Bag of Tricks for Inference-time Computation of LLM Reasoning

Fan Liu[1], **Wen-Shuo Chao**[1], **Naiqiang Tan**[2] , **Hao Liu**[1*]
[1]Hong Kong University of Science and Technology (GuangZhou)
[2]Didichuxing Co. Ltd
{fliu236,wschao829}@hkust-gz.edu.cn
tannaiqiang@didiglobal.com, liuh@ust.hk

Abstract

With the advancement of large language models (LLMs), solving complex reasoning tasks has gained increasing attention. Inference-time computation methods (e.g., Best-of-N, beam search) are particularly valuable as they can enhance reasoning performance without modifying model parameters or requiring additional training. However, these techniques come with implementation challenges, and most existing methods remain at the proof-of-concept stage with limited practical adoption due to their computational complexity and varying effectiveness across different tasks. In this paper, we investigate and benchmark diverse inference-time computation strategies across reasoning tasks of varying complexity. Since most current methods rely on a proposer-verifier pipeline that first generates candidate solutions (e.g., reasoning solutions) and then selects the best one based on reward signals (e.g., RLHF rewards, process rewards), our research focuses on optimizing both candidate solution generation (e.g., instructing prompts, hyperparameters such as temperature and top-p) and reward mechanisms (e.g., self-evaluation, reward types). Through extensive experiments (more than 20,000 A100-80G GPU hours with over 1,000 experiments) across a variety of models (e.g., Llama, Qwen, and Mistral families) of various sizes, our ablation studies reveal that previously overlooked strategies can significantly enhance performance (e.g., tuning temperature can improve reasoning task performance by up to 5%). Furthermore, we establish a standardized benchmark for inference-time computation by systematically evaluating six representative methods across eight reasoning tasks. These findings provide a stronger foundation for future research. The code is available at `https://github.com/usail-hkust/benchmark_inference_time_computation_LLM`.

1 Introduction

Large language models (LLMs) have demonstrated remarkable reasoning capabilities, enabling them to tackle increasingly sophisticated tasks in fields such as science, mathematics, and coding [39, 5]. While scaling model size and expanding high-quality training datasets have significantly driven these advancements, researchers are actively exploring complementary approaches to further enhance model performance. Inspired by human problem-solving behavior—where individuals often dedicate more time deliberating on complex problems to improve their decisions—there is growing interest [27] in leveraging *inference-time computation* (e.g., utilizing additional computation during testing to enhance the performance of reasoning tasks) to strengthen the reasoning abilities of LLMs.

*Correspondence to Hao Liu.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

Table 1: Configuration of inference-time computation methods. Inference-time computation involves two main steps: generating candidate solutions (e.g., chain-of-thought reasoning) and selecting the optimal solution. These configurations, though significant, often receive less attention and lack standardization.

Methods	Domain	*N	Prompt	Temperature	Top-p	Trajectory Selection	Verification	Reward Model
Q* [30]	Math/Code	6	COT	0.9/0.2	*	Best-of-N	Step-level reward (score value)	Policy model
MALT [23]	Math/CommonsenseQA	27	COT	0.3	*	Best-of-N	Trajectory-level reward	Verify agent
GenRM [39]	Math/Algorithmic	1-32	COT	*	*	Best-of-N	Step-level reward (yes or no)	Verifier
HiAR-ICL [33]	Math/CommonsenseQA	5	Automatic COT	0.8	0.9	MCTS	Step-level reward	Self-Verify
CPO [40]	Math/Fact Verification	5	COT	0.9/0.4	0.9	Tree of Search	Step-level reward (score value)	Self-Verify
AutoMathCritique [35]	Math	1-128	COT	0.7	*	Step-level refine	Step-level reward (correct or wrong)	Critique model

While inference-time computation holds significant potential for enhancing the reasoning performance of LLMs [31], existing studies reveal mixed results in inference-time computation (e.g., limited self-correction capabilities [15]). Its effectiveness on broader reasoning tasks (e.g., logical reasoning, code generation, question answering, and fact verification) remains limited, with most research narrowly focused on domains like math problems. Moreover, inference-time methods are sensitive to hyperparameters, such as temperature and top-p sampling, where small adjustments can lead to notable performance differences (e.g., a 5% improvement in solving math problems by tuning temperature). These challenges underscore the critical role of inference-time techniques (e.g., instructing prompt, sampling strategies, reward models), as shown in Table 1. Despite recent advancements, these gaps indicate that the field remains nascent, with many challenges yet to be addressed.

In this study, we investigate key tricks that influence the effectiveness of inference-time computation methods in LLM reasoning. Since most current methods rely on a proposer-verifier pipeline that first generates candidate solutions (e.g., chain-of-thought candidate solutions) and then selects the optimal solution based on specific reward signals (e.g., RLHF rewards, process rewards), our research focuses on strategies for both candidate solution generation (e.g., instructing prompts, hyperparameters such as temperature and top-p) and reward mechanisms (e.g., self-evaluation, reward types) across broader reasoning tasks, including logical reasoning, code generation, fact verification, complex mathematics, and arithmetic. Through ablation studies, we evaluate simple techniques in inference-time computation and empirically analyze their impact on reasoning performance. Our analysis reveals that inference-time computation methods are highly sensitive to the experimental setup. With proper configurations, our results demonstrate that previously overlooked techniques can significantly enhance performance. Furthermore, we emphasize the need for a standardized benchmarking framework for inference-time computation in LLM reasoning.

Our main contributions and observations include the following: 1) **Evaluation of Key Tricks**: We evaluate the impact of a broad range of key tricks (e.g., prompt type, temperature, top-p, reward model type) on LLMs. Key insights include: Instruction prompts significantly influence LLM reasoning, with self-correction often yielding mixed results compared to CoT prompting; self-evaluation frequently fails to assess solution quality effectively; reward models can cause performance inflation in LLM reasoning, with inconsistent effectiveness across tasks due to generalization issues. 2) **Combination of Techniques**: We further explore the effects of combining selected effective tricks on inference-time computation. Our empirical results suggest that improvements are not always additive when different techniques are combined, although prompt design, temperature adjustment, and reward models can be beneficial, as demonstrated in Section 4.2.3. 3) **Comprehensive Benchmarks**: We conduct extensive experiments, evaluating six representative inference-time computation methods across eight reasoning tasks, using over 20,000 A100-80G GPU hours and more than 1,000 individual tests. Our results establish a baseline benchmark for inference-time computation, as shown in Table 2.

2 Related Work

We briefly introduce related work, including reasoning with LLMs, inference-time computation methods for LLM reasoning, and benchmarks of LLM reasoning.

Reasoning with LLMs. LLMs have demonstrated strong reasoning abilities in complex tasks such as code generation, mathematical problem-solving, and research ideation [43]. Existing methods for enhancing LLM reasoning include: 1) Prompt engineering – Activates latent multi-step reasoning capabilities. For example, Chain of Thought (CoT) [32] guides step-by-step problem-solving but relies heavily on high-quality demonstrations for analogical learning. 2) Post-training techniques[3, 4] – Iteratively enrich training datasets to improve model performance. Self-training methods[3]

curate new high-quality examples to enhance reasoning, but these approaches demand significant computational resources. 3)Search-based methods[1, 9, 20] – Optimize reasoning paths at inference time using search algorithms. For instance, Tree of Thought[38] employs breadth-first search to refine solutions. This work focuses on test-time computation, leveraging inference-time optimization to enhance LLM reasoning without additional training overhead.

