indicate that for a robust model like GPT-4, investing more resources into self-evaluation—i.e., increasing the evaluation budget—contributes to a substantial improvement in performance. This emphasizes the integral role of self-evaluation in certain reasoning strategies such as the tree of thought. This also highlights the importance of training models to be better at self-evaluation as part of their pretraining. On the other hand, LLMs with less capable self-evaluation features, like GPT-3.5, do not benefit comparably from these strategies. Based on this, we also propose Self-Confidence-weighted self-evaluation SC^2 as another simple yet strong baseline for math reasoning tasks. Moreover, we identify a strong correlation between the calibration via a correctness prediction proxy and the success of reasoning strategies that leverage self-evaluation. By approx-





(a) MAD with GSM8k



(b) Reflexion with HotpotQA

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 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 on HotpotQA also illustrates the artifact of scale on performance. In both (a) and (b), we show both budgets, the number of total tokens, and the number of queries. All results were obtained from GPT-3.5.

imating the model's confidence in its answers, we observe that well-calibrated models are more likely to excel in self-evaluation-based reasoning strategies.

Through this research, we offer a more nuanced understanding of reasoning strategies in LLMs by considering both their performance and computational costs. Our results are agnostic in regard to the architecture or training of the models. 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.

Concretely, our contributions are

- We present a comprehensive head-to-head evaluation of multiple LLM reasoning strategies on multiple types of datasets using GPT-3.5, and GPT-4.

- We evaluate the performance of the strategies on a novel dimension – performance w.r.t. budget. Specifically, we propose 3 types of metrics: performance@number of queries, performance@number of tokens, and performance@monetary cost and find that SC is the strongest compared to all other strategies for most models and datasets except for ToT with GPT-4.

- We provide a theoretical analysis of why SC is strong. We also delve into the effectiveness of the evaluator of ToT in a budget-aware manner and found that self-evaluation is required for optimal performance.

- We then incorporate this self-evaluation into SC itself and present a new variant - Self-Confident Self-Consistency (SC^2) - which shows improvement over SC for math reasoning tasks while being on par for other tasks.

2 Related Work

2.1 Reasoning strategies for LLMs

There has been a flurry of activity to use language models for generating effective reasoning and planning strategies. An early work in the area was to prompt the language model to generate its chain-of-thought (CoT) (Wei et al., 2022) when solving a problem which lead to significant improvements

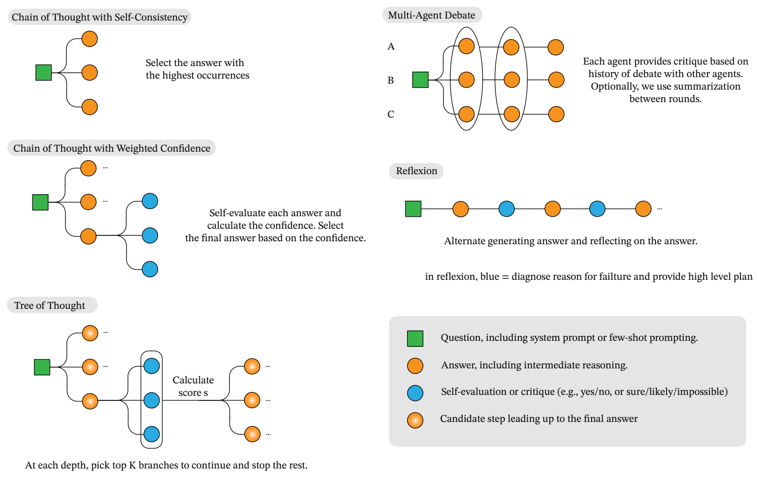

Chain of Thought with Self-Consistency
Select the answer with the highest occurrences
Chain of Thought with Weighted Confidence
Self-evaluate each answer and calculate the confidence. Select the final answer based on the confidence.
Tree of Thought
Calculate score s
At each depth, pick top K branches to continue and stop the rest.
Multi-Agent Debate
A
B
C
Each agent provides critique based on history of debate with other agents. Optionally, we use summarization between rounds.
Reflexion
Alternate generating answer and reflecting on the answer.
in reflexion, blue = diagnose reason for failture and provide high level plan
Question, including system prompt or few-shot prompting.
Answer, including intermediate reasoning.
Self-evaluation or critique (e.g., yes/no, or sure/likely/impossible)
Candidate step leading up to the final answer

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.

in the model's problem solving abilities. Later work has involved prompting the language model to come up with its plan for solving the problem before trying to solve it (Jiang et al., 2023), using CoT to solve a problem and then asking 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; Shinn et al., 2023), generating multiple CoT and combining them using the LLM (Yoran et al., 2023), setting up a tree search for CoT instead of sampling a single linear CoT (*Tree of Thoughts* - ToT) (Yao et al., 2023), aggregating LLM generated feedback for multiple prompts and their solutions into guidelines that can improve future generation (Chen et al., 2023b), and using multiple LLMs as debating agents to refine feedback for a solution (Du et al., 2023; Liang et al., 2023). However, they are all evaluated on different datasets and whether the baselines are computed or cost-matched is rarely considered. Notable exceptions are Shinn et al. (2023) where they consider performance as a function of the number of queries to the language model and Olausson et al. (2023) which evaluates the performance of a self-debug strategy for code generation as a function of the number of tokens generated.

2.2 LLM output evaluation

There has been considerable work on evaluating the output of LLMs – both via training custom models as well as using the LLMs themselves for self-evaluation. For trained verifiers/rerankers, in Cobbe et al. (2021), they train a verifier to rerank outputs of language models for math word problems and show strong improvements. In Inala et al. (2022), they do the same except for code generation. In Uesato et al. (2022); Yang et al. (2022), they train an evaluator for each step in a chain of thought and rerank using the combined score for each step in the chain. In Li et al. (2023), they weight the Self-Consistency by the trained verifier confidence. There has also been work recently on using the LLMs themselves to evaluate their 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 correct a deductive step is. While they can obtain high accuracy as to whether a step is valid or not, they are unable to improve the overall accuracy of answer generation using that. Tian et al. (2023) examine multiple strategies for eliciting LLM self-evaluation that is as calibrated as possible. The Self-Refine (Madaan et al., 2023) approach uses LLMs to get detailed self-evaluation which is used to improve the next round of generation. The Tree-of-Thoughts (Yao et al., 2023) paper uses LLM self-evaluation to rank which node to explore next.

