

MAGICoRE: Multi-Agent, Iterative, Coarse-to-Fine Refinement for Reasoning

Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha,
Elias Stengel-Eskin, Mohit Bansal

UNC Chapel Hill

Abstract

Large language model (LLM) reasoning can be improved by scaling test-time compute with aggregation, i.e., generating multiple samples and aggregating over them. While improving performance, this strategy often reaches a saturation point beyond which additional compute provides no return. Refinement offers an alternative by using model-generated feedback to improve answer quality. However, refinement faces three key challenges: (1) **Excessive refinement**: Uniformly refining all instances can cause over-correction and reduce overall performance. (2) **Inability to localize and address errors**: LLMs struggle to identify and correct their own mistakes. (3) **Insufficient refinement**: Stopping refinement too soon could leave errors unaddressed. To tackle these issues, we propose MAGICoRE, a framework for Multi-Agent Iteration for Coarse-to-fine Refinement. MAGICoRE mitigates excessive refinement by categorizing problems as easy or hard, solving easy problems with coarse-grained aggregation, and solving the hard ones with fine-grained multi-agent refinement. To better localize errors, we incorporate external step-wise reward model scores, and to ensure sufficient refinement, we iteratively refine the solutions using a multi-agent setup. We evaluate MAGICoRE on Llama-3-8B and GPT-3.5 and show its effectiveness across seven reasoning datasets. One iteration of MAGICoRE beats Self-Consistency by 3.4%, Best-of- k by 3.2%, and Self-Refine by 4.0% even when these baselines use $k = 120$, and MAGICoRE uses less than 50% of the compute.¹

1 Introduction

Imagine a person taking a math exam with problems of varying difficulty; even before answering any question, an effective exam-taker might first distinguish between easier and more challenging

problems, deciding how much effort to budget for each one (Son and Metcalfe, 2000; Dodeen, 2015). To maximize their score, the student would likely spend minimal time on the easy problems and focus more on the harder ones, refining their answers where needed. Misallocating effort could not only waste time but even lower their score, as overthinking simple problems might lead to mistakes; similarly failing to dedicate enough thought to hard problems will lead to poor results. For Large Language Models (LLMs) performing reasoning tasks, several test-time approaches dedicate more computation to improve performance. These approaches sample multiple solutions to the same question and aggregate over the resulting answers (e.g. Self-Consistency (SC; Wang et al., 2022), Best-of- k sampling (Lightman et al., 2023; Sun et al., 2024; Wang et al., 2023)). However, applying aggregation strategies uniformly may waste computation on easier problems where the performance saturates quickly, and has diminishing gains on the harder problems even when more samples are generated. Refinement – where solutions are instead critiqued and improved upon during resampling – offers a possible avenue for breaking out of the aggregation rut. This mirrors human reasoning, where incorporating feedback (rather than simply retrying) can improve answers, often in an iterative fashion. For example, a teacher might improve a student’s understanding by providing multiple rounds of feedback on a test (Pan and Sana, 2021; Roediger and Karpicke, 2006; Wojcikowski and Kirk, 2013).

While refinement seems promising, it faces three key challenges that current work has yet to fully address, as outlined in Fig. 1: (1) **Excessive refinement**: the LLM must know when to refine and when not to. While refinement can help on incorrectly solved problems, uniformly refining all instances can cause over-refinement, where solutions that were already correct before refinement

¹Code: <https://github.com/dinobby/MAGICoRE>

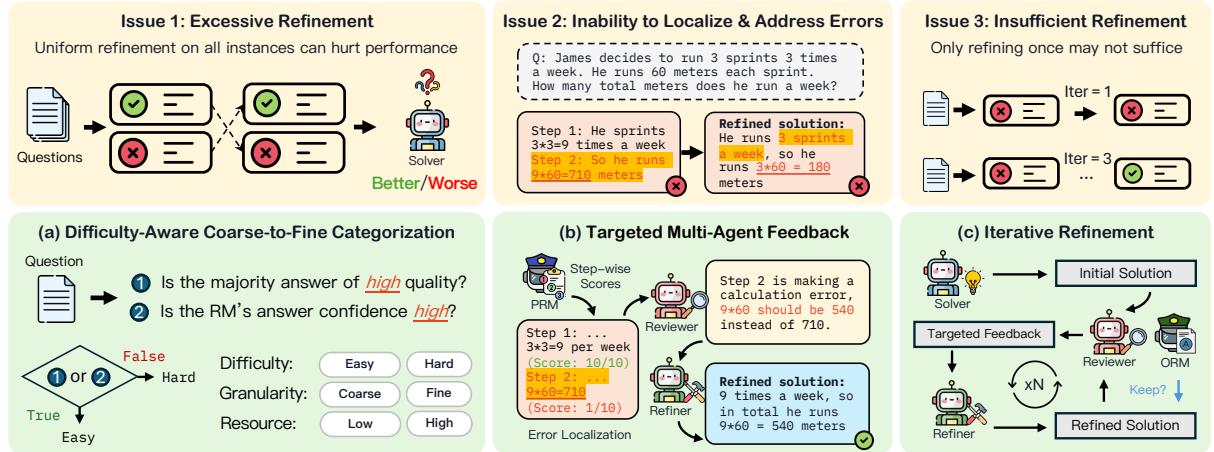


Figure 1: **Top:** Three main issues in refinement: (1) *Excessive refinement*; (2) *Inability to localize and address errors*; (3) *Insufficient refinement*. **Bottom:** Our joint solution to address these issues. MAGICORE adapts resource allocation based on problem difficulty and refines only when encountering hard problems to avoid excessive refinement. For hard cases requiring refinement, we employ a multi-agent setup that iteratively reviews and refines the solutions based on targeted feedback generated with step-wise PRM scores.

are “overthought” and flipped to incorrect, reducing the overall performance (Huang et al., 2024; Shridhar et al., 2024; Stechly et al., 2024). **(2) Inability to localize and address errors:** LLMs struggle to identify their own mistakes (i.e., steps needing refinement) and struggle to correct them without explicit instructions. **(3) Insufficient Refinement:** deciding how much refinement is needed is non-trivial – stopping refinement early could leave errors unaddressed, i.e., hard problems might be “underthought” by a single refinement iteration.

To enable better test-time scaling for aggregation and to address the three issues in refinement, we propose a unified solution, MAGICORE: Multi-Agent Iteration for Coarse-to-fine Refinement. As shown in Fig. 1, our approach leverages external global and local Reward Models (RMs) that enhance both aggregation and refinement processes. To avoid excessive refinement, we perform *selective refinement* (see Fig. 1(a)): we start by generating multiple reasoning chains from the LLM and scoring them with the RMs, using the entropy of the final answer distribution to classify examples as easy or hard. Given LLMs’ general inability to localize errors (Tyen et al., 2024), we leverage *step-by-step scores from a process reward model (PRM)* to help the LLM pinpoint low-scoring steps (which are likely to be incorrect); this process is shown in Fig. 1(b). Moreover, to help LLMs refine effectively once the errors have been localized, we propose a *multi-agent system* consisting of three agents: the Solver, the Reviewer, and the Refiner. For each problem, the Solver generates reasoning

chains, the Reviewer gives targeted feedback based on step-by-step RM scores, and the Refiner improves the solutions using this feedback. Finally, to address the issue of insufficient refinement, we *iterate the review-refine process*, using the quality and the entropy of the answers at each iteration as a stopping criterion (cf. Fig. 1(c)). While these three issues – selective refinement, error localization, and iterative refinement – might seem independent, addressing them jointly is more effective. Empirically, MAGICORE consistently outperforms baselines that tackle these issues in isolation, as confirmed by our ablation studies in Table 3.

We evaluate MAGICORE on seven reasoning datasets (including math, commonsense and logical reasoning) with two LLMs: Llama-3-8B and GPT-3.5. Notably, MAGICORE shows consistent improvements over all aggregation and refinement baselines across datasets and models. Specifically, just one iteration of MAGICORE on Llama-3-8B already outperforms Best-of- k sampling (Lightman et al., 2023) by 3.2% and Self-Consistency (Wang et al., 2022) by 3.4%, while using roughly half of the test-time compute. MAGICORE also outperforms a combination of Self-Refine (Madaan et al., 2023) and Self-Consistency by 4.0% and these trends also hold true for GPT-3.5. Moreover, MAGICORE effectively decides when to use refinement and when *not* to, leading to a 6.4% improvement over the strongest Best-of- k baseline on MATH (Hendrycks et al., 2021b), whereas uniformly applying refinement to all samples can result in a *drop* of 5.2%, highlighting the key role

played by selective refinement. MAGICORE also scales better with more iterations of refinement, scales to stronger base models and RMs, applies to both math/reasoning and commonsense tasks, and continues to improve while the baselines stagnate.

2 Methodology

In MAGICORE, we incorporate three types of models: (1) an LLM interchangeably performing three roles: the Solver, the Reviewer, and the Refiner, (2) an Outcome Reward Model (ORM) for generating global, solution-level correctness score, and (3) a Process Reward Model (PRM) for generating local step-by-step correctness scores. Both the ORM and PRM contribute to (1) assessing problem difficulty and (2) final answer selection via Weighted Self-Consistency (Li et al., 2023) (see Appendix B).

Overview. We present MAGICORE in Fig. 2. The process begins with the Solver generating k reasoning chains for each problem, followed by the ORM and PRM providing solution-level scores. Next, the input problem’s difficulty is classified based on two criteria (top-right of Fig. 2): (1) the quality of the majority answer and (2) the RMs’ answer confidence. Refinement is initiated only when the problem is deemed difficult, which occurs when the majority answer receives a low average RM score *and* the answer distribution is flat – indicating no single answer is significantly better than the others (i.e., low confidence). For these hard samples requiring refinement, we employ a multi-agent setup with three agents: the Solver, the Reviewer, and the Refiner (bottom of Fig. 2). The Reviewer uses the step-wise scores from the PRM to generate targeted feedback, and the Refiner then enhances the k solutions based on this feedback. The review-and-refine cycle can iterate multiple times to ensure sufficient and effective refinement.

