Let's Sample Step by Step: Adaptive-Consistency for Efficient Reasoning with LLMs

Anonymous ACL submission

Abstract

A popular approach for improving the correctness of output from large language models (LLMs) is Self-Consistency – poll the LLM multiple times and output the most frequent solution. Existing Self-Consistency techniques always draw a *constant* number of samples per question, where a better approach will be to non-uniformly distribute the available budget, based on the amount of agreement in the samples drawn so far. In response, we introduce Adaptive-Consistency, a cost-efficient, model-agnostic technique that *dynamically* adjusts the number of samples per question using a lightweight stopping criterion. Our experiments over 13 datasets and two LLMs demonstrate that Adaptive-Consistency reduces sample budget by up to 6.0 times with an average accuracy drop of less than 0.1%.[1]

1 Introduction

The increasing adoption of large language models (LLMs) across various tasks, such as text generation and reasoning (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2022a; Mishra et al., 2022), mathematical reasoning (Lewkowycz et al., 2022; Gao et al., 2022), and code generation (Li et al., 2022; Madaan et al., 2023), has underscored the importance of improving the correctness of their outputs. A popular method for achieving this goal is *Self-Consistency* (Wang et al., 2022b), a majority voting technique where multiple output samples are generated for a given input, and the final decision is based on the most frequently occurring output among the samples.

Current Self-Consistency methods typically employ a fixed budget approach, wherein a predetermined number of samples (e.g., 40) are drawn to make a decision. However, as LLMs continue to grow in size and complexity, the sampling time and computational costs associated with majority voting become increasingly challenging. This challenge is particularly evident in high-stakes applications like competition-level code generation (Li et al., 2022), where generating a large number of programs, sometimes up to a million, is essential for maximizing performance.

To address this challenge, we introduce *Adaptive-Consistency*, a cost-efficient, model-agnostic majority voting technique. Adaptive-Consistency employs a lightweight stopping criterion that dynamically adjusts the number of samples (n) for each input, as opposed to using a fixed budget (k). The intuition is that if a clear majority is established with high confidence after sampling fewer than k answers ($n < k$), there is no need to draw additional samples. Consequently, our method reduces computational costs without compromising output quality. Our experiments show that Adaptive-Consistency achieves $n < k$ in most cases (on average, $n \sim 0.25k$ for CODE-DAVINCI-002) compared to the fixed-budget approach of Self-Consistency.

Adaptive-Consistency models the probability distribution over unique samples using a Dirichlet distribution, allowing us to quantify the confidence in the lead of the majority element over other elements. For instance, if the majority element has a count of 9 out of the first 10 samples, the likelihood of it remaining the majority element even after 40 samples is very high ($> 99\%$). This allows Adaptive-Consistency to stop sampling at this point, reducing the cost by 30 samples, while Self-Consistency would continue to sample all 40 answers. As a fast inference-time technique requiring no additional training, Adaptive-Consistency provides a convenient off-the-shelf option for all pre-trained language models, offering the flexibility to balance computational cost and performance.

We evaluate Adaptive-Consistency on 13 diverse tasks and two LLMs of different scales (VICUNA-

[1] We will release all the code and LLM outputs upon acceptance.

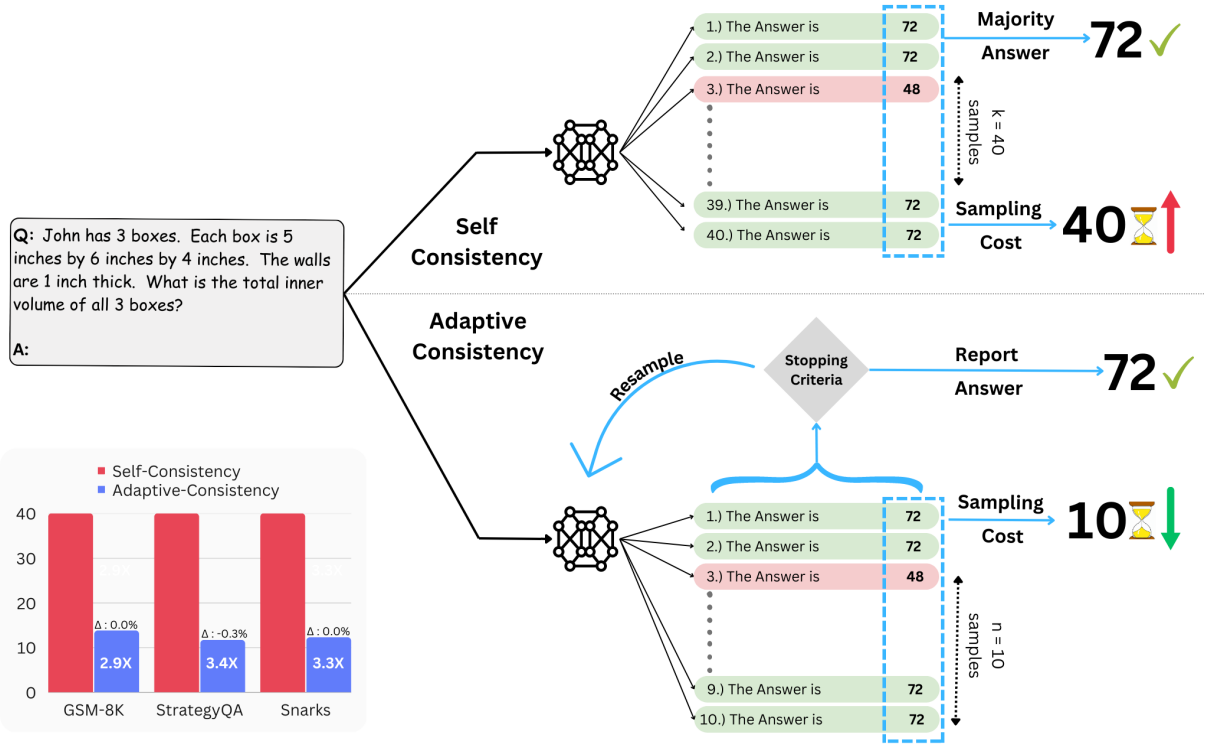


Figure 1: An overview of *Adaptive-Consistency*: Self-Consistency samples a predetermined number of answers, whereas Adaptive-Consistency iteratively samples until a lightweight Stopping Criteria, decides to report the majority answer. In the figure, we illustrate an example where *Adaptive-Consistency* achieves a significant reduction in sampling cost/time from 40 to 10 samples by reporting the majority answer after sampling only ten answers. The bottom-left graph compares *Adaptive-Consistency* with Self-Consistency on three representative reasoning datasets, demonstrating that *Adaptive-Consistency* consistently reduces the sample budget significantly (Average: $2.8\times$) with a negligible drop in accuracy (Average: -0.03%).

13B and CODE-DAVINCI-002). Our experimental results show that Adaptive-Consistency outperforms Self-Consistency regarding cost efficiency while maintaining comparable output quality. On CODE-DAVINCI-002, Adaptive-Consistency reduces the number of samples required by a factor of $3.7\times$, with an average drop in accuracy: 0.08%. On VICUNA-13B, Adaptive-Consistency requires sampling $2\times$ fewer samples, with almost no drop in accuracy.