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-constrained* is less well-studied (with the notable exception of Olausson et al. (2023)). 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. We furthermore describe how different reasoning strategies can scale in terms of each budget.

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.

i) **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, whereas n_O is tallied based on the sample count.

ii) **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.

iii) **Number of queries** of planned API calls can a rough proxy for budget. Such number can be determined before inference, which can give us a rough guidance before actually performing each reasoning strategies. Note that in case we want to sample multiple outputs from the LLM, we count those as *separate* queries

3.2 Reasoning Strategies

In this section, we give a brief overview of different reasoning strategies in the literature, illustrated in Figure 2 and how each can scale with budget. A detailed description is given in the Supplement.

a) **Self-Consistency** uses parallel sampling to generate multiple answers and select the answer that is generated most often. We can increase the budget by increasing the number of samples.

b) **Multi-agent debate** uses multiple agents to debate the previous answer and debate in rounds. We can increase the budget by either increasing the number of agents or rounds.

c) **Reflexion** alternates between answer generation and self-critique. We can increase the budget by using higher number of rounds of generation and critique.

d) **Tree of thoughts** alternates between thought generation (in a chain of thought) and self-critique, including branching out to multiple thoughts or multiple critiques via parallel decoding at each step. We can increase budget by increasing the breath and/or the depth of the tree. An answer is the collection of thoughts from the root to a leaf of the tree.

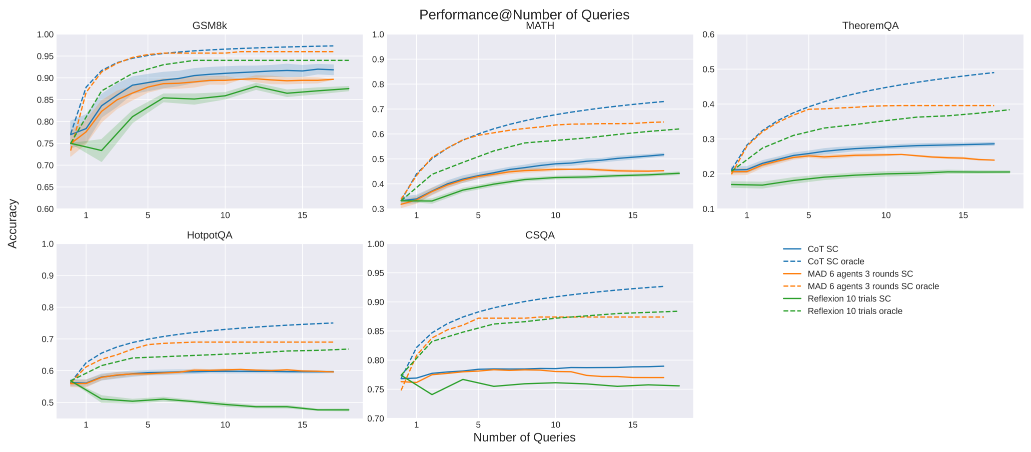
4 A Critical Evaluation in Budget-Aware Environments

This section aims to explore key components that can make reasoning strategies successful from the scale-aware perspective. First, we show that the inference scale is often overlooked but is one of the primary indicators of the success of a reasoning strategy. We claim that using a simple Self-Consistency baseline will, in many cases, outperform more complex reasoning strategies proposed in the literature. 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), 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 (Chen et al., 2023c), along with the commonsense reasoning task CSQA (Talmor et al., 2019), and the 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) to further our investigation on budget-aware evaluation. We use GPT-3.5 and GPT-4 for our experiments.

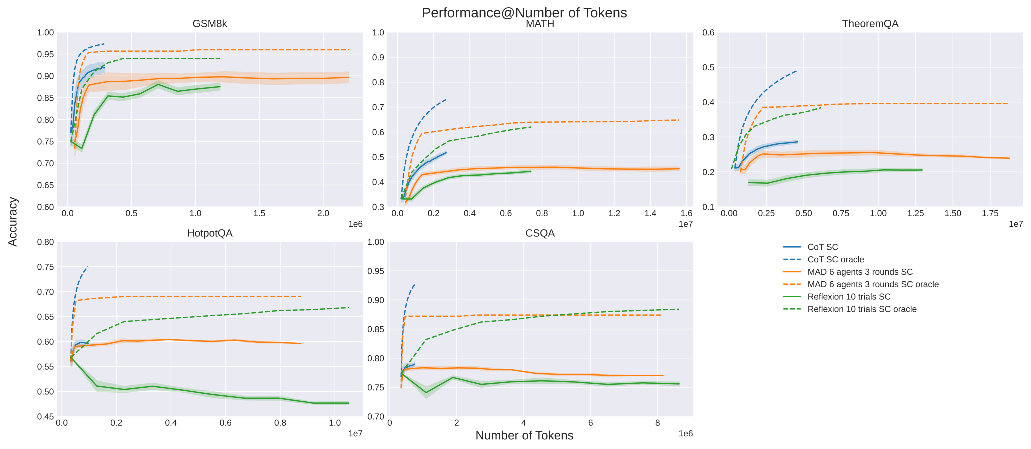
4.1 Importance of Inference Scale

In this section, we explore how 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 1 and 3 elucidate the efficacy of reasoning techniques, including multi-agent debate and Reflexion, in contrast with the SC baseline. A more comprehensive result is shown in Figure 3 with more queries and datasets. We keep the budget



(a)



(b)

Figure 3: (a) Performance@Number of Queries Plots for all 5 datasets. (b) Performance@Number of Tokens for all 5 datasets. All three methods CoT SC, MAD, and Reflexion are plotted. All experiments here are done with 18 queries for all three reasoning strategies to be comparable. The solid lines are regular performances and the dashed lines are oracle performances (If any candidate solution is correct, then we count it as correct). The MAD result is shown non-round-wise. All results are obtained from GPT-3.5.

for each question to 18 queries. This means for CoT SC, we would sample 18 times. For MAD, we set the number of agents to 6 and the number of rounds to 3 which result in exactly 18 queries. For Reflexion, we set it to reflect a maximum of 10 trials (s10 proposals 9 reflections for a total of 19 queries). We demonstrate two budgetary metrics which were discussed in (Section 3.1): a) Performance@Number of Queries b) Performance@Number of Tokens. As illustrated in Figure 1 and 3, when the inference budget of the baseline is aligned with that of each reasoning approach, the perceived benefits of the innovative strategies no longer apply. The SC baseline regularly outperforms more complex strategies when given equivalent budgets. Relying solely on scale-independent assessments, as is sometimes done in prior works, might lead to incomplete or potentially misleading interpretations.

