NPHardEval: Dynamic Benchmark on Reasoning Ability of Large Language Models via Complexity Classes

Lizhou Fan[1*] **Wenyue Hua**[2*] **Lingyao Li**[1] **Haoyang Ling**[1] **Yongfeng Zhang**[2]
[1]School of Information, University of Michigan, Ann Arbor, MI 48103
[2]Department of Computer Science, Rutgers University, New Brunswick, NJ 08854
{lizhouf, lingyaol, hyfrankl}@umich.edu, {wenyue.hua, yongfeng.zhang}@rutgers.edu
*Lizhou Fan and Wenyue Hua contribute equally.

Abstract

Complex reasoning ability is one of the most important features of Large Language Models (LLMs). Numerous benchmarks have been established to assess the reasoning abilities of LLMs. However, they are inadequate in offering a rigorous evaluation and prone to the risk of overfitting and memorization, as these publicly accessible and static benchmarks allow models to potentially tailor their responses to specific benchmark metrics, thereby inflating their performance. Addressing these limitations, we introduce a new benchmark **NPHard-Eval**. It contains a broad spectrum of 900 algorithmic questions belonging up to the NP-Hard complexity class, offering a rigorous measure of the reasoning ability of LLMs utilizing computational complexity. Moreover, this benchmark is designed with a *dynamic update mechanism*, where the datapoints are refreshed on a monthly basis. Such regular updates play a crucial role in mitigating the risk of LLMs overfitting or memorizing the benchmark, promoting a more accurate and reliable assessment of their reasoning capabilities. The benchmark dataset and code of NPHardEval are available at `https://github.com/casmlab/NPHardEval`.

1 Introduction

The advancement of LLMs has ushered in a transformative era in AI research (Fan et al., 2023b). One major advantage of LLM is its strong reasoning capabilities (Zhao et al., 2023). A range of benchmarks have been developed to assess the performance of LLMs in reasoning tasks (Cobbe et al., 2021; Valmeekam et al., 2022; Chen et al., 2023; Hendrycks et al., 2020, 2021). Despite these efforts, these benchmarks are not without their shortcomings. Firstly, they often fail to accurately characterize or classify the reasoning abilities of LLMs, leaving a gap in understanding the full extent of what LLMs can and cannot do. Secondly, there is a pronounced risk of models becoming overly specialized to these benchmarks, which may lead to an overestimation of their capabilities (Schaeffer, 2023). Lastly, the reliance on manual evaluation methods in some instances has proven to be inefficient, ineffective, and lacking in standardization (Frieder et al., 2023).

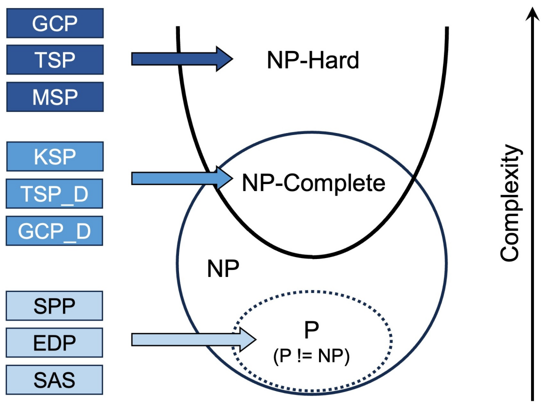

Figure 1: Computational complexity classes P, NP-complete, and NP-hard and corresponding tasks

Addressing the identified limitations and queries within the current benchmarking landscape, this paper introduces a novel benchmark, **NPHardEval**, which surpasses existing benchmarks in three critical aspects. Firstly, NPHardEval incorporates the established principles of computational complexity classes, offering a more rigorous and quantitative framework for assessing the reasoning capabilities of large language models. The benchmark is meticulously crafted, featuring nine reasoning tasks meticulously selected across three complexity classes—Polynomial-time complexity, NP-complete complexity, and NP-hard complexity—as delineated in (Johnson, 1990). Each class contains 100 instances, distributed over ten distinct levels of difficulty, enabling a comprehensive and measurable evaluation of LLMs' reasoning abilities. Secondly, NPHardEval benefits from an end-to-end automated framework for both task generation and results verification. This automation leverages mature algorithms for well-known tasks within the benchmark, facilitating the periodic update of data points on a monthly basis. Such a dynamic update mechanism significantly diminishes the risk of model overfitting, thereby maintaining the benchmark's stringency and applicability over time. Lastly, the benchmark's reliance on automatic verification for task solutions enhances the accuracy and reliability of evaluations, obviating the need for labor-intensive manual verification. This approach not only streamlines the evaluation process but also presents a theoretically intriguing opportunity to explore LLMs' proficiency in navigating the computational complexity hierarchy, with a particular focus on NP-hard and NP-complete problems (Johnson, 1990).

1.1 Research Questions

To elucidate the efficacy and comprehensiveness of the NPHardEval benchmark, our investigation will pivot around two principal research questions, which are as follows:

Model Benchmark Performance Our benchmark evaluates 12 models including closed-source models (GPT 4 Turbo (Achiam et al., 2023), Claude 2 (Anthropic, 2023a), GPT 3.5 Turbo (OpenAI, 2024), Claude Instant (Anthropic, 2023b), and PaLM 2 (Google, 2023)) and open-source models (Yi-34b (01-AI, 2023), Qwen-14b (Bai et al., 2023), Mistral-7b (Jiang et al., 2023), Phi-2 (Javaheripi and Bubeck, 2023), MPT-30b (MosaicML, 2023), Vicuna-13b (LMSYS, 2023), and Phi-1.5 (Li et al., 2023b)) across three complexity classes (P, NP-complete, NP-hard) each with 10 difficulty levels. This comparison sheds light on the relative strength and weakness of these models and determines the proficiency of them in solving progressively challenging problems, thus gauging their capability to handle tasks with escalating complexity.

Robustness of Benchmark Assessments This study examines whether the frequent updating of algorithmic benchmarks can effectively prevent the risk of the benchmark being "hacked". The dynamic updating of benchmarks is proposed as a strategy to reduce the likelihood of LLMs overfitting to these benchmarks. However, a pertinent question arises: does finetuning LLMs on benchmarks from the previous month lead to overfitting specific problem types? To explore this, we conducted an experiment where three open-source models – Phi-2, Mistral-7b, and Qwen-14b – were finetuned on five distinct versions of the benchmarks. The performance of each checkpoint of these models was evaluated on two versions of the benchmark, each differing in difficulty level. This approach allowed us to assess whether finetuning enables models to "hack" benchmarks of varying complexity.

2 Related Work

Reasoning ability of LLMs LLMs (Brown et al., 2020; Chowdhery et al., 2023; Chung et al., 2022) have made significant advancements in natural language processing and related fields. Recent research underscores the unprecedented reasoning abilities of LLMs in various fields, from biomedical and human-computer interaction research to humanities and social studies (Huang and Chang, 2022; Hua et al., 2023; Fan et al., 2023a; Gao et al., 2023; Li et al., 2023a). It has been discussed that these models exhibit "emergent" behaviors, including the ability to "reason" when they are large enough (Wei et al., 2022a; Schaeffer et al., 2023). By providing the models with the chain of thoughts with a simple prompt "Let's think step by step", these models are able to answer questions with explicit reasoning steps (Wei et al., 2022b). This has sparked considerable interest in the community since reasoning ability is a hallmark of human intelligence. Various variations of chain-of-thought have been developed to prompt models' reasoning ability (Kojima et al., 2022; Wang et al., 2022; Hua and Zhang, 2022), such as tree of thought (Yao et al., 2023), graph of thought (Besta et al., 2023), self-inspring technique (Wang et al., 2023). Later, various self-critique methods have been proposed to enhance LLM's reasoning performance. The Recursively Criticizes and Improves (RCI) approach, for example, iteratively refines outputs, proving more effective in automating computer tasks and elevating reasoning capabilities (Kim et al., 2023). Backward verification proposes an intuitive human-like mechanism for LLMs to self-check and improve their conclusions, reducing errors in reasoning tasks (Weng et al., 2022).

Benchmarks of LLMs' Performance Existing evaluation approaches predominantly rely on datasets comprising human-generated questions and their standard answers. For instance, MMLU

(Hendrycks et al., 2020) and GAOKAO (Zhang et al., 2023) both utilize human exam questions in their automated evaluations. Exam datasets such as GHOST (Graduate-Level High-Order Skill Tests) are utilized to assess LLMs' reasoning proficiency (Frieder et al., 2023). Nonetheless, the requirement for manual verification of answers in these datasets limits their practical utility. Big-Bench Hard (Suzgun et al., 2022), DROP (Dua et al., 2019), and HellaSwag (Zellers et al., 2019), while valuable, predominantly target multi-step reasoning, reading comprehension, and commonsense reasoning, respectively. They do not adequately prioritize complex logical reasoning in their assessment criteria. Zhu et al. (2023) proposes a dynamic graph-based reasoning benchmark whilst mainly focusing on polynomial time problems.

Other Benchmarks such as AlpacaEval (Dubois et al., 2023) and SuperCLUE (Xu et al., 2023) have attempted to incorporate open-ended questions in English and Chinese, respectively, to capture a diverse breadth of possible answers and enhance the comprehensiveness of LLM's evaluation. However, they are often constrained by language barriers and cultural contexts, potentially skewing the evaluation of reasoning abilities toward a specific scenario. Reasoning tasks should transcend linguistic and cultural specifics, focusing instead on universal logical principles.

The prevalent focus on question answering and math problems in current benchmarks insufficiently capture the essence of reasoning – the ability to logically process and deduce information beyond memorized knowledge. It also falls short on providing a rigorous metric on the reasoning ability. This gap highlights the importance of NPHardEval which is a dynamic logic-based reasoning benchmarks which provides a quantitative and rigorous evaluation on the logical reasoning capacity of LLMs.

3 Benchmark Construction

3.1 Complexity Classes

In our study, we employ the concept of complexity classes to categorize the reasoning tasks for LLMs. These classes are defined based on the computational resources, such as time or memory, required to solve the problems they contain (Johnson, 1990). Primarily, most complexity classes comprise decision problems that can be solved using a Turing machine, with differentiation based on their time or space (memory) requirements. For example, the class P includes decision problems that a deterministic Turing machine can solve in polynomial time. Tasks within this class often pose multi-dimensional cognitive challenges, enriching the evaluation framework of LLMs. This structured approach not only aids in assessing the reasoning capabilities of LLMs but also holds substantial relevance in various practical applications, particularly in optimization and decision-making process.

In particular, we use three complexity classes to define the task complexity in the benchmark, including P (polynomial time), NP-complete (nondeterministic polynomial-time complete), and NP-hard, which are increasingly complex in both the intrinsic difficulty and the resources needed to solve them. Figure 1 shows their relation regarding computational complexity in an Euler diagram. The details of the nine problems, including Graph Coloring Problem Optimization Version (GCP), Traveling Salesman Problem Optimization Version (TSP), Meeting Scheduling Problem (MSP), Knapsack Problem (KSP), Traveling Salesman Problem Decision Version (TSP-D), Graph Coloring Problem Decision Version (GCP-D), Shortest Path Problem (SPP), Edit Distance Problem (EDP), and Sorted Array Search (SAS), are provided in Appendix B. This approach aims to delineate the extent of complex reasoning achievable by LLMs, thus for each complexity class, we only choose tasks from the non-overlapping subset of the complexity class. In our selection criteria, we intentionally exclude tasks that demand intensive mathematical computations, such as matrix multiplication and logarithmic calculations. Thus, we do not list NP class (questions in NP but not P and not NP-complete), which is exemplified by the discrete logarithm and integer factorization problems, as the majority of such problems are characterized by their calculation-intensive nature (see details in Appendix C).

