StructTest: Benchmarking LLMs' Reasoning through Compositional Structured Outputs

Anonymous ACL submission

Abstract

The rapid development of large language models (LLMs) necessitates robust, unbiased, and scalable methods for evaluating their capabilities. However, human annotations are expensive to scale, model-based evaluations are prone to biases in answer style, while target-answer-based benchmarks are vulnerable to data contamination and cheating. To address these limitations, we propose StructTest, a novel benchmark that evaluates LLMs on their ability to produce compositionally specified structured outputs as an unbiased, cheap-to-run and difficult-to-cheat measure. The evaluation is done deterministically by a rule-based evaluator, which can be easily extended to new tasks. By testing structured outputs across diverse task domains — including Summarization, Code, HTML and Math — we demonstrate that StructTest serves as a good proxy for general reasoning abilities, as producing structured outputs often requires internal logical reasoning. We believe that StructTest offers a critical, complementary approach to objective and robust model evaluation.

1 Introduction

In the two years since ChatGPT's launch, the NLP community has seen a rapid acceleration in the release of large language models (LLMs). In 2024 alone, leading tech companies such as OpenAI, Anthropic, Google, and xAI all have introduced new versions of their proprietary models. The open-source community has been even more prolific, releasing a significantly higher number of models. To demonstrate improvements, many of these models, including Claude-3.5-Sonnet and Llama 3.1, have used benchmarks such as MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021a). However, as we will discuss in Section 2, existing benchmarks have notable limitations: 1) human annotations are expensive to acquire, maintain, and scale. 2) model-based evaluation suffers severely from model bias, and 3) target-answer-based datasets are prone to data contamination. Thus, there is a pressing need for an evaluation benchmark that is: 1) easy and cheap to evaluate, 2) unbiased, and 3) robust to data contamination.

To address these challenges, we propose **StructTest**, a benchmark designed to assess instruction-following capabilities through structured outputs. In StructTest, models are tasked with generating outputs in a specified structure, which are then verified for structural accuracy and, when applicable, for the correctness of parsed outputs. The evaluation in StructTest is done programmatically, which makes it unbiased, fast, and cheap to run. StructTest is not coupled with underlying task data, which allows us to easily sample a new StructTest set and extend to new tasks, making it robust to data contamination (more details in Section 3.5). StructTest instruction is compositional by design, which allows for adjustable difficulty levels, enabling our benchmark to stand the test of time and benchmark future generations of LLMs.

The benchmark covers multiple task domains, including summarization, code, HTML, and math. Evaluations conducted on 14 popular LLMs reveal that StructTest exhibits a correlation over 92% with both the human-annotated ChatBot Arena benchmark and the widely used MMLU dataset, verifying StructTest as a good proxy for general reasoning ability (see Section 4.2 for more details).

2 Literature Review

Evaluation of LLMs has become a critical area of research, particularly as these models are increasingly applied to diverse tasks requiring structured reasoning. Existing evaluation methodologies broadly fall into three categories: human-based, model-based, and target-answer-based evaluations. While each offers unique insights, they also suffer from notable limitations.

2.1 Human-Based Evaluation Benchmarks

A prominent example of human-based evaluation benchmark is **Chatbot Arena** (Chiang et al., 2024), which relies on human voting to determine the model ranking ELO score, offering reliable assessment but with significant limitations: resource-intensive evaluations requiring massive human annotations, limited scalability to only a few models, and sustainability challenges in keeping the community engaged for latest models.

2.2 Model-Based Evaluation Benchmarks

Model-based evaluation frameworks utilize LLM-as-a-judge to assess the capabilities of other models. Notable examples include **MT-Bench** (Zheng et al., 2023), **AlpacaEval** (Dubois et al., 2024), **Arena-Hard-Auto** (Li et al., 2024), **Fofo**(Xia et al., 2024).

While being flexible in evaluation tasks, model-based evaluation benchmarks are known to be biased: 1) **Cheating by Null-Models**: As highlighted by Zheng et al. (2024), a simple null-model with constant response can achieve a high ranking in these benchmarks, posing a big concern on the reliability of the evaluations. 2) **Length Bias**: Dubois et al. (2024) demonstrate that length-controlled outputs in model-generated benchmarks such as AlpacaEval may introduce a bias favoring responses of certain lengths, often confounding evaluations of content quality with irrelevant stylistic factors.

2.3 Target-Answer-Based Evaluation Benchmarks

Target-Answer-Based evaluations assess reasoning capabilities without relying on another LLM for comparison. Conventionally, most of LLMs benchmarks fall into this category, including ARC (Clark et al., 2018), GSM8K (Cobbe et al., 2021a), BIG-Bench (Zhong et al., 2024), AGIEval (Zhong et al., 2024) and **MMLU** (Hendrycks et al., 2020). In MMLU, the authors evaluate the LLMs reasoning capability in a curated datasets for different competitive exams. While being unbiased, target-answer-based benchmarks have a primary limitation: the issue of training **data contamination**. The pervasive use of large-scale, internet-derived datasets in pre-training LLMs often overlaps significantly with benchmark datasets, thereby inflating performance metrics and undermining the validity of these evaluations (Ravaut et al., 2024a).

To address the limitations of existing evaluation benchmarks, we introduce StructTest, which evaluates structured generation as a proxy for general reasoning. While prior works have explored how format instructions influence task performance (He et al., 2024; Do et al., 2024), StructTest goes beyond simple formatting by incorporating compositional structured outputs. It is the first benchmark specifically designed to assess the general instruction-following capabilities of LLMs through structured outputs.

3 StructTest Benchmark

In this section, we describe the tasks from which the StructTest benchmark is built upon: summarization, coding, HTML generation and mathematical reasoning.

3.1 Summarization

As a first task within StructTest, we consider summarization, a well-studied playground for LLMs. The vast majority of existing studies focus on the *content* of output summaries, evaluating properties such as coherence (Chang et al., 2023), faithfulness to the source (Laban et al., 2023), coverage of diverse information (Huang et al., 2023), and positional bias for context utilization (Ravaut et al., 2024b) and hallucination (Wan et al., 2024a). With the rapid progress in LLMs, meeting complex user requirements for summaries is important. In InstruSum, Liu et al. (2023) benchmark LLMs on such summary content instructions. However, an equally important user requirement is style or format of the summary, which has been relatively underexplored. To remedy, we design three format-following tasks in summarization:

- **Length** Controlling output length is a widely-researched property in summarization (Liu et al., 2018, 2022). Users looking for more granular details will prompt the system to output longer summaries.

To measure length-following ability, we verify whether the LLM's output $\boldsymbol{y}$ contains the required number of sentences N, which is sampled uniformly from a fixed interval across data points. Formally:

$$\text{Score} = \begin{cases} 1, & \text{if } \operatorname{len}(\boldsymbol{y}) = N, \\ 0, & \text{otherwise.} \end{cases}$$

- **Bullet points** Bullet points are a natural method to summarize and have yielded several of the most widely used datasets in summarization research (Hermann et al., 2015; Mukherjee et al., 2022). This format is appealing to users who wish to see a clear separation of ideas in the output summary. We prompt the LLM to summarize through a list of either unnumbered bullet (or other symbol) points, or numbered points, with a varying number of points (again, sampled uniformly from a fixed interval).

For unnumbered points, we check whether the output contains the specified symbol S in the correct number of times N:

$$\text{Score} = \begin{cases} 1, & \text{if count}(S \in \boldsymbol{y}) = N, \\ 0, & \text{otherwise.} \end{cases}$$

For numbered points, we verify that output lines $(\boldsymbol{y}_i, .., \boldsymbol{y}_M)$ are of the appropriate count and start with the correct sequence of numbers:

$$\text{Score} = \begin{cases} 1, & \text{if } (M = N) \wedge (\forall i \in [1, N], \boldsymbol{y}_{i,0} = \text{str}(i)) \\ 0, & \text{otherwise.} \end{cases}$$

- **Question** Yet another approach to summarization consists in answering key questions about the source, most notably the 5 Wh-questions of (what? why? who? when? where?). Question-answering is a popular paradigm in summarization evaluation (Deutsch et al., 2021; Scialom et al., 2021; Fabbri et al., 2021), as it naturally enables to review that key facts from the source are covered. To induce format following, we prompt the LLM to structure its summary such that it is composed of the list of 5 Wh-questions, each followed by its corresponding answer. This process is akin to query-focused summarization (Vig et al., 2022), where the Wh-questions form the query.

