

# Dynamic Self-Consistency: Leveraging Reasoning Paths for Efficient LLM Sampling

Guangya Wan<sup>1\*</sup>, Yuqi Wu<sup>2\*</sup>, Jie Chen<sup>2</sup>, Sheng Li<sup>1</sup>

<sup>1</sup>School of Data Science, University of Virginia

<sup>2</sup>Department of Electrical and Computer Engineering, University of Alberta  
{wxr9et,shengli}@virginia.edu, yuqi14@ualberta.ca, jc65@ualberta.ca

## Abstract

Self-Consistency (SC) is a widely used method to mitigate hallucinations in Large Language Models (LLMs) by sampling the LLM multiple times and outputting the most frequent solution. Despite its benefits, SC results in significant computational costs proportional to the number of samples generated. Previous early-stopping approaches, such as Early Stopping Self Consistency and Adaptive Consistency, have aimed to reduce these costs by considering output consistency, but they do not analyze the quality of the reasoning paths (RPs) themselves. To address this issue, we propose Reasoning-Aware Self-Consistency (RASC), an innovative early-stopping framework that dynamically adjusts the number of sample generations by considering both the output answer and the RPs from Chain of Thought (CoT) prompting. RASC assigns confidence scores sequentially to the generated samples, stops when certain criteria are met, and then employs weighted majority voting to optimize sample usage and enhance answer reliability. We comprehensively test RASC with multiple LLMs across varied QA datasets. RASC outperformed existing methods and significantly reduces sample usage by an average of 80% while maintaining or improving accuracy up to 5% compared to the original SC<sup>1</sup>.

and faithfulness of LLMs’ Reasoning Paths (RPs) remain questionable due to the phenomenon known as hallucination (Huang et al., 2023; Zhang et al., 2023). Consequently, many researchers have proposed different prompting methods to mitigate hallucination and improve overall performance. A widely used method is Self-Consistency (SC), a majority voting technique where multiple output samples (e.g., 64 samples) are generated for a given input, and the final decision is the mode among the samples (Wang et al., 2022b).

Despite its effectiveness, the SC strategy has significant computational overhead proportional to the number of sampled outputs, assuming these outputs are of equal length. For example, testing the entire MATH dataset with SC costs about \$2,000 using the GPT-4 API with a sampling size of 64, posing a substantial burden for many researchers and organizations (Lewkowycz et al., 2022). Thus, it is essential to minimize the cost of SC while maintaining performance. Various early-stopping mechanisms have been proposed to address this issue, showing significant reductions in sampling cost while preserving the original SC accuracy improvement (Li et al., 2024b; Aggarwal et al., 2023).

In the original Self-Consistency (SC) and previously proposed early-stopping variations, only the final answers derived from the sampled RPs were considered for improving QA accuracy or determining stop conditions due to the intuition that different ways of thinking lead to a unique correct answer in complex reasoning tasks (Wang et al., 2022b). However, these approaches neglect the quality and content of the RPs themselves, leading to the loss of critical information regarding potential hallucinations in the RPs. Additionally, studies by Wang et al. (2022a) and Jin et al. (2024) suggest that CoT reasoning is possible even with invalid demonstrations—prompting with invalid reasoning steps can achieve over 80-90% of the performance obtained using CoT under various metrics. This

## 1 Introduction

Large Language Models (LLMs) have demonstrated impressive performance across various tasks, such as question-answering (QA) (Tan et al., 2023; Arefeen et al., 2024; Li et al., 2024c), common-sense reasoning (Kojima et al., 2022a; Krause and Stolzenburg, 2023; Zhao et al., 2024), and math reasoning (Ahn et al., 2024; Imani et al., 2023; Zhang et al., 2024). However, the reliability

\*These authors contributed equally to this work.

<sup>1</sup>Code available at: [https://anonymous.4open.science/r/SC\\_conf-2D4B/README.md](https://anonymous.4open.science/r/SC_conf-2D4B/README.md)

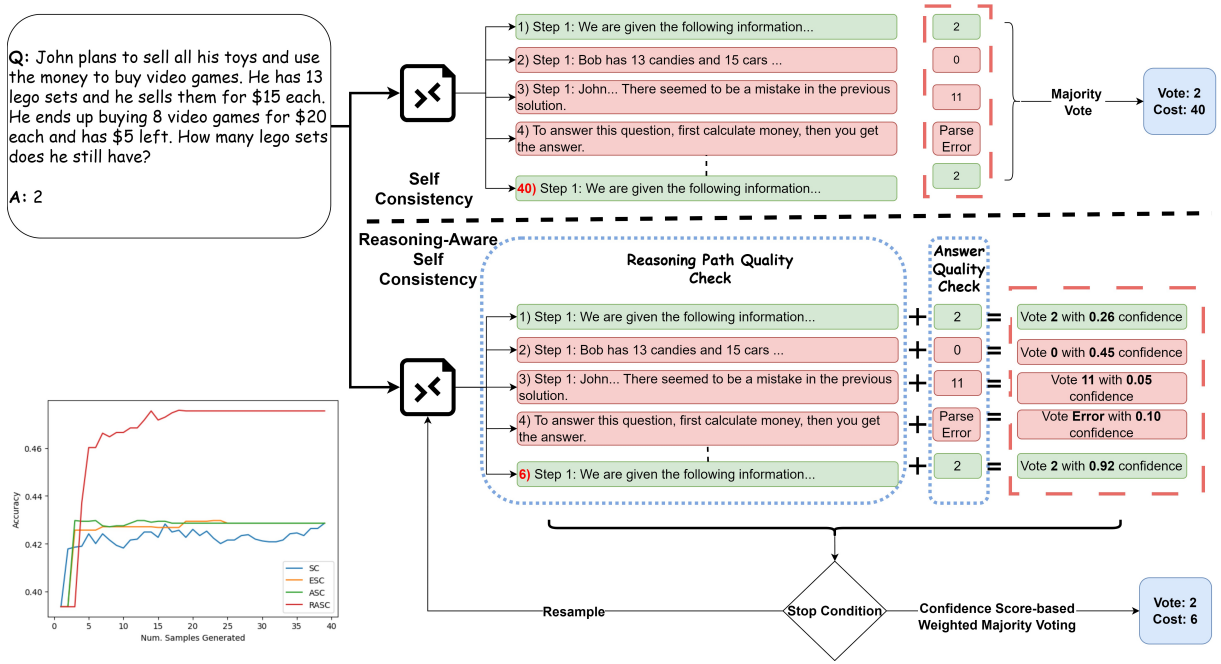


Figure 1: Comparison between original self-consistency method and our reasoning-aware self-consistency method.

implies that even if the output answer of a RP is correct, the RP’s contents may still be neither faithful nor logical. Moreover, previous SC approaches use a simple majority voting mechanism to determine the final answer, assigning equal weight to each sampled RP, which allows unfaithful outputs to influence the final decision. Therefore, a larger number of samples is required to achieve a more robust consensus.

To address these limitations, we propose Reasoning-Aware Self-Consistency (RASC) as shown in Fig. 1, which assigns an individual confidence score to each sampled RP based on its RP quality and output consistency, and then performs early-stopping based on the distribution of confidence scores. By utilizing weighted majority voting with the confidence score, we can optimally determine the minimum number of samples needed to generate a more reliable answer. RASC introduces a novel early-stopping mechanism for large language models that dynamically adjusts the number of sample evaluations, aiming to reduce redundant computational expense by considering both the quality of sample content and the consistency of RPs. We incorporate RASC to three different LLMs and evaluate the performance on five diverse datasets, including elementary math, commonsense, and challenging math problems. Our approach consistently outperforms fixed-budget and state-of-the-art early-stopping strategies, re-

ducing sample usage by an average of 80% without compromising accuracy. Additionally, increasing the budget allows our confidence score-driven weighted majority voting to improve QA accuracy by an average of 6%. We validated RASC on 14 out-of-domain QA datasets, demonstrating consistent significant improvements. In summary, our contributions are:

- **Reasoning-Aware Self-Consistency (RASC) Framework:** We propose an innovative early-stopping framework that dynamically modulates sample evaluations by considering both RP quality and output consistency.
- **Development of a Confidence Score Approximator:** We design seven lightweight textual indicators to evaluate content quality and faithfulness, combining with a weighted majority voting system that enhances sampling efficiency and accuracy.
- **Robust Evaluation:** RASC’s effectiveness and robustness are demonstrated across different LLMs and datasets, achieving significant improvements in both efficiency and accuracy.

## 2 Related Work

**Chain-of-Thoughts (CoT) Prompting:** Introduced by Wei et al. (2022), CoT prompting enhances the performance of Large Language Models

(LLMs) by guiding them through step-by-step reasoning, simplifying complex problems into manageable steps. This technique allows models to generate intermediate Reasoning Paths (RPs) and achieve more accurate conclusions. Several studies have built upon the original CoT concept (Zhou et al., 2023; Yao et al., 2023; Besta et al., 2024; Brown et al., 2020). In this work, we specifically focus on zero-shot CoT prompting (Kojima et al., 2022b) to construct RPs within our framework. This approach is notable for its straightforward prompt design, which requires no specific preparatory work or examples.

**Self-Consistency:** Wang et al. (2022b) introduced Self-Consistency (SC), which enhances the performance of Chain-of-Thought (CoT) prompting by sampling multiple diverse reasoning chains and aggregating their outcomes through a simple majority voting mechanism. While this approach achieves higher accuracy, it also incurs increased computational costs. Furthermore, the original SC settings employ a fixed number of samples for each question, and such a fixed sampling strategy becomes redundant and cost-inefficient, particularly for simpler questions. Addressing this inefficiency and redundancy is one of the main goals of our work.

**Self-Consistency Early Stopping:** To address the computational demands of traditional Self-Consistency (SC), innovative early stopping strategies have been developed. Li et al. (2024b) introduced the Early Stopping Self-Consistency (ESC), which halts RP sampling once the outputs within a fixed window are consistent. Similarly, Aggarwal et al. (2023) proposed the Adaptive Consistency (ASC), stopping sampling once a predefined probability threshold is reached, based on the distribution of sampled outputs. While both methods significantly reduce sampling costs by fourfold with minimal performance loss, they primarily focus on output consistency and neglect the quality of the RPs—a central aspect of our research.

### 3 Reasoning-Aware Self-Consistency

Reasoning-Aware Self-Consistency (RASC) is an early-stopping framework for Self-Consistency prompting that reduces redundant sampling costs while maintaining accuracy improvements. Unlike previous methods that consider only the final answer sequences, RASC determines stopping conditions by evaluating both the Reasoning Paths (RPs) and the output answers. In Fig. 1, we illustrate

the differences between traditional SC and RASC. Traditional SC improves accuracy by sampling a fixed number of RPs and determining the final answer based on the mode of these output answers. Our method, however, evaluates both the quality of the CoT RPs and the output answers. Each sample is assigned an individual confidence score based on these evaluations. The cumulative confidence score determines the stopping point. Finally, a confidence score-driven majority voting is used to decide the final answer.