In summary, our contributions are:

- We propose Adaptive-Consistency, a novel and cost-efficient sampling technique for large language models that dynamically adjusts the number of samples using a lightweight stopping criterion based on the stability of the majority element.
- We develop different *stopping criteria*s for our algorithm, each with its own advantages and disadvantages, such as providing probabilities, adaptability to various scenarios, and ease of implementation.
- We present experimental results demonstrating that Adaptive-Consistency requires fewer samples in most cases compared to the fixed-budget approach of Self-Consistency, while maintaining comparable output quality across various natural language reasoning tasks.

2 Methodology

2.1 Background

In-Context Few-Shot Prompting In-context few-shot prompting is a technique employed by large language models (LLMs) to learn and generalize from a limited number of examples provided within the input of a given task. The model can quickly adapt to novel tasks without fine-tuning or additional training by conditioning the model on a few examples. Specifically, a prompt p is constructed by concatenating multiple input-answer example pairs $< x_i, a_i >$. The prompt is then prepended to the test input x_{test}, and the model generates the corresponding answer a_{test}.

Listing 1 Comparison of Adaptive-Consistency (top) and Self-Consistency (bottom). Self-Consistency always draws a fixed number of samples. In contrast, Adaptive-Consistency uses a lightweight stopping criterion, allowing it to adaptively halt the sampling process, which can lead to improved efficiency and performance.

```python
def adaptive_consistency(max_gens, stop_criterion):
    observations = []
    for k in range(1, max_gens):
        observations.append(sample_from_llm())
        if stop_criterion(observations):
            break
    return majority(observations)

def stop_criterion(observations, threshold):
    # Implement your lightweight stopping criterion
```

```python
def self_consistency(max_gens):
    observations = []
    for k in range(1, max_gens):
        observations.append(sample_from_llm())
    return majority(observations)
```

Self-Consistency (Wang et al., 2022b) proposed Self-Consistency which improved performance over CoT reasoning by sampling multiple diverse reasoning chains and aggregating their outputs using a simple majority voting mechanism. However, higher accuracy is a trade-off for increased computational cost since the model must be prompted multiple times for the same question.

2.2 Adaptive-Consistency

As discussed, the Self-Consistency method consistently samples a predetermined number of answers (k) from the language model before returning the majority answer. In contrast, the Adaptive-Consistency method takes an incremental approach to sampling outputs from the language model. After generating each sample, Adaptive-Consistency employs a *stopping criteria* to determine whether it should 1.) produce an additional sample or 2.) cease sampling and report the current majority answer. This flexible strategy enables Adaptive-Consistency to dynamically adjust the number of samples generated so far (n) for each input. As our experiments demonstrate, n is typically less than k (up to $4\times$ less in some cases), which suggests that the Adaptive-Consistency method may offer greater cost-efficiency compared to the fixed budget approach employed by Self-Consistency.

The Adaptive-Consistency Algorithm (1) differs from Self-Consistency only in terms of the stopping criteria. The design of the stopping criteria is crucial to our method, as it aims to minimize the average number of samples drawn from the LLM while maximizing accuracy. The simplicity of our algorithm allows for the use of various stopping criterias interchangeably, each with their own advantages and disadvantages. We expand on a particular choice of stopping function next.

Dirichlet Stopping Criteria Let n be the number of samples drawn so far, with m unique elements. Let $v = [v_1, v_2, \ldots, v_m]$ be the counts of each element, and $p_i = \frac{v_i}{n}$ be the normalized count. Considering $n = 10$, and $m = 3$ (3 unique elements draw), if $v = [8, 1, 1]$, then we can be more confident that v_1 is the answer. On the other hand, if $v = [4, 4, 2]$, then more samples need to be drawn. Our goal is to formalize and quantify this intuition.

By convention, let $p_1 = \max(p_i)$. We want to assess the *stability* of p_1 as the majority element. Specifically, we want to ask the following question: what is the probability that p_1 will be the majority element if we repeat the process of drawing n samples again? Intuitively, if this probability is higher than some predetermined threshold C_{thresh}, then we can be more confident in our decision to stop sampling and return p_1 as the majority element:

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

To answer this question, we establish a connection with the Dirichlet distribution. Specifically, we note that the counts v parameterize a Dirichlet

distribution, Dir(V). [2]. Also, p represents a multinomial distribution and is the expected value of v. That is, $p = \mathbb{E}(\text{Dir}(V))$. This connection allows us to explore the behavior of the sampling process by drawing more samples from Dir(V) and observing the stability of p_1 as the majority element. To compute the probability of p_1 being the majority element, we can integrate the joint probability density function of the Dirichlet distribution over the appropriate region of the probability simplex. The integral can be expressed as follows:

$$P(p_1 > \max_{i=2}^{m} p_i \mid V) = \int_0^1 \int_{\mathcal{S}(p_1')} f(p_1', p_2, \ldots, p_m \mid V). \; dp_2 \cdots dp_m dp_1',$$

where

$$\mathcal{S}(p_1') = \{(p_2, \ldots, p_m) \mid p_1' > \max_{i=2}^{m} p_i, \; \sum_{i=2}^{m} p_i = 1 - p_1'\}. \quad (1)$$

The function $f(p_1', p_2, ..., p_m|V)$ represents the joint probability density function of the Dirichlet distribution conditioned on the counts V. In Equation 1, the bounds on the integral for p_1' range from 0 to 1. The probability simplex $\mathcal{S}(p_1')$ is defined for each p_1' value, such that $p_1' > \max_{i=2}^{m} p_i$, and the remaining p_i values sum to $1 - p_1'$. This constraint ensures that we are considering all possible values of p_1' that would maintain its majority status.

Beta Stopping Criteria Since the number of unique answers in the observation set can be large, Equation (1) is computationally expensive to solve. As an approximation, we observe that establishing the majority of p_1 over the next largest probability, p_2, is sufficient for our purpose.

In this setting, the probability in Equation (2.2) simplifies to a Beta distribution with parameters $(v_1 + 1, v_2 + 1)$, and Equation (1) is replaced by Equation (2). This approximation allows us to efficiently compute the confidence in p_1 being the majority, enabling early stopping decisions without incurring substantial computational overhead.

$$\int_0^{0.5} p_2^{v2} \cdot (1 - p_2)^{v_1} dp_2 \quad (2)$$

[2]Dirichlet is a distribution over multinomials; each draw from Dirichlet is a multinomial distribution. See Details in Appendix D

Empirically, we show the performance to be similar to Dirichlet stopping criteria but significantly faster (See Section 4.2). Throughout experiments, we refer to this Beta Stopping Criteria as Adaptive-Consistency.

Intuitively, the stopping criteria in equation (1) ensures we generate only a few samples for questions, where the majority answer can be inferred from a small number of samples with high confidence. For example, if the first five answers generated by the model are the same: a_1, then with $> 98\%$ probability, the majority answer will remain the same even on infinite generations. Thus we can save more sample generations at the cost of minimal loss in accuracy.

3 Experiments

In this section, we describe the experimental setup for evaluating our proposed method, on various reasoning benchmarks. We use three diverse categories of reasoning benchmarks, evaluate the performance on two different language models, and use prompts by PAL (Gao et al., 2022) and Self-Consistency (Wang et al., 2022b). We demonstrate marked improvements in terms of reduced sample generations when compared to Self-Consistency.