2.1 Classifying Problem Difficulty

We categorize each problem’s difficulty as easy or hard using the following conditions (cf. Fig. 2).

a) Is the Majority Answer of High Quality? The Solver generates k solutions for the input question and we group them by their final answers. From the largest cluster of solutions, we calculate the average RM score and normalize it by the average score across all solutions, denoted as \mathcal{S}_{avg} . If $\mathcal{S}_{avg} \geq 0$ after normalization, this condition will be *true*, meaning the majority answer is already high-quality (as measured by both ORM and PRM

scores, see Appendix C), and hence no refinement is needed. Otherwise, we deem the example to be a possible candidate for refinement and evaluate the second condition below.

b) Is Reward Models’ Answer Confidence High?

In this condition, we check if the RMs are confident in any single answer; if this is not the case, the problem is a possible candidate for refinement. We measure confidence via the entropy of the distribution over answers, obtained by weighting answer clusters by their average RM scores, in line with Weighted Self-Consistency (Li et al., 2023).

Coarse-to-Fine Decision. If *either* of the conditions is met (the quality of the answer is high *or* the RMs are confident on an answer), an instance is marked as easy and delegated to the coarse-grained method: Weighted Self-Consistency (Li et al., 2023), using the sum of the solution-level scores generated by both ORM and PRM. Conversely, if *both* conditions are not satisfied, the instance is marked as hard and delegated to the fine-grained method (as described in Section 2.2), addressing Issue 1 (excessive refinement) by only refining solutions for the hard problems.

2.2 Fine-Grained Multi-Agent Refinement

For hard instances that fail both conditions, we need to employ refinement to unlock improvements (see the bottom part of Fig. 2). Our refinement setup has three agents: (1) the *Solver*, which generates the initial solution (2) the *Reviewer*, which takes step-wise PRM scores and a reasoning chain as input, and generates targeted feedback that pinpoints the errors within the chain, and (3) the *Refiner*, which takes the feedback generated by the Reviewer to refine the previous chain.

Solver generates k solutions. The Solver is responsible for generating the initial k solutions. Recall that in Section 2.1, we assess problem difficulty using k generated solutions. When a problem is classified as easy, we aggregate the k solution without refinement. When a problem is classified as hard, we can directly re-use the k solutions *already generated* by the Solver.

Reviewer generates targeted feedback. To assist the Reviewer in generating useful feedback to localize errors better (“Issue 2” in Fig. 1), we supply the Reviewer with external step-wise PRM scores for each step of the solution. The goal of the Reviewer is to incorporate these step-wise correctness scores to generate actionable feedback. We append these

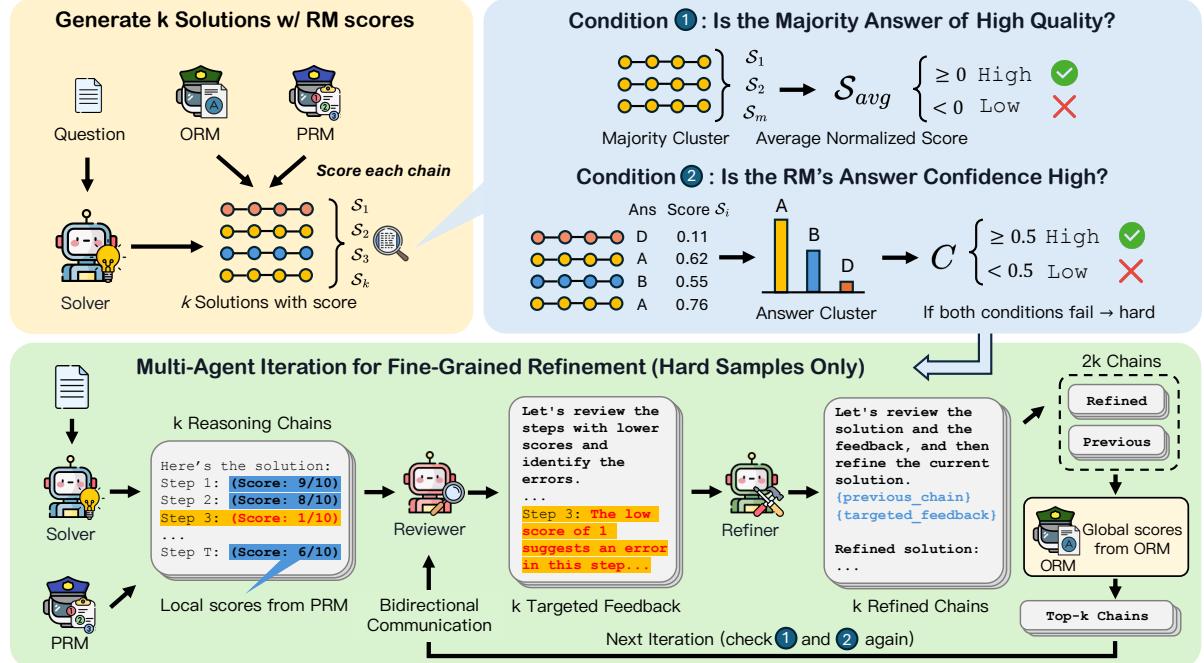


Figure 2: MAGI CORE classify problem difficulty based on two conditions: (1) the quality of the majority answer and (2) the RM’s answer confidence. A problem is considered hard when the majority answer receives a low average RM score *and* the answer distribution is flat (i.e., low confidence). For these hard problems, we employ a multi-agent setup – The Solver generates k reasoning chains, passing them to the PRM to pinpoint **errors**. The Reviewer turns scores into targeted feedback, and the Refiner improves the k solutions using the Reviewer’s feedback. This review-refine process repeats until either of the two conditions passes, or a maximum iteration is reached.

scores to the end of each step and pass the result to the Reviewer. That is, it takes a chain with the PRM scores as input, and is prompted to identify problematic steps that need refinement and possible ways to fix them.

Refiner improves solutions w/ feedback. Inspired by the finding that LLM can resolve errors when explicitly pointed out (Tyen et al., 2024), the Refiner agent focuses exclusively on *how the step should be modified* so as to resolve the error based on feedback from the Reviewer. That is, the Refiner uses the targeted feedback generated by the Reviewer to refine the reasoning chain that was generated by the Solver. The prompts for the Reviewer and the Refiner are shown in Appendix E.

Iterating the refinement process. For some hard instances, one round of refinement may be insufficient, as the Reviewer may have generated some irrelevant feedback or the Refiner may not have fixed the highlighted step adequately (“Issue 3” in Fig. 1). Thus, the Reviewer and the Refiner need to collaborate with each other over the course of multiple refinement iterations. To prevent excessive refinement, we re-evaluate the two conditions described in Section 2.1 in each iteration. The refinement continues until (a) one of the conditions

is met, or (b) a predefined maximum number of iterations is reached.

Final answer selection. The refinement process described above operates on all k chains simultaneously, producing k refined chains in each iteration. At the end of each iteration, we use the ORM to assess whether the refined solution has improved based on its global correctness score. In other words, by the end of each iteration, we have $2k$ reasoning chains – k initial and k refined – but retain only the top k based on their global ORM scores. Here we choose to base the decision on the ORM score because the targeted feedback is generated with PRM’s step-wise scores, so selecting the solution via another scoring model avoids overfitting. Finally, the answer is selected using Weighted Self-Consistency over these retained top k chains, at the end of each iteration.

3 Experimental Setup

Implementation Details. We develop MAGI CORE with Llama3-8B-Instruct (AI@Meta, 2024) and GPT-3.5-Turbo (OpenAI, 2022) as the base LLMs. Based on their strong performance on standard reward modeling evaluations (Lambert et al., 2024), we choose InternLM-7B (Cai et al.,

2024) as the ORM, and Math-Shepherd-7B (Wang et al., 2023) as the PRM for computing the RM scores. By default, we sample $k = 40$ reasoning chains in each iteration for MAGICORE, and the decoding temperature is set to 0.8. The maximum number of iterations is set to 3, with additional analysis in Fig. 3 and Table 15; we find that after 3, performance saturates, leading us to choose 3 given budget considerations. We compare against different categories of strong baselines as follows, and leave more comparisons against prompting-based baselines to Table 9 in the Appendix.

- **Vanilla Prompting.** The first baseline we compare to is zero-shot Chain-of-Thought (Wei et al., 2022); note that this only generates one reasoning chain per question without aggregation.
- **Iterative Prompting.** We also compare MAGICORE to an iterative prompting method, **Self-Refine (SR)** (Madaan et al., 2023), which refines the initial CoT answer via iteratively prompting the LLM to generate feedback and refine the previous output accordingly.
- **Aggregation-based Methods.** The third category generates multiple samples for each question. Here, we sample k solutions from the same LLM and select the final answer either via k -way **Self-Consistency (SC)** (Wang et al., 2022) or according to the highest ORM score (**Best-of- k**) (Lightman et al., 2023; Sun et al., 2024; Wang et al., 2023). Note that we give these baselines more samples than MAGICORE.
- **Iterative Baseline with Aggregation.** To enable a fair comparison, we also report a stronger version of self-refine by combining **Self-Refine and Self-Consistency (SR+SC)**, i.e., a baseline that is iterative, refines, and aggregates. Specifically, this baseline applies Self-Refine for k samples in parallel, and the final answer is derived by aggregating the k refined solutions.