#### 3.1 Reasoning-Aware Quality Features

To determine the content quality and answer consistency of sampled RPs, we establish a set of 7 quality features to calculate their confidence scores. The individual confidence score  $CS_i$  can be viewed as a soft measurement of hallucination, where higher confidence suggests less uncertainty in LLM’s generation. Since we aim for lightweight indicators requiring minimal feature extraction, we only consider textual features that do not require computationally intensive models or external tools. We classify these features into individual quality features and relative quality features (See Fig.2).

**Individual Quality Features:** We include 4 individual quality features, including content length  $L_i$ , input coherence  $C_i$ , format error  $F_i$ , and error-admitting  $E_i$ , to extract intrinsic information regarding each sampled RP  $x_i$ . We begin with factors identified by Jin et al. (2024) and Li et al. (2024a), who discuss the importance of RP textual length in terms of QA accuracy and human preference. Generally, lengthening the reasoning steps considerably enhances LLM’s reasoning abilities, and humans typically prefer longer reasoning chains. A common type of hallucination, instructional violation, occurs when LLMs do not follow user-defined instructions, implying a higher chance of hallucination and lower overall RP quality (Zhang et al., 2023; Huang et al., 2023; Hosking et al., 2023). The most obvious and easy-to-detect instructional violation is the RP format  $F_i$ . In QA tasks, users expect LLMs to respond in a clearly structured format to parse the necessary information. Additionally, consistency between the RP solution and the user question is crucial for instruction-following, implying coherence  $C_i$  between the input question and the output solution (Li et al., 2024a). Hence, we include instructional violations, such as format errors (parsing errors) and input coherence (deviation from input question), as quality features. With

advancements in LLM research, most LLMs can now detect errors in RP during generation. For instance, GPT-4 might acknowledge incorrect reasoning by stating, "There seemed to be a mistake in the previous calculations." Such admissions  $E_i$  typically imply that the RP quality is doubtful and the answer unreliable (Li et al., 2024a). Therefore, we identify common error-admitting terms to evaluate the quality and faithfulness of sampled RPs. We include detailed methods on individual feature extraction strategy in Appendix D.

**Relative Quality Features:** To reduce SC sample cost, the relative quality between additional samples and previous samples is essential to determine the stop condition. We design 3 features to capture relative relationships. Originating from Bang’s (2023) study, semantically similar RPs are more likely to generate similar answers. Inspired by Tu et al. (2020), we adopt semantic similarity  $S_i$  as a relative quality indicator. For each sampled RP  $RP_i$ , we compare the semantic similarity between this additional RP and all previously sampled RPs  $RP_1, \dots, RP_{i-1}$ . For answers, similarity refers to consistency between additional answer  $y_i$  and previous answers  $y_1, \dots, y_{i-1}$ . In this study, we adopt three similarity computation mechanisms: aggregation similarity  $S_i^a$ , pairwise similarity  $S_i^p$ , and bi-gram similarity  $S_i^b$ , as illustrated in Fig. 2. The selection of mechanisms depends on the nature of the target. For CoT RPs, which generally contain more vocabulary, aggregation similarity is more suitable due to its robustness. For answers with limited vocabulary (e.g., multiple-choice questions with four options), aggregation similarity becomes trivial. Pairwise comparison fits better for answers due to its high sensitivity. Detailed mathematical robustness and sensitivity proofs of aggregation and pairwise similarity are included in Appendix E.2 Theorem E.2, in which we prove that the aggregation similarity mechanism is more robust to noise while pairwise is more sensitive for large vocabulary documents. Bigram similarity  $S_i^b$  is defined as the similarity between the additional RP/answer and the preceding RP/answer. This mechanism ensures we capture local relative information since both aggregation and pairwise mechanisms capture global relationships. The validity of these newly proposed relative quality features is confirmed both theoretically through mathematical proof and empirical results. We also compared common similarity algorithms such as Euclidean distance and Cosine Similarity combined with various tokenizers,

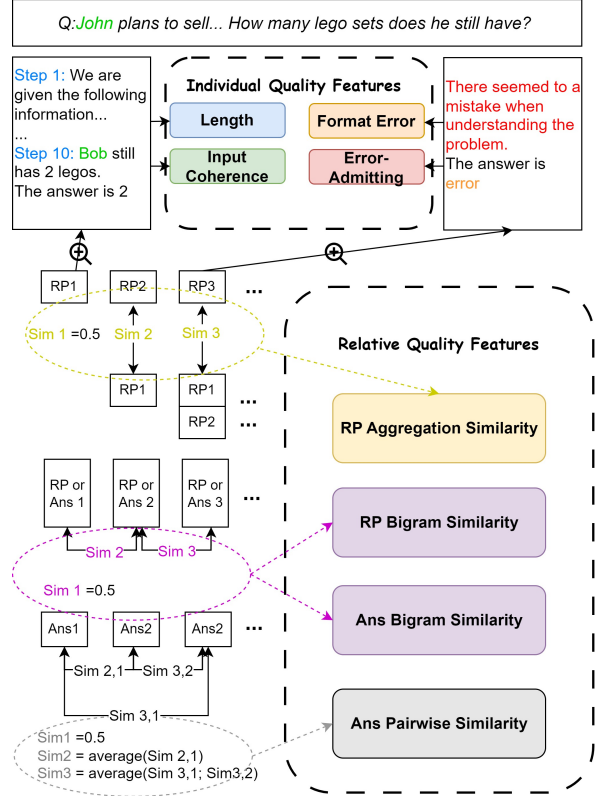


Figure 2: Reasoning-Aware quality features. Different colour corresponds to different feature visualizations. Ans: Output Answer; RP: Reasoning Path.

Jaccard similarity coefficient  $J$ , and Levenshtein distance (see Appendix E.1). Our empirical results indicate they have similar performance in semantic similarity (see Fig. 20 in Appendix E.1). Hence, we adopt Jaccard to compute similarity due to its lightweight and low computational overhead.

### 3.2 Confidence Score and Stop Condition

**Confidence Score Computation:** Consider a dataset  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^M$ , where each  $x_i \in \mathcal{R}^d$  is a feature vector representing the quality features of the  $i$ -th RP, and  $y_i \in \{0, 1\}$  is a binary label indicating the correctness of the answer. We seek to learn a confidence scoring function  $f : \mathcal{R}^d \rightarrow [0, 1]$  from a family of parameterized functions  $\mathcal{F}$ , where each function is denoted as  $f(\theta)$  for parameters  $\theta$ . The optimal parameters  $\theta^*$  are obtained by minimizing the following objective:

$$\theta = \arg \min_{\theta} \left( \frac{1}{M} \sum_{i=1}^M \mathcal{L}(f_{\theta}(x_i), y_i) \right). \quad (1)$$



Here,  $\mathcal{L}$  represents the loss function (e.g., cross-entropy) and  $\lambda$  is a hyperparameter that controls the regularization strength. We explore several lightweight classic classification models such as logistic regression (LR), Naive Bayes (NB), random forest (RF), and pre-trained deep learning models to represent  $\mathcal{F}$ . Specifically, the **input** to the confidence scoring function  $f(\theta)$  is a feature vector  $x_i = [L_i, C_i, F_i, E_i, S_i^a, S_i^p, S_i^b]$  representing the individual quality features (content length  $L_i$ , input coherence  $C_i$ , format error  $F_i$ , and error-admitting  $E_i$ ) and relative quality features (aggregation similarity  $S_i^a$ , pairwise similarity  $S_i^p$ , and bi-gram similarity  $S_i^b$ ) of the  $i$ -th sampled RP. The **output** of the confidence scoring function  $f(\theta)$  is a confidence score  $CS_i \in [0, 1]$  assigned to the corresponding RP. For the **Stop Condition**, let  $\{RP_i\}_{i=1}^K$  be a sequence of  $K$  sampled RPs, with corresponding confidence scores  $\{CS_i\}_{i=1}^K$  computed using the optimized confidence scoring function  $f_{\theta^*}$ . We define a buffer  $\mathcal{B}$  to store RPs with confidence scores above a predefined threshold  $T$ :

$$\mathcal{B} = RP_i \mid CS_i \geq T. \quad (2)$$

The sampling process terminates when the buffer’s size meets a predefined capacity  $N$  ( $|\mathcal{B}| \geq N$ ).

Upon reaching the stop condition, the final answer is determined through weighted majority voting. The weights applied are the confidence scores of the RPs in  $\mathcal{B}$ . Mathematically speaking, this is:

$$\text{Answer} = \arg \max_{a \in \mathcal{A}} \left( \sum_{\substack{RP_i \in \mathcal{B} \\ \text{Answer}(RP_i) = a}} CS_i \right), \quad (3)$$

where  $\mathcal{A}$  represents the set of all possible answers. This summation calculates the total confidence score for each possible answer  $a$ , accumulating scores only from those RPs in the buffer  $\mathcal{B}$  whose answers match the answer candidate  $a$ .

In this framework, the confidence scoring function  $f_{\theta^*}$  assigns a score to each sampled RP based on its quality. Sampling proceeds until the buffer  $\mathcal{B}$ , filled with RPs exceeding the threshold  $T$ , reaches the capacity  $N$ . The final answer is determined through weighted majority voting based on the confidence scores in  $\mathcal{B}$ . For details, see Algorithm 1.

## 4 Experiments

**Models:** In our experiments, we focus on three state-of-the-art language models: LLAMA3-8B

---

### Algorithm 1 RASC Early Stopping

---

```

1: Input: Query  $Q$ , threshold  $T$ , max size  $N$ 
2: Output: Best Answer  $A$ 
3:  $buffer \leftarrow \{Ans : [], CS : []\}$ 
4:  $stop \leftarrow \text{False}$ 
5: while not  $stop$  do
6:    $A_i, RP_i \leftarrow \text{LLM}(Q)$ 
7:    $CS_i \leftarrow \text{compute\_CS}(A_i, RP_i)$ 
8:   if  $CS_i \geq T$  then
9:     Append  $A_i$  to  $buffer[Ans]$ 
10:    Append  $CS_i$  to  $buffer[CS]$ 
11:    if  $\text{size}(buffer[Ans]) \geq N$  then
12:       $stop \leftarrow \text{True}$ 
13:    end if
14:  end if
15: end while
16:  $A \leftarrow \text{Weighted\_Majority\_Vote}(buffer)$ 
17: return  $A$ 

```

---

(Meta, 2024), GPT3.5-turbo (OpenAI, 2024), and Claude-3-Haiku (Anthropic, 2024). Details about the number of parameters, computational budget, and computing infrastructure used in our experiments can be found in Appendix A.