Benchmarks We evaluate our method on a diverse set of reasoning benchmarks, encompassing 13 datasets across three distinct categories:

1. **Mathematical Reasoning:** To assess mathematical reasoning capabilities, we utilize the following datasets: GSM-8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), and ASDIV (Miao et al., 2020). These datasets consist of grade-school level algebra word problems that necessitate arithmetic operations and problem-solving based on contextual information.

2. **Commonsense Reasoning Tasks:** We evaluate Adaptive-Consistency on four commonsense reasoning tasks. **1.) StrategyQA** (Geva et al., 2021) comprises questions that demand the model to infer a multi-hop strategy with reasoning steps implicitly embedded in the questions. **2.) Date Understanding** entails questions that require the model to deduce dates from natural language descriptions and perform arithmetic operations accordingly. **3.) Snarks** and **4.) Ruin Names** both focus on emotional understanding tasks.

3. **Logical Reasoning Tasks:** We examine the performance of our method on six diverse logical reasoning tasks. **1.) Tracking Shuffled Ob-**

JECTS is a tracking task that necessitates the model to infer the final state of a system, given its initial state and a sequence of modifications. **2.) LOGICAL DEDUCTION** is a logical deduction task that demands the model to deduce the order of a sequence of objects based on a minimal set of conditions. **3.) BOOLEAN EXPRESSIONS** is a boolean expressions task that evaluates whether a language model has learned the rules of deductive reasoning, i.e., formal (zeroth-order) logic associated with the words "and," "or," "not," etc. **4.) SALIENT TRANSLATION** is a salient translation error detection task that requires the model to identify the type of error in a translation. **5.) DISAMBIGUATION QA** is a disambiguation task that necessitates the model to select the person to whom the pronoun refers. **6.) PENGUINS** describes a table of penguins and requires the model to answer questions about the penguins' attributes.

Models We evaluate our method on two different language models: **1. CODE-DAVINCI-002:** A GPT-3-based publicly available model (Brown et al., 2020) which is a part of the Codex series (Chen et al., 2021) and has 175 billion parameters.[3] **2. VICUNA-13B:** (Chiang et al., 2023) an open-source transformer model fine-tuned on instruction-following dataset (Taori et al., 2023) from the base Llama series (Touvron et al., 2023).

Prompting and Sampling We use similar prompts as employed in the PAL (Gao et al., 2022) and the CHAIN OF THOUGHT (Wei et al., 2022). Specifically, for mathematical reasoning and DATE UNDERSTANDING tasks, we use the prompts from PAL. For other commonsense and logical reasoning tasks, we use prompts from COT (Wei et al., 2022).

In terms of sampling, we follow the scheme suggested in the Self-Consistency (Wang et al., 2022b). Specifically, we use a temperature of 0.7 for sampling and limit the number of generations to a maximum of 40.

Hyperparameters The only hyperparameters in Adaptive-Consistency, are those related to parameters in stopping criterias (C_{thresh}). We use a high $C_{thresh} = 0.95$ for Adaptive-Consistency. By using a high threshold, we aim to maintain high accuracy and prevent the algorithm from stopping too early. For other Stopping Criterias, we tune our parameters on the training set of GSM-8K, and use the same thresholds across all the datasets. The impact of the chosen threshold on the performance of our method is further analyzed in the Analysis Section (§ 4.1).

Baselines We compare our method against Self-Consistency, which is the current state-of-the-art method.

Further, in Analysis 4.2, we evaluate Adaptive-Consistency against different stopping criterias such as RANDOM stopping and MAJORITY (stopping at majority).

Evaluation Metrics We evaluate the performance of our method and the baselines using two metrics: average generations sampled from the large language models (LLMs) and overall reasoning accuracy. Our results show that Adaptive-Consistency achieves similar performance to Self-Consistency while often reducing sample budget upto $6\times$ times.

3.1 Results

We compare Adaptive-Consistency with Self-Consistency in Table 1. Our proposed method consistently reduces the computational budget while maintaining negligible drops in accuracy across benchmarks and models. Delta improvements over Self-Consistency are reported in the two rightmost columns.

On mathematical and logical reasoning tasks, Adaptive-Consistency significantly reduces the sample budget by factors ranging from $1.4\times$ to $6\times$ compared to Self-Consistency, depending on the task and model, with a average decrease in accuracy of less than 0.1%. This trend holds for both the VICUNA-13B and CODE-DAVINCI-002 models, which differ significantly in model size and base families.

Adaptive-Consistency also demonstrates efficiency across a diverse set of tasks, such as emotional understanding tasks (SNARKS and RUIN NAMES), translation error detection (SALIENT TRANSLATION), and ambiguity identification (DISAMBIGUATION QA). In these tasks, Adaptive-Consistency results in sample budget reductions of $1.2\times$ to $2.5\times$, with a maximum drop in accuracy of 0.4%.

Our experiments demonstrate that Adaptive-Consistency effectively enhances the performance

[3]We have access to Codex models through OpenAI's researcher access program. However, since we only need access to the model outputs for our purposes, we will release them for reproducibility.

		Self-Consistency	Adaptive-Consistency		Δ	
		Accuracy	Avg. Gen.	Accuracy	Gen. Reduc.	Acc. Diff. ↑
GSM-8K	VICUNA-13B	**31.6**	**26.8**	31.5	1.4×	−0.1
	CODE-DAVINCI-002	**81.1**	**13.8**	81.0	2.9×	−0.1
SVAMP	VICUNA-13B	**63.0**	**18.8**	62.8	2.1×	−0.2
	CODE-DAVINCI-002	**85.1**	**9.5**	85.0	4.2×	−0.1
ASDIV	VICUNA-13B	**64.0**	**16.5**	**64.0**	2.4×	0.0
	CODE-DAVINCI-002	**83.2**	**10.0**	83.2	4.0×	0.0
DATE UNDERSTANDING	VICUNA-13B	59.8	**17.3**	**60.2**	2.3×	+0.4
	CODE-DAVINCI-002	**80.3**	**10.7**	79.5	3.7×	−0.8
TRACKING SHUFFLED OBJECTS	VICUNA-13B	31.8	**20.3**	**32.0**	2.0×	+0.2
	CODE-DAVINCI-002	**77.2**	**9.7**	77.1	4.1×	−0.1
LOGICAL DEDUCTION	VICUNA-13B	51.2	**18.1**	**51.4**	2.2×	+0.2
	CODE-DAVINCI-002	**89.4**	**8.5**	**89.4**	4.7×	0.0
STRATEGYQA	VICUNA-13B	65.8	**16.3**	**65.8**	2.5×	0.0
	CODE-DAVINCI-002	**79.0**	**11.9**	78.8	3.4×	−0.2
BOOLEAN EXPRESSIONS	VICUNA-13B	**79.2**	**16.2**	78.4	2.5×	−0.8
	CODE-DAVINCI-002	**94.5**	**6.6**	**94.5**	6.0×	0.0
SNARKS	VICUNA-13B	73.2	**23.2**	**73.6**	1.7×	+0.4
	CODE-DAVINCI-002	**74.0**	**12.7**	**74.0**	3.1×	0.0
RUIN NAMES	VICUNA-13B	**43.6**	**33.8**	**43.6**	1.2×	0.0
	CODE-DAVINCI-002	**78.0**	**17.2**	**78.0**	2.3×	0.0
SALIENT TRANSLATION	VICUNA-13B	**28.9**	**28.7**	28.7	1.2×	−0.3
	CODE-DAVINCI-002	**64.3**	**11.8**	**64.3**	3.4×	0.0
DISAMBIGUATION QA	VICUNA-13B	**63.7**	**22.8**	63.5	1.8×	−0.3
	CODE-DAVINCI-002	74.9	**13.5**	**75.1**	3.0×	+0.1
PENGUINS	VICUNA-13B	46.8	**22.9**	**47.3**	1.7×	+0.5
	CODE-DAVINCI-002	83.8	**11.0**	**84.0**	3.6×	+0.2
Average		**67.2**	**16.5**	**67.2**	2.8×	**-0.03**

