Make Every Penny Count: Difficulty-Adaptive Self-Consistency for Cost-Efficient Reasoning

Xinglin Wang[1], **Shaoxiong Feng**[2], **Yiwei Li**[1], **Peiwen Yuan**[1], **Yueqi Zhang**[1], **Chuyi Tan**[1], **Boyuan Pan**[2], **Yao Hu**[2], **Kan Li**[1†]
[1] School of Computer Science, Beijing Institute of Technology
[2] Xiaohongshu Inc
{wangxinglin,liyiwei,peiwenyuan, zhangyq, tanchuyi, likan}@bit.edu.cn
{shaoxiongfeng2023}@gmail.com {panboyuan,xiahou}@xiaohongshu.com

Abstract

Self-consistency (SC), a widely used decoding strategy for chain-of-thought reasoning, shows significant gains across various multi-step reasoning tasks but comes with a high cost due to multiple sampling with the preset size. Its variants, Adaptive self-consistency (ASC) and Early-stopping self-consistency (ESC), dynamically adjust the number of samples based on the posterior distribution of a set of pre-samples, reducing the cost of SC with minimal impact on performance. Both methods, however, do not exploit the prior information about question difficulty. It often results in unnecessary repeated sampling for easy questions that could be accurately answered with just one attempt, wasting resources. To tackle this problem, we propose Difficulty-Adaptive Self-Consistency (DSC), which leverages the difficulty information of batch queries from both prior and posterior perspectives to adaptively allocate inference resources, further reducing the overall cost of SC. To demonstrate the effectiveness of DSC, we conduct extensive experiments on three popular categories of reasoning tasks: arithmetic, commonsense and symbolic reasoning on six benchmarks. The empirical results show that DSC consistently surpasses the strong baseline ASC and ESC in terms of costs by a significant margin, while attaining comparable performances.[1]

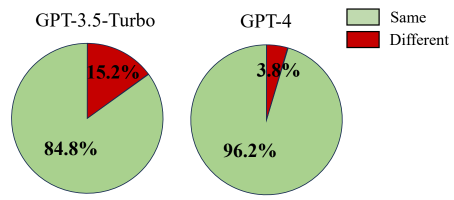


Figure 1: The proportion of identical inference results between CoT and SC on GSM8K with GPT-3.5-Turbo and GPT-4. We set sample size of SC as 40.

Model	Input		Output	
	Token	Cost	Token	Cost
GPT-3.5-Turbo	846.3	0.0004	163.7	0.0002
GPT-4	846.3	0.0254	142.1	0.0085

Table 1: Average tokens and cost ($) statistics of input and output for GPT-3.5-Turbo and GPT-4 on GSM8K. The cost is calculated according to https://openai.com/pricing. Given that the input for reasoning tasks usually involves several demonstrations (leading to lengthy contexts), the cost of input cannot be overlooked.

1 Introduction

Large language models (LLMs) have exhibited strong reasoning capabilities (Bubeck et al., 2023), especially with chain-of-thought (CoT) prompting (Wei et al., 2022b). Based on this, Wang et al. (2022) introduced a simple decoding strategy called self-consistency (SC) to further improve reasoning performance, leveraging the fact that challenging reasoning tasks typically require more reasoning paths to arrive at the correct answer. In contrast to the standard chain-of-thought prompting which only generates the greedy one, this method samples multiple reasoning paths according to a preset sample size, and then derives the final answer through majority-voting-based scheme.

Despite generally leading to improvements, SC introduces a significant overhead proportional to the number of sampled outputs. As LLMs continue to grow in size and complexity, the sampling time and computational costs associated with majority voting become increasingly challenging. Recently, some works seek to reduce the cost of SC by dynamically adjusting the number of samples based on the posterior distribution of pre-samples. ASC (Aggarwal et al., 2023) samples one by one, and

[†]Corresponding author.
[1]Our code and data have been released on https://github.com/WangXinglin/DSC.

Findings of the Association for Computational Linguistics: NAACL 2025, pages 6919–6932
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

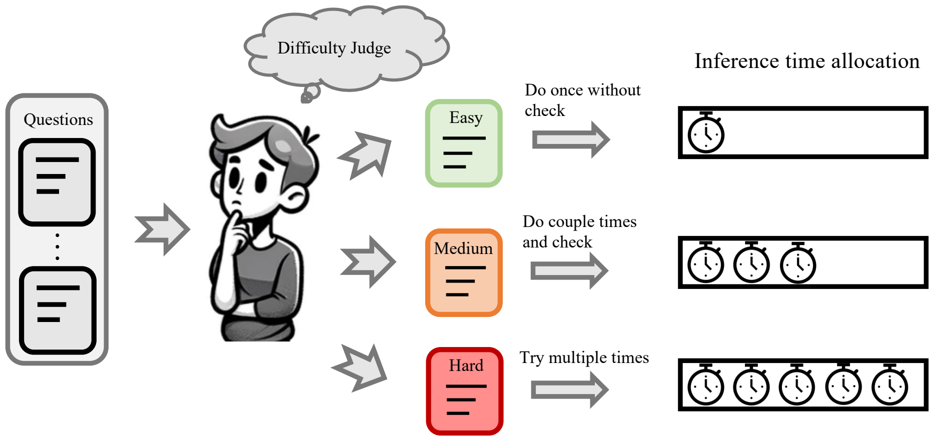


Figure 2: Leveraging their prior knowledge, humans assess the difficulty level of a problem before solving it, and allocate appropriate time for its resolution based on the difficulty.

stops sampling when the existing sample answers have established a clear majority as judged by a lightweight stopping criterion. Alternatively, ESC (Li et al., 2023) divides the large preset sample size into several sequential small windows, and stop sampling when answers within a window are all the same.

However, both approaches still suffer from the following two shortcomings: (1) Both require a certain amount of pre-sampling for all problems, which still results in redundant waste. As shown in Figure 1, there is a high overlap of inference results between SC and CoT, which suggests that only a small portion of questions (3.8% for GPT-4) benefit from SC. Generating multiple samples for the remaining questions compared to CoT (only sampling once) results in a significant waste of costs. Although ASC and ESC somewhat reduce the ineffective cost of SC by decreasing average sample size, there remains redundant sampling for simple problems[2]. (2) Multiple re-samples bring additional significant input costs. Both ASC and ESC focus solely on reducing the number of outputs, without considering the extra input costs brought by multiple re-samples (Table 1). Consequently, for problems requiring multiple sampling times, ASC and ESC will introduce substantial extra input costs, sometimes outweighing the savings achieved through output sampling reduction (Table 3).

