

Can Large Language Models Always Solve Easy Problems if They Can Solve Harder Ones?

Zhe Yang¹, Yichang Zhang², Tianyu Liu², Jian Yang², Junyang Lin²
Chang Zhou², Zhifang Sui¹

¹State Key Laboratory of Multimedia Information Processing,
School of Computer Science, Peking University

²Alibaba Group

{yz_young, szf}@pku.edu.cn

{yichang.zyc, tianyu0421, ericzhou.zc}@alibaba-inc.com

Abstract

Large language models (LLMs) have demonstrated impressive capabilities, but still suffer from inconsistency issues (e.g. LLMs can react differently to disturbances like rephrasing or inconsequential order change). In addition to these inconsistencies, we also observe that LLMs, while capable of solving hard problems, can paradoxically fail at easier ones. To evaluate this hard-to-easy inconsistency, we develop the ConsisEval benchmark, where each entry comprises a pair of questions with a strict order of difficulty. Furthermore, we introduce the concept of consistency score to quantitatively measure this inconsistency and analyze the potential for improvement in consistency by relative consistency score. Based on comprehensive experiments across a variety of existing models, we find: (1) GPT-4 achieves the highest consistency score of 92.2% but is still inconsistent to specific questions due to distraction by redundant information, misinterpretation of questions, etc.; (2) models with stronger capabilities typically exhibit higher consistency, but exceptions also exist; (3) hard data enhances consistency for both fine-tuning and in-context learning. Our data and code will be publicly available on GitHub.¹

1 Introduction

With the increases in pre-training corpora and the number of parameters (Radford et al., 2018, 2019; Brown et al., 2020), large language models (LLMs) have shown remarkable performance across various natural language processing (NLP) tasks, even generating expert-level responses to user queries. The extraordinary capabilities of LLMs hold potential for further real-world applications (Wang et al., 2023c; Guo et al., 2023; Driess et al., 2023), which necessitate higher requirements for model trustworthiness (Wang et al., 2023a; Li

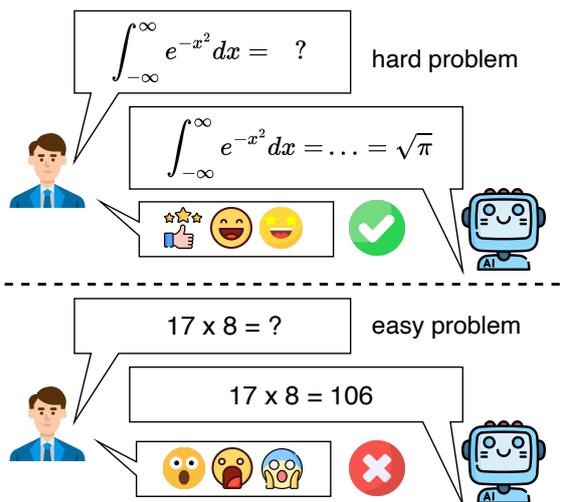


Figure 1: A hard-to-easy inconsistency case of LLMs. A counter-intuitive phenomenon occurs when an LLM, which can solve a harder problem, surprisingly goes wrong on an easier problem.

et al., 2023a; Sun et al., 2024a) and consistency (Jang and Lukasiewicz, 2023; Elazar et al., 2021).

However, LLMs still suffer from inconsistency issues: semantically equivalent queries (Elazar et al., 2021; Raj et al., 2023) and insignificant order changes of inputted contents (Wang et al., 2023b) can lead to divergent outcomes; LLMs can also behave differently in the generation versus validation of the same content (Li et al., 2023b); moreover, logical transformations like negation and symmetry can also induce inconsistent behaviors (Jang et al., 2022). In addition to previous work, we also find LLMs able to solve hard problems surprisingly fail to solve easier ones (as shown in Figure 1), suffering from the hard-to-easy inconsistency. Unlike LLMs, humans are naturally consistent reasoners, and it is undisputed that an individual proficient in calculus can easily address simpler arithmetic problems. However, why this difference exists is still unknown and relevant research to explore hard-to-easy consistency of LLMs is still lacking.

¹<https://github.com/QwenLM/ConsisEval>

To systematically evaluate this consistency of LLMs, we develop **ConsisEval**, a **Hard-to-easy Consistency Evaluation Benchmark**, through automatic generation and human annotation. ConsisEval encompasses data from three domains: instruction following, code, and mathematics, each entry consisting of a pair of questions with a strict order of difficulty. Considering the absence of an off-the-shelf metric, we propose a new metric consistency score, which is defined as the conditional probability of a model correctly answering easy questions provided that it has correctly answered harder ones, for quantitative assessment of consistency from a probabilistic stance. Further, to analyze the potential for improvement in consistency if model capability remains unchanged, we introduce the concept of relative consistency score. The calculation of our metrics relies on the probability of a model answering each question correctly through a single sampling, for which we design two probability estimation methods.

Based on our benchmark and metrics, we conduct extensive experiments on various LLMs. Among evaluated models, GPT-4 (Achiam et al., 2023) achieves the highest CS of 92.2%, demonstrating notable hard-to-easy consistency. Nonetheless, GPT-4 also exhibits inconsistent behaviors to specific prompts due to distraction by redundant information, misinterpretation of questions, etc. Further, we find models with stronger capabilities typically exhibit higher consistency, but exceptions where powerful models demonstrate poor consistency also exist. Additionally, we discover that models show higher consistency when trained under hard data than easy data, and that holds the same under few-shot setting (in-context learning with harder demonstration examples shows better consistency).

We summarize our contributions as follows:

1. To the best of our knowledge, we are the first to systematically study the hard-to-easy consistency of LLMs and establish a benchmark to evaluate this consistency.
2. We propose metrics grounded in probabilistic theory to quantitatively measure the hard-to-easy consistency, along with probability estimation methods for metric computation.
3. Based on our benchmark and metrics, we conduct extensive experiments across a variety of

LLMs and draw some conclusions that may benefit future research.

2 ConsisEval Benchmark

To systematically evaluate the hard-to-easy consistency of LLMs, we develop ConsisEval with data from code, mathematics, and instruction-following domains, which are widely considered to be difficult but of significant importance for LLMs (Wei et al., 2021; Cobbe et al., 2021a,b; Zhou et al., 2023). Different from traditional benchmarks in which data are usually individual, there are only pairwise data in ConsisEval: one datum is comprised of two questions (an easy question and a harder one) with a strict order of difficulty, and we present some example data from ConsisEval in Table 5. To construct ConsisEval, we collect easy data from some established public datasets (§2.1); then we acquire hard data through automatic generation by GPT-4 and human annotation (§2.2), and this process is shown in Figure 2.

2.1 Easy Data Collection

Mathematics easy data are collected from GSM8K (Cobbe et al., 2021a), a linguistically diverse collection of high-quality grade school math word problems crafted by human problem writers. The difficulty of these problems varies, requiring from 2 to 8 steps to solve, and solving these problems typically requires a series of fundamental calculations employing basic arithmetic operations ($+$ $-$ \times \div). To prevent easy data from being too difficult to be further improved in terms of difficulty, we only select the problems requiring 3 steps to solve in the test set of GSM8k as our easy data in the mathematics domain (298 entries).

Code easy data are collected from HumanEval (Cobbe et al., 2021b), a benchmark aiming at evaluating the capability of LLMs to generate standalone Python functions from docstrings. For each coding problem, a check function containing some test cases is provided for automatic correctness evaluation of code samples. Since HumanEval is relatively small, we select all of the data in HumanEval as our easy data in code domain (164 entries).

Instruction-following easy data are collected from IFEval (Zhou et al., 2023), a benchmark comprised of various instructions for LLMs to follow. Each instruction contains 1-3 verifiable constraints (e.g. maximum number of words in response or

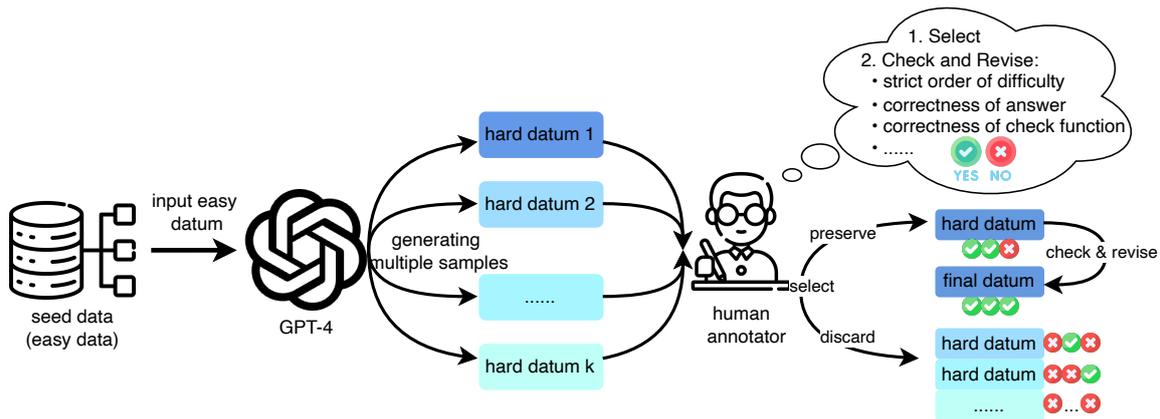


Figure 2: The hard data collection process of ConsisEval. An easy datum is fed into GPT-4 with a well-designed prompt and multiple hard data candidates are sampled. Human annotators select the one of best quality, then check and revise the sample to make it fit our criteria.

the appearance of specific keywords in response), whose correctness can be automatically evaluated by rule-based check functions. We only select the instructions with only one constraint as our easy data in instruction-following domain (270 entries).

2.2 Hard Data Collection

To build our pairwise dataset in which a strict order of difficulty is guaranteed for each pair of easy and hard problems, all of the hard data are modified from easy data. We employ a semi-automatic pipeline that integrates the automatic generation of GPT-4 with human annotation to acquire hard data, and the whole workflow is illustrated in Figure 2. Compared to traditional methods that rely solely on manual annotation, our semi-automatic approach can significantly alleviate the workload of human annotators.

Automatic generation. Considering the remarkable performance of GPT-4 on various text generation tasks, we employ GPT-4 as a strong modified data generator to acquire our hard data candidates for human annotators to choose from. To make GPT-4 understand our criteria better, we insert easy data into a well-designed prompt template (shown in Appendix J) before feeding them into GPT-4. Taking the code domain as an example, the prompt consists of 5 parts: (1) the #Instruction# part articulates the information we want GPT-4 to know, including but not limited to definition of our modification task, composition of a datum, and guarantee of strict order of difficulty; (2) the #Demonstrations# part requires insertion of easy and hard data pairs as demonstrations; (3) finally, an easy datum targeted for modification is decomposed into three

Easy Question: John has 2 houses with 3 bedrooms each. How many bedrooms are there in total?
Hard Question: John has 2 houses with 3 bedrooms each. Each bedroom has 2 windows. How many windows are there in total?

Table 1: An example question pair with a strict order of difficulty. Green text denotes the common part of questions and blue text denotes the additional part of hard question.

components and inserted into the #Problem#, #Answer#, and #Check Function# parts, respectively.