Table 1: Comparison of Adaptive-Consistency with Self-Consistency on 13 diverse reasoning datasets. The table presents the accuracy of Self-Consistency, the average number of generations (Avg. Gen.) for Adaptive-Consistency, and the accuracy of Adaptive-Consistency. Self-Consistency always draws 40 samples. The Δ columns display the reduction in generations (Gen. Reduc.) and the difference in accuracy (Acc. Diff.) between Self-Consistency and Adaptive-Consistency. The last row shows the average values across all datasets:,Adaptive-Consistency achieves a 2.8 times reduction in sample budget (*Gen. Reduc.*) compared to Self-Consistency, while maintaining a minimal average accuracy drop of 0.03% (*Acc. Diff.*).

of large language models on diverse reasoning tasks by adaptively adjusting the stopping criterion. With comparable accuracy to Self-Consistency and significantly reduced sample budgets, Adaptive-Consistency showcases its potential for more efficient and effective reasoning across various domains, output responses, and task difficulties.

In summary, our experiments show that Adaptive-Consistency enhances large language models' performance on diverse reasoning tasks by adaptively adjusting the stopping criterion. Adaptive-Consistency achieves accuracy comparable to Self-Consistency while significantly reducing the sample budget, highlighting Adaptive-Consistency' potential for more efficient and effective reasoning in large language models.

4 Analysis

4.1 Effect of Confidence Threshold in Adaptive-Consistency

The confidence threshold, C_{thresh}, is a crucial hyperparameter for Adaptive-Consistency, as it determines when to stop sampling based on the desired level of confidence in the majority element. While we set the threshold to a stringent value of 0.95 for all experiments, in this section, we analyze the impact of varying C_{thresh} from 0.5 to 1 to understand the trade-offs between model accuracy and cost-efficiency.

In Figure 2, we present a visualization that examines the relationship between the confidence threshold, C_{thresh}, and the performance of our adaptive consistency method in terms of both accuracy and cost-efficiency. The x-axis represents the confidence threshold, varying from 0.5 to 1. The left y-axis displays the model's accuracy, while the right y-axis shows the average number of samples drawn.

The plot shows the expected behavior of two curves: the blue curve (accuracy) increases gradually and then plateaus, while the red curve (average number of samples) initially increases linearly and then climbs more steeply. The plateau in accuracy signifies that the model has reached its maximum achievable accuracy, and further sampling doesn't improve it. Meanwhile, the red curve's climbing rate indicates that the model requires more samples to meet increasingly stringent confidence thresholds, highlighting the trade-off between accuracy and cost-efficiency.

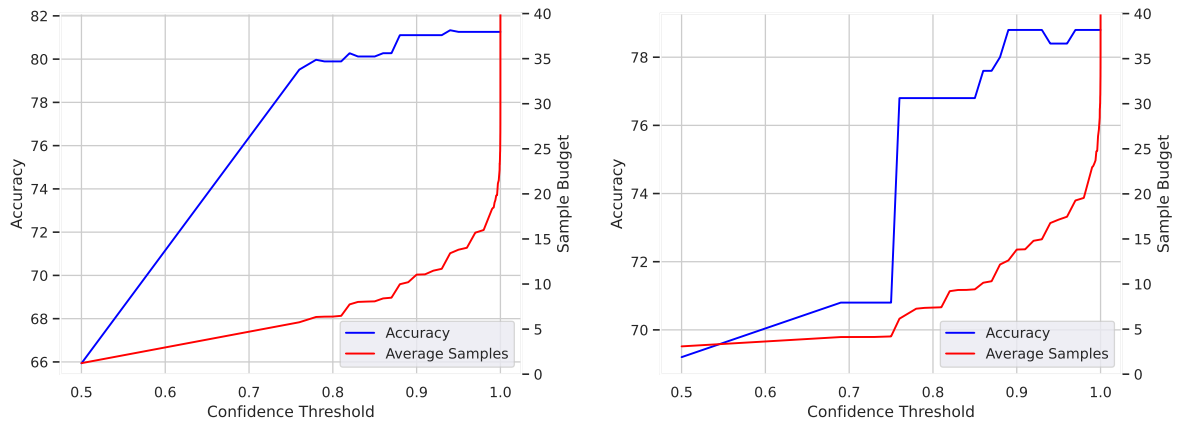
4.2 Evaluation of Different Stopping Functions

Adaptive-Consistency allows a flexible choice of stopping criteria, based on intended objective and requirements. Here we evaluate various such stopping functions. Specifically, we evaluate on five different functions: 1.) RANDOM: randomly stopping with a probability p, 2.) MAJORITY: stopping after most common answer has majority above a threshold, 3.) ENTROPY: stopping after entropy of answers is below a threshold, 4.) BETA: The main stopping criteria used in Adaptive-Consistency, based on the Equation (2), 5.) DIRICHLET: The stopping criteria, based on Equation (1). We show results in Table 4. The parameters for all these methods are tuned as discussed in Section 3. We note, that while RANDOM allows to control the sample budget consistently (fixed at 10), it is the least effective of all stopping criterias evaluated. MAJORITY almost consistently underperforms both BETA and ENTROPY. While ENTROPY is competitive to BETA, ENTROPY lack human-interpretable stopping rationale. Moreover, BETA can provide guarantees on the accuracy. DIRICHLET has similar performance to BETA, however is almost 1200× slower than BETA, because of the expensive multi-variate integral calculation. This motivates the simplification of DIRICHLET into BETA as used in Adaptive-Consistency. We refer readers to Appendix C.1 for more details.

5 Related Work

Architectures for adaptive computation A related body of work on adaptive computation aims to preempt computation based on intermediate representations (Liu et al., 2020; Zhou et al., 2020; Schuster et al., 2021; Geng et al., 2021; Xin et al., 2020). Schuster et al. (2022) present CLAM, a language model that performs language generation adaptively. Hou et al. (2020) propose Dynamic Bert, which can adapt the depth and width of the transformer to satisfy various computational constraints. Xing et al. (2020) propose a dynamic deep neural network with an early-exit strategy embedded for enhancing the quality of compressed images. Another direction of work focuses on pruning model weights or training sparse weights (Fan et al., 2019; Jayakumar et al., 2021) to reduce training and inference time. In contrast to these methods, our approach completely obviates making any architectural modifications.