To alleviate these issues, we take inspiration from the strategies humans employ when solving reasoning problems within a limited total time. As illustrated in Figure 2, humans pre-assess the difficulty of the problems before reasoning, and adaptively allocating problem-solving time based on the accessed difficulty. In light of this, we propose Difficulty-Adaptive Self-Consistency (DSC), a novel method which leverages the difficulty information from both prior and posterior perspectives to further reduce the inference computational cost of SC. As illustrated in Figure 3, DSC consists of three steps. Firstly, we propose Difficulty Ranking algorithm which utilizes the LLM itself to rank the difficulty of the given problem set. While using LLM for ranking will incur additional costs, later experiments will demonstrate that the prior difficulty information gained from this process can lead to a greater reduction in overall inference costs. Based on this, we propose a Problem Partition strategy that divides the problem set into two parts, easy and hard, using the information regarding difficulty rankings. For problems belonging to easy part, a single CoT sampling is performed, which saves cost without sacrificing performance. Lastly, we propose Sample Size Pre-Allocation algorithm to predict the sample size needed for problems belonging to hard part, which reduce the number of re-samples and save the cost of inputs.

We evaluate DSC on a wide range of arithmetic, commonsense and symbolic reasoning tasks over GPT-3.5-Turbo and GPT-4 models. The empirical results demonstrate that DSC outperforms the existing strong baselines ASC and ESC by a significant margin on six popular benchmarks, while attaining comparable performances. Further experiments confirm the effectiveness of the proposed three-step algorithms: Difficulty Ranking, Problem Partition, and Sample Size Pre-Allocation.

[2]At least 3 samples per problem for ASC and 4 for ESC.

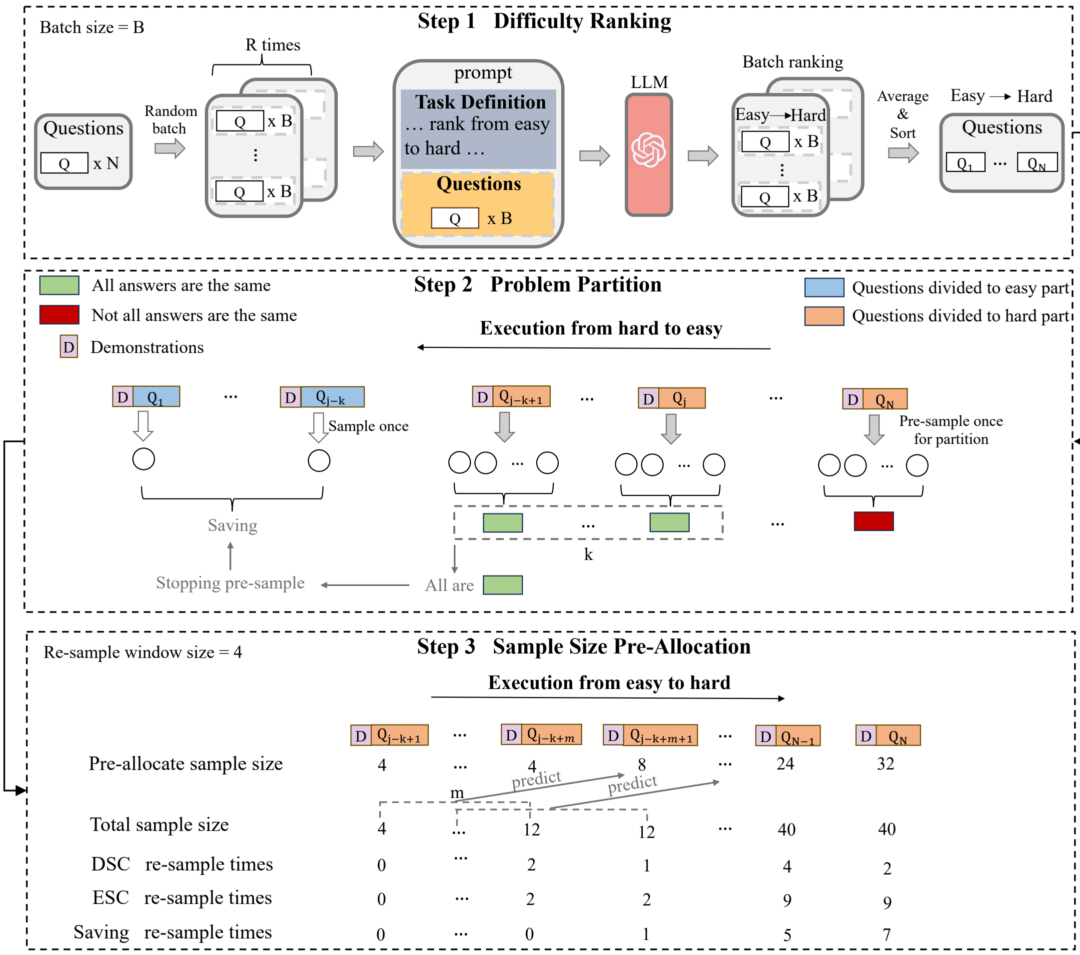


Figure 3: Overall workflow of proposed Difficulty-Adaptive Self-Consistency. DSC first ranks problem difficulty using LLM itself (step 1), then partitions problems into easy and hard to save sampling cost for easy ones (step 2), and finally pre-allocates sample sizes to reduce resampling costs for hard problems (step 3).

In summary, this work includes the following key contributions:

- We analyzed two common issues present in existing SC variants, ESC and ASC, designed for cost-efficient decoding.
- We proposed Difficulty-Adaptive Self-Consistency, a novel method consisting of three steps to fully utilize the difficulty information of given problems so as to reduce the inference computational cost.
- We conducted extensive experiments on six popular benchmarks and validated the effectiveness and generalizability of our proposed method.
- To the best of our knowledge, we are the first to propose dynamically allocating sampling resources based on query difficulty.

2 Methodology

The core idea behind DSC is to fully utilize the difficulty information of given problems, allocating computational resources based on their respective levels of difficulty. The overall workflow of DSC is presented in Figure 3, including three steps: Difficulty Ranking, Problem Partition and Sample Size Pre-Allocation.