Datasets. We evaluate MAGICORE mainly on five math reasoning datasets. Later in Section 4.2, we further show MAGICORE’s effectiveness on commonsense (ARC-challenge; Clark et al. (2018)) and logical reasoning (Date Understanding; Srivastava et al. (2022)) tasks. The first class of math datasets is math word problems: **GSM8K** (Cobbe et al., 2021), **SVAMP** (Patel et al., 2021), and **MATH** (Hendrycks et al., 2021b). GSM8K and SVAMP consist of grade school-level math problems, with 1,312 and 1,000 test samples. MATH comprises high-school math competitions span-

ning diverse topics and a total of 5,000 problems. Following previous works (Lightman et al., 2023; Wang et al., 2023), we evaluate MATH performance on a representative subset of 500 samples. We also evaluate on math splits of general benchmarks that test language models’ world knowledge and problem-solving abilities over various subjects such as **MMLU-Math** (Hendrycks et al., 2021a; Yue et al., 2024) and **SAT** (Zhong et al., 2023) with 974 and 220 test instances respectively.

4 Results and Analysis

4.1 Main Results

MAGICORE outperforms all baselines at the first iteration. We present our main results in Table 1. First, one iteration of MAGICORE already outperforms all baselines. Compared to aggregation-based methods, which generate multiple responses for each problem without refinement, MAGICORE improves over Best-of-120 by 3.2% (absolute) averaged across the five datasets on Llama-3-8B, despite using $2\times$ fewer samples. Note that our method’s first iteration only involves 40 samples for easy problems and 40 refined chains for the *subset* of hard problems, making our $k = 55$ on average. When compared to 120-way SC, our method shows an even greater average improvement of 3.3% on Llama-3-8B and 3.2% on GPT-3.5. Turning to refinement-based methods, we run them with up to 5 iterations and only report the best in Table 1 (denoted as “Best Iter”), leaving a more detailed comparison in Fig. 3 and Table 15. On average, MAGICORE shows 17.1% and 13.5% improvements over SR for Llama-3-8B and GPT-3.5. As SR alone is a weaker baseline without aggregating multiple samples, we also compare to SR + SC, and find that even with its best iteration, MAGICORE outperforms SR + SC by 5.4% (Llama-3-8B) and 4.9% (GPT-3.5) on average. This suggests that adaptively addressing challenging instances with targeted refinement improves overall performance, while reducing compute for easy problems.

MAGICORE continues to improve with more iterations. While MAGICORE already beats *all* baselines after the very first iteration, in Table 1, we also observe a clear upward trend in performance as the number of iterations increases. We illustrate this further in Fig. 3, which presents the accuracy across successive iterations. Our comparison includes Best-of- k and SR + SC with $k = 40$, with accuracy averaged across five datasets. We find that

	MMLU	MATH	SVAMP	GSM8K	SAT	Avg.
Llama3-8B-Instruct						
Zero-shot CoT	50.4	24.2	72.4	80.1	58.2	57.1
Self-Refine (Best Iter)	49.8	24.0	72.6	79.6	59.6	57.1
Best-of- k ($k = 120$)	62.6	41.4	88.7	90.1	72.4	71.0
k -way SC ($k = 120$)	63.0	40.6	89.8	90.3	70.5	70.8
Self-Refine + k -way SC (Best Iter)	62.3	41.0	89.2	90.3	68.0	70.2
MAGICORE (Iter=1)	67.3	46.0	91.4	91.1	75.0	74.2
MAGICORE (Iter=2)	68.4	47.2	91.1	92.3	76.4	75.1
MAGICORE (Iter=3)	68.9 (+5.6%)	47.8 (+5.2%)	91.3 (+1.7%)	91.6 (+1.3%)	78.2 (+5.8%)	75.6 (+4.3%)
GPT-3.5-Turbo						
Zero-shot CoT	62.5	37.2	78.1	78.5	76.8	66.6
Self-Refine (Best Iter)	61.1	37.4	77.9	78.4	77.1	66.4
Best-of- k ($k = 120$)	70.1	50.6	87.7	90.5	87.8	77.3
k -way SC ($k = 120$)	70.4	51.2	86.9	89.8	87.6	77.1
Self-Refine + k -way SC (Best Iter)	70.1	49.4	88.1	88.1	84.5	76.0
MAGICORE (Iter=1)	73.7	57.2	89.4	91.1	90.1	80.3
MAGICORE (Iter=2)	73.3	57.8	90.1	91.1	90.9	80.6
MAGICORE (Iter=3)	73.6 (+3.5%)	58.6 (+8.0%)	90.1 (+2.4%)	91.4 (+0.9%)	90.9 (+3.1%)	80.9 (+3.6%)

Table 1: Performance comparison of methods and models. (+x%) is compared to the strongest baseline (Best-of- k) shown in blue. Across models and datasets, MAGICORE consistently improves. Notably, MAGICORE surpasses all baselines after the *first iteration* of refinement, even when the baselines use a larger sample size ($k = 120$).

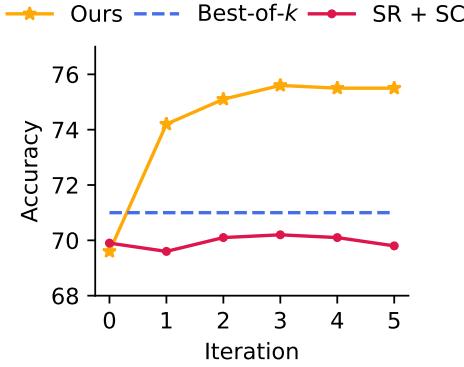


Figure 3: Comparison with baselines across iterations (avg. of 5 datasets with $k = 40$). Full results: Table 15.

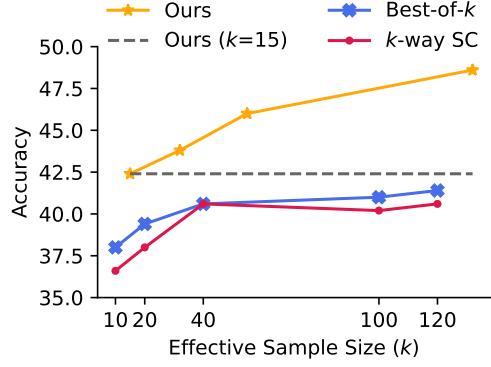


Figure 4: Comparison of MAGICORE, k -way SC, and Best-of- k with different k on MATH.

while SR + SC fluctuates around the same range of 70%, MAGICORE continues to improve and stabilize at the third iteration with 75.6% accuracy (with a noticeable 1.4% improvement compared to the first iteration). This highlights the importance of our iterative refinement and the ability to overcome insufficient refinement for hard instances, and indicates that the issue of over-refinement does not reappear in MAGICORE even after more iterations. **MAGICORE outperforms aggregation-based methods despite using less computation.** In Fig. 4, we further compare the cost and performance of MAGICORE with Best-of- k and SC on MATH and MMLU using Llama-3-8B, studying how performance changes as we increase the number of reasoning chains generated per question k .

Note that we sample k reasoning chains per question for baselines, whereas, in our method, we add k more samples in each iteration for a subset of hard problems, and plot the average number of samples in Fig. 4. The trend in Fig. 4 shows that MAGICORE consistently outperforms k -way SC and Best-of- k at any given k . Moreover, while SC saturates and stops improving at around $k = 40$, MAGICORE continues to improve with increasing k . Notably, MAGICORE with $k = 15$ already performs better than Best-of-120 and 120-way SC, highlighting the cost-effectiveness of our method. We also show that MAGICORE is also more cost-efficient in terms of token count in Fig. 5.

Method	MMLU	MATH	SVAMP	GSM8K	SAT	Avg.
Qwen2.5-Math-7B	73.9	78.8	91.8	94.9	92.3	86.3
k -way SC ($k = 40$)	81.3	87.0	95.5	97.2	97.3	91.7
k -way SC ($k = 120$)	82.0	86.8	95.4	97.3	97.3	91.8
Best-of- k ($k = 120$)	82.6	86.0	93.4	96.9	95.2	90.8
MAGICORE	84.6	91.4	95.8	97.3	97.3	93.3

Table 2: MAGICORE scales with the strength of reward models while also improves stronger base model like Qwen2.5-Math-7B. Here we use Skywork-Reward-Llama-3.1-8B as the ORM and Qwen-Math-PRM-7B as the PRM. Note that all the baselines are using the same models.

4.2 Additional Analyses

MAGICORE scales with stronger models. To evaluate the scalability of MAGICORE on more recent and capable models, we experiment with Qwen2.5-Math-7B (Yang et al., 2024), using Skywork-Reward-Llama-3.1-8B (Liu et al., 2024a) as the ORM and Qwen-Math-PRM-7B (Yang et al., 2024) as the PRM. As shown in Table 2, MAGICORE achieves the largest gains compared to Self-Consistency and Best-of- k under the same model setup. These results indicate that MAGICORE not only benefits stronger models like Qwen2.5-Math-7B but also generalizes well across datasets.

Method	MMLU	MATH
Only Address Issue 1	64.7	44.0
Only Address Issue 2	65.9	45.4
Only Address Issue 3	60.3	36.4
MAGICORE	68.9	47.8

Table 3: Ablation study on addressing each refinement (c.f. Fig. 1) issue one at a time.

All three issues must be addressed jointly. To investigate the importance of each refinement issue and component in MAGICORE, we conduct an ablation study to address each issue individually in Table 3 under the following settings: (1) *Only Address Issue 1 (Excessive Refinement)*: Here, we apply selective refinement only, without PRM step-wise scores for feedback generation and without iterations. (2) *Only Address Issue 2 (Inability to Localize and Address Errors)*: Here, we use PRM scores for feedback generation and refine all instances uniformly (i.e. no selective refinement) for one iteration. (3) *Only Address Issue 3 (Insufficient Refinement)*: Here, we iteratively refine all samples without incorporating PRM scores (i.e. no error localization) and without performing selective refinement. The results show that only addressing

one single refinement issue at a time leads to a performance drop, highlighting the need for a joint solution as we proposed in MAGICORE. We find that only addressing insufficient refinement (Issue 3) causes the highest drop in performance, as it fails to efficiently localize errors (without the help of PRM) and also performs excessive refinement.