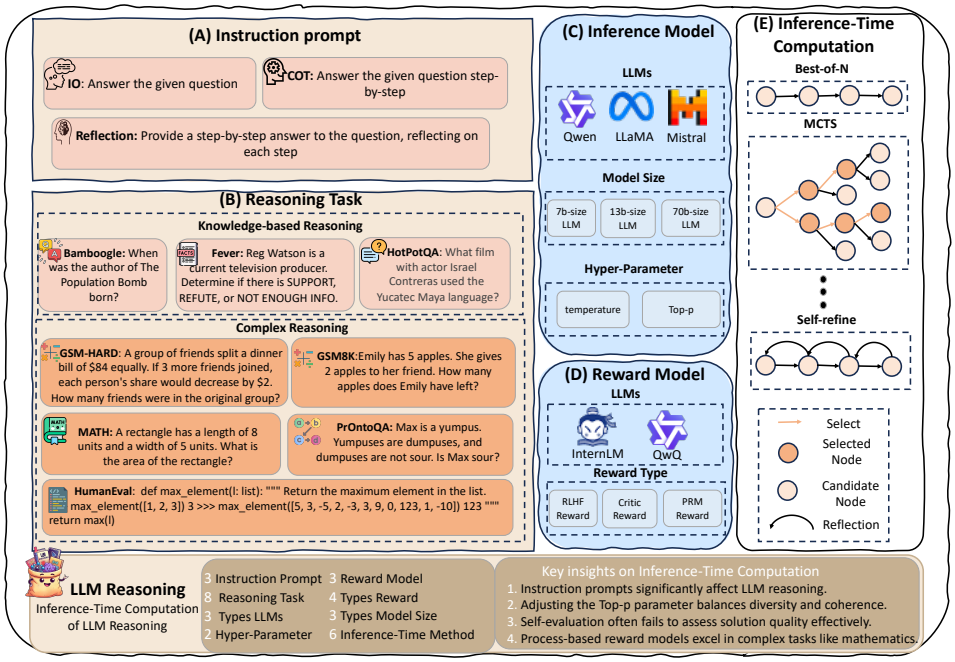

Figure 1: Overview of Decoding Inference-Time Computation for LLM Reasoning. **(A) Instruction Prompt**: Includes IO, Chain-of-Thought (CoT), and reflection-based CoT prompts. **(B) Reasoning Task**: Evaluates models on eight datasets: Arithmetic (GSM8K, GSM-Hard), Complex Math (MATH), Logical (PrOntoQA), Code Generation (HumanEval), Question Answering (Bamboogle), Fact Verification (FEVER), and Common Sense (HotpotQA). **(C) Inference Model**: Analyzes LLMs (LLaMA, Qwen, Mistral) of varying sizes and architectures, with performance assessed via temperature and top-p hyperparameters. **(D) Reward Model**: Explores reward types like RLHF, critic, and process-based models to enhance inference performance. **(E) Inference-Time Computation**: Investigates methods like Best-of-N Sampling, Step-Level Best-of-N, Self-Consistency, Monte Carlo Tree Search (MCTS), and Self-Refinement to optimize reasoning.

Inference-Time Computation of LLM Reasoning. Scaling inference-time computation has shown greater effectiveness than simply increasing model parameters [27]. Recent work emphasizes optimizing reasoning efficiency during inference rather than focusing solely on training-time computation. Best-of-N [6] improves reasoning by sampling N candidate solutions, scoring them with a learned verifier or reward model, and selecting the best. MCTS [29] further enhances inference by actively planning and choosing higher-quality responses. Together, these advances underscore inference-time optimization as critical for improving LLM reasoning beyond training-scale improvements.

Benchmarks of LLM Reasoning. LLMs have made remarkable progress in solving complex tasks in a zero-shot manner [12, 25, 18], positioning them as a key milestone toward artificial general intelligence. Consequently, benchmarking their reasoning abilities has become a central challenge. Recent studies have evaluated LLM reasoning across various domains, including mathematical reasoning [12], code generation [5], and factual QA [28], etc [21, 19]. While these benchmarks enhance our understanding of LLM reasoning, most research has focused on task performance rather than inference-time computation, leaving key optimization techniques underexplored.

Unique to this paper, we are the first to comprehensively study how LLM reasoning performance changes with the incorporation of previously overlooked key techniques. We hope that our work will provide valuable insights into the role of inference-time computation.

3 Preliminares

LLMs. Given an input context $\mathbf{x}$ (e.g., math problem, commonsense QA, etc.), the LLM aims to autoregressively predict the next token [8],

$$\pi_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{n} \pi_\theta(\mathbf{y}_t|\mathbf{x}, y_{<t}), \tag{1}$$

where $\pi_\theta(\cdot)$ is the LLM parameterized by θ, and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ is the output sequence. Here, $y_{<1} = \emptyset$ and $y_{<t} = (y_1, y_2, \cdots, y_{t-1})$. For a vocabulary size M, the probability of predicting the t-th token is determined using a softmax with temperature τ on logit scores z of all tokens, combined with top-p (nucleus sampling) to control the randomness and diversity of the sampling process.

Chain of Thought Prompting. Chain-of-thought (CoT) [32] is a method that prompts LLMs to generate a series of reasoning steps leading to the final answer. These intermediate steps, denoted as $y_1, \ldots, y_{n-1}$, connect the input $\mathbf{x}$ to the output y (omit n for simplicity), where n represents the total number of steps. For example, given an instruction $\mathbf{I}$ (e.g., "Let's solve this step by step") along with demonstration examples and the input question $\mathbf{x}$, the final answer is y. Each intermediate thought y_i is a part of the reasoning process that leads to the final answer. These thoughts are sequentially generated from the distribution $y_i \sim \pi_\theta(\cdot \mid \mathbf{I}, \mathbf{x}, \mathbf{y}_{<i-1})$, and the final output is sampled from: $y \sim \pi_\theta(\cdot \mid \mathbf{I}, \mathbf{x}, \mathbf{y}_{<n-1})$.

Temperature. The temperature [13] τ of LLM controls the level of randomness in the generated outputs, influencing their diversity. Instead of directly calculating the softmax, the logits are scaled by the temperature value. The conditional probability of generating a token in the sequence can be expressed as: $\pi_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t}) = \frac{\exp(z_t/\tau)}{\sum_{i=1}^{M} \exp(z_i/\tau)}$, where z_t represents the logit score: $\text{logit}_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t})$, and τ is the temperature parameter. A higher temperature τ results in a smoother probability distribution (introducing more randomness), while a lower temperature makes the distribution sharper, leading to more deterministic behavior.

Top-p. The top-p [14] controls the LLM output by augmenting the vocabulary size M as only those tokens are considered for which the cumulative probability ($C_k = \sum_{i=1}^{k} p_{(i)}$) is greater than the Top-p value (the cumulative probability is calculated by sorting the tokens by their probability in a descending order and then adding them up, where: probabilities: $\{p_{(1)}, p_{(2)}, \ldots, p_{(M)}\}$, where: $p_{(1)} \geq p_{(2)} \geq \cdots \geq p_{(M)}$). After the tokens are selected, it would be re-calculated their softmax with reduced vocab size. The truncated probability distribution can be defined as:

$$\pi_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t}) = \begin{cases} \frac{p_{(i)}}{\sum_{j=1}^{k} p_{(j)}}, & \text{if } i \leq k, \\ 0, & \text{if } i > k. \end{cases} \tag{2}$$

.