2.1 Difficulty Ranking

As problems usually do not carry difficulty label themselves, we seek to utilize the powerful comparative and ranking capabilities of LLM to obtain the difficulty rank of the given problem set. Considering the limited context window size of LLM and its *lost in the middle* (Liu et al., 2024b) issue in understanding long contexts, we randomly divide the problem set of N into batches of size B ($B << N$) and let LLM rank the difficulty of the

Algorithm 1 Difficulty Ranking.

Require: Questions $Q^{1:N}$, LLM $\mathcal{M}$, Ranking prompt P, random split rounds R, Batch size B
Ensure: Questions sorted by difficulty $\bar{Q}^{1:N}$

$D_{all} \leftarrow \{i : [\,] \text{ for } i \in [1, N]\}$
for $r \leftarrow 1, R$ **do**
    Randomly divide $Q^{1:N}$ into batches $b_r^{1:L}$, $L = \lceil \frac{N}{B} \rceil$
    for $i \leftarrow 1, N$ **do**
        $D_{current} \leftarrow \varnothing$
        for $j \leftarrow 1, L$ **do**:
            $D_{current} \leftarrow D_{current}$.Append($\mathcal{M}(P, b_r^j)$)
        end for
        $D_{all} \leftarrow D_{all}$.Merge($D_{current}$)
    end for
end for
$\bar{Q}^{1:N}$ ←Sort(Average(D_{all}))

Algorithm 2 Problem Partition.

Require: Questions from **hard to easy** $Q^{1:N}$, LLM $\mathcal{M}$, Demonstrations D, Pre-sample size p, Judge window k
Ensure: Anchor point A, Easy part questions Q_{Easy}, Hard part questions Q_{Hard}

$E_{all} \leftarrow [\,]; a \leftarrow 1$
for $i \leftarrow 1, N$ **do**
    $S_{current} \leftarrow \mathcal{M}(Q^i, D, p)$
    $E_{all} \leftarrow E_{all}$.Append($Entropy(S_{current})$)
    if $i > k$ and $Sum(E_{all}[i-k : i-1]) = 0$ **then**
        $A \leftarrow i$
        Break
    end if
end for
$Q_{Hard} \leftarrow Q^{1:A}$; $Q_{Easy} \leftarrow Q^{A+1:N}$

problems in each batch. Since a single random split only allows us to obtain the difficulty ranking of each problem within the corresponding batch, we perform R times random batch splitting, and sort the average difficulty rankings of each problem in different batches to obtain the difficulty ranking of entire problem set. Algorithm 1 illustrates the specific implementation of difficulty ranking. Specifically, D_{all} is a dictionary that stores the difficulty levels of all N questions. Its keys represent the indices of the questions, and its values are lists used to store the difficulties of the questions across different batches. Merge indicates appending the difficulty of question i in the current batch r to $D_{all}[i]$ (List). Average refers to calculating the average value of $D_{all}[i]$, so as to obtain difficulty value of question i across the entire problem set. Given the additional costs of using LLM for ranking, we carefully design the ranking prompts[3] to minimize these costs, and later experiments will prove that the prior difficulty information gained can significantly reduce overall inference costs.

2.2 Problem Partition

After obtaining the problem set with difficulty ranking, we aim to find which part of the problems only require LLM to perform CoT sampling once. A simple and intuitive idea is that when LLM is very confident about a number of continuous problems sorted by difficulty (the results of all pre-samples are the same), it can be inferred that the problems easier than these problems only need one CoT sampling. Guided by this, we design the Problem Partition algorithm, illustrated in Algorithm 2. Specifically, we pre-sample the problem set sorted by difficulty from hard to easy, and store the entropy of pre-sampling results of each problem ($S_{current}$) in a list. We stop pre-sampling when the latest k items in the list are all zero. The remaining problems without pre-sampling are then divided into the easy part, and a single CoT sampling is performed for each.

2.3 Sample Size Pre-Allocation

After performing CoT sampling on the easy part problems, we seek to predict and allocate the sample size for the problems belonging to hard part, so as to mitigate the substantial cost brought by multiple re-samples. Considering that the required sample size for each problem should be similar to those needed for problems of comparable difficulty, we predict the sample size of the current problem based on the total sample size of the nearest m easier[4] problems. Algorithm 3 shows the workflow of Sample Size Pre-Allocation. Specifically, we sample questions belonging to the hard part from easy to difficult. For the current question, we predict its pre-allocation sample size PA based on the average total sample size of its previous m questions. Then, we judge the distribution of the current samples based on the stopping criteria C[5]. When the criteria is not met, we re-sample based on the expansion window e, until the sampling distribution meets the criteria for stopping sampling or the number of samples reaches the max sample size L. After sampling for the current question ends, we add its samples and total sample size to S_{all} and N_{all} lists respectively, in order to pre-allocate the sample size for the next question.

[3] See Appendix A.1 for corresponding prompts.

[4] As the pre-allocated sample size could be larger than the required sample size (causing higher output costs), we use the sample size of easier rather than harder neighboring problems for prediction.

[5] Following ASC, we use Dirichlet Stopping Criteria as default. See Appendix A.8 for more details.

Algorithm 3 Sample Size Pre-Allocation.

Require: Hard part questions from **easy to hard** $Q^{1:A}$, LLM $\mathcal{M}$, Demonstrations D, Stopping Criteria C, Max sample size L, Extend window size e, Prediction window size m
Ensure: Sampling of $Q^{1:A}$ based on pre-allocation S_{all}

```
1:  S_all ← {i : [ ] for i ∈ [1, A]}; N_all ← [ ]
2:  for i ← 1, A do
3:      S_current ← ∅; PA ← 0; N_current ← 0
4:      if i > m then
5:          PA ← Average(N_all[i − m : i − 1])
6:          S_current ← S_current.Append(M(Q^i, D, PA))
7:          N_current ← N_current + PA
8:      end if
9:      while N_current < L and NOT C(S_current) do
10:         S_current ← S_current.Append(M(Q^i, D, e))
11:         N_current ← N_current + e
12:     end while
13:     S_all ← S_all.Merge(S_current)
14:     N_all ← N_all.Append(N_current)
15: end for
```

3 Experiments

3.1 Experimental Setup

3.1.1 Benchmarks

We evaluate the proposed DSC on six benchmark datasets from three categories of reasoning tasks: For arithmetic reasoning, we consider MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021). MultiArith (Roy and Roth, 2015), SVAMP (Patel et al., 2021), AddSub (Hosseini et al., 2014) and ASDiv (Miao et al., 2020) are not chosen in this paper because they are relatively simple. For commonsense reasoning, CommonsenseQA (Talmor et al., 2019) and StrategyQA (Geva et al., 2021) are used. For symbolic reasoning, we use Last Letter Concatenation and Coin Flip from Wei et al. (2022a). The data version is from Kojima et al. (2022).