**Baseline Methods:** We compare RASC against three established baseline methods: SC (Wang et al., 2022b), ASC (Aggarwal et al., 2023), and ESC (Li et al., 2024b). Please refer to Section 2 for more details on these methods

**Datasets:** Our evaluation leverages five Question Answering datasets. For Math Reasoning, we collected data from GSM-8K (Cobbe et al., 2021) and MathQA (Amini et al., 2019), focusing on specific subsets within these datasets, including GSM-8K-hard, GSM-8K-test, MathQA-challenge, and MathQA-dev, to test our models against both nuanced and direct queries. We also collected data from BigBench (bench authors, 2023) to assess the versatility of our methods across different reasoning domains. We stratified the data based on categories and models to build confidence scoring models using the training segments, with subsequent evaluations performed on the test data. All presented results pertain to the test data unless otherwise specified. To test generalizability, we incorporated data from previous works involving ESC and ASC, evaluating our pipeline using unseen domains and diverse prompts (See Appendix G for more details).

**Evaluation Metrics:** We focus on the trade-off between accuracy and the number of generated sam-

ples required to achieve a target reasoning accuracy that exceeds all baseline methods while minimizing the number of samples generated. To quantitatively assess this trade-off, we adopted a customized metric that balances the contributions of accuracy and cost in our evaluations. This metric considers both the normalized accuracy and the normalized cost of generating predictions compared to the SC baseline. The details of this metric are provided in the Appendix F. Note that this metric is primarily used for evaluating our pipeline and selecting the best hyperparameters. For the main results, we will focus on presenting the accuracy and the number of generated samples.

**Implementation Details:** After extensive experiments (See Figure 3 and Figure 4), we determined that using three individual models ( $N = 3$ ) and a confidence threshold of 0.1 optimizes our customized metric. LR was selected as the confidence scoring model due to its superior performance on the test set. For generating predictions from language models, we employed a zero-shot CoT prompting approach. Details about the specific prompts used can be found in Appendix C. We adhered to the standard practices established in the SC methodology, setting the temperature parameter to 0.7 for all models, and the baseline method of SC utilizes a default of 40 samples. Due to budget constraints, we collected 500 samples per category per model, resulting in a total sample size of 9,000 in the final data. We also adopted a parser from Langchain and obtained the correctness label; please refer to Appendix D for details.

## 4.1 Main Results

**Quantitative Analysis:** The step reduction of different methods across various benchmarks is summarized in Table 1. RASC consistently outperforms other methods, reducing the number of samples required by up to 87.5% while simultaneously preserving comparable reasoning accuracy (refer to Figure 8 in Appendix B.2. for more information) across all datasets. This success can be attributed to RASC’s unique approach of leveraging information from both the generated answers and the reasoning paths (RPs) from the CoT prompting to determine the extent of hallucination in the generated contents. Such dynamic stopping criteria allow RASC to achieve superior performance in terms of both sample efficiency and reasoning accuracy across a wide range of datasets and models. The robustness and consistency of RASC’s perfor-

mance across different random seeds are further demonstrated in Tables 4 and 5 in Appendix B.1. Note that the tradeoff between sample efficiency and accuracy may vary across different categories and datasets. This is because the hyperparameters, such as the number of individual models ( $N$ ) and the confidence threshold ( $T$ ), are tuned based on the overall performance instead for individual models or categories. In our experiments, we adopted a consistent setting of  $N=3$  and  $T=0.1$  across all categories and datasets to maintain comparability.

**Cost Analysis:** Table 2 compares the accuracy and API cost of different methods when applied to GPT-3.5-TURBO, using SC as the baseline. This comparison highlights RASC’s significant reduction in API costs by 84.6%, alongside a 7.9% improvement in accuracy. These results demonstrate the main contribution of our pipeline: reducing cost as a result of reducing the number of samples generated while preserving the accuracy of the original SC method. Refer to the table 6 in the appendix B.2 for details on analysis on other models.

## 4.2 Analysis

### Controlling Accuracy-Samples Generation Tradeoff:

Our ablation study, illustrated in Figures 3 and 4, explores the impacts of hyperparameters ( $N$ ) and the confidence threshold ( $T$ ) on the performance of the RASC method. These results provide valuable insights for fine-tuning RASC to achieve optimal performance by balancing accuracy and computational cost.

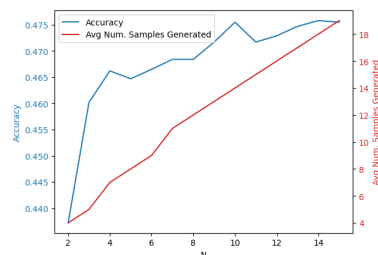


Figure 3: Effects of varying  $N$  on performance tradeoffs, illustrating how changes in  $N$  impact both accuracy and the average number of samples generated.

Figure 3 shows the effect of varying the number of individual models ( $N$ ) on the accuracy and the average number of samples generated by RASC. As  $N$  increases, the accuracy improves, but at the cost of generating more samples. This is because a larger  $N$  allows for more diverse predictions and a higher chance of reaching consensus for improved

Table 1: Num. Generated Sample of Different Methods and Models on Various Benchmarks.

Model	Method	Benchmark Category				
		MathQA_Challenge	MathQA_dev	BigBench	GSM8K_hard	GSM8K_test
GPT3.5 <sup>a</sup>	SC	40	40	40	40	40
	ESC	18.4 (-54.0%)	17.6 (-56.0%)	13.8 (-65.5%)	22.8 (-43.0%)	13.4 (-66.5%)
	ASC	16.0 (-60.0%)	14.7 (-63.2%)	12.3 (-69.2%)	19.0 (-52.5%)	10.9 (-72.8%)
	<b>RASC</b>	7.6 (-81.0%)	8.4 (-79.0%)	5.1 (-87.3%)	4.3 (-89.2%)	6.3 (-84.2%)
Claude3 <sup>b</sup>	SC	40	40	40	40	40
	ESC	17.2 (-57.0%)	15.4 (-61.5%)	11.5 (-71.2%)	17.2 (-57.0%)	11.5 (-71.2%)
	ASC	15.0 (-62.5%)	12.9 (-67.8%)	10.8 (-73.0%)	14.4 (-64.0%)	8.7 (-78.2%)
	<b>RASC</b>	12.0 (-70.0%)	6.7 (-83.2%)	5.2 (-87.0%)	3.3 (-91.8%)	4.1 (-89.8%)
Llama3 <sup>c</sup>	SC	40	40	40	40	40
	ESC	10.5 (-73.8%)	11.0 (-72.5%)	16.1 (-59.8%)	29.5 (-26.2%)	16.1 (-59.8%)
	ASC	9.2 (-77.0%)	8.8 (-78.0%)	13.7 (-65.8%)	25.9 (-35.2%)	12.3 (-69.2%)
	<b>RASC</b>	8.7 (-78.2%)	6.1 (-84.8%)	11.8 (-70.5%)	4.2 (-89.5%)	6.0 (-85.0%)

<sup>a</sup>gpt-3.5-turbo <sup>b</sup>claude-3-haiku <sup>c</sup>llama3-7B

Table 2: Cost Analysis of Methods on GPT-3.5 Turbo on Average per one thousand questions

Method	Accuracy (%)	Cost 1k samples (\$)
SC	50.7	4.23
ESC	50.3 (-0.8%)	1.76 (-58.4%)
ASC	50.6 (-0.2%)	1.50 (-64.5%)
<b>RASC</b>	54.7 (+7.9%)	0.65 (-84.6%)

accuracy. However, this also requires more computational resources. The optimal value of  $N$  should be chosen based on the desired balance between accuracy and computational cost. In our experiments, we found that  $N = 3$  provides a good trade-off.

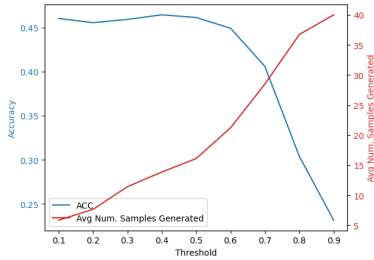
Figure 4: Effects of varying the confidence threshold ( $T$ ) on performance tradeoffs, illustrating how changes in  $T$  impact both accuracy and the average number of samples generated.

Figure 4 shows how changing the confidence threshold ( $T$ ) affects the accuracy and average number of samples generated by RASC. As  $T$  increases from 0.1 to 0.5, accuracy improves, but more sam-

ples are needed, thus increasing computational cost. However, when  $T$  exceeds 0.5, accuracy drops. This is because a high threshold makes RASC too selective, thus rejecting correct samples that don't meet the strict confidence requirement. This can lead to difficulty reaching consensus and may bias towards common answers. The optimal threshold aims to balance accuracy and inclusivity. In our experiments, a threshold of 0.1 struck a good balance between accuracy and efficiency.

These ablation studies demonstrate the flexibility of the RASC method in controlling the trade-off between accuracy and computational cost by adjusting the hyperparameters  $N$  and  $T$ . By carefully tuning these parameters, RASC can be optimized for different scenarios and requirements, depending on the priority given to accuracy or efficiency.

**Impact of Confidence Scoring Model:** The performance of different confidence scoring models, as summarized in Table 3, identifies Logistic Regression as the most effective model in terms of accuracy and sample utilization. This finding shows the importance of selecting an appropriate confidence scoring model to maximize the benefits of the RASC approach. These results are obtained by training the confidence scoring models on the training set and then fine-tuning the best set of hyperparameters ( $N$  and  $T$ ) using the customized metrics. The final evaluation is performed on the test set to ensure the robustness of the model.

Among the confidence scoring models evaluated **Logistic Regression** achieves the highest accuracy

Table 3: Performance Comparison on Different Individual Confidence Scoring Models

Model	Accuracy (%)	Num. Samples
Random	41.9	7.77
NB	45.9	5.91
<b>LR</b>	46.0	5.87
RF	45.8	6.38
HHEM <sup>1</sup>	42.4	7.52

<sup>1</sup>Hallucination Detector Model based on microsoft/deberta-v3-base and is trained initially on NLI data

of 46.0% while requiring the lowest average number of samples (5.87) compared to other models. This indicates that it effectively captures the relationship between the features extracted from the RP and the likelihood of hallucination, which allows RASC to make informed decisions during the weighted majority voting process.

The accuracy of Random model drops below the baseline of Self-Consistency. The poor performance of the Random model, which assigns confidence scores randomly, emphasizes the importance of using a well-designed confidence scoring model in the RASC approach. Without a meaningful assessment of the generated content’s quality and consistency, the weighted majority voting process cannot effectively distinguish between reliable and unreliable samples, leading to suboptimal results.