3.2 Difficulty Level for Tasks

NPHardEval categorizes each challenge into a hierarchy of difficulty, spanning from the simplest to the most difficult with 10 levels. This gradation allows for a nuanced assessment of an LLM's problem-solving abilities across a spectrum of increasingly difficult tasks. For instance, the GCP-D problem has difficulty levels 1 to 10 with questions of 6, 8, 10, 12, 14, 16, 18, 20, 22, and 24 average edges and 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15

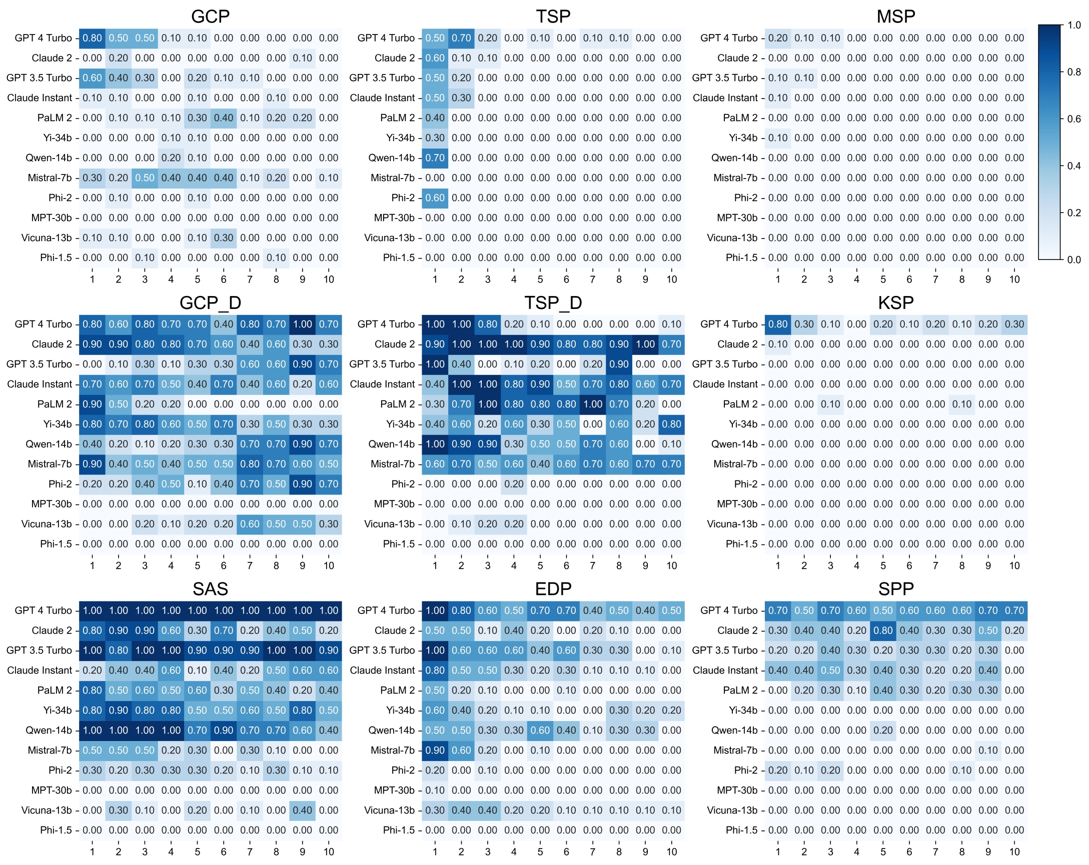

Figure 2: Zero-shot model performance on the nine tasks from P to NP-Complete bottom-up.

nodes. Beginning with graphs of 6 nodes and 6 edges, each subsequent level incorporates an additional 2 edges and 1 node, culminating in graphs of 24 edges and 15 nodes.

The difficulty level is not strictly bounded to a linear scaling of difficulty; rather, it is designed to explore the nuances of performance degradation. By observing how LLMs cope with an escalating series of challenges, we aim to identify the inflection point where the performance notably diminishes. This approach provides a comprehensive understanding of where LLMs excel and where they falter, informing potential pathways for the enhancement of their reasoning capabilities.

3.3 Data Synthesis

In the context of data synthesis for complex tasks, the approach can be categorized into two distinct methodologies, each corresponding to a different type of data structure: graph data (e.g., GCP) and linear data (e.g., MSP). The synthesis process in both cases is governed by a progression of complexity across a spectrum of predefined levels. This structured approach enables the creation of diverse datasets, suitable for evaluating and benchmarking LLMs' reasoning ability. We provide examples of the synthesized data and how they are used in prompts in Appendix A. We also justify the benchmark consistency and size in Appendix D.

Graph Data Synthesis The complexity in graph data synthesis escalates through a series of levels, each defined by a set of parameters that dictate the graph's size and intricacy. These parameters typically include the number of vertices, the number of edges, and the range of edge weights. At lower levels, graphs are simpler with fewer vertices and edges, and a limited range of edge weights. As the level increases, the graphs become progressively more complex, featuring more vertices, a higher density of edges, and a wider variety of edge weights.

Linear Data Synthesis In linear data synthesis, complexity is modulated by manipulating the length of the data array and the range of its constituent elements. Initial levels are characterized by shorter arrays with elements drawn from a narrow range. As the difficulty level ascends, the arrays lengthen, and the range of possible element values expands, thus introducing greater variability and complexity to the problem.

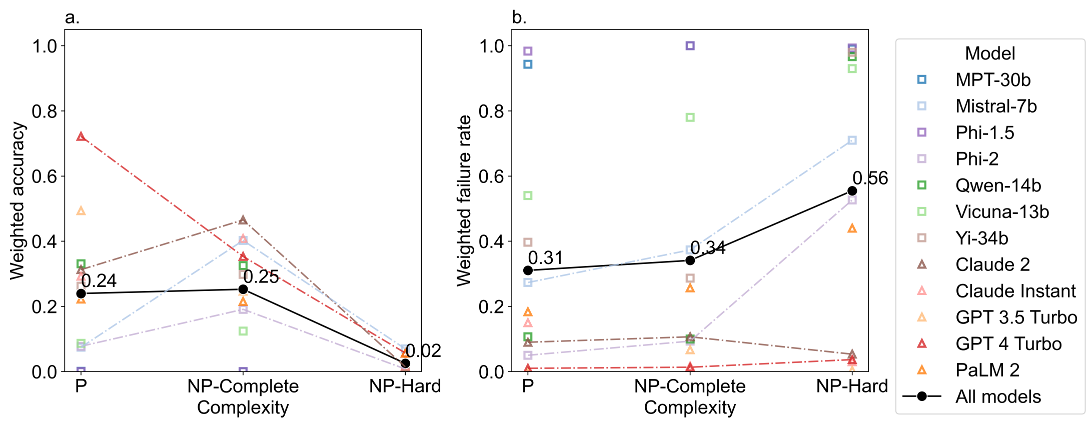

Figure 3: Model performance on different complexity problems: (a) weighted accuracy (b) (weighted) failure rate. Open models are denoted in squares and close models are denoted in triangles. Trends of metrics are demonstrated for models with outstanding performances in both weighted accuracy and failure rate, including both close-source (GPT 4 Turbo and Claude 2) and open-source (Mistral-7B and Phi-2) models.

4 Experimental Setting

This section presents the experiment setting to answer the three research questions. Our approach assessed 12 distinct LLMs, with a dichotomy between five proprietary (closed-source) models and five open-source models, including GPT 4 Turbo, Claude 2, GPT 3.5 Turbo, Claude Instant 1.2, PaLM 2, Vicuna-13b, Yi-34b, Mistral-7b, MPT-30b, Phi-1.5, Phi-2, and Qwen-14b.

4.1 Experiment 1: Model Performance Comparison

To evaluate the reasoning abilities of different LLMs through the NPHardEval benchmark, we employ a comparative experimental design. We use zero-shot prompts containing task descriptions and a specific question as the foundational measure of performance. Each model's performance was evaluated based on two primary metrics: weighted accuracy and failure rate across the different complexity classes of problems, as we will discuss in Section 4.3. We also conduct few-shot experiments on SAS and EDP using GPT-4 generated examples in prompts and observe interesting performance change across different types of prompts.

4.2 Experiment 2: Benchmark Robustness

The primary objective of this experiment is to ascertain whether it is possible to "hack" our benchmark by finetuning models on its previous versions. To simulate this, we constructed five versions of the benchmark, maintaining a consistent difficulty level. Additionally, we utilize two distinct versions of the benchmark, each varying in difficulty, to evaluate the potential for hacking under varying conditions. To replicate the progression of time, models were finetuned sequentially on one to five benchmarks, each finetuned checkpoint is tested on the two distinct benchmarks for evaluation.

The experiment involved finetuning three high-performing open-source models: Phi-2, Mistral-7b, and Qwen-14b. Due to constraints in computing resources, the Yi-34b model was not included in the finetuning process. For the finetuning process, we employed the QLoRA technique, applying specific hyperparameters: batch size set to 8, a single epoch, a warmup proportion of 0.03, a learning rate of 1e-4, lora_r at 64, lora_alpha at 16, and a lora_dropout of 0.1. This approach aims to rigorously test the robustness of our benchmark against potential overfitting strategies.

4.3 Evaluation Metrics

To evaluate the reasoning ability of LLMs, we utilize two metrics, the Weighted Accuracy and the Failure Rate, to comprehensively quantify the correctness of LLMs' reasoning outputs.

Weighted Accuracy (WA) is calculated for each problem either through the comparison with the correct answer or through step-by-step results checking, for those problems without the only answer. To better represent the comparative accuracy, we assign weights to different difficulty levels so that each level has a weight corresponding to its relative importance or challenge. Higher difficulty levels are given more weight in a linear manner (e.g., level 1 has weight 1, level 2 has weight 2,