Inference-time adaptive computation These methods focus on adaptive computation at inference time without making architectural modifications to the models. Schwarzschild et al. (2021b,a) focus on three different generalization tasks. They observe that increasing the number of test iterations (which corresponds to the network depth in their setting) helps the models in generalizing better to difficult problems. Madaan and Yang (2022) leverage two different networks trained for the same task, a larger variant (slow) and a smaller variant (fast). The switch from fast to slow happens during inference, based on the complexity of generation at the current step. Xue et al. (2023) train language models to adaptively read tokens from a tape bank for each input. Different from these works, our focus is tasks where the multiple samples are drawn



(a) GSM-8K (b) RUIN NAMES

Figure 2: Impact of Confidence Threshold (C_{thresh}) on Adaptive-Consistency: As C_{thresh} varies, the accuracy of Adaptive-Consistency increases gradually, eventually plateauing. Initially, the average number of generations also increases gradually but then sharply climbs, reflecting the accuracy-confidence trade-off.

from a model (vs. iteratively solving a task, which is a focus of these works).

Adaptive Sampling in Training and Active Learning Another line of work focuses on importance-based sampling of input instances during training (Bengio and Senecal, 2008; Prabhu et al., 2019; Berger et al., 2017). For instance, Bengio and Senecal (Bengio and Senecal, 2008) introduce an adaptive importance sampling technique for expediting the training of neural probabilistic language models. Prabhu et al.(Prabhu et al., 2019) investigate sampling bias in deep active classification, proposing an empirical method to mitigate it. Berger et al.(Berger et al., 2017) present an adaptive sampling scheme for training fully convolutional networks efficiently in semantic segmentation tasks involving large and imbalanced datasets. These approaches primarily focus on sampling input instances during the training phase.

In a related context, active learning techniques aim to optimize the learning process by selectively acquiring labels for the most informative instances (Settles, 2009; Cohn et al., 1994; Tong and Koller, 2001). In active learning scenarios, a learner communicates with an oracle, typically a human annotator, to request labels for specific instances that are anticipated to maximize the model's performance improvement. In contrast to the aforementioned methods, our approach centers on adaptively sampling multiple outputs per input instance during the inference phase, without soliciting additional labels. Our method is crafted to efficiently obtain reliable predictions from pre-trained language models by adaptively sampling their outputs, distinguishing it from both adaptive sampling in training and active learning, which focus on the training phase.

6 Conclusion and Future Work

This paper presented Adaptive-Consistency, a cost-efficient and model-agnostic technique for improving the correctness of output from large language models (LLMs) using dynamic sampling. Our approach builds upon the Self-Consistency method and introduces a lightweight stopping criterion that allows for adaptive sampling based on the amount of agreement in the samples drawn so far. Adaptive-Consistency is effective across 13 datasets and two LLMs, reducing the required sample budget by up to 6.0 times while maintaining comparable accuracy, with an average drop of less than 0.1%.

Our work opens up several avenues for future research. First, investigating alternative stopping criteria or combining multiple criteria could lead to even more efficient sampling techniques. Finally, in our current approach, the majority decision relies on using exact matches to determine the most common answer. However, this may not always capture the true majority in cases where the output has minor variations that do not affect the overall correctness or relevance of the answer. As a future direction, we plan to investigate the use of approximate matching techniques to identify and count elements that are *close enough* as identical.

Limitations

Despite the promising results of our proposed Adaptive-Consistency method, it comes with several limitations and opportunities for future improvement.

- **Stopping criterion sensitivity:** Our current stopping criterion is based on the stability of the majority element in the set of samples. While this has shown to be effective in our experiments, it may not always be the best indicator of agreement in the samples. There could be cases where the majority element is not stable enough, but the stopping criterion still triggers, potentially leading to suboptimal decisions. Exploring alternative or more robust stopping criteria could help mitigate this issue.
- **Generalizability:** Although we have tested our method across a diverse range of 13 datasets and two different LLMs of contrastive scale, there may still be tasks or models where the Adaptive-Consistency approach might not be as effective. Notably, Adaptive-Consistency is expected to fail where Self-Consistency fails.
- **Task-specific adaptations:** The current implementation of Adaptive-Consistency is task-agnostic, which might limit its performance on specific tasks that could benefit from task-specific adaptations. Designing specialized versions of Adaptive-Consistency for particular tasks or domains could potentially lead to better performance in those areas.
- **Reliance on the pretrained LLM:** Our method builds upon the Self-Consistency technique and relies on the pretrained LLM to generate multiple samples. Consequently, any limitations or biases present in the underlying LLM would still be carried over to the Adaptive-Consistency method. Addressing these issues might require improvements in the LLM training process itself or the incorporation of external knowledge sources.

References

Yoshua Bengio and Jean-Sébastien Senecal. 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model. IEEE Transactions on Neural Networks, 19:713–722.

Lorenz Berger, Eoin R. Hyde, M. Jorge Cardoso, and Sébastien Ourselin. 2017. An adaptive sampling scheme to efficiently train fully convolutional networks for semantic segmentation. In Annual Conference on Medical Image Understanding and Analysis.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. ArXiv, abs/2107.03374.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. ArXiv, abs/2110.14168.

David A Cohn, Les Atlas, and Richard E Ladner. 1994. Improving generalization with active learning. In Machine Learning Proceedings 1994, pages 201–221. Elsevier.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. ArXiv, abs/1909.11556.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. ArXiv, abs/2211.10435.

Shijie Geng, Peng Gao, Zuohui Fu, and Yongfeng Zhang. 2021. Romebert: Robust training of multi-exit bert. arXiv preprint arXiv:2101.09755.

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. Transactions of the Association for Computational Linguistics, 9:346–361.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. ArXiv, abs/2004.04037.

Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. 2021. Top-kast: Top-k always sparse training. ArXiv, abs/2106.03517.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. arXiv preprint arXiv:2205.11916.

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. arXiv preprint arXiv:2206.14858.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom, Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de, Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey, Cherepanov, James Molloy, Daniel Jaymin Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de, Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. Science, 378:1092 – 1097.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6035–6044.

Aman Madaan, Alexander Shypula, Uri Alon, Milad Hashemi, Parthasarathy Ranganathan, Yiming Yang, Graham Neubig, and Amir Yazdanbakhsh. 2023. Learning performance-improving code edits. arXiv preprint arXiv:2302.07867.

Aman Madaan and Yiming Yang. 2022. Flowgen: Fast and slow graph generation. DyNN workshop at the 39 th International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

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, pages 975–984, Online. Association for Computational Linguistics.

Bhavana Dalvi Mishra, Oyvind Tafjord, and Peter Clark. 2022. Towards teachable reasoning systems: Using a dynamic memory of user feedback for continual system improvement.

Arkil Patel, S. Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In North American Chapter of the Association for Computational Linguistics.

Ameya Prabhu, Charles Dognin, and Maneesh Kumar Singh. 2019. Sampling bias in deep active classification: An empirical study. In Conference on Empirical Methods in Natural Language Processing.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. arXiv preprint arXiv:2207.07061.

Tal Schuster, Adam Fisch, Tommi Jaakkola, and Regina Barzilay. 2021. Consistent accelerated inference via confident adaptive transformers. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 4962–4979.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Arpit Bansal, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. 2021a. Datasets for Studying Generalization from Easy to Hard Examples. arXiv:2108.06011 [cs]. ArXiv: 2108.06011.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. 2021b. Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks. In Advances in Neural Information Processing Systems, volume 34, pages 6695–6706. Curran Associates, Inc.