Inference-Time Computation Methods for LLM Reasoning. Inference-time computation methods [24] typically follows a pipeline comprising two main steps: generating candidate solutions (e.g., chain-of-thought reasoning candidates) and selecting the optimal solution based on specific reward signals (e.g., numerical reward, self-consistency, process reward, or binary feedback such as "Yes" or "No"). Formally, given a problem $\mathbf{x}$, the inference-time computation methods sample K candidate solutions: $y^{(k)} \sim \pi_\theta(y \mid \mathbf{I}, \mathbf{x}, \mathbf{y}_{<n})$,for $k = 1, 2, \ldots, K$, where $y^{(k)}$ represents the k-th candidate solution. After sampling, each candidate is evaluated using a reward model to produce a reward signal:$\mathbf{r}^{(k)} = \text{reward}(\mathbf{I}, \mathbf{x}, \mathbf{y}_{<n-1}, y^{(k)})$, where the reward model can take various forms. For example, it may be a general LLM that evaluates solutions using instructions $\mathbf{I}$ (e.g., "Let's verify the step-by-step reasoning. Is the answer correct (Yes/No)?"). Alternatively, the reward model could be specifically trained to output a scalar value between 0 and 1, with higher values indicating better solutions. The final solution $\hat{y}$ is then selected based on the reward signals. For numerical rewards, the solution with the highest reward is chosen:$\hat{y} = \arg\max_{y_k} r_k$.

4 Decoding Inference-Time Computation of LLM Reasoning

In this section, we decode the inference-time computation of LLM reasoning. First, we introduce the experimental setup, followed by the main bag of tricks for improving inference-time computation of

LLM reasoning. Finally, we benchmark various inference-time computation methods. The overall framework is illustrated in Figure 1.

4.1 Experiments Setup

Models. *Inference Model.* We evaluate several widely studied LLMs of varying sizes and configurations: (1) LLaMA 3.3[8]: Meta AI's latest LLaMA iteration, available in 8B and 70B parameters, known for its open-source accessibility and strong benchmark performance. (2) Qwen 2.5[36]: Developed by Alibaba Cloud, this model provides 7B and 72B variants, showcasing diverse architectures and training approaches. (3) Mistral 7B Instruct v0.3 [16]: A 7B model from Mistral AI, recognized for its efficiency and performance comparable to larger models. These models offer a broad view of reasoning capabilities across architectures and training paradigms. *Reward Model.* We employ four types of reward models: (1) Process Reward[42]: Evaluates each reasoning step incrementally. (2) Result Reward: Assesses only the final answer's correctness. (3) RLHF Reward[2]: Based on human or AI-generated preference samples. (4) Proof-Critical Reward: Designed for formal mathematical reasoning across multiple benchmarks. Further experimental settings are detailed in Appendix A.

Tasks. Our research focuses on the following reasoning tasks: 1) Arithmetic Reasoning: Evaluating models on GSM8K [7] and GSM-Hard [11] datasets to test their arithmetic calculation skills. 2) Complex mathematical reasoning: Using the MATH [12] to assess proficiency in solving advanced mathematical problems. 3) Logical Reasoning: measuring logical deduction and inference abilities with the ProntoQA [26] dataset. 4) Code Generation: Testing code generation skills on the HumanEval [5] dataset. 5) Question Answering: Evaluating performance in answering diverse questions using the Bamboogle [25]. 6) Fact Verification: Assessing factual verification using the FEVER [28] dataset. 7) Common Sense Reasoning: Testing understanding of common sense knowledge and reasoning with the HotpotQA [37] dataset.

Inference-Time Computation Methods. This study examines common inference-time computation methods: 1)Best-of-N [6]: Generates N candidate outputs and selects the best based on a reward model. 2) Step-Level Best-of-N Sampling [6]: Applies Best-of-N at each generation step to select the most promising continuation. 3) Self-Consistency [31]: Generates multiple reasoning paths and selects the most consistent answer. 4) Beam Search [24]: Expands all nodes at the current depth before proceeding, maintaining multiple hypotheses. 5) Monte Carlo Tree Search (MCTS) [10]: Builds a search tree via random sampling to identify high-quality outputs. 6) Self-Refine [22]: Enables LLMs to iteratively revise outputs during inference.

Evaluation Metrics. For most reasoning tasks, accuracy serves as the primary evaluation metric. For code generation, we use the pass@k metric [5], which considers a problem solved if at least one of k generated code samples passes all test cases. In our evaluation, we focus on pass@1 by generating multiple samples and selecting the best one using the reward model.

4.2 Bag of Tricks

Our goal is to investigate how previously overlooked tricks can critically affect the performance of inference-time computation methods, which typically consist of two main steps: generating candidate solutions (e.g., prompt type, temperature, top-p, etc.) and selecting the optimal solution based on specific reward signals (e.g., self-evaluation, reward type, reward process). In our default setup, we primarily adopt the Best-of-N inference-time computation with the number of candidates $N = 32$, the temperature $\tau = 0.7$, and top-p set to 0.9. Additionally, the instruction prompt type is set to Chain-of-Thought (CoT). Without further modifications, we conduct ablation studies, varying only the specific tricks under investigation. We focus primarily on complex reasoning tasks, including math problems, and code generation tasks, etc, while additional tasks are detailed in the Appendix B. All additional details regarding the experimental implementation settings can be found in the Appendix A.

Note that our empirical observations and conclusions may not generalize to all datasets and models. However, we emphasize the necessity of using consistent implementation details to ensure fair comparisons among different inference-time computation methods.

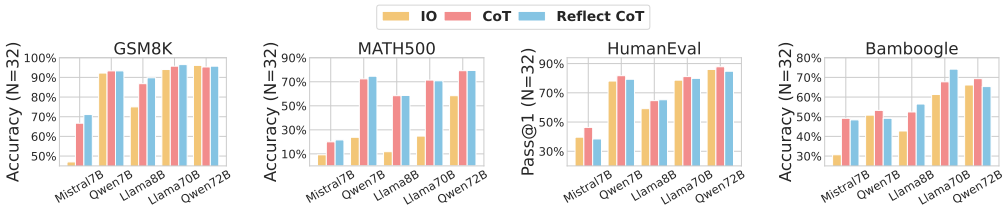

Figure 2: Accuracy (%) across benchmark tasks under different instruction prompts. The results underscore the substantial influence of prompt design on inference-time computation methods. Notably, self-correction mechanisms produced mixed outcomes.

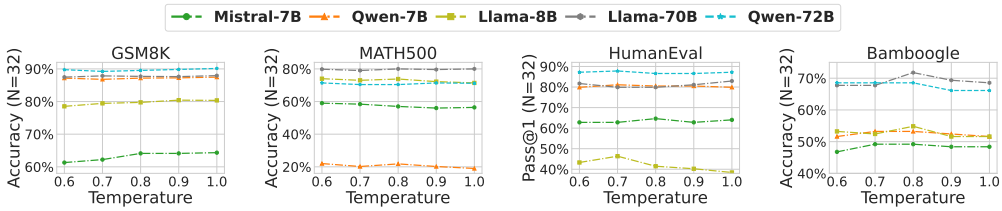

Figure 3: Accuracy (%) versus temperature during inference. Temperature varies from 0.6 to 1.0, highlighting its impact on LLM reasoning, with performance stabilizing around the recommended default of 0.8.

