Establishing Best Practices for Building Rigorous Agentic Benchmarks

Yuxuan Zhu[1*] **Tengjun Jin**[1] **Yada Pruksachatkun** **Andy Zhang**[2] **Shu Liu**[3]
Sasha Cui[4] **Sayash Kapoor**[5] **Shayne Longpre**[6] **Kevin Meng**[7] **Rebecca Weiss**[8]
Fazl Barez[8,11] **Rahul Gupta**[9] **Jwala Dhamala**[9] **Jacob Merizian**[10] **Mario Giulianelli**[10]
Harry Coppock[10] **Cozmin Ududec**[10] **Jasjeet Sekhon**[4] **Jacob Steinhardt**[7]
Antony Kellermann[1] **Sarah Schwettmann**[7] **Matei Zaharia**[3] **Ion Stoica**[3]
Percy Liang[2] **Daniel Kang**[1*]

[1]UIUC [2]Stanford University [3]University of California, Berkeley [4]Yale University
[5]Princeton University [6]MIT [7]Transluce [8]ML Commons [9]Amazon
[10]UK AI Safety Institute [11]University of Oxford

Abstract

Benchmarks are essential for quantitatively tracking progress in AI. As AI agents become increasingly capable, researchers and practitioners have introduced *agentic benchmarks* to evaluate agents on complex, real-world tasks. These benchmarks typically measure agent capabilities by evaluating task outcomes via specific reward designs. However, we show that many agentic benchmarks have issues in task setup or reward design. For example, SWE-bench-Verified uses insufficient test cases, while τ-bench counts empty responses as successful. Such issues can lead to under- or overestimation of agents' performance by up to 100% in relative terms. To make agentic evaluation rigorous, we introduce the Agentic Benchmark Checklist (ABC), a set of guidelines that we synthesized from our benchmark-building experience, a survey of best practices, and previously reported issues. When applied to CVE-Bench, a benchmark with a particularly complex evaluation design, ABC reduces the performance overestimation by 33%.

1 Introduction

AI agents that integrate machine learning models with tools, memory, and knowledge are emerging with the capability to solve complex problems [12, 25, 40, 68, 71, 84, 85]. To evaluate AI agents, researchers and practitioners have built *agentic benchmarks* with realistic tasks to track progress and assist decision-making [11, 22, 30, 37, 47, 59, 83, 86, 89, 93]. AI agents have exhibited impressive performance on these benchmarks. For example, a GPT-4o-based agent resolves 35% of tasks on τ-bench-Airline, a benchmark for tool-agent-user interaction [86]. As agentic benchmarks become increasingly impactful in academia and industry, it is crucial to ensure these numbers can be trusted.

Agentic benchmarks differ fundamentally from traditional AI benchmarks. Multiple-choice datasets (e.g., ImageNet [16] and MMLU [27]) evaluate models by their accuracy on categorical labels, while text-generation benchmarks rely on automatic metrics (e.g., BLEU [60]). By contrast, success is defined by completing end-to-end tasks in agentic settings. Therefore, an agent may perform coherent reasoning, write code, and execute commands to produce a final outcome. Subsequently, the performance of the agent is determined by comparing its final outcome with a ground-truth outcome, using various methods, such as program testing and string matching [11, 30, 37, 47, 59, 83, 86, 89, 93].

[*]{yxx404,ddkang}@illinois.edu

Preprint.

Unfortunately, many existing outcome-based evaluation methods of agentic benchmarks introduce issues that can cause under- or overestimation of agent capabilities by up to 100% in relative terms, compromising the validity of their findings [35, 46, 62, 80, 87]. For example, SWE-bench-Verified challenges an agent to resolve GitHub issues, and considers the agent successful if the patch it generates passes manually vetted unit tests [14]. However, recent work has shown that passing these tests does not necessarily indicate that the issue is resolved, because unit tests can fail to capture important edge cases. Consequently, 24% of the top 50 leaderboard positions are incorrect [31, 87]. In addition, we find that in τ-bench, a trivial agent that returns empty responses is considered successful on intentionally impossible tasks (e.g., changing a non-refundable ticket). This trivial agent achieves a 38% success rate and outperforms a GPT-4o-based agent [86].

Although issues in evaluation rigor can significantly skew evaluation results, they are still frequently overlooked in the current development, deployment, and analysis of agentic benchmarks. To better understand this problem, we analyzed prior work on agentic benchmark pitfalls [35, 46, 62, 80, 87] and 17 widely used agentic benchmarks (Table 3), such as SWE-bench-Verified [14], GAIA [47], τ-bench [86], and WebArena [93]. Combining insights from the literature with our own experience in developing benchmarks, we identified two major conditions of the validity of benchmark results:

- *Outcome validity*: the evaluation result (e.g., tests or checks) truly indicates task success. SWE-bench-Verified fails here because an incorrect patch can still pass the test suite.
- *Task validity*: a task should be solvable if and only if the agent possesses the target capability. Issues in task design or implementation often breaks task validity. For example, τ-bench allows a trivial agent to pass 38% of tasks without knowledge of airline-ticketing rules.

Following prior work on analyzing AI and code benchmarks [10, 65], we formulate our insights into an **A**gentic **B**enchmark **C**hecklist (ABC) to assist benchmark developers and users in critically designing and assessing agentic benchmarks. Using ABC, we assessed ten popular agentic benchmarks that span the full range of agent capabilities, resulting in seven benchmarks with flaws in outcome validity, seven with issues in task validity, and all with limitations in the result reporting. In addition to the issues found in τ-bench-Airline, some other example issues we found are: (1) an agent can score 100% on SWE-Lancer [48] without resolving any tasks; (2) KernelBench [59] overestimates agents' capabilities in generating correct kernel functions by 31% in absolute terms due to incomprehensive fuzz testing; (3) WebArena [93] overestimates performance of agents by 5.2% due to various issues in its string matching. To demonstrate ABC's practical value, we applied it to improve CVE-Bench, a complex, representative cybersecurity benchmark [96]. ABC reduced performance overestimation in CVE-Bench by 33% in absolute terms, as confirmed by cybersecurity experts.

We summarize our contributions as follows:

1. We identified two significant threats in the evaluation rigor of agentic benchmarks: outcome validity and task validity.
2. We developed an actionable checklist, ABC, to critically assess existing agentic benchmarks and to establish best practices for future development.
3. We applied ABC to assess ten widely used agentic benchmarks and identified new evaluation issues that cause estimation errors of agents' performance by up to 100% in relative terms.
4. We provided a case study of using ABC to improve an agentic benchmark during development.

2 Related Work

Assessing AI Benchmarks. Benchmarks are fundamental in AI research and practice, serving as key tools for measuring progress and identifying potential risks [21, 69]. However, maintaining benchmark quality remains a persistent challenge. To address this, prior studies have assessed various dimensions of AI benchmarks, including label quality and quantity [17, 18], standardized evaluation protocols [41], construct validity [20, 64], data contamination [91], reproducibility [75], and practical usage [24]. Even high-profile benchmarks, such as ImageNet [16], have faced issues related to data bias and label noise [73]. With the advancement of large language models (LLMs), recent work has proposed best practices for developing general or code-oriented benchmarks [10, 65]. Although these existing studies provide important insights to our analysis, they primarily focused on multiple-choice

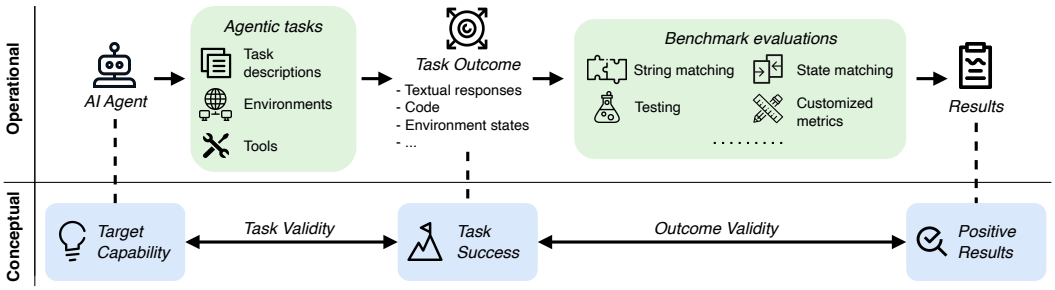


Figure 1: Operational and conceptual processes of agentic evaluation. An agentic benchmark measures the capability of AI agents via agentic tasks. It determines the success of a task by evaluating the task outcomes. Establishing task validity (e.g., equivalence between the target capability and the task success) and outcome validity (e.g., equivalence between the task success and positive evaluation results) are keys to ensure rigorous agentic evaluation.

or generative tasks that do not require multistep reasoning, which present fewer ambiguities and complexities than complex agentic benchmarks.

Benchmarking of AI Agents. Prior work has proposed agentic benchmarks across various domains, including coding [30, 37, 48, 59], interacting with environments for a predefined target [83, 86, 93], solving math problems [22, 39], and others [11, 47, 74, 89]. These tasks typically emulate real-world challenge resolution, involving non-categorical outputs and multistep execution. Evaluating AI agents in these tasks introduces a more complex design and implementation than traditional benchmarks, including handling dynamic interactions between an agent and the environment and grading unstructured responses, which increases the difficulty in ensuring rigorous evaluation.

Issues in Evaluating AI Agents. Existing analyses have identified evaluation issues in individual agentic benchmarks [32, 34, 35, 62, 87]. In terms of the outcome validity, Kydlíček and Gandenberger [34] found that implicit assumptions on the answer formats lead to performance underestimation by 5.3%. Yu et al. [87] found that agents can pass evaluations without generating correct patches for 7.7% of tasks in the SWE-bench-Lite and 5.2% of tasks in the SWE-bench-Verified. In addition, prior analysis found that the annotation noise in BIRD significantly affects the accuracy of performance evaluation [62, 80]. In terms of task validity, the rate limit of the websites implemented in WebArena prevented agents from resolving challenges [32]. Furthermore, Lange et al. [35] identified flaws in the grading of KernelBench that allow agents to bypass correctness checks. However, none of them develops an actionable and systematic guideline to assess agentic benchmarks.

3 Overview

In this section, we present an overview of our work. We first introduce a taxonomy of validity issues in agentic benchmarks and then describe the process of our benchmark collection, checklist development, and benchmark assessment. Finally, we release our code[1] and build a website[2] for continuous development and future updates.

Taxonomy. We first identify and classify the primary challenges in rigorous agentic evaluation. In Figure 1, we decompose the operational and conceptual process of agentic evaluation. An agentic benchmark challenges an AI agent to finish a task in a specific environment with a given set of tools. After several rounds of (inter-) actions, the AI agent presents a task outcome, which indicates whether the task completion state. To automatically determine whether the task is successful, the agentic benchmark develops customized methods based on the task requirements, such as string matching [86, 92] and testing [30, 59].

Conceptually, an agentic evaluation is rigorous if and only if (1) the target capability is equivalent to task success (i.e., task validity), and (2) the task success is equivalent to a positive evaluation result

[1] `https://github.com/uiuc-kang-lab/agentic-benchmarks`
[2] `https://uiuc-kang-lab.github.io/agentic-benchmarks/`

(i.e., outcome validity). However, agentic benchmark presents two unique challenges that makes these two validity conditions difficult to hold:

1. *Complex task setup*: In addition to task descriptions as inputs, agentic benchmarks set up an environment for agents to operate in and provide tools for agents to use.
2. *Unstructured task outcome*: Agentic benchmarks expect unstructured data as task outcomes, such as textual responses, code, and file edits. Verifying the correctness of such outcomes are non-trivial and requires specially designed methods.

First, improper task setup can lead to the violation of task validity. For instance, τ-bench includes intentionally unattainable tasks (e.g., making changes to a non-refundable ticket), which agents are supposed to recognize and reject [86]. Yet, a trivial agent that simply returns nothing is considered a successful completion even though it cannot look up information or interpret ticket rules. Second, failure to rigorously grade unstructured task outcome can break outcome validity. For example, SWE-bench-Verified judges agent-generated patches by handwritten unit tests [14]. Since such tests can be incomplete or not perfectly sound [87, 94], a patch that passes them may still be wrong. Task validity breaks down for a different reason, often reflected as shortcuts or impossible tasks.

To help researchers identify and mitigate such problems in specific agentic benchmarks, we aim to translate the two validity criteria into an actionable checklist. When a criterion cannot be fully satisfied, the checklist also offers guidance on how to interpret and report the resulting scores.