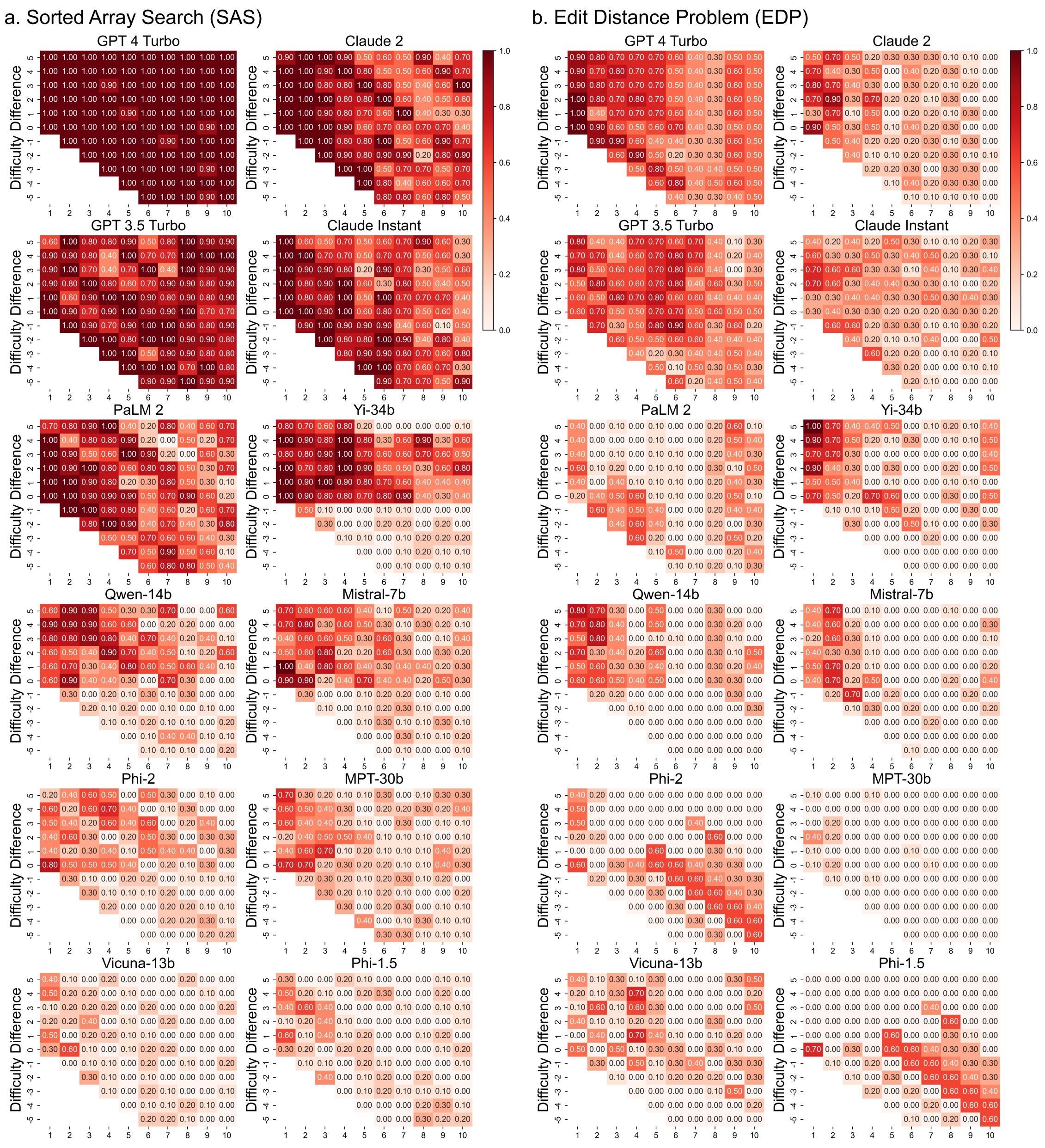

Figure 4: Few-shot learning results on SAS and EDP of all models

etc.). The Weighted Accuracy is defined as:

$$WA = \frac{\sum_{i=1}^{10}(w_i \times A_i)}{\sum_{i=1}^{10} w_i}$$

where w_i represents the weight assigned to difficulty level i, from 1 to 10, and A_i is the accuracy at that level.

Failure Rate (FR) is a measure used to assess the frequency of unsuccessful outcomes across the different problems and difficulty levels. It is particularly useful for identifying cases where an LLM's result does not comply with the expected output format. The Failure Rate is calculated by considering the proportion of failed attempts relative to the total number of attempts for each difficulty level. An attempt is defined as failed if the model generates results that cannot be successfully parsed in all endpoint calls, and we set the maximum times of try as 10. For each problem, the Failure Rate is then aggregated across all difficulty levels, taking into account the total 10 attempts at each level. The formal definition of Failure Rate is given by:

$$FR = \frac{\sum_{i=1}^{10} F_i}{100}$$

where F_i denotes the number of failed attempts at difficulty level i.

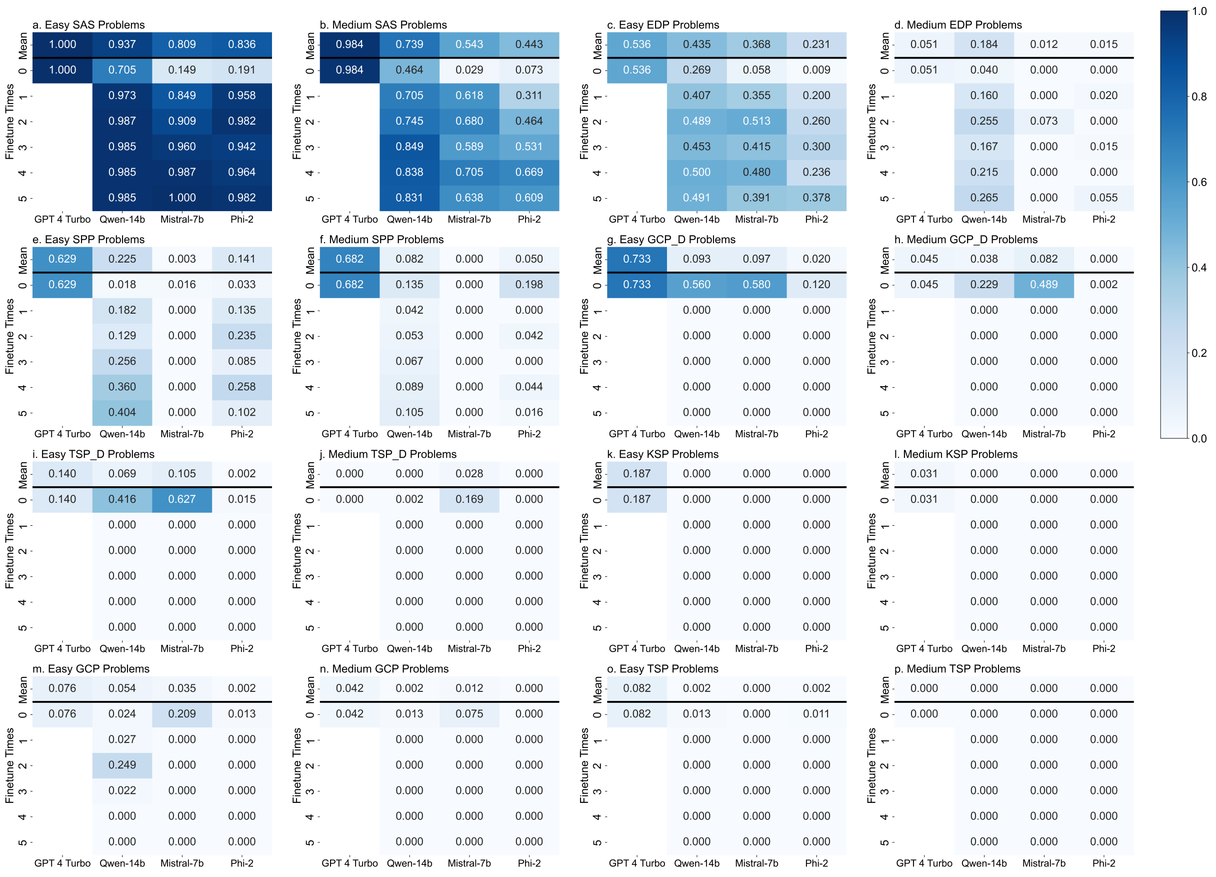

Figure 5: Model's robustness on different problems and difficulty levels.

5 Results

5.1 Reasoning Ability of Foundation Models

Experiment 1 focuses on a comprehensive comparison among various foundation models and across complexity classes and difficulty levels. In Figure 2, we present the overall zero-shot accuracy for each problem, providing a visual representation of the performance of different models.

Our observations reveal that closed-source models generally demonstrate higher accuracy and a lower rate of failure compared to their open-source counterparts. Notably, GPT-4 Turbo often emerges as the frontrunner in performance across the majority of tasks, indicating its superior problem-solving capabilities, while Claude 2, on the other hand, often performs the best on medium-level (NP-complete) complexity in zero-shot settings. Within the realm of open-source models, Yi-34b, Qwen-14b, and Mistral-7b distinguish themselves by significantly outperforming other models in this category. We observe a disparity between the performance of these three models and other open-source options, highlighting a notable performance gap and suggesting that these models possess more advanced reasoning abilities.

In particular, we use the weighted accuracy and the failure rate metrics to further quantify different models' performance. The trends observed below in both weighted accuracy and failure rates point to a nuanced understanding of the capabilities and limitations of current LLMs. These observations are also supported by statistical tests within and across complexity classes, indicating the model performance differences among complexity classes (see Appendix E for details).

Weighted Accuracy Figure 3(a) shows the weighted accuracy for different models across problem complexities. The general trend is all models experiencing a decrease in accuracy as problem complexity increased. Notably, there are two detailed findings for overall reasoning ability change. First, regarding the performance decay speed, among the 12 models we tested, the average performance demonstrated a higher accuracy at the P and NP-Complete complexity levels (with similar weighted accuracies of 0.24 and 0.25) but saw a sharper decline as the problems became more complex when proceeding to the NP-hard level (with a weighted accuracy of 0.02). There is a *performance decay* on average when models are tested against NP-Hard problems. Second, close-source models usually perform better than open-source models: there are more triangles in the upper locations than squares in Figure 3(a).

Failure Rate Figure 3(b) indicates that the failure rates mirrored the trends observed in weighted accuracy but in reverse. On average, the models showed an increase in failure rates corresponding to the complexity of the problems. Open-source models fail more often (with more squares on the top) than the close-source models (with more triangles on the bottom), indicating close-source's models advanced ability in following the prompt to understand the reasoning problems and generate answers with correct format.

Few-shot Learning Results Figure 4 illustrates the outcomes of few-shot learning experiments on SAS and EDP across various models, each subjected to 11 distinct few-shot prompts. These prompts were systematically varied in terms of difficulty level, ranging from -5 to +5 relative to the difficulty level of the target question each model was tasked to solve. The results underscore a notable enhancement in model performance while exhibiting significant variability across the different prompts on open-source models. Such observations indicate a potential limitation in the open-source models' capacity for acquiring the underlying task-solving skills and generalizing from the examples from prompts. This highlights a crucial area for further investigation and development.

5.2 Evaluating Benchmark Robustness

In Experiment 2, we explore the robustness of benchmark against hacking attempts through a process of finetuning on pairs of question and gold answer. We experiment using 3 well-performing open-source models: Qwen-14b, Mistral-7b, and Phi-2 on two versions of benchmarks. Figure 5 presents the result[1]: each problem has two graphs with one displaying evaluation results at difficulty levels 1-10 and one displaying evaluation results at difficulty levels 11-20. In each graph, the first row indicates the mean accuracy of each model, averaged over the outcomes at 5 finetuning checkpoints, ranging from tuning using zero (no finetuning) to five distinct benchmarks.

Our findings are twofold: (1) While finetuning yields improvements in solving polynomial-time problems, its impact on the more complex NP-complete and NP-hard problems are negative. This suggests the inherent difficulty of hacking NP-complete, and potentially NP-hard, problems through the basic finetuning with question-and-answer approach. Manual annotation of the chain-of-thought, which is not provided in the benchmarks, could potentially enhance effectiveness, albeit with challenges in annotation. (2) Finetuning appears beneficial for performance within the same difficulty level of all P problems, yet shows limited out-of-distribution (OOD) adaptability and struggles to generalize to more difficult problems (as evidenced in graphs a and c) except for SAS. For instance, Qwen-14b demonstrates notable proficiency on SPP challenges at levels 1-10 following finetuning; its performance is comparable to that of GPT-4. However, its performance significantly diminishes on SPP problems at levels 11-20, even underperforming compared to its unfinetuned checkpoint. This indicates that finetuning on these benchmarks can only benefit very simple questions such as SAS but could potentially impede generalization capabilities and render finetuning hacking useless. In conclusion, our benchmark is challenging to hack due to two primary factors: (1) the inherent complexity of NP-complete and NP-hard problems, which are difficult to learn solely from question-answer pairs, and (2) the propensity for P problems to become overfitted through finetuning on these pairs, while the real "reasoning" ability can be easily exposed by increasing the problem difficulty level.