PRM	ORM	Acc.
MS-7B	ILM-7B	47.8
QM-7B	ILM-7B	52.6
QM-72B	ILM-7B	55.4
MS-7B	SRL-8B	49.4

Table 4: Performance of MAGICORE with different RMs, which can be swapped in without modification.

Modularity of MAGICORE. In Table 1 we report performance using InternLM-7B (ILM-7B) as the ORM and Math-Shepherd-7B (MS-7B) as the PRM. Here, we illustrate the modularity of MAGICORE by incorporating different ORMs and PRMs; note that this can be done without changes to the code. In Table 4, we report the performance of MAGICORE on MATH when using different ORMs and PRMs, holding the other fixed. We test Qwen-Math PRM 7B and 72B (QM-7B and 72B; Zhang et al., 2025) as PRMs and Skywork-Reward-Llama-3.1-8B (SRL-8B; Liu et al., 2024a) as an alternate ORM. In all cases, we find that MAGICORE benefits from other RM selections, and that these changes can be made trivially.

Selective refinement avoids over-correcting and improves overall performance. In Section 1, we noted that excessive refinement could potentially hurt performance by flipping correct answers to incorrect ones. Here, we provide a quantitative analysis of this claim. Recall that we have two methods: coarse aggregation (Weighted SC) and fine refinement (multi-agent iteration) which we

Method	MMLU	MATH
Aggregation-Only	64.7	44.0
Refinement-Only	60.9	38.8
MAGICORE	67.3	46.0

Table 5: Comparison when uniformly adopting aggregation (i.e., Weighted SC) or refinement to *all instances*.

apply *selectively* depending on predicted problem difficulty (c.f. Section 2.1). In Table 5, we measure the performance of each method when applied uniformly to *all* instances, regardless of the problem difficulty. We find that uniformly applying refinement actually degrades performance; comparing Weighted SC (the “Aggregation-Only” in row 1) to refinement-only (row 2), we see that refining all samples leads to 3.8% and 5.2% drops on MMLU and MATH, respectively, pointing to the over-correction issue. Conversely, one iteration of our selective refinement (row 3) targets only the challenging instances where the weighted majority vote is unlikely to succeed, resulting in up to 2.6% improvement compared to uniformly applying aggregation (row 1). This demonstrates that our selective refinement not only avoids over-correction but also enhances overall performance by effectively allocating more resources to harder problems.

Refinement Variants	MMLU	MATH
LLM Self-Refinement	65.9	44.4
Random Step Score	66.4	43.8
ORM Score (No Step Score)	66.8	45.2
Ours (PRM Step Score)	67.3	46.0

Table 6: Refinement variants in MAGICORE. Using PRM scores for refinement performs the best.

PRM-based feedback enables better refinement. Having demonstrated that selectively applying refinement is crucial for achieving improvements, we now compare the refinement process with and without using a PRM. To this end, *without using a PRM*, we ask the LLM to generate an updated solution based on its own previous reasoning, referring to this as LLM Self-Refinement. Compared to MAGICORE in row 4 of Table 6, using LLM’s self-refinement (row 1) results in an average drop of 1.5%, indicating that using the LLM for refinement is less effective than using a PRM. To further examine how sensitive the refinement process is to the score quality, in row 2, we replace the actual PRM scores with random scores. The result is

Method	ARC	Date
Zero-shot	66.5	52.5
40-way SC	85.5	72.5
120-way SC	86.0	72.5
MAgICoRE (Iter = 1)	87.5	79.5
MAgICoRE (Iter = 2)	88.0	79.5
MAgICoRE (Iter = 3)	88.5	80.5

Table 7: MAGICORE also generalizes to commonsense reasoning and logical reasoning tasks.

worse than row 4, indicating that PRM scores help in localizing errors. Finally, we test whether the global ORM score can offer a similar advantage as using the local PRM score. Result in row 3 shows that it performs slightly worse than using the PRM score, suggesting that while global correctness is also a strong signal, local correctness scores help identify and correct errors more effectively.

MAGICORE generalizes to other domains.

Table 1 shows the benefits of MAGICORE on math reasoning; however, LLMs have been applied to a wide variety of tasks beyond math. Here, we explore expanding MAGICORE to other domains, specifically to a commonsense reasoning task: ARC-Challenge (Clark et al., 2018), and a logical reasoning task: Date Understanding (Srivastava et al., 2022). We sample 200 instances from each dataset and use GPT4o-mini as a PRM for the experiments, as existing standalone PRMs generally only exist for math. Specifically, we prompt GPT4o-mini to provide step-wise correctness scores *without any textual explanations* or reasoning, acting the same as a PRM. The prompt is provided in Appendix F. This approach ensures that our agents do not have access to explanations from a stronger model. We conduct this experiment with Llama3-8B-Instruct as the base LLM. Table 7 shows that MAGICORE transfers to commonsense and logical reasoning, outperforming 120-way SC by 2.5% and 8.0%, respectively.

Method	Accuracy
Zero-shot	72.0
40-way SC	79.2
40-way SC + PRM	79.4
MAgICoRE (Iter = 1)	80.2
MAgICoRE (Iter = 2)	80.4
MAgICoRE (Iter = 3)	80.4

Table 8: MAGICORE can also improve GPT4o-mini.

MAGICORE also improves stronger models like GPT4o-mini. Table 1 shows results with GPT-

3.5-Turbo; here, we show that MAGICORE scales to its stronger variant as well. Specifically, we run MAGICORE using GPT4o-mini on a subset of MATH data. Due to the high cost and the fact that Fig. 4 shows *decreasing* performance at $k = 120$ for MATH, we only compare to the 40-way SC with the weighted variation that incorporates PRM scores for vote weighting (40-way SC + PRM). Table 8 demonstrates that MAGICORE can also enhance stronger model’s performance, albeit with a smaller margin of improvement compared to Llama3-8B and GPT-3.5 shown in Table 1.

5 Conclusion

Building on the observation that different problems require varying amounts of computation, we introduced MAGICORE, a method that adaptively allocates more computational resources to more challenging problems and selectively applies refinement where appropriate, i.e., on harder problems. MAGICORE addresses three key issues in refinement: excessive refinement on easy examples, the inability of LLMs to detect and correct errors, and insufficient refinement on hard instances. Our approach tackles these issues by employing both global and local reward models to decide which samples to refine. We then incorporate local correctness scores to generate targeted feedback and an iterative multi-agent communication framework to refine solutions for hard problems. Results across five math datasets and two models show that our coarse-to-fine method consistently outperforms both coarse-grained aggregation and fine-grained refinement alone at any given budget, and even outperforms baselines using substantially more computation. In our ablations, we demonstrate the importance of selective refinement, showing that performance generally drops when refining all samples uniformly. We also highlight the role of iteration in our framework, showing increased performance across iterations even as baselines stagnate.

Limitations

Like all test-time scaling, MAGICORE improves performance by adding computation via additional samples, trading some efficiency for better performance. We show that MAGICORE makes better use of additional compute than the baselines by performing targeted refinement and thus better using inference-time compute; indeed, while baselines like Best-of- k and Self-Consistency stagnate

with additional compute, MAGICORE continues to improve. Nevertheless, our method increases the computational cost of inference, and relies on starting with a base number of samples to establish the difficulty and quality of existing solutions. In addition to requiring multiple solutions, MAGICORE uses feedback from both ORMs and PRMs to improve refinement. These models must be separately trained to provide rewards for a given domain and therefore do not exist for all problem types. However, we also note that MAGICORE is modular, and thus allows for newer and better ORMs and PRMs to be swapped in as they become available. MAGICORE is designed to improve the reasoning of LLMs, and thus has no additional risks beyond those inherent to LLMs generally.

Acknowledgments

We would like to thank the anonymous reviewers for their feedback. This work was supported by NSF-CAREER Award 1846185, NSF-AI Engage Institute DRL-2112635, DARPA MCS Grant N66001-19-2-4031, a Capital One Research Award, a Cisco Research Award, and a Google PhD Fellowship, and the Accelerate Foundation Models Research program. The views contained in this article are those of the authors and not of the funding agency.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. 2023. Let’s sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12375–12396.

AI@Meta. 2024. [Llama 3 model card](#).

Zheng Cai, Maosong Cao, Haojong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yingning Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong

Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahu Lin. 2024. [Internlm2 technical report](#). *Preprint*, arXiv:2403.17297.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. 2024a. [ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7066–7085, Bangkok, Thailand. Association for Computational Linguistics.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024b. [Are more llm calls all you need? towards scaling laws of compound inference systems](#). *arXiv preprint arXiv:2403.02419*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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*.

Hamzeh Dodeen. 2015. Teaching test-taking strategies: Importance and techniques. *Psychology Research*, 5(2):108–113.

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*.

Shangbin Feng, Weijia Shi, Yike Wang, Wenxuan Ding, Vidhisha Balachandran, and Yulia Tsvetkov. 2024. [Don’t hallucinate, abstain: Identifying LLM knowledge gaps via multi-LLM collaboration](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14664–14690, Bangkok, Thailand. Association for Computational Linguistics.

Xinyu Guan, Li Lyra Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. [rstar-math: Small llms can master math reasoning with self-evolved deep thinking](#). *Preprint*, arXiv:2501.04519.

Alex Havrilla, Sharath Raparthy, Christoforus Nalpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, and Roberta Railneau. 2024. [Glore: When, where, and how to improve llm reasoning via global and local refinements](#). *arXiv preprint arXiv:2402.10963*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. [Large language models cannot self-correct reasoning yet](#). In *The Twelfth International Conference on Learning Representations*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Muhammad Khalifa, Lajanugen Logeswaran, Moon-tae Lee, Honglak Lee, and Lu Wang. 2023. [Grace: Discriminator-guided chain-of-thought reasoning](#). In *ACL Findings*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. 2024. [Rewardbench: Evaluating reward models for language modeling](#). *arXiv preprint arXiv:2403.13787*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024a. [More agents is all you need](#). *arXiv preprint arXiv:2402.05120*.

