WHEN LESS IS MORE: ONE STRATEGIC STEP IN LLM REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Addressing hallucinations in LLMs for Math World Problems (MWPs) is key to reliability and efficiency. We optimize the trade-off between accuracy and computation in CoT reasoning by verifying only the first step before proceeding. A *verifier* assesses correctness, halting generation if incorrect. This approach reduces token generation time by 30% with under 5% accuracy loss, while corrections improve accuracy by up to 10%. By skipping flawed reasoning early, our method balances accuracy and efficiency, cutting unnecessary computation.

1 INTRODUCTION

Large Language Models (LLMs) exhibit strong reasoning but are prone to hallucinations (Xu et al., 2024; Li et al., 2024; Huang et al., 2024b), especially in Math Word Problems (MWPs), where early errors can propagate. Detecting and correcting mistakes early is key to improving reliability and efficiency. Our approach optimizes this balance by introducing a verification mechanism that halts incorrect reasoning early, reducing computational overhead while preserving accuracy.

LLM reasoning methods include CoT (Wei et al., 2022; Lyu et al., 2024; Zhang et al., 2023), subquestion decomposition (Min et al., 2019; Radhakrishnan et al., 2023; Perez et al., 2020; Shridhar et al., 2022), and Program of Thought (PoT) (Chen et al., 2023). Subquestioning improves reliability, while verification methods like Chain-of-Verification and self-refinement refine outputs iteratively. Relevant works include QuestCoT (Jain et al., 2024), which prioritizes a strong first step but lacks verification, and ART (Shridhar et al., 2023b), which verifies reasoning but increases token costs. Our method integrates QuestCoT's emphasis on the first step with ART's verification, making it more efficient.

We prompt an LLM to generate only the first reasoning step, which a separate *verifier* evaluates. If incorrect, generation halts, preventing wasted computation. Refining incorrect first steps further enhances accuracy while reducing token usage. Evaluated on GSM8K (Cobbe et al., 2021), our method achieves an optimal trade-off between accuracy and efficiency, with potential applications in code generation and multiple-choice QA.

2 METHODOLOGY

Our methodology builds upon the foundational frameworks of the ART model (Shridhar et al., 2023b) and the QuestCoT approach (Jain et al., 2024), aiming to optimize the balance between accuracy and computational efficiency in solving MWPs using LLMs.

In our framework, we propose to use two models as follows:

- **Generator Model** (G): An LLM trained/prompted on the CoT training examples and generates reasoning steps for MWPs. During inference, it is prompted to produce only the first step initially.
- **Verifier Model** (V): A separate LLM (potentially of different sizes) trained on the Socratic version of dataset that involves subquestions for each reasoning step following Shridhar et al., it generates the first subquestion and subanswer during inference to verify the correctness of the generator's first step.

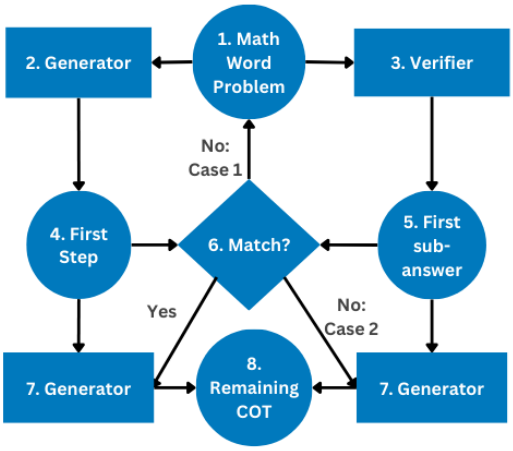


Figure 1: Flowchart outlining the process for solving math word problems. The generator and verifier collaboratively handle the first step generation (steps 1-5), followed by a match check (step 6). Depending on the match outcome, the process either completes with Case 1 or proceeds to generate the remaining solution (steps 7-8).