4.2.1 Generating Candidate Solutions

Generating candidate solutions is a critical step in inference-time computation for LLM reasoning, but the inherent randomness in this process significantly influences diversity. Hyperparameters such as temperature and top-p, along with strategies like instruction prompts, play a vital role in shaping and guiding the solution trajectory. For example, temperature, as a sampling strategy in token generation, increases diversity at higher values. Therefore, this study focuses on the candidate solution generation process, including instruction prompt types, temperature, and top-p sampling.

Instruction Prompt Type. Different instruction prompts can guide an LLM to generate distinct reasoning paths. Specifically, Input-Output (IO) prompts directly provide the answer, whereas Chain-of-Thought (CoT) prompts encourage the LLM to reason step by step. Recent research [15] suggests that self-correction or self-reflection mechanisms are often ineffective when LLMs operate without external feedback under certain prompt types. We further explores the impact of various prompt types, including IO prompts, standard Chain-of-Thought (CoT) prompts, and reflection-based CoT. As shown in Figure 2, the results illustrate that different prompts have a significant effect on LLM reasoning performance. Specifically, we observe that while CoT prompts generally improve reasoning accuracy, Reflection-based CoT prompts show more mixed results across the datasets. These findings are consistent with the observations in [15], where self-correction mechanisms failed to show a consistent improvement, with the outcomes varying across different tasks.

Temperature. Temperature [13] τ regulates the diversity of candidate solutions in LLMs. A higher τ decreases prediction confidence but increases output variability. We revisit the previous inference-time computation settings in Table 1, where the temperature is set differently for each case. Figure 3 illustrates its effect. In most reasoning scenarios, the LLM's performance is optimized at τ= 0.8, yielding an improvement of approximately 2.32% to 4.83% across four datasets. In most cases, both larger and smaller values of τ result in decreased performance.

Top-p. The top-p parameter regulates the output of an LLM by modifying the effective vocabulary size M , considering only those tokens whose cumulative probability exceeds a predefined threshold. In general, top-p strikes a balance between diversity and quality in the model's generated output. As p decreases, the model becomes increasingly constrained to a smaller set of high-probability tokens, leading to more focused and deterministic outputs. Conversely, higher values of p allow for a broader selection of tokens, which increases diversity but may also result in less coherent outputs. Figure 4 demonstrates that the impact of top-p on inference-time computation in LLM reasoning is significant, with an optimal value of top-p = 0.9 , resulting in an overall improvement of 2.32%–5.88%.

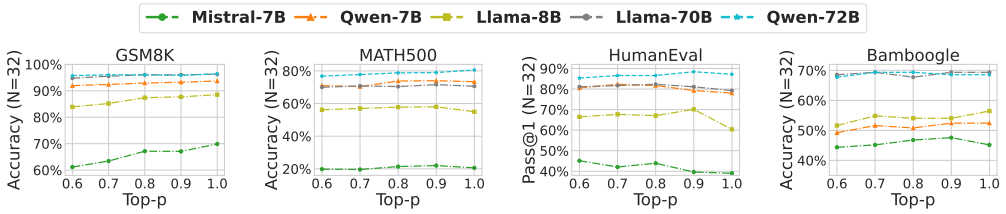

Figure 4: Accuracy (%) versus Top-p values during inference. Top-p varies from 0.6 to 1.0, highlighting its impact on LLM reasoning, with performance stabilizing around the recommended default of 0.9.

Figure 5: Accuracy (%) across four benchmark tasks with different evaluation strategies. The results shows that Self-evaluation often fails to assess solution quality effectively.

4.2.2 Selecting Optimal Solutions

Selecting optimal solutions is a critical step in the inference-time computation of LLM reasoning. This process typically involves either selection by the inference model itself (e.g., voting or prompt-based selection) or the use of external reward models (e.g., RLHF, Proof-Critical, or process reward models). A key question is whether LLMs can effectively evaluate their own solutions. However, self-evaluation methods often fall short, as LLMs struggle to correct errors without external guidance. Moreover, reward models frequently fail to distinguish truly correct answers from superficially correct ones, leading to inflated performance evaluations. This challenge underscores the need for more reliable evaluation mechanisms. To address these gaps, we study the selection process, focusing on self-evaluation, reward types, and investigating generalization of improved reward models.

Self-Evaluation. Previous studies [40, 38] have explored self-evaluation methods for selecting the optimal candidate solution in the inference phase, where the model assesses its own generated solutions using fuzzy judgments (e.g., categorizing solutions as "impossible," "likely," or "sure" to solve a problem). These fuzzy evaluations are then translated into probabilities, such as mapping "impossible" to 0.01 and "sure" to 1. We examine the effectiveness of self-evaluation approaches, including self-process evaluation and self-result evaluation, in comparison to random selection and majority voting. Figure 5 shows that self-evaluation does not consistently improve performance; in fact, in some worst-case scenarios, it performs worse than random selection.

Reward Type. Recently developed reward models (e.g., reward models, critic models, process reward models) have become key tools in enhancing the reasoning capabilities of LLMs during the inference-time phase [42, 41]. We investigate various reward types, including RLHF Reward [2], Proof-Critical Reward, LLM-as-Judge, and majority voting. Figure 6 illustrates the significant impact of different reward models on inference-time computation performance. Specifically, for knowledge-based reasoning, certain reward models can substantially improve performance. In contrast, for more complex reasoning tasks such as mathematics and code generation, the LLM-as-Judge process reward seems to provide a greater performance boost.

Performance Inflation with Reward Model. Scaling test-time computation with reward models presents complexities. While one might expect LLM reasoning performance to improve steadily as the reward model is optimized, this is not always the case. Figure 7 reports that the reward model does not perform consistently across all cases during scaling. For example, in the challenging MATH task, the Proof-Critical Reward model can lead to a decrease in performance rather than a progressive improvement. This inflation of LLM reasoning performance can be attributed to the generalization issues of the reward model, as reported in [42, 41]. Currently, the reward model does not perform well across all tasks.

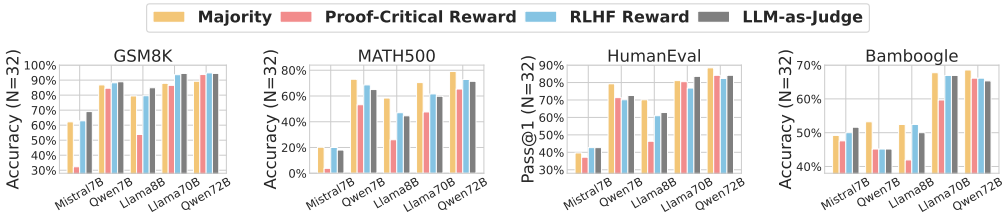

Figure 6: Comparison of different reward models across benchmarks showcasing their impact on accuracy and pass@1 for multiple LLMs.

Table 2: LLM Reasoning Performance under Inference-Time Computation with Fixed Token Budget. The table shows accuracy and token consumption across various reasoning tasks for Llama-3.1-8B and Qwen-2.5-7B models.