6 Conclusion

We present a novel benchmark, NPHardEval, designed to rigorously evaluate LLMs' reasoning capabilities across a spectrum of complex tasks, up to the complexity class of NP-hard. By eschewing standard QA formats in favor of complex, logic-oriented problems, this benchmark aims to provide a more accurate measure of a model's reasoning prowess. This approach is crucial for developing LLMs capable of handling sophisticated, real-world tasks that demand high-level cognitive processing, steering the evaluation of LLMs from potentially "useful" to fundamentally "logical".

[1]We do not present the result on MSP as this problem does not have a fixed solution and no finetuning was conduct on it.

7 Acknowledgement

We extend our sincere gratitude to Libby Hemphill for her invaluable support in this work. We also thank Jinkui Chi, who kindly contributed to the maintenance of the benchmark's code repository and user guidance. Additionally, we are grateful for the diverse feedback we received from Siqi Liu and many others, which illuminated our path towards enhancing the quality of this paper.

Limitations

While our study offers a novel approach to assessing the reasoning abilities of LLMs, it is important to reflect on the limitations of our current methodology to provide a comprehensive understanding and guide future research.

Task Complexity's Comparison A significant limitation lies in the scope of our task selection and the definition of complexity within our benchmark. While we have delineated criteria for task selection in the appendix, a more resource-intensive approach could involve the inclusion of a larger variety of questions for each task type, enhancing the depth and breadth of our evaluation. Additionally, our current approach to defining complexity is based on a linear increment of weights. This simplistic weighting heuristic may not accurately represent the nuanced complexity increase in real-world tasks. More experimental work is needed to refine this approach and determine the most effective weight assignment that truly reflects the intricacies of task complexity.

Randomness Another critical aspect to consider is the inherent randomness in the generation of responses by LLMs. This randomness can introduce variability in performance, making it challenging to draw consistent conclusions about a model's reasoning capabilities. **Notably, decision questions in the NP-complete level, including GCP-D and TSP-D, use true or false results as the evaluation criteria. Thus, it is hard to directly rule out the random positive cases, although the model may not go through a correct reasoning process, leading to potentially inflated performance.** Addressing this issue requires a more nuanced approach to evaluating responses, possibly through repeated trials or the incorporation of statistical methods to account for this variability.

Model Updates and Emergence The fast-paced evolution of LLMs also presents a significant challenge. With the continuous version updates and emergence of advanced models like Gemini Ultra (DeepMind, 2023) and miniCPM, as well as an increasing number of open-source options, the analysis based on our benchmark may quickly become outdated. Thus we will monitor and experiment on new models, together with the LLMs research community, to keep pace with these rapid developments is crucial for maintaining the relevance and applicability of our findings. This dynamic nature of the field necessitates a flexible and adaptable approach to benchmarking, where updates and revisions are integral to the evaluation process.

Future research should aim to expand the scope and depth of task selection, refine the complexity definition, account for generation randomness, and adapt to the evolving landscape of LLMs. Addressing these challenges will enhance the accuracy and relevance of our benchmark, contributing to the development of LLMs that are capable of sophisticated reasoning in complex, real-world scenarios.

Ethics

In this paper, we present a novel benchmark aimed at rigorously evaluating the reasoning capabilities of LLMs through algorithmic questions up to the NP-Hard complexity class. The introduction of a dynamic update mechanism is a significant innovation designed to prevent the overfitting of LLMs to our benchmark, thereby fostering a more genuine assessment of their reasoning prowess. In the development process and release of the benchmark, we are aware of the potential ethical challenges and address them to our best knowledge as follows.

Fair and Objective Evaluation We have taken significant measures to ensure that the NPHardEval benchmark is unbiased and equitable, providing a fair ground for evaluating all LLMs regardless of their origin or the entity developing them. Our selection of algorithmic questions and the subsequent categorization into complexity classes were conducted with utmost objectivity, aiming to reflect a broad spectrum of reasoning capabilities without favoring any specific model or approach.

Data Privacy and Security Although our benchmark does not directly involve human subjects or personally identifiable information, we are committed to maintaining high standards of data privacy and security. The dynamic update mechanism, while designed to refresh datapoints regularly, will only keep the most current three months of data available, aiming to prevent benchmark hacking.

Transparency and Reproducibility We have made efforts to ensure that our methodology, including the design and implementation of the dynamic update mechanism, is transparent and well-documented. This allows for the reproducibility of our benchmark by the research community, facilitating further studies and advancements in the field. Our code repository is currently anonymously available online.

Responsible Use of AI Recognizing the potential of LLMs to impact society significantly, our research is grounded in the principle of promoting the responsible development and use of AI technologies. The NPHardEval benchmark is intended to contribute positively to the field by encouraging the development of LLMs that are not only advanced in their reasoning capabilities but are also ethically aligned and beneficial for society.

Avoidance of Overfitting A primary ethical concern in AI benchmarking is the risk of models being overfitted to specific datasets, which can misrepresent their true capabilities. Our dynamic update mechanism directly addresses this concern, promoting an ethical approach to AI evaluation that emphasizes genuine progress over superficial performance metrics.

Accessibility and Inclusivity We are committed to ensuring that our benchmark is accessible to a wide range of researchers and practitioners in the AI community. We include detailed guidance on how to use our benchmark. This inclusivity fosters a diverse and rich environment for AI research and development, where insights and innovations can emerge from various perspectives and backgrounds.

References

01-AI. 2023. Building the next generation of open-source and bilingual llms.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shamim Ahmed. 2012. Applications of graph coloring in modern computer science. *International Journal of Computer and Information Technology*, 3(2):1–7.

Anthropic. 2023a. Claude 2.

Anthropic. 2023b. Releasing claude instant 1.2.

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*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Miquel Bofill, Jordi Coll, Marc Garcia, Jesús Giráldez-Cru, Gilles Pesant, Josep Suy, and Mateu Villaret. 2022. Constraint solving approaches to the business-to-business meeting scheduling problem. *Journal of Artificial Intelligence Research*, 74:263–301.

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.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. *arXiv preprint arXiv:2305.12524*.

Michael Cho. 2019. The knapsack problem and its applications to the cargo loading problem. *Anal. Appl. Math*, 13:48–63.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

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

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. 2022. *Introduction to Algorithms, fourth edition*. MIT Press.

Google DeepMind. 2023. Gemini.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*.

Lizhou Fan, Sara Lafia, Lingyao Li, Fangyuan Yang, and Libby Hemphill. 2023a. Datachat: Prototyping a conversational agent for dataset search and visualization. *arXiv preprint arXiv:2305.18358*.

Lizhou Fan, Lingyao Li, Zihui Ma, Sanggyu Lee, Huizi Yu, and Libby Hemphill. 2023b. A bibliometric review of large language models research from 2017 to 2023. *arXiv preprint arXiv:2304.02020*.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. 2023. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*.

Zhenxiang Gao, Lingyao Li, Siyuan Ma, Qinyong Wang, Libby Hemphill, and Rong Xu. 2023. Examining the potential of chatgpt on biomedical information retrieval: Fact-checking drug-disease associations. *Annals of Biomedical Engineering*, pages 1–9.

Yingqiang Ge, Wenyue Hua, Jianchao Ji, Juntao Tan, Shuyuan Xu, and Yongfeng Zhang. 2023a. Openagi: When llm meets domain experts. *arXiv preprint arXiv:2304.04370*.

Yingqiang Ge, Yujie Ren, Wenyue Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023b. Llm as os (llmao), agents as apps: Envisioning aios, agents and the aios-agent ecosystem. *arXiv preprint arXiv:2312.03815*.

Google. 2023. Palm 2.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2023. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *arXiv preprint arXiv:2311.17227*.

Wenyue Hua and Yongfeng Zhang. 2022. System 1+ system 2= better world: Neural-symbolic chain of logic reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 601–612.

Jie Huang and Kevin Chen-Chuan Chang. 2022. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Glenn D Israel et al. 1992. Determining sample size.

Mojan Javaheripi and Sébastien Bubeck. 2023. Phi-2.

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*.

David S Johnson. 1990. A catalog of complexity classes. In *Algorithms and complexity*, pages 67–161. Elsevier.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2019. Sosd: A benchmark for learned indexes.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Lingyao Li, Lizhou Fan, Shubham Atreja, and Libby Hemphill. 2023a. "hot" chatgpt: The promise of chatgpt in detecting and discriminating hateful, offensive, and toxic comments on social media. *arXiv preprint arXiv:2304.10619*.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023b. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*.

Carla Negri Lintzmayer, Mauro Henrique Mulati, and Anderson Faustino da Silva. 2011. Register allocation with graph coloring by ant colony optimization. In *2011 30th International Conference of the Chilean Computer Science Society*, pages 247–255. IEEE.

LMSYS. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90% chatgpt quality.

Robert C MacCallum, Keith F Widaman, Shaobo Zhang, and Sehee Hong. 1999. Sample size in factor analysis. *Psychological methods*, 4(1):84.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*.

Janice M Morse. 1992. *Qualitative health research*, volume 22. Sage Newbury Park, CA.

MosaicML. 2023. Mpt-30b: Raising the bar for open-source foundation models.

OpenAI. 2024. Gpt-3.5 turbo.

Roberto Roberti and Mario Ruthmair. 2021. Exact methods for the traveling salesman problem with drone. *Transportation Science*, 55(2):315–335.

Rylan Schaeffer. 2023. Pretraining on the test set is all you need. *arXiv preprint arXiv:2309.08632*.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? *arXiv preprint arXiv:2304.15004*.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*.

Bo Sun, Ali Zeynali, Tongxin Li, Mohammad Hajiesmaili, Adam Wierman, and Danny HK Tsang. 2020. Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(3):1–32.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*.

Boshi Wang, Xiang Deng, and Huan Sun. 2022. Iteratively prompt pre-trained language models for chain of thought. *arXiv preprint arXiv:2203.08383*.

Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Yingzhen Yang. 2023. Recmind: Large language model powered agent for recommendation. *arXiv preprint arXiv:2308.14296*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

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

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. 2023. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*.

Yixuan Weng, Minjun Zhu, Shizhu He, Kang Liu, and Jun Zhao. 2022. Large language models are reasoners with self-verification. *arXiv preprint arXiv:2212.09561*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Liang Xu, Anqi Li, Lei Zhu, Hang Xue, Changtai Zhu, Kangkang Zhao, Haonan He, Xuanwei Zhang, Qiyue Kang, and Zhenzhong Lan. 2023. Superclue: A comprehensive chinese large language model benchmark. *arXiv preprint arXiv:2307.15020*.

Taro Yamane. 1973. Statistics: An introductory analysis.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. 2023. Dyval: Graph-informed dynamic evaluation of large language models. *arXiv preprint arXiv:2309.17167*.

A Examples of Synthesized Data, the Corresponding Prompts, and LLMs' Outputs

To further demonstrate the synthesized Data, the corresponding prompts, and LLMs' outputs, we choose two specific problems with different attributes, including the EDP problem from the P complexity class with linear data synthesis and the GCP problem from the NP-Hard complexity class with graph data synthesis. We provide the zero-shot prompt for these questions and the output based on the GPT 4 Turbo. The details of all prompts and results available at https://anonymous.4open.science/r/NPHardEval-5EF3.