Below we have discussed the steps given in Figure 1:

- **First Step Generation**: The generator G receives an MWP Q and outputs the first reasoning step S_G.
- **Verification**: The verifier V processes Q and produces the verification step S_V.
- **Comparison**: We compare S_G and S_V:
  - If $S_G = S_V$, the first step is deemed correct, and G proceeds to generate the remaining steps.
  - If $S_G \neq S_V$, we choose between terminating the process or correcting S_G with S_V and continuing.

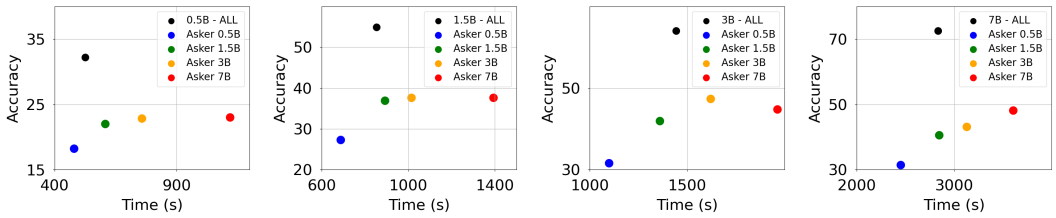
A detailed step by step analysis of our methodology is presented in Algorithm 1.

Algorithm 1 Overall proposed methodology

Require: Math Word Problem Q
Ensure: Final Answer or Decision to Skip
Generate first step $S_G \leftarrow G(Q)$
Generate verification step $S_V \leftarrow V(Q)$
if $S_G = S_V$ **then**
    Proceed to generate remaining steps $\{S_G^{(2)}, S_G^{(3)}, \dots\} \leftarrow G(Q, S_G)$
    Obtain final answer $A \leftarrow G$
else
    Option 1: Terminate and skip to next problem
    Option 2: Replace S_G with S_V and continue
    Proceed to generate remaining steps $\{S_G^{(2)}, S_G^{(3)}, \dots\} \leftarrow G(Q, S_V)$
    Obtain final answer $A \leftarrow G$
end if

3 Experimental Setup

We conducted experiments using Gemma (Team et al., 2024) (2B, 7B) and Qwen2.5 (Team, 2024) (0.5B, 1.5B, 3B, 7B) models, using each as both generator and verifier. To enable parameter-efficient



(a) Generator 0.5B (b) Generator 1.5B (c) Generator 3B (d) Generator 7B

Figure 2: Trade-off between processing time (in seconds) and accuracy (in percentage) for different generator-verifier setups in the *Incorrect FS* case for *Qwen*. Each subplot corresponds to a generator of a specific size (0.5B, 1.5B, 3B, 7B). The black dot represents the baseline scenario where the generator produces COT for all samples. Colored dots represent configurations where the verifier assists the generator, with verifier sizes ranging from 0.5B to 7B.

fine-tuning, we applied LoRA (Hu et al., 2021), allowing updates without modifying all model parameters.

The models were fine-tuned on GSM8K (Cobbe et al., 2021), with generators trained on the CoT set and verifiers on the Socratic set, focusing on generating the first subquestion-subanswer. We used Unsloth[1] to download models and VLLM (Kwon et al., 2023) for fast inference. Experiments were conducted on Nvidia RTX 3090/4090 GPUs (24GB VRAM).

All model pairings were explored within the same family, ensuring consistency, with one model acting as verifier and another as generator.

3.1 Evaluation Metrics

We define:

- **Accuracy** (A): The proportion of MWPs correctly solved, calculated as
$$A = \frac{\text{Number of Correct Answers}}{\text{Total Number of MWPs}}. \quad (1)$$
- **Time Cost** (T_{total}): The computational time for generating solutions, calculated as
$$T_{\text{total}} = N_{\text{verifier}} \times t_{\text{verifier}} + N_{\text{generator}} \times t_{\text{generator}}, \quad (2)$$
where N_{verifier} is the number of tokens generated by the verifier, t_{verifier} is the time per token for the verifier model, $N_{\text{generator}}$ is the number of tokens generated by the generator, and $t_{\text{generator}}$ is the time per token for the generator model.