3.1.2 Baselines

We compare DSC to the following self-consistency methods: (1) SC (Wang et al., 2023) is the standard self-consistency which samples multiple reasoning paths and then derives the final answer through majority-voting; (2) ASC (Aggarwal et al., 2023) samples one by one, and stops sampling when the existing samples meet a designed stopping criteria, which measures the LLM's confidence in its current samples; (3) ESC (Li et al., 2023) proposes using small window detection to save cost, which divides the large preset sample size into several sequential small windows, and stops sampling when answers within a window are all the same. Specifically, ASC and ESC are two strong baselines for cost-efficient self-consistency methods, we reproduce both methods according to their original implementation.

3.1.3 Implementation details

We perform experiments using two powerful LLMs: GPT-4 (OpenAI, 2023) and GPT-3.5-Turbo[6]. We calculate the cost based on the price of the API we use. All experiments are conducted in the few-shot setting without training or fine-tuning the language models. To ensure a fair comparison, we use the same prompts as Wei et al. (2022a). Following Li et al. (2023), The sampling temperature T for MATH is 0.5 while for other datasets is 0.7. For difficulty ranking, we use CoT sampling. We set the default parameters as follows: batch size B as 8 and random split rounds R as 5 for Difficulty Ranking; pre-sample size p as 4 and judge window size k as 32 for Problem Partition; extend window size e as 4 and prediction window size m as 16 for Sample Size Pre-Allocation; max sample size L as 40 for all baselines. All experiments are repeated 100 times and the average performance is reported. Unless otherwise specified, the reported cost of DSC includes the cost brought by all three substeps.

3.2 Main Results

DSC significantly reduces costs while barely affecting performance. Table 2 summarizes the cost and accuracy of SC, ASC, ESC, and proposed DSC for each dataset. We show that DSC consistently outperforms all baselines on cost by a significant margin across all datasets, while barely affecting performance. Specifically, DSC reduces the cost on GPT-4 by an average of 65.29% and on GPT-3.5-Turbo by 56.04% compared to SC. In comparison to the strongest baseline method ESC, DSC reduces the cost on GPT-4 by an average of 24.81% and on GPT-3.5-Turbo by 21.86%, which demonstrates the effectiveness of DSC. Furthermore, the above results show that the prior difficulty information gained from difficulty ranking can help greatly lower overall inference costs, even though it first introduces some additional costs.

DSC is scalable across various max sampling size. We conduct experiments with various max sampling size to validate the scalability of ESC. Table 3 shows the performance across different maximum sampling sizes. First we can see the

[6]We use the "2023-05-15" version of API for both.

Model	Method	MATH		GSM8K		CSQA		SQA		Letter		Coinflip	
		Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc
GPT-4	SC	0.4220	58.52	0.3509	95.81	0.1280	85.82	0.1493	81.89	0.1691	94.51	0.1528	100
	ASC	0.5726	58.48	0.1712	95.79	0.1639	85.82	0.1323	81.89	0.0602	94.50	0.0657	100
	ESC	0.4062	58.49	0.1014	95.80	0.0767	85.81	0.0647	81.89	0.0375	94.51	0.0338	100
	DSC	**0.3142**	58.51	**0.0699**	95.79	**0.0598**	85.81	**0.0516**	81.88	**0.0259**	94.50	**0.0264**	100
GPT-3.5-Turbo	SC	0.0181	49.43	0.0094	83.22	0.0030	76.80	0.0043	71.47	0.0043	80.44	0.0037	76.60
	ASC	0.0193	49.39	0.0064	83.18	0.0034	76.78	0.0032	71.47	0.0018	80.43	0.0034	76.59
	ESC	0.0172	49.41	0.0048	83.19	0.0017	76.79	0.0019	71.47	0.0014	80.43	0.0020	76.59
	DSC	**0.0148**	49.42	**0.0036**	83.19	**0.0012**	76.79	**0.0015**	71.46	**0.0011**	80.42	**0.0016**	76.60

Table 2: Accuracy (%) and cost ($) across six reasoning benchmarks. The best performance of cost is highlighted in **bold**.

Model	Method	16		24		32		40		48	
		Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc	Cost	Acc
GPT-4	SC	0.1791 -	57.34	0.2602 -	57.95	0.3411 -	58.27	0.4220 -	58.52	0.5030 -	58.67
	ASC	0.3038 ↓	57.33	0.4038 ↓	57.93	0.4923 ↓	58.23	0.5726 ↓	58.48	0.6470 ↓	58.62
	ESC	0.1870 ↓	57.34	0.2631 ↓	57.94	0.3361 ↑	58.25	0.4062 ↑	58.49	0.4723 ↑	58.64
	DSC	**0.1659** ↑	57.34	**0.2192** ↑	57.94	**0.2680** ↑	58.25	**0.3142** ↑	58.51	**0.3585** ↑	58.66
GPT-3.5-Turbo	SC	0.0074 -	47.98	0.0110 -	48.71	0.0145 -	49.13	0.0181 -	49.43	0.0216 -	49.67
	ASC	0.0095 ↓	47.97	0.0131 ↓	48.69	0.0163 ↓	49.10	0.0193 ↓	49.39	0.0220 ↓	49.62
	ESC	0.0075 ↓	47.98	0.0108 ↑	48.70	0.0140 ↑	49.12	0.0172 ↑	49.41	0.0203 ↑	49.65
	DSC	**0.0069** ↑	47.98	**0.0097** ↑	48.71	**0.0123** ↑	49.13	**0.0148** ↑	49.42	**0.0171** ↑	49.66

Table 3: Reasoning accuracy (%) and corresponding cost ($) with various max sampling size on MATH. We mark the performance of cost poorer than SC with ↓ and that better than SC with ↑. The best performance of cost is highlighted in **bold**.

Method	Input tokens	Output tokens	Cost	Acc
SC	846	6045	0.3881	73.19
ASC	15982	3265	0.6754	73.14
ESC	4458	4415	0.3986	73.18
DSC	1617	3857	0.2799	73.18

Table 4: Accuracy (%) and cost ($) on GSM8K with small open-source language model Mistral-7B-Instruct-v0.3.

performance of SC continuously improves as max sampling size L increases, which is consistent with the results in (Wang et al., 2023). On this basis, ESC can significantly save costs while maintaining performance for different L. On the contrary, due to the substantial additional input cost brought by multiple re-samples, ASC and ESC result in higher cost than SC when the cost savings of output tokens are limited.