A.1 Example for P: EDP

```
# Synthesized Data
{
    "string_a": "cef",
    "string_b": "ccb"
}

# Prompt with Synthesized Data
The Edit Distance Problem (EDP) involves finding the minimum number of operations required to transform one string into another, where each operation is either an insertion, deletion, or substitution of a single character.
Find the minimum number of operations required to transform the first string cef into the second string ccb. The operations are insertion, deletion, and substitution of a single character, each requiring 1 edit operation.
Please provide the minimum number of operations required to transform the first string into the second string. Offer a brief explanation of your methodology. Keep your response concise and focused.
Enclose your output within <root></root> tags. Present your reasoning in <reasoning></reasoning> tags and the final minimum number of operations in <final_answer></final_answer> tags, like <final_answer>{'Operations': 'MINIMUM_NUMBER_OF_OPERATIONS'}</final_answer>.

# Output (formatted for easier reading)
<root>
    <reasoning>
        To transform \"cef\" into \"ccb\", we can follow these steps:\n
        1. Substitute 'e' with 'c' (cef -> ccf)\n
        2. Substitute 'f' with 'b' (ccf -> ccb)\n
        This results in two substitution operations.
    </reasoning>
    <final_answer>
        {"Operations": 2}
    </final_answer>
</root>
```

Listing 1: EDP Example

A.2 Example for NP-Hard: GCP

```
# Synthesized Data
c This is a generated graph with chromatic number 2
p edge 6 4
e 1 6
e 2 6
e 3 4
e 3 5

# Prompt with Synthesized Data
Graph coloring refers to the problem of coloring vertices of a graph in such a way that no two adjacent vertices have the same color.
There are 6 vertices 1 to 6 in a graph. You may use 4 colors with alphabats from A, B, C,... to color the graph.
Please label every vertex, even if it is disconnected from the rest of the graph. Please provide each vertex's color. Do not skip any vertices. You should also provide very short step by step reasoning. Do not use multiple lines and try your best to save output tokens.
Your output should contain two parts enclosed by <root></root>. First, your step by step reasoning wraped by <reasoning></reasoning>. Second, the final output of all vertex numbers and their associated colors, wrapped by final_answer tag, like <final_answer>{0:'COLOR_1', 1:'COLOR_2', ...}</final_answer>.
The graph is below:
    Vertex 1 is connected to vertex 6.
    Vertex 2 is connected to vertex 6.
    Vertex 3 is connected to vertex 4.
    Vertex 3 is connected to vertex 5.

# Output (formatted for easier reading)
<root>
    <reasoning>
        Start with vertex 1, color it A; color adjacent vertex 3 with B; vertex 2 can be A as it's not adjacent to 1; vertex 4 connected to 2 must be different, use B; vertex 5 connected to both 2 and 3, use C; vertex 6 connected to 3 and 4, use A.
    </reasoning>
    <final_answer>
        {1:'A', 2:'A', 3:'B', 4:'B', 5:'C', 6:'A'}
    </final_answer>
```

```
</root>
```

Listing 2: GCP Example

A.3 Example of a Failed Case and Error Analysis

As a representative failed case, in an Edit Distance Problem (EDP), we have the following results:

```
# Output (formatted for easier reading)
{
    "output": {"Operations": "5"},
    "correctness": [false, "The reported
     edit distance (5) is incorrect.
   Actual distance: 4."],
    "reasoning": "To transform \"ieebi\"
     into \"ihahc\", we can follow these
     steps:\n
        1. Substitute 'e' with 'h' (
   ieebi -> ihhbi)\n
        2. Delete 'h' (ihhbi -> ihbi)\n
        3. Substitute 'b' with 'a' (ihbi
     -> iha)\n
        4. Insert 'h' (iha -> ihah)\n
        5. Substitute 'i' with 'c' (ihah
     -> ihahc)\n
        This results in a total of 5
   operations. It's possible that there
     might be a more efficient sequence
   of operations, but without
   performing a detailed dynamic
   programming analysis (such as using
   the Wagner-Fischer algorithm), this
   is a reasonable estimate."
}.
```

Listing 3: EDP Failed Case Example

The correct answer for this EDT problem should be 4, and here is our analysis. In this case, the correct steps should be:

- We start with two strings, "ieebi" and "ihahc". Our goal is to transform "ieebi" into "ihahc" using a minimum number of operations: insertion, deletion, or substitution.
- "i" and "i" are the same, so no operation is needed here. We move to the next character in both strings.
- We have "e" in the first string and "h" in the second. They are different, so we need to perform an operation. The best choice here is substitution (changing "e" to "h"), as it directly aligns with our target string. This counts as 1 operation.
- Now we have "e" in the first string and "a" in the second. Again, they are different, necessitating another substitution ("e" to "a"). This is our second operation.
- The fourth characters are "b" and "h". They are different, requiring a substitution ("b" to "h"). This is the third operation.
- Finally, we compare "i" and "c". These are different, requiring one last substitution ("i" to "c"). This brings our total to four operations.

The main problem occurs in "3. Substitute 'b' with 'a' (ihbi -> iha)", where the substitution did not influence the character 'i' at the end. Therefore, there is no need for additional insertion. Since there are four different letters in total, 4 substitutions will work. The main reason why this happens because of the model's ignoring of information, where it wrongly get rid of a letter. This error might due to the hallucination of a in-step generated fact (in our case the word) in the process of reasoning.

B Details of Complexity Classes

There are nine category problems (tasks) in total in our benchmark and each complexity class have three unique problem categories.

B.0.1 P (Polynomial time) Tasks

This class consists of tasks that can be solved by a deterministic Turing machine in polynomial time. Essentially, it represents tasks that are efficiently solvable. We include three P problems in the benchmark, namely Sorted Array Search (SAS), Edit Distance Problem (EDP), and Shortest Path Problem (SPP).

Sorted Array Search (SAS) SAS is about finding the position of a target value after sorting a given array. Given an array A of n elements and a target value T, the goal is to determine the index at which T is located in A after sorting. Renowned algorithms like binary search efficiently accomplish this task by iteratively halving the search interval, operating in logarithmic time. The problem can be formally stated as finding an index i such that $A[i] = T$, or determining that no such index exists. It is commonly used in databases and search engines to quickly find specific data within a large dataset (Kipf et al., 2019). SAS consists of 10 complexity levels, with each level designated by an array length (starting from 3 up to 12) and a number range that extends from (1, 15) to (1, 60). The complexity scales with each level by increasing the array length by one and the number range's upper limit by 5, thereby broadening the scope of sequential assignments required.

Edit Distance Problem (EDP) EDP is about finding the minimum number of operations required to transform one string into another. Given two strings, A and B, of lengths m and n respectively, the aim is to determine the minimum number of operations needed to convert A into B. The allowable operations are insertion, deletion, and substitution of a single character. Formally, the problem can be defined as finding a minimum number d such that string A can be transformed into string B using d operations. This algorithm has a time complexity of $\mathcal{O}(ab)$ where a and b are the lengths of the strings. When the full dynamic programming table is constructed, its space complexity is also $\mathcal{O}(ab)$. EDP has widespread applications, especially in fields like computational biology for sequence alignment, natural language processing for spell checking and correction, and in data analysis for measuring similarity between data strings. EDP is structured into 10 complexity levels, tailored to measure the minimal number of edits required to transform one string into another. Each level is characterized by two strings whose lengths are equal and progressively increase from 3 to 12 characters from Level 1 to Level 10. Concurrently, the character range for constructing these strings is expanded, starting with the first 6 letters and extending by 2 additional letters at each subsequent level.

Shortest Path Problem (SPP) SPP is about finding the shortest path between two nodes in a non-negative weighted graph. In our experiments, we ask for the shortest path between the first and last nodes. Given a graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}$ assigning weights to edges, and two vertices u and v in V, the task is to find the path from u to v that minimizes the total weight. This is often solved using Dijkstra's algorithm which systematically expands the shortest path from the starting node until it reaches the target node. Formally, the problem is to find a path $P = (v_1, v_2, ..., v_k)$, where $v_1 = u$ and $v_k = v$, such that the sum of weights of consecutive edges in P, $\sum_{i=1}^{k-1} w(v_i, v_{i+1})$, is minimized. This problem can be used in network routing, GPS navigation systems, and logistics to find the shortest or most efficient path between two points. It helps in reducing travel time and costs in transportation and communication networks. SPP involves determining the shortest path across different complexity levels, ranging from Level 1 to Level 10. Each level is characterized by an increasing number of nodes (starting from 4 to 13), edges (5 to 14), and a maximum weight (6 to 15) that escalates linearly with each level. The complexity of the problem scales by adding one node, one edge, and increasing the maximum weight by one for each subsequent level.

B.0.2 NP-complete problems

This is a subset of NP. A problem is NP-complete if it is in NP and as hard as any problem in NP. If any NP-complete problem can be solved in polynomial time, then every problem in NP can also be solved in polynomial time. We include three NP-complete problems that are not in P in the benchmark, namely Traveling Salesman Problem Decision Version (TSP-D), Graph Coloring Problem Decision Version (GCP-D), and Knapsack Problem (KSP).

Traveling Salesman Problem (Decision Version, TSP-D) TSP-D is concerned with determining if a salesman can complete a route, visiting each city at least once, with the total travel distance being less than a specified value. Given a complete graph $G = (V, E)$ with vertices V representing cities and edges E representing paths between cities, each edge (i, j) is assigned a distance $d(i, j)$. The decision version of this problem asks whether there exists a tour (a sequence of cities) such that the total distance of the tour is less than or equal to a given value D. Formally, the problem can be stated as finding a permutation P of the set of cities $1, 2, ..., n$ that satisfies the condition $\sum_{i=1}^{n-1} d(P(i), P(i+1)) + d(P(n), P(1)) \leq D$. This problem is useful in logistics and supply chain management in planning efficient delivery routes and schedules (Roberti and Ruthmair, 2021). TSP-D configuration spans 10 complexity levels with node counts from 4 to 13, similar to the TSP. It also introduces a threshold of 0.75, setting factor of the allowed travel distance to the total possible distance.

Graph Coloring Problem (Decision Version, GCP-D) GCP-D involves determining if it is possible to color the vertices of a graph using a given number of colors so that no two adjacent vertices share the same color. Given an undirected graph $G = (V, E)$, with V representing vertices and E representing edges, the goal is to find out if there is a way to assign one of k colors to each vertex such that for any edge $(u, v) \in E$, the vertices u and v have different colors. The formal statement is to determine if there exists a coloring function $c : V \to 1, 2, ..., k$ such that for every edge $(u, v) \in E$, $c(u) \neq c(v)$. It has wide applications in Round-Robin Sports Scheduling, Aircraft scheduling, and Biprocessor tasks (Ahmed, 2012). GCP-D has difficulty levels 1 to 10 with questions of 6, 8, 10, 12, 14, 16, 18, 20, 22, and 24 average edges and 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15 nodes. Beginning with graphs of 6 nodes and 6 edges, each subsequent level incorporates an additional 2 edges and 1 node, culminating in graphs of 24 edges and 15 nodes.