The HHEM Model, which is a DeBERTa-based hallucination detector sourced from Hugging Face (Honovich et al., 2022), does not perform as well as the other models in this context. This suggests that relying solely on a pre-trained model without considering the specific characteristics and requirements of the RASC approach may not yield the best results. The superior performance of models like Logistic Regression (LR), Naive Bayes (NB), and Random Forest (RF), which utilize manual feature engineering tailored to the RASC framework, highlights the importance of crafting features that capture the nuances of the generated content as we discussed in the method section.

**Out-of-Distribution Performance:** We evaluate RASC’s performance on out-of-distribution (OOD) data to assess its generalizability in unseen queries.

The left panel of Figure 5 presents the results for the dataset obtained from the work by ASC, showcasing RASC’s performance on various unseen categories. Across all categories, RASC consistently

achieves higher accuracy than the SC method while requiring fewer steps. The right panel of Figure 5 illustrates RASC’s performance on the dataset obtained from the work by ES, which includes different LLMs and prompts. Once again, RASC demonstrates superior performance across all categories, maintaining high accuracy while significantly reducing the number of steps. These results underscore RASC’s adaptability to different language models and its effectiveness in handling diverse prompts. For more details regarding the performance comparison, refer to Figure 10, 14, 12, 16, in the Appendix B.2

The results presented in Figure 5 provide strong evidence for RASC’s ability to maintain high performance and efficiency even when faced with unseen data, LLMs, and prompts. Such robustness and generalizability are crucial for real-world applications, where the model must adapt to a wide range of scenarios and challenges without sacrificing accuracy or computational resources.

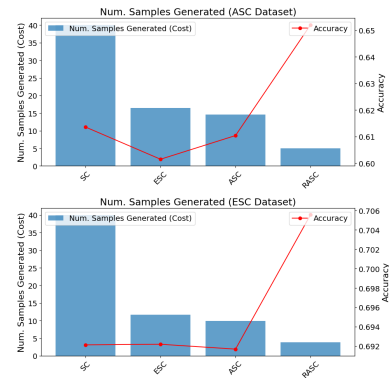


Figure 5: Performance of models on data with unseen categories and different prompts, assessing their stability and reliability across unanticipated scenarios.

## 5 Conclusion

In this paper, we introduce Reasoning-Aware Self-Consistency (RASC), a novel approach that enhances the reliability and efficiency of large language models (LLMs) by dynamically adjusting the number of samples based on the quality and consistency of Reasoning Paths (RPs). RASC assigns confidence scores to each sampled RP and employs weighted majority voting, significantly improving the process of generating reliable answers while reducing computational costs. Our evaluations demonstrate RASC’s effectiveness in improving effectiveness and efficiency compared to traditional Self-Consistency methods, marking



a significant step towards optimizing LLMs for practical applications.

## 6 Limitations

Despite the demonstrated effectiveness of Reasoning-Aware Self-Consistency (RASC) in improving the efficiency and reliability of large language models (LLMs), several limitations remain:

- **Hyperparameter Sensitivity and Computational Overhead:** RASC’s performance heavily relies on carefully tuned hyperparameters, which may vary across datasets, models, and applications. Additionally, the feature extraction and confidence scoring introduce computational overhead that could be significant in large-scale or real-time scenarios.
- **Limited Feature Set and Soft Approximation:** The current preliminary feature set may not capture all nuances of generated content, potentially affecting the model’s effectiveness. Moreover, the individual confidence scores provide a soft approximation of hallucination likelihood, which may not accurately capture all instances.
- **Potential Bias in Training Data:** Biases present in the training data used to build confidence scoring models could influence RASC’s outputs, emphasizing the importance of ensuring data diversity and fairness to mitigate biases and enhance the fairness of generated RPs.

These limitations highlight areas for future research and underscore the need for continued improvement and adaptation of RASC to ensure its applicability and effectiveness in diverse settings.

## References

- Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. 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.
- Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.
- Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. *Mathqa: Towards interpretable math word problem solving with operation-based formalisms*. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.
- Anthropic. 2024. *The claude 3 model family: Opus, sonnet, haiku*. Available online. Accessed: 2024-06-01.
- Md Adnan Arefeen, Biplob Debnath, and Srimat Chakradhar. 2024. Leancontext: Cost-efficient domain-specific question answering using llms. *Natural Language Processing Journal*, 7:100065.
- Fu Bang. 2023. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218.
- BIG bench authors. 2023. *Beyond the imitation game: Quantifying and extrapolating the capabilities of language models*. *Transactions on Machine Learning Research*.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Grestenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefer. 2024. *Graph of Thoughts: Solving Elaborate Problems with Large Language Models*. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. *Training verifiers to solve math word problems*. *Preprint*, arXiv:2110.14168.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. *PAL: Program-aided language models*. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.
- Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. *Did aris-totle use a laptop? a question answering benchmark with implicit reasoning strategies*. *Preprint*, arXiv:2101.02235.

- Or Honovich, Roei Aharoni, Jonathan Herzig, Hagai Taitelbaum, Doron Kukliansky, Vered Cohen, Thomas Scialom, Idan Szpektor, Avinatan Hassidim, and Yossi Matias. 2022. [True: Re-evaluating factual consistency evaluation](#). *Preprint*, arXiv:2204.04991.
- Tom Hosking, Phil Blunsom, and Max Bartolo. 2023. Human feedback is not gold standard. *arXiv preprint arXiv:2309.16349*.
- Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.
- Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*.
- Mingyu Jin, Qinkai Yu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, Mengnan Du, et al. 2024. The impact of reasoning step length on large language models. *arXiv preprint arXiv:2401.04925*.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022a. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.
- Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022b. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213.
- Stefanie Krause and Frieder Stolzenburg. 2023. Commonsense reasoning and explainable artificial intelligence using large language models. In *European Conference on Artificial Intelligence*, pages 302–319. Springer.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.
- Junlong Li, Fan Zhou, Shichao Sun, Yikai Zhang, Hai Zhao, and Pengfei Liu. 2024a. Dissecting human and llm preferences. *arXiv preprint arXiv:2402.11296*.
- 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*.
- Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2024c. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18608–18616.
- Meta. 2024. Meetllama3. Blog post. Available: <https://llama.meta.com/llama3/>.
- OpenAI. 2024. Introducing gpt-3.5 turbo. Blog post. Available: <https://platform.openai.com/docs/models/gpt-3-5-turbo>.
- Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-bert: Sentence embeddings using siamese bert-networks](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520.
- Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. [Commonsenseqa: A question answering challenge targeting commonsense knowledge](#). *Preprint*, arXiv:1811.00937.
- Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023. Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family. In *International Semantic Web Conference*, pages 348–367. Springer.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. [Llama: Open and efficient foundation language models](#). *Preprint*, arXiv:2302.13971.
- Zhengkai Tu, Wei Yang, Zihang Fu, Yuqing Xie, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2020. Approximate nearest neighbor search and lightweight dense vector reranking in multi-stage retrieval architectures. In *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*, pages 97–100.
- Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2022a. Towards understanding chain-of-thought prompting: An empirical study of what matters. *arXiv preprint arXiv:2212.10001*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022b. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of Thoughts: Deliberate problem solving with large language models](#). *Preprint*, arXiv:2305.10601.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Peng Gao, et al. 2024. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? *arXiv preprint arXiv:2403.14624*.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren’s song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. [Least-to-most prompting enables complex reasoning in large language models](#). *Preprint*, arXiv:2205.10625.

## A Computational Details

### A.1 Number of Parameters

The language models used in our experiments have varying numbers of parameters:

- LLAMA3-8B: This model has 8 billion parameters ([Meta, 2024](#)).
- GPT3.5-turbo: The exact number of parameters for this model is not publicly disclosed by OpenAI ([OpenAI, 2024](#)).
- Claude-3-Haiku: The number of parameters for this model is not publicly available from Anthropic ([Anthropic, 2024](#)).

### A.2 Computational Budget

To generate all the collected data for our experiments using the LLAMA3-8B model, we consumed a total of approximately 100 GPU hours. This computational budget was used for running the model, generating CoT samples, and processing the results.

### A.3 Computing Infrastructure

Our experiments were conducted on a computing infrastructure equipped with the following hardware:

- GPU: NVIDIA GeForce RTX 3070 Ti
- CPU: 16 cores 11th Generation Intel Core i7 Processors

### A.4 Used Packages

In this study, several online available Python packages are used to conduct experiment and analysis, the packages are as follows:

- NLTK: For calculating Jaccard Similarity, Ngram, tokenizer.
- statistics: For computing logistic regression.
- PyTorch: For using pre-trained LLM.
- pandas: For data manipulation.
- json: loading and saving json data.
- sklearn: For supervised learning models training and evaluation.
- adaptive\_consistency: For implementing adaptive consistency algorithm.
- Levenshtein: For computing Levenshtein distance.
- transformer: For huggingface PTM usage.
- LangChain: For LLM API usage and answer parser.

## B Experiments: Additional Results

### B.1 Robustness of RASC Across Different Seeds

To evaluate the robustness and consistency of our Reasoning-Aware Self-Consistency (RASC) method, we conducted experiments using multiple random seeds for data sampling and model initialization. Tables 4 and 5 present the mean, standard deviation, and p-values of accuracy and cost (number of generated samples) of the RASC method on various benchmark categories across different models. Table 4 shows the mean accuracy of RASC along with the standard deviation across different random seeds. The small standard deviations indicate that the performance of RASC

remains consistent and stable regardless of the specific data split or model initialization. Additionally, the p-values are calculated to test if the accuracy of RASC is significantly better than the performance of the Self-Consistency (SC) method. The majority of the p-values are well below the significance level of 0.05, suggesting that RASC achieves statistically significant improvements over SC in most cases. Similarly, Table 5 presents the mean cost (number of generated samples) of RASC along with the standard deviation across different random seeds. The low standard deviations demonstrate that the sample efficiency of RASC is robust and consistent across various data splits and model initializations. The p-values in this table are calculated to test if the cost of RASC is significantly smaller than the fixed budget of 40 samples used in SC. The extremely low p-values (all below  $10^{-10}$ ) provide strong evidence that RASC significantly reduces the number of required samples compared to SC, regardless of the random seed. The results in these tables highlight the robustness and reliability of our RASC method. The consistent performance and sample efficiency across different random seeds suggest that the improvements achieved by RASC are not sensitive to specific data splits or model initializations. This robustness is crucial for the practical application of RASC in real-world scenarios, where the method needs to perform well under various conditions and datasets.