To evaluate Wh-questions summary formatting, we check that lines of output summary start with the Wh-questions. We also enforce that all questions are present, in any order. Formally, following the previous notation and noting $\boldsymbol{Q}$ the set of Wh-questions:

$$\text{Score} = \begin{cases} 1, & \text{if } (\boldsymbol{Q} \subset \boldsymbol{y}) \wedge (\forall i \in [1, N], \boldsymbol{y}_{i,0} \in \boldsymbol{Q}) \\ 0, & \text{otherwise.} \end{cases}$$

Creating more complex tasks StructTest instructions following one of the aforementioned summarization formats are referred to as **Easy Summarization**. To build a more challenging benchmark, we combine different formats. We use two types of combinations: first, we specify the number of bullet points or numbered points and the desired length (in sentences) of each point; second, we ask the LLM to nest bullet points within existing points, where nested points start with the tab symbol marking indentation. These instructions combining two summarization formats are referred to as **Hard Summarization**. In this latter case, evaluation metrics defined above are also combined together and the LLM needs to verify each property. We provide examples for each task in Appendix B.

3.2 Code

Programming languages, with their highly structured and rule-based nature, offer an ideal framework for evaluating the format-following capabilities of LLMs. In addition, compilers and interpreters efficiently validate correctness, offering clear binary feedback on whether the generated code meets both syntactic and functional expectations. This makes coding tasks a practical and scalable benchmark for assessing LLM performance in structured environments.

Given the widespread use of programming in daily workflows, and the significant success and adoption of Code-LLMs in real-world systems (Jimenez et al., 2024; Xie et al., 2024), understanding complex instructions in a language code-interleaved environment has become increasingly important. To closely measure the capability of LLMs in application-like scenarios, we have developed the following tasks:

- **Add 'print' statements** One class of general editing need is asking for revision of a code snippet. We propose a simple edit task by asking the LLM to add a 'print' statement after **each time a new variable is initialized**. We create two sets based on the difficulty level. The **Easy** set contains code with the number of lines ranging from 3 to 30, while those in the **Hard** set have 50-200 lines.

Since the instruction is fixed, we can easily manipulate the expected code snippet through automatic programs. Specifically, we first use the *ast*[1] package to extract the variables by parsing the abstract syntax grammar. Afterwards, the expected target code snippet can be obtained by inserting the print statements through fixed templates. The evaluation metric is **exact match** by comparing the predicted code snippet with the synthesized one.

- **Replace variables** Another edit-based task is *replacing variables*. For the data construction, we

[1] https://docs.python.org/3/library/ast.html

first use *ast* to extract the variables contained in a code snippet and randomly generate some meaningless strings for each of them as the target variable names to be replaced with. The mapping is shown as part of the instruction, which asks the LLM to replace all the usage of the mentioned source variables with the corresponding target ones.

Similarly, the expected code snippet can be obtained by string replacements according to the mapping. The prediction is evaluated by checking its consistency with the synthesized expected program. We use **exact match** as the evaluation metric.

- **Test case input generation** Serving as a fundamental part of software engineering, writing high quality unit tests (i.e., sample input-output pairs) plays an important role in verifying the program correctness. Considering that predicting the unit test outputs is still quite challenging for current LLMs (Li et al., 2022; Jain et al., 2024; Jiao et al., 2024), we choose to formulate an easier task by asking the LLMs to generate **5** different groups of test case inputs for a given programming question and the corresponding program.

We evaluate the validity by executing the program on the predicted test case inputs, and if no runtime error is raised for all inputs, the generation is deemed correct. We use the **averaged pass rate** over all problems as the evaluation metric.

- **Simulate program execution** Simulating the process of program execution presents several challenges, such as understanding and following each action in the program, tracking runtime states of the variables, and associating them together for the outcome successfully. These tasks are strongly correlated with reasoning and agent-based operations, making program simulation a useful proxy for evaluating the ability to follow compositional instructions and perform logical reasoning. We thus prompt the LLM to simulate step-by-step execution of a given program with specific inputs, and derive the expected output. The task is also divided into two difficulty levels – **Easy** and **Hard**, according to the length of the code snippet for simulation.

For Easy level, we introduce multiple test cases in the original dataset for each question to ensure robust evaluation. If all predicted outputs **exactly match** the ground-truth ones, the generation for the question is considered as correct. For Hard level, we only use one simple test case to evaluate the generation, since (1) the difficulty of the code snippet itself in Hard level is challenging enough, and (2) it is hard to unify the test case scaling. For example, some test cases can contain millions of input numbers in a line. The final metric is also the averaged **exact match** over all questions.

3.3 HTML Generation

Application of LLMs in generating websites has been regarded as a useful task that can ease the workload of web designers and developers and help in democratizing web development for non-technical users (Calò and De Russis, 2023). In such applications, adherence to the user specified HTML structure is a crucial part. Tang et al. (2023) show that LLMs may struggle to generate structured HTMLs. However, their study is limited to simple structures and the evaluation is content-based which requires human evaluation.

In contrast, we formulate this task as to generate a specific number of standard HTML tags ("html", "head", "title", "div", "body", "h1", "h2", "p","footer") as instructed with the following structural constraints: "title" should be nested inside "head", "div" and "footer" are nested inside "body", and the rest of the tags are nested inside "div".[2] An example prompt with our prompt template is: *"Generate only an html code that has 1 html tag. Inside the html tag, generate 1 head tag and 1 body tag. Inside of each head tag, generate 1 title tag and inside of each body tag, generate 2 div tags and 1 footer tag. Inside of each div tag, generate 1 h1 tag, 1 h2 tag and 1 p tag. Your generated html code:"*, and the expected generation should be an html code block as follows:

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <div>
            <h1></h1>
            <h2></h2>
            <p></p>
        </div>
        <div>
            <h1></h1>
            <h2></h2>
            <p></p>
        </div>
        <footer></footer>
    </body>
</html>
```

The counts of each tag to be generated are sampled uniformly from a fixed interval. Based on the range of the interval, we create two sets, **Easy** where the interval range is 2-5, and **Hard** where the range is 2-12.

[2]The number of each tag except "html" varies across examples. For "html", it is fixed to 1.

We consider a generation to be successful if the count of the tags is equal to the ones provided in the prompt taking into account their nested structure and all the tags are properly formatted, i.e., an opened HTML tag has to be closed.

3.4 Math Reasoning

Math reasoning is a prevalent task in LLM evaluations, with benchmarks like GSM8K and MATH (Gao et al., 2024; Cobbe et al., 2021b; Hendrycks et al., 2021). However, the impact of varying format templates on these tasks is often overlooked, potentially causing inconsistencies as many efforts may not use impartial templates (Yu et al., 2023; Shao et al., 2024; Wei et al., 2022; Toshniwal et al., 2024). The variability in solutions, which could be numbers, fractions, or LaTeX expressions, means extraction heuristics may differ across studies, leading to biased comparisons favoring models optimized for specific frameworks. For example, MetaMathQA (Yu et al., 2023) produced a dataset where answers follow specific phrases which their evaluation procedure uses to extract answers, disadvantaging models that don't use these phrases. Meanwhile, a reliable model should not only provide the correct answer but also present a chain of thoughts in a pre-defined format consistently (Wei et al., 2022). Being able to extract reasoning steps reliably can be beneficial, such as to produce thought chains for process supervision (Lightman et al., 2023). Thus, we focus on structuring our math evaluations in two aspects: final answer parsing and chain of thought bullet point formatting.

- **Final answer parsing** We crafted 7 styles for final answer production and created prompts instructing models to adhere to these styles. We then wrote Python rules to determine a model response's consistency to a respective style. To evaluate models, standard benchmarks like GSM8K are used, with each question assigned a random style for evaluation. This format consistency accuracy can then be derived and combined with the math accuracy score to paint a clearer and fairer comparison across LLMs. Final answer parsing is considered **Easy** in our setup.

- **Chain of thought bullet points.** Solutions typically involve multiple reasoning steps, and we designed 5 different presentation styles. Some are Markdown styles, like "**Step 1** ...", or JSON styles. We also set a range for the number of steps, requiring models to adjust step granularity. That is, the models should break down a step into smaller steps if the solution is simple to satisfy the constraint. If the solution is complex, however, they should instead merge multiple steps into longer ones. Pairing each bullet point style with a unique final answer style resulted in 20 formats, categorized as **Hard**. We hypothesize that these styles may be intuitive to some LLM models but unfamiliar to others, potentially causing significant discrepancies in performance, as shown later in Section 4.1.4. While the number of styles could be extended infinitely, we rely on manually crafted styles to ensure accuracy and consistency.

3.5 Robustness to Contamination and Benchmark Scalability

A key challenge in benchmarking LLMs is the risk of data contamination, where models are exposed to test data during training. StructTest aims to mitigate this issue in two ways. First, the tasks in StructTest are crafted in a way that makes it highly unlikely for existing models to have encountered them during training. By focusing on carefully constructed structured output tasks, StructTest minimizes the possibility of data contamination. Second, the nature of the tasks allows us to periodically update the benchmark with respect to new StructTest samples, new task underlying data, new task domains and their complexity levels. To safeguard from future generations of LLMs, we aim to maintain a confidential, held-out test set that is periodically updated, ensuring that model performance accurately reflects generalization capabilities rather than memorization of pre-exposed data.

StructTest is highly scalable, allowing easy extension to new tasks by simply writing new prompts and rule-based evaluation. It enables cost-efficient evaluation for new models, with minimal overhead beyond inference costs. This design ensures flexibility and adaptability for diverse needs.

4 Evaluation Results

4.1 Results Overview

We run StructTest benchmark against a representative list of open-source and closed-source models[3]. Table 1 summarizes evaluation results across all domains of StructTest for all LLMs. For all open-source models, we use their instruction-tuned version instead of pre-trained version. It

[3]See Appendix A for detailed model versions for close-source models

LLM	Average			Summarization		Code		HTML		Math	
	All	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Phi-3-mini-128k	19.30	31.97	6.63	57.42	11.67	49.79	10.59	0.00	0.00	25.47	4.25
Qwen-2-7B	17.94	28.56	7.33	48.77	12.03	49.79	12.88	0.00	0.00	15.69	4.40
Mistral-7B	13.94	22.32	5.57	50.54	14.72	32.92	6.35	3.00	0.00	2.81	1.21
Llama-3.1-8B	33.73	46.85	20.61	95.46	35.58	51.36	16.06	9.33	0.00	30.86	30.78
Mistal-nemo	26.06	41.86	10.27	69.35	17.89	60.62	16.59	5.00	0.00	32.45	6.60
Mixtral-8x7B	16.85	27.73	5.98	59.10	13.36	34.37	5.49	3.33	0.00	14.10	5.08
Llama-3.1-70B	**65.99**	**86.36**	**45.63**	**98.48**	**51.36**	**79.06**	**27.55**	**98.67**	**42.67**	**69.22**	**60.96**
GPT-3.5-turbo	38.43	62.05	14.81	86.77	22.11	74.48	19.38	47.67	7.00	39.27	10.77
GPT-4o-mini	57.57	74.09	41.05	**98.79**	64.81	82.40	25.67	38.00	6.33	77.18	67.40
GPT-4o	**72.46**	89.47	**55.44**	96.19	**66.08**	**86.36**	29.34	99.00	57.67	76.35	68.69
Gemini-1.5-pro	61.51	81.44	41.58	84.58	23.58	82.19	**38.01**	81.67	31.33	77.33	**73.39**
Claude-3-haiku	35.95	53.31	18.59	72.19	20.44	66.25	22.18	41.00	10.33	33.81	21.38
Claude-3-opus	68.81	89.14	48.47	91.21	46.14	85.00	36.04	**100.00**	56.67	80.36	55.04
Claude-3.5-sonnet	69.26	**91.55**	46.97	96.33	44.33	84.79	29.70	**100.00**	**58.67**	**85.06**	55.19

Table 1: Overview of Evaluation Results on StructTest. Within each family of models, best numbers are in bold, and second-best are underlined.

LLM	Easy				Hard		
	Length	Bullet points	Numbered points	Wh-questions	Bullets + length	Numbers + length	Indented points
Phi-3-mini-128k	35.17	29.25	90.58	74.67	8.50	26.50	0.00
Qwen-2-7B	27.75	67.33	**100.00**	0.00	20.50	15.58	0.00
Mistral-7B	22.58	63.67	99.50	16.42	19.33	24.83	0.00
Llama-3.1-8B	90.17	95.00	**100.00**	96.67	47.58	55.00	4.17
Mistral-nemo	50.25	91.00	99.17	37.00	25.42	27.75	0.50
Mixtral-8x7B	67.66	41.16	90.83	36.75	8.33	31.25	0.50
Llama-3.1-70B	**94.08**	**99.83**	**100.00**	**100.00**	**63.50**	**63.83**	**26.75**
GPT-3.5-turbo	51.33	99.42	**100.00**	96.33	26.67	33.83	5.83
GPT-4o-mini	**96.92**	**100.00**	99.92	98.33	**75.83**	**76.17**	42.42
GPT-4o	84.75	**100.00**	**100.00**	**100.00**	66.75	71.67	**59.83**
Gemini-1.5-pro	66.50	99.42	99.50	72.92	41.00	23.08	6.67
Claude-3-haiku	67.25	99.33	99.75	22.42	29.25	32.08	0.00
Claude-3-opus	65.58	99.67	99.58	**100.00**	54.08	56.33	28.00
Claude-3.5-sonnet	85.58	99.83	99.92	**100.00**	66.50	66.17	0.33

Table 2: Performance comparison across LLMs on summarization-based tasks.

is worth noting that the best scoring LLM, GPT-4o, only achieves 72.46% on StructTest-All and 55.44% accuracy on StructTest-Hard, demonstrating that StructTest is a highly challenging benchmark. In addition, Claude-3.5-sonnet is a close runner-up and close-source models generally outperform open-source models.

4.1.1 Summarization Results

Among open-source models, the Llama-3.1 series stand much above the rest, as their performance is comparable to GPT-4 series on the Easy subset (98.48 for Llama-3.1-70B; 98.79 for GPT-4o-mini). Closed-source LLMs on average perform better than open-source ones, especially on the Hard subset. Indeed, open LLMs loose 70% in accuracy on Hard setups compared to Easy ones, compared to a 55% relative loss for closed-source LLMs. In both cases, such a plunge in performance highlights how challenging it is for even the best LLMs to follow more elaborate formatting instructions.

When breaking down performance across formatting tasks shown in Table 2, we notice that generating numbered points is easier for LLMs than bullet points, probably because generated numbers help the LLM stop at the correct length. Although all LLMs seemingly master producing numbered points, adding a constraint on the length of each point proves much harder: performance is divided by 4 for many open-source LLMs. Indenting points proves to be the hardest task. On this task, 7 LLMs out of the 14 evaluated stay at null or near null accuracy, including even Claude-3.5-sonnet.