In subsequent sections, we analyze various components that can affect LLM reasoning abilities.

4.2 Does a higher inference budget always lead to better reasoning?

The scale-conscious perspective of reasoning strategy performance offers clear a guideline to what reasoning strategies make sense. That is, higher inference budget of a proposed reasoning strategy should lead to increased performance *compared to a baseline*; otherwise, the incurred cost could not be justified since the baseline with comparable cost performs better (in terms of FLOPs, latency, monetary cost, or any cost we care about). A question arises whether we can keep increasing the budget to obtain the best possible abilities.

As seen in Figure 3, we find that the SC baseline exhibits a smooth increase in scores with respect to scale. However, such a trend does not always hold for other reasoning strategies. For instance, in multi-agent debate (MAD), an augmented inference budget eventually experiences a performance plateau. For the MAD setting with 6 agents, the graph for MAD and SD overlaps up to 6 queries since they correspond to the exact same strategy up to that point. After 6 queries, the MAD strategy switches to the second round where the performance gain noticeably lessens compared to SC. The lowered performance compared to the SC baseline may arise because subsequent rounds of MAD may incite a cascading effect of cumulative mistakes, or snowballed hallucinations suggested in Zhang et al. (2023).

4.2.1 Complex reasoning strategies can hurt response diversity

To show this, we compared – the entropy of the solutions generated at each round for MAD vs. SC. The results are in Figure 15 (Appendix). The entropy consistently declines for MAD as we move to the next round suggesting exactly the kind of cascading effect we hypothesized. By contrast, the SC does not suffer such negative consequences and even increases its solution diversity since the responses are generated independently without affecting one another.

4.2.2 Effectiveness of independent sampling with CoT prompt

Next, we outline a framework that helps explain what makes self-consistency successful. We model the answer generation process by LLM 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 SC 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 scores over the entire dataset converge to non-extreme values, resembling the behavior observed in our results.

Our findings indicate that the model's performance improves with the number of trials if the distribution of p_i has a mean greater than 0.5. If the mean is lower than 0.5, then performance can degrade over time. [1] This finding suggests that the reason SC 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. In Appendix C, we detail the analysis with extension to multinomial setting with Dirichlet prior.

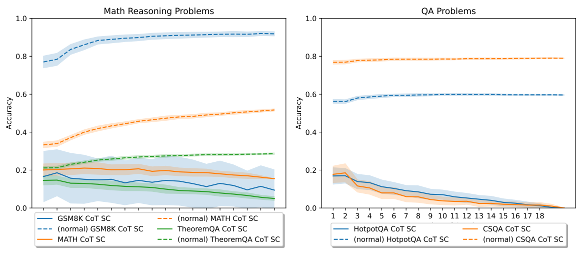


Figure 4: SC making things worse.

4.3 Tree-of-thoughts Strategy

We also evaluated Tree-of-Thoughts strategy in the scale-aware manner on the logical game Game of 24. Our findings mirrored the above conclusions when GPT-3.5 is used, where the integration of CoT coupled with Self-Consistency exhibited improved performance with reduced computational resources. However, notable discrepancies emerged in the behavior of the model when transitioning to GPT-4.

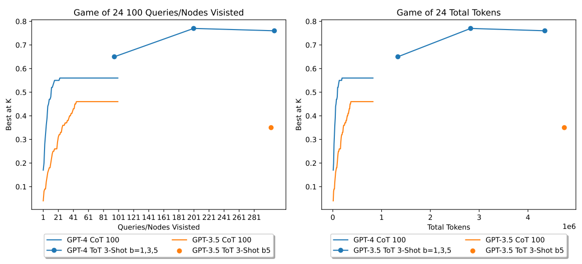
4.3.1 Tree-of-thoughts needs a strong model to be competitive

In Figure 5, we show the performance of GPT-3.5 with the Tree-of-Thoughts reasoning strategy on Game of 24[2]. 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 all other strategies plateau very early and Tree-of-Thoughts beats all of them by a big margin. This remains the case even when we account for the budget (query or token budget) as the other strategies have a low performance ceiling. However, note that 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 simpler strategies

[1]We confirmed this possibility by finding a subset of problems where the performance decreases shown in Figure 4

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

like SC which outperforms ToT by a considerable margin (Figure 5).



(a) Queries/Nodes Visited (b) Total Tokens

Figure 5: ToT vs. CoT SC for both GPT-3.5 and GPT-4. The dotted lines represent the performance of ToT with different settings. For ToT using GPT-4 (blue), results for three settings are included.

4.3.2 Tree-of-Thought Ablation Study

In this section, we delve deeper into the factors that contribute to the enhanced performance of the ToT strategy when compared to the robust baseline of CoT with SC while being budget-aware. ToT strategy 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 Tree-of-Thoughts strategy where we use GPT-4 to evaluate the new thoughts.
2. The standard Tree-of-Thoughts strategy except we now do an evaluation of a thought only once as opposed to 3 times.
3. Using a weaker model (GPT-3.5) as the evaluator while using GPT-4 as the thought generator.
4. Random evaluator, where we randomly select the subset of thoughts to prune.