DSC Maintains Generalizability on Small Open-source Language Model. To further explore the effectiveness of DSC on small open-source models, we conduct experiments on the open-source model Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) using GSM8K dataset. As shown in Table 4, we convert the token cost into price according to that of GPT-4 for a simpler comparison and keep the other settings completely consistent with the main experiment[7]. The experimental results indicate that DSC has the potential to work effectively on smaller models.

3.3 Analysis of Three Sub-steps

To further validate the effectiveness of proposed three sub-steps, including Difficulty Ranking, Problem Partition, and Sample Size Pre-Allocation, we conduct experimental analyses on each of them respectively.

3.3.1 Difficulty Ranking

Difficulty Ranking exhibits a good correlation with humans. As shown in Table 5, we calculate Spearman, Pearson, and Kendall correlation coefficients on seven subsets of MATH benchmark. Overall, GPT-4 demonstrates a high consistency with human labels, while the weaker GPT-3.5-Turbo shows a moderate consistency. Specifically, both GPT-4 and GPT-3.5-Turbo rank weakly on Geometry difficulty, which may be due to the fact that text LLMs cannot intuitively assess the difficulty of geometry problems through visual information

[7]Please note that for DSC, the cost of all three sub-steps are included.

Model	Metric	Geometry	Counting & Probability	Prealgebra	Intermediate Algebra	Number Theory	Precalculus	Algebra	Avg
GPT-4	Spearman	0.540	0.578	0.565	0.654	0.654	0.684	0.691	0.624
	Pearson	0.535	0.585	0.567	0.660	0.654	0.691	0.695	0.627
	Kendall	0.417	0.448	0.440	0.519	0.520	0.547	0.547	0.491
GPT-3.5-Turbo	Spearman	0.415	0.307	0.362	0.484	0.445	0.497	0.573	0.440
	Pearson	0.415	0.314	0.370	0.495	0.452	0.514	0.578	0.448
	Kendall	0.318	0.233	0.279	0.375	0.344	0.383	0.446	0.340

Table 5: Performance of Difficulty Ranking on MATH dataset.

Model	Method	MATH		GSM8K		CSQA		SQA		Letter		Coinflip	
		Input	Output	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
GPT-4	SC	576	6746	846	5426	726	1770	611	2184	302	2667	428	2332
	ASC	11100	3994	4266	719	4805	327	3726	351	1397	308	1722	234
	ESC	3574	4983	1423	982	1644	455	1228	465	454	401	540	294
	DSC	834	4606	975	495	978	271	822	310	314	113	428	58
	w/o step 2	847	4701	1035	755	1081	370	873	381	353	313	431	235
	w/o step 3	2896	4142	1033	479	1068	253	903	290	328	108	428	58
GPT-3.5-Turbo	SC	576	11851	846	6005	726	1749	611	2644	302	2735	428	2316
	ASC	13421	8380	8075	1577	5633	365	4698	535	2150	487	4932	624
	ESC	4223	10045	2889	2273	1946	506	1599	731	742	673	1613	819
	DSC	793	9456	1410	1809	1140	376	985	591	470	535	704	736
	w/o step 2	799	9610	1426	1879	1172	401	997	605	482	584	707	761
	w/o step 3	3485	8634	2144	1679	1467	362	1238	560	568	513	1290	648

Table 6: Ablation study of Problem Partition (step 2) and Sample Size Pre-Allocation (step 3). We count input and output tokens across six reasoning benchmarks. To simplify the comparison, the tokens produced by Difficulty Ranking are not counted here.

Task	MATH	GSM8K	CSQA	SQA	Letter	Coinflip
Ranking	403	364	261	130	161	308
Reasoning	576	846	726	611	302	428

Table 7: Comparison of average input tokens for each question brought by Difficulty Ranking and few-shot reasoning across six reasoning datasets.

like humans can.

Difficulty Ranking produces an acceptable cost. As we instruct LLM to produce extremely concise outputs for Difficulty Ranking, the cost of output tokens brought by Difficulty ranking is very low[8]. Table 7 gives comparison of input tokens brought by Difficulty Ranking and few-shot reasoning across six datasets. We show that the average input cost for each problem's difficulty ranking is less than the average single input cost for its few-shot reasoning. Considering that ESC and ASC require multiple re-samples for reasoning (multiple few-shot inputs), this additional input cost of Difficulty Ranking is totally acceptable.

[8]With an average consumption of 15 tokens for each question.

3.3.2 Problem Partition

Problem Partition proves to be effective across all datasets. As shown in Table 6, we conduct an ablation study on Problem Partition (step 2). We show that the removal of Problem Partition results in an increase in both input and output tokens of DSC across all six datasets, which validates the effectiveness and generalizability of Problem Partition. Furthermore, we find that the removal of Problem Partition has a small impact on output tokens when it comes to MATH dataset, which could be due to the fact that for datasets with overall higher difficulty, the easy part problem (only require one CoT sample) for LLM is fewer.

Judge window size is essential for the accuracy of DSC. Regarding the intuitive idea proposed in Section 2.2, a straightforward question is how large the judge window size k needs to be to ensure the accuracy on simpler problems with single CoT sampling. To answer this, we conduct an experiment of DSC under different judge window size k with GPT-4 on GSM8K. As shown in Figure 4, the accuracy increases with the rise of k until it reaches 32. Meanwhile, the cost of DSC also increases with the enlargement of k, as the easy part divided by

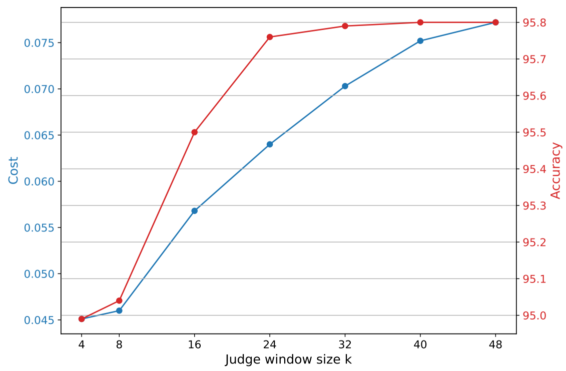


Figure 4: Cost ($) and accuracy (%) of DSC under different judge window size k with GPT-4 on GSM8K.