Burr Settles. 2009. Active learning literature survey.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Simon Tong and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. Journal of machine learning research, 2(Nov):45–66.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. ArXiv, abs/2302.13971.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022a. Rationale-Augmented Ensembles in Language Models. arXiv preprints arXiv:2207.00747.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. 2022b. Self-consistency improves chain of thought reasoning in language models. ArXiv, abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. ArXiv, abs/2201.11903.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy J. Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In Annual Meeting of the Association for Computational Linguistics.

Qunliang Xing, Mai Xu, Tianyi Li, and Zhenyu Guan. 2020. Early exit or not: Resource-efficient blind quality enhancement for compressed images. In European Conference on Computer Vision.

Fuzhao Xue, Valerii Likhosherstov, Anurag Arnab, Neil Houlsby, Mostafa Dehghani, and Yang You. 2023. Adaptive computation with elastic input sequence. ArXiv, abs/2301.13195.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. Advances in Neural Information Processing Systems, 33:18330–18341.

A Experimental Setup

A.1 Hyperparameters

The only hyperparameters in Adaptive-Consistency, are those related to parameters in stopping criterias (C_{thresh}). We use a high $C_{thresh} = 0.95$ for Adaptive-Consistency. By using a high threshold, we aim to maintain high accuracy and prevent the algorithm from stopping too early. For other Stopping Criterias, we tune our parameters on the training set of GSM-8K, and use the same thresholds across all the datasets. The impact of the chosen threshold on the performance of our method is further analyzed in the Analysis Section (§ 4.1). We further evaluate all methods on a set of 3 seeds, and report the table with standard deviation in Table 3.

A.2 Tools and Framework

For querying CODE-DAVINCI-002 models (Chen et al., 2021), we use the api library provided by OpenAI[4]. We use the official code provided for running VICUNA-13B model (Chiang et al., 2023). We run inference on VICUNA-13B models on single A100 gpus. stopping criteria in Adaptive-Consistency are fast to run, and we use single-cpu core machine for it. For numerical integration, we use the scipy library in Python.

A.3 Benchmarks

As described in Section 3, we evaluate our method on 13 diverse reasoning datasets. Dataset Statistics are presented in Table 2.

Dataset	$\|N\_test\|$	Answer Format
GSM-8K	1319	Numerical
ASDIV	2096	Numerical
SVAMP	1000	Numerical
DATE UNDERSTANDING	369	String
TRACKING SHUFFLED OBJECTS	250	MCQ
LOGICAL DEDUCTION	250	MCQ
STRATEGYQA	2279	Binary
BOOLEAN EXPRESSIONS	250	Binary
SNARKS	250	Binary
RUIN NAMES	178	MCQ
SALIENT TRANSLATION	250	MCQ
DISAMBIGUATION QA	250	MCQ
PENGUINS	146	MCQ

Table 2: Dataset Statistics. We evaluate on 13 diverse reasoning datasets, having different difficulty, domain, answer type, size (N_test).

B Results

We present the complete results with standard deviation in Table 3.

[4]API available at: https://platform.openai.com/

C Analysis

C.1 Stopping Criterias

This section follows from main discussion in Section 4.2. We evaluate different stopping criterias for Adaptive-Consistency. We evaluate on 5 different functions:

1. RANDOM: randomly stopping with a probability p, 2.)
2. MAJORITY: stopping after most common answer has majority above a threshold,
3. ENTROPY: stopping after entropy of answers is below a threshold,
4. BETA: The main stopping criteria used in Adaptive-Consistency, based on the Equation (2),
5. DIRICHLET: The stopping criteria, based on Equation (1).

For comparison, we tune the C_{thresh} in each case on the training set of GSM-8K dataset. Results are presented in Table 4.

C.2 Effect of Confidence Threshold on Adaptive-Consistency

We follow the discussion in Section 4.1, and present complete results on all datasets for CODE-DAVINCI-002.

D Derivation of DIRICHLET stopping criteria

Consider for a given input (I), the model can generate one of m distinct answers $A := \{a_1, a_2, \ldots a_m\}$. Define the probability of generating an answer given an input as $p_i := P(a_i \mid I)$. Now, consider an observation set (O) with counts of each of a_i as v_i, such that $\sum_{i=1}^{m} v_i = n$. Now, without loss of generality, consider $p_1 > \max_{i=2}^{m} p_i$. Now, based on Equation (2.2), we need to find the probability:

$$P(p_1 > \max_{i=2}^{m} p_i \mid O)$$

.

However, here the p_is are latent variables, and only O is available to us. We next make the following

Assumption 1: The vector $\vec{p} = \{p_1, p_2 \ldots p_m\}$ is

		Self-Consistency	Adaptive-Consistency		Δ	
		Accuracy	Avg. Gen.	Accuracy	Gen. Reduc.	Acc. Diff. ↑
GSM-8K	VICUNA-13B	**31.6**(±3.0)	**26.8**(±2.2)	31.5(±3.0)	1.4×	−0.1
	CODE-DAVINCI-002	**81.1**(±0.3)	**13.8**(±0.0)	81.0(±0.3)	2.9×	−0.1
SVAMP	VICUNA-13B	**63.0**(±0.3)	**18.8**(±0.1)	62.8(±0.4)	2.1×	−0.2
	CODE-DAVINCI-002	**85.1**(±0.3)	**9.5**(±0.1)	85.0(±0.3)	4.2×	−0.1
ASDIV	VICUNA-13B	**64.0**(±0.3)	**16.5**(±0.2)	**64.0**(±0.3)	2.4×	0.0
	CODE-DAVINCI-002	**83.2**(±0.2)	**10.0**(±0.0)	83.2(±0.2)	4.0×	0.0
DATE UNDERSTANDING	VICUNA-13B	59.8(±0.3)	**17.3**(±0.3)	**60.2**(±0.4)	2.3×	+0.4
	CODE-DAVINCI-002	**80.3**(±0.1)	**10.7**(±0.3)	79.5(±0.3)	3.7×	−0.8
TRACKING SHUFFLED OBJECTS	VICUNA-13B	31.8(±1.0)	**20.3**(±0.0)	**32.0**(±1.2)	2.0×	+0.2
	CODE-DAVINCI-002	**77.2**(±1.3)	**9.7**(±0.1)	77.1(±1.6)	4.1×	−0.1
LOGICAL DEDUCTION	VICUNA-13B	51.2(±0.8)	**18.1**(±0.2)	**51.4**(±0.6)	2.2×	+0.2
	CODE-DAVINCI-002	**89.4**(±0.2)	**8.5**(±0.1)	**89.4**(±0.2)	4.7×	0.0
STRATEGYQA	VICUNA-13B	65.8(±0.5)	**16.3**(±0.1)	**65.8**(±0.4)	2.5×	0.0
	CODE-DAVINCI-002	**79.0**(±0.2)	**11.9**(±0.2)	78.8(±0.1)	3.4×	−0.2
BOOLEAN EXPRESSIONS	VICUNA-13B	**79.2**(±0.6)	**16.2**(±0.3)	78.4(±0.3)	2.5×	−0.8
	CODE-DAVINCI-002	**94.5**(±0.4)	**6.6**(±0.1)	**94.5**(±0.4)	6.0×	0.0
SNARKS	VICUNA-13B	73.2(±1.0)	**23.2**(±0.7)	**73.6**(±0.8)	1.7×	+0.4
	CODE-DAVINCI-002	**74.0**(±1.0)	**12.7**(±0.4)	**74.0**(±1.5)	3.1×	0.0
RUIN NAMES	VICUNA-13B	**43.6**(±2.1)	**33.8**(±0.6)	**43.6**(±2.1)	1.2×	0.0
	CODE-DAVINCI-002	**78.0**(±0.9)	**17.2**(±0.1)	**78.0**(±0.6)	2.3×	0.0
SALIENT TRANSLATION	VICUNA-13B	**28.9**(±2.4)	**28.7**(±2.5)	28.7(±2.5)	1.2×	−0.3
	CODE-DAVINCI-002	**64.3**(±0.2)	**11.8**(±0.5)	**64.3**(±0.2)	3.4×	0.0
DISAMBIGUATION QA	VICUNA-13B	**63.7**(±0.7)	**22.8**(±1.0)	63.5(±1.1)	1.8×	−0.3
	CODE-DAVINCI-002	74.9(±0.8)	**13.5**(±0.6)	**75.1**(±0.7)	3.0×	+0.1
PENGUINS	VICUNA-13B	46.8(±1.8)	**22.9**(±0.7)	**47.3**(±1.9)	1.7×	+0.5
	CODE-DAVINCI-002	83.8(±0.9)	**11.0**(±0.4)	**84.0**(±0.6)	3.6×	+0.2
Average		**67.2**(±0.8)	**16.5**(±0.5)	**67.2**(±0.8)	2.8×	**-0.03**