As observed in Figure 6, a random evaluator leads to a very steep performance drop for ToT for both best@k as well as total accuracy. We found similar results in a similar experiment conducted on Reflexion shown in Figure 14. Both results imply that an evaluator does matter. Evaluation done only once per thought as opposed to multiple times also leads to significant performance drops. However, if we use a weaker evaluator like GPT-3.5, we can maintain most of the performance while being very cost-efficient. For example, running Tree-of-Thoughts (with beam width set to 5) with GPT-4 as thought proposer and GPT-3.5 as evaluator on 100 Game of 24 instances costs \$33.53 while getting an accuracy of 72%. Using GPT-4 as the evaluator on the other hand increases the cost almost 5x to \$159.87 while only improving accuracy to 76%. Even if we restrict ourselves to $b = 1$ with GPT-4 as an evaluator, we still get a higher cost of \$49.9 while getting a worse performance of 65%.

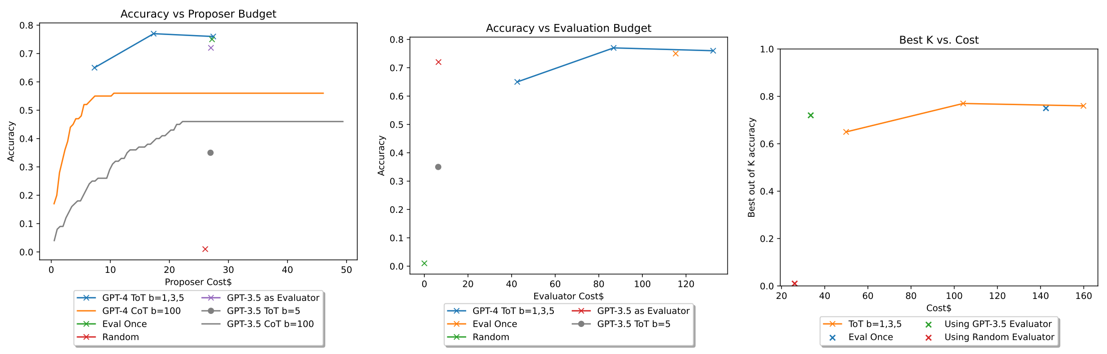
We also found that the proposer has a greater impact than the evaluator on the performance, you can find more ablation results in the Appendix Table2. We refrain from delving extensively into this aspect here, as virtually all reasoning strategies inherently necessitate some form of a proposer. Our primary focus lies in assessing the extent of the advantages conferred by the unique self-evaluator component.

4.4 Self-Confident Self-Consistency (SC^2)

Although the self-evaluation incurs additional costs to the reasoning strategy, plenty of evidence including some past work demonstrates the usefulness of self-evaluation (Olausson et al., 2023; Li et al., 2023). We thus propose to leverage that ability by weighting the SC by the confidence the model has in its answer, derived via the Binary evaluation strategy. We call this score the *Self-Confident Self-Consistency* (SC^2) score. We take the answer which has the highest SC score as the predicted answer. Formally the definition is

$$SC^2_a = \sum_{a_i = a} \text{confidence}(a_i) \tag{1}$$

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 7. 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



(a) Proposer Budget (b) Evaluator Budget (c) All Budget

Figure 6: Separate proposer budget and evaluation budget on the dataset of game24. ToT with different generator and evaluator permutations. For cost computation, we assumed the prices OpenAI had as of Aug 14, 2023 which were \$0.002 per 1K tokens for GPT-3.5 (encoding or decoding), \$0.03 per 1K encoded tokens and \$0.06 per 1K decoded tokens for GPT-4.

can save a lot of budget and the new results would look like Figure 8. 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 gains.

4.4.1 Budget-efficiency

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

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 5 datasets. Correct accuracy denotes self-evaluation accuracy for answers that turn out to be correct. Incorrect accuracy is the self-evaluation accuracy of incorrect answers. All numbers are done with GPT-4-0613

4.4.2 Self-evaluation ability is dataset and model dependent

Table 1 shows the self-evaluation accuracy for both GPT-3.5 and GPT-4 for multiple datasets. The self-evaluation accuracy turns out to be heavily dependent on the dataset. We dove deeper and examined the individual self-evaluations that GPT-3.5 made for HotpotQA and that GPT-4 made for TheoremQA. It is possible that for problems that are too hard for the model, it ends up weighing the writing *style* of the answer much more heavily than the correctness of all the intermediate steps when doing the evaluation. We examine this in more detail in the appendix D, which will also include how we specifically do self-evaluation and how well-calibrated they are.

5 Conclusion

In this paper, we examined the performance of four reasoning strategies – SC, reflexion, multi-agent debate, and tree of thoughts with respect to the oft-overlooked metric of budget. We used budget metrics of number of queries and number of tokens to reflect the different ways LLMs are used (querying LLM APIs and self-hosting models). We identified self-evaluation as an important aspect of multiple reasoning strategies and analyzed different prompting strategies to have the model evaluate its generations. We then introduced a new variant of SC weighted by the confidence of the model in its own answer after self-evaluation which showed gains for the MATH dataset. We also analyzed the answer generation and evaluation budget separately finding that in budget-constrained situations, it can be sometimes beneficial to use a weaker evaluator and still be able to obtain most of the performance one would get with a stronger evaluator.

6 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.

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*.

Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-aware neural code rankers. *Advances in Neural Information Processing Systems*, 35:13419–13432.

Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2023. 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*.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2023. Demystifying gpt self-repair for code generation. *arXiv preprint arXiv:2306.09896*.

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

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.

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*.

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*.

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.

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.