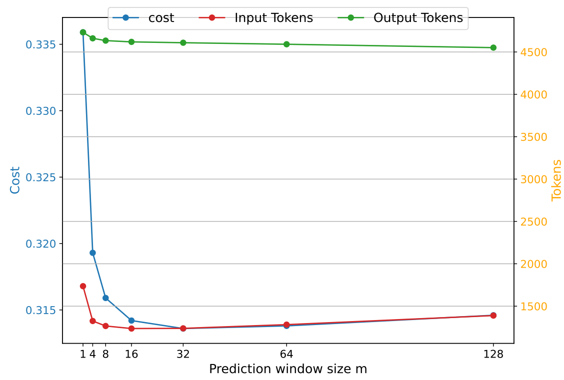


Figure 5: Tokens of input and output and the corresponding cost ($) of DSC under different prediction window size m with GPT-4 on MATH.

the problem partition decreases with its increase. Therefore, it is a good choice to set a smaller judge window size while ensuring accuracy.[9]

3.3.3 Sample Size Pre-Allocation

Sample Size Pre-Allocation significantly alleviates the substantial input token costs brought by multiple re-samples. We conduct an ablation study of proposed Sample Size Pre-Allocation (step 3) to validate its effectiveness. As shown in Table 6, with the removal of Sample Size Pre-Allocation, the count of input tokens on DSC rose on all datasets on GPT-3.5-Turbo and GPT-4 (with the exception of GPT-4 on Coinflip), validating the effectiveness and generalizability of Sample Size Pre-Allocation. Moreover, we notice that the input tokens of ASC and ESC far exceed that of SC, which once again confirms that multiple re-samples will bring a large amount of additional overhead. Meanwhile, DSC significantly reduced the number of input tokens through Pre-Allocation, keeping it comparable to SC. In addition, we find that Sample Size Pre-Allocation leads to a slight increase in output tokens (due to the predicted sample size of some questions exceeding the actual requirement). Overall, we believe it is a good choice to trade a small number of output tokens for a significant saving in input tokens.

Prediction window size is the key to trade output tokens for more savings in input tokens. Given that Sample Size Pre-Allocation can trade output tokens for savings in input tokens, a natural question is when the max total savings can be achieved. As shown in Figure 5, we count the tokens of input and output and the corresponding cost of DSC under different prediction window size m. The experimental results show that the cost of DSC decreases rapidly first and then slighly increases with the increase of m, and the input tokens also follow a similar trend. As m rises, simpler questions predict larger sample sizes for the current query. However, when m becomes sufficiently large, the predicted sample size falls below the actual value, leading to more re-sampling. In contrast, output tokens gradually decrease because, as m increases, the predicted sample size continues to decline, reducing redundancy in output token sampling. In summary, it is suggested to select m within a relatively moderate range (e.g. from 8 to 32).

[9]We select 32 as the default value for k.

4 Related Work

Chain-of-thought Reasoning Chain-of-thought prompting has been proven to be an effective method of solving complex reasoning problems (Wei et al., 2022a). By following the pattern of gradually solving sub-problems, few-shot CoT (Fu et al., 2023) are capable of stimulating LLM reasoning abilities. On this basis, Least-to-most prompting (Zhou et al., 2023) suggest explicitly splitting the problem and solving them step by step. Zheng et al. (2023) reach the final answer by iteratively generating answers and using the previously generated answers as context hints.

Self-Consistency Self-consistency (Wang et al., 2023) refers to a simple decoding strategy for further improving reasoning performance, leveraging the fact that complex reasoning tasks typically allow for more than one correct reasoning path. Li et al. (2024) assign appropriate weights for answer aggregation to achieve adaptive self-consistency.

Jain et al. (2023) and Wang et al. (2024) extend it for open-ended generation tasks like code generation and text summarization. However, they require multiple sampling with the pre-set size, which will incur much more computation cost. To achieve cost-efficient self-consistency, Aggarwal et al. (2023) introduce an adaptive stopping criterion based on the amount of agreement between the samples so far. Li et al. (2023) divide the large preset sample size into several sequential small windows, and stop sampling when answers within a window are all the same.

Difficulty-adaptive Test-Time Scaling Test-time scaling (Wu et al., 2024a; Snell et al., 2024; Brown et al., 2024) has been widely explored as an effective approach to improving model performance. However, conventional test-time scaling applies a uniform amount of computation to all queries, regardless of their difficulty, resulting in unnecessary computational overhead. To mitigate this inefficiency, recent work has proposed difficulty-adaptive scaling strategies that dynamically allocate computational resources based on query difficulty. Snell et al. (2024) validate the feasibility of optimizing test-time scaling compute based on query difficulty. To address excessive refinement in self-refinement, Chen et al. (2024) classify problems as easy or hard, solving easy ones with coarse aggregation and hard ones with fine-grained, iterative multi-agent refinement. Manvi et al. (2024) introduce capability-aware and mid-generation self-evaluations to enable difficulty-aware adaptive test-time computation by dynamically allocating compute based on query difficulty and model capability. Similarly, Damani et al. (2024) train a lightweight model to estimate query difficulty and allocate extra computation to those queries that would benefit most from a more computationally intensive decoding procedure.

5 Conclusion

In this work, we propose a novel cost-efficient decoding method called Difficulty-Adaptive Self-Consistency (DSC), which fully leverages the difficulty information of given problem set to allocate computational resources based on their respective levels of difficulty, so as to alleviate issues of redundant sampling on simple questions and multiple re-sampling limitations that current cost-efficient self-consistency methods face. Extensive experiments show that DSC consistently surpasses two strong cost-effcient self-consistency baselines ASC and ESC by a significant margin on six popular benchmarks, while attaining comparable accuracy. Additional experiments confirm the effectiveness of the proposed three-step algorithms.

Limitations

Despite the remarkable efficiency gain on variety of reasoning tasks, the current implementation of DSC still suffers from the following two limitations:

- As the performance of difficulty ranking declines when handling mixed-type problems (Table 10), DSC may encounter challenges when applied in scenarios requiring real-time reasoning for mixed-type problems. Exploring further classification of problems by type or optimizing the current Difficulty Ranking algorithm could enhance its applicability.
- Given that DSC demands an awareness of the test set to rank samples based on their difficulty, its use could be restricted in scenarios where a single user is only permitted one input at a time. Nevertheless, the application scenarios of DSC include but are not limited to: (1) The server end (such as OpenAI company, etc) simultaneously receives a large number of query requests from users. (2) A user possesses a batch of data that requires a one-time inference.