4 Results

As outlined in Algorithm 1, incorrect first steps are handled by either terminating the chain (**Incorrect FS**) or replacing the first step with the verifier's (**Replaced FS**). In both cases, the first term in Equation 2 remains unchanged, while only the second term varies.

Figures 2, 3, and 4 summarize results for each model family. The X-axis represents total time cost from the formula, while the Y-axis shows final answer accuracy. The black dot marks the baseline accuracy and cost of the generator processing all test samples. Colored dots indicate performance with different-sized verifiers assisting in the first step.

4.0.1 Incorrect FS (Termination)

In this case, if the first step is incorrect, we terminate and skip to the next question. This approach aims to save computational time by not processing MWPs that are likely to result in incorrect answers. We see in Table 1 that identifying the errors earlier can lead to time gains but the accuracy is lower than the baseline.

[1] https://github.com/unslothai/unsloth

Table 1: Performance comparison of different generator-verifier setups in terms of processing time (in seconds) and accuracy (in percentage) for the *Qwen* family. *All Samples* indicates the baseline where the generator produces COT for all samples without using a verifier. Rows correspond to different verifier sizes (0.5B, 1.5B, 3B, 7B). The following sub-tables represent the *Incorrect FS* case.

Model	Time (secs), Acc
All Samples	(525, 32.22)
0.5B	(479, 18.27)
1.5B	(607, 22.06)
3B	(757, 22.89)
7B	(1119, 23.04)

(a) Generator 0.5B

Model	Time (secs), Acc
All Samples	(853, 54.96)
0.5B	**(687, 27.36)**
1.5B	(891, 36.92)
3B	(1014, 37.68)
7B	(1392, 37.68)

(b) Generator 1.5B

Model	Time (secs), Acc
All Samples	(1444, 64.06)
0.5B	**(1098, 31.61)**
1.5B	**(1360, 41.93)**
3B	(1620, 47.38)
7B	(1963, 44.80)

(c) Generator 3B

Model	Time (secs), Acc
All Samples	(2833, 72.55)
0.5B	**(2449, 31.46)**
1.5B	(2845, 40.56)
3B	(3130, 43.13)
7B	(3605, 48.21)

(d) Generator 7B

4.0.2 Replaced FS (Correction)

In this scenario, if the first step is incorrect, we replace it with the verifier's first step and continue generating the remaining CoT. This approach aims to improve accuracy at the expense of increased computational time. This is evident from Table 2 that using a larger verifier replaced answers can improve accuracy but increases overall compute time.

4.1 Key Findings

The key observations from our study are summarized as follows [2]:

- **Time Savings with Smaller Verifiers**: A smaller verifier than the generator reduces computational time with minimal accuracy loss. For a 1.5B generator, a 0.5B verifier cuts down time from 853s to 687s (Table 1b).
- **Marginal Accuracy Gains with Replacement**: Correcting incorrect first steps significantly increases time cost but yields only minor accuracy improvements (e.g., 32.22 $\rightarrow$ 36.08 with a 7B verifier; Table 2a).
- **Higher Accuracy with Larger Verifiers**: Replacing incorrect first steps with a larger verifier significantly improves accuracy for rejected samples. For a 0.5B generator, baseline accuracy (0.22) improves notably with larger verifiers (Table 4).

5 Conclusion

We propose a novel approach to optimizing accuracy and computational efficiency in CoT reasoning for MWPs by verifying the first reasoning step. Our results on GSM8K demonstrate that different model sizes and configurations can significantly reduce computation time with minimal accuracy loss or enhance accuracy with acceptable cost increases. Future work could explore adaptive strategies for dynamic termination and correction, reinforcement learning for optimization, and applications in domains like code generation and scientific problem-solving. Additionally, automating the decision of when to replace the answer or continue reasoning could refine the accuracy-cost tradeoff.

[2]Gemma and other experiments given in the Appendix.