A further analysis with GPT-4o in Figure 1 shows error rate for binned values of the Hard formatting condition of controlling the length of each bullet point. Length control error rate jumps beyond 20 total sentences, or 4 sentences per point. This finding proves that longer outputs are hard to structure and format for LLMs.

4.1.2 Code Results

From the model aspect, we find that Llama-3.1-70B achieves the best performance among the open-source models, due to its larger size. For the closed-source models, Claude-3.5-sonnet and Claude-3-opus are the two performing better.

From the perspective of Code tasks shown in Table 3, we find the Hard level problems demonstrate significantly more complexity, as longer code snippet will increase the difficulty in understanding. Besides, tasks requiring more comprehension and memorization present higher difficulty. For example, on the Easy level of *Add Print Statements* and *Replace Variables*, even the open-source small models, e.g., Llama-3.1-8B, can achieve strong performance. Most closed-source models can also demonstrate good performance on Hard level *Re-*

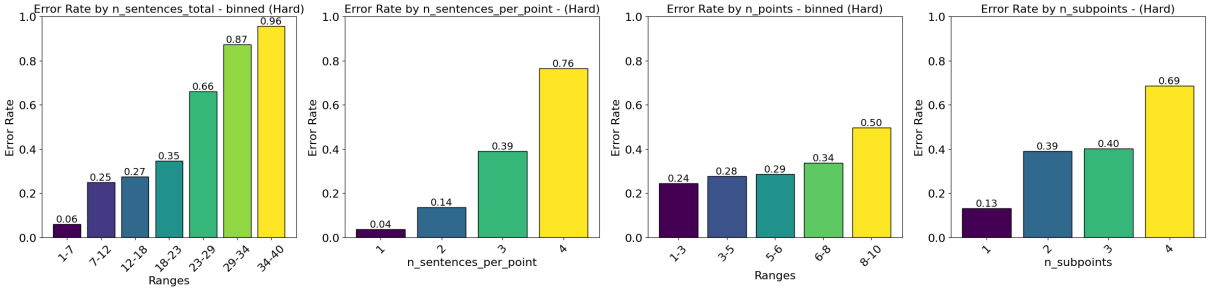

Figure 1: Error rate of GPT-4o across various features of the Summarization Bullet Points Length task.

LLM	Add Print		Replace Vars		Input Gen		Simulate Exec	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Phi-3-mini-128k	66.25	0.00	85.00	42.37	9.17	0.00	19.58	0.00
Qwen-2-7B	60.42	1.01	79.17	41.46	20.42	0.00	39.17	9.05
Mistral-7B	47.92	0.50	29.17	21.87	37.92	0.00	16.67	3.02
Llama-3.1-8B	78.33	3.02	82.92	49.66	6.25	**2.51**	39.58	9.05
Mistral-nemo	72.50	1.01	81.67	50.80	42.50	0.00	45.83	14.57
Mixtral-8x7B	41.67	1.51	10.83	8.88	45.42	1.01	39.58	10.55
Llama-3.1-70B	**95.00**	**22.61**	**88.75**	**64.46**	**58.33**	1.01	**74.17**	**22.11**
GPT-3.5-turbo	76.25	0.00	90.42	57.40	72.92	1.51	58.33	18.59
GPT-4o-mini	90.00	10.55	91.25	66.51	66.25	3.02	82.08	22.61
GPT-4o	85.00	9.55	86.67	70.62	**79.58**	4.52	**94.17**	32.66
Gemini-1.5-pro	94.17	34.17	83.33	70.62	65.83	4.02	85.42	**43.22**
Claude-3-haiku	75.42	5.03	86.67	60.59	40.00	5.53	62.92	17.59
Claude-3-opus	**96.25**	**40.20**	**91.67**	**78.82**	69.58	2.01	82.50	23.12
Claude-3.5-sonnet	90.00	9.55	91.25	78.59	70.42	**6.03**	87.50	24.62

Table 3: Performance comparison across LLMs on code-related tasks.

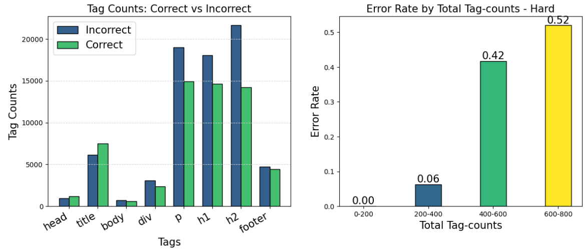

Figure 2: Error rate of GPT-4o by total tag counts (binned) and Tag-counts for correct vs. incorrect HTML code generations (hard task).

place Variables. Moreover, nearly all models perform really poorly on *Test Case Inputs Gen* in Hard level. One reason is that the problems in Hard level mainly use standard input-output stream. As a result, incorrect spaces or newline symbols will also lead to input errors.

4.1.3 HTML Results

In Table 1, we notice that, in general, open-source models perform significantly worse than closed-source models in both easy and hard HTML generation tasks and also model accuracies are significantly better in the easy task compared to the hard task. The clear winner in open-source model category is Llama-3.1-70B, and for closed-source models it is Claude-3.5-sonnet. It is to be noted that Claude-3.5-sonnet is also the overall winner and has the best MMLU score among all the models we experimented with (Table 4). In general we find that models with higher number of parameters are significantly better than the models with lower number of parameters, e.g. Llama-3.1-70B vs Llama-3.1-8B.

We further provide two types of analyses based on the performance of ChatGPT-4o on the hard task; one reflects the distribution of cumulative tag-counts for each tag (Section 3.3) in both correct and incorrect HTML code generation samples (Figure 2), and another, the distribution of all tag-counts in incorrect HTML code generations samples (Figure 2). In both of the figures, we notice a common trend of increasing error rates with the increasing number of tag-counts, which confirms that LLMs struggle in structured HTML code generation, especially when they are tasked with generating greater number of HTML tags. It should also be noted that this observation is more pronounced in deeply nested tags such as “div”, “p”, “h1” and “h2” in contrast to the rest of the shallow nested tags, because, depending on the nesting structure, these labels are generated many folds compared to their containing parents (Figure 2).

4.1.4 Math Results

Table 1 reports the math format following the percentage accuracy in the Easy (final answer style) and Hard (final answer and bullet point style) categories, using GSM8K as the underlying benchmark. As consistently shown, most models produce significantly worse GSM8K scores in both Easy and Hard settings compared to how they are normally evaluated in standard benchmarks (Gao et al., 2024). For example, Gemini-1.5-pro achieves 77.3% for Easy and 73.39% for Hard settings while originally achieving 91.7% in the standardized test. In fact, while most closed-source models presented in Table 1 achieve more than 90% in the standard benchmark (Gao et al., 2024), they suffer significant performance drops in our

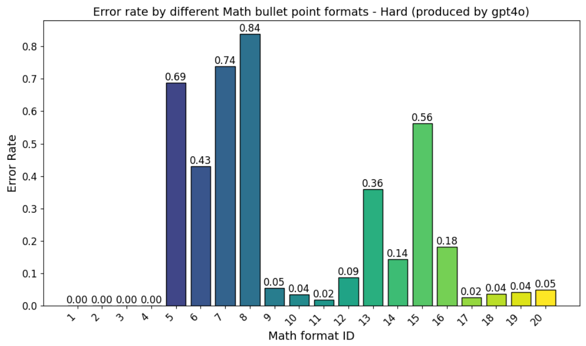

Figure 3: Error Rates of GPT-4o in GSM8K math reasoning across 20 Hard formats.

LLM	StructTest	Arena	MMLU
Phi-3-mini-128k	19.30	1,037	68.10
Mistral-7B	13.94	1,072	60.10
Llama-3.1-8B	33.73	1,175	73.00
Mixtral-8x7B	16.85	1,114	70.60
Llama-3.1-70B	65.99	1,248	86.00
GPT-3.5-turbo	38.43	1,117	70.00
GPT-4o-mini	57.57	1,273	82.00
GPT-4o	72.46	1,366	88.70
Gemini-1.5-pro	61.51	1,302	85.90
Claude-3-haiku	35.95	1,179	75.20
Claude-3-opus	68.81	1,248	86.80
Claude-3.5-sonnet	69.26	1,283	88.70

Table 4: Comparison of StructTest average accuracy with ChatBot Arena score and MMLU accuracy

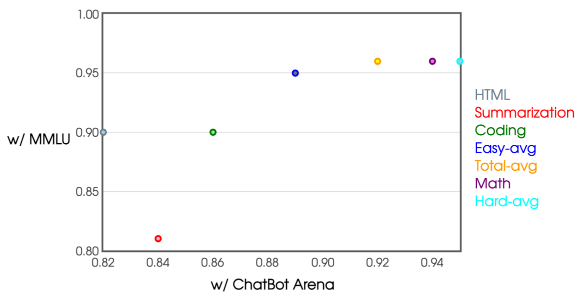

Figure 4: Correlation of various StructTest setups against ChatBot Arena and MMLU

math evaluations, with a margin as high as 70%. This shows that those models are not as reliably and consistently good at math as previously thought. Instead, they may have overfitted to certain formats and styles. Notably, among closed-source group, smaller and older models such as GPT-3.5-turbo or Claude-3-haiku exhibit considerable degradation with scores below 40%. Likewise, the trend is even worse for most open-source models, such as Mixtral-8x7B, which degrade to below 10% accuracy. Overall, the results generally suggest that existing math reasoning comparisons between models are likely unreliable and unfair unless tested on a large number of different, diverse, and impartial formats, which our framework may serve as a more robust evaluation alternative.

To provide better insights, in Figure 3, we plot the error rates of GPT-4o in GSM8K when tested in 20 hard formats. Despite being a significantly strong frontier model, GPT-4o clocks in vastly different performances depending on particular formats. Specifically, it achieves perfect scores, with zero error rate, in format IDs 1 to 4, but fails in other formats with error rates as high as 84%. This suggests that the model may have been overfitted to certain popular formats, while struggles with novel ones. Further manual inspections suggest, however, that the model may still produce accurate final answers, but resist to follow the instructed formats, leading to the samples being marked as failure.

4.2 Correlation to General Reasoning

To understand whether StructTest could be a “cheap” proxy of general reasoning ability in LLMs, we compare the average accuracy in StructTest with popular benchmarks including LLM Arena and MMLU in Table 4. We include all the models for which we could find both Arena and MMLU scores. The correlation (Pearson’s product-moment coefficient) between StructTest and Arena is 92.5% and the same for StructTest and MMLU is 96.3%. Such results highlight that StructTest, though being naturally unbiased, cheap to evaluate, and robust to data contamination, offers strongly correlated evaluation results compared to expensive and resource-intensive benchmarks like ChatBot Arena and MMLU.

To better understand how various task domains and difficulty settings affect correlation to existing benchmarks, we show a 2D scatter plot in Figure 4. We can see that Math has the highest correlation in four task domains and Hard settings yield better correlation than Easy settings.

5 Conclusion

We have proposed StructTest, a programmatically verifiable benchmark for evaluating instruction-following capabilities through structured outputs. StructTest is a cheap-to-run, unbiased, robust benchmark with adjustable difficulty levels. Evaluation results show StructTest to be a good proxy for evaluating general reasoning ability in LLMs. We believe StructTest offers a critical, complementary approach to existing LLM evaluations.