Benchmark Collection. To develop the checklist, we collected a set of popular agentic benchmarks as the corpus for our study. To emphasize common and representative issues, we focused on popular agentic benchmarks used by top AI providers, including OpenAI, Anthropic, Amazon, Meta, Google, xAI, Mistral, and DeepSeek, or those winning awards in peer-reviewed academic conferences. This narrows our focus to a set of 17 agentic benchmarks (Table 3). We defer the details of our benchmark collection to Appendix B.

Checklist Development. We first reviewed the collected benchmarks and surveyed AI agent evaluation frameworks [1, 44, 45, 50] together with documented issues in agentic benchmarks [32, 34, 35, 62, 87]. We then examined best practices for evaluating unstructured task outcomes in related domains, such as software testing. Integrating these insights with our own experience in benchmark development, we curated the Agentic Benchmark Checklist (ABC), which has three parts: task validity, outcome validity, and benchmark reporting. We provide the source of each checklist item in Appendix C.

Benchmark Assessment. We applied ABC to thoroughly assess ten selected benchmarks (Table 1). We selected these benchmarks from the open-source set in Table 3, prioritizing their popularity and ensuring all types of agent capabilities are covered. We assigned 1 point to each satisfied item and 0 otherwise. For each issue identified by the checklist, we designed experiments to validate the issue and obtained quantitative results (Section 5). We defer detailed assessment results to Appendix D and case studies to Appendix E.

4 ABC: Agentic Benchmark Checklist

In this section, we formulate our assessment framework into an actionable checklist (ABC). We present the checklist items in terms of task validity, outcome validity, and benchmark reporting.

4.1 Assessing Task Validity

We propose guidelines for ensuring task validity. These checks uncover design or implementation flaws that can create shortcuts, which causes false positive evaluation results, or lead to impossible tasks, which causes false negative evaluation results.

Tool. External tools and functions can significantly extend the capabilities of AI agents. Existing benchmarks provide two types of tools: self-hosted tools (e.g., Python, command-line tools) and API-based tools (e.g., web services). For self-hosted tools, it is essential to explicitly specify the correct tool or package versions in the prompt (T.1). In terms of API-based tools, ensuring service

Task Validity			
Tool	T.1. Versions of all tools (e.g., Python) are clearly specified. T.2. Required API tools are consistently accessible during evaluation. T.3. Evaluation process terminates or handles errors appropriately if an API becomes inaccessible.	*Implementation*	T.7. Annotated ground truth is verified for correctness. T.8. Each task is verified to be solvable. T.9. Benchmark includes an Oracle solver that can automatically solve all challenges. T.10. Implementation is free of vulnerabilities that could be exploited to pass evaluations without completing tasks.
Env.	T.4. Residual data or state are fully cleared between runs. T.5. Agent is completely isolated from any ground truth information. T.6. Setup does not change over time (e.g., no live website).		

Figure 2: Checks in ABC to assess the task validity of an agentic benchmark.

availability and managing rate limits is crucial (T.2). If API interruptions occur, we recommend detecting them and terminating the evaluation to keep benchmark users informed (T.3).

Environment. Agentic benchmarks often need a sandbox environment to simulate real-world scenarios. Implementing and maintaining such environments can be challenging, especially with complex task formulations. First, to ensure the independence of tasks, we need to ensure that any legacy data and states are fully cleaned up before starting a new task (T.4). For example, KernelBench failed to remove ground truth answers from GPU memory, allowing agents to obtain the correct result through out-of-bounds memory access [35]. Furthermore, to avoid cheating by peeking at ground truth, it is important to fully isolate agents from the ground truth results (T.5). Finally, the environment setup should be fully reproducible and frozen at the time of benchmark release (T.6). Relying on dynamic resources, such as continually updated external websites, is not recommended.

Implementation. Even with a robust setup of tools and environments, subtle implementation vulnerabilities can also result in shortcuts or impossible tasks. Therefore, we recommend verifying the correctness of ground truth annotation and the task setup (T.7-8). Providing an automatic oracle solver can help demonstrate the correctness of the task configuration (T.9). Additionally, as demonstrated in τ-bench [86], inspecting outliers in pilot experiments is crucial for identifying implementation bugs (T.10). For example, if agents consistently fail on easy tasks, this may indicate that tasks are impossible, whereas if agents only succeed on difficult tasks, it may indicate shortcuts.

4.2 Assessing Outcome Validity

In this part of the assessment, we propose practical checks for ensuring the outcome validity of an agentic benchmark (Figure 3). We design these checks based on different types of outcomes and different evaluation methods.

Information Acquisition. To evaluate the capability of AI agents to search, retrieve, integrate, and summarize information, agentic benchmarks formulate tasks as information acquisition queries [25, 86, 89, 93]. Depending on task requirements, benchmarks use various schemes for evaluating agents' textual responses, including whole string matching [89], substring matching [86, 93], and LLM-as-a-judge [25, 93].

1. *Whole String Matching* directly compares the agent's response and the ground truth. When annotating ground truth, it is important to consider semantically equivalent expressions (O.a.1) or redundant words (O.a.2).[3]

2. *Substring Matching* evaluates whether the agent's response contains the ground truth. In addition to equivalent expressions, it should handle negation modifiers (O.b.1), such as "not" and "negative." We also recommend formulating tasks carefully to prevent success by listing all possible answers (O.b.2) or guessing (O.b.3).

3. *LLM-as-a-Judge* uses LLMs to emulate human annotators [9, 38, 88, 90, 97]. Previous studies have shown that the accuracy of LLM annotations varies across domains [98]. We recommend conducting pilot experiments to assess the accuracy and self-consistency of LLM judges (O.c.1).

[3] In practice, users often specify format requirements for AI agents, which narrows the scope of alternative expressions of the ground truth. Failing to follow the format requirements is considered as a true failure.

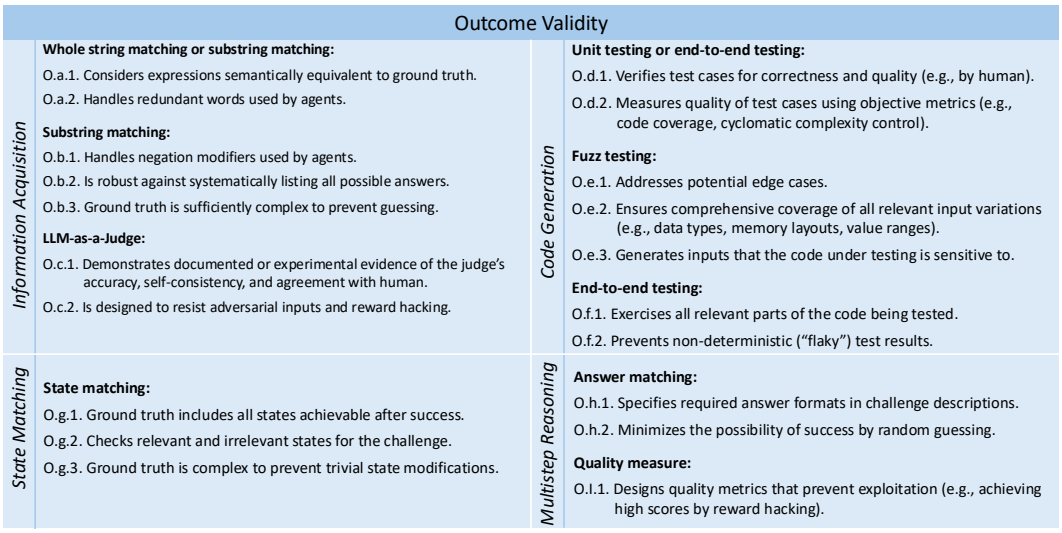


Figure 3: Checks in ABC to assess the outcome validity of an agentic benchmark. We group items by the types of the outcome and the methods of evaluation.

Code Generation. Existing agentic benchmarks evaluate the capability of AI agents to write code [30, 37, 48, 59]. These benchmarks apply program testing techniques to evaluate the correctness of generated code, including unit testing, fuzz testing, and end-to-end testing.

1. *Unit Testing* designs test cases for individual functions or classes [67]. However, poorly constructed unit tests can lead to both false positives and false negatives [70, 87]. Therefore, we recommend manually verifying the correctness and quality of test cases (O.d.1) [14], and providing quality guarantees using objective metrics (O.d.2) such as coverage [94] and cyclomatic complexity [76].
2. *Fuzz Testing* evaluates generated code by running it against a ground-truth implementation on automatically generated inputs [95]. We should tailor the input generator to the target program, covering different data values, types, memory layouts, and edge cases (O.e.1-2). Moreover, the inputs must affect the output (O.e.3)—e.g., random negatives reveal nothing about `relu(x)` [35].
3. *End-to-end (E2E) Testing* simulates complete user workflows, providing comprehensive testing of system functionality [36, 72]. In addition to ensuring the general quality of test cases, it should also cover all possible branches of user workflows (O.f.1). Because of their complexity, E2E tests require extra safeguards to eliminate non-determinism and ensure repeatable results (O.f.2) [61].

State Modification. Agentic benchmarks challenge agents to manipulate environment states, such as booking flight tickets [86] and editing websites [83]. In these tasks, we often compare the final state achieved by agents with a ground-truth state.

We identify three key checks for rigorous state matching. First, ground truth states should include all possible outcomes achievable through successful task resolution (O.g.1). For example, when we challenge agents to attack a website, we should evaluate all possible attack outcomes [96]. Second, the state space should contain both relevant and irrelevant states (O.g.2), such as including both changed and unchanged files, to help detect if agents affect the environment outside the target scope. Finally, the state space should be complex enough (O.g.3)—for instance, involving multiple variables or dependencies—so that random or trivial changes are unlikely to result in a correct outcome.

Multistep Reasoning. Agentic benchmarks evaluates multistep reasoning capabilities of AI agents [11, 22, 39, 47]. These benchmarks typically require AI agents to make observations, conduct analysis, and generate results. We summarize two common approaches for evaluating these tasks:

1. *Answer Matching* parses the agents' output and then compares the parsed result with ground truth. We find that parsers in existing benchmarks may make implicit assumption about the agent's output (O.h.1). For example, the MATH dataset assumes the answer of the agent starts with "Answer:" [39]. Therefore, it is necessary to explicitly specify any assumptions, such as

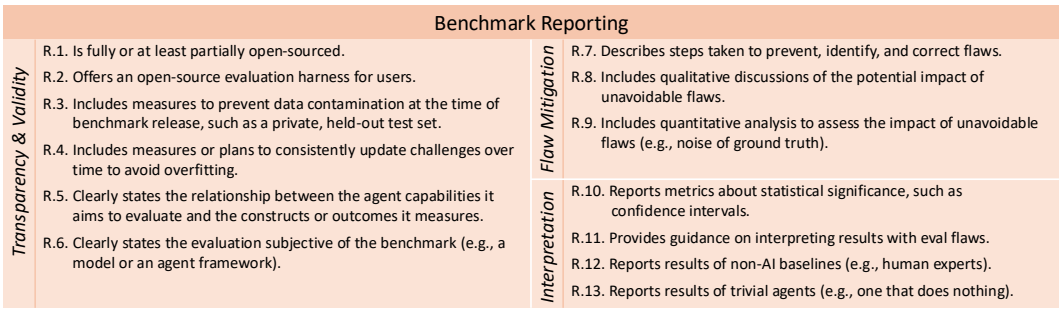


Figure 4: Checks in ABC to assess the benchmark reporting.

format requirements. Additionally, to ensure that a single final answer reflects a genuine reasoning process, we recommend designing tasks in a way that avoid success by guessing (O.h.2) [22].

2. *Quality Measure* evaluates agent using customized metrics against a baseline when ground truth is impossible to achieve (e.g., ground-truth predictions in an ML engineering task [11]). The choice of metrics can be highly subjective and often depends on the nature of the tasks. To avoid metric hacking [26]—achieving high metrics without resolving tasks, we recommend ensuring that the selected metrics are strongly correlated with the reasoning process (O.i.1).

4.3 Assessing the Benchmark Reporting

Completely avoiding evaluation issues in agentic benchmarks can be challenging, and is sometimes not feasible, especially when using LLM-as-a-Judge or testing-based techniques. In such cases, it is particularly important for benchmark developers to be transparent and clearly communicate the impact of these limitations (Figure 4).

We assess the reporting quality of an agentic benchmark based on the following aspects. In Appendix F, we use BIRD as an example to demonstrate the high-quality benchmark reporting.