REFERENCES

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. URL https://arxiv.org/abs/2211.12588.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models, 2023. URL https://arxiv.org/abs/2309.11495.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024a.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, November 2024b. ISSN 1558-2868. doi: 10.1145/3703155. URL http://dx.doi.org/10.1145/3703155.

Kushal Jain, Niket Tandon, and Kumar Shridhar. Well begun is half done: Importance of starting right in multi-step math reasoning, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. The dawn after the dark: An empirical study on factuality hallucination in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10879–10899, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.586. URL https://aclanthology.org/2024.acl-long.586.

Qing Lyu, Kumar Shridhar, Chaitanya Malaviya, Li Zhang, Yanai Elazar, Niket Tandon, Marianna Apidianaki, Mrinmaya Sachan, and Chris Callison-Burch. Calibrating large language models with sample consistency, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. Multi-hop reading comprehension through question decomposition and rescoring. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 6097–6109, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1613. URL https://aclanthology.org/P19-1613.

Masato Mita, Shun Kiyono, Masahiro Kaneko, Jun Suzuki, and Kentaro Inui. A self-refinement strategy for noise reduction in grammatical error correction. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 267–280, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.26. URL https://aclanthology.org/2020.findings-emnlp.26.

Sajad Mousavi, Ricardo Luna Gutiérrez, Desik Rengarajan, Vineet Gundecha, Ashwin Ramesh Babu, Avisek Naug, Antonio Guillen, and Soumyendu Sarkar. N-critics: Self-refinement of large language models with ensemble of critics, 2023. URL https://arxiv.org/abs/2310.18679.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations, 2024. URL https://arxiv.org/abs/2304.01904.

Ethan Perez, Patrick Lewis, Wen tau Yih, Kyunghyun Cho, and Douwe Kiela. Unsupervised question decomposition for question answering, 2020. URL https://arxiv.org/abs/2002.09758.

Ansh Radhakrishnan, Karina Nguyen, Anna Chen, Carol Chen, Carson Denison, Danny Hernandez, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Sam McCandlish, Sheer El Showk, Tamera Lanham, Tim Maxwell, Venkatesa Chandrasekaran, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Question decomposition improves the faithfulness of model-generated reasoning, 2023. URL https://arxiv.org/abs/2307.11768.

Kumar Shridhar, Jakub Macina, Mennatallah El-Assady, Tanmay Sinha, Manu Kapur, and Mrinmaya Sachan. Automatic generation of socratic subquestions for teaching math word problems. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4136–4149, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.277. URL https://aclanthology.org/2022.emnlp-main.277.

Kumar Shridhar, Harsh Jhamtani, Hao Fang, Benjamin Van Durme, Jason Eisner, and Patrick Xia. Screws: A modular framework for reasoning with revisions, 2023a.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ram Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. The art of llm refinement: Ask, refine, and trust, 2023b.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024. URL https://arxiv.org/abs/2403.08295.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hH36JeQZDaO.

Yuan Xia, Jingbo Zhou, Zhenhui Shi, Jun Chen, and Haifeng Huang. Improving retrieval augmented language model with self-reasoning, 2024. URL https://arxiv.org/abs/2407.19813.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models, 2024. URL https://arxiv.org/abs/2401.11817.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. Answering questions by meta-reasoning over multiple chains of thought, 2024. URL https://arxiv.org/abs/2304.13007.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=5NTt8GFjUHkr.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures, 2024. URL https://arxiv.org/abs/2402.03620.