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.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024b. [Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning](#). *arXiv preprint arXiv:2401.10480*.

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*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. 2024a. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*.

Dancheng Liu, Amir Nasseredine, Ziming Yang, Chen-hui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Chang-jae Lee, and Jinjun Xiong. 2024b. Large language models have intrinsic self-correction ability. *arXiv preprint arXiv:2406.15673*.

Li-Chun Lu, Shou-Jen Chen, Tsung-Min Pai, Chan-Hung Yu, Hung yi Lee, and Shao-Hua Sun. 2024. **LLM discussion: Enhancing the creativity of large language models via discussion framework and role-play**. In *First Conference on Language Modeling*.

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. *Advances in Neural Information Processing Systems*, 36.

OpenAI. 2022. **Chatgpt: Optimizing language models for dialogue**.

Steven C Pan and Faria Sana. 2021. **Pretesting versus posttesting: Comparing the pedagogical benefits of errorful generation and retrieval practice**. *Journal of Experimental Psychology: Applied*, 27(2):237–257.

Arkil Patel, Satwik Bhattacharya, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Henry L Roediger and Jeffrey D Karpicke. 2006. **Test-enhanced learning: Taking memory tests improves long-term retention**. *Psychological Science*, 17(3):249–255.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ramakanth Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. 2024. The art of llm refinement: Ask, refine, and trust. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Lisa K Son and Janet Metcalfe. 2000. Metacognitive and control strategies in study-time allocation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 26(1):204.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. Gpt-4 doesn’t know it’s wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*.

Kaya Stechly, Karthik Valmeeekam, and Subbarao Kambhampati. 2024. **On the self-verification limitations of large language models on reasoning and planning tasks**. *arXiv preprint arXiv:2402.08115*.

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. 2024. **Easy-to-hard generalization: Scalable alignment beyond human supervision**. *arXiv preprint arXiv:2403.09472*.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. 2024. **Llms cannot find reasoning errors, but can correct them given the error location**. *arXiv preprint arXiv:2311.08516*.

Manya Wadhwa, Xinyu Zhao, Junyi Jessy Li, and Greg Durrett. 2024. **Learning to refine with fine-grained natural language feedback**. *arXiv preprint arXiv:2407.02397*.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. 2023. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *CoRR*, *abs/2312.08935*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2024. **Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration**. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 257–279, Mexico City, Mexico. 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.

Ken Wojcikowski and Leslie Kirk. 2013. **Immediate detailed feedback to test-enhanced learning: an effective online educational tool**. *Medical Teacher*, 35(11):915–919.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. *Qwen2.5-math technical report: Toward mathematical expert model via self-improvement*.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. *Exchange-of-thought: Enhancing large language model capabilities through cross-model communication*. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153, Singapore. Association for Computational Linguistics.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Zhiyuan Zeng, Xiaonan Li, Tianxiang Sun, Cheng Chang, Qinyuan Cheng, Ding Wang, Xiaofeng Mou, Xipeng Qiu, and XuanJing Huang. 2024. *Aggregation of reasoning: A hierarchical framework for enhancing answer selection in large language models*. *arXiv preprint arXiv:2405.12939*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wen-hao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. *MAmmoTH: Building math generalist models through hybrid instruction tuning*. In *The Twelfth International Conference on Learning Representations*.

Di Zhang, Jiatong Li, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2023. Least-to-most prompting enables complex reasoning in large language models. *The Eleventh International Conference on Learning Representations*.

Appendix

A Related Work

Improving Reasoning by Aggregation. Self-Consistency (SC; Wang et al., 2022) generates k reasoning chains and marginalizes over the chains to obtain answer clusters; the most frequent answer

is selected as the final prediction. While simple and effective, it generates k solutions *for every sample*, as both past work and our work show that SC saturates when k increases (Chen et al., 2024b; Li et al., 2024a). Several studies adaptively determine the number of samples (k) required *for each instance* to address this (Aggarwal et al., 2023; Li et al., 2024b). However, the performance of such approaches is upper-bounded by SC – they address the cost issue but do not enhance overall performance. To surpass SC, Yin et al. (2024) propose using the LLM to evaluate answer clusters, taking into account both frequency and the LLM-evaluated quality of the answers. Instead, we propose using external RMs to decide between coarse-grained aggregation and fine-grained refinement, identify errors, and aid refinement; this allows us to improve over aggregation or refinement alone.

LLM-based Verification and Refinement. Past work mostly uses RMs for verification purposes (Li et al., 2023; Khalifa et al., 2023; Cobbe et al., 2021; Lightman et al., 2023). Havrilla et al. (2024) considers local correctness for refinement in a non-adaptive way and requires specific data curation with fine-tuning, while our work is adaptive and uses off-the-shelf global and local models. Another line of work has proposed using the LLM itself as a verifier, in place of an RM (Liu et al., 2024b; Zhang et al., 2024; Aggarwal et al., 2023; Madaan et al., 2023). However, recent work shows the inability of LLMs to “self-verify” their own reasoning (Huang et al., 2024; Stechly et al., 2023; Kamoi et al., 2024; Tyen et al., 2024; Kamoi et al., 2024). Hence, MAGICORE uses external global and local reward models (Wang et al., 2023; Cai et al., 2024) for selective coarse-to-fine refinement. Shridhar et al. (2024) trained specific models to decide when to refine and when to trust refined solutions. This contrasts with our method, where the decision to refine is based on a coarse-to-fine resource allocation method that differentiates easy from hard problems (for which we use global and local reward models), and where refinement is done based on off-the-shelf models. Past work has also used RMs to guide MCTS search for math problems (Guan et al., 2025). We do not compare to such methods, as their use of multiple rollouts makes generation-matched comparisons like the kind we do challenging. Moreover, while MAGICORE explores how reward models can effectively address issues in refinement, MCTS-based methods primarily investigate how reward models can guide the

search toward the final answer. These distinct goals make direct comparison less meaningful. We also iteratively refine guided by global correctness to ensure sufficient refinement, whereas Shridhar et al. (2024) refine only once. Wadhwa et al. (2024) propose a multi-agent detect-critique-refine pipeline for generation tasks. MAGICORE instead focuses on reasoning tasks and uses external RMs for selective coarse-to-fine refinement (whereas Wadhwa et al. (2024)’s detection uses the same metric as their evaluation, which is infeasible in reasoning where the metric – accuracy – requires access to the gold answer).

Multi-Agent Systems with LLMs. LLMs can be used in multi-agent systems, where the agents interact, collaborate, and compete (Wang et al., 2024; Lu et al., 2024; Feng et al., 2024). Related to our work, one line of multi-agent research focuses on structured debates or discussions between LLM agents, where the interaction helps refine and improve previously generated solutions (Du et al., 2023; Liang et al., 2023; Yin et al., 2023; Chen et al., 2024a). These studies show improvements over single-agent systems, but a major challenge in multi-agent systems is achieving a correct consensus among LLMs; external feedback can help prevent this consensus from aligning with the agents’ internal and possibly erroneous outputs. Therefore, MAGICORE’s multi-agent refinement incorporates external RMs for more objective scoring, enabling the generation of targeted feedback for better refinement.

B Self-Consistency and Weighted Self-Consistency

Self-Consistency (Wang et al., 2022) is a popular decoding method that uses majority voting to aggregate predictions from different reasoning chains, thus marginalizing over chains. It generates k solutions per question and selects the most frequent final answer from these samples. While simple and effective, this method assigns uniform weight to each reasoning chain, which fails to account for the quality of each solution. To address this limitation, Li et al. (2023) propose Weighted Self-Consistency, accounting for each solution’s quality. Formally, both Self-Consistency and Weighted Self-Consistency choose a final answer via:

$$\hat{y} = \arg \max_y \sum_{i=1}^k \mathbb{1}_{y_i=y} \mathcal{V}(q; r_i)$$

where $\mathcal{V}(\cdot)$ is a constant 1 in Self-Consistency and the quality measurement (e.g., RM score) in Weighted Self-Consistency. In MAGICORE’s final answer selection, we use the sum of the solution-level scores generated by both ORM and PRM as $\mathcal{V}(\cdot)$. Throughout MAGICORE, in cases where we need solution-level PRM scores (compatible with ORM scores), we accumulate the PRM step scores by taking their product (Sun et al., 2024), so that the aggregated PRM score corresponds to a solution.

C Details of the Conditions

Condition 1: Is the Majority Answer of High Quality? Given a problem q , to determine the difficulty of the problem at hand, the Solver generates k solutions $R = \{r_1, \dots, r_k\}$ and final answers $A = \{a_1, \dots, a_k\}$ per question and cluster the solutions by their final answer. This produces a partition \mathcal{A} with elements \mathcal{A}_i , where $\mathcal{A}_i = \{r_j \in R \mid a_j = a_i\}$. The majority cluster \mathcal{A}_g has the most “votes”, i.e., $\mathcal{A}_g = \operatorname{argmax}_{i \in |\mathcal{A}|} |\mathcal{A}_i|$. We evaluate the majority answer quality by both ORM and PRM separately but with the same procedure, as described below. First, we score every reasoning chain r_i within the majority cluster \mathcal{A}_g . Both ORM and PRM are able to produce a solution-level score, which we denoted as $\mathcal{S}_i^{\text{RM}}$. Note that we perform this check using ORM and PRM separately, but for simplicity, we use the same notation for solution-level score, which either comes from the ORM or the PRM. We calculate the average of the solution-level scores from the majority group:

$$\mathcal{S}_{\text{avg}}^{\text{RM}} = \frac{1}{|\mathcal{A}_g|} \sum_{i=1}^{|\mathcal{A}_g|} \mathcal{S}_i^{\text{RM}}$$

This average score informs us of the majority answer’s quality. To set a threshold, we normalize $\mathcal{S}_{\text{avg}}^{\text{RM}}$ by using the sample average RM scores (by computing $\mathcal{S}_i^{\text{RM}}$ for each sample and then take the average of these scores). Importantly, this process does not require any labeled data. After normalization, if the average reward of the majority group $\mathcal{S}_{\text{avg}}^{\text{RM}} \geq 0$, indicating that the quality of the majority answer is high, Condition 1 will be *true*. Otherwise, if $\mathcal{S}_{\text{avg}}^{\text{RM}} < 0$, Condition 1 will be *false*, suggesting that even the most frequent answer is of poor quality and that the instance might benefit from refinement.