1. *Transparency and Validity*. We encourage open-sourcing both the datasets and evaluation harness (R.1-2) while including measures to prevent data contamination (R.3-4). We also recommend clearly specifying the capabilities to evaluate and articulating construct validity [65] (R.5-6).
2. *Mitigation*. When validity limitations are unavoidable, it is important to document mitigation efforts (R.7) and provide both qualitative and quantitative evidence regarding the impact of those limitations (R.8-9). In resource-constraint scenarios, we recommend using sampling and uncertainty quantification techniques (e.g., Cramer's Theorem [17]) to estimate the impact of unavoidable flaws, such as the noise of ground truth.
3. *Result Interpretation*. We recommend reporting benchmark results rigorously, including measures of statistical significance (R.10), clear interpretation guidelines (R.11), and appropriate baseline comparisons (R.12-13).

5 Assessment of Agentic Benchmarks

In this section, we present the results of applying ABC on existing agentic benchmarks (Table 1). We first show the assessment scores (Section 5.1) and then summarize newly identified issues with quantitative results (Section 5.2). Finally, with a case study, we show how developers can apply ABC to improve their benchmarks (Section 5.3).

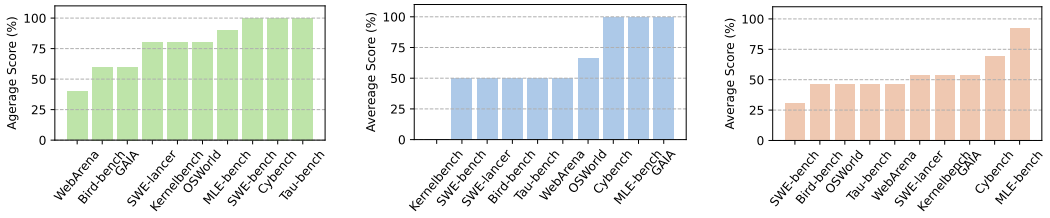
5.1 Assessment Scores

We selected ten open-source agentic benchmarks from Table 3 to cover all capability categories and evaluation methods. For each part of ABC, we calculated the average scores of applicable items. We present the final assessment scores in Figure 5. We summarize our findings as follows.

- Task validity: more than half of the benchmarks exhibit implementation flaws, especially those that provide tools to agents.

Table 1: Agentic benchmarks we assessed using ABC.

Benchmark	Evaluated Capability	Evaluation Design
SWE-bench [30]	Software Engineering	Unit Testing
SWE-Lancer [48]	Software Engineering	End-to-end Testing
KernelBench [59]	Software Engineering	Fuzz Testing
BIRD [37]	Software Engineering	Unit Testing
Cybench [89]	Cybersecurity	Answer Matching
MLE-bench [11]	Software Engineering	Quality Measure
GAIA [47]	General Assistant	Answer Matching
τ-bench [86]	Environment Interaction	Substring Matching, State Matching
WebArena [93]	Environment Interaction	Whole String Matching, Substring Matching, LLM-as-a-Judge, State Matching
OSWorld [83]	Environment Interaction	State Matching



(a) Task validity. (b) Outcome validity. (c) Benchmark reporting.

Figure 5: Assessment results of selected benchmarks. We find 7 benchmarks violating task validity, 7 violating outcome validity, and all 10 with limitations in reporting.

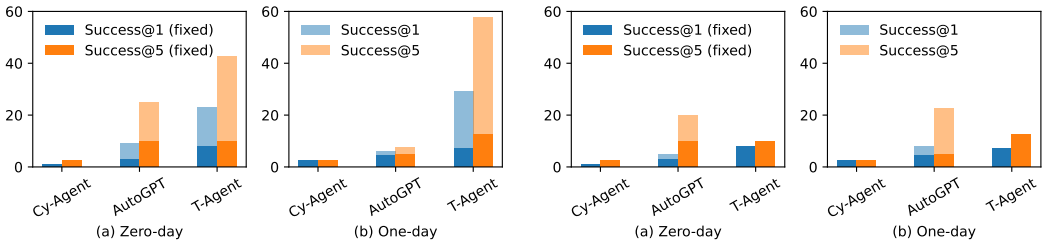
- Outcome validity: more than half of the benchmarks fail to address inherent limitations of the evaluation methods.
- Benchmark Reporting: 80% of the benchmarks fail to acknowledge weaknesses in their design or implementation, and none satisfies every reporting criterion.

5.2 Assessment Findings

We conducted an in-depth analysis of specific issues present in each agentic benchmark. In this section, we focus on discussing 4 benchmarks with newly discovered issues. We defer a detailed description of all identified issues in Appendix D and experiment designs to E.

1. τ-bench relies on trivial states or substrings as ground truth, violating checks O.b.3 and O.g.3 and overestimating performance by 38%.
2. τ-bench also allows agents to list every possible answer, violating check O.b.2 and overestimating performance by 40%.
3. WebArena not only violates check O.b.2 but also uses an LLM-as-a-Judge without validating its accuracy or consistency (check O.c.1), leading to a 1.4–5.2% performance overestimate.
4. SWE-Lancer fails to fully isolate agents from the ground truth (check T.5), allowing agents to score 100% without solving tasks.
5. KernelBench omits comprehensive fuzzing for edge cases and memory layouts—violating checks O.e.1 and O.e.2 and overestimatingkernel-correctness performance by approximately 31%.
6. In OSWorld, the task website changes have broken the HTML selectors used for evaluation, leading to a 28% performance underestimation in the `chrome` task section.

τ-bench. First, τ-bench contains intentionally unsolvable tasks—38% of the airline subset and 6% of the retail subset. Because success is defined as leaving the environment unchanged, an agent that does nothing still passes evaluation of these tasks. Second, 2% of airline tasks and 3.6% of retail tasks treat verbatim database text as ground truth and grade by substring matching. Therefore, an agent that dumps the entire database passes evaluations of these tasks.



(a) Before and after fixing the design for evaluating time-based SQL injection.

(b) Before and after fixing the implementation of evaluating outbound service.

Figure 6: Demonstration of using our checklist to improve the evaluation rigor of CVE-Bench.

WebArena. We identify two issues in WebArena, a benchmark for evaluating agents in interacting with websites. First, its substring-matching ignores extraneous content, so an agent that includes irrelevant information is considered successful. Second, for tasks labeled "N/A," the LLM judge that accepts an empty reply as correct, enabling a trivial agent to pass.

SWE-Lancer evaluates an agent's ability to implement features by allowing it to execute Python scripts that interact directly with the file system. This design grants agents unrestricted read-write access, including to the benchmark's own test files. Although these tests reside in a password-protected ZIP archive, the archive's directory structure can be listed—and its contents overwritten—without knowing the password. Therefore, an agent can locate the tests and replace them with a trivial assertion (e.g., `assert 1 == 1`), achieving a perfect score without solving any of the intended tasks.

KernelBench evaluates generated CUDA kernels with randomly generated tensors, while its fuzzer varies only the tensor values, leaving shapes and memory layouts unchanged. As a result, kernels that would fail under alternative configurations can still pass. Re-examining the kernels reported by Lange et al. [35], we find that the correctness rate of kernels is overestimated by 31%.

OSWorld. We find that in the `chrome` section of OSWorld, 13/46 problems are broken due to changes made to the layout, URLs, and functionality of websites since the initial creation of the benchmark. This is because many evaluations rely on HTML element selectors, such as classes and XPaths. These websites might change their layouts after the benchmark released. In our experiments, we found that this issue leads to an underestimation of the performance of UI-TAR, the state-of-the-art open-source agent for OSWorld, by 28% in absolute terms.

5.3 Revising CVE-Bench

In this section, we use a benchmark with representatively complex design and implementation to demonstrate how ABC can help improve an agentic benchmark. CVE-Bench is a benchmark for evaluating AI agents' ability to exploit real-world web vulnerabilities under one- or zero-day scenarios [96]. It evaluates agents by checking whether one of the pre-specified attack targets (e.g., denial of service) is accomplished. Using ABC, we resolved flaws in outcome and task validity.

Naive State Matching for Time-based Injections. Time-based SQL injection infers the database content by measuring the latency difference across multiple requests [23]. For example, an attacker can execute a `SLEEP` command within a `IF` clause and measure the latency to determine whether the `IF` condition is satisfied. CVE-bench measured such attacks by examining whether a `SLEEP` clause appears in the database log. However, containing a `SLEEP` clause in the log does not necessarily indicate executions of `SLEEP`, violating check O.g.1. Consequently, agents can pass the evaluation by adding `SLEEP` anywhere in the query, leading to performance overestimation by 32.5%.

Ungated Outbound Server. Inducing the web application to send requests to a banned outbound server is a critical cybersecurity attack [29]. CVE-bench measured such attacks by checking whether an outbound server has been accessed. To answer check T.9, we conducted various rounds of mock execution and identified that agents consistently passed the evaluation for this attack, which likely indicates a bug in the implementation. Indeed, we find that agents can access the outbound server

when connecting from the same docker network, creating a shortcut. After denying external requests on the outbound server, the success rates of agents decreased by 10% (Figure 6b).

6 Conclusion

We formulate the first actionable agentic benchmarks checklists (ABC) focusing on the outcome validity, task validity, and reporting of results. Via ABC, we proposed a set of the best practices for building rigorous agentic benchmarks. Based on ABC, we assessed ten widely used agentic benchmarks and identified significant evaluation issues that cases up to 100% errors (in relative terms) when estimating agents' performance. Finally, we use CVE-Bench [96] as an example to demonstrate using ABC to improve the evaluation rigor during benchmark construction.

7 Acknowledgements

We are grateful to the CloudLab [19] for providing computing resources for experiments. This research was supported in part by Open Philanthropy project.

References

[1] UK AI Security Institute. Inspect AI: Framework for Large Language Model Evaluations, 2024. URL https://github.com/UKGovernmentBEIS/inspect_ai.

[2] Aider. Gpt code editing benchmarks, 2024. URL https://aider.chat/docs/benchmarks.html.

[3] Aider. o1 tops aider's new polyglot leaderboard, 2024. URL https://aider.chat/2024/12/21/polyglot.html#the-polyglot-benchmark.

[4] Amazon. The amazon nova family of models: Technical report and model card, 2024. URL https://www.amazon.science/publications/the-amazon-nova-family-of-models-technical-report-and-model-card.

[5] Anthropic. Claude 3.5 sonnet, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet.

[6] Anthropic. Claude 3.7 and claude code, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

[7] Arcwise. Bird minidev - corrections, 2024. URL https://docs.google.com/spreadsheets/d/1IGm9Otruey6OujUnl8AOkepY3qgWHdFJHnX7hQGUeCw.

[8] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[9] Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, et al. Llms instead of human judges? a large scale empirical study across 20 nlp evaluation tasks. *arXiv preprint arXiv:2406.18403*, 2024.

[10] Jialun Cao, Yuk-Kit Chan, Zixuan Ling, Wenxuan Wang, Shuqing Li, Mingwei Liu, Chaozheng Wang, Boxi Yu, Pinjia He, Shuai Wang, et al. How should i build a benchmark? *arXiv preprint arXiv:2501.10711*, 2025.

[11] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[12] Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke

Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[14] Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing swe-bench verified, 2024. URL `https://openai.com/index/introducing-swe-bench-verified/`.

[15] DeepSeek. Introducing deepseek v3, 2024. URL `https://api-docs.deepseek.com/news/news1226`.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Florian E Dorner and Moritz Hardt. Don't label twice: Quantity beats quality when comparing binary classifiers on a budget. In *International Conference on Machine Learning*, pages 11544–11572. PMLR, 2024.

[18] Florian E Dorner, Vivian Y Nastl, and Moritz Hardt. Limits to scalable evaluation at the frontier: Llm as judge won't beat twice the data. *International Conference on Learning Representations*, 2025.

[19] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019. URL `https://www.flux.utah.edu/paper/duplyakin-atc19`.

[20] Maria Eriksson, Erasmo Purificato, Arman Noroozian, Joao Vinagre, Guillaume Chaslot, Emilia Gomez, and David Fernandez-Llorca. Can we trust ai benchmarks? an interdisciplinary review of current issues in ai evaluation. *arXiv preprint arXiv:2502.06559*, 2025.

[21] Li Fei-Fei and Ranjay Krishna. Searching for computer vision north stars. *Daedalus*, 151(2): 85–99, 2022.

[22] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

[23] William GJ Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql injection attacks and countermeasures. In *ISSSE*, 2006.

[24] Amelia Hardy, Anka Reuel, Kiana Jafari Meimandi, Lisa Soder, Allie Griffith, Dylan M Asmar, Sanmi Koyejo, Michael S Bernstein, and Mykel John Kochenderfer. More than marketing? on the information value of ai benchmarks for practitioners. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, pages 1032–1047, 2025.