## B.2 Additional Results on Our Datasets

In this subsection, we present additional experimental results on our own datasets to further analyze the performance and cost-effectiveness of the proposed RASC method compared to other baseline methods. Table 2 shows a cost analysis of different methods on the Claude-3-Haiku model, averaged over one thousand questions. RASC achieves the highest accuracy while significantly reducing the cost compared to the standard Self-Consistency (SC) method. Figure 6 compares the accuracy of our RASC method across different categories on our collected datasets. In most categories, we observe a performance gain with our method, with a slight drop in the GSM8K\_test category due to a more significant reduction in sample generation. Figure 7 presents the accuracy comparison of RASC across different models on our collected datasets. We find that the performance either significantly improves (GPT3.5/llama3) or remains comparable to the previous method for all models. Figure 8 pro-

vides a complete view of the accuracy comparison of RASC across different categories and models of our collected datasets using a heatmap representation.

## B.3 Additional Results on ASC’s Datasets

To further validate the effectiveness of RASC, we evaluate its performance on the datasets used in the Adaptive Self-Consistency (ASC) paper (Aggarwal et al., 2023). Figure 9 compares the accuracy of RASC across different models on ASC’s datasets. We observe a significant improvement on the vicuna-13b model but a slight degradation in performance on the code-davinci-002 model, which is compensated by a more substantial reduction in sample generation, as shown in Figure 11. Figure 10 presents a comprehensive view of the accuracy comparison of RASC across different categories and models of ASC’s datasets using a heatmap representation. Figure 11 compares the reduction in the number of samples generated by RASC across different models on ASC’s datasets. We find a smaller improvement on the vicuna-13b model but a more significant reduction on the code-davinci-002 model, despite a slight performance drop in accuracy, as shown in Figure 9. Figure 12 provides a complete view of the number of samples generated by RASC across different categories and models of ASC’s datasets using a heatmap representation.

## B.4 Additional Results on ESC’s Datasets

We also evaluate the performance of RASC on the datasets used in the Early-Stopping Self-Consistency (ESC) paper (Li et al., 2024b). Figure 13 compares the accuracy of RASC across different models on ESC’s datasets, showing consistency with existing methods for all three models they used. Figure 14 presents a comprehensive view of the accuracy comparison of RASC across different categories and models of ESC’s datasets using a heatmap representation. Figure 15 compares the number of samples generated by RASC across different models on ESC’s datasets, demonstrating a significant improvement over existing methods for all three models they used. Figure 16 provides a complete view of the number of samples generated by RASC across different categories and models of ESC’s datasets using a heatmap representation.



Table 4: Mean, Standard Deviation, and p-value (to test if it’s significantly better than the performance of Self-Consistency) of Accuracy of RASC Method on Various Benchmarks on 10 different seeds

Model	Metric	Benchmark Category				
		MathQA_Challenge	MathQA_dev	BigBench_easy	GSM8K_hard	GSM8K_test
GPT3.5 <sup>a</sup>	Mean	0.499	0.498	0.276	0.465	0.764
	Std	(0.037)	(0.019)	(0.024)	(0.023)	(0.032)
	p-value	$2.16 \times 10^{-2}$	$8.50 \times 10^{-5}$	$7.74 \times 10^{-1}$	$3.54 \times 10^{-1}$	$2.10 \times 10^{-5}$
Claude3 <sup>b</sup>	Mean	0.512	0.529	0.263	0.395	0.801
	Std	(0.035)	(0.029)	(0.030)	(0.043)	(0.024)
	p-value	$3.70 \times 10^{-4}$	$1.83 \times 10^{-4}$	$6.53 \times 10^{-1}$	$6.57 \times 10^{-1}$	$7.11 \times 10^{-6}$
Llama3 <sup>c</sup>	Mean	0.131	0.161	0.303	0.247	0.708
	Std	(0.023)	(0.023)	(0.033)	(0.028)	(0.028)
	p-value	$7.39 \times 10^{-6}$	$6.35 \times 10^{-2}$	$8.81 \times 10^{-10}$	$1.63 \times 10^{-4}$	$6.68 \times 10^{-4}$

<sup>a</sup>GPT-3.5-Turbo, <sup>b</sup>Claude-3-Haiku, <sup>c</sup>Llama3-7B

Table 5: Mean, Standard Deviation, and p-value (to test if it’s significantly lower than the fixed number of generations, 40, of Self-Consistency) of the number of generated samples of RASC Method on Various Benchmarks on 10 different seeds

Model	Metric	Benchmark Category				
		MathQA_Challenge	MathQA_dev	BigBench_easy	GSM8K_hard	GSM8K_test
GPT3.5 <sup>a</sup>	Mean	8.900	8.461	4.559	4.061	3.106
	Std	(1.011)	(0.859)	(0.293)	(0.353)	(0.061)
	p-value	$3.26 \times 10^{-15}$	$6.61 \times 10^{-16}$	$1.47 \times 10^{-20}$	$6.81 \times 10^{-20}$	$7.68 \times 10^{-27}$
Claude3 <sup>b</sup>	Mean	7.907	7.747	3.004	3.191	3.127
	Std	(0.974)	(1.308)	(0.005)	(0.132)	(0.120)
	p-value	$1.76 \times 10^{-15}$	$2.38 \times 10^{-14}$	$6.44 \times 10^{-37}$	$7.73 \times 10^{-24}$	$3.29 \times 10^{-24}$
Llama3 <sup>c</sup>	Mean	7.803	7.617	11.569	3.056	3.001
	Std	(0.739)	(0.872)	(0.715)	(0.033)	(0.003)
	p-value	$1.41 \times 10^{-16}$	$5.96 \times 10^{-16}$	$3.22 \times 10^{-16}$	$3.03 \times 10^{-29}$	$6.79 \times 10^{-39}$

<sup>a</sup>GPT-3.5-Turbo, <sup>b</sup>Claude-3-Haiku, <sup>c</sup>Llama3-7B

## C Sample Generation Prompt

Table 6: Cost Analysis of Methods on Claude-3-Haiku on Average per one thousand questions

Method	Accuracy (%)	Cost 1k samples (\$)
SC	55.1	4.23
ESC	54.9 (-0.36%)	1.49 (-64.78%)
ASC	54.9 (-0.36%)	1.28 (-69.74%)
RASC	55.6 (+0.91%)	0.64 (-84.87%)

In this study, we adopt CoT prompting which explicitly requires LLMs to RP to the question *step by step*. Since the OOD test dataset uses few-shot prompting and budget constrain, we instead utilize zero-shot prompting to generate RPs in our own datasets. The prompt is defined as follows:

**System Message:** *You are a professional specialized in {subject}. You need to help me answer the given question. Notice that you need to solve the question step by step and as detailed as possible. Do not jump to the answer directly. You must follow the RP format instructions.*  
**Human Message:** {question}

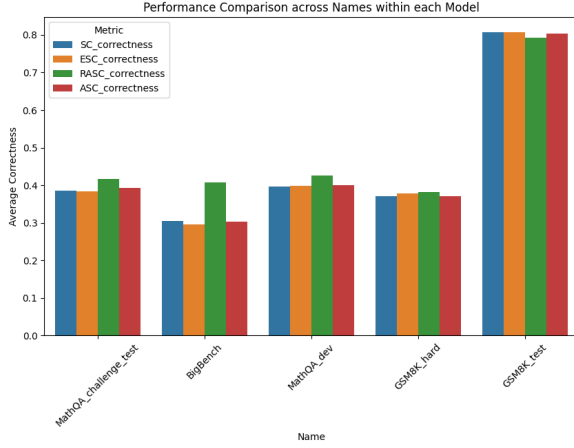


Figure 6: Comparison of Accuracy of our method (RASC) on different categories on our collected datasets. In most of categories we see a performance gain of our methods; we saw a drop in GSM8K\_test, but this is mostly due to the more significant reduction in samples generation in this specific category

## D Individual Quality Features Extraction

In this study, we design four individual quality features to measure the reliability of each individual reasoning path. The four individual quality features are RP length, input coherence, format error, and error-admitting terms.

### D.1 Reasoning Path Length

RP length varies from question category to question category. For instance, the required length of RP for solving a simple math problem (such as 'what is  $3^2 + 4^2$ ') is significantly less than solving partial differential equations. Therefore, we pre-define some thresholds for different categories. If the proposed RP length is more than this threshold, we consider it as long (1 as a binary indicator), otherwise is small. In our RP generation stage, we explicitly require LLM to state the step number (Step 1: ...; Step 2: ...). Therefore, in our study, the threshold is used to compare the number of proposed steps from RP. In the OOD dataset or in general cases where this instruction is not provided, we consider the number of sentences as the length of PR.

### D.2 Input coherence

Since we are aiming for a lightweight feature extraction method, advanced name entity recognition or similar keyword comparison methods are not used to compare the RP with the input question. Instead, semantic similarity with Jaccard is used

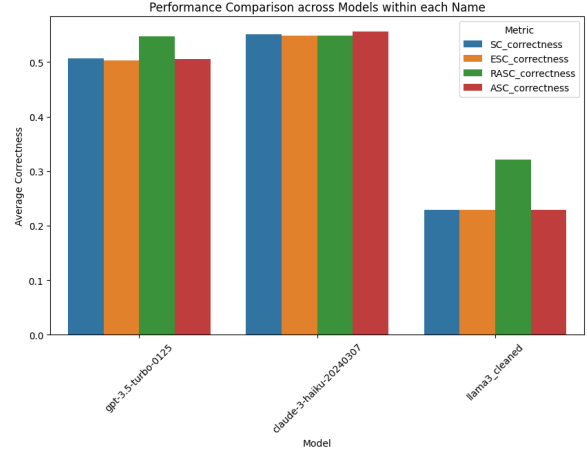


Figure 7: Comparison of Accuracy of our method (RASC) on different models on our collected datasets. In all models we found that the performance either gain significantly (GPT3.5/llama3) or stays around the same as the previous method

to approximate the coherence between the RP and input questions. The higher the similarity, the RP is considered to have more input coherence.

### D.3 Format Error

The prompt-response nature of LLMs does not allow users to explicitly obtain well-structured fields (such as JSON or dictionary in Python) since the responses are often in string format. Therefore, the common practice is asking LLMs to output response strings with clear format and then using regular expressions to parse out these fields of interest. When the testing QA dataset is large and diverse, it is hard to propose a uniform parser to extract information. Moreover, instructional following hallucination is another common behaviour for LLMs. This prevents users from really extracting what they want from responses, especially in massive test cases.

A common instruction that people use to extract answers in QA tasks is asking LLMs to explicitly end their response with fixed term *The Answer is xxx*. Once the LLM varies its expression (for instance *xxx is the answer*), the parser fails and outputs the wrong answer even if the RP is actually correct. However, this is an unavoidable problem as long as instruction following hallucination still exists. Therefore, it is considered a low-quality indicator since it definitely outputs a false answer (parser fails). In our study, we adopt parsers provided by LangChain since it contains well-structured parsing logic. Yet due to the unsta-



Figure 8: Comparison of Accuracy of our method (RASC) on different categories and models of our collected datasets from a complete view via heatmap

ble behaviour of LLMs, this parser fails often as well.