Condition 2: Are Reward Models’ Answer Confidence High? Besides the quality of the major-

ity answer, we also consider whether the RMs are confident enough in any single answer among the answer clusters. Again we evaluate both ORM and PRM’s answer confidence separately but with the same procedure, as described below. First, the answer distribution is formed by (1) the frequency of each unique answer and (2) the total RM score of each answer cluster. We estimate the RM’s confidence according to this distribution. If the distribution is concentrated, meaning that only one answer cluster stands out, the RM’s answer confidence is treated as high. Conversely, if the distribution is diffused and the clusters’ scores are more uniform, then there is no single answer for which the RM has high confidence, i.e., the RM’s confidence is low. This motivates a targeted step-wise refinement process to select a more definite answer. Again we use both ORM and PRM to generate the solution-level score S_i^{RM} . Given the k reasoning chains generated along with the solution-level score, we compute the RM’s answer confidence (denoted as C) using the entropy of the answer cluster weighed by the RM scores, passing the result through a sigmoid function to normalize it onto $[0, 1]$. Formally, the calculation of the entropy can be expressed as:

$$H = - \sum_{i=1}^n p(\mathcal{A}_i) \log p(\mathcal{A}_i),$$

$$p(\mathcal{A}_i) = \frac{\sum_{i=1}^{|\mathcal{A}_i|} S_i^{\text{RM}}}{\sum_{\mathcal{A}_j \in \mathcal{A}} \sum_{k=1}^{|\mathcal{A}_j|} S_k^{\text{RM}}}$$

where n is the number of unique answers among the k chains, \mathcal{A}_i is the i -th answer cluster (a set of reasoning chains leading to the same answer) and \mathcal{A} is the set of all clusters. Each answer in a cluster is weighed by its unnormalized solution-level score S_i^{RM} . To normalize entropy onto a confidence scale, we invert it so that high entropy corresponds to low confidence. We then apply a sigmoid function $\sigma(\cdot)$, mapping the values to the range $[0, 1]$: $C = \sigma(\alpha * (1 - H))$. We set α to 2 to let the distribution stretch more evenly between 0 and 1. This transformation establishes 0.5 as a natural threshold for differentiating low and high confidence, thereby eliminating the need for any threshold tuning. That is, if an instance has $C \geq 0.5$, Condition 2 is *true*, meaning that the RMs are confident on a single answer cluster. Otherwise, if $C < 0.5$, Condition 2 is *false*, suggesting that the RMs’ uncertainty among the k chains is high, necessitating a finer refinement.

D Additional Experimental Results

Comparison with additional baselines. In addition to Table 1, we also compare with the following baselines: (1) 120-way SC + PRM: The product of step-wise PRM scores is used as the solution-level score. This score is then employed for weighted Self-Consistency, following (Li et al., 2023). (2) Self-correct + 120-way SC: We use the “Self-Correct RCI” prompt from (Kim et al., 2024) to generate 120 solutions per question, which are subsequently aggregated using Self-Consistency. (3) Least-to-Most + 120-way SC: We use the zero-shot Least-to-Most prompt from (Zhou et al., 2023) to generate 120 solutions per question, followed by aggregation via Self-Consistency. (4) Multi-Agent Debate + SC: Following Du et al. (2023), we conduct a three-agent debate over four rounds, repeating this process ten times. The final answers from these ten debates are aggregated using Self-Consistency, yielding 120 generations per question.

We use Llama3-8B-Instruct as the base model. Results show that a single iteration of MAGICORE already outperforms methods that rely on PRM for aggregation (120-way SC + PRM), as well as approaches like Self-Correction, advanced prompting, and multi-agent debate. On average, MAGICORE outperforms 120-way SC + PRM by 2.8% despite using fewer samples, highlighting the limitations of using PRM solely for aggregation. Additionally, MAGICORE exceeds Least-to-Most by 5.3%, showcasing superior adaptability to problem difficulty. Finally, MAGICORE surpasses Multi-agent Debate by 3.9%, indicating that our aggregation and refinement mechanisms scale more effectively at test time.

Separating Reviewer and Refiner roles outperforms combining these roles. In Appendix D, we examine the effects of combining the roles of Reviewer and Refiner by merging their prompts, instructing the model to simultaneously generate both feedback and a refined solution. This method is referred to as “Joint Roles”. In MAGICORE, the Reviewer and Refiner have distinct, clearly defined roles, which we refer to as the “Distinct Agents” approach. As before, the performance comparison is based on the first iteration, with all other variables held constant. Our findings show that maintaining separate roles (as in our multi-agent setup) leads to better performance, with the “Joint Roles” configuration resulting in a 0.6% drop in MMLU and a 1.2% decrease in MATH. The larger drop in

Method	MMLU	MATH	SVAMP	GSM8K	SAT	Avg.
120-way SC	63.0	40.6	89.8	90.3	70.5	70.8
120-way SC + PRM (Li et al., 2023)	65.4	44.6	90.8	90.7	72.5	72.8
Self-correct + 120-way SC (Kim et al., 2024)	62.1	38.6	86.2	88.1	65.6	68.1
Least-to-Most + 120-way SC (Zhou et al., 2023)	62.6	40.6	89.0	90.3	68.9	70.3
Multi-Agent Debate + SC (Du et al., 2023)	64.6	41.0	89.6	90.8	72.5	71.7
MAgICORE (Iter=1)	67.3	46.0	91.4	91.1	75.0	74.2
MAgICORE (Iter=2)	68.4	47.2	91.1	92.3	76.4	75.1
MAgICORE (Iter=3)	68.9	47.8	91.3	91.6	78.2	75.6

Table 9: Performance comparison with additional baselines using Llama3-8B-Instruct. Notably, MAGICORE with only one iteration outperforms all baselines despite using fewer samples.

Aggregation	MMLU	MATH
ORM-Only	66.9	45.4
PRM-Only	66.1	45.0
Both	67.3	46.0

Table 10: Ablation study on the final answer selection, using ORM-only, PRM-only or both.

	MMLU	MATH
Joint Roles	66.7	44.8
Distinct Agents (Ours)	67.3	46.0

Table 11: MAGICORE’s separation of the Reviewer and Refiner roles is more effective than combining them into a single role.

MATH suggests that its problems are more complex and often require extended reasoning, making the combined roles less effective, whereas maintaining separate roles proves to be more beneficial.

Ablations on reward models for final answer selection. We report MAGICORE up to three iterations in Table 1 and only report the best-performing iteration of Self-Refine + k -way SC. Here, we provide extended results in table Table 15. We also conducted another ablation study to evaluate the performance when using ORM, PRM, or a the summation of both scores for final answer selection. As shown in Appendix D, utilizing ORM’s global correctness score yields better results than aggregating PRM’s local correctness score. However, the best performance is achieved when both scores are combined for the final answer aggregation.

Reliable step-wise scores enable LLM refinement. To compare with an oracle PRM, we sample 500 instances from the Math-Shepherd dataset (Wang et al., 2023), which includes gold label cor-

Refinement Variants	Accuracy
No feedback (LLM self-refine)	48.30
Random PRM score	49.60
PRM predicted score	51.20
Oracle PRM score	52.40

Table 12: Comparison of different refinement variants in MAGICORE.

Criterion for Refinement	MMLU	MATH
Prompt (classification)	65.2	45.0
Prompt (confidence)	64.7	44.4
Condition 1 only	66.4	43.6
Condition 2 only	66.1	44.2
Cond. 1 & Cond. 2	67.3	46.0

Table 13: Different ways of detecting hard problems (i.e. criterion for refinement). Our two conditions, when used together, are the most effective.

rectness for each step. Besides the three settings we evaluated in Table 6, we also evaluate the oracle PRM score, where feedback uses the gold correctness labels. Appendix D shows that the oracle PRM score performs the best, followed by the predicted PRM score, suggesting that given reliable stepwise scores, LLMs can effectively refine their solutions and improve.

	P	R	F1
Random	68.4	49.6	57.5
Prompt-based (classification)	65.9	10.3	17.8
Prompt-based (confidence)	0.0	0.0	0.0
MAGICORE	86.3	67.6	75.8

Table 14: The Precision (P), Recall (R) and F1 of the model predicted problem difficulty.

Effectiveness of the two conditions for classifying problem difficulty. In MAGICORE, we use reward models to classify each instance as easy or hard. Given that the RMs are also fine-tuned LLMs, we investigate whether prompting the LLM to perform this classification directly could replace the external RMs. We compare two settings in the first two rows, where we prompt Llama3-8B-Instruct to evaluate the difficulty of an instance. In the first setting (classification), the LLM generates a binary label. In the second setting (confidence), it produces a confidence score ranging from 0 to 1, indicating whether refinement is required – that is, whether the example is easy or hard. Results in Table 13 show that the LLM is less effective at determining instance difficulty compared to a reward model, as evidenced by a performance drop of 1.6% – 2.6%. In rows 3 and 4, we also examine the performance when only one of the conditions of MAGICORE (c.f. Section 2.1) is used to decide difficulty. Specifically, when only condition 1 is applied, an instance is classified as hard if the majority answer’s quality is low. Conversely, when only condition 2 is applied, an instance is classified as hard if the RM’s answer confidence is low, regardless of the majority answer’s quality. Results indicate that while each condition individually outperforms LLM self-verification, combining both yields the best performance. Indeed, in Appendix D, we find that MAGICORE’s assessment of problem difficulty shows the highest agreement with human-annotated labels.