Method	Knowledge-based Reasoning Accuracy (%) / #Tokens			Complex Reasoning Pass@1 / #Tokens	Complex Reasoning Accuracy (%) / #Tokens			
	Bamboogle	HotPotQA	Fever	HumanEval	GSM8K	GSM-HARD	MATH	PrOntoQA
Llama-3.1-8B								
Best-of-N	48.4 / 1077	**48.6** / 876	61.1 / 1191	**60.4** / 1244	83.1 / 976	**33.6** / 1174	44.6 / 1412	**94.0** / 969
Step-Level Best-of-N	52.4 / 1119	45.7 / 632	60.3 / 581	55.5 / 921	83.0 / 845	32.9 / 1013	39.8 / 708	93.8 / 881
Beam Search	33.1 / 1801	31.8 / 2091	57.3 / 917	50.6 / 1019	78.7 / 1066	25.1 / 1328	43.2 / 1014	89.6 / 1241
MCTS	49.2 / 896	41.3 / 956	59.9 / 843	48.7 / 1114	81.3 / 582	31.6 / 627	31.0 / 1898	93.0 / 1511
Self-Consistency	**54.8** / 1077	48.3 / 876	**61.9** / 1191	60.3 / 1261	**84.3** / 976	33.2 / 1174	**45.0** / 1412	93.4 / 969
Self-Refine	41.9 / 287	40.5 / 280	19.1 / 265	41.4 / 495	64.5 / 410	25.1 / 564	37.8 / 743	62.2 / 525
Qwen-2.5-7B								
Best-of-N	49.2 / 881	**44.6** / 1189	55.6 / 830	78.0 / 1102	90.5 / 1462	50.1 / 1174	61.2 / 1827	**96.2** / 1271
Step-Level Best-of-N	**51.6** / 863	43.8 / 1055	50.2 / 995	73.1 / 691	89.7 / 709	**50.7** / 877	63.0 / 1162	92.8 / 625
Beam Search	29.0 / 1552	44.3 / 1280	**55.7** / 997	75.6 / 1019	88.3 / 1341	50.2 / 1417	49.2 / 836	90.2 / 1089
MCTS	47.6 / 1467	42.6 / 2585	53.9 / 1442	69.8 / 724	66.2 / 1416	48.7 / 1114	26.2/ 1805	70.6 / 2582
Self-Consistency	49.2 / 881	44.5 / 1189	56.7 / 830	**79.9** / 1271	**90.9** / 1462	50.0 / 1174	**70.2** / 1878	94.8 / 1271
Self-Refine	48.4 / 232	42.3 / 316	16.8 / 639	41.4 / 496	86.1 / 601	43.4 / 982	55.8 / 947	58.4 / 1105

4.2.3 Combination of Inference-Time Computation Tricks.

We further investigate the combination of selected useful techniques, including prompt type, temperature, top-p, and the reward model. As demonstrated in Table 4, the improvements are not always additive when combining different techniques, although methods such as prompt, temperature, and reward model can be effective. For example, as shown in Table 3, using Reflect COT with a temperature of 0.8 and top-p of 0.9 does not result in an improvement for the Bamboogle and MATH tasks. These results suggest that careful selection and combination of techniques can enhance performance, though the impact may vary across different models.

Table 3: Combination of Inference-Time Tricks on Llama3.1-8b. We highlight optimal trick combinations.

Prompt	Reward	Temp	Top-p	Bamboogle	MATH
IO	Majority	0.7	0.9	42.7	11.8
CoT	Majority	0.7	0.9	52.4	58.4
CoT	Majority	0.8	0.9	54.8	57.0
CoT	Majority	0.6	0.9	53.2	58.0
CoT	Majority	0.7	1.0	54.0	55.0
CoT	Majority	0.7	0.8	**56.4**	57.8
CoT	Random	0.7	0.9	44.4	40.6
CoT	Self-Process-Eval.	0.7	0.9	53.2	40.6
CoT	Self-Result-Eval.	0.7	0.9	45.1	39.2
Reflect CoT	Majority	0.7	0.9	**56.4**	**58.6**
Reflect CoT	Majority	0.8	0.9	55.6	57.0

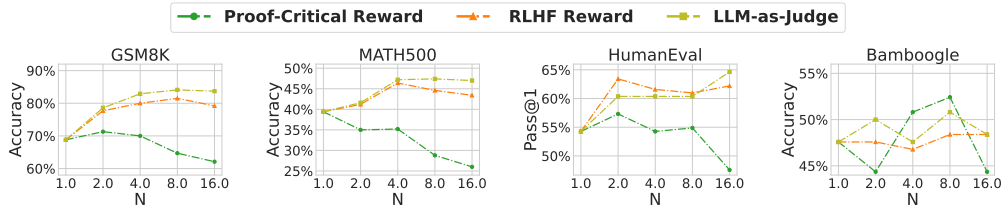

Figure 7: Scaling test-time performance with reward models (Proof-Critical, RLHF, LLM-as-Judge) across benchmarks.

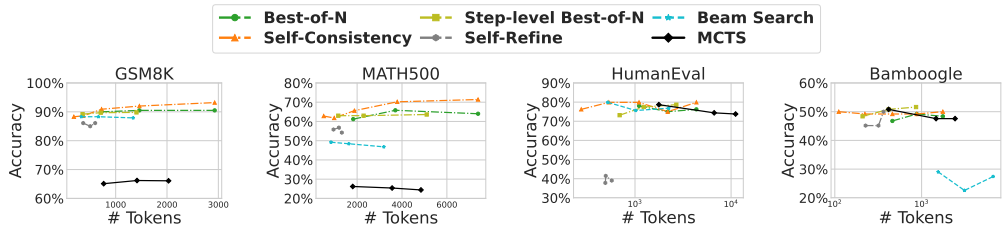

Figure 8: Performance versus token consumption across benchmarks for inference-time computation strategies.

Takeaways. *(1) Instruction prompts significantly influence LLM reasoning, with self-correction yielding mixed results compared to CoT prompting; (2) A recommended temperature of $\tau = 0.8$ balances diversity and confidence, optimizing reasoning performance, with Top-p performing best at 0.9; (3) LLMs are not effective at self-evaluation; (4) Reward models enhance inference-time computation, with process-based rewards excelling in complex tasks like mathematics and code generation; (5) Reward models can inflate performance evaluations due to generalization issues, leading to inconsistent task effectiveness.*

4.3 Benchmarking of Inference-time computation of LLMs reasoning under Computation Budgets

We further examine various inference-time computation methods, including Best-of-N, Step-level Best-of-N, Beam Search, MCTS, Self-Consistency, and Self-Refine, under a fixed budget.

Setup. We evaluate six inference-time computation methods under a fixed computation budget (equal-token results to standardize computational consumption across tasks) on Qwen-2.5-7B and LLaMA-3.3-8B. To ensure fairness, consistent settings are applied to all methods. For knowledge-based reasoning tasks (e.g., Bamboogle, HotpotQA, Fever), we use the RLHF reward model. For complex reasoning tasks (e.g., MATH, code generation), the QwQ-32B model provides the process reward. Implementation details are available in Appendix A.

Table 2 summarizes the results of six inference-time computation methods across eight reasoning tasks. Key observations include: (1) Performance varies significantly across tasks. Best-of-N and Self-Consistency perform well in knowledge-based reasoning tasks, but for complex tasks like MATH and GSM-HARD, Qwen-2.5-7B outperforms Llama-3.1-8B. (2) Adding more completion tokens does not always improve accuracy. While Beam Search may leverage higher token counts, the additional tokens often fail to yield proportional accuracy gains. (3) The Self-Refine method generally underperforms compared to Consistency, suggesting that overly restrictive processes may hinder optimal outputs for complex reasoning tasks.