Knapsack Problem (KSP) KSP asks whether a subset of items can be chosen to fit into a knapsack of fixed capacity without exceeding it, while also maximizing the total value of the selected items. Consider a set of items, each with a weight w_i and a value v_i, and a knapsack with a weight capacity W. The problem is to select a subset of these items such that the total weight does not exceed W and the total value is maximized. Formally, let x_i be a binary variable indicating whether item i is included in the knapsack ($x_i = 1$) or not ($x_i = 0$). The problem can be stated as maximizing $\sum_{i=1}^{n} v_i x_i$ subject to the constraint $\sum_{i=1}^{n} w_i x_i \leq W$, where n is the number of items. It is used in resource allocation and budgeting where the goal is to maximize the total value of a selection under a weight or cost constraint. Applications include cargo loading, and electric vehicle charging (Sun et al., 2020; Cho, 2019).KSP is organized into 10 levels, each marked by an ascending number of items (from 4 to 13), a weight range (1 to the level number), a value range (identical to the weight range), and a knapsack capacity that starts at 20 and increases by 5 with each level. The complexity elevates by broadening the weight and value ranges, adding more items, and enlarging the knapsack's capacity.

B.0.3 NP-hard problems

These problems are at least as hard as the hardest problems in NP. They may not necessarily be in NP (i.e., they may not have solutions verifiable in polynomial time) but solving an NP-hard problem in polynomial time would imply that P = NP. We include three NP-hard problems that are not reducible to NP-complete problems in the benchmark, namely Traveling Salesman Problem Optimization Version (TSP), Graph Coloring Problem Optimization Version (GCP), and Meeting Scheduling Problem (MSP).

Traveling Salesman Problem (Optimization Version, TSP) TSP-O involves finding the shortest route for a salesman to visit each city exactly once and return to the starting city. Given a complete graph K_n with n vertices, where each vertex represents a city and each edge (i, j) is assigned a non-negative cost or distance $d(i, j)$, the problem is to find the shortest possible route that visits each city exactly once and returns to the origin city. Formally, let P be a permutation of the set of cities $1, 2, ..., n$ representing the order in which the cities are visited. The traveling salesman problem can be formulated as finding the permutation P that minimizes the total travel cost, given by the function $f(P) = d(P(n), P(1)) + \sum_{i=1}^{n-1} d(P(i), P(i+1))$. This problem is important in operational research and logistics to find the most efficient route to visit multiple locations and return to the origin, particularly route planning for delivery services, maintenance operations, and sales. TSP is structured across 10 complexity levels, each defined by a set number of nodes ranging from 4 to 13. The complexity of the problem increases with the addition of more nodes, enhancing the challenge of finding the shortest possible route that visits each node exactly once and returns to the starting point.

Graph Coloring Problem (Optimization Version, GCP) GCP-O refers to the problem of coloring vertices of a graph in such a way that no two adjacent vertices have the same color. Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, assign a color to each vertex such that no two adjacent vertices have the same color. Formally, let $c : V \to C$ be a function that assigns a color from a set of colors C to each vertex in V. The graph coloring problem can be formulated as finding a proper coloring, i.e., a function c such that for every edge $(u, v) \in E$, $c(u) \neq c(v)$. This problem is used in constraint satisfaction problems and applied in exam timetabling and register allocation in compilers (Lintzmayer et al., 2011). GCP is devised with 10 levels of complexity, featuring questions with average edges ranging from 6 to 24 and nodes from 6 to 15. Beginning with graphs of 6 nodes and 6 edges, each subsequent level incorporates an additional 2 edges and 1 node, culminating in graphs of 15 nodes and 24 edges, progressively elevating the difficulty of assigning colors to each node without any two adjacent nodes sharing the same color.

Meeting Scheduling Problem (MSP) MSP deals with allocating time slots for meetings such that all constraints, including participant availability and room capacity, are satisfied without overlaps. Given a set of n participants and their availability for m time slots, find a schedule that maximizes the number of participants who can attend the meeting. Formally, let $A = a_1, a_2, ..., a_n$ be the set of participants and $T = t_1, t_2, ..., t_m$ be the set of time slots. For each participant a_i, let S_i be a subset of T representing the times when a_i is available and m_i be a subset of meetings that are required to attend. The meeting scheduling problem can be formulated as finding a subset $S \subseteq T$ such that $|a_i \in A|S_i \cap S \neq \emptyset|$ is maximized. In other words, the aim is to find a scheduling subset S_i where the collective availability of participants intersects with S_i, ensuring maximum participation. This problem is crucial in organizational management for scheduling meetings involving multiple participants with varying availability. It ensures optimal utilization of time and resources and is used in corporate scheduling systems and collaborative software (Bofill et al., 2022). MSP outlines complexity across 10 levels, determined by an increasing number of meetings (2 to 11), participants (one more than the number of meetings, i.e., 3 to 12), and time slots (two more than the number of meetings, i.e., 4 to 13). Each level advances the problem's complexity by adding one meeting, consequently increasing the number of participants and time slots required for scheduling.

C Choices of Problems

In the benchmark, we exclude calculation-only (math intensive) tasks for each of the complexity classes, due to the overlap with already exist benchmarks and the known uncertainty of LLMs' math ability. For other reasoning, we provide detailed explanations and highlight them in bold.

C.1 Excluded P problems

Prime Number Determination Using algorithms like AKS primality test to determine if a given number is prime. Reason: Math-intensive.

Solving Linear Equations Finding solutions for a system of linear equations. Reason: Math-intensive.

Maximum Flow Problem Finding the maximum flow from a source node to a sink node in a flow network. A flow network is a directed graph $G = (V, E)$ where each edge $(u, v) \in E$ has a capacity $c(u, v)$ and flow $f(u, v)$, with a designated source s and sink t. The objective is to maximize the total flow from s to t under the constraints that the flow on an edge does not exceed its capacity and the incoming flow is equal to the outgoing flow for every vertex except s and t. Reason: **Most open source algorithms cannot follow the question and the prompt to provide outputs with mostly correct formats**.

C.2 Excluded NP-Complete problems

3-SAT Problem Deciding whether a given Boolean formula in conjunctive normal form with three literals per clause is satisfiable. Reason: Math-intensive.

C.3 Excluded NP-hard problems

Integer Linear Programming Finding the best integer solution for a set of linear equations and inequalities. Reason: Math-intensive.

D Benchmark Statistics: Consistency and Size

This section presents the statistical analysis of our benchmark's consistency and size, providing insights into its reliability and comprehensiveness.

D.1 Benchmark Consistency

Our benchmark demonstrates high consistency, as evidenced by preliminary assessments and ongoing developments to facilitate open-source consistency checks. Currently, the benchmark comprises three versions (V0, V1, and V2), which are utilized to assess the consistency of model performance. We tested seven open-source models, including Yi-34b, Qwen-14b, Mistral-7b, Phi-2, MPT-30b, Vicuna-13b, and Phi-1.5, along with GPT-4 Turbo. By calculating the variance across these versions, we generated 72 data points for model-specific observations. The summary statistics are as follows:

- Minimum variance: 0.0 (approximately half of the variances are near zero)
- Mean variance: 0.0072
- Maximum variance: 0.1114

These statistics indicate minimal variance across the benchmark versions, suggesting a high level of consistency with each update.

D.2 Benchmark Size

The benchmark is designed to be comprehensive, supported by a dynamic update mechanism that ensures continual renewal over six months to a year. All archived questions remain accessible for further evaluations, such as testing model consistency. Over time, each task within the benchmark will feature 1,200 questions, totaling 10,800 unique questions.

Furthermore, we justify our sample size selection through rigorous statistical analysis. While an infinite number of reasoning questions would ideally be required to precisely evaluate a Large Language Model's (LLM) capability on a specific task, practical constraints necessitate a finite sample. Based on sample size estimations (MacCallum et al., 1999; Morse, 1992) and margin of error calculations (Israel et al., 1992; Yamane, 1973), 100 questions per task are necessary to achieve a 95% confidence level with a 10% margin of error. For instance, in Appendix E, our statistical analysis at the $p < 0.05$ significance level confirms the robustness of the results.

E Model Performance across Task Complexity and Difficulty Levels

To evaluate across task complexity, specifically comparing the complexity among P, NP-Complete, and NP-Hard pairs, we initially pinned the data based on complexity levels. Subsequently, we applied the Wilcoxon test to each pair of complexity sets. Wilcoxon is a non-parametric statistical hypothesis test that allows us to compare two populations with matched samples. To evaluate problem difficulty, aiming to discern differences among problems within the complexity category, we pinned the data based on the specific problems and then used the Wilcoxon test to compare pairs of different problem sets.

E.1 Across Complexity Levels

Figure 6 shows the accuracy of each model across different complexity levels. The test results reveal statistical significance ($p < 0.05$) in the p-values between P and NP-Hard, as well as NP-Complete and NP-Hard. These findings indicate that our investigated LLMs performed significantly worse when confronted with NP-Hard problems compared to P and NP-Complete problems.

E.2 Across Models

Figure 7 presents the accuracy of each model across various problems associated with P, NP-Complete, and NP-Hard complexities. Regarding P complexity, notable differences emerged among the models. GPT 3.5 Turbo, GPT 4 Turbo, Yi-34b, and Qwen-14b models exhibited significantly superior performance on the SAS problem compared to the other two problems. GPT 3.5 Turbo, Yi-34b, and Vicuna-13b models demonstrated markedly better performance on the EDP problem compared to the SPP problem. Only the Vicuna-13b model displayed slightly better performance, although not significant, on the EDP problem compared to SAS across all investigated models.

E.3 Other observations

GPT 4 Turbo showcased very similar performance between the EDP and SPP problems, while Claude Instant 1.2 exhibited similar performance for all these three problems. Yi-34b, Qwen-14b, GPT 3.5 Turbo, and GPT 4 Turbo displayed remarkably high accuracy specifically for the SAS task. MPT-30b and Phi-1.5 showed very limited performance in identifying these three problems.

Regarding NP-Complete complexity, there are several observations to highlight. Still, neither MPT-30b nor Phi-1.5 could deliver any identification for problems in the NP-Complete complexity. In the case of GCP-D and TSP-D problems, the performance of these models varied significantly. Phi-2, Vicuna-13b and GPT 4 Turbo outperformed in the GCP-D problem compared to TSP-D, whereas Claude Instant 1.2, Claude 2, and PaLM 2 exhibited better performance in TSP-D over GCP-D. On the other hand, models like Mistral-7b, Yi-34b, Qwen-14b, and GPT 3.5 Turbo showcased relatively similar performance between these two tasks. For the KSP task, only GPT 4 Turbo demonstrated promising performance, while the remaining models faltered.