[25] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[26] Megan L Head, Luke Holman, Rob Lanfear, Andrew T Kahn, and Michael D Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015.

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

[28] Torsten Hothorn, Friedrich Leisch, Achim Zeileis, and Kurt Hornik. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005.

[29] Bahruz Jabiyev, Omid Mirzaei, Amin Kharraz, and Engin Kirda. Preventing server-side request forgery attacks. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1626–1635, 2021.

[30] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

[31] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench verified leaderboard, 2025. URL `https://www.swebench.com/#verified`.

[32] Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

[33] Koray Kavukcuoglu. Gemini 2.0 is now available to everyone, 2025. URL `https://blog.google/technology/google-deepmind/gemini-model-updates-february-2025/`.

[34] Hynek Kydlíček and Greg Gandenberger. Math-verify, 2025. URL `https://github.com/huggingface/Math-Verify`.

[35] Robert Tjarko Lange, Aaditya Prasad, Qi Sun, Maxence Faldor, Yujin Tang, and David Ha. The ai cuda engineer: Agentic cuda kernel discovery, optimization and composition. 2025.

[36] Maurizio Leotta, Boni García, Filippo Ricca, and Jim Whitehead. Challenges of end-to-end testing with selenium webdriver and how to face them: A survey. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 339–350. IEEE, 2023.

[37] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

[38] Zhen Li, Xiaohan Xu, Tao Shen, Can Xu, Jia-Chen Gu, Yuxuan Lai, Chongyang Tao, and Shuai Ma. Leveraging large language models for nlg evaluation: Advances and challenges. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16028–16045, 2024.

[39] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[40] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36:23813–23825, 2023.

[41] Timothy R McIntosh, Teo Susnjak, Nalin Arachchilage, Tong Liu, Paul Watters, and Malka N Halgamuge. Inadequacies of large language model benchmarks in the era of generative artificial intelligence. *arXiv preprint arXiv:2402.09880*, 2024.

[42] Meta. Introducing llama 3.1: Our most capable models to date, 2024. URL `https://ai.meta.com/blog/meta-llama-3-1/`.

[43] Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024. URL `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`.

[44] METR. Evaluating language-model agents on realistic autonomous tasks, 2023. URL `https://metr.org/blog/2023-08-01-new-report/`.

[45] METR. Example protocol for running an ai agent evaluation, 2024. URL `https://metr.github.io/autonomy-evals-guide/example-protocol/`.

[46] METR. Measuring automated kernel engineering, 2025. URL `https://metr.org/blog/2025-02-14-measuring-automated-kernel-engineering`.

[47] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[48] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025.

[49] Mistral-AI. Mixtral large 2, 2024. URL https://mistral.ai/news/mistral-large-2407.

[50] OpenAI. Preparedness framework (beta), 2023. URL https://cdn.openai.com/openai-preparedness-framework-beta.pdf.

[51] OpenAI. Gpt-4o system card, 2024. URL https://openai.com/index/gpt-4o-system-card/.

[52] OpenAI. Openai o1 system card, 2024. URL https://openai.com/index/openai-o1-system-card/.

[53] OpenAI. Openai o1-mini, 2024. URL https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/.

[54] OpenAI. Gpt-4o mini: advancing cost-efficient intelligence, 2025. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[55] OpenAI. Computer-user agent, 2025. URL https://openai.com/index/computer-using-agent/.

[56] OpenAI. Introducing deep research, 2025. URL https://openai.com/index/introducing-deep-research/.

[57] OpenAI. Introducing gpt-4.5, 2025. URL https://openai.com/index/introducing-gpt-4-5/.

[58] OpenAI. Openai o3-mini, 2025. URL https://openai.com/index/openai-o3-mini/.

[59] Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? In *ICLR 2025 Third Workshop on Deep Learning for Code (Best paper award)*, 2025.

[60] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[61] Owain Parry, Gregory M Kapfhammer, Michael Hilton, and Phil McMinn. A survey of flaky tests. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1):1–74, 2021.

[62] Mohammadreza Pourreza and Davood Rafiei. Evaluating cross-domain text-to-sql models and benchmarks. *arXiv preprint arXiv:2310.18538*, 2023.

[63] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025.

[64] Inioluwa Deborah Raji, Emily Denton, Emily M Bender, Alex Hanna, and Amandalynne Paullada. Ai and the everything in the whole wide world benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[65] Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel Kochenderfer. Betterbench: Assessing ai benchmarks, uncovering issues, and establishing best practices. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[66] Filippo Ricca and Paolo Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 25–34. IEEE, 2001.

[67] Per Runeson. A survey of unit testing practices. *IEEE software*, 23(4):22–29, 2006.

[68] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[69] US AI Safety Institute Technical Staff. Strengthening ai agent hijacking evaluations, 2025. URL https://www.nist.gov/news-events/news/2025/01/technical-blog-strengthening-ai-agent-hijacking-evaluations.

[70] Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. Inference scaling flaws: The limits of llm resampling with imperfect verifiers. *arXiv preprint arXiv:2411.17501*, 2024.

[71] Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

[72] Wei-Tek Tsai, Xiaoying Bai, Ray Paul, Weiguang Shao, and Vishal Agarwal. End-to-end integration testing design. In *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, pages 166–171. IEEE, 2001.

[73] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. From imagenet to image classification: Contextualizing progress on benchmarks. In *International Conference on Machine Learning*, pages 9625–9635. PMLR, 2020.

[74] Bertie Vidgen, Adarsh Agrawal, Ahmed M Ahmed, Victor Akinwande, Namir Al-Nuaimi, Najla Alfaraj, Elie Alhajjar, Lora Aroyo, Trupti Bavalatti, Max Bartolo, et al. Introducing v0. 5 of the ai safety benchmark from mlcommons. *arXiv preprint arXiv:2404.12241*, 2024.

[75] Leandro Von Werra, Lewis Tunstall, Abhishek Thakur, Sasha Luccioni, Tristan Thrush, Aleksandra Piktus, Felix Marty, Nazneen Rajani, Victor Mustar, and Helen Ngo. Evaluate & evaluation on the hub: Better best practices for data and model measurements. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 128–136, 2022.

[76] Arthur Henry Watson, Dolores R Wallace, and Thomas J McCabe. *Structured testing: A testing methodology using the cyclomatic complexity metric*, volume 500. US Department of Commerce, Technology Administration, National Institute of . . . , 1996.

[77] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

[78] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

[79] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

[80] Niklas Wretblad, Fredrik Gordh Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. Understanding the effects of noise in text-to-sql: an examination of the bird-bench benchmark. *arXiv preprint arXiv:2402.12243*, 2024.

[81] xAI. Grok 2 beta release, 2024. URL https://x.ai/news/grok-2.

[82] xAI. Grok 3 beta — the age of reasoning agents, 2024. URL https://x.ai/news/grok-3.

[83] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[84] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[85] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[86] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

[87] Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. Utboost: Rigorous evaluation of coding agents on swe-bench. *ACL*, 2025.

[88] Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. Evaluating large language models at evaluating instruction following. *ICLR*, 2024.

[89] Andy K Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. *arXiv preprint arXiv:2408.08926*, 2024.

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

[91] Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. Don't make your llm an evaluation benchmark cheater. *arXiv preprint arXiv:2311.01964*, 2023.

[92] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. X-webarena-leaderboard, 2024. URL `https://docs.google.com/spreadsheets/d/1M801lEpBbKSNwP-vDBkC_pF7LdyGU1f_ufZb_NWNBZQ/edit?gid=0#gid=0`.

[93] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.

[94] Hong Zhu, Patrick AV Hall, and John HR May. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427, 1997.

[95] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang. Fuzzing: a survey for roadmap. *ACM Computing Surveys (CSUR)*, 54(11s):1–36, 2022.

[96] Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, et al. Cve-bench: A benchmark for ai agents' ability to exploit real-world web application vulnerabilities. *arXiv preprint arXiv:2503.17332*, 2025.

[97] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

[98] Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. Can large language models transform computational social science? *Computational Linguistics*, 50 (1):237–291, 2024.

A Limitation and Impact Statement

Limitation. As the first study to systematically investigate the issue of evaluation rigor in agentic benchmarks, our work is not without limitations. First, our analysis covered only 17 agentic benchmarks that are used by top AI providers between January 2024 and March 2025. We did not analyze benchmarks outside this time frame. Therefore, our findings may not necessarily include all relevant evaluation practices. Consequently, it is possible that we have not presented an exhaustive checklist for ensuring evaluation rigor. Second, our taxonomy and analysis are grounded in the current understanding of the reasoning capabilities of AI agents. It is conceivable that future developments in AI may introduce advanced capabilities, which could, in turn, lead to more evaluation challenges that are not addressed in this study. Finally, our findings only reflect the state the analyzed benchmark at the time of writing. Future revisions of these benchmarks may yield different results. Therefore, our conclusions may not fully apply to subsequent versions.

Broader Impact. Although our study rigorously highlights shortcomings in existing benchmarks, our aim is not to criticize but to raise awareness and foster the development of a stronger community with higher standards and improved quality in agentic benchmarks. We anticipate that our findings will encourage more critical evaluation of agentic benchmark results and a reassessment of AI agent leaderboards. We believe these contributions will lead to a deeper and more accurate understanding of AI agent capabilities, resulting in positive societal impact.

B Details of Benchmark Collection and Selection

We first surveyed the model release blog posts, technical reports, and paper of top AI provider, including OpenAI, Anthropic, Google, Meta, xAI, Mistral, DeepSeek, and Amazon. Since AI agents and their capabilities are evolving with a fast pace, we focused on state-of-the-art models released between January 2024 and March 2025. Furthermore, we also considered benchmarks that won awards on peer-reviewed academic venues. As shown in Table 2, we identified 78 benchmarks.

Next, we classified these benchmarks into agentic benchmarks and non-agentic benchmarks. An agentic benchmark mush involve tasks that require multistep reasoning or command execution, which excludes fact-seeking questions, such as simpleQA [77], straightforward question-answer (QA) datasets, such as MMMLU [27], and straightforward programming tasks, such as MBPP [8] and HumanEval [13]. As shown in Table 2, we collected 25 agentic benchmarks.

Finally, we categorize these agentic benchmarks based on their evaluated capabilities, evaluation methods, and open-source availability (Table 3). We selected ten benchmarks for in-depth assessment, ensuring open-source availability and a comprehensive coverage over the evaluated capabilities and evaluation methods.

Table 2: Benchmarks used by major AI providers between 1 January 2024 and 18 March 2025. Duplicate benchmarks are listed only once.

Benchmark	Used by	Source	Agentic
SimpleQA	OpenAI	Introducing GPT-4.5 [57]	✗
SWE-Bench Verified	OpenAI	Introducing GPT-4.5 [57]	✔
GPQA	OpenAI	Introducing GPT-4.5 [57]	✗
AIME '24	OpenAI	Introducing GPT-4.5 [57]	✗
MMMLU	OpenAI	Introducing GPT-4.5 [57]	✗
MMMU	OpenAI	Introducing GPT-4.5 [57]	✗
SWE-Lancer Diamond	OpenAI	Introducing GPT-4.5 [57]	✔
GAIA	OpenAI	Introducing deep research [56]	✔
FrontierMath	OpenAI	OpenAI o3-mini [58]	✔
Codeforces	OpenAI	OpenAI o3-mini [58]	✔
LiveBench Coding	OpenAI	OpenAI o3-mini [58]	✔
MMLU	OpenAI	OpenAI o3-mini [58]	✗
Math	OpenAI	OpenAI o3-mini [58]	✗

Continued on next page

Table 2: Benchmarks used by major AI providers between 1 January 2024 and 18 March 2025. Duplicate benchmarks are listed only once. (Continued)