Human annotation. Though we have endeavored to request GPT-4 to generate hard data that fully adheres to our criteria through a well-designed prompt, the generated contents may still not meet our standards (e.g. some samples lack a strict order of difficulty and check functions of some other samples are incorrect). To address potential issues in generated samples, we have engaged human annotators to inspect, select, and revise these samples. Firstly, the annotators are required to select the sample of the highest quality from multiple candidates and discard all the other samples. To ensure compliance with our criteria, the selected sample is checked from two aspects:

1. **Strict order of difficulty:** the steps or knowledge (or ability) required to solve an easy problem should be a proper subset of those for the hard problem (example shown in Table 1).
2. **Correctness:** the standard answer or check function (for automatic judgment of model-

generated answers) should be correct.

If one sample fails to comply with our criteria during the checking process, the annotators will revise it to ensure full alignment with our standards.

3 Evaluation Metrics

Firstly, we formulate the evaluation problem and introduce mathematical notations in §3.1. Considering that there is no off-the-shelf metric to utilize, then we propose a new metric named Consistency Score (§3.2) to measure the hard-to-easy consistency quantitatively. Further, we introduce the concept of Relative Consistency Score (§3.3) to analyze the potential for improvement in consistency. We model sampling an answer from an LLM for a given question as a stochastic process, wherein the answer is correct with a fixed probability p . The computation of our metrics requires access to p , and §3.4 discusses how to estimate p by maximum likelihood estimation.

3.1 Problem Formulation and Notation

Initially, we have a partially ordered set comprising N pairs of data, denoted as $A \odot B = \{(a_1, b_1), (a_2, b_2), \dots, (a_N, b_N)\}$, where $A = \{a_1, a_2, \dots, a_N\}$ represents a set of easy questions, and $B = \{b_1, b_2, \dots, b_N\}$ constitutes a set of hard questions. A stringent guarantee exists that the difficulty order satisfies $a_i < b_i$, for b_i is derived from a_i by increasing the difficulty level. For a given question a_i (or b_i), the model generates a correct answer through a single temperature-based sampling with probability $P(a_i)$ (or $P(b_i)$). We employ \hat{P} to symbolize estimates (e.g. $\hat{P}(a_i)$ represents the estimate of the true value $P(a_i)$). For convenience, all of the notations mentioned and their meanings are shown in Appendix A.

3.2 Consistency Score

Can large language models solve easy problems if they can solve harder ones? To answer this question from a probabilistic perspective, we introduce a metric termed **Consistency Score (CS)**, which is the conditional probability of a model correctly answering easy questions given that it has correctly answered harder ones. The higher CS indicates the lower probability humans encounter inconsistency phenomena when using LLMs, so CS is almost equal to human perceptions of model consistency. Let $P(a|b)$ be the conditional proba-

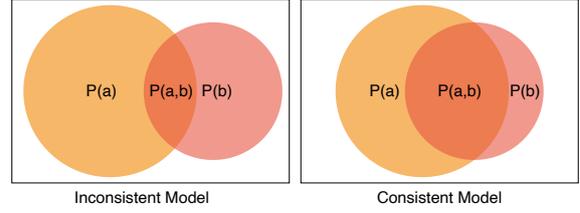


Figure 3: Venn diagram for consistent/inconsistent models in complete probability space. The orange, red circles and their overlap area denote the probability of a model correctly answering easy questions, hard questions, and both respectively. the overlap area of consistent models is much larger than that of inconsistent models.

bility of a model correctly answering a given that it has answered b correctly, and we have:

$$CS = P(a|b) = \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i)}{\sum_{i=1, \dots, N} P(b_i)} \quad (1)$$

The detailed derivation of CS is shown in Appendix B. To intuitively understand the distinctions between consistent and inconsistent models and better illustrate CS, we present a Venn diagram in Figure 3. The more consistent a model is, the larger overlap area $P(a, b)$ in Venn diagram, and consequently the higher CS of the model. Fundamentally, CS represents the ratio of $P(a, b)$ to $P(b)$.

3.3 Relative Consistency Score

In addition to CS that directly reveals consistency probability of LLMs, we also endeavor to analyze the potential for improvement in consistency if model capability remains unchanged. To analyze what the CS of an evaluated model M_0 should be if it behaves extremely consistently/inconsistently, we formally define a model set $\Omega = \{M_0, M_1, \dots\}$ (detailed definition shown in Appendix C) in which models possess similar capabilities to M_0 and derive the upper and lower bounds of CS (denoted as CS_{upp} and CS_{low}) among these hypothetical models. Based on these bounds, we propose **Relative Consistency Score (RCS)** (as shown in Figure 4) to indicate the potential for improvement in consistency, and low RCS can reveal high potential for improvement in CS. The RCS is given by:

$$RCS = \frac{CS - CS_{low}}{CS_{upp} - CS_{low}} \quad (2)$$

According to the definition of Ω and rearrangement inequality, we can obtain strict mathematics

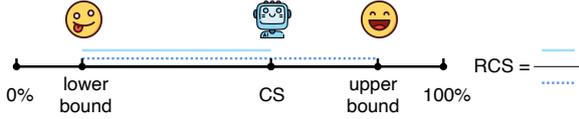


Figure 4: Visualized expression of relative consistency score.

bounds. However, these bounds are empirically too loose, and thus we utilize tighter bounds derived from two heuristics:

$$CS_{low} = \frac{\sum_{i=1, \dots, N} P(a_i)}{N}, \quad (3)$$

$$CS_{upp} = \frac{\sum_{i=1, \dots, N} (P(b_i) + \hat{\mu}) P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}, \quad (4)$$

where $\hat{\mu} = \frac{\sum_{i=1, \dots, N} (P(a_i) - P(b_i))}{N}$, and the derivation of boundaries and discussion are shown in Appendix D.

3.4 Probability Estimation

For a given question a_i and a given model, the probability $P(a_i)$ that the model produces a correct answer in a single sampling is an unknown constant. We propose two methods for estimating $P(a_i)$ based on repeated sampling. For open-source models that can be deployed locally, estimate $\hat{P}(a_i)$ is obtained by sampling multiple answers independently. For proprietary models that require payment for API calls, an early stopping strategy is employed during answer sampling to obtain estimate $\hat{P}(a_i)$ with fewer API calls.

Multiple Sampling Estimation For a given question a_i , answers are sampled m times to obtain a sequence $a_i^1, a_i^2, \dots, a_i^m$. If the model generates a correct answer on the j th sampling, we denote $a_i^j = 1$; otherwise, $a_i^j = 0$. In this scenario, a_i^j follows a Bernoulli distribution, and $\sum_{j=1, \dots, m} a_i^j$ follows a Binomial distribution (i.e. $\sum_{j=1, \dots, m} a_i^j \sim B(m, P(a_i))$). It can be derived that the maximum likelihood estimate of $P(a_i)$ (refer to Appendix E.1 for the derivation details):

$$\hat{P}(a_i) = \frac{\sum_{j=1, \dots, m} a_i^j}{m} \quad (5)$$

Early Stopping Estimation Estimating through multiple sampling necessitates generating a multitude of answers for the same question (e.g. in §4 we utilize Llama2-7b-chat to sample 20 answers for a question). However, considering the high payment

for the API calls and the typically high accuracy of closed-source models, an early stopping technique is employed to estimate with fewer API calls.

Details of early stopping strategy: Initially, we set the minimum and maximum number of sampling times k_{min} and k_{max} . For a given question a_i , initially, k_{min} answers are sampled. If at least one correct answer exists in these answers, the sampling process will be terminated; otherwise, sampling will continue repeatedly until a correct answer appears for the first time. Besides, the sampling procedure will be forcibly terminated if a correct answer still does not emerge after sampling k_{max} answers.

The total number of samples in the above process and the number of correct answers are denoted as k and k_c , respectively. The maximum likelihood estimation of $P(a_i)$ can be derived as follows (refer to Appendix E.2 for the derivation details):

$$\hat{P}(a_i) = \frac{k_c}{k} \quad (6)$$

Besides, we also show the pseudo-code of Early Stopping Estimation, discuss the trade-off, and compare these two methods in Appendix E.3.

4 Experiments

4.1 Experimental Setup

For closed-source models, we evaluate GPT-4 Turbo ² (Achiam et al., 2023), GPT-3.5 Turbo ³, Qwen Max (Bai et al., 2023), and Claude-3 Opus ⁴, which can only be accessed via API calls. For open-source models, we experiment on Llama2-(7B,13B,70B) (Touvron et al., 2023), Llama3-(8B,70B) (AI@Meta, 2024), Qwen-1.5-(7B,14B,72B) (Bai et al., 2023), ChatGLM3-6B (Du et al., 2022), DeepseekLLM-(7B,67B) (DeepSeek-AI, 2024), Mistral-7B (Jiang et al., 2023), Baichuan2-(7B,13B) (Baichuan, 2023), and Yi-6B (Young et al., 2024). Most of these open-source models are released with two versions, the pre-trained base model and the chat model (based model + instruction tuning and alignment), and we focus our evaluation solely on chat models. More implementation details can be found in Appendix G.1.

²gpt-4-0125-preview

³gpt-3.5-turbo-0125

⁴claude-3-opus-20240229

Models	Code			Instruction Following			Maths			Avg CS
	Hard	Easy	CS	Hard	Easy	CS	Hard	Easy	CS	
GPT-4 Turbo	80.8	85.5	88.1	74.4	84.2	91.8	92.8	96.2	96.8	92.2
GPT-3.5 Turbo	62.3	71.4	81.2	53.0	76.1	88.6	65.6	86.9	90.7	86.8
Claude-3 Opus	79.0	81.1	85.5	78.0	87.7	93.4	93.7	96.5	96.6	91.8
Qwen Max	66.9	75.0	82.4	53.2	74.3	89.6	86.8	95.2	96.8	89.6
Llama3-70B-Instruct	69.2	73.9	84.3	74.7	86.7	94.4	80.8	94.9	96.9	91.9
Llama2-70B-Chat	20.7	34.5	74.7	36.3	56.6	81.0	23.2	70.5	83.7	79.8
Qwen1.5-72B-Chat	47.0	62.3	79.4	34.9	56.5	87.3	75.7	90.6	93.6	86.8
DeepseekLLM-67B-Chat	56.9	68.6	77.9	29.6	52.5	83.8	66.9	90.2	94.8	85.5
Llama2-13B-Chat	14.2	20.2	61.9	24.9	48.3	84.2	8.1	48.6	67.2	71.1
Qwen1.5-14B-Chat	36.1	51.4	74.6	29.3	55.4	83.6	58.2	82.6	90.7	83.0
Baichuan2-13B-Chat	15.7	21.5	59.1	13.0	31.0	63.3	14.2	48.6	65.8	62.7
Llama3-8B-Instruct	41.7	53.6	71.4	62.6	78.5	87.9	38.3	77.8	87.4	82.2
Llama2-7B-Chat	10.2	14.9	63.1	26.6	43.7	75.6	4.7	34.3	57.9	65.5
Qwen1.5-7B-Chat	28.6	40.9	68.4	21.8	47.2	82.5	34.7	68.6	83.6	78.2
ChatGLM3-6B	24.1	50.8	68.5	16.4	36.6	64.7	16.7	64.4	83.9	72.4
DeepseekLLM-7B-Chat	26.6	40.3	62.6	24.1	47.5	71.0	20.8	69.0	84.8	72.8
Mistral-7B-Instruct	20.3	28.4	57.0	37.1	60.8	84.3	11.6	51.8	67.4	69.6
Yi-6B-Chat	8.7	13.2	49.3	15.4	37.4	76.0	10.2	50.9	69.7	65.0
Baichuan2-7B-Chat	8.8	12.4	43.0	12.1	29.9	60.0	5.0	28.4	50.1	51.0

Table 2: Consistency evaluation results. A variety of LLMs are evaluated on code, instruction-following, and maths domains. On each domain, we report consistency score (CS), accuracy (%) on hard set and easy set (denoted as **Hard** and **Easy**). We also report the average consistency score (**Avg CS**) among three domains.