6 Limitations

Constrained Decoding Adding a selective constraint to output space while decoding, has been shown to be effective in boosting performance in entity linking (Cao et al., 2021), machine translation (Post and Vilar, 2018), and secure code generation (Fu et al., 2024). While it is interesting to see how constrained decoding can help StructTest, it is not included in the scope of this study: applying constraints to all domains in StructTest can be a highly complex setup as the allowed output structure in our benchmark is quite flexible for Summarization, Code and Math domains.

Inference Scaling In this paper, we acquire model response based on direct generation given input prompt. In literature, researchers have explored inference scaling methods which increase inference compute for better performance including Monte Carlo Tree Search (MCTS)(Wan et al., 2024b), best-of-n(Li et al., 2023), majority voting(Wang et al., 2023), and reflexion(Shinn et al., 2023). While it is possible to apply inference scaling methods like reflexion to StructTest as a new setting to benchmark how LLMs handle feedback and multi-turn generation, we have not included this aspect in the current version of StructTest and leave it for future work

References

Tommaso Calò and Luigi De Russis. 2023. Leveraging large language models for end-user website generation. In *International Symposium on End User Development*, pages 52–61. Springer.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2021. Autoregressive entity retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. 2023. Booookscore: A systematic exploration of book-length summarization in the era of llms. *arXiv preprint arXiv:2310.00785*.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot arena: An open platform for evaluating llms by human preference. *Preprint*, arXiv:2403.04132.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021a. Training verifiers to solve math word problems. *CoRR*, abs/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*.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. 2021. Towards question-answering as an automatic metric for evaluating the content quality of a summary. *Transactions of the Association for Computational Linguistics*, 9:774–789.

Xuan Long Do, Hai Nguyen Ngoc, Tiviatis Sim, Hieu Dao, Shafiq Joty, Kenji Kawaguchi, Nancy F. Chen, and Min-Yen Kan. 2024. Llms are biased towards output formats! systematically evaluating and mitigating output format bias of llms. *CoRR*, abs/2408.08656.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2024. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *Preprint*, arXiv:2404.04475.

Alexander R Fabbri, Chien-Sheng Wu, Wenhao Liu, and Caiming Xiong. 2021. Qafacteval: Improved qa-based factual consistency evaluation for summarization. *arXiv preprint arXiv:2112.08542*.

Yanjun Fu, Ethan Baker, and Yizheng Chen. 2024. Constrained decoding for secure code generation. *CoRR*, abs/2405.00218.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. 2024. Does prompt formatting have any impact on llm performance? *Preprint*, arXiv:2411.10541.

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. *arXiv preprint arXiv:2103.03874*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Kung-Hsiang Huang, Philippe Laban, Alexander R Fabbri, Prafulla Kumar Choubey, Shafiq Joty, Caiming Xiong, and Chien-Sheng Wu. 2023. Embrace divergence for richer insights: A multi-document summarization benchmark and a case study on summarizing diverse information from news articles. *arXiv preprint arXiv:2309.09369*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint*.

Fangkai Jiao, Geyang Guo, Xingxing Zhang, Nancy F. Chen, Shafiq Joty, and Furu Wei. 2024. Preference optimization for reasoning with pseudo feedback. *CoRR*, abs/2411.16345.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Philippe Laban, Wojciech Kryscinski, Divyansh Agarwal, Alexander Fabbri, Caiming Xiong, Shafiq Joty, and Chien-Sheng Wu. 2023. Summedits: Measuring llm ability at factual reasoning through the lens of summarization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9662–9676, Singapore. Association for Computational Linguistics.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2024. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *Preprint*, arXiv:2406.11939.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *ACL (1)*, pages 5315–5333. Association for Computational Linguistics.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Yixin Liu, Alexander R Fabbri, Jiawen Chen, Yilun Zhao, Simeng Han, Shafiq Joty, Pengfei Liu, Dragomir Radev, Chien-Sheng Wu, and Arman Cohan. 2023. Benchmarking generation and evaluation capabilities of large language models for instruction controllable summarization. *arXiv preprint arXiv:2311.09184*.

Yizhu Liu, Qi Jia, and Kenny Zhu. 2022. Length control in abstractive summarization by pretraining information selection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6885–6895, Dublin, Ireland. Association for Computational Linguistics.

Yizhu Liu, Zhiyi Luo, and Kenny Zhu. 2018. Controlling length in abstractive summarization using a convolutional neural network. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4110–4119, Brussels, Belgium. Association for Computational Linguistics.

Rajdeep Mukherjee, Abhinav Bohra, Akash Banerjee, Soumya Sharma, Manjunath Hegde, Afreen Shaikh, Shivani Shrivastava, Koustuv Dasgupta, Niloy Ganguly, Saptarshi Ghosh, and Pawan Goyal. 2022. ECTSum: A new benchmark dataset for bullet point summarization of long earnings call transcripts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10893–10906, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1314–1324. Association for Computational Linguistics.

Mathieu Ravaut, Bosheng Ding, Fangkai Jiao, Hailin Chen, Xingxuan Li, Ruochen Zhao, Chengwei Qin, Caiming Xiong, and Shafiq Joty. 2024a. How much are llms contaminated? a comprehensive survey and the llmsanitize library. *arXiv preprint arXiv:2404.00699*.

Mathieu Ravaut, Aixin Sun, Nancy Chen, and Shafiq Joty. 2024b. On context utilization in summarization with large language models. In *Proceedings* of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2764–2781, Bangkok, Thailand. Association for Computational Linguistics.

Thomas Scialom, Paul-Alexis Dray, Patrick Gallinari, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, and Alex Wang. 2021. Questeval: Summarization asks for fact-based evaluation. *arXiv preprint arXiv:2103.12693*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*.

Xiangru Tang, Yiming Zong, Jason Phang, Yilun Zhao, Wangchunshu Zhou, Arman Cohan, and Mark Gerstein. 2023. Struc-bench: Are large language models really good at generating complex structured data? *arXiv preprint arXiv:2309.08963*.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv:2402.10176*.

Jesse Vig, Alexander Fabbri, Wojciech Kryscinski, Chien-Sheng Wu, and Wenhao Liu. 2022. Exploring neural models for query-focused summarization. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1455–1468, Seattle, United States. Association for Computational Linguistics.

David Wan, Jesse Vig, Mohit Bansal, and Shafiq Joty. 2024a. On positional bias of faithfulness for long-form summarization. *arXiv preprint arXiv:2410.23609*.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024b. Alphazero-like tree-search can guide large language model decoding and training. In *ICML*. OpenReview.net.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*. OpenReview.net.

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

Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. 2024. FOFO: A benchmark to evaluate LLMs' format-following capability. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 680–699, Bangkok, Thailand. Association for Computational Linguistics.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *CoRR*, abs/2404.07972.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Jing Jiang, and Min Lin. 2024. Cheating automatic llm benchmarks: Null models achieve high win rates. *Preprint*, arXiv:2410.07137.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. Agieval: A human-centric benchmark for evaluating foundation models. In *NAACL-HLT (Findings)*, pages 2299–2314. Association for Computational Linguistics.

A Closed-Source Model Versions

We show the API version used in our evaluation results for close-source models in Appendix A. The inference for all closed-source models was performed during 27th November 2024 to 14th December 2024.

Model	API Version
GPT-3.5-turbo	gpt-3.5-turbo-0125
GPT-4o-mini	gpt-4o-mini-2024-07-18
GPT-4o	gpt-4o-2024-08-06
Gemini-1.5-pro	gemini-1.5-pro-002
Claude-3-haiku	claude-3-haiku-20240307
Claude-3-opus	claude-3-opus-20240229
Claude-3.5-sonnet	claude-3-5-sonnet-20241022

Table 5: Closed-source model versions used in Evaluation Results

B Examples for Different Tasks

We show examples for each summarization task in Figures 5 to 11, and examples for each coding task in Figures 12 to 15.