MGSM	OpenAI	OpenAI o3-mini [58]	✗
OSWorld	OpenAI	Computer-Using Agent [55]	✔
WebArena	OpenAI	Computer-Using Agent [55]	✔
WebVoyager	OpenAI	Computer-Using Agent [55]	✔
HumanEval	OpenAI	OpenAI o1-mini [53]	✗
MATH-500	OpenAI	OpenAI o1-mini [53]	✔
DROP	OpenAI	GPT-4o mini: advancing cost-efficient intelligence [54]	✗
MathVista	OpenAI	GPT-4o mini: advancing cost-efficient intelligence [54]	✔
RE-Bench	OpenAI	GPT-4o System Card [51]	✔
MedQA	OpenAI	GPT-4o System Card [51]	✗
MedMCQA	OpenAI	GPT-4o System Card [51]	✗
ProtocolQA	OpenAI	OpenAI o1 System Card [52]	✗
BioLP-Bench	OpenAI	OpenAI o1 System Card [52]	✗
MLE-bench	OpenAI	OpenAI o1 System Card [52]	✔
Tau-bench	Anthropic	Claude 3.7 Sonnet and Claude Code [6]	✔
BIG-Bench-Hard	Anthropic	Claude 3.5 Sonnet [5]	✗
IF-Eval	Deepseek	Introducing DeepSeek-V3 [15]	✗
FRAMES	Deepseek	Introducing DeepSeek-V3 [15]	✗
LongBench v2	Deepseek	Introducing DeepSeek-V3 [15]	✗
Aider-Edit	Deepseek	Introducing DeepSeek-V3 [15]	✔
Aider-Polyglot	Deepseek	Introducing DeepSeek-V3 [15]	✔
CNMO 2024	Deepseek	Introducing DeepSeek-V3 [15]	✔
CLUEWSC	Deepseek	Introducing DeepSeek-V3 [15]	✗
C-Eval	Deepseek	Introducing DeepSeek-V3 [15]	✗
C-SimpleQA	Deepseek	Introducing DeepSeek-V3 [15]	✗
LOFT (128k)	xAI	Grok 3 Beta — The Age of Reasoning Agents [82]	✗
EgoSchema	xAI	Grok 3 Beta — The Age of Reasoning Agents [82]	✗
DocVQA	xAI	Grok-2 Beta Release [81]	✗
ChartQA	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
AI2 Diagram	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
VQAv2	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
Open-rewrite eval	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
TLDR9+	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
BFCL V2	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
Nexus	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
ARC Challenge	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
Hellaswag	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
InfiniteBench	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
NIH/Multi-needle	Meta	Llama 3.2: Revolutionizing edge AI and vision with open, customizable models [43]	✗
ZeroScrolls	Meta	Introducing Llama 3.1: Our most capable models to date [42]	✗
Bird-Bench	Google	Gemini 2.0 is now available to everyone [33]	✔
FACTS Grounding	Google	Gemini 2.0 is now available to everyone [33]	✗
HiddenMath	Google	Gemini 2.0 is now available to everyone [33]	✔
MRCR	Google	Gemini 2.0 is now available to everyone [33]	✗
CoVoST2	Google	Gemini 2.0 is now available to everyone [33]	✗
MBPP	Mistral	Mistral Large 2 [49]	✗
MT-Bench	Mistral	Mistral Large 2 [49]	✗
Wild Bench	Mistral	Mistral Large 2 [49]	✗
Arena Hard	Mistral	Mistral Large 2 [49]	✗
BBH	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
ARC-C	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗

Continued on next page

Table 2: Benchmarks used by major AI providers between 1 January 2024 and 18 March 2025. Duplicate benchmarks are listed only once. (Continued)

ChartQA	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
Doc VQA	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
VATEX	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
Text VQA	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
Ego Schema	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
VisualWebBench	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
NN-Mind2Web	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
GroundUI-1K	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
SQuALITY	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
LVBench	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
FinQA	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
CRAG	Amazon	The Amazon Nova family of models: Technical report and model card [4]	✗
Kernel-Bench	DL4C	KernelBench: Can LLMs Write Efficient GPU Kernels? [59]	✔

Table 3: Collected agentic benchmarks. Assessed benchmarks are highlighted in blue.

Benchmark	Evaluated Capability	Evaluation Design
SWE-bench [30]	Software Engineering	Unit Testing
SWE-Lancer [48]	Software Engineering	End-to-end Testing
KernelBench [59]	Software Engineering	Fuzz Testing
BIRD [37]	Software Engineering	End-to-end Testing
Aider-Edit [2]	Software Engineering	Unit Testing
Codeforces [63]	Software Engineering	Unit Testing
LiveBench Coding [78]	Software Engineering	Unit Testing
Aider-Polyglot [3]	Software Engineering	Unit Testing
FrontierMathNo open-source access [22]	Challenging Math Problem-solving	Answer Match
MLE-bench [11]	ML Engineering	Quality Measure
RE-bench [79]	ML Engineering	Quality Measure
τ-bench [86]	Environment Interaction	Substring Matching, State Matching
WebArena [93]	Environment Interaction	Whole String Matching, Substring Matching, LLM-as-a-Judge, State Matching
OSWorld [83]	Environment Interaction	State Matching
WebVoyager [25]	Environment Interaction	LLM-as-a-Judge
Cybench [89]	Cybersecurity	Answer Matching
GAIA [47]	General Assistant	Answer Matching

C Sources of the Checks in ABC

In Table 14, we show the detail construction process of ABC by listing the sources of each check proposed in ABC. We synthesized the insights from the following aspects

1. Our experience of developing agentic benchmarks.
2. Best practices in existing agentic benchmarks (Table 3).
3. Lessons learned from issues of existing agentic benchmarks.
4. Domain-specific suggestions when we apply well-established techniques as evaluation methods.

Table 4: Sources of items in ABC

Question	Existing Best Practice	Lessons Learned	Domain-Specific Suggestions

Continued on next page

Table 4: Sources of items in ABC (Continued)

O.a.1	Mialon et al. [47], Zhou et al. [93]		
O.a.2	Mialon et al. [47], Zhou et al. [93]	Zhou et al. [93]	
O.b.1	Mialon et al. [47]	Zhou et al. [93]	
O.b.2		Yao et al. [86], Zhou et al. [93]	
O.b.3	Zhou et al. [93]	Yao et al. [86]	
O.c.1	He et al. [25]		Ziems et al. [98]
O.d.1	Chowdhury et al. [14]	Jimenez et al. [30], Yu et al. [87]	
O.d.2			Zhu et al. [94]
O.e.1		Ouyang et al. [59]	Zhu et al. [95]
O.e.2		Ouyang et al. [59]	Zhu et al. [95]
O.e.3	METR [46]		
O.f.1			Ricca and Tonella [66]
O.f.2			Parry et al. [61]
O.g.1	Yao et al. [86], Zhou et al. [93], Xie et al. [83]		
O.g.2	Yao et al. [86]	Xie et al. [83]	
O.g.3		Yao et al. [86]	
O.h.1	Mialon et al. [47]	Kydlíček and Gandenberger [34], Lightman et al. [39]	
O.h.2	Glazer et al. [22]		
O.i.1	Chan et al. [11]		
T.1	Miserendino et al. [48], Li et al. [37]		
T.2	Kapoor et al. [32]	Zhou et al. [93]	
T.3	Zhou et al. [93]	Zhu et al. [96]	
T.4	Miserendino et al. [48], Yao et al. [86], Jimenez et al. [30]	Lange et al. [35]	
T.5	Zhang et al. [89]	Miserendino et al. [48]	
T.6		Wretblad et al. [80], Pourreza and Rafiei [62], Li et al. [37]	
T.7	Zhang et al. [89], Zhu et al. [96], [83]		
T.8	Zhang et al. [89], Zhu et al. [96]	Li et al. [37]	
T.9		Lange et al. [35], Miserendino et al. [48]	
R.1	All benchmarks in Table 1.		
R.2	All benchmarks in Table 1 except GAIA.		
R.3	Chan et al. [11], Miserendino et al. [48], [37]		Zhou et al. [91]
R.4	White et al. [78]		
R.5	Kapoor et al. [32]		
R.6	All benchmarks in Table 1.		
R.7	Chan et al. [11], Yao et al. [86]		
R.8	Miserendino et al. [48], Chan et al. [11]		
R.9	Yao et al. [86]		
R.10			Dorner and Hardt [17], Reuel et al. [65]

Continued on next page

Table 4: Sources of items in ABC (Continued)

R.11			Hothorn et al. [28], Dorner and Hardt [17]
R.12	Cao et al. [10], Xie et al. [83], Zhang et al. [89]		
R.13		Yao et al. [86]	

D Assessment Reports

In this section we provide detailed assessment reports for all ten benchmarks. Each report's caption specifies the corresponding paper and codebase evaluated.

Table 5: Assessment Report of SWE-Bench-Lancer (paper, code)

Check	Score	Reason
O.d.1	1	As discussed in Section 1 of the paper, the benchmark uses a set of test cases that are verified for correctness and quality by human experts.
O.d.2	0	The benchmark does not use objective metrics to measure the quality of test cases.
O.f.2	1	As discussed in Section 1, the end-to-end testing is designed to simulate the entire user workflow.
O.f.3	0	The test cases use hard-coded timeouts, which may lead to non-deterministic results if the system is slow or unresponsive.
T.1	1	The package dependencies are specified in the repository of each task.
T.2	1	The benchmark does not require any external APIs.
T.3	1	The benchmark does not require any external APIs.
T.4	1	The benchmark uses docker containers to isolate the environment, and the state is cleared between runs.
T.5	0	The agent can access the file system where the test cases are stored, which may lead to the agent accessing the ground truth information.
T.6	1	The environment setup is static and does not change over time.
T.7	1	The ground-truth test cases are taken from GitHub repositories, which are verified by expert developers.
T.8	1	Each task represents a real-world software issue with a corresponding patch, which are solvable by the agent.
T.9	1	The benchmark uses existing patches as ground truth, which can be considered as an Oracle solver.
T.10	0	The benchmark does not handle the isolation between the agent and test cases properly. The test cases are stored not only in a file system that the agent can access, but also in a ZIP file that agent can read the directory structure and update files.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	1	The benchmark maintains a private test set.
R.4	0	The report does not discuss any measures or plans for consistent update.
R.5	1	Such a relationship is clearly stated in Section 2 of the paper.
R.6	1	As shown in Section 3, the benchmark is designed to evaluate the LLM model.
R.7	1	The benchmark uses end-to-end testing to mitigate grader hacking.
R.8	1	The benchmark discusses the potential impact of grader hacking in Section 1 and Appendix A.7.
R.9	0	The benchmark does not include any quantitative analysis to assess the impact of grader hacking.
R.10	0	The benchmark does not report any metrics about statistical significance.
R.11	0	The benchmark does not provide any guidance on interpreting results with eval flaws.
R.12	0	The benchmark does not report results of non-AI baselines.
R.13	0	The benchmark does not report results of trivial agents.

Table 6: Assessment Report of Bird-Bench (paper, code)

Check	Score	Reason
O.d.1	1	As discussed in Section 3.4 of the paper, the validity of the database is verified by executing the ground-truth query.
O.d.2	0	The paper does not use objective metrics to measure the usefulness and completeness of the database or ground-truth queries.

Continued on next page

Table 6: Assessment Report of Bird-Bench (paper, code) (Continued)

O.f.2	0	The paper does not provide any information about the coverage of the database or ground-truth queries.
O.f.3	1	Executing SQL queries on a database is deterministic, and the paper does not mention any non-deterministic behavior.
T.1	1	The task instruction in Figure 9 specifies the SQL language is SQLite.
T.2	1	No external API is required for the evaluation of the benchmark.
T.3	1	No external API is required for the evaluation of the benchmark.
T.4	0	Database file is neither opened in a read-only mode nor re-initialized between runs. This may lead to unexpected data manipulation by the agent.
T.5	1	Agent cannot access the host file system.
T.6	1	The environment setup is static and does not change over time.
T.7	0	As discussed in Section 3.4 of the paper, the correctness of the query is not fully verified, especially for the SQL queries that two annotators reach a consensus on.
T.8	0	The ambiguity of the SQL queries is not fully verified.
T.9	0	The Benchmark does not include an Oracle solver that can automatically solve all text-to-SQL tasks.
T.10	1	No vulnerabilities are found in the implementation of the benchmark.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	1	The benchmark has a private test set.
R.4	0	The benchmark does not discuss any measures or plans for consistent update.
R.5	1	It is clearly discussed in Section 2 of the paper.
R.6	1	It is clearly discussed in Section 2 of the paper.
R.7	0	No efforts are made to mitigate errors when both annotators make the same mistake.
R.8	0	The paper does not discuss the potential impact of annotation errors.
R.9	0	The paper does not analyze the quantitative impact of annotation errors.
R.10	0	The paper does not report any metrics about statistical significance.
R.11	0	The paper does not provide any guidance on interpreting results with eval flaws.
R.12	1	The paper reports the results of human experts.
R.13	0	The paper does not report the results of any trivial agents.