4.2 Main Results

As illustrated in Table 2, we evaluate the hard-to-easy consistency of LLMs on ConsisEval and report the consistency score (CS) in three domains and the average consistency score (Avg CS). The accuracy (%) on easy and hard sets (indicating model capability) is also shown for comparison. Among the evaluated LLMs, GPT-4 Turbo showcases outstanding performance in three domains and achieves the highest Avg CS of 92.2%, closely followed by Claude-3 Opus with an Avg CS is 91.8%. Llama3-(8B,70B)-Instruct exhibit high capability and consistency among open-source models, superior to other models of comparable size. For comparison, CS of humans is theoretically 100% if not take carelessness cases into consideration. Therefore, the potential for further improvement in consistency still exists.

We also observe a strong correlation between capability and consistency of LLMs. For example, Kendall rank correlation coefficient between accuracy on hard set and CS across all evaluated LLMs on code domain is 0.801 (further discussion is provided in Appendix G.2). However, higher capability does not necessarily lead to higher consistency (e.g. in math domain, Claude-3 Oplus outperforms GPT-4 Turbo in capability, yet exhibits a lower consistency). Additionally, empirical results also show CS is always larger than easy accuracy across

all evaluated models, suggesting that answering hard questions correctly benefits answering easy questions.

4.3 Relative Consistency Analysis

To analyze the potential for improvement in consistency, we attempt to compare the consistency of an evaluated model with other hypothetical models of similar capability ("capability" can be intuitively but not strictly understood as "performance on accuracy", with a formal definition provided in Appendix C). For each evaluated model, we present its CS, upper and lower bounds of CS along with the relative consistency score (RCS), which can be utilized to analyze potential improvement in consistency within the current capability.

The experimental results in code domain are presented in Figure 5, while the comprehensive results across all domains can be found in Appendix G.3. In code domain, we find that while GPT-4 Turbo exhibits high consistency with a CS of 88.1%, there is still considerable potential for improvement compared to the upper bound 93.0%. Furthermore, the RCS for GPT-4 Turbo is 34.8%, indicating a relative improvement potential of 65.2%. Conversely, Llama2-70B-Chat, despite showing a low CS of merely 74.7%, achieves an RCS of 81.5%, indicating notable consistency within its current capability.

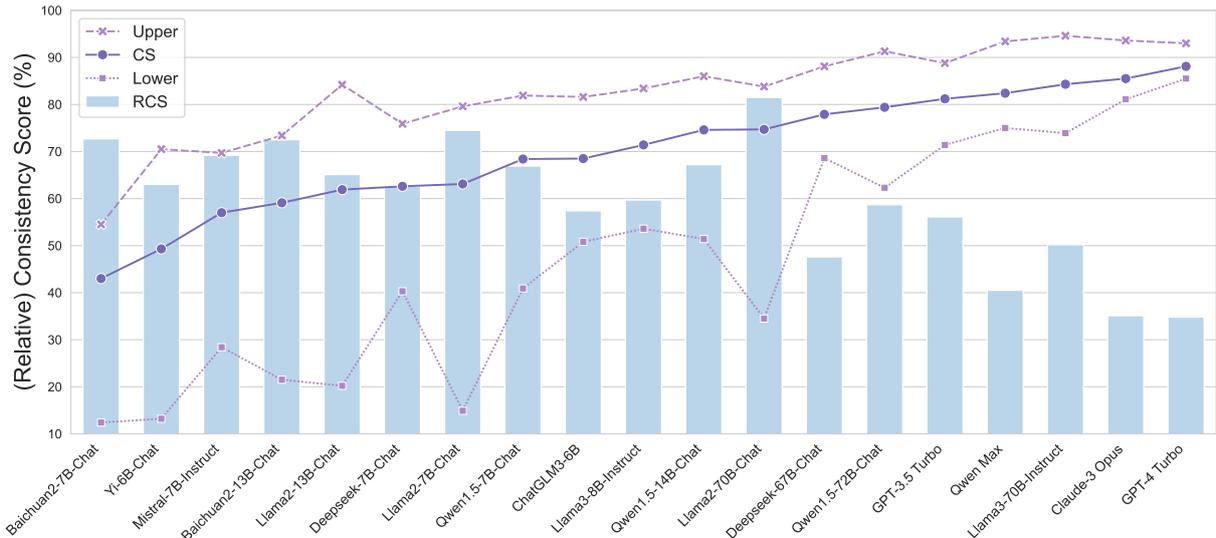


Figure 5: Relative consistency results in code domain (shown in ascending order of CS). Except for showing RCS for each evaluated model in a bar, we also show CS, upper and lower bounds of CS in lines of different colors for comparison.

5 Analysis

5.1 Hard Training Data Benefits Consistency

To investigate the impact of the ratio between easy and hard data in the training set on model consistency, we select 2,500 easy and 2,500 hard entries from the training set of gsm8k (Cobbe et al., 2021a) based on the number of reasoning steps. We adjust the ratio between easy and hard data while keeping the total amount constant at 2,500 entries to construct a series of training sets with varying proportions. We then fine-tune Llama3-8B on these training sets (each group is repeated three times under different random seeds with Lora (Hu et al., 2021)) and observe the consistency behaviors. As shown in Figure 6, both the CS and RCS generally increase as the proportion of hard data increases, suggesting that hard training data benefits model consistency. Moreover, compared to a dataset composed entirely of hard data, a combination of 80% hard and 20% easy data yields better consistency, indicating proper easy data also contributes to enhancing model consistency.

5.2 Hard ICL Examples Benefits Consistency

Similar to §5.1, we also explore the impact of easy and hard in-context learning (ICL) (Brown et al., 2020; Dong et al., 2022; Yang et al., 2023) demonstration examples on model consistency. The experiments are under 1-4 shot setting, and for each setting we randomly select 20 easy and 20 hard ICL examples to evaluate the consistency of Llama-8B-

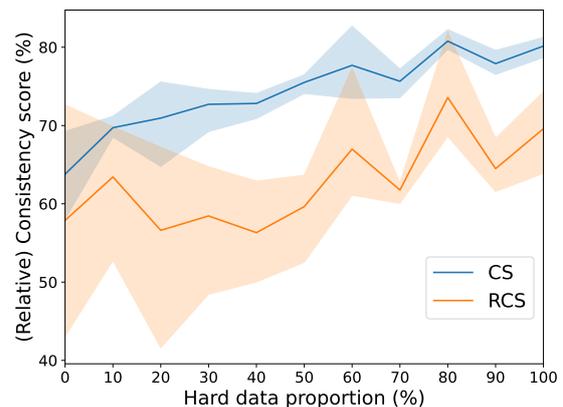


Figure 6: Consistency of models fine-tuned on training sets of different proportions of easy and hard data. Fine-tuned models show higher consistency with more hard training data.

Instruct. As shown in Figure 7, hard examples display better consistency than easy ones, and model consistency progressively increases with the number of shots.

5.3 Case Study: Why are LLMs Inconsistent?

Through investigations on math inconsistency cases (shown in Appendix I), where the probability of solving hard problems is higher than that of easier ones, we find even state-of-the-art GPT-4 still behaves inconsistently due to the following reasons: (1) **Distracted by redundant information:** As the case shown in Table 6, for the easy question with redundant conditions, GPT-4 incorrectly proceeds with an additional step after having already

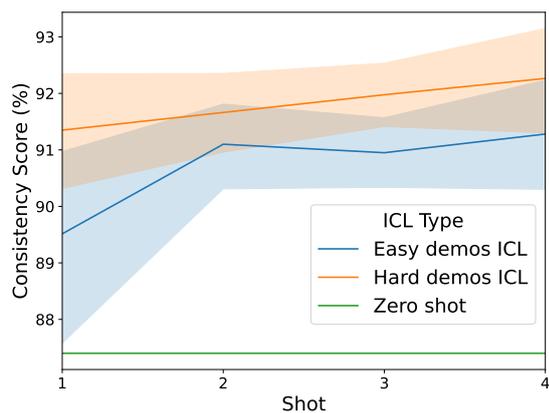


Figure 7: Consistency behavior of ICL with easy and hard examples under 1-4 shot settings. ICL with harder examples shows higher consistency.

arrived at the correct answer, leading to a final incorrect result. (2) **Data mismatch**: As the case shown in Table 7, GPT-4 could accurately analyze the usage of "dancing time on Tuesday" for computation, but it erroneously utilizes "dancing time on Thursday" when conducting computation. (3) **Misinterpretation of questions**: As the case shown in Table 8, the easy question requires finding the "cost of travel," GPT-4 misinterprets the requirement as the "cost of tickets for travel". (4) **Logical error (Off-by-one error)**: As the case shown in Table 9, the initial state should be recorded as "Day 0" in the easy question, but GPT-4 erroneously began recording from "Day 1". (5) **Computational error**: As the case shown in Table 10, GPT-4 encounters computational errors while solving an equation for the easy question. Superficially, the inconsistency of GPT-4 stems from the occurrence of the above mistakes on the easy questions but not on the corresponding hard questions. However, deeper underlying reasons remain unclear.

6 Related Work

Consistency of LLMs Consistency constitutes an important part of trustworthiness and reliability (Wang et al., 2023a; Li et al., 2023a; Chai et al., 2024; Liu et al., 2023) of LLMs. Humans are inherently consistent reasoners, but LLMs suffer from inconsistency problems. Wang et al. (2023b) find LLMs, when acting as evaluators, show inconsistency with insignificant order changes of evaluation content; Li et al. (2023b) observe that LLMs also show inconsistency when generating and validating the same knowledge; Elazar et al. (2021); Raj et al. (2023) endeavor to evaluate and enhance the consistency

with semantically identical expressions; Jang et al. (2022); Jang and Lukasiewicz (2023) evaluate and analyze consistency to logical transformations, such as negation and symmetry. Different from perspectives presented in previous works, our research focuses on the hard-to-easy consistency of LLMs.

Easy-to-Hard Generalization Hupkes et al. (2020); Xu and Wang (2024) study the generalization ability of models trained on simple elements to complex element combinations; likewise, Burns et al. (2023); Hase et al. (2024); Sun et al. (2024b) find models trained on easy data exhibit strong generalization capabilities to hard data. However, we have observed that training models solely on easy data can lead to inconsistent behaviors.

Leveled Evaluation Liu et al. (2024); Xu et al. (2024a) hierarchically evaluate the capability of LLMs to solve problems of different difficulty levels by data categorized from easy to hard. Similarly but differently, we evaluate the consistency of LLMs by pairwise hard-to-easy data. Unlike previous work whose difficulty level is roughly divided by the number of reasoning steps (Hase et al., 2024), the difficulty order in our work is constrained to pairwise questions and more strict.