B Additional Result for Budget-aware Performance Metrics

B.1 Budget Metrics on All Datasets

Budget metrics on all datasets are shown in Figure 10 and 3

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 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 1b 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

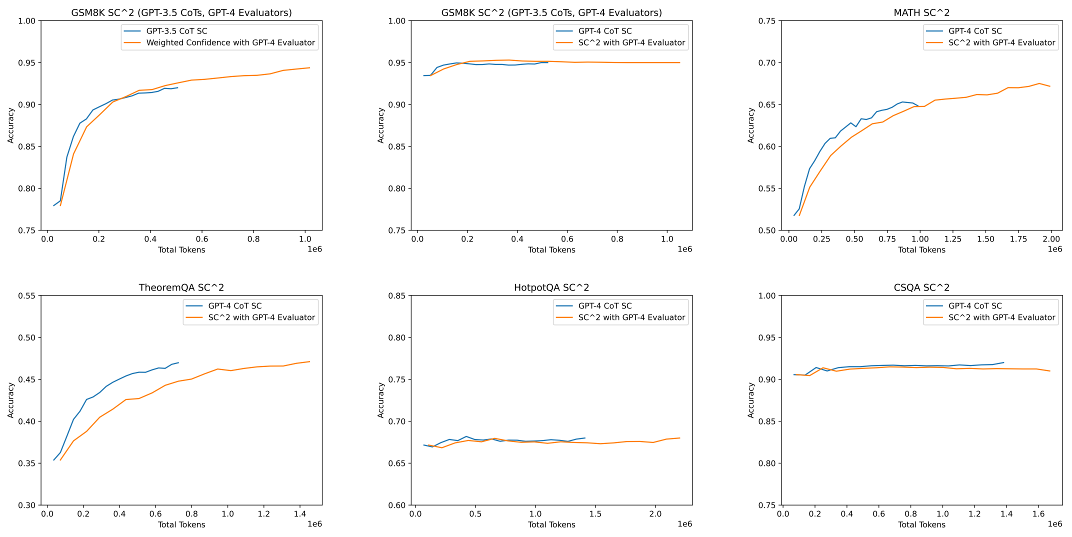


Figure 7: SC^2 with total tokens being the budget. There are sizable improvements in using our method SC^2 on math reasoning tasks.

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 10 illustrates 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

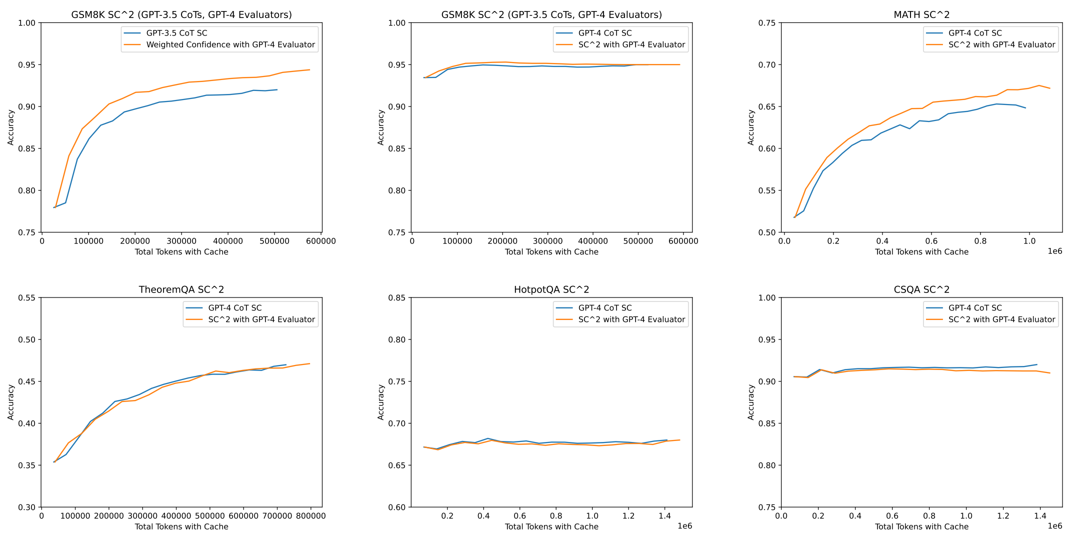


Figure 8: SC^2 with total tokens being the budget if caching is enabled.

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:

$$f(X_i = k) = \binom{n}{k} p_i^k (1 - p_i)^{n-k}, \tag{2}$$

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 $n/2$. This is expressed as:

$$P(\text{MV correct}|x_i) = \sum_{k=\lceil n/2 \rceil}^{n} \binom{n}{k} p_i^k (1 - p_i)^{n-k}. \tag{3}$$

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 11.

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

$$\lim_{n\to\infty} P(\text{MV correct}|x_i) = \begin{cases} 1 & \text{if } p_i > 0.5, \\ 0 & \text{if } p_i < 0.5. \end{cases} \tag{4}$$

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.

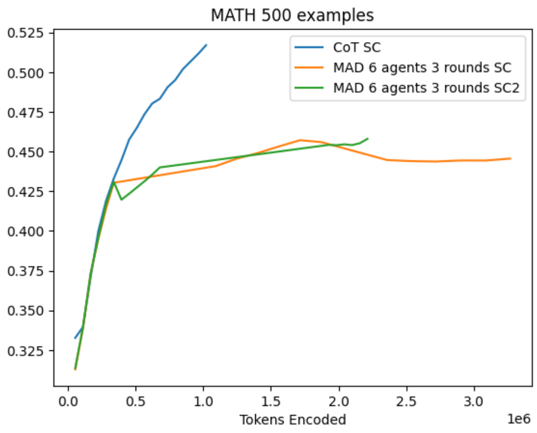
Over the set of all problems we consider, we place a beta distribution over p_i and integrate $P(\text{MV correct}|x_i)$ over the set of all problems to obtain $P(\text{MV correct})$. This can be expressed mathematically as:

$$\begin{aligned} &P(\text{MV correct}) \\ &\quad = \int_0^1 P(\text{MV correct}|p_i) \cdot f(p_i|\alpha, \beta)\, dp_i, \end{aligned} \tag{5}$$

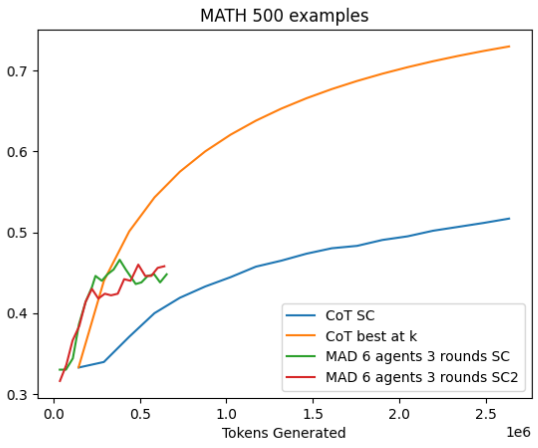
where $f(p_i|\alpha, \beta)$ is the probability density function of the beta distribution with parameters α and β.