Figure 8 illustrates how performance varies with increased computational consumption. As token usage rises, Self-Consistency and Self-Refine achieve higher accuracy than other methods, while Step-level Best-of-N, and MCTS show slower improvements. In contrast, Beam Search exhibits minimal gains, indicating lower token efficiency. Notably, the GSM8K task demonstrates a sharper accuracy increase with higher token consumption compared to MATH500, underscoring the task-specific nature of each method's performance. Additionally, we observe some performance inflation with increased computational consumption, attributed to the generalization limitations of the reward model, which does not perform well on this task.

5 Conclusion

In this paper, we investigate the role of inference-time computation in enhancing the reasoning capabilities of LLMs. Our extensive experimental evaluation reveals that seemingly simple yet overlooked tricks—such as sampling strategies and reward mechanisms—can yield substantial improvements in reasoning performance. The results demonstrate that careful tuning of inference parameters, such as temperature settings, top-k sampling, reward models, plays a crucial role in optimizing the performance of LLMs during inference time.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62572417, No.92370204), National Key R&D Program of China (Grant No.2023YFF0725004), the Guangzhou Basic and Applied Basic Research Program under Grant No. 2024A04J3279, Education Bureau of Guangzhou Municipality, and CCF-DiDi GAIA Collaborative Research Funds.

References

[1] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[2] Zheng Cai, Maosong Cao, Haojiong 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, Yining 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 Dahua Lin. Internlm2 technical report, 2024.

[3] Changyu Chen, Zichen Liu, Chao Du, Tianyu Pang, Qian Liu, Arunesh Sinha, Pradeep Varakantham, and Min Lin. Bootstrapping language models with dpo implicit rewards. *arXiv preprint arXiv:2406.09760*, 2024.

[4] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

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

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

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[9] Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

[10] Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

[11] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

[12] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[13] Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[14] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[15] 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. *arXiv preprint arXiv:2310.01798*, 2023.

[16] AQ Jiang, A Sablayrolles, A Mensch, C Bamford, DS Chaplot, D de las Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. Mistral 7b (2023). *arXiv preprint arXiv:2310.06825*, 2023.

[17] P. Langley. Crafting papers on machine learning. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

[18] Fan Liu, Yue Feng, Zhao Xu, Lixin Su, Xinyu Ma, Dawei Yin, and Hao Liu. Jailjudge: A comprehensive jailbreak judge benchmark with multi-agent enhanced explanation evaluation framework, 2024. URL `https://arxiv.org/abs/2410.12855`.

[19] Fan Liu, Zhao Xu, and Hao Liu. Adversarial tuning: Defending against jailbreak attacks for llms. *arXiv preprint arXiv:2406.06622*, 2024.

[20] Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. Making ppo even better: Value-guided monte-carlo tree search decoding. *arXiv preprint arXiv:2309.15028*, 2023.

[21] Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. Regmix: Data mixture as regression for language model pre-training. *arXiv preprint arXiv:2407.01492*, 2024.

[22] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Markian Rybchuk, Philip HS Torr, Ivan Laptev, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024.

[24] Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning*, pages 3956–3965. PMLR, 2018.

[25] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

[26] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=qFVVBzXxR2V.

[27] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[28] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *NAACL-HLT*, 2018.

[29] Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*, 2024.

[30] Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*, 2024.

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

[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[33] Jinyang Wu, Mingkuan Feng, Shuai Zhang, Feihu Che, Zengqi Wen, and Jianhua Tao. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478*, 2024.

[34] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems, 2024. URL https://arxiv.org/abs/2410.15700.

[35] Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, et al. Enhancing llm reasoning via critique models with test-time and training-time supervision. *arXiv preprint arXiv:2411.16579*, 2024.

[36] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[37] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[38] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[39] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[40] Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in llms. *arXiv preprint arXiv:2406.09136*, 2024.

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

[42] Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024.

[43] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: no

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: yes

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: In the experimental setup, we provide comprehensive details on the training and testing procedures.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: yes

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: yes

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [No]

    Justification: no

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: CC BY-NC

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: we have provided the dataset and open-sourced the code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: yes

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: yes

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We have developed a mathematical modeling agent based on a large language model.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Impact Statement

This study explores the optimization of inference-time computation strategies to enhance the reasoning abilities of LLMs. By addressing the limitations of existing methods, our findings contribute to improving LLM performance across diverse reasoning tasks, including logical reasoning, code generation, and fact verification. Our work establishes a comprehensive benchmark and demonstrates that previously overlooked techniques—such as temperature and top-p sampling adjustments—can significantly enhance reasoning accuracy. These advancements provide a critical foundation for future research and practical applications, particularly in resource-constrained settings. Optimizing LLMs at inference time without extensive retraining opens new possibilities for improving AI systems' efficiency and reliability in real-world use cases. As the field evolves, these insights will be instrumental in guiding the development of more capable and adaptable LLMs while ensuring robustness and scalability.

A Experiments Setup

In this section, we provide a more detailed explanation of the experimental setup. Our objective is to evaluate the impact of previously overlooked techniques on the performance of inference-time computation methods. These methods typically consist of two primary steps: (1) generating candidate solutions using specified parameters (e.g., prompt type, temperature, and top-p), and (2) selecting the optimal solution based on predefined reward signals (e.g., self-evaluation strategies, reward types, and processes).

Default Configuration. In our default setup, we employ the Best-of-N inference-time computation, where: The number of candidate solutions N is set to 32. The temperature (τ) is set to 0.7, which introduces moderate stochasticity during candidate generation. The nucleus sampling parameter, top-p, is configured at 0.9, ensuring diversity by sampling from the top 90% cumulative probability distribution of the predicted tokens. We utilize a Chain-of-Thought (CoT) instruction prompt type to facilitate step-by-step reasoning for more complex tasks. Unless explicitly modified, all experiments adhere to this baseline configuration.

Reward Model. Specifically, for process and result rewards, we employ a prompt-driven approach to guide QwQ-32B, a reasoning model, in evaluating candidate solutions. The prompts used are the process evaluation prompt and the result evaluation prompt, which refer to the evaluation of step-level solutions and the final results, respectively. For RLHF and proof-critical rewards, which are numerical reward models, scores are directly assigned to candidate solutions. RLHF Reward: Based on InternLM2-Chat-1.8B-SFT [2], trained on over 2.4 million preference samples. It balances performance, helpfulness, and alignment. Proof-Critical Reward Derived from the InternLM2-5-Step-Prover-Critic model [34], which excels in multiple benchmarks.

Process-Evaluation.

Evaluate whether the language model can decompose the question into relevant sub-questions and assess whether this decomposition aids in answering the original question, either partially or directly. The evaluation result will be classified as "Sure," "Likely," or "Impossible" based on the effectiveness of the decomposition.
Evaluation Process: 1. Relevance of Sub-Questions: Determine if the sub-questions are directly related to solving the original question. 2. Effectiveness of Decomposition: Assess whether the sub-questions, when answered, lead to a comprehensive response to the original question.
Evaluation Outcomes: Sure: The model successfully decomposes the question into relevant sub-questions, each structured to contribute to an accurate final answer.Likely: The model decomposes the question into relevant sub-questions, but minor improvements in structure or relevance may enhance the response. Impossible: The model fails to decompose the question effectively or the sub-questions are not relevant to the original question.