7 Conclusion

We observe an anomalous phenomenon where LLMs able to solve hard problems paradoxically fail at easier ones. To evaluate this hard-to-easy inconsistency, we construct ConsisEval by automatic generation and human annotation. Furthermore, we propose consistency score to measure this inconsistency quantitatively and relative consistency score to analyze the potential for improvement in consistency. Based on our dataset and metrics, we conduct comprehensive experiments on numerous existing models, finding that there are exceptions where some powerful models demonstrate poor consistency, though models with stronger capabilities usually exhibit higher consistency. Case study shows though state-of-the-art GPT-4 achieves the highest CS of 92.2%, still suffers from inconsistency due to distraction by redundant information, misinterpretation of questions, etc. Besides, we also find hard data benefits consistency for both fine-tuning and ICL. Our benchmark and metrics can facilitate research in consistency of LLMs, ultimately paving the way for building more trustworthy and reliable AI in the future.

Limitations

Our evaluation requires repeated sampling for the same question to estimate the probability, which is more computationally expensive than traditional non-probability evaluation. Our metric CS can only reflect the overall consistency of a model and can hardly identify to which types of problems it is more inconsistent. We also find different models behave inconsistently to totally different questions, and identifying these questions for a given model still requires human efforts in case studies.

Data contamination (or data leakage) (Magar and Schwartz, 2022; Xu et al., 2024b) can affect our evaluation. As detailedly discussed in Appendix F, leakage of easy and hard data can lead to higher and lower CS, respectively. Considering that easy data are from public data and thereby suffer from a higher risk of data leakage (e.g. Achiam et al. (2023) reports 25% of HumanEval has been contaminated in their training data), model consistency can be overrated.

Our evaluation does not include human results. Theoretically, consistency of humans should equate to 100%, yet incorrectness on easy questions caused by carelessness can diminish this consistency. Human evaluation results can vary due to the variance of carelessness among individuals; besides, having humans complete all questions in ConsisEval is exceedingly time-consuming. Therefore, determining the human level consistency for LLMs as a reference needs more discussion and exploration.

Our benchmark focuses on evaluating the hard-to-easy consistency of LLMs but does not investigate the underlying reasons and how inconsistency comes into being. The knowledge acquisition process of humans and LLMs is totally different, and humans are inherently consistent reasoners yet LLMs are not. Will pre-training and fine-tuning paradigm of LLMs necessarily lead to inconsistency? Further discussion and exploration is needed. Though our preliminary findings suggest that hard training data can mitigate this inconsistency, how to solve this inconsistency problem is still unknown, and we leave it to future work.

Ethical Considerations

The easy part of our benchmark originates from publicly available datasets, which is allowed for research usage. Our dataset encompasses code, maths, and instruction-following domains, which

are safe and can hardly be utilized in harmful ways. Besides, the evaluated LLMs are all publicly available by either parameters or API calls. Therefore, we do not anticipate any ethical concerns in our research.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- AI@Meta. 2024. [Llama 3 model card](#).
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- Baichuan. 2023. [Baichuan 2: Open large-scale language models](#). *arXiv preprint arXiv:2309.10305*.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. 2023. [Weak-to-strong generalization: Eliciting strong capabilities with weak supervision](#). *arXiv preprint arXiv:2312.09390*.
- Linzheng Chai, Jian Yang, Tao Sun, Hongcheng Guo, Jiaheng Liu, Bing Wang, Xinnian Liang, Jiaqi Bai, Tongliang Li, Qiyao Peng, and Zhoujun Li. 2024. [xcot: Cross-lingual instruction tuning for cross-lingual chain-of-thought reasoning](#). *arXiv preprint arXiv:2401.07037*, abs/2401.07037.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021a. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021b. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- DeepSeek-AI. 2024. [Deepseek llm: Scaling open-source language models with longtermism](#). *arXiv preprint arXiv:2401.02954*.
- Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

- Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.
- Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. [GLM: General language model pretraining with autoregressive blank infilling](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, Dublin, Ireland. Association for Computational Linguistics.
- Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhishava Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. [Measuring and improving consistency in pretrained language models](#). *Transactions of the Association for Computational Linguistics*, 9:1012–1031.
- Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, Xu Shi, Tiejiao Zheng, Liangfan Zheng, Bo Zhang, Ke Xu, and Zhoujun Li. 2023. [OWL: A large language model for IT operations](#). *CoRR*, abs/2309.09298.
- Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegrefe. 2024. [The unreasonable effectiveness of easy training data for hard tasks](#). *arXiv preprint arXiv:2401.06751*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. [Compositionality decomposed: How do neural networks generalise?](#) *J. Artif. Intell. Res.*, 67:757–795.
- Myeongjun Jang, Deuk Sin Kwon, and Thomas Lukasiewicz. 2022. [BECEL: Benchmark for consistency evaluation of language models](#). In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3680–3696, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.
- Myeongjun Jang and Thomas Lukasiewicz. 2023. [Consistency analysis of ChatGPT](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15970–15985, Singapore. Association for Computational Linguistics.
- 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*.
- Bo Li, Peng Qi, Bo Liu, Shuai Di, Jingen Liu, Jiquan Pei, Jinfeng Yi, and Bowen Zhou. 2023a. Trustworthy ai: From principles to practices. *ACM Computing Surveys*, 55(9):1–46.
- Xiang Lisa Li, Vaishnavi Shrivastava, Siyan Li, Tatsunori Hashimoto, and Percy Liang. 2023b. Benchmarking and improving generator-validator consistency of language models. *arXiv preprint arXiv:2310.01846*.
- Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. [Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark](#). *arXiv preprint arXiv:2405.12209*.
- Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo Hao Cheng, Yegor Klochov, Muhammad Faaiz Taufiq, and Hang Li. 2023. Trustworthy llms: a survey and guideline for evaluating large language models’ alignment. *arXiv preprint arXiv:2308.05374*.
- Inbal Magar and Roy Schwartz. 2022. [Data contamination: From memorization to exploitation](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 157–165, Dublin, Ireland. Association for Computational Linguistics.
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Harsh Raj, Vipul Gupta, Domenic Rosati, and Subhabrata Majumdar. 2023. Semantic consistency for assuring reliability of large language models. *arXiv preprint arXiv:2308.09138*.
- Lichao Sun, Yue Huang, Haoran Wang, Siyuan Wu, Qihui Zhang, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, et al. 2024a. Trustllm: Trustworthiness in large language models. *arXiv preprint arXiv:2401.05561*.
- Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. 2024b. [Easy-to-hard generalization: Scalable alignment beyond human supervision](#). *arXiv preprint arXiv:2403.09472*.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

- Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, Sang Truong, Simran Arora, Mantas Mazeika, Dan Hendrycks, Zinan Lin, Yu Cheng, Sanmi Koyejo, Dawn Song, and Bo Li. 2023a. [Decodingtrust: A comprehensive assessment of trustworthiness in gpt models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 31232–31339. Curran Associates, Inc.
- Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023b. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*.
- Sheng Wang, Zihao Zhao, Xi Ouyang, Qian Wang, and Dinggang Shen. 2023c. Chatcad: Interactive computer-aided diagnosis on medical image using large language models. *arXiv preprint arXiv:2302.07257*.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.
- Liang Xu, Hang Xue, Lei Zhu, and Kangkang Zhao. 2024a. [Superclue-math6: Graded multi-step math reasoning benchmark for llms in chinese](#). *arXiv preprint arXiv:2401.11819*.
- Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. 2024b. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824*.
- Ziyao Xu and Houfeng Wang. 2024. [Spor: A comprehensive and practical evaluation method for compositional generalization in data-to-text generation](#). *arXiv preprint arXiv:2405.10650*.
- Zhe Yang, Damai Dai, Peiyi Wang, and Zhifang Sui. 2023. [Not all demonstration examples are equally beneficial: Reweighting demonstration examples for in-context learning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13209–13221, Singapore. Association for Computational Linguistics.
- Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*.
- Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Appendix

A Mathematical Notations

This section shows all of the mathematical notations used in this paper. If you forget the meaning of any notation, please refer to Table 4. We leverage $\hat{\cdot}$ to symbolize estimates (e.g. $\hat{P}(a_i)$ represents the estimate of the true value $P(a_i)$). For simplicity, we only show true values in Table 4, and estimates are omitted.

B Derivation of Consistency Score

§3.2 only shows the result for CS, and we show the derivation process of CS in this section. We have:

$$\begin{aligned}
 CS &= P(a|b) \\
 &= \frac{P(a, b)}{P(b)} \\
 &= \frac{\sum_{i=1, \dots, N} P(a_i, b_i)/N}{\sum_{i=1, \dots, N} P(b_i)/N} \quad (7) \\
 &= \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}
 \end{aligned}$$

It is worth noting that for a given question pair (a_i, b_i) , the probability that a model correctly answers a_i, b_i (i.e. $P(a_i)$ and $P(b_i)$) are unknown constants. When sampling answers, whether the model answers one question correctly does not affect answering the other, which allows us to deduce that the simultaneous probability of correctly answering both is $P(a_i, b_i) = P(a_i)P(b_i)$. However, this does not hold for random questions a and b , as $P(a, b) \neq P(a)P(b)$.

The above derivation process does not specify how the random questions a and b are obtained. We provide a more rigorous proof by defining the random process through which a and b are selected, as well as the random variables $P(a)$ and $P(b)$. Firstly, we outline the following stochastic process:

Randomly sampling a pair of questions (a, b) from $A \odot B$ with equal probability.

Based on this stochastic process, we define the random variables $P(a)$ and $P(b)$ as the probabilities of the model correctly answering a and b respectively, through a single temperature-based sampling. It is noteworthy that $P(a), P(b)$ are constant in the previous derivation, but here we treat them as random variables. Initially, the prior probability of selecting b_i in the above random process is

$P(\text{select } b_i) = \frac{1}{N}$. Upon introducing the condition that model answers b correctly, the posterior probability of b_i being selected in the random process becomes $P(\text{select } b_i) = \frac{P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}$. leveraging this posterior probability for the calculation of expected values, we have:

$$\begin{aligned}
 CS &= E[P(a|b)] \\
 &= \sum_{i=1, \dots, N} P(a_i|b_i)P(\text{select } b_i) \\
 &= \sum_{i=1, \dots, N} \frac{P(a_i, b_i)}{P(b_i)} \frac{P(b_i)}{\sum_{j=1, \dots, N} P(b_j)} \quad (8) \\
 &= \sum_{i=1, \dots, N} \frac{P(a_i)P(b_i)}{\sum_{j=1, \dots, N} P(b_j)} \\
 &= \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}
 \end{aligned}$$

C Formal Definition of Models with Similar Capabilities

For an evaluated model M_0 and a question pair (a_i, b_i) from dataset $A \odot B$, the probability of M_0 answer a_i, b_i correctly through a single temperature-based sampling is denoted as $P_{M_0}(a_i), P_{M_0}(b_i)$. We define a model set $\Omega = \{M_0, M_1, \dots\}$ in which models have similar capabilities (but consistency is not necessarily similar). For any $M_j \in \Omega$, we have:

1. $P_{M_0}(b_i) = P_{M_j}(b_i)$ for any $i \in \{1, \dots, N\}$
2. $Mset\{P_{M_0}(a_0), \dots, P_{M_0}(a_N)\}$
 $= Mset\{P_{M_j}(a_0), \dots, P_{M_j}(a_N)\},$

where $Mset$ denotes multiset (a.k.a. bag), a generalization of a set where repetition of elements matters.