#### D.4 Frequent Error-admitting Terms for Various LLMs

By observing the generated samples, we observe several common error-admitting terms that LLMs often use to acknowledge the previous mistakes they made during the RP. We conduct systematic evaluation and list out some of the most frequent error-admitting terms used by various LLMs in Fig.17-19. These error-admitting terms, such as "There seemed to be a mistake in the previous calculation" have demonstrated a high risk of sample hallucination. Therefore, by parsing out these terms, we obtain an error-admitting quality feature.

### E Relative Quality Feature Extraction

In this study, we use semantic similarity to determine the relative quality features between reasoning paths. To further describe the method for obtaining semantic similarities, we include a thorough discussion of similarity algorithms and mechanisms.

#### E.1 Similarity Algorithms

The proposed relative quality feature requires computing the semantic similarity between different sampled RPs. Therefore, it is essential to find (1) a tokenization method that captures the semantics of the given text, and (2) an appropriate calculation to compute similarity between embeddings. In our experiment, we tested several widely used tokenization methods, including TF-IDF (Robertson, 2004), Count Tokenizer, GloVe (Pennington et al., 2014), and Sentence Transformer (Reimers and Gurevych, 2019). For similarity computation, we tested classic Euclidean distance and cosine similarity. On the other hand, Jaccard Similarity and Levenshtein distance are two NLP textual similarity algorithms which can be applied directly to the corpus, they are included in our test as well. The comparison of some of the tokenizer and computation results is shown in Fig.E.1. Our results indicate that these combinations can all perform good semantic similarity calculations. Therefore, we use the Jaccard Similarity Index as our primary similarity algorithm in our study due to its computational efficiency (required by our method) and comparable performance.

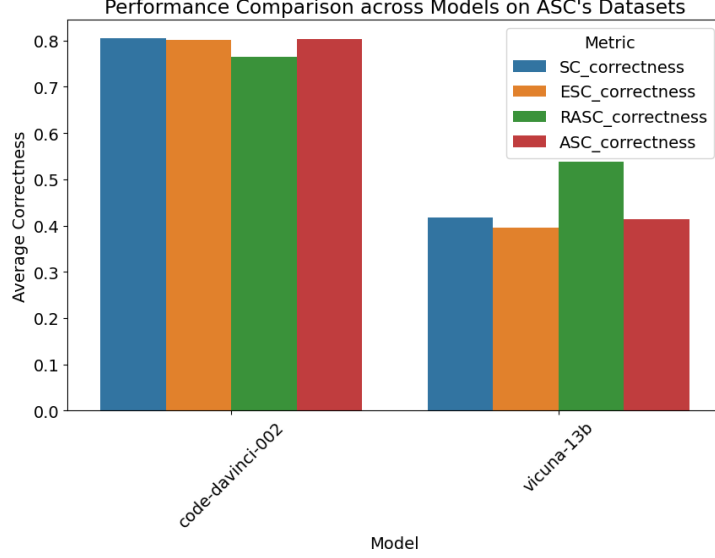


Figure 9: Comparison of Accuracy of our method (RASC) on different models of ASC’s datasets. we found a significant improvement on the vicuna-13b model but a slight degradation on code-davinci-002 model (but with a more significant reduction on samples generation as shown in Fig 11

## E.2 Similarity Mechanism Robust and Sensitivity Proof

In this study, we adopt three mechanisms to calculate the semantic similarity between sampled RPs and the consistency between sampled answers. The sensitivity and robustness of these mechanisms are shown in Fig.21. To prove the sensitivity and robustness of aggregation and pairwise similarity, we conduct the following proof.

Given that we are using the Jaccard Similarity Index, the equation to compute similarity is given by:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (4)$$

where  $A$  and  $B$  represent two documents/corpus and  $|A|$  is the cardinality of the document  $|A|$ . In our study, we denote  $C_t$  as the document at sampling step  $t$ . Before getting into the proof, we have the following Lemma:

**Lemma E.1.** *Let  $N(X)$  be a system follows Gaussian Distribution with  $X = \{X_1, X_2, \dots, X_n\}$ , a set of random variable  $X_i$ . The individual sensitivity of  $X_i$  will accumulate and affect the overall sensitivity of  $N(X)$  regardless of the averaging operation.*

*Proof.* By Chebyshev’s Inequality:  $P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$  for any random variable  $X$  with mean  $\mu$  and variance  $\sigma^2$ . For each  $X_i$ , it has its own mean  $\mu_i$  and variance  $\sigma_i^2$ . The averaging effect of  $X_i$  is

given by:

$$\bar{X} = \frac{1}{t-1} \sum_{i=1}^{t-1} X_i$$

The variance of the average is:

$$\text{Var}(\bar{X}) = \frac{1}{(t-1)^2} \sum_{i=1}^{t-1} \sigma_i^2$$

Applying Chebyshev’s Inequality to  $\bar{X}$ , we get:

$$P(|\bar{X} - \mu| \geq k\sigma_{\bar{X}}) \leq \frac{1}{k^2}$$

where  $\sigma_{\bar{X}} = \sqrt{\text{Var}(\bar{X})} = \frac{1}{t-1} \sqrt{\sum_{i=1}^{t-1} \sigma_i^2}$ .

Despite averaging, if the individual  $X_i$  have high variances  $\sigma_i^2$ , then  $\sigma_{\bar{X}}$  will also be large. This implies that:

$$P\left(|\bar{X} - \mu| \geq k \cdot \frac{1}{t-1} \sqrt{\sum_{i=1}^{t-1} \sigma_i^2}\right) \leq \frac{1}{k^2}$$

Thus,  $\bar{X}$  still has a considerable probability of deviating from its mean, indicating persistent sensitivity. Hence, the individual sensitivity of  $X_i$  accumulates and affects the overall sensitivity of  $N(X)$  regardless of the averaging operation.  $\square$

To prove the robustness and sensitivity of the adopted similarity mechanisms, we introduce the following theorem:



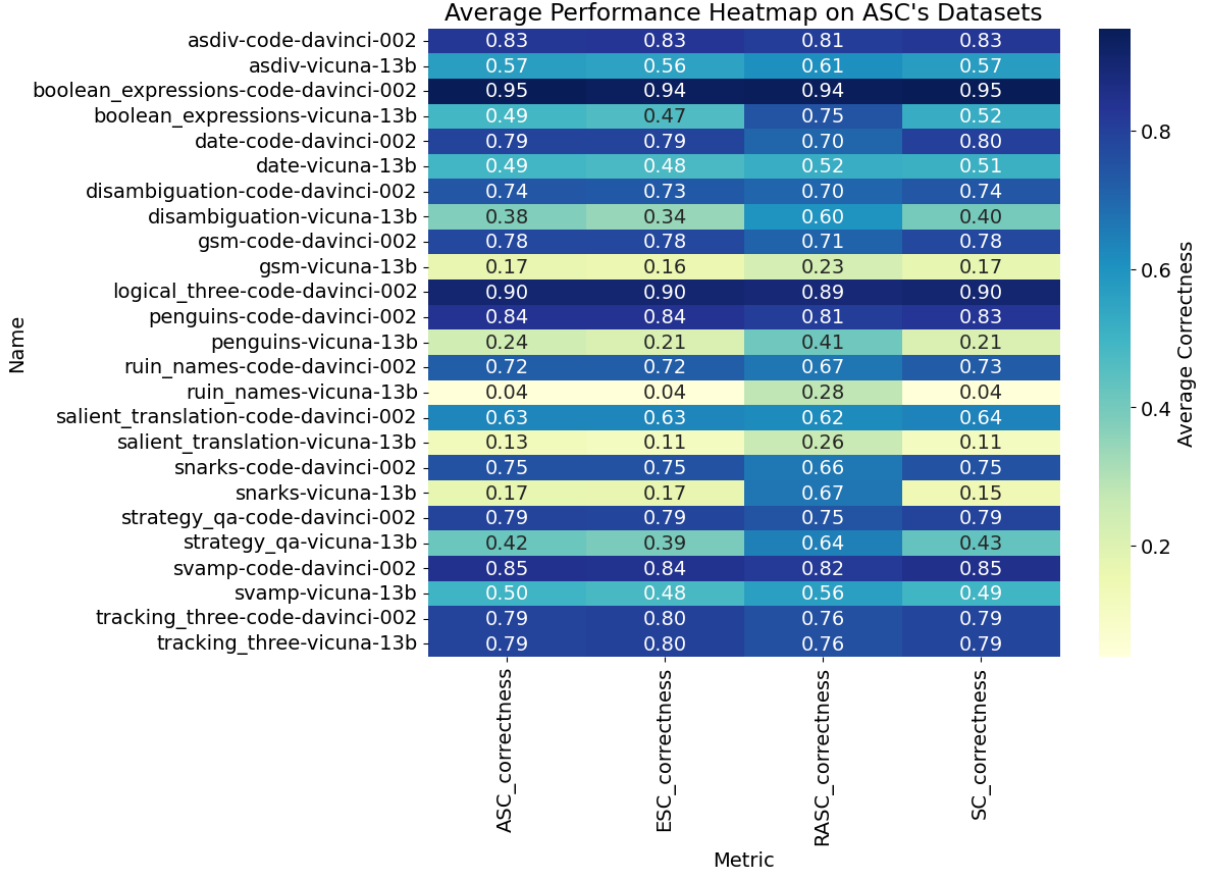


Figure 10: Comparison of Accuracy of our method (RASC) on different categories and models of ASC's datasets from a complete view via heatmap

**Theorem E.2.** *Aggregation similarity is more robust to noise compared to pairwise similarity when the document  $C_t$  vocabulary size is large. Both aggregation and pairwise similarity are sensitive when the document vocabulary size is small.*

*Proof.* Let  $C_{\text{agg},t-1} = C_{1:t-1} = \bigcup_{i=1}^{t-1} C_i$ :

$$J_{\text{agg},t} = J(C_t, C_{1:t-1}) = \frac{|C_t \cap C_{1:t-1}|}{|C_t \cup C_{1:t-1}|}$$

- If  $C_t \subseteq C_{1:t-1} \Rightarrow J_{\text{agg},t} = \frac{|C_t|}{|C_{1:t-1}|}$
- If  $C_t \cap C_{1:t-1} = \emptyset \Rightarrow J_{\text{agg},t} = 0$
- If  $C_t \cap C_{1:t-1} = \lambda, 0 < \lambda < |C_t|, \lambda \in \mathcal{Z}$ :

$$J_{\text{agg},t} = \frac{\lambda}{|C_t| + |C_{1:t-1}| - \lambda}$$

$$\frac{dJ_{\text{agg},t}}{d\lambda} = \frac{|C_t| + |C_{1:t-1}|}{(|C_t| + |C_{1:t-1}| - \lambda)^2} \quad (*)$$