Result-Evaluation.

The Result-Evaluation prompt evaluates the final outcome of the language model's reasoning process based on the accuracy, clarity, and completeness of its answer. Each evaluation is categorized as "Sure," "Likely," or "Impossible." The result is judged by verifying whether the language model's final answer is both directly relevant and valid in response to the question posed. The process involves reviewing whether the model's conclusion is definitive and supported by logical reasoning. If the answer is clear and unambiguous, it is marked as "Sure." If there are minor ambiguities, it is categorized as "Likely." If the answer is incorrect or irrelevant, it is deemed "Impossible."

Tasks. Our research focuses on the following reasoning tasks: (1) **Arithmetic Reasoning: GSM8K:** A dataset with grade school math word problems requiring 2 to 8 calculation steps using basic arithmetic operations. **GSM-Hard:** A harder variant of GSM8K with larger, less common numbers, increasing the challenge of arithmetic reasoning. (2) **Complex Mathematical Reasoning: MATH Dataset:** A dataset covering advanced topics like algebra, geometry, and calculus. Each problem includes detailed solutions to evaluate both final answers and problem-solving processes. We use the MATH 500 to test the complex mathematical reasoning ability of LLM. (3) **Logical Reasoning: ProntoQA:** Measures logical deduction and inference, requiring the application of logical principles to reach correct conclusions. (4) **Code Generation: HumanEval:** A benchmark for generating functional code snippets, with programming problems that include prompts and test cases to verify correctness. (5) **Question Answering: Bamboogle:** Evaluates diverse question-answering performance across various topics, testing comprehension and accurate responses. (6) **Fact Verification: FEVER:** Assesses the ability to verify factual claims using a document corpus, promoting fact-checking system development. (7) **Common Sense Reasoning: HotpotQA:** Features multi-hop questions requiring reasoning across multiple facts, testing common sense knowledge and the ability to link disparate information.

Instruction Prompt Type. Different instruction prompts can guide an LLM to generate distinct reasoning paths. Specifically, Input-Output (IO) prompts directly provide the answer, whereas Chain-of-Thought (CoT) prompts encourage the LLM to reason step by step. Recent research [15] suggests that self-correction or self-reflection mechanisms are often ineffective when LLMs operate without external feedback under certain prompt types. We further explores the impact of various prompt types, including IO prompts, standard Chain-of-Thought (CoT) prompts, and reflection-based CoT.

Input-Output (IO) Prompt.

The Input-Output (IO) Prompt directly answers a given question without intermediate reasoning steps. It is designed to generate concise and accurate responses, ending with the phrase *"so the final answer is:"*. This prompt structure is particularly suitable for straightforward queries where minimal context or explanation is required.

Chain-of-Thought (CoT) Prompt.

The Chain-of-Thought (CoT) Prompt guides the model to answer questions step-by-step, breaking down the reasoning process into intermediate steps before concluding with the final answer. For instance, when comparing the lifespans of Theodor Haecker and Harry Vaughan Watkins, the CoT prompt explicitly calculates their ages step-by-step before determining the longer-lived individual. This structured approach enhances reasoning transparency and aligns the response with logical steps.

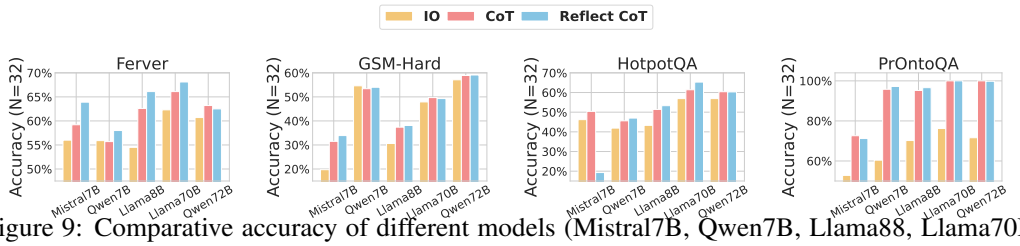

Figure 9: Comparative accuracy of different models (Mistral7B, Qwen7B, Llama88, Llama70B, and Qwen72B) across four datasets (Ferver, GSM-Hard, HotpotQA, and PrOntoQA) using three instruction prompts: Input-Output (IO), Chain-of-Thought (CoT), and Reflect Chain-of-Thought (Reflect CoT).

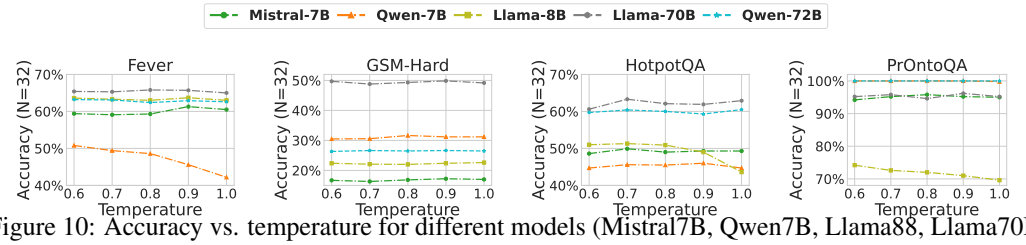

Figure 10: Accuracy vs. temperature for different models (Mistral7B, Qwen7B, Llama88, Llama70B, and Qwen72B) across four datasets (Ferver, GSM-Hard, HotpotQA, and PrOntoQA), with temperature settings ranging from 0.6 to 1.0.

Reflect CoT.

The Reflect Chain-of-Thought (Reflect CoT) prompt introduces a structured reasoning approach where each step in answering a question is followed by a reflection to verify its accuracy and reliability. For example, when comparing the lifespans of Theodor Haecker and Harry Vaughan Watkins, the process involves step-by-step reasoning to establish each individual's age at death. After each step, a "Reflection" line is used to ensure the validity of the information, such as verifying directly provided ages or confirming the consistency of the comparison. The final conclusion, supported by reflections, ensures a reliable and transparent reasoning process.

B Other Experiments

Instruction Prompt Type. The experimental results on additional datasets reveal a consistent finding: the type of instruction prompt significantly influences the inference-time computation of LLM reasoning, as illustrated in Figure 9.

Tempature. The experimental results on additional datasets reveal a consistent finding: the type of instruction prompt significantly influences the inference-time computation of LLM reasoning, as illustrated in Figure 10.

Top-p. We further present the ablation study on top-p applied to other reasoning tasks. The experimental results are shown in Figure 11. The impact of top-p is significant; generally, as top-p increases, the LLM's reasoning performance also improves, with optimal performance observed at 0.9 for most reasoning tasks.

Self-Evaluation. We further present the self-evaluation conducted on additional datasets. Figure 12 illustrates the experimental results, which reveal that the LLM still cannot effectively distinguish the correct solution.

Reward Type. Figure 13 further reports the impact of reward types on other datasets, confirming the same findings. Specifically, for knowledge-based reasoning, certain reward models can substantially improve performance. In contrast, for more complex reasoning tasks, such as PrOntoQA, the LLM-as-Judge process reward provides a more significant performance boost.