In this scope, we define models with similar abilities as models whose correct probability on each datum in B are exactly the same and multisets of correct probability on each datum in A are identical to each other. The fact that different models from Ω demonstrate the same accuracy on A (and B) intuitively makes one feel that these models have similar capabilities. It is worth noting that only M_0 is an existing model in the real world; all other models in Ω are hypothetical for analysis of consistency score boundaries.

D Boundaries for Consistency Score

This section discusses the derivation of boundaries for consistency score utilized in §3.3, and we show both strict mathematical boundaries and tighter heuristic boundaries.

D.1 Mathematical Boundaries

Without any loss of generality, we assume that $P(b_0), \dots, P(b_N)$ is an ascending sequence (otherwise, the order of elements can be adjusted properly to meet this condition). After arranging the sequence $P(a_0), \dots, P(a_N)$ in ascending order, we denote the resulting sequence as $P(a_{(0)}), \dots, P(a_{(N)})$. According to the rearrangement inequality, we have:

$$\begin{aligned} & \frac{\sum_{i=1, \dots, N} P(a_{(N+1-i)})P(b_i)}{\sum_{i=1, \dots, N} P(b_i)} \\ & \leq \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i)}{\sum_{i=1, \dots, N} P(b_i)} \quad (9) \\ & \leq \frac{\sum_{i=1, \dots, N} P(a_{(i)})P(b_i)}{\sum_{i=1, \dots, N} P(b_i)} \end{aligned}$$

From this inequality, we obtain the mathematical upper bound $CS_{upp} = \frac{\sum_{i=1, \dots, N} P(a_{(i)})P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}$ and mathematical lower bound $CS_{low} = \frac{\sum_{i=1, \dots, N} P(a_{(N+1-i)})P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}$.

D.2 Heuristic Boundaries

Although the aforementioned boundaries are mathematically rigorous, they are too loose, as the lower bound sometimes approaches 0 and the upper bound approaches 1 in the experiments. Empirically, CS lies within a narrower interval. To find more precise boundaries, we leverage two heuristic assumptions:

Lower Bound Heuristic: *For the most inconsistent model, probabilities of correctly answering easy and hard questions $P(a)$ and $P(b)$ are independent (instead of negatively correlated).*

Upper Bound Heuristic: *For the most consistent model, the difference in probabilities of correctly answering easy and hard questions is directly proportional to the degree of increased difficulty level.*

These two hypotheses specify the behavior of the model of best and worst consistency. We assume that for a model of worst consistency, there

might be independence between correctly answering easy and hard questions, rather than a negative correlation where an increased probability of correctly answering hard questions leads to a lower probability of correctly answering easy questions. Conversely, for a model with best consistency, the probability of correctly answering easy and hard questions is entirely dependent on the difficulty level of the questions. Thus, the difference in probability between correctly answering easy and hard questions, $P(a_i) - P(b_i)$, is solely reliant on the gradient of difficulty from a_i to b_i . When constructing our dataset, it's almost impossible to ensure that each a_i scales up in difficulty uniformly to obtain b_i ; therefore, we hypothesize that the difficulty scaling from a_i to b_i follows a normal distribution (i.e. $(P(a) - P(b)) \sim N(\mu, \sigma)$).

Based on the Lower Bound Heuristic, we have a tighter heuristic lower bound:

$$\begin{aligned} CS_{low} &= P(a|b) = \frac{P(a, b)}{P(b)} \\ &= \frac{P(a)P(b)}{p(b)} = P(a) \quad (10) \\ &= \frac{\sum_{i=1, \dots, N} P(a_i)}{N} \end{aligned}$$

Based on the Upper Bound Heuristic, we have $P(a_i) - P(b_i) = \mu + \epsilon_i\sigma$, where ϵ_i is a random variable that follows a standard normal distribution. The maximum likelihood estimation of μ, σ is:

$$\begin{aligned} \hat{\mu} &= \frac{\sum_{i=1, \dots, N} (P(a_i) - P(b_i))}{N}, \quad (11) \\ \hat{\sigma} &= \sqrt{\frac{\sum_{i=1, \dots, N} (P(a_i) - P(b_i) - \hat{\mu})^2}{N}} \end{aligned}$$

Substitute actual values μ, σ with estimated ones $\hat{\mu}, \hat{\sigma}$, then we have the theoretical value of $P(a_i)$ in a consistent model: $P(a_i) = P(b_i) + \hat{\mu} + \epsilon_i\hat{\sigma}$. Empirically, the value of σ does not affect final results if averaging on multiple sampling of ϵ , so we directly let $\sigma = 0$. Then by substituting the theoretical values of $P(a_i)$ in consistent model for the true values of $P(a_i)$ used in calculation of CS, we can obtain the heuristic upper bound as follows:

$$CS_{upp} = \frac{\sum_{i=1, \dots, N} (P(b_i) + \hat{\mu})P(b_i)}{\sum_{i=1, \dots, N} P(b_i)} \quad (12)$$

E Probability Estimation

This section shows the derivation of the maximum likelihood estimate of $P(a_i)$ in Multiple Sampling

Estimation (§E.1) and Early Stopping Estimation (§E.2), respectively. Besides, we also show the pseudo-code and more discussion about Early Stopping Estimation in §E.3

E.1 Multiple Sampling Estimation

For problem a_i , we sample answers m times independently to obtain a sequence $a_i^1, a_i^2, \dots, a_i^m$. Let $a_i^j = 1$ if the model generates a correct answer on the j th sampling; otherwise, $a_i^j = 0$. In this case, a_i^j follows a Bernoulli distribution. Let $k = \sum_{j=1, \dots, m} a_i^j$, we have the likelihood function:

$$\begin{aligned} L(P(a_i); k) &= \prod_{j=1}^m P(a_i)^{a_i^j} (1 - P(a_i))^{1-a_i^j} \\ &= P(a_i)^k (1 - P(a_i))^{m-k}, \end{aligned} \quad (13)$$

the derivative of the likelihood function:

$$\begin{aligned} &\frac{\partial L(P(a_i); k)}{\partial P(a_i)} \\ &= kP(a_i)^{k-1} (1 - P(a_i))^{m-k} \\ &\quad - (m - k)P(a_i)^k (1 - P(a_i))^{m-k-1} \\ &\propto k(1 - P(a_i)) - (m - k)P(a_i) \\ &\propto k - mP(a_i) \end{aligned} \quad (14)$$

$L(P(a_i); k)$ is monotonically increasing when $P(a_i) \in [0, \frac{k}{m}]$ and monotonically decreasing when $P(a_i) \in [\frac{k}{m}, 1]$. When $P(a_i) = \frac{k}{m}$, it maximizes the likelihood function, so the maximum likelihood estimate of $P(a_i)$ is:

$$\hat{P}(a_i) = \frac{k}{m} = \frac{\sum_{j=1, \dots, m} a_i^j}{m} \quad (15)$$

E.2 Early Stopping Estimation

In Early Stopping Estimation, the minimum and the maximum number of sampling times k_{min} and k_{max} are set as hyper-parameters for a given question a_i . Initially, k_{min} answers are sampled, and the sampling process will be terminated if at least one correct answer exists in these k_{min} answers; otherwise, answers will be sampled one by one until a correct answer appears for the first time. Besides, the sampling procedure will be forcibly terminated if a correct answer still does not emerge after sampling k_{max} answers. Let $P(k, k_c)$ be the probability of sampling k answers in total in which k_c answers are correct, and let $L(P(a_i); k, k_c)$ be the likelihood function. The discussion is divided into the following three cases based on the different values of k :

Case 1: $k = k_{min}$

We have the likelihood function:

$$\begin{aligned} L(P(a_i); k, k_c) &= P(k, k_c) \\ &= \binom{k_{min}}{k_c} P(a_i)^{k_c} (1 - P(a_i))^{k_{min} - k_c}, \end{aligned} \quad (16)$$

the derivative of the likelihood function:

$$\begin{aligned} &\frac{\partial L(P(a_i); k, k_c)}{\partial P(a_i)} \\ &= \binom{k_{min}}{k_c} [k_c P(a_i)^{k_c-1} (1 - P(a_i))^{k_{min} - k_c} \\ &\quad - (k_{min} - k_c) P(a_i)^{k_c} (1 - P(a_i))^{k_{min} - k_c - 1}] \\ &\propto k_c (1 - P(a_i)) - (k_{min} - k_c) P(a_i) \\ &\propto k_c - k_{min} P(a_i) \end{aligned} \quad (17)$$

$L(P(a_i); k, k_c)$ is monotonically increasing when $P(a_i) \in [0, \frac{k_c}{k_{min}}]$ and monotonically decreasing when $P(a_i) \in [\frac{k_c}{k_{min}}, 1]$. When $P(a_i) = \frac{k_c}{k_{min}}$, it maximizes the likelihood function, so the maximum likelihood estimate of $P(a_i)$ is: $\hat{P}(a_i) = \frac{k_c}{k_{min}}$

Case 2: $k_{min} < k < k_{max}$

We have the likelihood function:

$$\begin{aligned} L(P(a_i); k, k_c) &= P(k, k_c) \\ &= (1 - P(a_i))^{k-1} P(a_i), \end{aligned} \quad (18)$$

the derivative of the likelihood function:

$$\begin{aligned} &\frac{\partial L(P(a_i); k, k_c)}{\partial P(a_i)} \\ &= -(k - 1)(1 - P(a_i))^{k-2} P(a_i) \\ &\quad + (1 - P(a_i))^{k-1} \\ &\propto -(k - 1)P(a_i) + 1 - P(a_i) \\ &\propto 1 - kP(a_i) \end{aligned} \quad (19)$$

$L(P(a_i); k, k_c)$ is monotonically increasing when $P(a_i) \in [0, \frac{1}{k}]$ and monotonically decreasing when $P(a_i) \in [\frac{1}{k}, 1]$. When $P(a_i) = \frac{1}{k}$, it maximizes the likelihood function, so the maximum likelihood estimate of $P(a_i)$ is: $\hat{P}(a_i) = \frac{1}{k}$

Case 3: $k = k_{max}$ We have the likelihood function:

$$\begin{aligned} L(P(a_i); k, k_c) &= P(k, k_c) \\ &= (1 - P(a_i))^{k_{max} - \mathbb{I}(k_c \neq 0)} P(a_i)^{\mathbb{I}(k_c \neq 0)}, \end{aligned} \quad (20)$$

where \mathbb{I} denoted indicator function. If $k_c \neq 0$, the likelihood function is the same as Case 2, we have

$\hat{P}(a_i) = \frac{1}{k_{max}}$ by the same reasoning. If $k_c = 0$, the likelihood function is monotonically decreasing on $[0, 1]$, so the maximum likelihood estimate of $P(a_i)$ is: $\hat{P}(a_i) = 0$.