```
## Instruction

The following is a source document.


Source:
one surprising result that has come out of the more than 200 extrasolar planet discoveries to date is the wide range
of eccentricities observed . unlike our own solar system , many of the extrasolar planets which are not tidally
locked to their host stars have moderate eccentricities ( @xmath1 ) , and 15 planets have high eccentricities (
@xmath0 ) . these observations have spawned several theories as to the origin of highly eccentric extrasolar planets
. one such method , planet - planet scattering , occurs when multiple jovian planets form several astronomical units
( au ) from the host star and then interact , leaving one in an eccentric orbit and often ejecting the other @xcite .
(truncated)


Format Instruction: Please summarize the above text in 5 sentences.{Place the summary between <summary>
and </summary>.

Summary:

#### Response

<summary>
The discovery of over 200 extrasolar planets has revealed a wide range of eccentricities, unlike our
solar system, with many planets exhibiting moderate to high eccentricities. Various theories have been
proposed to explain these eccentricities, including planet-planet scattering and interactions with
stellar companions. This study focuses on six highly eccentric planetary systems (HD 3651, HD 37605, HD
45350, HD 80606, HD 89744, and 16 Cyg B) using observational data and dynamical simulations to explore
the possibility of additional planets. The results show no evidence for additional planets in these
systems, but the simulations indicate regions where additional planets could exist in stable orbits. The
study also sets detection limits for potential additional planets, ruling out short-period planets down
to a few Neptune masses in some systems.
</summary>
```

Figure 5: Test example for **length** task in Summarization.

```
## Instruction

The following is a source document.

Source:

supersymmetry ( susy ) is one of the most attractive extensions of the standard model . this symmetry
solves the naturalness problem and predicts gauge coupling unification at the gut scale @xmath1 . it
also predicts the existence of superpartner of the standard model ( sm ) particles . from the
naturalness argument , their masses should be below tev range , hence these particles will be discovered
at tevatron or large hadron collider ( lhc ) . (truncated)


Format Instruction: Please summarize the above text in {n_points} bullet points using the following
symbol: * to start each bullet point.
Place the summary between <summary> and </summary>. Please don't generate any introductory comments.
Summary:

#### Response

<summary>
* Supersymmetry (SUSY) is an extension of the Standard Model that addresses the naturalness problem,
predicts gauge coupling unification at the GUT scale, and suggests the existence of superpartners for
Standard Model particles, which should be detectable at the Tevatron or LHC.
* The no-scale type Kähler potential, where the hidden and observable sectors are separated, results in
all scalar masses and trilinear scalar couplings vanishing as the cosmological constant vanishes, making
gaugino masses the only source of SUSY breaking.
* Three scenarios for SUSY particle mass spectrum are considered: universal gaugino masses at the GUT
scale, universal gaugino masses above the GUT scale, and non-universal gaugino masses at the GUT scale.
The minimal scenario is severely constrained by cosmological and direct search bounds, while the other
two scenarios offer more viable parameter regions.
* Current constraints from the lightest Higgs mass and the branching ratio for \( b \rightarrow s \gamma
\) further limit the parameter space, with the minimal scenario being almost entirely excluded. The
non-universal gaugino mass scenario allows for wino-like or higgsino-like neutralino as the LSP, which
remains cosmologically viable.
</summary>
```

Figure 6: Test example for **bullet points** task in Summarization.

```
## Instruction

The following is a source document.

Source:

x - ray studies of fairly normal " galaxies , with high - energy emission not obviously dominated by a luminous active galactic
nucleus ( agn ) , have recently been extended to cosmologically interesting distances in the deep field ( cdf ) surveys , which
have now reached 1 ms of exposure ( cdf - n : hornschemeier et al . 2001 , hereafter paper ii ; brandt et al . 2001b , hereafter
paper v ; cdf - s : tozzi et al . 2001 ; p. rosati et al . , in prep . ) . galaxies with @xmath8 are detected in appreciable
numbers at 0.52 kev fluxes below @xmath9 erg @xmath6 s@xmath7 ( e.g. , paper ii ) ; the cdf - n survey goes almost two orders of
magnitude fainter , detecting significant numbers of normal galaxies among the population of x - ray sources making the diffuse x
- ray background ( xrb ; paper ii ; a.j . barger et al . , in prep . ) . (truncated)

Format Instruction: Please summarize the above text in 4 numbered points where each point starts with a number and numbers follow
the correct increasing order.
Place the summary between <summary> and </summary>. Please don't generate any introductory comments.

Summary:


#### Response

<summary>
1. Recent deep field (CDF) surveys have extended X-ray studies of normal galaxies to cosmologically significant distances,
detecting significant numbers of these galaxies at faint flux levels and contributing to the diffuse X-ray background (XRB).

2. The study focuses on the evolution of the X-ray luminosity to optical luminosity ratio of normal spiral galaxies up to redshift
z ≈ 1.5, using spectroscopic and photometric redshift samples to filter galaxies by optical luminosity and morphology.

3. Stacking analysis of X-ray data from the CDF-N survey reveals significant detections in the soft X-ray band for galaxies up to
z ≈ 1.5, with the average X-ray luminosity of these galaxies showing a slight increase with redshift, consistent with heightened
star formation rates in the past.