Table 3: Comparison of Adaptive-Consistency with Self-Consistency on 13 diverse reasoning datasets. The table presents the accuracy of Self-Consistency, the average number of generations (Avg. Gen.) for Adaptive-Consistency, and the accuracy of Adaptive-Consistency. Self-Consistency always draws 40 samples. The Δ columns display the reduction in generations (Gen. Reduc.) and the difference in accuracy (Acc. Diff.) between Self-Consistency and Adaptive-Consistency. The last row shows the average values across all datasets:,Adaptive-Consistency achieves a 2.8 times reduction in sample budget (*Gen. Reduc.*) compared to Self-Consistency, while maintaining a minimal average accuracy drop of 0.03% (*Acc. Diff.*).

sampled from uniform distribution over $(m-1)$-simplex.

Thus, $p_1 = 1 - \sum_{i=1}^{m-1} p_i$. Since the observation set follows a multinomial distribution with parameters $\vec{p}$, conditional joint probability distribution of O given $\vec{p}$ can be written as:

$$P(O \mid \vec{p}) = \frac{n!}{\prod_{i=1}^{m}(v_i!)} \prod_{i=1}^{m} p_i^{v_i} = Dir(v_1{+}1, v_2{+}1 \ldots, v_m{+}1)$$

, where Dir represents the dirichlet distribution with $v_i + 1$, as its parameters. Applying Baye's Rule,

$$P(\vec{p} \mid O) = \frac{P(O \mid \vec{p}) \cdot P(\vec{p})}{P(O)}$$

Here $P(O)$ is a normalizing constant and can be omitted for computation. From Assumption 1,

		Random		Majority		Beta (Adaptive-Consistency)		Entropy		Dirichlet	
		Average ↓ Generations	Accuracy ↑	Average ↓ Generations	Accuracy ↑	Average ↑ Generations	Accuracy ↑	Average ↑ Generations	Accuracy ↑	Average ↑ Generations	Accuracy ↑
GSM-8K	Vicuna-13B	**9.4**	23.4	10.6	24.6	20.9	**28.8**	20.3	28.5	21.6	**28.8**
	code-davinci-002	10.2	75.8	11.1	78.9	**10.0**	80.2	10.2	81.1	10.5	**81.2**
SVAMP	Vicuna-13B	10.1	57.6	**10.0**	59.1	13.7	62.7	13.9	**62.9**	14.7	62.7
	code-davinci-002	9.8	82.7	7.4	**85.4**	**6.0**	85.1	6.5	85.1	6.6	85.2
ASDIV	Vicuna-13B	10.5	61.4	**7.6**	61.8	12.7	64.0	13.0	64.2	13.5	**64.2**
	code-davinci-002	10.2	81.9	**6.9**	83.5	7.5	83.7	7.6	**83.8**	7.6	83.5
Date Understanding	Vicuna-13B	**10.0**	55.0	10.2	57.2	12.2	58.5	13.1	**60.2**	13.6	**60.2**
	code-davinci-002	10.0	74.3	8.9	77.0	**6.5**	77.0	7.0	**78.3**	7.4	**78.3**
Tracking Shuffled Objects	Vicuna-13B	2.6	**4.1**	1.9	3.6	**1.7**	4.0	2.1	4.0	2.6	4.0
	code-davinci-002	10.0	77.2	**5.8**	78.0	6.4	78.8	6.8	**79.6**	7.0	**79.6**
Logical Deduction	Vicuna-13B	2.7	7.5	1.7	6.8	**1.6**	**8.0**	2.0	7.6	2.1	**8.0**
	code-davinci-002	10.3	88.4	6.3	89.5	**4.8**	**91.4**	5.8	90.0	6.0	89.5
StrategyQA	Vicuna-13B	10.9	64.1	**6.2**	64.1	10.4	65.2	11.6	65.2	11.8	**65.3**
	code-davinci-002	9.9	76.3	7.0	77.7	**6.9**	78.1	7.4	**78.6**	7.4	78.5
Boolean Expressions	Vicuna-13B	10.1	75.5	**3.7**	74.0	9.7	76.8	10.6	**78.0**	10.7	77.6
	code-davinci-002	9.8	93.1	**3.3**	**94.4**	4.0	93.6	4.1	**94.4**	4.1	**94.4**
Snarks	Vicuna-13B	10.8	66.9	**4.1**	61.8	13.8	73.0	15.9	73.0	17.0	**73.6**
	code-davinci-002	9.9	72.6	8.8	72.6	**6.3**	73.6	8.5	74.5	8.3	**75.5**
Ruin Names	Vicuna-13B	10.8	33.3	**8.4**	30.8	25.8	41.2	27.0	**41.6**	28.2	**41.6**
	code-davinci-002	**10.5**	70.2	11.8	71.2	11.0	76.4	12.6	**78.8**	13.1	76.8
Salient Translation	Vicuna-13B	**9.5**	27.4	11.4	24.8	18.4	**30.4**	20.8	**30.4**	22.2	**30.4**
	code-davinci-002	10.0	62.7	9.2	63.6	**7.5**	64.4	8.6	**65.2**	9.0	64.8
Disambiguation QA	Vicuna-13B	10.6	64.1	**8.5**	60.4	14.0	**65.6**	17.5	63.6	18.4	64.0
	code-davinci-002	10.1	71.0	10.1	70.4	**7.8**	72.0	8.8	**72.8**	9.6	**72.8**
Penguins	Vicuna-13B	10.3	43.3	**10.3**	43.8	14.2	**45.9**	17.0	**45.9**	18.1	**45.9**
	code-davinci-002	9.7	80.5	9.7	83.6	**7.1**	**85.6**	7.9	84.2	8.2	83.6

Table 4: Comparison of various Stopping Criterias in Adaptive-Consistency. In general, Beta outperforms Random and Majority by decent margins across all datasets. Beta has comporable performance to Dirichlet, but the latter is much more slower. Entropy performs similar to Beta but lacks human-interpretable stopping rationale.

since $\vec{p}$ is sampled from uniform distribution,

$$P(\vec{p}) = \prod_{i=2}^{m} dp_i$$

. Thus conditional joint probability distribution of $\vec{p}$ given O can be written as:

$$P(\vec{p} \mid O) = Dir(v_1{+}1, v_2{+}1 \dots v_m{+}1)\ dp_m dp_{m-1} \dots dp_2$$

.