To summarize, the maximum likelihood estimate of $P(a_i)$ is shown as below:

1. if $k = k_{min}$, then $\hat{P}(a_i) = \frac{k_c}{k_{min}}$
2. if $k_{min} < k < k_{max}$, then $\hat{P}(a_i) = \frac{1}{k}$
3. if $k = k_{max}$, then $\hat{P}(a_i) = \frac{\mathbb{I}(k_c \neq 0)}{k_{max}}$

The above three cases can be formulated as:

$$\hat{P}(a_i) = \frac{k_c}{k} \quad (21)$$

E.3 More Details about Early Stopping Estimation

The pseudo-code for Early Stopping Estimation is shown in Algorithm 1. If we set k_{max} equal to the number of sampling m in Multiple Sampling Estimation, in the worst-case scenario, the number of sampling of Early Stopping Estimation could equal that of Multiple Sampling Estimation, theoretically. However, empirical results suggest that, due to the high accuracy of these closed-source models, the actual number of samples required with early stopping is typically low. While introducing an early stopping strategy might slightly reduce the accuracy of estimation, the reduction in the number of API calls required makes it a worthwhile trade-off.

Algorithm 1: Early Stopping Estimation

input : a question a_i ; function to generate an answer by sampling $generate()$;
minimum number of samples k_{min} ;
maximum number of samples k_{max}

output : estimated probability $\hat{P}(a_i)$ of model answer a_i correctly through a single sampling

```

1 Initialize  $answer\_list \leftarrow []$ 
2 for  $j \leftarrow 1$  to  $k_{min}$  do
3    $answer \leftarrow generate(a_i)$ 
4    $answer\_list.append(answer)$ 
5 if not exist_correct( $answer\_list, a_i$ ) then
6   for  $j \leftarrow k_{min} + 1$  to  $k_{max}$  do
7      $answer \leftarrow generate(a)$ 
8      $answer\_list.append(answer)$ 
9     if  $answer$  is correct then
10      Break
11  $correct\_num \leftarrow CountCorrect(answer\_list)$ 
12  $\hat{P}(a_i) \leftarrow correct\_num / Len(answer\_list)$ 
13 Return  $\hat{P}(a_i)$ 

```

Multiple Sampling Estimation v.s. Early Stopping Estimation

If we sample fewer times in Multiple Sampling Estimation, resulting in a roughly equal total number of samples across the entire dataset for both methods, which method yields a more accurate estimation? For questions with a low probability of being answered correctly (near 0%), a large number of samples are required to obtain a correct answer and thus accurately estimate this probability; otherwise, there is a high risk of erroneously deeming the probability to be zero. On the contrary, for questions that models have a high probability of answering correctly (near 100%), almost all samples will be correct, and therefore, fewer samples are needed to accurately estimate the probability. The Early Stopping Estimation method adapts the number of sampling times dynamically for different questions, making better use of each sampling opportunity compared to the Multiple Sampling Estimation. Consequently, it achieves higher precision in its final estimates when the sampling times are limited.

F Impact of Data Leakage

Data leakage can affect our evaluation. We find leakage of easy and hard data can lead to higher and lower CS, respectively. We analyze data leaking on datum a_i (or b_i) by modeling the leaking as an increment in probability $P(a_i)$ (or $P(b_i)$). For example, if a_i is not leaked, model answers it correctly with probability $P(a_i)$; after a_i is leaked, model answers it correctly with higher probability $P(a_i) + \Delta P(a_i)$. The original CS is $\frac{\sum_{i=1, \dots, N} P(a_i)P(b_i)}{\sum_{i=1, \dots, N} P(b_i)}$, and we numerically analyze the change of CS after data leakage.

F.1 Leakage of Easy Data

After leakage on an easy datum a_j , the new CS after leakage is :

$$\begin{aligned}
CS_{leak} &= \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i) + \Delta P(a_j)P(b_j)}{\sum_{i=1, \dots, N} P(b_i)} \\
&= CS + \frac{\Delta P(a_j)P(b_j)}{\sum_{i=1, \dots, N} P(b_i)} \\
&> CS
\end{aligned} \quad (22)$$

So leakage of easy data will lead to a higher CS.

F.2 Leakage of Hard Data

After leakage on a hard datum b_j , the new CS after leakage is :

$$CS_{leak} = \frac{\sum_{i=1, \dots, N} P(a_i)P(b_i) + P(a_j)\Delta P(b_j)}{\sum_{i=1, \dots, N} P(b_i) + \Delta P(b_j)} \quad (23)$$

If $\frac{P(a_j)\Delta P(b_j)}{\Delta P(b_j)} = P(a_j) > CS$, $CS_{leak} > CS$; If $\frac{P(a_j)\Delta P(b_j)}{\Delta P(b_j)} = P(a_j) < CS$, $CS_{leak} < CS$. The expected value of $P(a_j)$ is the accuracy on easy data, so we have $E(P(a_j)) < CS$, and $CS_{leak} < CS$ on average. So leakage of hard data will lead to a lower CS on average.

G More Details and Results for Experiments

We show more implementation details and results for main experiments in §4.

G.1 Implement Experiment Details

For small open-source models with roughly 7B or 13B parameters, we employ the Multiple Sampling Estimation and independently sample 20 answers for each question. As for the large models with around 70B parameters and closed-source models, we utilize the Early Stopping Estimation to reduce computational costs and API calls, and we set the minimum number of samples at $k_{min} = 3$ and the maximum at $k_{max} = 20$. For each small open-source model (7B or 13B), we run the experiments on a single Nvidia A100 80G GPU; for each large model (70B), experiments are conducted on three Nvidia A100 80G GPUs. All of the open-source models are acquired from Huggingface⁵, and we utilize the default sampling hyper-parameters (e.g. temperature, top-p) released by model developers. All evaluations are under zero-shot setting: for mathematics and instruction-following data, questions as fed into LLMs directly; code data are transformed into instruction format⁶ before inputted into models.

G.2 Correlation between Capability and Consistency

We find there is a strong correlation between capability and consistency of LLM in all of our evaluated domains. Taking code domain as an example,

⁵<https://huggingface.co/>

⁶<https://huggingface.co/datasets/codeparrot/instructhumaneval>

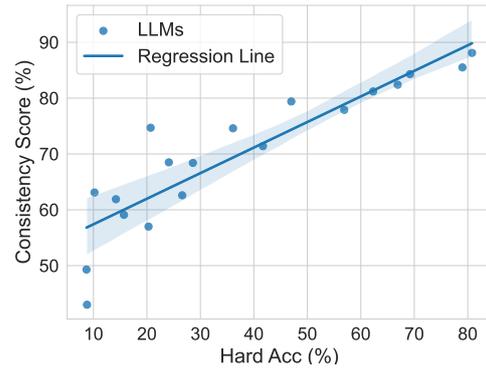


Figure 8: Correlation of capability and consistency.

Kendall’s coefficient of correlation between accuracy on hard set and CS of all evaluated LLMs on code domain is 0.801, and the linear regression line is shown in Figure 8 (each dot represents an LLM).

G.3 Full Experiment Results on Relative Consistency Score

Due to space limitation, §4 only shows experiment results on RCS in code domain. We show full experiment results in Table 3.

H Metric Convergence

The calculation of our evaluation metric consistency score (CS) and relative consistency score (RCS) relies on repeated sampling for a given question. We show the value change and variance of these metrics as the increase in sampling times. As the convergence results for Llama3-8B-Instruct on mathematics domain shown in Figure 9, CS converges faster than RCS and achieves a stable value at about 5 samples. The value of RCS converges relatively slower and becomes stable after about 15 samples.

We also explore leveraging consistent rate as an evaluation metric. Taking the case where the probability of answering an easy question correctly is larger than that of the hard question as a consistent case, we have consistent rate = $\frac{\text{number of consistent cases}}{\text{number of all cases}} * 100\%$. However, we find that for the case where the probability of answering easy and hard questions correctly is close, reaching a convergent result requires too many times of sampling. We abandon this metric due to its high computational cost.

Moldes	Code				Instruction following				Maths			
	low	CS	upp	RCS	low	CS	upp	RCS	low	CS	upper	RCS
GPT-4 Turbo	85.5	88.1	93.0	34.8	84.2	91.8	93.1	85.3	96.2	96.8	97.2	54.4
GPT-3.5 Turbo	71.4	81.2	88.8	56.1	76.1	88.6	91.7	80.5	86.9	90.7	96.2	40.8
Claude-3 Opus	81.1	85.5	93.6	35.1	87.7	93.4	95.7	70.7	96.5	96.5	98.1	0.6
Qwen Max	75.0	82.4	93.4	40.5	74.3	89.6	94.3	76.7	95.2	96.8	98.2	51.9
Llama3-70B-Instruct	73.9	84.3	94.6	50.2	86.7	94.4	95.1	90.7	94.9	96.9	98.0	64.1
Llama2-70B-Chat	34.5	74.7	83.8	81.5	56.6	81.0	91.6	69.7	70.5	83.7	90.3	66.9
Qwen1.5-72B-Chat	62.3	79.4	91.3	58.7	56.5	87.3	90.7	89.9	90.6	93.6	94.0	87.2
Deepseek-67B-Chat	68.6	77.9	88.1	47.6	52.5	83.8	88.1	87.8	90.2	94.8	98.8	54.0
Llama2-13B-Chat	20.2	61.9	84.2	65.1	48.3	84.2	89.2	87.7	48.6	67.2	76.1	67.4
Qwen1.5-14B-Chat	51.4	74.6	86.0	67.2	55.4	83.6	90.8	79.6	82.6	90.7	92.2	84.7
Baichuan2-13B-Chat	21.5	59.1	73.4	72.5	31.0	63.3	75.2	73.2	48.6	65.8	78.1	58.3
Llama3-8B-Instruct	53.6	71.4	83.4	59.7	78.5	87.9	91.8	70.7	77.8	87.4	89.2	84.6
Llama2-7B-Chat	14.9	63.1	79.6	74.5	43.7	75.6	86.2	75.0	34.3	57.9	76.5	55.9
Qwen1.5-7B-Chat	40.9	68.4	81.9	66.9	47.2	82.5	87.9	86.7	68.6	83.6	88.8	74.3
ChatGLM3_6B	50.8	68.5	81.6	57.4	36.6	64.7	75.3	72.5	64.4	83.8	86.2	89.0
Deepseek-7B-Chat	40.3	62.6	75.9	62.6	47.5	71.0	82.3	67.7	69.0	84.8	88.6	80.8
Mistral-7B-Instruct	28.4	57.0	69.7	69.2	60.8	84.3	88.3	85.3	51.8	67.4	75.3	66.5
Yi-6B-Chat	13.2	49.3	70.5	63.0	37.4	76.0	80.2	90.1	50.9	69.7	76.9	72.4
Baichuan2-7B-Chat	12.4	43.0	54.5	72.7	29.9	60.0	69.8	75.5	28.4	50.1	56.6	76.9

Table 3: Relative consistency results. A variety of LLMs are evaluated on code, instruction-following, and maths domains. On each domain, we report consistency score (CS), lower and upper bounds of CS (denoted as **low** and **upp**).

I Case Study

We show inconsistent cases of GPT-4 in Table 6,7,8,9,10. More analyses are shown in §5.3.

J Prompts for Data Generation

The prompts for data generation on code, maths and instruction-following domains are shown in Figure 10, 11, 12 respectively.

K Example Data

We show example data in Table 5.