Model-Predicted vs. Human-Annotated Problem Difficulty. We analyze the model’s prediction of problem difficulty. Specifically, we utilize the MATH dataset, which includes human-annotated difficulty levels ranging from 1 to 5, with higher levels indicating increased problem complexity. For our analysis, we split the problems as follows: (1) Easy: Levels 1 and 2 and (2) Hard: Levels 4 and 5. We exclude Level 3 problems to create a clearer distinction between easy and hard categories. We compare the overlap between our model’s predictions and these human-annotated levels. We treat hard as the positive label. The results are presented in Table 14. To provide a comparative analysis, we include: (1) a random baseline that assigns easy and hard labels at random, (2) a prompt-based baseline that directly prompts the LLM to classify the problem difficulty, and (3) another prompt-based baseline that prompts the LLM

to generate a confidence score when answering, where a confidence score of ≥ 0.5 is classified as “easy”. Results show that our conditions substantially outperform all baselines. Interestingly, the prompt-based methods perform worse than the random baseline, particularly the one relying on confidence scores, which classifies *all problems as easy*; this method scores 0 for both precision and recall since we treat “hard” as the positive label, so it has 0 true positives. This suggests that our framework is highly effective at distinguishing true problem difficulty based on the conditions outlined in our methodology.

Token Count Analysis. In Fig. 4, we are mainly comparing the number of generations (k) per question with the baselines. To provide a more granular analysis, we break down the generations at the token level and compare costs in terms of token counts. The results are detailed in Fig. 5. For Self-Consistency, the input tokens are counted only once per question, as it uses the same input to generate k responses. In contrast, the input token count for MAGICORE includes all prompts across all agents – Solver, Reviewer, and Refiner. We also include the token count for the ORM and PRM in MAGICORE. Since the cost of input tokens is typically $0.25 \times$ that of output tokens², we present the normalized total token cost as $0.25 \times \text{input} + 1 \times \text{output}$. Results in Fig. 5 show that (1) scaling Self-Consistency from $k = 40$ to $k = 120$ largely increases token overhead while yielding marginal improvements. (2) MAGICORE exhibits superior scalability, achieving substantially higher performance gains with increased token usage. On MMLU, MATH and SAT, we observe a clear upward trend with an increased token count; MAGICORE consistently improves with additional tokens (unlike SC which tends to stagnate). (3) The first iteration of MAGICORE outperforms 120-way SC fewer tokens.

Discussion of external reward models. External reward models play an important role MAGICORE and are used in the solutions to all three problems (excessive refinement, inability to localize and address errors, and insufficient refinement). While MAGICORE does utilize external reward models, our framework is modular and can readily incorporate new reward models as they emerge. As

²See <https://openai.com/api/pricing>, <https://www.anthropic.com/pricing#anthropic-api>, and https://ai.google.dev/pricing#1_5pro

	MMLU	MATH	SVAMP	GSM8K	SAT	Avg.
Llama3-8B-Instruct						
Zero-shot CoT	50.4	24.2	72.4	80.1	58.2	57.1
Self-Refine (Iter=1)	49.6	24.6	72.0	79.0	57.7	56.3
Self-Refine (Iter=2)	50.2	23.8	72.8	79.6	59.3	57.1
Self-Refine (Iter=3)	49.8	24.0	72.6	79.6	59.6	57.1
Best-of- k ($k = 120$)	62.6	41.4	88.7	90.1	72.4	71.0
k -way SC ($k = 120$)	63.0	40.6	89.8	90.3	70.5	70.8
Self-Refine + k -way SC (Iter=0)	62.1	40.4	88.6	90.1	68.2	69.9
Self-Refine + k -way SC (Iter=1)	61.3	40.6	88.9	89.7	67.7	69.6
Self-Refine + k -way SC (Iter=2)	62.7	40.0	88.9	90.1	68.6	70.1
Self-Refine + k -way SC (Iter=3)	62.3	41.0	89.2	90.3	68.0	70.2
Self-Refine + k -way SC (Iter=4)	62.1	41.4	89.2	90.1	67.7	70.1
Self-Refine + k -way SC (Iter=5)	62.7	40.4	88.6	89.7	67.7	69.8
MAGICORE (Iter=1)	67.3	46.0	91.4	91.1	75.0	74.2
MAGICORE (Iter=2)	68.4	47.2	91.1	92.3	76.4	75.1
MAGICORE (Iter=3)	68.9	47.8	91.3	91.6	78.2	75.6
MAGICORE (Iter=4)	68.9	48.0	91.3	91.1	78.2	75.5
MAGICORE (Iter=5)	68.4	48.0	91.1	91.6	78.2	75.5
GPT-3.5-Turbo						
Zero-shot CoT	62.5	37.2	78.1	78.5	76.8	66.6
Self-Refine (Iter=1)	62.4	37.4	77.7	77.4	77.3	66.4
Self-Refine (Iter=2)	61.6	37.6	78.6	77.9	76.9	66.5
Self-Refine (Iter=3)	61.1	37.4	77.9	78.4	77.1	66.4
Best-of- k ($k = 120$)	70.1	50.6	87.7	90.5	87.8	77.3
k -way SC ($k = 120$)	70.4	51.2	86.9	89.8	87.6	77.1
Self-Refine + k -way SC (Iter=0)	69.4	49.8	86.9	88.1	85.6	76.0
Self-Refine + k -way SC (Iter=1)	69.8	49.0	87.1	88.3	85.0	75.8
Self-Refine + k -way SC (Iter=2)	70.1	49.4	88.1	88.1	84.5	76.0
Self-Refine + k -way SC (Iter=3)	69.6	48.8	87.3	87.8	85.2	75.7
Self-Refine + k -way SC (Iter=4)	69.8	48.4	87.1	87.1	85.0	75.5
Self-Refine + k -way SC (Iter=5)	69.6	48.6	87.3	87.4	84.5	75.5
MAGICORE (Iter=1)	73.7	57.2	89.4	91.1	90.1	80.3
MAGICORE (Iter=2)	73.3	57.8	90.1	91.1	90.9	80.6
MAGICORE (Iter=3)	73.6	58.6	90.1	91.4	90.9	80.9
MAGICORE (Iter=4)	73.6	58.0	89.9	91.4	90.9	80.8
MAGICORE (Iter=5)	73.4	57.6	89.4	91.1	90.9	80.5

Table 15: Extended version of Table 1. Here we show all more iterations for Self-Refine + k -way SC and MAGICORE. While SR + SC does not show a clear improvement with more iterations, MAGICORE continues to improve, peaking at the third iteration.

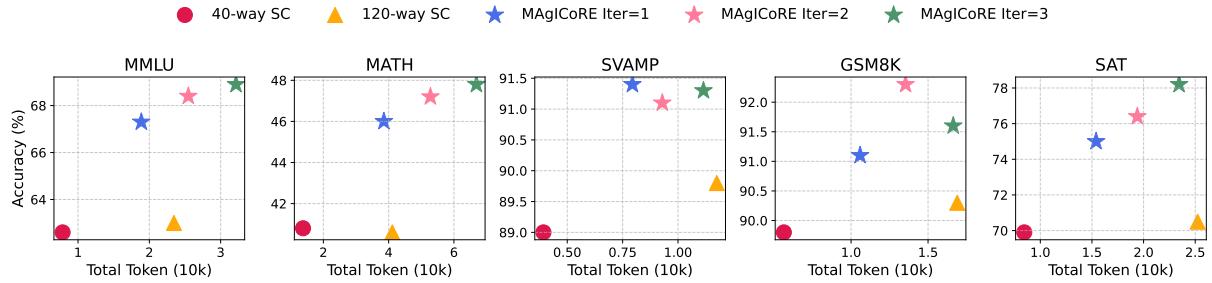


Figure 5: Token count comparison with Self-Consistency across different datasets. Scaling Self-Consistency from $k = 40$ to $k = 120$ introduces substantial token overhead while providing marginal improvements. In contrast, MAGICORE demonstrates superior scalability, delivering much higher performance gains with an increased token count. Notably, the first iteration of MAGICORE consistently outperforms 120-way SC while using fewer tokens.

the community is actively advancing the performance of reward models evidenced by a benchmark for reward models (Lambert et al., 2024), MAGICORE is thus complementary to and enhanced by progress in reward modeling, rather than constrained by it. While it is possible to train a custom error-identification model, this approach is often data-dependent and prone to obsolescence. In contrast, MAGICORE’s modular design overcomes this limitation by enabling the integration of new state-of-the-art models as they become available. Moreover, our experiments in Table 7 indicate that when trained reward models are unavailable, we can use sufficiently strong LLMs in place of trained RMs. For example, we use GPT4o-mini as a reward model for commonsense and logical reasoning.

E Prompt for the Reviewer and the Refiner

Reviewer's Prompt

Your task is to provide step-by-step feedback to the current solution.

You will be given a math problem and a current solution, along with the scores for each step based on its correctness.

- You will find (Score: n/10) at the end of each step.
- The maximum (best) score is 10, which means that this step is 100% correct (and 0% incorrect).
- The minimum (worst) score is 0, which means that this step is 100% incorrect (and 0% correct).
- Pay attention to the steps having scores lower than 6, and carefully identify the errors in those steps.
- Provide your explanation of the error and how it can be fixed. DO NOT propose a new solution, just the explanation.

Question:

{question}

Current Solution:

{solution}

Let's review the steps with lower scores and identify the errors.