For pairwise similarity:

$$\begin{aligned} E(J_{\text{pww},t}) &= \frac{1}{t-1} \sum_{i=1}^{t-1} J(C_t, C_i) \\ &= \frac{1}{t-1} \sum_{i=1}^{t-1} \frac{|C_t \cap C_i|}{|C_t \cup C_i|} \end{aligned}$$

- If  $C_t \subseteq C_i$  or  $C_i \subseteq C_t$   
 $\Rightarrow J(C_t, C_i) = \frac{\min(|C_t|, |C_i|)}{\max(|C_t|, |C_i|)}$
- If  $C_t \cap C_i = \emptyset \Rightarrow J(C_t, C_i) = 0$
- If  $C_t \cap C_i = \beta_i$ ,  
 $0 < \beta_i < \min(|C_t|, |C_i|), \beta_i \in \mathcal{Z}$ :

$$J(C_t, C_i) = \frac{\beta_i}{|C_t| + |C_i| - \beta_i}$$

$$\frac{dJ(C_t, C_i)}{d\beta_i} = \frac{|C_t| + |C_i|}{(|C_t| + |C_i| - \beta_i)^2}$$

$$\nabla E(J_{\text{pww},t}) = \frac{1}{t-1} \sum_{i=1}^{t-1} \frac{|C_t| + |C_i|}{(|C_t| + |C_i| - \beta_i)^2} \quad (**)$$

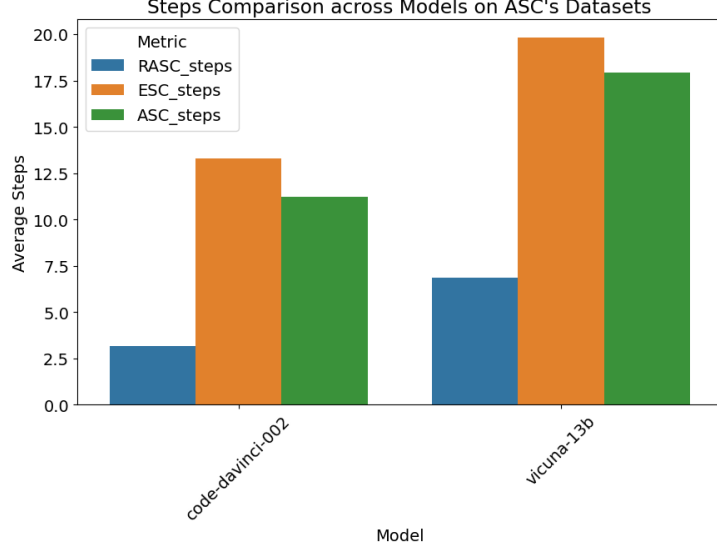


Figure 11: Comparison of Number of Samples (steps) Reduction of our method (RASC) on different models of ASC’s datasets. we found a less improvement on the vicuna-13b model but more on code-davinci-002 model (but with a slight performance drop on accuracy as shown in Fig 11

From the above derivation, we know that:

$$\frac{dJ_{agg,t}}{d\lambda} = \frac{|C_t| + |C_{1:t-1}|}{(|C_t| + |C_{1:t-1}| - \lambda)^2}$$

$$\nabla E(J_{pww,t}) = \frac{1}{t-1} \sum_{i=1}^{t-1} \frac{|C_t| + |C_i|}{(|C_t| + |C_i| - \beta_i)^2}$$

From the derivative of aggregation similarity, we can observe that the cardinality of the aggregation size plays a significant role when it gets large. As the vocabulary set expands ( $|C_{1:t-1}|$  gets larger), the derivative  $\frac{dJ_{agg,t}}{d\lambda}$  gets smaller regardless of the intersection size  $\lambda$ . Formally speaking:

$$\lim_{t \rightarrow \infty} \frac{dJ_{agg,t}}{d\lambda} \rightarrow 0$$

On the other hand, the individual pairwise similarity  $J(C_t, C_i)$  is not affected by the size  $t$ , and  $|C_t| \ll |C_{1:t-1}|$  when  $t$  gets large. Hence, the change in the interception size  $\beta_i$  dominates the rate of change, which implies it is very sensitive to the error due to the additional RP.

In addition,  $\beta_i$  has a large variance from RP to RP since they are i.i.d. This large variance, or sensitivity will accumulate as  $t$  gets larger (as more individual terms  $J(C_t, C_i)$  get in) by Lemma.E.1.  $\square$

Therefore, from Theorem.E.2, CoT RPs always contain way larger vocabulary size than answers, hence we adopt aggregation similarity mechanism for CoT RPs while pairwise similarity mechanism for answers.

## F Evaluation Metric for Evaluating Accuracy-Cost Trade-Off

In order to quantitatively assess the trade-off between accuracy and cost in our evaluations, we introduce a custom metric function that balances the contributions of both factors. This metric normalizes the accuracy and cost values and computes a weighted average to provide a single score representing the overall performance of a method.

Let  $acc$  denote the accuracy of a method, and  $cost$  denote the computational cost, measured as the number of samples generated. Additionally, let  $sc\_acc$  and  $sc\_cost$  denote the accuracy and cost of the Self-Consistency (SC) method, respectively, while  $single\_sample\_acc$  and  $direct\_cost$  denote the accuracy obtained from using only the first sample’s answer and the cost of the direct sampling method, respectively.

The metric function is defined as:

$$metric = 0.5 \times acc\_factor + 0.5 \times cost\_factor \quad (5)$$

where  $acc\_factor$  and  $cost\_factor$  are normalized values of accuracy and cost, respectively, calculated as follows:

$$acc\_factor = \begin{cases} 1 & \text{if } acc \geq sc\_acc \\ 0 & \text{if } acc \leq single\_acc \\ \frac{acc - single\_acc}{sc\_acc - single\_acc} & \text{otherwise} \end{cases} \quad (6)$$

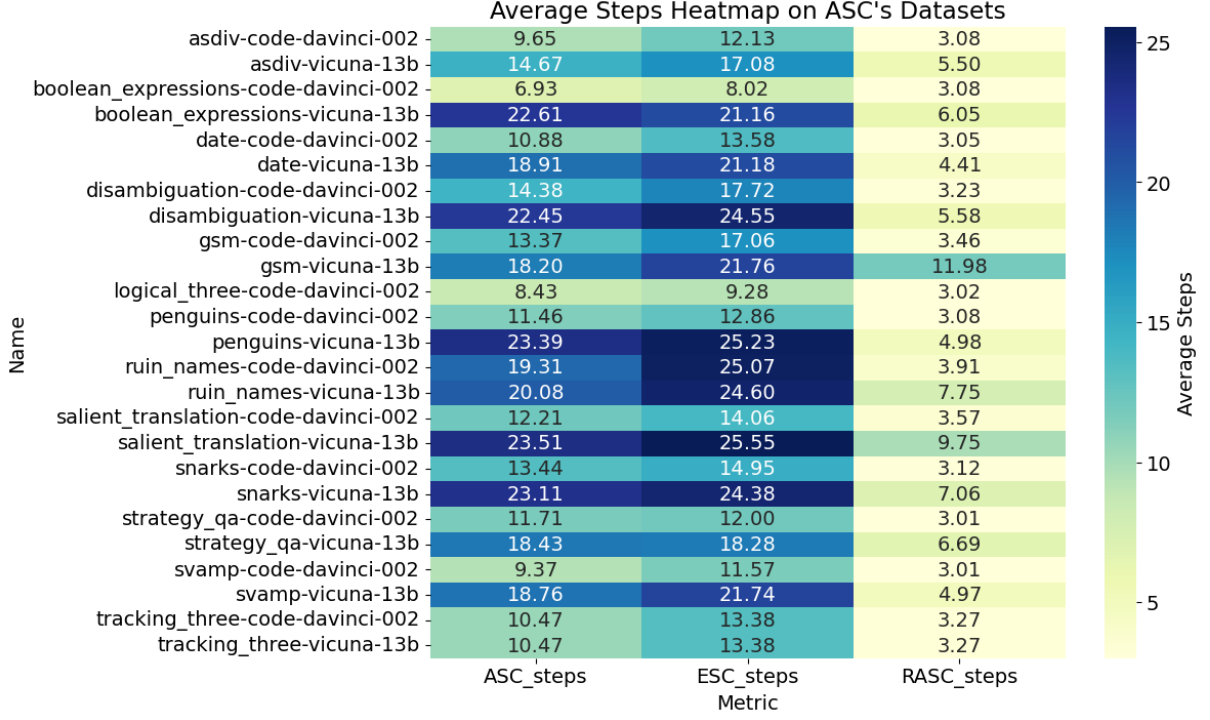


Figure 12: Comparison of Number of Samples(steps) generations of our method (RASC) on different categories and models of ASC’s datasets from a complete view via heatmap

$$\text{cost\_factor} = \begin{cases} 1 & \text{if cost} \leq \text{dir\_cost} \\ 0 & \text{if cost} \geq \text{sc\_cost} \\ \frac{\text{sc\_cost} - \text{cost}}{\text{sc\_cost} - \text{direct\_cost}} & \text{otherwise} \end{cases} \quad (7)$$

The  $\text{acc\_factor}$  is normalized to be between 0 and 1, where 1 corresponds to an accuracy greater than or equal to the SC method, and 0 corresponds to an accuracy less than or equal to the accuracy using just the first sample’s answer. Similarly, the  $\text{cost\_factor}$  is normalized to be between 0 and 1, where 1 corresponds to a cost less than or equal to the direct sampling method, and 0 corresponds to a cost greater than or equal to the SC method.

The metric function calculates the weighted average of  $\text{acc\_factor}$  and  $\text{cost\_factor}$ , giving equal importance to both accuracy and cost. A higher value of the metric indicates a better trade-off between accuracy and cost. This function enables us to compare different methods and configurations based on their ability to achieve high accuracy while minimizing computational cost. By balancing the contributions of accuracy and cost, the metric provides a comprehensive evaluation of each method’s performance in the context of the accuracy-cost trade-off.

## G Out-of-Domain Datasets

To assess the generalizability and robustness of our proposed Reasoning-Aware Self-Consistency (RASC) method, we incorporate out-of-domain (OOD) data from two seminal works: Early-Stopping Self-Consistency (ESC) (Li et al., 2024b) and Adaptive Self-Consistency (ASC) (Aggarwal et al., 2023).