Notations	Meanings
A, B	easy question set and hard question set
$A \odot B$	dataset with pairwise easy and hard questions
N	number of data in $A \odot B$ (also for A or B)
a_i, b_i	the i -th easy question and the i -th hard question (they are a pair)
$P(a_i)$ (or $P(b_i)$)	the probability of model answer a_i (or b_i) correctly through a single temperature-based sampling
(a, b)	a pair of questions sampled from $A \odot B$ with equal probability.
$P(a)$ (or $P(b)$)	the probability of model answer a (or b) correctly through a single temperature-based sampling
$P(a b)$	the probability of model correctly answering a given that it has answered b correctly through a single temperature-based sampling
$P(a, b)$	the probability of model correctly answering a and b correctly through a single temperature-based sampling
$E[*]$	expected values
M_0	a language model to be tested
Ω	a set of models with similar abilities with M_0
$P_{M_0}(a_i)$ (or $P_{M_0}(b_i)$)	the probability of model M_0 answer a_i (or b_i) correctly through a single temperature-based sampling
$Mset\{\}$	multiset (a.k.a. bag), a generalization of a set where repetition of elements matter
$upp\ CS(\Omega), low\ CS(\Omega)$	the upper and lower bounds of consistency scores for all models in Ω
ϵ_i	random variable following the standard normal distribution
μ, σ	mean and standard variance for $(P(a_i) - P(b_i))$
m	sampling times in Multiple Sampling Estimation
k_{min}, k_{max}, k, k_c	minimum sampling times, maximum sampling times, actual sampling times and correct answer number in early Stopping Estimation
L	likelihood function

Table 4: mathematical notations and their meanings.

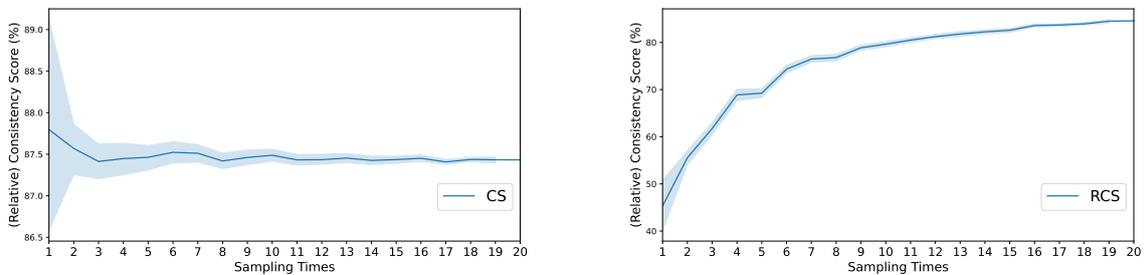


Figure 9: Convergence of metrics with the increase of sampling times.

Mathematics	easy	<p>Question: George has 45% more pears than bananas. If George has 200 bananas, how many fruits does George have?</p> <p>Answer: To find out the number of pears George has, we must first calculate 45% of the bananas, then add that to the number of bananas he has. If George has 200 bananas, we will find 45% of this number like so: $0.45 * 200 = 90$. This means George has 90 more pears than bananas. Now, add the extra pears to the 200 bananas to find the total number of pears: $200 + 90 = 290$ pears. Now, to find the total number of fruits, add the number of bananas to the number of pears: Number of bananas (200) + Number of pears (290) = Total fruits. $200 + 290 = 490$ fruits. Therefore, George has 490 fruits in total.</p> <p>Answer (Number Only): 490</p>
	hard	<p>Question: George has 45% more pears than bananas. If George has 200 bananas, 270 oranges, and 150 apples, how much are George's fruits worth if pears cost 3 dollars each, bananas 2 dollars each, oranges 50 cents each, and apples 1 dollar each?</p> <p>Answer: First, let's calculate how many pears George has. Since he has 45% more pears than bananas, and he has 200 bananas, we calculate the number of pears as follows: Number of pears = Number of bananas + $0.45 * \text{Number of bananas} = 200 + 0.45 * 200 = 200 + 90 = 290$. Next, let's calculate the total cost of each type of fruit. Bananas (200 bananas at \$2 each): $200 * 2 = \\$400$. Pears (290 pears at \$3 each): $290 * 3 = \\$870$. Oranges (270 oranges at \$0.50 each): $270 * 0.50 = \\$135$. Apples (150 apples at \$1 each): $150 * 1 = \\$150$. Finally, to find the total value of all the fruits, we sum these amounts: $\\$400 + \\$870 + \\$135 + \\$150 = \\$1555$. Therefore, George's fruits are worth \$1555 in total.</p> <p>Answer (Number Only): 1555</p>
Instruction	easy	<p>Question: Write a riddle for kids about auspices but make sure you don't use any commas.</p> <p>Constraint Type List: ["punctuation:no_comma"]</p> <p>Constraint Kwargs: [{}]</p>
	hard	<p>Question: Write a riddle for kids about auspices but make sure you don't use any commas. Also, the riddle must be at least 6 sentences long.</p> <p>Constraint Type List: ["punctuation:no_comma", "length_constraints:number_sentences"]</p> <p>Constraint Kwargs: [{}, {"relation": "at least", "num_sentences": 6}]</p>
Code	easy	<p>Question:</p> <pre>def strlen(string: str) -> int: """ Return length of given string >>> strlen("") 0 >>> strlen('abc') 3 """</pre> <p>Answer:</p> <pre>return len(string)</pre> <p>Check Function:</p> <pre>def check(candidate): assert candidate("") == 0 assert candidate('x') == 1 assert candidate('asdasnakj') == 9</pre>
	hard	<p>Question:</p> <pre>def strlen_add(string1: str, string2: str) -> str: """ Return length sum of two given strings >>> strlen_add('abc', 'd') 4 """</pre> <p>Answer:</p> <pre>return len(string1 + string2)</pre> <p>Check Function:</p> <pre>def check(candidate): assert candidate('abc', 'd') == 4 assert candidate('', 'z') == 1 assert candidate('x', 'y') == 2 assert candidate('hello', '!') == 6</pre>

Table 5: Example data from ConsisEval. For each domain (mathematics, instruction-following, and code), a pairwise example datum is shown.

	Easy	Hard
Questions	<p>A team of 4 painters worked on a mansion for 3/8ths of a day every day for 3 weeks. How many hours of work did each painter put in? Standard Answer: 189</p>	<p>A team of 4 painters worked on a mansion for 3/8ths of a day every day for 3 weeks. Each painter earns 80 yuan per hour. After working for 3 weeks, 70000 yuan were allocated for the wages. How much money will remain after paying the wages of the 4 painters? Standard Answer: 9,520</p>
Model Answers	<p>First, let's figure out how many days are in 3 weeks. There are 7 days in a week, so 3 weeks have $3 \times 7 = 21$ days. Each day, the team worked for 3/8 of a day. There are 24 hours in a full day, so 3/8 of a day in hours is:</p> $24 \times \frac{3}{8} = 3 \times 3 = 9 \text{ hours}$ <p>If they worked 9 hours each day, over 21 days, the total hours worked by the team is:</p> $9 \text{ hours/day} \times 21 \text{ days} = 189 \text{ hours}$ <p>Since the total hours worked by the team is 189 and the team consists of 4 painters, the hours worked by each painter is:</p> $189 \text{ hours} \div 4 = 47.25 \text{ hours}$ <p>So, each painter put in 47.25 hours of work.</p>	<p>First, let's calculate the total hours worked by each painter: 1. A day is considered to have 24 hours, so 3/8ths of a day is: $\frac{3}{8} \times 24$ hours. 2. $\frac{3}{8}$ of 24 hours = 9 hours per day. Each painter works for 9 hours a day. Now, let's find out the total days worked in 3 weeks. Since there are 7 days in a week: 3 weeks \times 7 days/week = 21 days. Each painter worked for 9 hours a day for 21 days, so their total hours worked are: 9 hours/day \times 21 days = 189 hours in total for 3 weeks. Since there are 4 painters: 189 hours \times 4 = 756 hours worked in total by all painters. Given that each painter earns 80 yuan per hour, the total wages for all painters are: 756 hours \times 80 yuan/hour = 60,480 yuan. Finally, subtracting the total wages from the initially allocated funds gives the remaining amount: 70,000 yuan - 60,480 yuan = 9,520 yuan. Therefore, 9,520 yuan will remain after paying the wages of the 4 painters.</p>

Table 6: An inconsistency case for GPT-4. Red texts denote wrong reasoning steps. GPT-4 is distracted by the redundant condition of "4 painters", leading it to perform an additional unnecessary calculation after arriving at the correct answer, ultimately resulting in the wrong final answer.

	Easy	Hard
Questions	Hallie had dance practice for 1 hour on Tuesdays and 2 hours on Thursdays. On Saturdays, she had dance practice that lasted twice as long as Tuesday's night class. How many hours a week did she have dance practice? Standard Answer: 5	Hallie had dance practice for 1 hour on Tuesdays and 2 hours on Thursdays. On Saturdays, she had dance practice that lasted twice as long as Tuesday's night class. On Sundays, her dance practice was 3 hours longer than Saturday's. Mary is also a dance enthusiast, and her weekly dance practice is 4 hours less than Hallie's. Alice's weekly dance practice is 5 times the duration of Hallie's Thursday practice. How many more hours does Alice practice per week than Mary? Standard Answer: 4
Model Answers	Hallie has dance practice for 1 hour on Tuesdays. On Thursdays, she has dance practice for 2 hours. Since Saturday's dance practice is twice as long as Tuesday's, it would be 2 hours * 2 = 4 hours. Adding all the hours together, we have: 1 hour (Tuesday) + 2 hours (Thursday) + 4 hours (Saturday) = 7 hours So, Hallie had dance practice for 7 hours a week.	First, let's calculate the total weekly practice duration for Hallie. Tuesday practice = 1 hour Thursday practice = 2 hours Saturday practice = 2 * Tuesday = 2 * 1 hour = 2 hours Sunday practice = Saturday practice + 3 hours = 2 hours + 3 hours = 5 hours Total weekly practice for Hallie = Tuesday + Thursday + Saturday + Sunday = 1 hour + 2 hours + 2 hours + 5 hours = 10 hours Mary's weekly dance practice is 4 hours less than Hallie's: Mary's weekly practice = Hallie's weekly practice - 4 hours = 10 hours - 4 hours = 6 hours Alice's weekly dance practice is 5 times the duration of Hallie's Thursday practice: Alice's weekly practice = 5 * Thursday (of Hallie) = 5 * 2 hours = 10 hours Comparing how many more hours Alice practices per week than Mary: Alice's weekly practice - Mary's weekly practice = 10 hours - 6 hours = 4 hours Therefore, Alice practices 4 more hours per week than Mary.

Table 7: An inconsistency case for GPT-4. Red texts denote wrong reasoning steps. Misapplication of data occurs with GPT-4: Tuesday involves 1 hour of dancing, while GPT-4 mistakenly uses Thursday's 2 hours in its calculation, resulting in a wrong final answer.