A Appendix

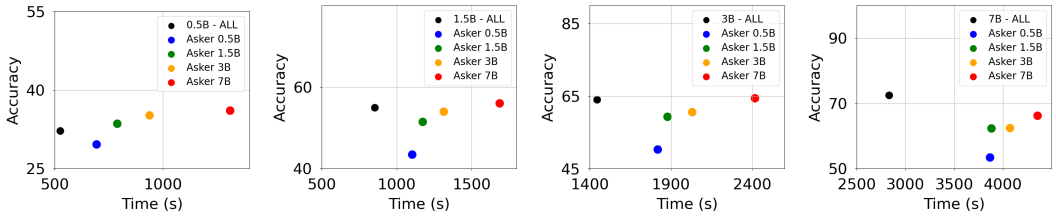
A.1 Related Work

Recently, there has been a surge of work in the field of self-refinement (Mousavi et al., 2023; Mita et al., 2020) and reasoning (Xia et al., 2024; Zhou et al., 2024) capabilities in Large Language Models (LLMs). There are three primary approaches to reasoning: Chain-of-Thought (CoT) (Wei et al., 2022; Lyu et al., 2024; Zhang et al., 2023); subquestion decomposition (Min et al., 2019; Radhakrishnan et al., 2023; Perez et al., 2020; Shridhar et al., 2022); and Program of Thought (Chen et al., 2023). The Program of Thought approach involves solving questions using coding, which is outside the scope of our project.

Building on these reasoning techniques, researchers have begun developing various self-refinement strategies. Most LLM refinement techniques leverage one of these methods (Madaan et al., 2023; Welleck et al., 2023; Paul et al., 2024; Huang et al., 2024a; Yoran et al., 2024) or combine multiple steps into a unified framework (Shridhar et al., 2023a). Meanwhile, Dhuliawala et al. introduces the idea of using subquestions to verify facts and mitigate hallucination. Similarly, Shridhar et al. assigns the task of verifying the Chain-of-Thought generated by the main LLM to a smaller language model, which carries out verification using subquestions.

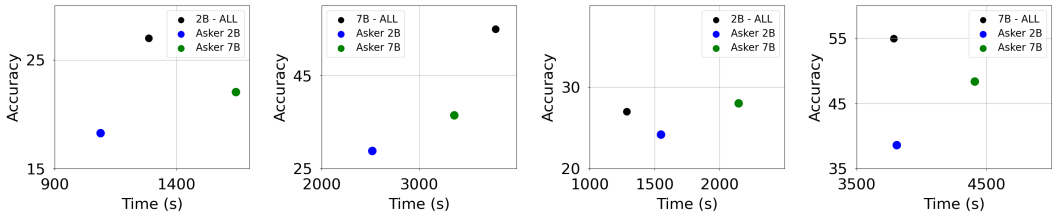
A key influence for our project was (Jain et al., 2024), which demonstrated that improving the first step in a reasoning chain significantly boosts final answer accuracy. This work shows that focusing on the initial step can lead to substantial performance improvements, aligning with our approach of verifying the first step to optimize both accuracy and computational efficiency.

Our study identifies a unique intersection between self-refinement algorithms and the strategic use of subanswers. By integrating the principles of QuestCoT (Jain et al., 2024) into the ART framework (Shridhar et al., 2023b), we aim to enhance the efficiency of LLM reasoning by verifying only the first step, thereby reducing unnecessary computations. This convergence forms the core of our investigation, allowing us to explore how these elements can be effectively combined to improve the performance of self-reasoning LLMs.



(a) Generator 0.5B (b) Generator 1.5B (c) Generator 3B (d) Generator 7B

Figure 3: Trade-off between processing time (in seconds) and accuracy (in percentage) for different generator-verifier setups in the *Replaced FS* case (below) for *Qwen*.



(a) Generator 2B (b) Generator 7B (c) Generator 2B (d) Generator 7B

Figure 4: The first two plots correspond to the *Incorrect FS* case, while the last two (c, d) represent the *Replaced FS* case for the *Gemma* family. In (a) and (b), the 2B verifier reduces time cost with only a slight decrease in accuracy compared to the baseline. Notably, in (b), the 7B verifier also achieves marginally lower time cost with minimal accuracy loss. In (c), the 7B verifier provides higher accuracy than the baseline but at the expense of a significant increase in time cost.

Table 2: This set of tables corresponds to the Replaced FS case for Qwen.