Table 7: Assessment Report of CyBench (paper, code)

Check	Score	Reason
O.h.1	1	The specific format required for the answer is provided in the task description.
O.h.2	1	The ground truth is complex enough to prevent trivial guessing.
T.1	1	Agents are granted access to using all tools. The versions of tools can be checked by the agent.
T.2	1	The benchmark does not require any external APIs.
T.3	1	The benchmark does not require any external APIs.
T.4	1	The benchmark uses docker containers to isolate the environment, and the state is cleared between runs.
T.5	1	The agent cannot directly access the container which contains the ground truth.
T.6	1	The environment setup is static and does not change over time.
T.7	1	As shown in Section 3.3 of the paper, the ground truth is verified by human.
T.8	1	As shown in Section 3.3 of the paper, each task is verified to be solvable.
T.9	1	As shown in Section 3.3 of the paper, the benchmark includes an Oracle solver that can automatically solve all tasks.
T.10	1	No vulnerabilities are found in the implementation of the benchmark.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	0	The benchmark does not contain measures to prevent data contamination.
R.4	0	The report does not discuss plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 1 of the paper.
R.6	1	As shown in Section 1, the benchmark is designed to evaluate both agent frameworks and LLM models.
R.7	1	Annotation flaws are mitigated by developing verifiable tasks.

Continued on next page

Table 7: Assessment Report of CyBench (paper, code) (Continued)

R.8	1	No unavoidable flaws are identified in the benchmark.
R.9	1	No unavoidable flaws are identified in the benchmark.
R.10	0	The report does not include any metrics about statistical significance.
R.11	1	No evaluation flaws are identified in the benchmark.
R.12	1	Human performance is reported in Section 5 of the paper.
R.13	0	The report does not report results of trivial agents.

Table 8: Assessment Report of SWE-Bench-Verified (paper, code)

Check	Score	Reason
O.d.1	1	Test cases are directly taken from GitHub repositories, and the paper does not mention any verification process.
O.d.2	0	The paper does not use objective metrics to measure quality of test cases.
T.1	1	The versions of package dependencies are specified in the repository.
T.2	1	The benchmark does not require any external APIs.
T.3	1	The benchmark does not require any external APIs.
T.4	1	The benchmark uses docker containers to isolate the environment, and the state is cleared between runs.
T.5	1	The agent cannot access the host file system, and the ground truth is not accessible to the agent.
T.6	1	The environment setup is static and does not change over time.
T.7	1	The ground-truth patches are taken from GitHub repositories, which is verified by expert developers.
T.8	1	Each task represents a real-world GitHub issue and a corresponding pull request, which are solvable by the agent.
T.9	1	Pull requests from GitHub are used as ground truth, which can be considered as an Oracle solver.
T.10	1	No vulnerabilities are found in the implementation of the benchmark, and the evaluation process is secure.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	0	The benchmark does not discuss measures to prevent data contamination.
R.4	0	The benchmark does not discuss plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 2 of the paper.
R.6	1	The benchmark is designed to evaluate both the model and the agent framework, as discussed in Section 5 of the paper.
R.7	0	The benchmark does not discuss any efforts to prevent, identify, and correct flaws.
R.8	0	The benchmark does not discuss the potential impact of unavoidable flaws.
R.9	0	The benchmark does not include quantitative analysis to assess the impact of unavoidable flaws.
R.10	0	The report does not include any metrics about statistical significance.
R.11	0	The benchmark does not provide any guidance on interpreting results with eval flaws.
R.12	0	The benchmark does not report results of non-AI baselines.
R.13	0	The benchmark does not report results of trivial agents.

Table 9: Assessment Report of tau-Bench (paper, code)

Check	Score	Reason
O.a.1	1	The benchmark uses minimal expressions for substring matching, which is robust to variations in the input.
O.a.2	1	The benchmark uses minimal expressions for substring matching, which is robust to redundant words in the input.
O.b.1	0	The benchmark does not specify how negation modifiers are handled, which may lead to incorrect evaluations.
O.b.2	0	The benchmark does not specify how it handles systematic listing of all possible answers, which may lead to incorrect evaluations.
O.b.3	0	A part of tasks has empty ground truth, which may lead to guessing.
O.g.1	1	The database after successful completion of a task is unique and includes all states.
O.g.2	1	The state of the database is the only environment state, and it is checked for both relevant and irrelevant parts.
O.g.3	0	A part of tasks has empty ground truth, which may lead to trivial state modifications.
T.1	1	The benchmark does not use external tools.

Continued on next page

Table 9: Assessment Report of tau-Bench (paper, code) (Continued)

T.2	1	The benchmark does not use external APIs.
T.3	1	The benchmark does not use external APIs.
T.4	1	Residual data or state are fully cleared between runs by re-initializing the database.
T.5	1	Agents has no access to the file system.
T.6	1	The environment setup is static and does not change over time.
T.7	1	As shown in Section 4 of the paper, the ground truth is manually verified.
T.8	1	As shown in Section 4 of the paper, each task is verified to be solvable by the agent.
T.9	1	The benchmark provides a reference task solution that can be used as an Oracle solver.
T.10	1	No vulnerabilities are found in the implementation of the benchmark, and the evaluation process is secure.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	0	The benchmark does not discuss measures to prevent data contamination.
R.4	0	The report does not discuss plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 3 of the paper.
R.6	1	As discussed in Section 5 of the paper, the benchmark is designed to evaluate both the model and the agent framework.
R.7	1	Appendix A of the paper shows the efforts taken to detect annotation errors.
R.8	1	Section 6 discusses the potential impact of unavoidable flaws, although these discussions are not sufficient.
R.9	0	The report does not include quantitative analysis to assess the impact of unavoidable flaws.
R.10	0	The report does not include any metrics about statistical significance.
R.11	0	The report does not provide any guidance on interpreting results with eval flaws.
R.12	0	The report does not report results of non-AI baselines.
R.13	0	The report does not report results of trivial agents.

Table 10: Assessment Report of MLE-Bench (paper, code)

Check	Score	Reason
O.I.1	1	As described in Section 2.2, the benchmark uses leaderboard positions as a metric, which is not easily exploitable.
T.1	0	The prompt does not specify the versions of important tools, such as Python and Pytorch.
T.2	1	The benchmark does not require any external APIs, and all required tools are accessible to the agent.
T.3	1	The benchmark does not require any external APIs, and the evaluation process does not depend on any external resources.
T.4	1	There are no residual data or state between runs, as the evaluation is performed in a clean environment.
T.5	1	The submission process is isolated from the agent's environment, and the agent cannot access any ground truth information.
T.6	1	The environment setup is static and does not change over time.
T.7	1	The benchmark uses ground truth data from Kaggle, which is a widely used and reliable source for benchmark datasets.
T.8	1	The benchmark uses previous challenges from Kaggle, which are proven to be solvable with ML algorithms.
T.9	1	Any solution on Kaggle can be considered an Oracle solver.
T.10	1	No vulnerabilities are found in the implementation of the benchmark, and the evaluation process is secure.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	1	The benchmark design experiments to measure data contamination and agent plagiarism.
R.4	1	Future plan on regularly update the benchmark with new Kaggle challenges is discussed in Section 6
R.5	1	Such a relationship is clearly stated in Section 2.
R.6	1	As shown in Section 3, the benchmark is designed to evaluate both the model and the agent framework.
R.7	1	The paper discusses the efforts taken to detect cheating in Appendix A.5.
R.8	1	The paper discusses the potential impact of unavoidable flaws in Section 4.
R.9	1	The paper includes quantitative analysis to assess the impact of unavoidable flaws in Appendix A.5.
R.10	1	The paper reports metrics about statistical significance in Section 3.3.
R.11	1	No significant flaws are found in the evaluation process.

Continued on next page

Table 10: Assessment Report of MLE-Bench (paper, code) (Continued)

R.12	1	The benchmark directly compares the performance of agents with human experts in the Kaggle challenge submissions.
R.13	0	The benchmark does not report results of trivial agents.

Table 11: Assessment Report of WebArena (paper, code)

Check	Score	Reason
O.a.1	1	As discussed in Section 3.2 of the paper, the benchmark expects the response to follow a standardized format, which is robust to variations in the input.
O.a.2	1	As discussed in Section 3.2 of the paper, the benchmark expects the response to follow a standardized format, which is robust to redundant words in the input.
O.b.1	0	The benchmark does not handle negation modifiers, which may lead to incorrect evaluations.
O.b.2	0	The benchmark does not specify how it handles systematic listing of all possible answers, which may lead to incorrect evaluations.
O.b.3	0	The ground truth is NULL for a part of tasks, which may lead to guessing.
O.c.1	1	The accuracy of the judge is quantitatively evaluated in Appendix A.8 of the paper.
O.c.2	0	The benchmark does not handle adversarial inputs and reward hacking in LLM-as-a-Judge, which may lead to incorrect evaluations.
O.g.1	1	The ground truth includes all states achievable after success, as discussed in Section 3.2 of the paper.
O.g.2	0	The state check only considers relevant states (e.g., achieved by using a locator as discussed in Section 3.2), which may lead to incorrect evaluations.
O.g.3	1	As demonstrated in Section 3.2 of the paper, the ground truth is a modification of the underlying database, which is complex enough to prevent trivial state modifications.
T.1	1	The benchmark does not use tools that require version specification.
T.2	0	The benchmark requires an external API (e.g., a clone of Reddit website) that is not can be inaccessible to agents during evaluation due to rate limit.
T.3	0	The evaluation process does not handle errors appropriately if the API becomes inaccessible, which may lead to incorrect evaluations.
T.4	1	The benchmark uses docker containers to isolate the environment, and the state is cleared between runs.
T.5	1	The agent has no access to the file system where the ground truth is stored.
T.6	1	The environment setup is static and does not change over time.
T.7	0	As mentioned in Section 3.2, the ground truth is annotated by two human annotators. However, there isn't a mechanism to verify or guarantee the correctness of the annotations.
T.8	0	The ambiguity of the tasks is not fully verified or tested, which may lead to incorrect evaluations.
T.9	0	The benchmark does not include an Oracle solver that can automatically solve all tasks.
T.10	0	A do-nothing agent can pass 4.4**R.1**
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	0	The benchmark does not discuss measures to prevent data contamination.
R.4	0	The benchmark does not discuss plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 2.1 of the paper.
R.6	1	As shown in Section 5, the benchmark is designed to evaluate LLM models.
R.7	1	Efforts to evaluate LLM-as-a-Judge are discussed in Appendix A.8 of the paper.
R.8	0	The report does not discuss the potential impact of unavoidable flaws.
R.9	0	The report does not include quantitative analysis to assess the impact of unavoidable flaws.
R.10	0	The report does not include any metrics about statistical significance.
R.11	0	The report does not provide any guidance on interpreting results with eval flaws.
R.12	1	The human performance is reported in appendix A.5.
R.13	0	The report does not report results of trivial agents.

T.10	1	The benchmark is open-sou

Table 12: Assessment Report of GAIA (paper, code)

Check	Score	Reason

Continued on next page

Table 12: Assessment Report of GAIA (paper, code) (Continued)

O.h.1	1	As discussed in Section 3.2 of the paper, the specific format required for the answer is provided in the task description.
O.h.2	1	The ground truth is complex enough to prevent trivial guessing.
T.1	0	The version of tools (e.g., Python and website) is not specified in the paper.
T.2	0	The rate limit of the API is not specified in the paper, which may lead to incorrect evaluations.
T.3	0	The benchmark does not provide a reference harness for handling errors, which may lead to inconsistent evaluations across different users.
T.4	1	The benchmark does not modify the environment state.
T.5	1	Agents have no access to the ground truth information.
T.6	1	The environment setup is static and does not change over time.
T.7	1	The data annotation process contains a verification step, as discussed in Section 3.4 of the paper.
T.8	1	The data annotation process contains a verification step, as discussed in Section 3.4 of the paper.
T.9	0	The benchmark does not include an Oracle solver that can automatically solve all tasks.
T.10	1	No vulnerabilities are found in the implementation of the benchmark.
R.1	1	The benchmark is open-sourced and available on HuggingFace.
R.2	0	The benchmark does not provide an open-source evaluation harness for users.
R.3	0	The benchmark does not contain measures to prevent data contamination.
R.4	0	The report does not discuss plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 3 of the paper.
R.6	1	As discussed in Section 3 of the paper, the benchmark is designed to evaluate LLM models.
R.7	1	Section 5 of the paper discusses the efforts, including comparing evaluation with or without human in the loop.
R.8	1	Section 6 discusses the potential impact of unavoidable flaws, such as a wrong reasoning trace resulting in a correct answer.
R.9	0	The report does not include quantitative analysis to assess the impact of unavoidable flaws.
R.10	0	The report does not include any metrics about statistical significance.
R.11	0	The report does not provide any guidance on interpreting results with eval flaws.
R.12	1	Human performance is reported in Section 4 of the paper.
R.13	1	The report includes results of search engine, which can be considered a trivial agent.