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 12. This behavior explains our observation in real datasets directly.

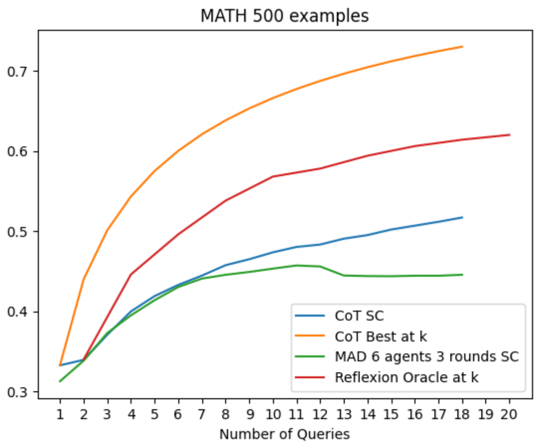
This also implies that for datasets where majority vote leads to consistent improvement, the distribution of p_i needs to be peaked greater than 0.5.



(a) Performance@Tokens Encoded



(b) Performance@Tokens Generated



(c) Performance@Number of Queries

Figure 9: Reasoning strategies performance versus different types of budgets. Here we demonstrate performance@Tokens Encoded, performance@Tokens Generated, performance@Number of Queries.

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.

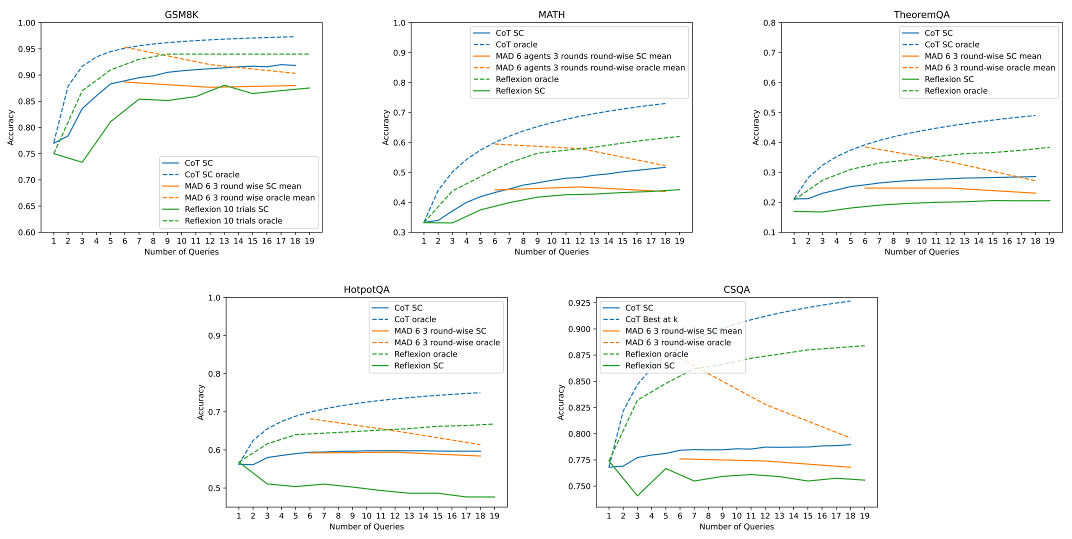
Let $p = (p_1, p_2, \ldots, p_K)$ be the probabilities associated with the K choices, and let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_K)$ be the parameters of the corresponding Dirichlet distribution. The probability of obtaining a correct majority vote for a given problem is then:

$$P(\text{MV correct}|p) = \sum_{k=\lceil n/2 \rceil}^{n} \text{multinomial}(k; n, p), \tag{6}$$

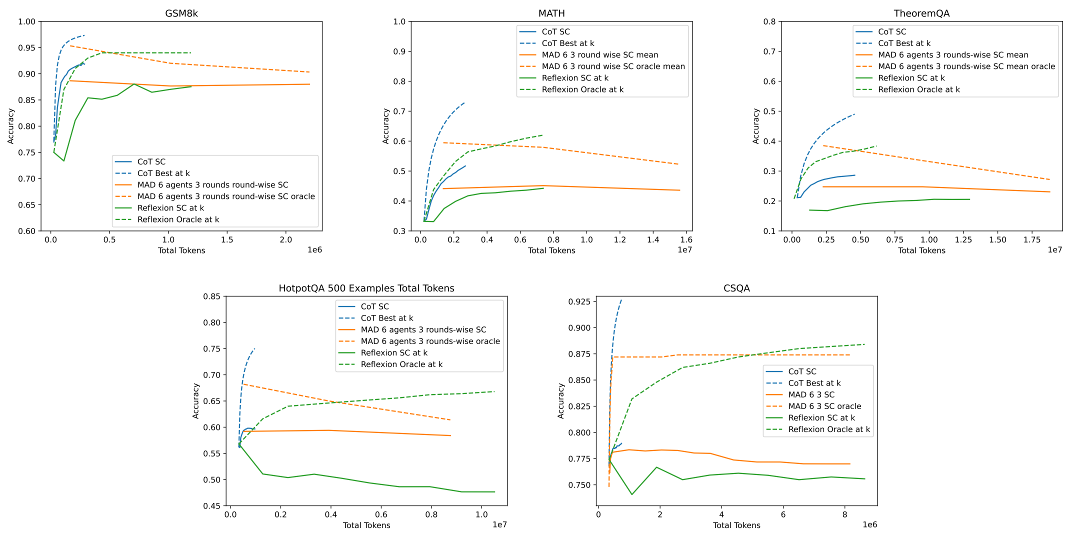
where the sum is taken over all combinations of k votes that would result in a majority for the correct choice.

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

$$P(\text{MV correct}) = \int P(\text{MV correct}|p) \cdot f(p|\alpha)\, dp, \tag{7}$$



(a)



(b)

Figure 10: (a)Performance@Number of Queries Plots for all 5 datasets. (b)Performance@Number of Tokens for all 5 datasets. All three methods CoT SC, MAD, and Reflexion are plotted. The solid lines are regular performances and the dashed lines are oracle performances (If any candidate solution is correct, then we count it as correct).