Model	Time (secs), Acc
All Samples	(525, 32.22)
0.5B	(693, 29.64)
1.5B	**(788, 33.58)**
3B	**(937, 35.17)**
7B	**(1310, 36.08)**

(a) Generator 0.5B

Model	Time (secs), Acc
All Samples	(853, 54.96)
0.5B	(1102, 43.51)
1.5B	(1172, 51.55)
3B	(1314, 53.98)
7B	**(1685, 56.10)**

(b) Generator 1.5B

Model	Time (secs), Acc
All Samples	(1444, 64.06)
0.5B	(1818, 50.34)
1.5B	(1878, 59.36)
3B	(2030, 60.65)
7B	**(2416, 64.44)**

(c) Generator 3B

Model	Time (secs), Acc
All Samples	(2833, 72.55)
0.5B	(3867, 53.44)
1.5B	(3882, 62.39)
3B	(4071, 62.47)
7B	(4355, 66.33)

(d) Generator 7B

Table 3: Sub-tables (a) and (b) represent the *Incorrect FS* case, while (c) and (d) correspond to the *Replaced FS* case. A consistent trend is observed: smaller verifiers reduce time cost, whereas larger verifiers improve accuracy. In (b), the baseline time cost is 3784 seconds, while the 2B and 7B verifiers achieve 2518 seconds and 3358 seconds, respectively. In (c), the 7B verifier improves accuracy to **28.05%**, compared to the baseline accuracy of **26.99%**.

Model	Time (secs), Acc
All Samples	(1286, 26.99)
2B	**(1088, 18.27)**
7B	(1644, 22.06)

(a) Generator 2B

Model	Time (secs), Acc
All Samples	(3784, 54.96)
2B	**(2518, 28.88)**
7B	**(3358, 36.46)**

(b) Generator 7B

Model	Time (secs), Acc
All Samples	(1286, 26.99)
2B	(1547, 24.18)
7B	**(2145, 28.05)**

(c) Generator 2B

Model	Time (secs), Acc
All Samples	(3784, 54.96)
2B	(3808, 38.66)
7B	(4408, 48.36)

(d) Generator 7B

Table 4: This table presents the performance of the *Qwen* family on rejected samples, where the generator's first step was incorrect, and those samples were replaced with the verifier's first step. The bold values for the 0.5B and 1.5B generators demonstrate that larger verifiers significantly improve accuracy. Conversely, the bold values for the 7B generator indicate a decline in accuracy when paired with smaller verifiers, showing that smaller verifiers are less accurate and negatively impact the performance of larger generators.

		0.5B Verifier	1.5B Verifier	3B Verifier	7B Verifier
0.5B Generator	**Baseline**	0.223 (163/729)	0.179 (113/633)	0.167 (102/610)	0.170 (108/634)
	Replaced	0.200 (146/729)	**0.232 (147/633)**	**0.256 (156/610)**	**0.267 (169/634)**
1.5B Generator	**Baseline**	0.464 (363/782)	0.386 (203/526)	0.393 (217/552)	0.397 (220/554)
	Replaced	0.271 (212/782)	0.335 (176/526)	0.373 (206/552)	**0.430 (238/554)**
3B Generator	**Baseline**	0.574 (430/749)	0.520 (293/564)	0.494 (219/443)	0.519 (255/491)
	Replaced	0.321 (240/749)	0.404 (228/564)	0.400 (177/443)	0.519 (255/491)
7B Generator	**Baseline**	0.667 (496/744)	0.619 (361/583)	0.623 (317/509)	0.578 (247/427)
	Replaced	**0.387 (288/744)**	**0.497 (290/583)**	**0.507 (258/509)**	0.569 (243/427)

Table 5: Here we have the results of the *Gemma* family for the rejected samples. The impact of using a larger verifier appears minimal for this family. However, further experiments with additional model sizes may be needed to confirm this behavior.

		2B Verifier	7B Verifier
2B Generator	**Baseline**	0.18 (128/703)	0.179 (133/743)
	Replaced	0.122 (86/703)	0.179 (133/743)
7B Generator	**Baseline**	0.47 (332/705)	0.428 (247/577)
	Replaced	0.177 (127/705)	0.282 (163/577)