Now we can integrate the above equation over a subset of $(m-1)$-simplex, such that $p_1 > \max_{i=2}^{m} p_i$. This gives us the equation:

$$P(p_1 > \max_{i=2}^{m} p_i \mid O) = \int_0^1 \int_{\mathcal{S}(p_1')} P(\vec{p} \mid O)\ dp_2 \cdots dp_m dp_1',$$

where

$$\mathcal{S}(p_1') = \{(p_2, \dots, p_m) \mid p_1' > \max_{i=2}^{m} p_i, \sum_{i=2}^{m} p_i = 1 - p_1'\}. \quad (3)$$

We note that the integration has no closed-form solution, and we use numerical approximation to compute the above integral.

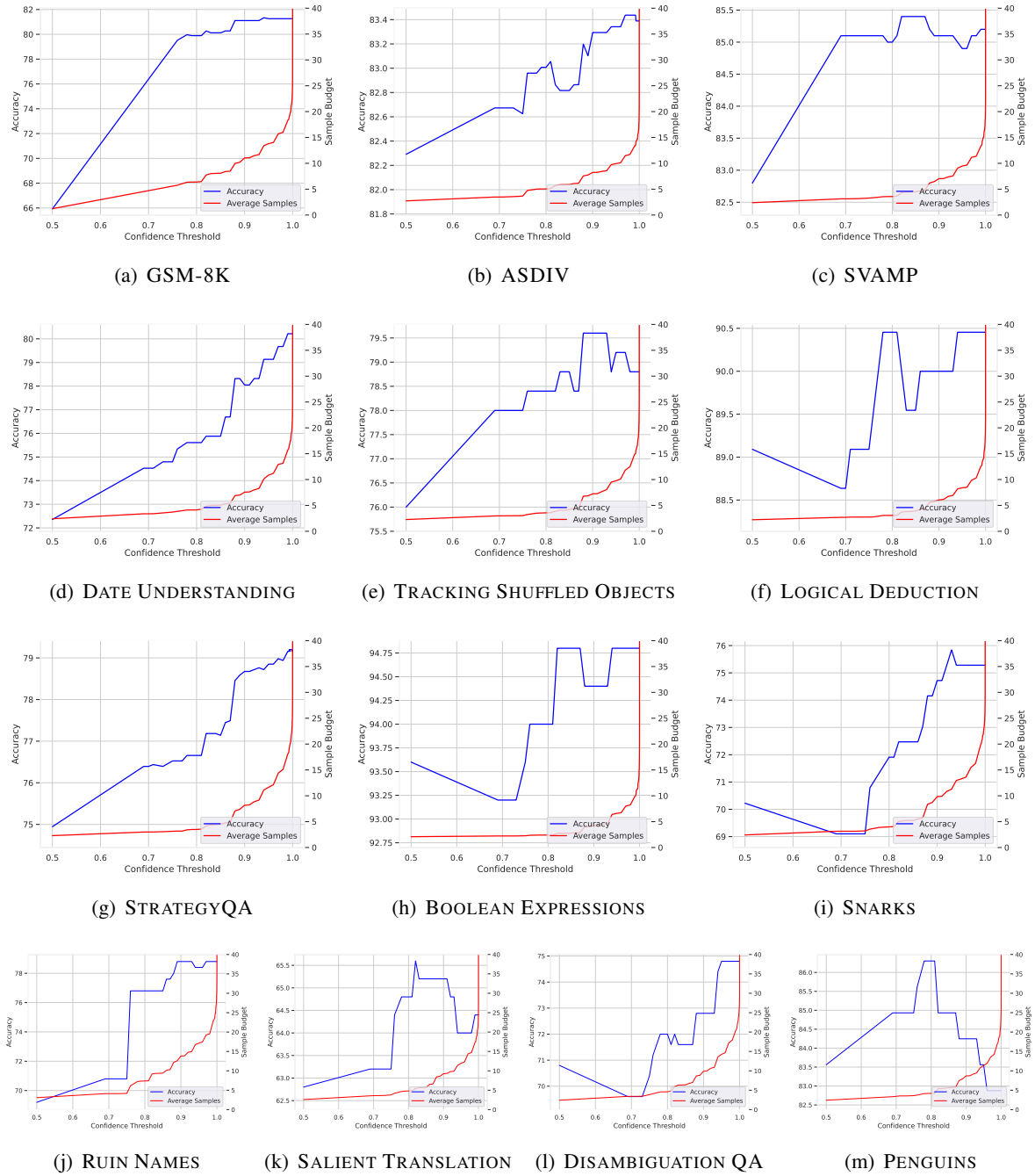
Defining region of integration: $\mathcal{S}(p_1')$ Next, for computation of Equation (3), we need to precisely calculate the limits of each integration such that they represent the region $\mathcal{S}(p_1')$. We do so by noting the following constraints on p_i: 1.) The $p_i = 0$ is valid $\forall 2 \leq i \leq m$, 2.) Given $\{p_m, p_{m-1} \dots p_{i+1}$ are fixed and in region $\mathcal{S}(p_1')$,

$p_i < \frac{1-\sum_{j=i+1}^{m} p_j}{2}$ as else $p_i \geq p_1$ which is not allowed, 3.) Since $p_1 > \max_{j=i+1}^{m} p_j$, so $p_i < 1 - \sum_{j=i+1}^{m} p_j - \max_{j=i+1}^{m} p_j$, as else the $\vec{p}$, will lie outside the $(m-1)$-simplex, which is invalid. The first condition makes the lower limit for each integration as 0, and minimum of condition 2 and condition 3 gives upper bound (limit) on each of the integrations.

BETA stopping criteria Due to $m-1$ dimensions integrations involved, with m often getting larger than 10, computing Equation (3) is not efficient. Instead, we observe that establishing the majority of p_1 over the next largest probability, p_2, is sufficient for our purpose. Then, pdf simplifies to BETA distribution with parameters v_1+1, v_2+1, and Equation (3) simplifies to:

$$\int_0^{0.5} p_2^{v2} \cdot (1-p_2)^{v_1} dp_2 \tag{4}$$

We use scipy library in python numerically compute the above equation.



(a) GSM-8K (b) ASDIV (c) SVAMP

(d) DATE UNDERSTANDING (e) TRACKING SHUFFLED OBJECTS (f) LOGICAL DEDUCTION

(g) STRATEGYQA (h) BOOLEAN EXPRESSIONS (i) SNARKS

(j) RUIN NAMES (k) SALIENT TRANSLATION (l) DISAMBIGUATION QA (m) PENGUINS

Figure 3: Impact of Confidence Threshold (C_{thresh}) on Adaptive-Consistency: As C_{thresh} varies, the accuracy of Adaptive-Consistency increases gradually, eventually plateauing. Initially, the average number of generations also increases gradually but then sharply climbs, reflecting the accuracy-confidence trade-off. The trend is observed across almost consistently across all datasets.