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 13). 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.

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**[3] - we ask the model to output Yes/No as to whether the answer is correct. We do this

[3]We also investigate a variant where we ask the model to think step by step before evaluating. While we see a small

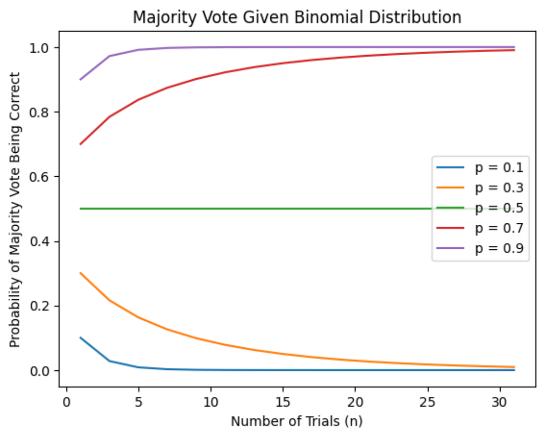


Figure 11: Probability of majority voting being correct for a given problem with varying p.

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 2: 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.

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.

D.2 Accuracy and Calibration of Self-Evaluation

In Table 3, we show the accuracy of self-evaluation of GPT-4 on the MATH dataset as a function of the number of samples, and in Figure 17 we show the calibration plots. The binary strategy performs the best by far in terms of both accuracy and calibration. The numerical confidence strategy is worse with the confidence probability strategy being the worst by far. One hypothesis as to why this might be the case is that the pretraining or RLHF data may be much more likely to have evaluations of various thoughts and answers where the evaluation consists of a binary (or multi-valued) *sentiment* compared to a numerical confidence rating or even less likely, a confidence probability. We also plotted calibration based on answer popularity in Figure 18.

increase in performance for such a strategy, it also necessitates a big increase in the token budget. Further analysis is in the Supplement.

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

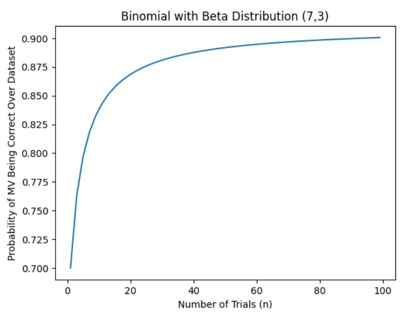
Table 3: Self-evaluation accuracy on MATH with different methods

D.3 Self-evaluation is correlated with problem difficulty

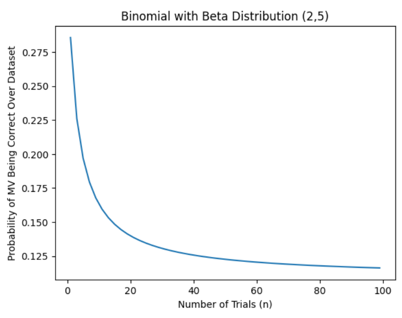
To get an understanding of whether models found it easier to evaluate answers to easier problems, we computed the following metric for a 100 problem subset of the GSM8K dataset. For each problem i, let a_{ij} be the jth answer. We had 20 sampled answers per problem. We computed the fraction c_i of answers 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.

E Terms and Licenses

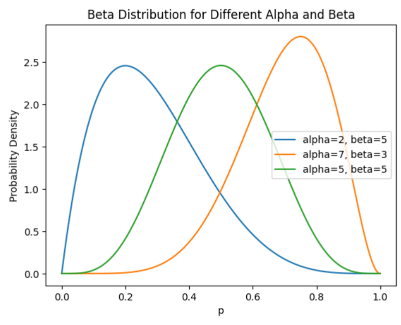
GSM8K, MATH, TheoremQA, CSQA are under the MIT license. HotpotQA is under the CC BY-SA 4.0 License. All the datasets and models are used for their intended use.



(a)

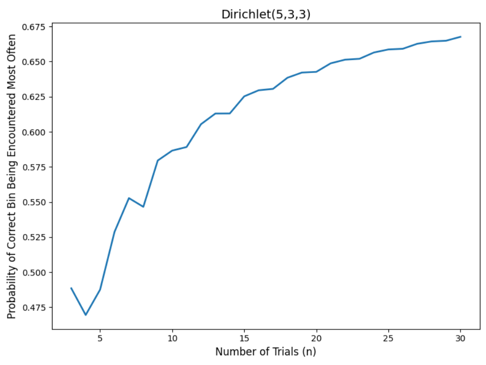


(b)

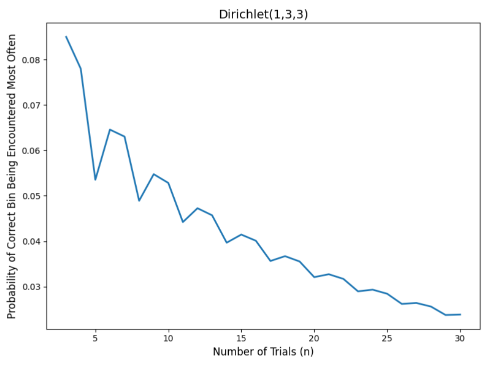


(c)

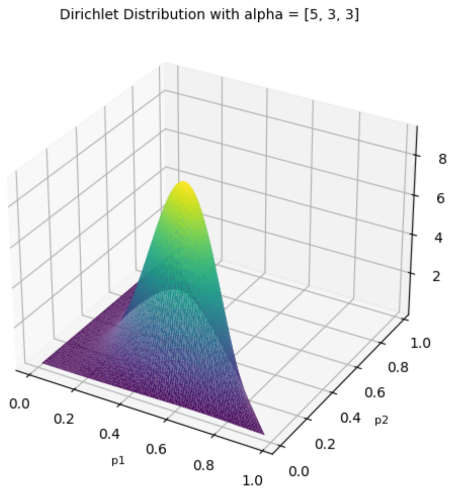
Figure 12: Convergence of majority vote under different beta distributions