Ethics Statement

All of the datasets used in this study were publicly available, and no annotators were employed for our data collection. We confirm that the datasets we used did not contain any harmful content and was consistent with their intended use (research). We have cited the datasets and relevant works used in this study.

Acknowledgements

This work is supported by the Beijing Natural Science Foundation, China (Nos. 4222037, L181010).

References

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie CY Chan, Biao Zhang, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. In *ICML 2024 Workshop on In-Context Learning*.

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.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. 2024. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. *arXiv preprint arXiv:2409.12147*.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 792–810.

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. *CoRR*, abs/2110.14168.

Peizhuang Cong, Chen Qizhi, Haochen Zhao, and Tong Yang. 2025. Baton: Enhancing batch-wise inference efficiency for large language models via dynamic re-batching. In *THE WEB CONFERENCE 2025*.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. 2024. Learning how hard to think: Input-adaptive allocation of lm computation. *arXiv preprint arXiv:2410.04707*.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning. *CoRR*, abs/2301.12726.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? A question answering benchmark with implicit reasoning strategies. *Trans. Assoc. Comput. Linguistics*, 9:346–361.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 523–533. ACL.

Siddhartha Jain, Xiaofei Ma, Anoop Deoras, and Bing Xiang. 2023. Self-consistency for open-ended generations. *CoRR*, abs/2307.06857.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *NeurIPS*.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li. 2024. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18591–18599.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2023. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *The Twelfth International Conference on Learning Representations*.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. 2024. Batchprompt: Accomplish more with less. In *The Twelfth International Conference on Learning Representations*.

Jiayi Liu, Tinghan Yang, and Jennifer Neville. 2024a. Cliqueparcel: An approach for batching llm prompts that jointly optimizes efficiency and faithfulness. *arXiv preprint arXiv:2402.14833*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024b. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 11:157–173.

Rohin Manvi, Anikait Singh, and Stefano Ermon. 2024. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 975–984. Association for Computational Linguistics.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2080–2094. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752. The Association for Computational Linguistics.

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

Guijin Son, SangWon Baek, Sangdae Nam, Ilgyun Jeong, and Seungone Kim. 2024. Multi-task inference: Can large language models follow multiple instructions at once? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5606–5627, Bangkok, Thailand. Association for Computational Linguistics.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4149–4158. Association for Computational Linguistics.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Boyuan Pan, Heda Wang, Yao Hu, and Kan Li. 2024. Integrate the essence and eliminate the dross: Fine-grained self-consistency for free-form language generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11782–11794.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022a. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

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

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024a. An empirical analysis of compute-optimal inference for problem-solving with language models. *URL https://arxiv.org/abs/2408.00724*.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024b. Large language models can self-correct with key condition verification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12846–12867.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. *CoRR*, abs/2304.09797.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

A Appendix

A.1 Prompt for Difficulty Ranking

Your task is to rank the given questions from easy to hard based on their difficulty level. Questions to be evaluated:

Q1:{Question 1}

Q2:{Question 2}

...

Qn: {Question n}

The output format should be a comma-separated list containing the Qnumber of corresponding question. Do not give any explanation.

Difficulty Ranking result (from easy to hard):

A.2 Comparison of Inference Time Between Different Methods

Considering the inference time is important for real world scenarios, we calculate the inference time of different methods on the MATH test set (5000 questions) with GPT-4. As shown in Table 8, for DSC, the ranking time corresponds to the time produced by step 1, while sampling time corresponds to the time generated by step 2 and step 3. The results show that, compared to SC, both ASC and ESC require significantly more time due to the need for a large amount of resampling. DSC, on the other hand, effectively mitigates this issue by pre-allocating the sample size for questions.

A.3 Comparison with ESC under Different Window Size

To further demonstrate the effectiveness of DSC compared to ESC, we make a comparison between the performance of DSC and ESC under different window sizes on the MATH dataset using GPT-4, while maintaining the rest of the settings completely consistent with the main experiment. As shown in Table 9, when the window size is greater than or equal to 5, ESC can maintain its Accuracy and its cost increases as the window size enlarges. However, when the window size is less than 5, the Accuracy of ESC drops significantly due to excessively relaxed constraints. Overall, the performance of DSC consistently surpasses that of ESC by a large margin.

Time (hours)	Ranking Time	Sampling Time	Total Time
SC	0	11.75	11.75
ASC	0	221	221
ESC	0	57.26	57.26
DSC	2.26	17.08	19.34

Table 8: Inference time analysis of different methods on MATH using GPT-4.

Window	3	4	5	6	7	DSC
Acc	58.10	58.40	58.49	58.51	58.51	58.51
Cost	0.3505	0.3905	0.4062	0.4135	0.4173	0.3142

Table 9: Comparison of ESC under different window size and DSC on MATH with GPT-4.

A.4 Performance of Difficulty Ranking under Mixing Types of Questions

To explore whether Difficulty Ranking can effectively rank problems of different types in terms of difficulty, we compare the performance of Difficulty Ranking under mixed and unmixed problem types on MATH, as shown in Table 10. Specifically, for unmixed, we perform Difficulty Ranking on seven subsets separately; for mixed, we directly execute Difficulty Ranking on random shuffled MATH dataset, and calculate the correlation on the entire dataset for both. The results indicate that mixing different types of questions for LLM to rank can lead to a decrease in performance. This might be due to the challenge for LLM in adhering to the same assessment standards for various types of questions.[10]

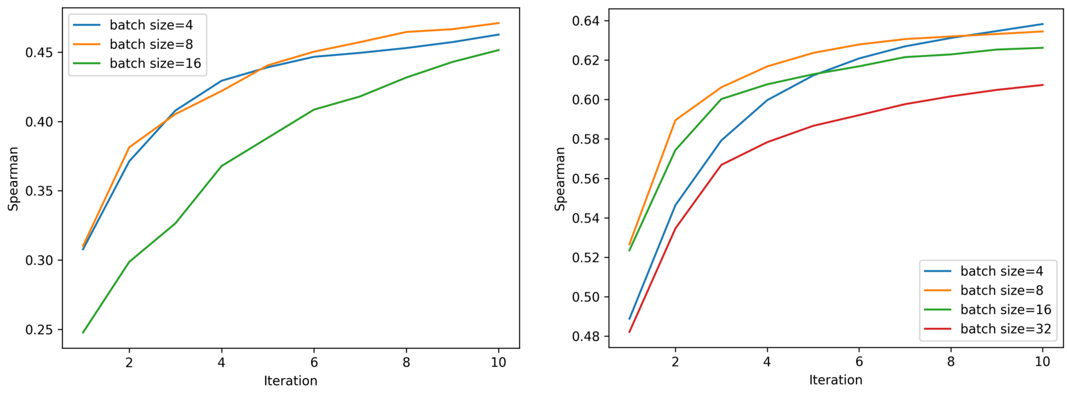
A.5 Hyperparameter Experiment of Difficulty Ranking

As shown in Figure 6, we conduct hyperparameter experiments on the proposed Difficulty Ranking algorithm. The experimental results indicate that a large batch size (≥ 16) leads to a decrease in LLM ranking performance, which is consistent with our expectations. For both GPT-3.5-Turbo and GPT-4, difficulty ranking approximately converges in the 5th round. Therefore, we choose 5 and 8 as the default values for iteration and batch size for Difficulty Ranking, respectively.