	Easy	Hard
Questions	Jen decides to travel to 3 different countries. He has to pay \$400 for the supplies he needs, in total. The tickets for travel cost, in total, 50% more than the supplies. How much does travel cost? Standard Answer: 1,000	Jen decides to travel to 3 different countries. He has to pay \$400 for the supplies he needs, in total. The tickets for travel cost, in total, 50% more than the supplies. Additionally, the accommodation costs for the travel are 3 times the cost of the supplies. The total travel expenses amount to one-quarter of Jen's monthly salary. If Eily's monthly salary is \$3200 higher than that of Jen, what is Eily's monthly salary? Standard Answer: 12,000
Model Answers	If the supplies cost \$400 in total and the tickets for travel cost 50% more than the supplies, we calculate the total cost of the tickets by adding 50% of \$400 to the original \$400. 50% of \$400 is $0.50 * \$400 = \200 . So, the tickets cost \$400 (the cost of supplies) + \$200 (50% more than the supplies) = \$600 in total.	Let's break this down step by step. 1. Cost of Supplies : Given as \$400. 2. Cost of Tickets : Stated as 50% more than the supplies, which is: $\text{Cost of Supplies} \times 150\% = 400 \times 1.5 = \600 . 3. Cost of Accommodation : Three times the cost of the supplies, which is: $3 \times \text{Cost of Supplies} = 3 \times 400 = \1200 . 4. Total Travel Expenses : Sum of Supplies, Tickets, and Accommodation costs: $400 + 600 + 1200 = \$2200$. 5. Total Travel Expenses as a Quarter of Jen's Monthly Salary : This means: $\text{Jen's Monthly Salary} = 2200 \times 4 = \8800 . 6. Eily's Monthly Salary is \$3200 more than Jen's Monthly Salary. Therefore, it's: $\text{Eily's Monthly Salary} = 8800 + 3200 = \12000 .

Table 8: An inconsistency case for GPT-4. Misunderstanding of the question leads to an error: the question asks for the total cost of the trip, whereas GPT-4 only calculates the cost of travel tickets.

	Easy	Hard
Questions	<p>Mark plants a beanstalk below his second-story window, which is 20 feet off the ground. The beanstalk doubles its height every day. If the beanstalk starts out 4 feet tall, how many days will it take to be taller than Mark's window?</p> <p>Standard Answer: 3</p>	<p>Mark plants a beanstalk below his second-story window, which is 20 feet off the ground. The beanstalk doubles its height every day. If the beanstalk starts out 4 feet tall, on the third day after planting it, Mark cuts off the part of the beanstalk that has grown above his window to sell. The beanstalk sells for \$3 per foot. The money Mark makes from selling the beanstalk is one fourteenth of his daily wage. How much does Mark earn per day?</p> <p>Standard Answer: 504</p>
Model Answers	<p>To determine how many days it will take for the beanstalk to be taller than Mark's second-story window, 20 feet off the ground, we can follow the growth pattern of the beanstalk, which doubles in height each day, starting at 4 feet.</p> <p>Day 1: 4 feet, Day 2: 4 feet \times 2 = 8 feet, Day 3: 8 feet \times 2 = 16 feet, Day 4: 16 feet \times 2 = 32 feet</p> <p>On day 4, the beanstalk's height becomes 32 feet, which exceeds the 20-foot height of the window. Therefore, it will take 4 days for the beanstalk to grow taller than Mark's second-story window.</p>	<p>To solve the problem, let's first find out how tall the beanstalk is on the third day after it starts growing.</p> <p>Starting height: 4 feet</p> <p>Growth pattern: doubles height every day</p> <p>1st day height: 4 feet \times 2 = 8 feet, 2nd day height: 8 feet \times 2 = 16 feet, 3rd day height: 16 feet \times 2 = 32 feet</p> <p>By the third day, the beanstalk is 32 feet tall. Since Mark's window is 20 feet off the ground, the part of the beanstalk that grows above the window is the difference between its third-day height and the window's height.</p> <p>Height above the window: 32 feet - 20 feet = 12 feet</p> <p>Mark cuts and sells this 12-foot section for \$3 per foot.</p> <p>Money made from selling the beanstalk: 12 feet \times \$3/foot = \$36</p> <p>It is stated that the money Mark makes from selling the beanstalk is one fourteenth of his daily wage. Let's represent his daily wage as W.</p> <p>Therefore, according to the problem:</p> $\$36 = 1/14 * W$ <p>To find W, multiply both sides of the equation by 14:</p> $W = \$36 * 14 = \504 <p>Mark earns \$504 per day.</p>

Table 9: An inconsistency case for GPT-4. Red texts denote wrong reasoning steps. GPT-4 encounters an off-by-one error: the initial state should be recorded as Day 0, but GPT-4 starts recording from Day 1 instead.

	Easy	Hard
Questions	<p>Kenny is selling his Pokemon cards to buy a ticket to an amusement park, which costs \$100. He has a collection of cards and plans to sell them for \$1.5 each. He keeps 1/3 of them and gets to go to the amusement park with \$50 in spending cash. How many cards did he start with? Standard Answer: 150</p>	<p>Kenny is selling his Pokemon cards to buy a ticket to an amusement park, which costs \$100. He has a collection of cards and plans to sell them for \$1.5 each. He keeps 1/3 of them and, after buying the ticket, has \$50 in spending money for the amusement park. His initial number of cards is 5 times the number of cards Eily has. John has 37 more cards than Eily. Mary has 4 times as many cards as John. How many cards does Mary have? Standard Answer: 268</p>
Model Answers	<p>Let's denote the original number of Pokemon cards that Kenny has as X. Kenny decides to sell his cards for \$1.5 each. However, he keeps one-third of them, which means he sells two-thirds of his cards. The number of cards he sells is $\frac{2}{3} * X$. From the sale of his cards, he earns enough to buy the amusement park ticket, which costs \$100, and he has \$50 in spending cash. Therefore, the total amount he earns from selling his cards is \$100 for the ticket plus \$50 spending cash, which equals \$150. The amount he earns from selling the cards can be found by multiplying the number of cards he sells ($\frac{2}{3} * X$) by the price per card (\$1.5): $(\frac{2}{3} * X) * \\$1.5 = \\150 Let's solve for X: $\\$1.5 * (\frac{2X}{3}) = \\150 $(\frac{3}{2}) * \\$1.5 * (\frac{2X}{3}) = \\$150 * (\frac{3}{2})$ $\\$2.25 * X = \\225 Divide both sides by \$2.25 to find X: $X = \\$225 / \\2.25 X = 100 Kenny started with 100 Pokemon cards.</p>	<p>Let's break down the problem step by step. 1. We know that Kenny sells his cards for \$1.5 each and wants to end up with \$150 in total (\$100 for the ticket and \$50 in spending money). 2. Let's find out how many cards he sells. We take the total money he wants to have, which is \$150, and divide it by the price per card: Total cards sold = total money needed / price per card Total cards sold = \$150 / \$1.5 Total cards sold = 100 cards 3. Kenny keeps 1/3 of his cards and sells the rest, so 2/3 of his cards were sold. Let's calculate the initial number of cards he had: Let x be the number of cards Kenny initially had. $(\frac{2}{3})x = 100$ cards To find x, we solve for x: $x = (100 \text{ cards}) / (\frac{2}{3})$ $x = (100 * 3) / 2$ $x = 300 / 2$ $x = 150$ cards So Kenny originally had 150 cards. 4. Kenny's initial number of cards is 5 times the number of cards Eily has. Let's call the number of cards Eily has E. Therefore: $5E = 150$ Solving for E gives us: $E = 150 / 5$ $E = 30$ cards Eily has 30 cards. 5. John has 37 more cards than Eily: John's cards = E + 37 John's cards = 30 + 37 John's cards = 67 cards 6. Mary has 4 times as many cards as John: Mary's cards = 4 * John's cards Mary's cards = 4 * 67 Mary's cards = 268 cards So Mary has 268 cards.</p>

Table 10: An inconsistency case for GPT-4. Red texts denote wrong reasoning steps. GPT-4 encounters a computational error while solving equations.

Prompt for Code Data Generation

#Instruction#:

I want you to act as a helpful assistant. Please help me modify some programming problems and make them harder. A programming problem datum consists of three parts: #Problem#, #Answer#, and #Check Function#. The #Problem# includes the name of a python function, function signature, and docstring; the #Answer# is the specific code that fulfills the function's purpose; in addition to that, there is a #Check Function# to verify whether the answer is correct. Please follow the format of the following demonstrations, modify the original problem, and make it more challenging. To ensure that there is a strict order in difficulty between the original problem and modified one, steps to solve the original problem should be included in that of the modified problem. In other words, steps to solve the original problem is a proper subset of that of the modified problem. Except the modified #Problem#, you should also provide #Answer# and #Check Function# to the modified #Problem#.

#Demonstrations#:

<insert demonstrations>

The above are some demonstrations showing how to modify the original problems. Please follow their format and modify the following problem:

#Problem#:

<insert the original problem>

#Answer#:

<insert the answer>

#Check Function#:

<insert the check function>

Please modified the above #Problem# and then provide #Answer# and #Check Function# to the modified #Problem#:

Figure 10: Our prompt fed to GPT-4 for code data generation. Our prompt is comprised of intention instruction, demonstrations, and one datum to be modified. The instruction offers a clear description of the composition of the datum and outlines the task we expect the model to accomplish. Demonstrations are provided as a format reference for the model, followed by the original datum for the model to modify.

Prompt for Math Data Generation

#Instruction#:

I want you to act as a helpful assistant. Please help me modify some grade school math problems and make them harder. A math problem datum consists of two parts: #Problem# and #Answer#. The #Problem# provides a background description of a real-world mathematical problem, along with the conditions known and the unknown content to be solved. There is a strict gurrantee that the unknown value can be derived through a few proper computational steps based on konwn conditions. The #Answer# encompasses several computational steps based on logical reasoning with the known conditions, culminating in the numerical value of the final answer. Please follow the format of the following demonstrations, modify the original problem and make it more challenging. To ensure that there is a strict order in difficulty between the original problem and modified one, steps to solve the original should be included in that of the modified problem. In other words, steps to solve the original problem is a proper subset of that of the modified problem. Except for the modified #Problem#, you should also provide #Answer# to the modified #Problem#.

#Demonstrations#:

<insert demonstrations>

The above are some demonstrations showing how to modify the original problems. Please follow their format and modify the following problem:

#Problem#:

<insert the original problem>

#Answer#:

<insert the answer>

Please modified the above #Problem# and then provide #Answer# to the modified #Problem#:

Figure 11: Our prompt fed into GPT-4 for math data generation.

Prompt for Instruction Following Data Generation

#Instruction#:

I want you to act as a helpful assistant. Please help me modify some instruction following problems and make them harder. An instruction following problem datum consists of three parts: #Prompt#, #Constraint Type List#, and #Constraint Kwargs#. The #Prompt# consists of several constraints that guide the model to generate text. The #Constraint Type List# and #Constraint Kwargs# include the types and keyword arguments of the constraints contained within the #Prompt#, respectively. They are utilized to verify whether the text generated by the model meets the constraints. We provide a #Candidate Constraint Set# containing a variety of constraints. Please select an appropriate constraint from this set and follow the format of the demonstrations provided to add to the original #Prompt#. By doing so, you will create a more challenging new #Prompt#. Except for the modified #Prompt#, you should also provide #Constraint Type List#, and #Constraint Kwargs# to the modified #Prompt#.

#Candidate Constraint Set#:

<insert the candidate constraint set>

#Demonstrations#:

<insert demonstrations>

The above are some demonstrations showing how to modify the original problems. Please follow their format and modify the following problem:

#Prompt#:

<insert the original prompt>

#Constraint Type List#:

<insert the constraint type list>

#Constraint Kwargs#:

<insert the constraint keyword arguments>

Please modified the above #Prompt# and then provide #Constraint Type List# and #Constraint Kwargs# to the modified #Prompt#:

Figure 12: Our prompt fed into GPT-4 for instruction following data generation.