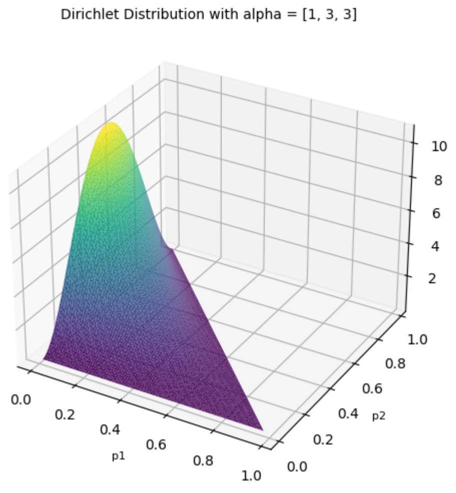
(a)



(b)



(c)



(d)

Figure 13: Convergence of majority vote under different Dirichlet distributions with $K = 3$

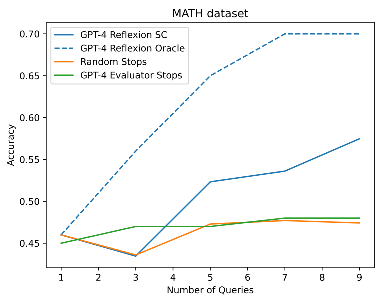


Figure 14: Ablation Study of the effect of Evaluator on Reflexion with GPT-4. Having an oracle evaluator still underperforms SC. However, it is better than having no evaluator.

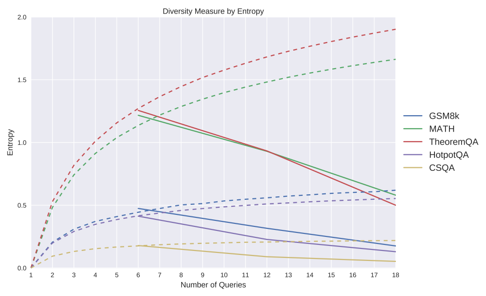


Figure 15: The diversity measure of the answers proposed on GPT-3.5. Dashed lines indicate CoT samples. Solid lines indicate the entropy of MAD with 6 agents and 3 rounds at each round.

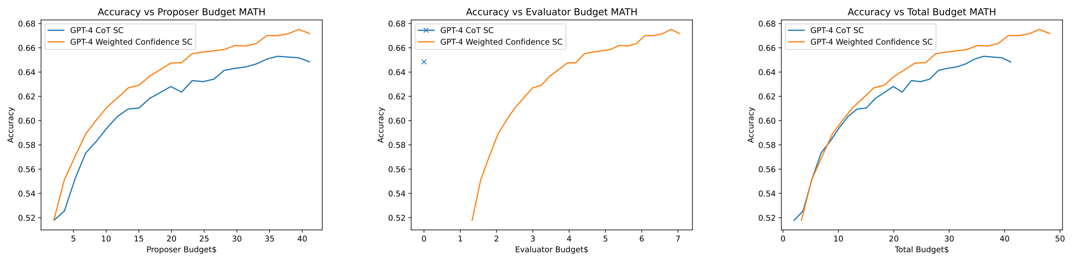


Figure 16: Separate proposer budget and evaluation budget on the dataset of MATH.

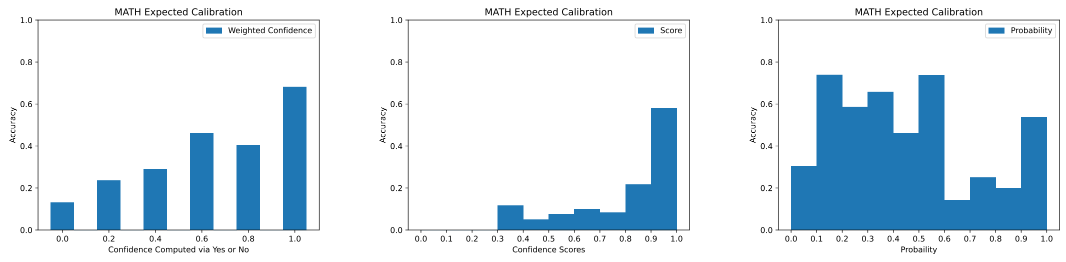


Figure 17: Calibration result for the math reasoning datasets. Three different self-evaluation methods are calibrated here.

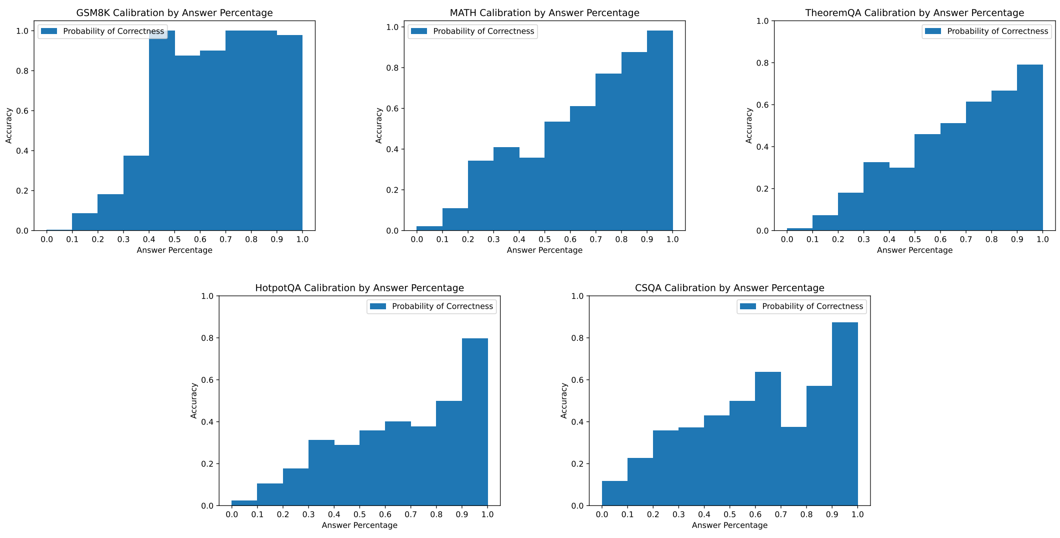


Figure 18: Calibration result binned by answer percentages.