Considering NP-Hard complexity as the most intricate task set among the three (as evidenced in Figure 6), many of the examined models encountered challenges in identifying tasks within this complexity. For the GCP task, Mistral-7b, PaLM 2, GPT 3.5 Turbo, and GPT 4 Turbo exhibited some potential, while Vicuna-13b and Claude Instant 1.2 showed limited performance. For the TSP task, identification was observed only in Claude 2 and GPT 4 Turbo. Of all the investigated models, GPT

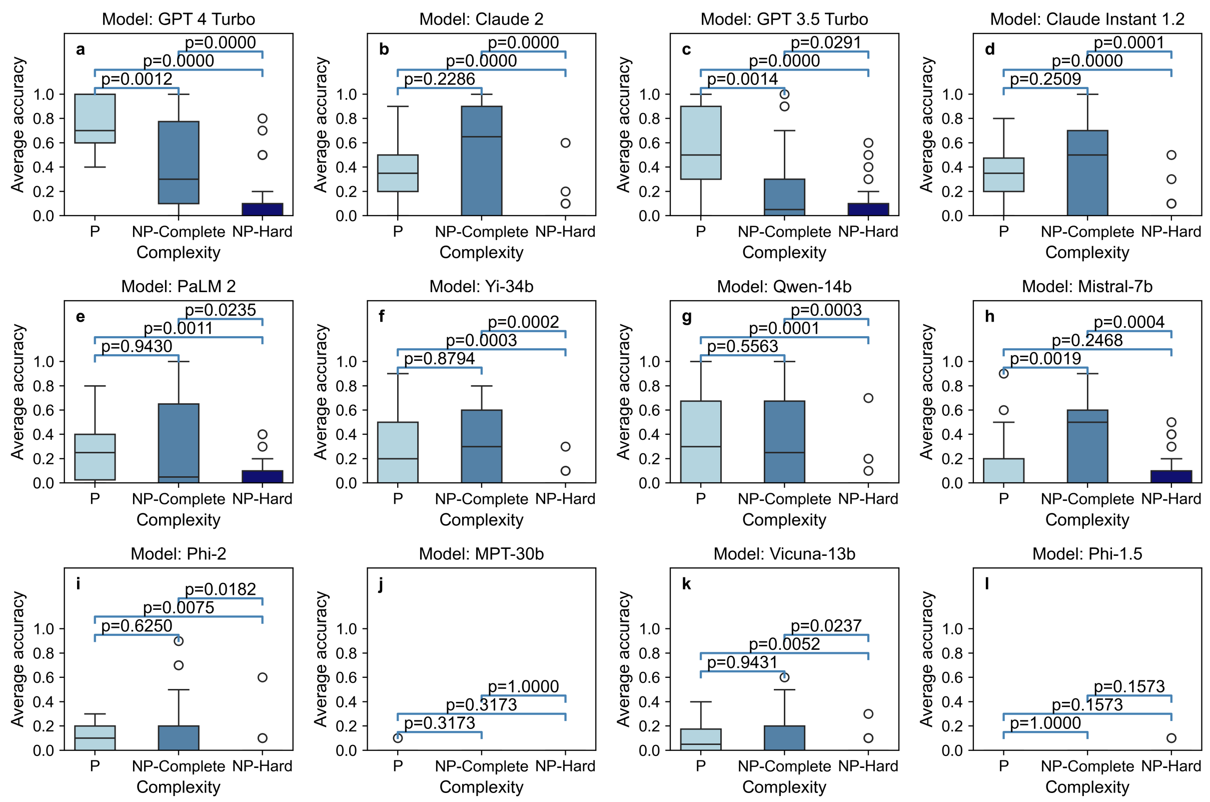

Figure 6: Models' performance on each complexity level. (a) GPT 4 Turbo. (b) Claude 2. (c) GPT 3.5 Turbo. (d) Claude Instant 1.2. (e) PaLM 2. (f) Yi-34b. (g) Qwen-14b. (h) Mistral-7b. (i) Phi-2. (j) MPT-30b. (k) Vicuna-13b. (l) Phi-1.5.

4 Turbo exhibited promise in identifying these three tasks within the NP-Hard complexity. However, the performance in GCP and TSP identification significantly surpassed that of the MSP task across these models. For the MSP task, only GPT 4 Turbo displayed some ability for identification, while with notably low accuracy.

F Model Generalization through In-context Learning

Given examples in the context, can LLMs genuinely learn and apply algorithmic skills presented in contextual examples as opposed to merely mimicking problem-solving processes (Wei et al., 2023; Min et al., 2022)? We differentiate between "learning" and "mimicking" by evaluating whether LLMs can generalize solutions to new problems of varying difficulty levels within the same task, after being exposed to examples. Our hypothesis is that if an LLM has truly learned the underlying algorithmic skill, it should be able to tackle problems across different difficulty levels within the same task. Conversely, if an LLM is merely mimicking, its performance may falter when faced with variations in problem difficulty.

F.1 Experiment: Comparative Analysis of Learnability by In-context Learning

A prevalent approach in current few-shot learning involves using examples that bear similarity to the test question. However, this raises a question about the extent to which the model is replicating the problem-solving process from the examples as opposed to genuinely acquiring reasoning skills. Consequently, it becomes pertinent to investigate whether the problem-solving abilities developed through example-based learning are generalizable.

To delve deeper into the models' in-context learning abilities, we utilize various few-shot in-context learning prompts to discern whether the model is "learning" from the few-shot examples or merely "mimicking" the behavior. In our benchmark, since we distinctly classify the difficulty level of each question, it allows for the use of questions from the same task but with varying difficulty levels as few-shot examples. The crux of this analysis lies in varying the difficulty levels of examples within the prompts. Since the fundamental algo-

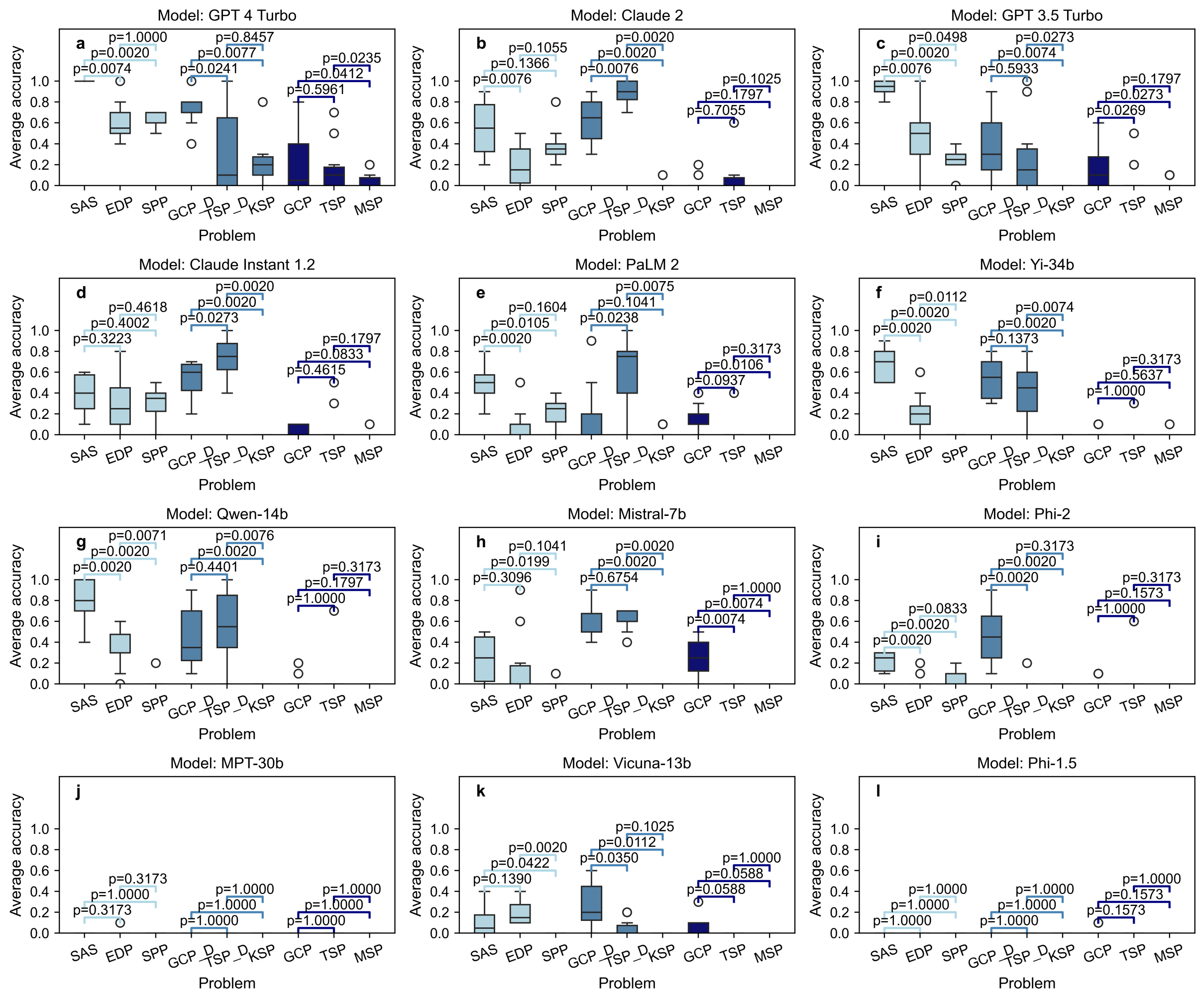

Figure 7: Models' performance on tasks across complexity levels. (a) GPT 4 Turbo. (b) Claude 2. (c) GPT 3.5 Turbo. (d) Claude Instant 1.2. (e) PaLM 2. (f) Yi-34b. (g) Qwen-14b. (h) Mistral-7b. (i) Phi-2. (j) MPT-30b. (k) Vicuna-13b. (l) Phi-1.5.

rithmic skill required to solve a question remains constant across varying difficulty levels under the same task, a model that truly learns this skill should show consistent performance irrespective of the example difficulty in the prompt. We propose the following hypotheses about the relationship between in-context learning ability and the difference of difficulty level between the given examples and the question being asked in context:

- Models possessing optimal generalization capabilities should demonstrate consistent performance improvement regardless of the difficulty level of the prompt examples in context. This assumption is based on the premise that models with robust learning abilities are capable of discerning and applying the intrinsic problem-solving skills learned in the examples. Given that questions within the same task fundamentally require similar skills, variations in difficulty are unlikely to significantly affect the model's performance.
- If a model exhibits the ability to generalize only from some types of examples but is unable to extend this learning to others, it reveals a deficiency in its capacity for generalization in terms of reasoning. This suggests that the model is not genuinely acquiring problem-solving skills from the examples but merely recognizing and applying patterns from examples that are of equal or greater complexity to the problem at hand.
- If a model is unable to generalize from either more difficult or easier examples and is restricted to examples of the same difficulty level, it strongly suggests that the model is merely replicating the process presented in the context rather than internalizing any fundamental

problem-solving techniques or pattern recognition embedded within the examples. This behavior indicates a profound deficiency in the model's ability to comprehend and understand the underlying principles. It points to an absence of transferable, logic-learning skills, reflecting a superficial form of learning that is limited to surface-level imitation rather than a deeper, conceptual grasp.

We categorize the few-shot prompts into three types:

- Few-shot prompts with examples of the same difficulty level: Here, the model is provided with five examples in the prompt, all of which are at the same difficulty level and distinct from the question being asked.
- Few-shot prompts with examples that are easier than the question: This set comprises five variations of prompts, each with examples that are 1, 2, 3, 4, and 5 levels easier than the question, respectively.
- Few-shot prompts with examples that are more challenging than the question: Similarly, we prepare five sets of prompts, each containing examples that are 1, 2, 3, 4, and 5 levels more difficult than the question, offering a gradient of increased challenge.

Through this diverse array of prompts, we aim to provide a nuanced understanding of the LLMs' ability to learn from examples, thereby offering valuable insights into their underlying learning capabilities.

F.2 Effects of Few-shot Examples' Difficulty on Reasoning Ability Enhancement

In the experiment above, we focused on the tasks of SAS and EDP to investigate the nature of the in-context learning capabilities of LLMs. This experiment empirically distinguishes between "learning" and "mimicking" as exhibited by LLMs during in-context learning scenarios. Our findings also revealed a clear dichotomy in the approach to learning and generalization from examples between closed-source and open-source models.