Performance Inflation with Reward Model. Scaling test-time computation with reward models presents complexities. Figure 14 reports that the reward model does not perform consistently across

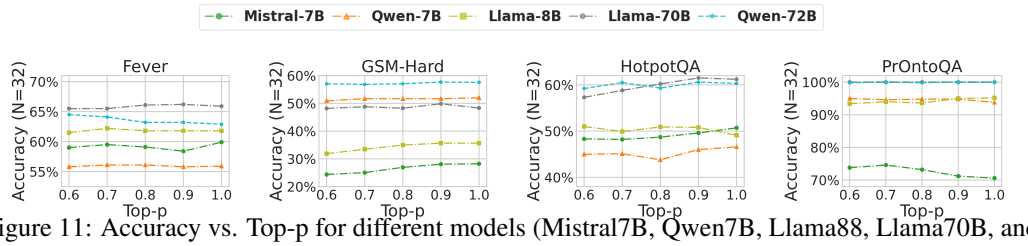

Figure 11: Accuracy vs. Top-p for different models (Mistral7B, Qwen7B, Llama88, Llama70B, and Qwen72B) across four datasets (Ferver, GSM-Hard, HotpotQA, and PrOntoQA), with Top-p values ranging from 0.6 to 1.0.

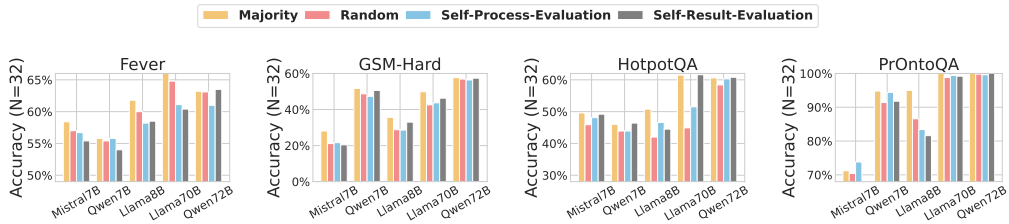

Figure 12: Accuracy (%) across four benchmark tasks with different evaluation strategies The results shows that Self-evaluation often fails to assess solution quality effectively.

all cases during scaling on other datasets. For example, in the challenging MATH task, the proof-critical reward model can lead to a decrease in performance rather than a progressive improvement. This inflation of LLM reasoning performance can be attributed to the generalization issues of the reward model, as reported in [42, 41]. Currently, the reward model does not perform well across all tasks.

Combination of Tricks. Table 4 further investigates the combination of selected useful techniques, including prompt type, temperature, top-p, and the reward model on Qwen2.5-7B. As demonstrated in Table 4, the improvements are not always additive when combining different techniques, although methods such as prompt type, temperature, and reward models can be individually effective. For example, as shown in Tables 4 using Reflect CoT with a temperature of 0.8 and a top-p of 0.9 does not lead to improvements for the Bamboogle and MATH tasks. These results suggest that the careful selection and combination of techniques can enhance performance, though the impact may vary across different models and tasks.

Table 4: Combination of Inference-Time Tricks on Qwen2.5-7B. We highlight optimal trick combinations.

Prompt	Reward	Temp	Top-p	Bamboogle	MATH
IO	Majority	0.7	0.9	42.7	23.8
CoT	Majority	0.7	0.9	**53.2**	72.4
CoT	Majority	0.8	0.9	53.2	73.8
CoT	Majority	0.6	0.9	51.6	74.0
CoT	Majority	0.7	1.0	52.4	73.4
CoT	Majority	0.7	0.8	50.8	73.8
CoT	Random	0.7	0.9	45.9	63.2
CoT	Self-Process-Evaluation	0.7	0.9	45.2	62.8
CoT	Self-Result-Evaluation	0.7	0.9	46.7	66.4
Reflect CoT	Majority	0.7	0.9	49.2	**74.6**
Reflect CoT	Majority	0.8	0.9	49.2	69.2
Reflect CoT	Majority	0.8	0.9	49.2	69.2

Main experiments of Inference-time computation methods. Figure 15 illustrates how performance varies with increased computational consumption. As token usage rises, Self-Consistency and Self-Refine achieve higher accuracy than other methods, while Step-level Best-of-N, and MCTS show slower improvements. In contrast, Beam Search exhibits minimal gains, indicating lower token efficiency. Notably, the GSM8K task demonstrates a sharper accuracy increase with higher token consumption compared to MATH500, underscoring the task-specific nature of each method's performance. Additionally, we observe some performance inflation with increased computational consumption, attributed to the generalization limitations of the reward model, which does not perform well on this task.

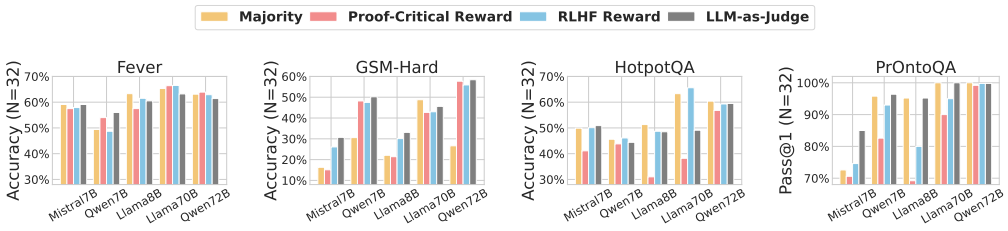

Figure 13: Comparison of different reward models across benchmarks showcasing their impact on accuracy.

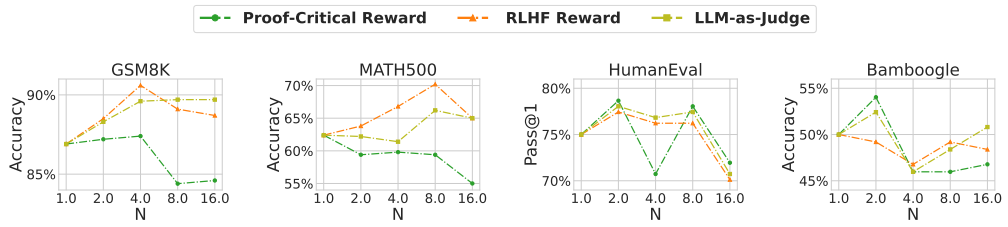

Figure 14: Scaling test-time performance with reward models (Proof-Critical, RLHF, LLM-as-Judge) across benchmarks, highlighting inconsistent trends and generalization issues, with notable declines in challenging tasks like MATH500 On Qwen-2.5-7B.

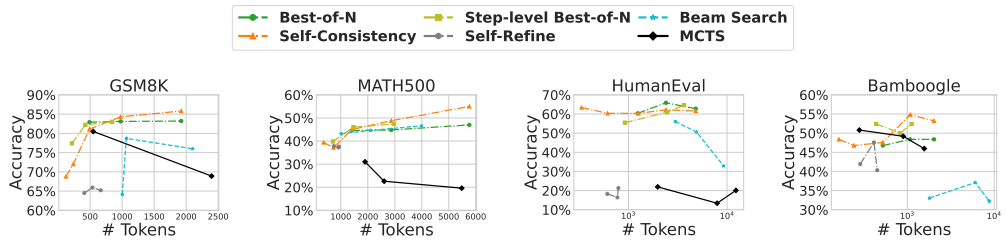

Figure 15: Performance versus token consumption across benchmarks for decoding strategies (Self-Consistency, Self-Refine, Best-of-N, Greedy, Beam Search, MCTS), highlighting task-specific trends and token efficiency disparities, with GSM8K showing sharp accuracy gains and MCTS lagging in improvement