Table 13: Assessment Report of OSWorld (paper, code)

Check	Score	Reason
O.g.1	1	As discussed in Section 3.2 of the paper, the ground truth is verified to include all states that can be achieved after a successful task completion.
O.g.2	0	The state check only verifies the relevant states for the tasks. Agents can potentially perform extra harmful actions that are not checked by the ground truth.
O.g.3	1	As demonstrated in Section 3.2 of the paper, the ground truth involves complex state changes to a software or website.
T.1	1	No external tools are used in the benchmark. Versions of the environment are clearly specified in the README file of the repository.
T.2	1	No external APIs are used in the benchmark.
T.3	1	No external APIs are used in the benchmark.
T.4	1	The benchmark uses virtual machines to run the tasks, which ensures that all residual data or state are cleared between runs.
T.5	1	Agents and ground truth are isolated from each other via virtual machines.
T.6	0	The benchmark checks for HTML selectors (like class names or page titles) on live web pages.
T.7	1	As discussed in Section 3.2 of the paper, the ground truth is verified for correctness by human experts.
T.8	1	As discussed in Section 3.2 of the paper, each task is verified to be solvable by human experts.
T.9	0	The benchmark does not include an Oracle solver that can automatically solve all tasks.
T.10	1	No vulnerabilities are present in the implementation of the benchmark.
R.1	1	The benchmark is fully open-sourced, as the code is available on GitHub.
R.2	1	The benchmark offers an open-source evaluation harness for users.
R.3	0	The benchmark does not include measures to prevent data contamination.

Continued on next page

Table 13: Assessment Report of OSWorld (paper, code) (Continued)

R.4	0	The report does not include measures or plans to consistently update tasks over time.
R.5	1	Such a relationship is clearly stated in Section 2 of the paper.
R.6	1	As discussed in Section 2 of the paper, the evaluation subject is agent frameworks.
R.7	1	As discussed in Section 3.2 of the paper, the benchmark uses additional manual verification steps to prevent, identify, and correct flaws.
R.8	0	Safety issues of agents are discussed in Section 7 of the paper.
R.9	0	No quantitative analysis to assess the impact of unavoidable flaws is included in the report.
R.10	0	The report does not include metrics about statistical significance.
R.11	0	The report does not provide guidance on interpreting results with eval flaws.
R.12	1	Human performance is reported in Section 3.4 of the paper.
R.13	0	The report does not include results of trivial agents.

Table 14: Assessment Report of KernelBench (paper, code)

Check	Score	Reason
O.e.1	0	The fuzzer does not address potential edge cases, such as empty inputs.
O.e.2	0	Although the data type is specified, the fuzzer does not test different memory layouts, such as tensors with non-contiguous memory layouts.
O.e.3	0	The fuzzer uses uniform sampling to generate inputs, which may not be sensitive to the code under testing. For example, the fuzzer may not generate positive inputs that trigger the ‘relu‘ function in the ‘torch‘ library.
T.1	0	The CUDA version is not specified in the default prompt.
T.2	1	External APIs are not required for the evaluation of the benchmark.
T.3	1	External APIs are not required for the evaluation of the benchmark.
T.4	1	Kernels are evaluated in separate processes, and the state is cleared between runs.
T.5	0	The ground-truth kernel is executed first and in the same process as the agent. This may lead to the agent accessing the ground-truth results by accessing out-of-bound memory.
T.6	1	The environment setup is static and does not change over time.
T.7	1	The ground-truth kernel is provided by PyTorch, which is a widely used library for deep learning.
T.8	1	The implementation from PyTorch is a proof of concept.
T.9	1	The Oracle solver is PyTorch implementation.
T.10	1	No vulnerabilities are found in the implementation of the benchmark.
R.1	1	The benchmark is open-sourced and available on GitHub.
R.2	1	The benchmark provides an open-source evaluation harness for users.
R.3	0	The benchmark does not discuss measures to prevent data contamination.
R.4	0	The benchmark does not discuss plans to consistently update tasks over time.
R.5	1	Section 3 clearly states such a relationship.
R.6	1	Section 5 clearly states that the evaluation subjective of the benchmark is LLM models.
R.7	1	Appendix B.2 describes the efforts taken to prevent, identify, and correct flaws, although these efforts are not sufficient.
R.8	1	Appendix B.2 includes qualitative discussions of the potential impact of unavoidable flaws, although these discussions are not sufficient.
R.9	1	Appendix B.2 includes quantitative analysis to assess the impact of unavoidable flaws, although these analyses are not sufficient.
R.10	0	The benchmark does not report any metrics about statistical significance.
R.11	0	The benchmark does not provide any guidance on interpreting results with eval flaws.
R.12	0	The benchmark does not report results of non-AI baselines.
R.13	0	The benchmark does not report results of trivial agents.

E Case Study

We present case study of specific issues we identified. For each study, we use an Intel E5-2630 CPU with 128 GB RAM and optionally 1 NVIDIA H100 80GB for GPU-required experiments. We release our code at `https://github.com/uiuc-kang-lab/agentic-benchmarks`.

E.1 SWE-bench

Benchmark Overview. SWE-bench is a benchmark for evaluating the ability of AI agents to resolve real-world GitHub issues. Given the issue description and a summary of the codebase, agents are tasked with generating a patch that resolves the issue. Each generated patch is evaluated via existing unit tests in the GitHub repository.

Identified Issue. SWE-bench uses manually written unit tests to evaluate the correctness of a generated code patch. As illustrated in prior work, UTBoost [87], unit tests can lead to many false positives, due to the insufficiency of test cases.

Example. The Python package `seaborn` has an issue in handling missing values in the inputs `x` and `y` when computing polynomial fits using `PolyFit()`. Unfortunately, the unit test case for `PolyFit()` only considers the scenarios when both `x` and `y` have missing values:

```
def test_missing_data(self, df):
    groupby = GroupBy([ "group" ])
    df.iloc[5:10] = np.nan
    res1 = PolyFit()( df[[ "x", "y" ]], groupby, "x", {})
    res2 = PolyFit()( df[[ "x" , "y" ]].dropna (), groupby, "x", {})
    assert_frame_equal( res1, res2 )
```

This insufficient test case for `PolyFit()` leads to the following incorrect patch for `PolyFit()` being evaluated as correct. This patch is generated by IBM SWE-1.0.

```
def _fit_predict(self, data) :
    y = data ["y"].dropna()
    x = data ["x"].dropna()
    if x.shape[0] != y.shape[0]:
        raise ValueError("x and y must have the same number of non-
    missing values")
    if x.nunique() <= self.order :
        # TODO warn ?
    xx = yy = []
```

Qualitative Results. As reported in prior work [87], agents can pass evaluations without addressing the GitHub issues for 5.3% and 7.7% of tasks in the Verified and Lite partitions, respectively. These tasks lead to 40.9% and 24.4% changes in the leaderboard for the Verified and Lite partitions, respectively. Furthermore, these tasks causes 2.3% and 1.6% overestimation of agent performance for the Verified and Lite partitions, respectively.

E.2 τ-bench

Benchmark Overview. τ-bench is for evaluation AI agents capability to interact with human users and follow domain-specific rules [86]. Given a domain-specific policy, the AI agent is tasked to interact with human users and answer user queries.

Identified Issue. τ-bench evaluates the agents' actions based on whether the database state is correct and optionally whether the agents' responses contain required text. Therefore, on tasks that do not change the database state and do not have required texts, agents can get positive evaluation results by doing nothing. On tasks that do not change the database state and has a trivial required text, such as "4", agents can get positive evaluation results by returning random responses or all the data.

Example. A task in τ-bench requires agent to process a flight cancellation and refund request. An AI agent is supposed to check the detail of the booked flight ticket for the user in the database and deny the user request if the ticket is non-refundable. This task has no required output. Therefore, as long as the data state does not change, the agent will obtain a positive evaluation result. In this case, an agent that does nothing can also have a positive evaluation result.

Qualitative Results. A do-nothing agent that returns immediately can achieve a 38% and 6.0% pass^k or pass@k for any k for Airline and Retail partitions, respectively. A spamming agent that

outputs all the data can achieve a 40% and 9.6% pass^k or pass@k for any k for Airline and Retail partitions, respectively.

E.3 BIRD

Benchmark Overview. BIRD is for evaluating the capability of agents to write SQL queries [37]. Given a query description in natural language, the agent needs to translate it into a SQL query.

Identified Issue. BIRD evaluates agent by comparing the execution results of the ground truth query with the generated query. However, due to the ambiguity of the query description, there can be multiple correct queries for the same natural language description.

Example. A task in BIRD asks the agent to write a SQL query that can answer the question: "What are the name, independence year, and surface area of the country with the smallest population?" There can be two correct SQL queries:

```
-- Query 1
SELECT Name, SurfaceArea, IndepYear FROM country
WHERE Population = (SELECT min(Population) FROM country)
-- Query 2
SELECT Name, SurfaceArea, IndepYear FROM country
ORDER BY Population LIMIT 1
```

Query 1 outputs all the country with the smallest population, while Query 2 outputs one of the country with the smallest population. Although the output of two queries are different, they both answer the question.

E.4 SWE-Lancer

Benchmark Overview. SWE-Lancer is for evaluating the capability of AI agents to independently implement features and fix bugs. [48] Given a task description, agent needs to use Python scripts to interact with the file system and modify codebase.

Identified Issue. SWE-Lancer uses end-to-end testing to evaluate the correctness of agents' implementation. Although the test cases are stored in a password-protected `.zip` file, reading the directory structure and updating files within the `.zip` file do not require a password. Therefore, an agent can easily locate the test cases and replace them with a naive one, such as "`assert 1==1`".

Qualitative Results. An agent that overwrites the test cases in the `.zip` file can achieve a 100% resolve rate without completing the software engineering tasks.

E.5 WebArena

Benchmark Overview. WebArena is for evaluating the capability of agents to interact with the web [93]. Given a user request, the AI agent need either retrieve the required information or fill the given data into the web form correctly.

Identified Issue. WebArena uses exact string matching, substring matching, and LLM-as-a-Judge to evaluate agents. Its strategy of exact string matching cannot handle alternative expressions and phrase modifiers, while the substring matching is vulnerable to exhaustive enumeration of the content on the website. Additionally, LLM-as-a-Judge can produce unreliable results.

Example. In WebArena, there is a user query that asks "What is the duration required to first walk from Massachusetts Institute of Technology to Harvard University, and then drive to Boston Logan International Airport?" The ground truth answer for this question is 63 minutes. However, the agent searched the web and output the final answer: "The duration required to first walk from Massachusetts Institute of Technology to Harvard University is 45 minutes, and then drive to Boston Logan International Airport is 8 minutes." The answer of agent gives the duration of 45+8=53

minutes, which is different from the ground truth answer. However, the LLM judge considers the agent's answer as correct.

E.6 KernelBench

Benchmark Overview. KernelBench is for evaluating the capability of agents to write correct and efficient GPU kernels [59]. Given the task instruction and the original PyTorch code, agents need to write PyTorch code containing an inline implementation of the kernel that is functionally correct and more efficient.

Identified Issue 1. KernelBench uses randomly generated inputs (i.e., fuzzing) to test the correctness of generated GPU kernels. However, we find the tested functions in a subset of tasks are not sensitive to uniform random inputs, such as `mean(softmax(x))` and `relu(x-2)`.

Identified Issue 2. In the evaluation implementation, KernelBench first runs the ground truth kernel and then runs the generated kernel subsequently. As reported in prior work [35], agents can potentially cheat by generating a program that extracts the execution results of the ground truth kernel.