4. The study finds that normal spiral galaxies contribute up to 5-10% of the XRB flux in the 0.5-2 keV band, with their average
X-ray luminosities evolving by a factor of up to 2.5 by z ≈ 1.5, providing constraints on models of X-ray binary production and
star formation history in galaxies.
</summary>
```

Figure 7: Test example for **numbered points** task in Summarization.

```
## Instruction

The following is a source document.

Source:

one surprising result that has come out of the more than 200 extrasolar planet discoveries to date is the wide range of
eccentricities observed . unlike our own solar system , many of the extrasolar planets which are not tidally locked to their host
stars have moderate eccentricities ( @xmath1 ) , and 15 planets have high eccentricities ( @xmath0 ) . these observations have
spawned several theories as to the origin of highly eccentric extrasolar planets . one such method , planet - planet scattering ,
occurs when multiple jovian planets form several astronomical units ( au ) from the host star and then interact , leaving one in
an eccentric orbit and often ejecting the other @xcite . (truncated)

Place the summary between <summary> and </summary>. Please don't generate any introductory comments.

Summary:


#### Response

<summary>
[What is the main point?]: The main point is the investigation of the presence of additional low-mass planets in six highly
eccentric extrasolar planetary systems using observational data and dynamical simulations.

[Why is it happening?]: This research is happening to understand the architecture of planetary systems, particularly those with
highly eccentric orbits, and to determine where additional planets could exist in stable orbits.

[Who is involved?]: The study involves astronomers and researchers using data from the Hobby-Eberly Telescope (HET) and other
observatories, with contributions from various institutions including the University of Texas at Austin, Pennsylvania State
University, and others.

[When is the action happening?]: The observations and data collection span several years, with specific data points mentioned from
2004 to 2007.

[Where is the action happening?]: The action is happening at multiple observatories, primarily the Hobby-Eberly Telescope (HET) at
McDonald Observatory, and also includes data from the Harlan J. Smith Telescope and other sources.
</summary>
```

Figure 8: Test example for **questions** task in Summarization.

```
## Instruction

The following is a source document.

Source:

the transport properties of nonlinear non - equilibrium dynamical systems are far from well - understood@xcite . \n consider in
particular so - called ratchet systems which are asymmetric periodic potentials where an ensemble of particles experience directed
transport@xcite . \n the origins of the interest in this lie in considerations about extracting useful work from unbiased noisy
fluctuations as seems to happen in biological systems@xcite . (truncated)

Format Instruction: Please summarize the above text in 8 bullet points using the following symbol: • to start each bullet point.
The total length should be 8 sentences, and each bullet point should have exactly 1 sentences. Place the summary between <summary>
and </summary>.

Summary:

#### Response

<summary>
• The transport properties of nonlinear non-equilibrium dynamical systems, particularly chaotic deterministic ratchets, are
complex and sensitive to system parameters and initial conditions.
• Mateos proposed that current reversals in such systems are linked to bifurcations from chaotic to periodic behavior, but Barbi
and Salerno challenged this, suggesting current reversals can occur without such bifurcations.
• The study investigates the impact of different particle ensembles on the current, showing that the current depends significantly
on the initial ensemble and the need to discard transients.
• The results indicate that while bifurcations often correspond to abrupt changes in the current, not all bifurcations lead to
current reversals, and not all current reversals are associated with bifurcations.
• The study finds that single trajectory results can differ significantly from ensemble averages, especially in chaotic regimes,
highlighting the importance of considering ensemble characteristics.
• The research supports a generalized version of Mateos's conjecture, suggesting that bifurcations are associated with sudden
current changes, which may or may not lead to current reversals.
• The study emphasizes the need for detailed analysis of specific systems to discover heuristic rules for understanding chaotic
ratchets.
• The findings suggest that while broad heuristics are rare, understanding ensemble dependence is critical for studying the
transport properties of chaotic systems.
</summary>
```

Figure 9: Test example for **combination of bullet points and length** task in Summarization.

```
## Instruction

The following is a source document.

Source:

the transport properties of nonlinear non - equilibrium dynamical systems are far from well - understood@xcite . \n consider in
particular so - called ratchet systems which are asymmetric periodic potentials where an ensemble of particles experience directed
transport@xcite . \n the origins of the interest in this lie in considerations about extracting useful work from unbiased noisy
fluctuations as seems to happen in biological systems@xcite . (truncated)

Format Instruction:  Please summarize the above text in 8 numbered points where each point starts with a number and numbers follow
the correct increasing order. The total length should be 8 sentences, and each numbered point should have exactly 1 sentences.
Place the summary between <summary> and </summary>.

Summary:

#### Response

<summary>
<summary>
1. The transport properties of nonlinear non-equilibrium dynamical systems, particularly chaotic deterministic ratchets, are
complex and sensitive to system parameters and initial conditions.
2. Mateos proposed that current reversals in such systems are linked to bifurcations from chaotic to periodic behavior, but Barbi
and Salerno challenged this, suggesting current reversals can occur without such bifurcations.
3. The study investigates the impact of different particle ensembles on the current, showing that the current depends
significantly on the initial ensemble and the need to discard transients.
4. The results indicate that while bifurcations often correspond to abrupt changes in the current, not all bifurcations lead to
current reversals, and not all current reversals are associated with bifurcations.
5. The study finds that single trajectory results can differ significantly from ensemble averages, especially in chaotic regimes,
highlighting the importance of considering ensemble characteristics.
6. The research supports a generalized version of Mateos's conjecture, suggesting that bifurcations are associated with sudden
current changes, which may or may not lead to current reversals.
7. The study emphasizes the need for detailed analysis of specific systems to discover heuristic rules for understanding chaotic
ratchets.
8. The findings suggest that while broad heuristics are rare, understanding ensemble dependence is critical for studying the
transport properties of chaotic systems.
</summary>
```

Figure 10: Test example for **combination of numbered points and length** task in Summarization.

```
## Instruction

The following is a source document.

Source:

the transport properties of nonlinear non - equilibrium dynamical systems are far from well - understood@xcite . \n consider in
particular so - called ratchet systems which are asymmetric periodic potentials where an ensemble of particles experience directed
transport@xcite . \n the origins of the interest in this lie in considerations about extracting useful work from unbiased noisy
fluctuations as seems to happen in biological systems@xcite . (truncated)

Format Instruction: Please summarize the following text using 8 bullet points and 2 levels of indentation. Each bullet point starts
with the symbol •, repeated only once. Then, each bullet point should be followed by 1 sub-points, each starting with a tab followed
by the same symbol •, only repeated once as well. Place the summary between <summary> and </summary>.

Summary:

#### Response

<summary>
<summary>
•• The transport properties of nonlinear non-equilibrium dynamical systems, particularly ratchet systems, are complex and not
well-understood.
        •• Ratchet systems involve asymmetric periodic potentials where particles experience directed transport, with interest stemming
from extracting work from noisy fluctuations.
•• Recent studies have focused on deterministic chaotic and Hamiltonian ratchets, with chaotic systems being sensitive to initial
conditions and system parameters.
        •• The phase-space structure of these systems is complicated, with stability islands, chaotic seas, and co-existing attractors
that change with parameters.
•• Mateos proposed a model showing current direction reversal correlated with bifurcations from chaotic to periodic behavior, but this
was contested by Barbi and Salerno.
        •• Barbi and Salerno argued that current reversals could occur without such bifurcations, suggesting stability properties of
rotating periodic orbits as the cause.
•• The current study investigates the impact of different particle ensembles on the current in a chaotic deterministic ratchet system.
        •• It shows that the current depends significantly on the initial ensemble details and emphasizes the importance of discarding
transients.
•• The study finds that while bifurcations often correspond to abrupt changes in the current, not all bifurcations lead to current
reversals.
        •• It suggests that most current reversals are associated with bifurcations, but not all current changes are reversals.
•• The study highlights the importance of ensemble characteristics in understanding transport properties, especially near bifurcation
points.
        •• It proposes a heuristic that bifurcations correspond to sudden current changes, which may or may not be reversals.
•• The study concludes that while general rules of thumb are useful, the behavior of chaotic ratchets is complex and requires detailed
consideration of ensemble characteristics.
        •• Future research should aim to concretize the reasons for when these heuristics are valid or violated.
•• Acknowledgments are given to various individuals and institutions for their support and discussions.
        •• The study references several works in the field of nonlinear physics and complex systems.
</summary>
```

Figure 11: Test example for **indented bullet points** task in Summarization.