{feedback}

Refiner's Prompt

Your task is to fix the error in the given solution, based on the teacher's feedback.

- After reviewing the solution and feedback, provide a better solution to fix the mistakes you found.

Question:

{question}

Current Solution:

{solution}

Teacher Feedback:

{reviewer_feedback}

Revised Solution:

Let's review the solution and the feedback, and then refine the current solution.

After reviewing the solution and feedback, let's solve this math problem again.

{refined_solution}

F Prompt for GPT4o-mini as PRM

GPT4o-mini as PRM Prompt

You will be given a question and its corresponding step-by-step solution.

Your task is to evaluate the correctness of each step in the solution by assigning a score between 1 and 10.

Your response should be formatted as follows:

score: <comma separated scores>

where the length of the list corresponds to the number of steps in the solution.

Each step is clearly separated in the solution by <Step 1:>, <Step 2:> ... <Step k:>, your output score list should be the exact same length as k.

Question: {question}

Solution: {solution}

score:

G Dataset Statistics and Licenses

We provide the sample sizes and licenses of the datasets used in this work in Table 16. All the datasets are in English and all datasets are used in a fashion consistent with their intended use.

	Sample Size	License
MMLU-Math (Hendrycks et al., 2021a)	974	MIT License
MATH (Hendrycks et al., 2021b)	500	MIT License
SVAMP (Patel et al., 2021)	1,000	MIT License
GSM8K (Cobbe et al., 2021)	1,312	MIT License
SAT (Zhong et al., 2023)	220	MIT License
ARC-Challenge (Clark et al., 2018)	200	CC-by-SA-4.0
Date Understanding (Srivastava et al., 2022)	200	Apache License

Table 16: The statistics and licenses of the datasets used in this study.

H Qualitative Examples of Refinement

QUESTION: Josh decides to try flipping a house. He buys a house for \$80,000 and then puts in \$50,000 in repairs. This increased the value of the house by 150%. How much profit did he make? (Gold answer: **70000**)

SOLVER'S OUTPUT (BEFORE REFINEMENT):

Step 1: Calculate the value of the house after repairs. The house is worth 80,000 after repairs. It is 150% higher than before. So the value is $(1 + 150/100) * 80,000$. $80,000 * (3/2)$ is 120,000. So the house is worth 120,000 after repairs. (Score: 2/10)

Step 2: Calculate the cost of repairs. The cost of repairs is 50,000. (Score: 2/10)

Step 3: Calculate the profit. The profit is the difference between the value after repairs and the cost before repairs. $120,000 - 80,000 - 50,000$ is 30,000 (**incorrect**).

REVIEWER'S OUTPUT:

Step 1: The low score of 2 suggests an error in this step. The problem states that the value of the house increased by 150%, not that the value is 150% higher than before.

Step 2: The cost of repairs is indeed \$50,000.

Step 3: The low score of 3 suggests an error in this step. The profit would be: 200,000 (value after repairs) - 80,000 (cost before repairs) - 50,000 (cost of repairs) = 70,000

REFINER'S OUTPUT:

Step 1: Calculate the value of the house after repairs. The house is worth 80,000 after repairs. It is increased by 150%. So the value is $80,000 \times (1 + 150/100) = 80,000 \times 2.5 = 200,000$.

Step 2: Calculate the cost of repairs. The cost of repairs is 50,000.

Step 3: Calculate the profit. The profit is the difference between the value after repairs and the cost before repairs. $200,000 - 80,000 - 50,000$ is 70,000 (**correct**).

Table 17: A qualitative example of our multi-agent refinement on GSM8K using Llama3-8B-Instruct . The score at the end of each step is provided by the PRM.

QUESTION: The Greek army contained two types of soldiers: the upper class and the lower class soldiers. If there were a total of 5 upper class soldiers, and 10 lower class soldiers in a certain part of Athens, and the battle of Thermopylae demands a force of 4 upper class soldiers and 8 lower class soldiers, how many different battalions can be sent? (Gold answer: **225**)

SOLVER'S OUTPUT (BEFORE REFINEMENT):

Step 1: Find out the number of upper class soldiers left. We have 5 upper class soldiers and we need 4. So we have $5 - 4 = 1$ upper class soldiers left. (Score: 9/10)

Step 2: Find out the number of lower class soldiers left. We have 10 lower class soldiers and we need 8. So we have $10 - 8 = 2$ lower class soldiers left. (Score: 9/10)

Step 3: Find out the number of different battalions. We have 1 upper class soldier and 2 lower class soldiers. The number of different battalions is the number of ways we can arrange 1 upper class soldier and 2 lower class soldiers. This is 3. (Score: 5/10) (**incorrect**).

REVIEWER'S OUTPUT:

Step 1 and Step 2: The scores of 9/10 suggest that these steps are mostly correct. However, we should be careful when calculating the number of upper and lower class soldiers left.

Step 3: The low score of 5 suggests an error in this step. The problem asks for the number of different battalions, not the number of ways to arrange the soldiers. The correct solution is to find the number of ways to choose 4 upper class soldiers from the available 5 and 8 lower class soldiers from the available 10.

REFINER'S OUTPUT:

Step 1: Find out the number of upper class soldiers needed. We need 4 upper class soldiers.

Step 2: Find out the number of lower class soldiers needed. We need 8 lower class soldiers.

Step 3: Find out the number of different battalions. We can choose 4 upper class soldiers from the 5 available in $5C4 = 5$ ways. We can choose 8 lower class soldiers from the 10 available in $10C8 = 45$ ways. The number of different battalions is the number of ways to choose 4 upper class soldiers and 8 lower class soldiers, which is $5 \times 45 = 225$. (**correct**).

Table 18: A qualitative example of our multi-agent refinement on MATH using Llama3-8B-Instruct . The score at the end of each step is provided by the PRM.

I Qualitative Examples of Over-Correction

QUESTION: A new program had 60 downloads in the first month. The number of downloads in the second month was three times as many as the downloads in the first month, but then reduced by 30% in the third month. How many downloads did the program have total over the three months? (Gold answer: **366**)

SOLVER'S OUTPUT (BEFORE REFINEMENT):

Step 1: Determine the number of downloads for the second month. The number of downloads for the second month was three times as many as the first month, so it was $60 * 3 = 180$. (Score: 10/10)

Step 2: Determine the number of downloads for the third month. There was a reduction of 30%, so we calculate $180 * (1 - 0.30) = 180 * 0.7 = 126$. (Score: 10/10)

Step 3: Determine the total number of downloads for the three months. There were 60 downloads in the first month, and 180 downloads in the second month. So the total number of downloads for the first two months is $60 + 180 = 240$. Then, there were 126 downloads in the third month. So the total number of downloads over the three months is $240 + 126 = 366$. (Score: 6/10) (**correct**).

REVIEWER'S OUTPUT:

Step 1-2: The high scores of 10 suggest that these steps are correct.

Step 3: The score of 6 suggests that this step is mostly correct. However, the improvement you could make is to explicitly state the calculation for the total number of downloads, rather than just stating the result.

REFINER'S OUTPUT:

Step 1: Determine the downloads in the second month. Three times 60 is $3 \times 60 = 180$. Then 30% of 180 is $180 \times 0.3 = 54$. $180 - 54$ is 126. So 126 downloads in the second month.

Step 2: Determine the total downloads over the three months. 60 downloads in the first month, and 126 downloads in the second month, and $126 - 54 = 72$ downloads in the third month. $60 + 126 + 72$ is 258. The answer is 258. (**incorrect**).

Table 19: A qualitative example of over-correction on GSM8K using Llama3-8B-Instruct.

QUESTION: If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2) + f(-1) + f(0)$? Express your answer as a common fraction. (Gold answer: $\frac{14}{3}$)

SOLVER'S OUTPUT (BEFORE REFINEMENT):

Step 1: Determine the values of $f(-2)$, $f(-1)$, and $f(0)$ by plugging the values into the expression for $f(x)$.

$$f(-2) = \frac{3(-2)-2}{-2-2} = \frac{-6-2}{-4} = \frac{-8}{-4} = 2,$$

$$f(-1) = \frac{3(-1)-2}{-1-2} = \frac{-3-2}{-3} = \frac{-5}{-3} = \frac{5}{3},$$

$$f(0) = \frac{3(0)-2}{0-2} = \frac{0-2}{-2} = \frac{-2}{-2} = 1. \text{ (Score: 10/10)}$$

Step 2: Add the values of $f(-2)$, $f(-1)$, and $f(0)$ to obtain $f(-2) + f(-1) + f(0) = 2 + \frac{5}{3} + 1$

$$\frac{6}{3} + \frac{5}{3} + \frac{3}{3} = \frac{14}{3}.$$

Final Answer: The final answer is $\boxed{\frac{14}{3}}$. I hope it is correct. (Score: 7/10) (correct).

REVIEWER'S OUTPUT:

Step 1: The scores of 10/10 suggest that this step is correct.

Step 2: The high score of 7 suggests that this step is mostly correct. However, it might be beneficial to include intermediate steps for clarity.

REFINER'S OUTPUT:

Step 1: We can calculate $f(-2)$, $f(-1)$, and $f(0)$ using the given expression for $f(x)$: $f(-2) = \frac{3(-2)-2}{-2-2} = \frac{-8}{-4} = 2$,

$$f(-1) = \frac{3(-1)-2}{-1-2} = \frac{-5}{-3} = \frac{5}{3},$$

$$f(0) = \frac{3(0)-2}{0-2} = \frac{-2}{-2} = 1.$$

Step 2: Now we can add these values together: $f(-2) + f(-1) + f(0) = 2 + \frac{5}{3} + 1 = \frac{3}{1}$

Final Answer: The final answer is $\boxed{\frac{3}{1}}$. I hope it is correct. (incorrect).

Table 20: A qualitative example of over-correction on MATH using Llama3-8B-Instruct .