A.6 Necessity of Few-Shot Setting in DSC

Previous studies (Wei et al., 2022a; Wang et al., 2023) have experimentally demonstrated that compared to zero-shot (without demonstrations), few-

[10] It could also lead to inaccurate ranking for humans.



GPT-3.5-Turbo

GPT-4

Figure 6: Performance of Difficulty Ranking under different batch sizes and iterations on MATH with GPT-3.5-Turbo and GPT-4.

Model	Metric	Unmixed	Mixed
GPT-4	Spearman	0.6451	0.5148
	Pearson	0.6487	0.5197
	Kendall	0.5060	0.3931
GPT-3.5-Turbo	Spearman	0.4828	0.4070
	Pearson	0.4887	0.4145
	Kendall	0.3697	0.3073

Table 10: Performance comparison of Difficulty Ranking under mixed and unmixed problem types on MATH.

Setting	N				
	1	5	10	20	40
Zero-shot	27.01	31.44	33.84	35.21	35.93
Few-shot	33.08	38.54	41.39	43.01	43.94

Table 11: Accuracy Comparison of SC under Zero-Shot and Few-Shot Settings with Different Sample Sizes N on MATH500 Using GPT-3.5-Turbo

shot (with demonstrations) significantly improves the performance of LLMs on reasoning tasks. Therefore, we adopted the same few-shot setting as SC, ASC, and ESC. To further verify this, we test the Accuracy of SC on GPT-3.5-Turbo under both zero-shot and few-shot (8-shot as default) settings with different sample sizes N on MATH500.

The results (Table 11) show that removing demonstrations leads to a significant performance drop, validating the necessity of multiple demonstrations. As a matter of fact, during our research, we tried to dynamically allocate the number of demonstrations based on question difficulty to reduce input costs. However, we find that reducing the number of demonstrations significantly decreased performance, regardless of the difficulty of the questions.

Moreover, recent study (Agarwal et al., 2024) has demonstrated that a greater number of demonstrations generally leads to more performance improvements. For most tasks, the optimal number of demonstrations is often greater than 512, far exceeding the default setting of 8 in DSC, which results in much higher input costs. This further highlights the value of DSC.

A.7 Comparison of DSC with Self-Correction and Verifier-Based Methods

We consider that self-correct (Wu et al., 2024b), verifier-based methods (Wu et al., 2024a), and Self-consistency (Wang et al., 2023) can all be seen as approaches that aim to improve model performance by increasing computational effort, albeit through different implementation paths. In comparison, DSC takes a different direction by focusing on optimizing efficiency—reducing computational costs as much as possible while maintaining model performance. This distinction positions DSC as complementary to self-correct, verifier-based methods, and Self-consistency, with a unique focus on efficiency.

Furthermore, while DSC is implemented based on Self-consistency, its concept of dynamically allocating computational resources according to question difficulty could also be adapted to self-correct and verifier-based methods to enhance efficiency. For example, in the case of self-correct, this might involve assessing whether a query requires self-correction based on its difficulty and estimating the necessary degree of correction (e.g., the number of iterations). Similarly, for verifier-based methods, which already involve multiple sampling followed by verification, integrating DSC could potentially lead to more efficient inference processes.

A.8 Background of ESC and ASC

ESC ESC proposes extension window sampling, where the extension window is a fixed preset value w, Each time, w entries are sampled through the LLM and added to the sampling set. If the answers of the current w samples are the same or the preset maximum sampling value is reached, the sampling is stopped. The core idea of ESC is that if the model's one-time w sample answers are completely the same, it can be considered that it has a high confidence in this answer, and sampling can be stopped.

ASC ASC (Aggarwal et al., 2023) proposes the Dirichlet Stopping Criteria, where sampling is conducted one by one. After each sampling, the Dirichlet Stopping Criteria is used to determine whether all current samples meet a specific distribution. If they do, the sampling is stopped. If not, the one-by-one sampling continues until the preset maximum sampling value is reached. The Dirichlet Stopping Criteria is shown in Equation 1, where v represents the current set of samples, m is the number of v, C_{thresh} is the preset threshold (set to 0.95 for ASC in default), and p_1 is the probability of the most frequently occurring answer in the set v. The core idea of ASC is to measure the two answers with the highest probability in the current set of samples. If the difference between the probability of the answer with the highest probability and the second highest probability exceeds a threshold (C_{thresh}), it can be considered that the model is very confident in the answer with the highest probability, and sampling can be stopped. For further details, please refer to the original paper.

$$P\left(p_1 > \max_{i=2}^{m} p_i \mid v\right) > C_{\text{thresh}} \tag{1}$$

A.9 Application Scenarios of DSC

As discussed in the Limitations section, DSC could be constrained in scenarios where a single user is restricted to one input at a time. However, it is widely applicable to common scenarios including: (1) servers (e.g., OpenAI or similar providers) managing a large number of simultaneous query requests from different users (e.g., ChatGPT receives requests from multiple users to solve multiple math problems simultaneously), and (2) cases where a user perform inference on a batch of data (E.g., Submitting multiple requests at once to efficiently receive feedback).

For these application scenarios, Recently, an increasing number of studies have focused on performing inference on large volumes of samples with LLMs (Cheng et al., 2023; Lin et al., 2024; Son et al., 2024; Liu et al., 2024a; Cong et al., 2025), which is computationally and financially costly in industrial and real-world applications. Therefore, the application scenarios of DSC have become a key focus in current research, attracting extensive attention from the academic and industrial communities, and we believe DSC's potential applications and value will be more and more substantial in the future.