````
## Instruction

I will show you a code snippet. Please add a print statement whenever a new variable is defined,
following the format below:

print("X: ", str(x), "Y: ": str(Y), …)

where `X` and `Y` should be the newly defined variables.

### Notes:

- If there is no newly defined variables, do not add any print statements.
- If some variables that were initialized previously are assigned with new values, add print statements
for these variables after the newest assignments, too.
- Do not change any content of the other code
- Overlook the temperature variables like those defined in for loops.

### Response Format

Please include your answer within <ans> and </ans> tags.

### Example

Here is an example for your reference:

#### The code to add print statements

```
def get_last_checkpoint(folder):
    content = os.listdir(folder)
    checkpoints = [
        path
        for path in content
        if _re_checkpoint.search(path) is not None and os.path.isdir(os.path.join(folder, path))
    ]
    if len(checkpoints) == 0:
        return None
    return os.path.join(folder, max(checkpoints, key=lambda x:
int(_re_checkpoint.search(x).groups()[0])))
```

#### Response

<ans>
def get_last_checkpoint(folder):
    content = os.listdir(folder)
    print("content: ", str(content))
    checkpoints = [
        path
        for path in content
        if _re_checkpoint.search(path) is not None and os.path.isdir(os.path.join(folder, path))
    ]
    print("checkpoints: ", str(checkpoints))
    if len(checkpoints) == 0:
        return None
    return os.path.join(folder, max(checkpoints, key=lambda x:
int(_re_checkpoint.search(x).groups()[0])))
</ans>

Now, let's get started:

#### The code to add print statements

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

#### Response
````

Figure 12: Test example for **add 'print' statements** task with one-shot prompting.

````
## Instruction

I will show you a code snippet. Your task is to replace the name of variables to different ones
according to the mapping I give to you, and return me back the new code snippet after replacement.

### Response format

Please inclcude your answer within <ans> and </ans> tags.

Here is an example for your reference:

#### Code Snippet

```
def get_last_checkpoint(folder):
    content = os.listdir(folder)
    checkpoints = [
        path
        for path in content
        if _re_checkpoint.search(path) is not None and os.path.isdir(os.path.join(folder, path))
    ]
    if len(checkpoints) == 0:
        return None
    return os.path.join(folder, max(checkpoints, key=lambda x:
int(_re_checkpoint.search(x).groups()[0])))
```

#### Variable Renaming

```
path -> ppp
content -> ccc
```

#### Response

<ans>
def get_last_checkpoint(folder):
    ccc = os.listdir(folder)
    checkpoints = [
        ppp
        for ppp in ccc
        if _re_checkpoint.search(ppp) is not None and os.path.isdir(os.path.join(folder, ppp))
    ]
    if len(checkpoints) == 0:
        return None
    return os.path.join(folder, max(checkpoints, key=lambda x:
int(_re_checkpoint.search(x).groups()[0])))
</ans>

Now, let's get started:

#### Code Snippet

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

#### Variable Renaming

```
s -> str_var
ch -> char_var
i -> index_var
```

#### Response
````

Figure 13: Test example for **replace variables** task with one-shot prompting.

~~~
## Instruction

You are an expert programmer. I will show you a programming problem as well as one solution program.
Please help me to generate **5** groups of test case inputs to this function.

### Response format

1. Your test case inputs should be in the correct python object format so that we can initialize them
into an argument list by calling `func(*eval(inputs))`.
2. Separate each group of test case inputs simply by new lines.
3. Include all the generated test case inputs within `<ans>` and `</ans>` tags.

Here is an example for your reference:

#### Problem description

Your music player contains N different songs and she wants to listen to L (not necessarily different)
songs during your trip.  You create a playlist so that:

Every song is played at least once
A song can only be played again only if K other songs have been played

Return the number of possible playlists.  As the answer can be very large, return it modulo 10^9 + 7.

#### Solution program

```
def numMusicPlaylists(N: int, L: int, K: int) -> int:
    s=0
    c=0
    r=0
    x=math.factorial(N)
    while(True):
        c=x*((N-r-K)**(L-K))*(-1)**(r)//(math.factorial(N-r-K)*math.factorial(r))
        if(c!=0):
            s=(s+c)%(10**9+7)
            r+=1
        else:
            return s
```

#### Response

<ans>
[3, 3, 1]
[2, 3, 0]
[2, 3, 1]
[4, 3, 1]
[4, 2, 2]
</ans>

Now, let's get started:

#### Program description

Write a python function to remove first and last occurrence of a given character from the string.

#### Solution program

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

#### Response
~~~

Figure 14: Test example for **test case input generation (easy)** task with one-shot prompting. Easy level task contains only function-based code, whereas the hard level task is mainly composed of problems using standard input-output stream.

````
## Instruction

I will show you a program as well as a group of inputs. Please simulate the execution process of this function, and return me
back to the outputs.

## Response Format

Please include your final results following the expected output format within <ans> and </ans> tags.


## Notes:

- You can simulate the program step by step via arbitrary formats. Just remember to tag the final results last.
- Please follow the expected output format defined by the program.
- The output(s) should be in proper python object so that we can use `eval(x)` to initialize it/them.
- The values of different arguments are separated by commas.

Here is an example for your reference:

### Code snippet

```
def maxScore(cardPoints: List[int], k: int) -> int:
    max_score = 0
    curr_score= 0
    init_hand = cardPoints[len(cardPoints)-k:]
    max_score = sum(init_hand)
    curr_score = max_score
    for i in range(k):
        curr_score -= init_hand[i]
        curr_score += cardPoints[i]
        if curr_score > max_score:
            max_score = curr_score
    return max_score
```

### Inputs

```
[1,2,3,4,5,6,1], 3
```

### Response

Let's think step by step.
1. Initialize `max_score` and `curr_score` to 0.
2. Initialize `init_hand` to the last `k` elements of `cardPoints`. As a result, we have `init_hand = [5,6,1]`.
3. Initialize `max_score` to the sum of `init_hand`. As a result, we have `max_score = 12`.
4. Initialize `curr_score` to `max_score`. As a result, we have `curr_score = 12`.
5. Iterate over the range of `k`:
    - In the first iteration, we have `i = 0`.
    - Update `curr_score` by subtracting the first element of `init_hand` and adding the first element of `cardPoints`. As a
result, we have `curr_score = 12 - 5 + 1 = 8`.
    - Check if `curr_score` is greater than `max_score`. Since `8` is not greater than `12`, we don't update `max_score`.
    - In the second iteration, we have `i = 1`.
    - Update `curr_score` by subtracting the second element of `init_hand` and adding the second element of `cardPoints`. As a
result, we have `curr_score = 8 - 6 + 2 = 4`.
    - Check if `curr_score` is greater than `max_score`. Since `4` is not greater than `12`, we don't update `max_score`.
    - In the third iteration, we have `i = 2`.
    - Update `curr_score` by subtracting the third element of `init_hand` and adding the third element of `cardPoints`. As a
result, we have `curr_score = 4 - 1 + 3 = 6`.
    - Check if `curr_score` is greater than `max_score`. Since `6` is not greater than `12`, we don't update `max_score`.
6. Return `max_score`. As a result, we have `12`.

So the final result is
<ans>
12
</ans>
----

Now, let's get started:

### Code snippet

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

### Inputs

```
"hello", "l"
```

### Response
````

Figure 15: Test example for **simulate execution** task with one-shot prompting.