For closed-source models, including GPT 4 Turbo, Claude 2, GPT 3.5 Turbo, PaLM 2, and Claude Instant 1.2, the results were notably close to the ideal scenario. We observed minimal variation in performance across different levels of difficulty in the examples provided. This consistency suggests that these models are not merely mimicking the solutions but are indeed learning the algorithmic skills presented in the context of the examples.

In contrast, the performance of open-source models, particularly Yi-34b and Mistral-7b, exhibits a clear pattern where the models generally generalize well from examples that are more challenging than the given question, yet they struggle to do so from simpler examples. Other open-source models display less distinct patterns, but a notable trend is still evident: these models demonstrate some capacity to generalize from more challenging to simpler questions, but they are less successful in generalizing from simpler to more complex questions. An exception is observed with the Phi-1.5 model in EDP, where it appears to generalize better from easier examples than from harder examples at certain difficulty levels. However, broadly speaking, none of the open-source models consistently learn from both harder and easier examples. The difficulty level significantly influences the models' performance, suggesting a tendency for these models to mimic patterns rather than engage in genuine learning from the context.

This phenomenon underscores that the differentiation between powerful closed-source and open-source models lies not only in their raw reasoning ability but also significantly in their capacity to learn from in-context examples. This insight highlights the importance of considering both reasoning and learning abilities when evaluating the effectiveness and potential applications of LLMs.

G Research Outlook

Our research outlook includes future investigations that can extend and enrich our understanding of the reasoning abilities of LLMs.

Fine-grained Time Complexity under Polynomial (P) with Big $\mathcal{O}$ notation We will further the investigation of the P complexity class with fine-grained time complexity notation, the Big $\mathcal{O}$ notation. For example, the time complexity of SAS is $\mathcal{O}(\log n)$, while the time complexity of the Dijkstra algorithm, the solution to SPP, is $\mathcal{O}(V \log V + E)$ with Fibonacci heaps (Cormen et al., 2022). This approach will enable a detailed evaluation of models within the same complexity, proving a complement perspective to the current difficulty levels and enabling a possible cross-comparison among

Accuracy	**GPT 4 Turbo**	**Claude 2**	**GPT 3.5 Turbo**	**Claude Instant**	**PaLM 2**	**Yi-34b**	**Qwen-14b**	**Mistral-7b**	**Phi-2**	**MPT-30b**	**Vicuna-13b**	**Phi-1.5**
Prompts on SAS												
Zeroshot	**1.000**	0.445	**0.942**	0.442	0.416	0.620	**0.706**	0.149	0.191	0.000	**0.113**	0.000
Fewshot (-5)	0.978	0.685	0.920	0.735	0.603	0.065	0.103	0.043	0.095	0.165	0.085	**0.155**
Fewshot (-4)	**1.000**	0.662	0.902	0.667	0.516	0.093	0.189	0.129	0.149	0.156	0.084	0.118
Fewshot (-3)	0.982	0.694	0.831	**0.769**	0.496	0.143	0.114	0.153	0.116	0.131	0.067	0.084
Fewshot (-2)	**1.000**	**0.771**	0.910	0.710	0.617	0.102	0.070	0.094	0.073	0.117	0.037	0.087
Fewshot (-1)	0.987	0.770	0.896	0.589	0.607	0.048	0.126	0.085	0.107	0.157	0.087	0.057
Fewshot (0)	0.984	0.671	0.846	0.651	**0.660**	0.598	0.255	**0.413**	0.222	0.258	0.089	0.098
Fewshot (1)	0.991	0.580	0.878	0.696	0.455	0.593	0.455	0.386	**0.287**	0.233	0.055	0.109
Fewshot (2)	**1.000**	0.675	0.829	0.587	0.656	0.647	0.444	0.296	0.260	0.175	0.067	0.056
Fewshot (3)	0.993	0.736	0.800	0.598	0.489	**0.662**	0.427	0.318	0.275	0.144	0.093	0.098
Fewshot (4)	**1.000**	0.729	0.869	0.580	0.471	0.638	0.251	0.287	0.195	**0.269**	0.053	0.106
Fewshot (5)	**1.000**	0.671	0.844	0.602	0.607	0.167	0.387	0.356	0.196	0.202	0.055	0.064
Prompts on EDP												
Zeroshot	0.536	0.120	0.318	0.176	0.033	0.166	**0.269**	0.058	0.009	0.002	0.147	0.000
Fewshot (-5)	0.387	0.075	0.417	0.048	0.170	0.000	0.000	0.015	0.210	0.000	0.000	0.205
Fewshot (-4)	**0.556**	**0.209**	0.367	0.102	0.207	0.000	0.000	0.000	0.300	0.000	0.044	0.284
Fewshot (-3)	0.500	0.178	0.386	0.167	0.235	0.029	0.000	0.029	0.327	0.000	0.108	0.331
Fewshot (-2)	0.462	0.173	0.479	0.210	0.208	0.146	0.065	0.090	0.329	0.000	0.154	**0.335**
Fewshot (-1)	0.485	0.200	0.513	0.246	**0.289**	0.135	0.069	0.098	**0.348**	0.011	**0.248**	0.328
Fewshot (0)	0.518	**0.209**	**0.564**	0.253	0.238	**0.282**	0.227	**0.182**	0.320	**0.022**	0.164	0.293
Fewshot (1)	0.535	0.184	0.535	**0.355**	0.205	0.089	0.266	0.089	0.115	0.013	0.160	0.115
Fewshot (2)	0.545	**0.209**	0.544	0.238	0.196	0.195	0.266	0.042	0.098	0.015	0.093	0.087
Fewshot (3)	0.536	0.189	0.449	0.315	0.182	0.127	0.140	0.067	0.060	0.007	0.191	0.051
Fewshot (4)	0.538	0.209	0.507	0.305	0.200	0.247	0.186	0.095	0.009	0.000	0.129	0.000
Fewshot (5)	0.531	0.205	0.449	0.244	0.167	0.271	0.146	0.055	0.015	0.009	0.202	0.000

Table 1: Weighted accuracy of Zero-shot and Few-shot on SAS and EDP. The best performance for each column is highlighted with bold font (respectively for SAS and EDP).

different tasks' difficulty levels.

Self-correction for Reasoning Another promising avenue is the enhancement of LLM reasoning abilities. A key strategy here is the implementation of iterative self-correction mechanisms. Pioneered by self-correction experiments in (Huang et al., 2023; Stechly et al., 2023), allowing LLMs to go through multiple rounds (e.g., ranging from 1 to 10) of self-correction, we can observe how the refinement process affects the accuracy and sophistication of their responses. This iterative process mimics human problem-solving, where multiple drafts and revisions lead to improved outcomes.

Multi-agent Systems for Reasoning Moreover, exploring a multi-agent system (Wu et al., 2023; Chan et al., 2023; Ge et al., 2023a,b) approach could significantly advance LLMs' reasoning abilities. In such a system, different LLM agents, each potentially specialized in certain types of reasoning or knowledge areas, collaborate to solve complex problems. This collaborative approach could mimic a team of experts, each contributing their expertise, leading to more comprehensive and nuanced solutions. It also opens the door to understanding how LLMs can interact and augment each other's capabilities, which is crucial for their application in real-world, multi-faceted problem-solving scenarios.

These future research directions hold the potential not only to deepen our understanding of the current capabilities and limitations of LLMs but also to drive forward the development of more sophisticated and reliable AI systems. By focusing on robustness testing and enhancing reasoning abilities through innovative methods like iterative self-correction and multi-agent systems, we can make significant strides towards realizing the full potential of LLMs in complex decision-making and problem-solving tasks.

Failure Rate	**GPT 4 Turbo**	**Claude 2**	**GPT 3.5 Turbo**	**Claude Instant**	**PaLM 2**	**Yi-34b**	**Qwen-14b**	**Mistral-7b**	**Phi-2**	**MPT-30b**	**Vicuna-13b**	**Phi-1.5**
Prompts on SAS												
Zeroshot	0.000	0.260	0.000	0.400	0.110	0.330	0.200	0.070	0.150	1.000	0.480	1.000
Fewshot (-5)	0.000	0.060	0.020	0.000	0.000	0.940	0.900	0.480	0.920	0.160	0.640	0.860
Fewshot (-4)	0.000	0.033	0.000	0.000	0.017	0.917	0.817	0.617	0.867	0.117	0.633	0.900
Fewshot (-3)	0.000	0.057	0.057	0.014	0.000	0.871	0.886	0.471	0.886	0.043	0.600	0.914
Fewshot (-2)	0.000	0.075	0.025	0.000	0.013	0.888	0.913	0.538	0.900	0.063	0.613	0.888
Fewshot (-1)	0.000	0.044	0.022	0.000	0.011	0.911	0.856	0.622	0.878	0.078	0.589	0.933
Fewshot (0)	0.000	0.060	0.020	0.000	0.020	0.300	0.640	0.380	0.670	0.040	0.540	0.880
Fewshot (1)	0.000	0.040	0.020	0.010	0.010	0.290	0.500	0.420	0.700	0.060	0.520	0.830
Fewshot (2)	0.000	0.040	0.040	0.000	0.010	0.290	0.510	0.440	0.710	0.030	0.640	0.910
Fewshot (3)	0.000	0.020	0.050	0.010	0.030	0.280	0.450	0.460	0.670	0.030	0.650	0.830
Fewshot (4)	0.000	0.050	0.030	0.000	0.040	0.280	0.570	0.530	0.700	0.080	0.630	0.850
Fewshot (5)	0.000	0.050	0.040	0.000	0.020	0.680	0.520	0.440	0.740	0.050	0.650	0.900
Prompts on EDP												
Zeroshot	0.000	0.000	0.000	0.000	0.440	0.000	0.000	0.040	0.000	0.960	0.160	0.950
Fewshot (-5)	0.000	0.000	0.000	0.140	0.160	0.000	0.320	0.140	0.540	0.880	0.460	0.640
Fewshot (-4)	0.000	0.000	0.000	0.100	0.150	0.000	0.233	0.050	0.400	0.817	0.383	0.483
Fewshot (-3)	0.000	0.043	0.000	0.057	0.100	0.000	0.100	0.029	0.300	0.871	0.329	0.400
Fewshot (-2)	0.000	0.038	0.000	0.025	0.075	0.000	0.038	0.025	0.263	0.700	0.238	0.350
Fewshot (-1)	0.000	0.011	0.000	0.056	0.033	0.000	0.022	0.000	0.167	0.667	0.200	0.256
Fewshot (0)	0.000	0.020	0.000	0.060	0.000	0.000	0.040	0.000	0.130	0.730	0.190	0.200
Fewshot (1)	0.000	0.000	0.000	0.070	0.000	0.000	0.010	0.000	0.100	0.800	0.190	0.210
Fewshot (2)	0.000	0.040	0.000	0.050	0.000	0.000	0.020	0.000	0.000	0.750	0.090	0.100
Fewshot (3)	0.000	0.020	0.000	0.060	0.040	0.000	0.000	0.000	0.010	0.710	0.040	0.100
Fewshot (4)	0.000	0.030	0.000	0.030	0.000	0.000	0.000	0.000	0.000	0.810	0.000	0.000
Fewshot (5)	0.000	0.050	0.000	0.030	0.000	0.000	0.000	0.000	0.000	0.820	0.030	0.000

Table 2: Weighted failure rate of Zero-shot and Few-shot on SAS and EDP.