### G.1 Datasets, Prompts, and Models from Early-Stopping Self-Consistency (ESC)

The ESC methodology (Li et al., 2024b) is evaluated across six benchmark datasets grouped into three reasoning task categories:

- **Arithmetic Reasoning:** MATH and GSM8K (Cobbe et al., 2021).
- **Commonsense Reasoning:** StrategyQA (Geva et al., 2021) and CommonsenseQA (Talmor et al., 2019).
- **Symbolic Reasoning:** Last Letter Concatenation and Coin Flip (Wei et al., 2022).

The prompts used are consistent with those in Wei et al. (2022), ensuring a fair comparison across datasets. Three language models of varying scales

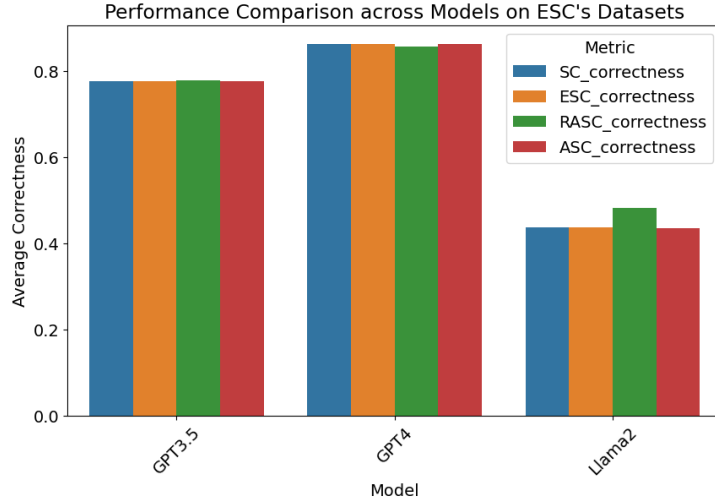


Figure 13: Comparison of Accuracy of our method (RASC) on different models of ESC’s datasets. we found a consistency among our method with existing methods for all three models they used

were used for evaluation: GPT-4, GPT-3.5-Turbo, and LLaMA-2 7B (Touvron et al., 2023).

## G.2 Datasets, Prompts, and Models from Adaptive Self-Consistency (ASC)

The ASC approach (Aggarwal et al., 2023) is tested using 17 diverse datasets categorized into four distinct areas:

- **Mathematical Reasoning:** Includes GSM-8K, SVAMP , and ASDIV .
- **Commonsense Reasoning:** StrategyQA, Date Understanding, Salient Translation, SNARKS, and RUIN Names.
- **Symbolic Reasoning:** Encompasses Tracking Shuffled Objects, Logical Deduction, Boolean Expressions, Disambiguation QA, and Penguins.
- **Code Generation:** HumanEval , MBPP, APPS , and CodeContests .

For tasks in mathematical reasoning and Date Understanding, prompts from Gao et al. (2023) were used. For commonsense and symbolic reasoning tasks, prompts from Wei et al. (2022) were employed. Code generation tasks followed were conducted using three language models: GPT-3.5-Turbo, VICUNA-13B, and CODE-DAVINCI-002.

By leveraging these datasets, prompts, and models from these studies as OOD data, we aim to rigorously test the adaptability and robustness of our RASC method across different problem domains,

task types, and datasets. This comprehensive evaluation helps illuminate the real-world applicability and resilience of our approach in varying computational environments and across diverse language model architectures.





Figure 14: Comparison of Accuracy of our method (RASC) on different categories and models of ESC's datasets from a complete view via heatmap

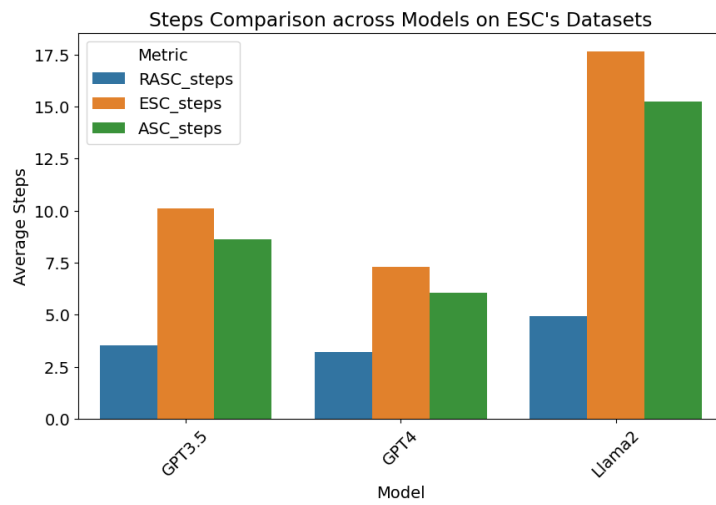


Figure 15: Comparison of Number of Samples(steps) generations of our method (RASC) on different models of ESC's datasets. we found a significant improvement among our method with existing methods for all three models they used

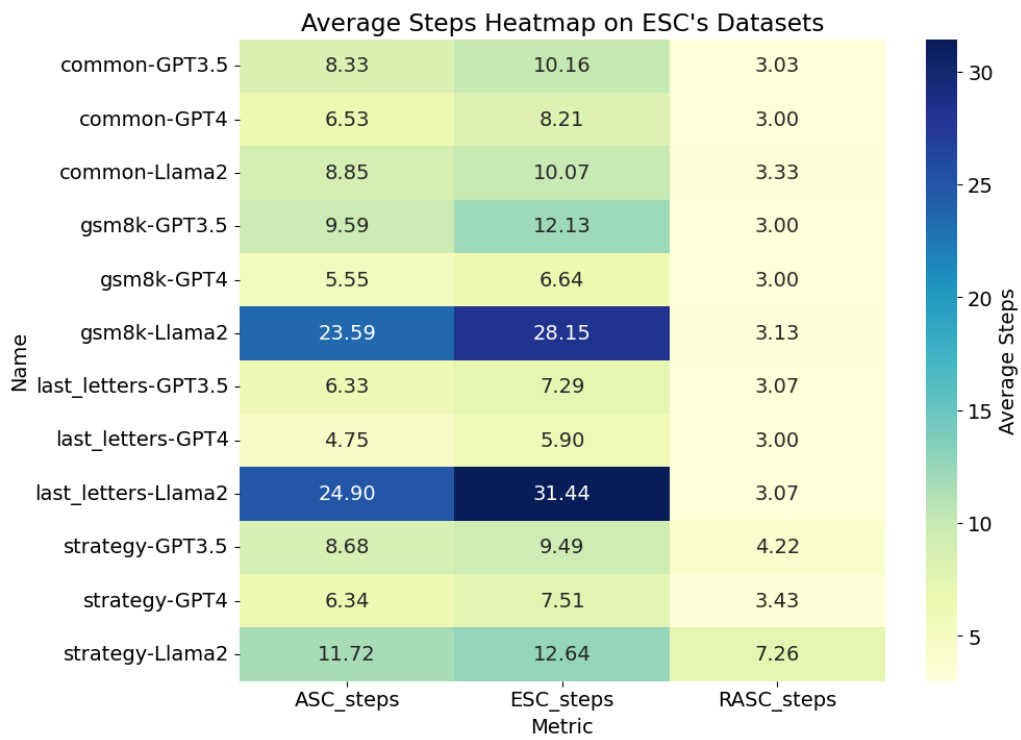


Figure 16: Comparison of Number of Samples(steps) generations of our method (RASC) on different categories and models of ESC's datasets from a complete view via heatmap

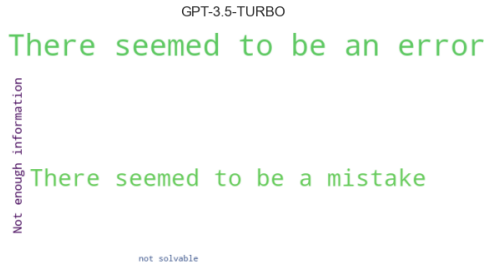


Figure 17: GPT-3.5-TURBO error-admitting terms.

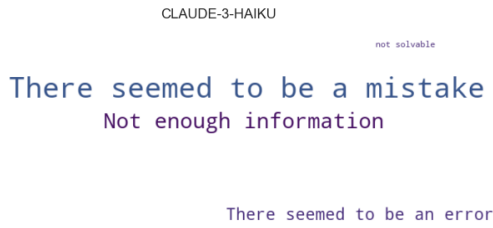


Figure 18: CLAUDE-3-HAIKU error-admitting terms.

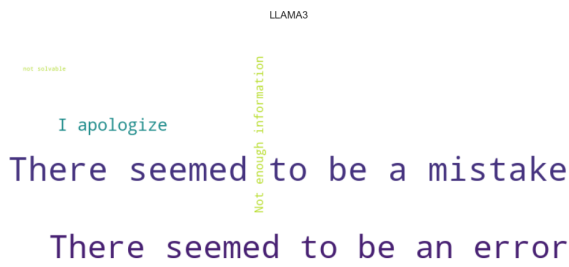


Figure 19: LLAMA3 error-admitting terms.

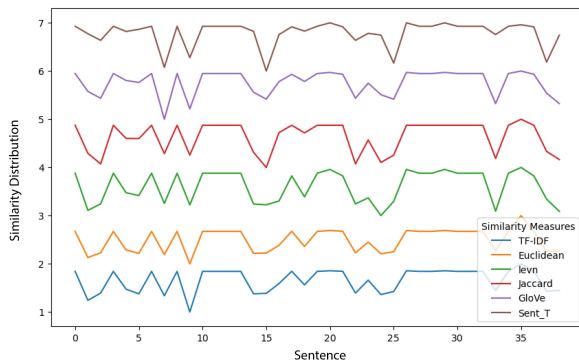


Figure 20: Similarity algorithms comparison. The original similarity distribution was from 0-1, to separate them for better visualization, we shifted different methods by a constant. TF-IDF: TF-IDF tokenizer + cosine sim; Euclidean: Count tokenizer + Euclidean Distance; levn: Levenshtein Distance; Jaccard: Jaccard Similarity Index; GloVe: GloVe tokenizer + cosine sim; Sent\_T: Sentence Transformer tokenizer + cosine sim;

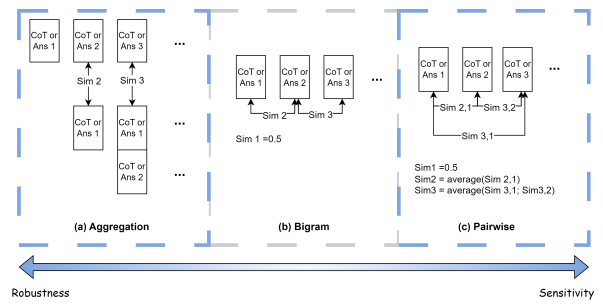


Figure 21: Similarity mechanisms.