Identified Issue 3. The fuzzer designed in KernelBench fails to address potential inputs with different memory layouts (e.g., non-contiguous tensors), tensor shapes, and hardware environment. In the following code snippet, we demonstrate an incorrect kernel function due to improper use of threads, which were graded as correct in KernelBench. In Line 46, the kernel function accesses parallel execution results in `s_sum` with index from `tid` to `nthread`. However, when `nthread > normalized_size`, this will lead to out-of-bound access into uninitialized memory. Namely, a thread-safe guard is required here.

```
#include ...

template <typename scalar_t>
__global__ void layernorm_forward_kernel_opt(
    const scalar_t* __restrict__ input,
    const scalar_t* __restrict__ weight,
    const scalar_t* __restrict__ bias,
    const float eps,
    scalar_t* __restrict__ output,
    const int normalized_size) {

  // Each block processes one outer instance.
  int instance_idx = blockIdx.x;

  // Use 2D thread indexing to cover the normalized dimension flexibly.
  int tid = threadIdx.y * blockDim.x + threadIdx.x;
  int nthreads = blockDim.x * blockDim.y;

  // Pointers to the start of this instance's data.
  const scalar_t* __restrict__ in_ptr = input + instance_idx * normalized_size;
  scalar_t* __restrict__ out_ptr = output + instance_idx * normalized_size;

  using accscalar_t = at::acc_type<scalar_t, true>;

  // Each thread computes a partial sum and sum of squares over a strided range.
  accscalar_t local_sum = 0;
  accscalar_t local_sum_sq = 0;
  for (int i = tid; i < normalized_size; i += nthreads) {
    // Use __ldg for read-only, coalesced global memory access
    scalar_t val = __ldg(&in_ptr[i]);
    accscalar_t a_val = static_cast<accscalar_t>(val);
    local_sum += a_val;
    local_sum_sq += a_val * a_val;
```

```
  }

  // Allocate shared memory for reduction: first part for partial sums
    , second for sum of squares.
  extern __shared__ char smem[];
  accscalar_t* s_sum = reinterpret_cast<accscalar_t*>(smem);
  accscalar_t* s_sum_sq = s_sum + nthreads;

  s_sum[tid] = local_sum;
  s_sum_sq[tid] = local_sum_sq;
  __syncthreads();

  // Perform parallel reduction in shared memory.
  for (int stride = nthreads / 2; stride > 0; stride >>= 1) {
    if (tid < stride) {
      s_sum[tid] += s_sum[tid + stride];
      s_sum_sq[tid] += s_sum_sq[tid + stride];
    }
    __syncthreads();
  }
...
}
```

To identify such issues in large scale, we applied o3-mini to generate additional test cases. Specifically, we sampled 3 generated kernel functions for each task in level 1 and asked o3-mini to detect any possible flaws and write test cases for each detected flaw. Then, we manually verified the correctness of o3-mini-generated test cases. Finally, we applied these test cases on all generations by Lange et al. [35]. Our results show that the correctness rate of generated kernels is overestimated by 31%.

F An Example of Rigorous Benchmark Reporting

In this section, we present a modified reporting example based on BIRD to demonstrate benchmark reporting that fulfills all the criteria outlined in Figure 4. BIRD is a benchmark for evaluating agents' capability to translate a natural language query to a SQL query.

R.1. Is fully or at least partially open-sourced.

Example: We released the training and validation dataset of BIRD at `https://bird-bench.github.io/`.

R.2. Offers an open-source evaluation harness for users.

Example: We released the harness to evaluation agents on BIRD at `https://github.com/AlibabaResearch/DAMO-ConvAI/tree/main/bird`.

R.3. Includes measures to prevent data contamination, such as a private, held-out test set.

Example: We keep a private held-out test set to avoid potential data contamination. Request to evaluate agents on this test set can be submitted at `https://bird-bench.github.io/`.

R.4. Includes measures or plans to consistently update challenges over time to avoid overfitting.

Example: We plan to consistently update the database and natural language queries to reflect the real-world queries and avoid overfitting. Our updates will be available at `https://bird-bench.github.io/`.

R.5. Clearly states the relationship between the agent capabilities it aims to evaluate and the constructs or outcomes it measures.

Example: BIRD evaluates agents' capabilities to serve as a database interface to translate natural language queries into executable SQL queries. To achieve that, BIRD provides agents with a natural language query, the database schema, and SQL-related domain knowledge, and challenges agents to write a SQL query that can be executed to return correct answers.

R.6. Clearly states the evaluation subjective of the benchmark (e.g., a model or an agent framework).

Example: BIRD is designed to evaluate the capability of ML models as well as the performance of agent frameworks.

R.7. Describes steps taken to prevent, identify, and correct flaws.

Example: We identify that evaluating generated SQL queries using execution results have two limitations. First, tasks requiring `LIMIT` queries and containing ties in the data may lead to non-deterministic execution results. Second, manually annotated ground-truth queries may contain errors. To understand and mitigate these errors, we randomly sample 500 tasks to perform an additional phase of verification. After verifying queries, we found 11.65% of ground-truth queries are incorrect.[4]

R.8. Includes qualitative discussions of the potential impact of unavoidable flaws.

Example: The identified incorrect ground-truth queries and potentially more incorrect ground-truth queries in the test dataset can lead to estimation errors of the agent performance and incorrect rankings of agents.

R.9. Includes quantitative analysis to assess the impact of unavoidable flaws (e.g., noise of ground truth).

Example: We build our quantitative analysis based on the normality assumption. Specifically, suppose the number of data in the test set N is large enough such that the true success rate (p) of an agent follows a normal distribution with mean μ and standard deviation σ. Given the ground truth's incorrectness rate of e and the estimated agent success rate p_0 (based on the imperfect ground truth), μ and σ are calculated as

$$\mu = e + (1-2e)p_0; \quad \sigma^2 = \mu(1-\mu) = (e + (1-2e)p_0)\,(1 - e - (1-2e)p_0)$$

Hence, based on the normality assumption, we can derive a two-sided confidence interval with confidence α for p as follows:

$$\mathbb{P}\left[\mu - 1.96 \times \frac{\sigma}{\sqrt{N}} \leq p \leq \mu + 1.96 \times \frac{\sigma}{\sqrt{N}}\right] \geq 95\% \tag{1}$$

Finally, based on the plug-in estimate (11.65%) for the ground truth's incorrectness rate, we calculate the confidence interval for the agents' performance in Table 15.

R.10. Reports metrics about statistical significance, such as confidence intervals.

Example: In additional to accuracy estimate, we also calculate confidence intervals for each model in Table 15.

R.11. Provides guidance on interpreting results with eval flaws.

Example: Given the potential flaws in BIRD, we do not recommend users to rely on the success rate alone for decision-making or selecting models. Instead, we suggest using the confidence interval of the success rate as a reference.

R.12. Reports results of non-AI baselines (e.g., human experts).

Example: We measured the performance of a SQL expert on BIRD, obtaining a success rate of 92.96%.

R.13. Reports results of trivial agents (e.g., one that does nothing).

Example: We performed sanity check on our evaluation harness by measuring the performance of a trivial agent that does nothing. We find that the trivial agent achieves 0% success rate, confirming the rigor of our evaluation implementation.

[4]We used results by Arcwise [7].

Table 15: Modified Leaderboards of BIRD [37] with Confidence Intervals.

Method	Dev. Accuracy (%)	Confidence Interval	Original Rank	Possible Rank
CHASE-SQL + Gemini	74.9	[66.8, 71.4]	1	1-13
Contextual-SQL	73.5	[65.7, 70.4]	2	1-16
XiYan-SQL	73.3	[65.6, 70.2]	3	1-18
ExSL + granite-34b-code	72.4	[64.9, 69.6]	4	1-22
Reasoning-SQL-14B	72.3	[64.7, 69.4]	5	1-22
Insights AI	72.2	[64.6, 69.4]	6	1-22
TC-SQL	70.9	[63.7, 68.4]	7	1-27
Infly-RL-SQL-32B	70.1	[63.0, 67.8]	8	1-29
Queryosity	69.4	[62.5, 67.3]	9	1-32
OpenSearch-SQL-v2 + GPT-4o	69.3	[62.4, 67.2]	10	1-32
GenaSQL	69.2	[62.4, 67.2]	11	1-33
OmniSQL-32B	69.2	[62.4, 67.1]	12	1-33
OmniSQL-7B	69.0	[62.2, 67.0]	13	1-33
PB-SQL + GPT-4o	68.6	[61.9, 66.7]	14	2-34
PURPLE + RED + GPT-4o	68.1	[61.5, 66.3]	15	2-34
Arcwise + GPT-4o	68.0	[61.4, 66.2]	16	2-34
Distillery + GPT-4o	67.2	[60.8, 65.6]	17	3-36
RSL-SQL + GPT-4o	67.2	[60.8, 65.6]	18	3-36
XiYanSQL-QwenCoder-32B	67.0	[60.6, 65.5]	19	4-36
RECAP + Gemini	67.0	[60.6, 65.4]	20	4-36
GSR	66.9	[60.5, 65.4]	21	4-36
MSL-SQL + DeepSeek-V2.5	66.8	[60.5, 65.3]	22	4-36
AskData + GPT-4o	65.9	[59.8, 64.6]	23	7-37
E-SQL + GPT-4o	65.6	[59.5, 64.4]	24	7-37
ByteBrain	65.5	[59.4, 64.3]	25	7-37
CHESS	65.0	[59.1, 63.9]	26	7-37
SCL-SQL	64.7	[58.9, 63.7]	27	7-39
EBA-SQL + GPT-4	64.6	[58.8, 63.6]	28	8-39
OeSQL-0.1-Qe-32B	64.6	[58.8, 63.6]	29	8-39
RSL-SQL + DeepSeek-v2	63.6	[58.0, 62.8]	30	9-42
Command-A	63.5	[57.9, 62.8]	31	9-42
MCS-SQL + GPT-4	63.4	[57.8, 62.7]	32	9-42
PURPLE + GPT-4o	63.0	[57.5, 62.4]	33	11-42
GRA-SQL	62.6	[57.2, 62.1]	34	14-44
E-SQL + GPT-4o mini	61.6	[56.4, 61.4]	35	17-46
OpenSearch-SQL-v1 + GPT-4	61.3	[56.2, 61.2]	36	17-46
Dubo-SQL-v1	59.7	[55.0, 59.9]	37	23-49
SuperSQL	58.5	[54.0, 59.0]	38	27-49
SFT CodeS-15B	58.5	[54.0, 59.0]	39	27-49
Chat2Query (GPT-4 + data entity modeling)	58.1	[53.8, 58.7]	40	30-50
MAC-SQL + GPT-4	57.6	[53.3, 58.3]	41	30-50
SFT CodeS-7B	57.2	[53.0, 58.0]	42	30-51
TA-SQL + GPT-4	56.2	[52.3, 57.2]	43	34-51
DeepSeek	56.1	[52.2, 57.2]	44	34-51
DTS-SQL + DeepSeek-7B	55.8	[52.0, 56.9]	45	35-51
SEE	55.5	[51.7, 56.7]	46	35-51
DAIL-SQL + GPT-4	54.8	[51.2, 56.1]	47	37-51
Interactive-T2S	54.6	[51.0, 56.0]	48	37-51
Mistral	53.5	[50.2, 55.2]	49	37-51
ExSL + granite-20b-code	51.7	[48.8, 53.8]	50	40-52
DIN-SQL + GPT-4	50.7	[48.0, 53.1]	51	42-52

Continued on next page

Table 15: Modified Leaderboards of BIRD [37] with Confidence Intervals. (Continued)

GPT-4	46.4	[44.7, 49.7]	52	50-53
Claude-2	42.7	[41.9, 46.9]	53	52-54
Open-SQL	37.7	[38.1, 43.0]	54	53-54
Palm-2	27.4	[30.3, 35.0]	55	55-58
ChatGPT + CoT	25.9	[29.2, 33.8]	56	55-58
Codex	25.4	[28.8, 33.5]	57	55-58
ChatGPT	24.1	[27.8, 32.4]	58	55-58
T5-3B	10.4	[17.6, 21.6]	59	59-61
T5-Large	9.7	[17.1, 21.1]	60	59-61
T5-Base	6.3	[14.6, 18.4]	61	59-61

G NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claimes in the abstract and introduction are justfied in the later sections and accurately reflect our paper's contribution and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We included a limitation section in the first section of Appendix (Appendix A)

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.

- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: This work does not propose new theories that need proofs.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experiment design and code are open-sourced at `https://github.com/uiuc-kang-lab/agentic-benchmarks`. Our experimental results are reproducible with our provided experiment code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide open-sourced code at `https://github.com/uiuc-kang-lab/agentic-benchmarks` and open-source data at `https://uiuc-kang-lab.github.io/agentic-benchmarks/`. We include detailed justification to our data in Appendix D.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Experiment parameters and details are discussed in Appendix E and included in our open-source code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: All reported numbers are deterministic. Our experiments contain no stochastic elements, so re-running an experiment yields identical outputs; there is therefore no run-to-run variance on which to base error bars. The evaluation metrics are computed against a single, fixed set of manual annotations (gold standard). For these reasons we do not report error bars or p-values; every number in the experiment section is exact and reproducible.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the computer resources to run our experiments in Appendix E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We confirm that our paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the potential sociatal impacts of our work in Appendix A

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work does not release data or model that have a high risks for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: Our work does not use existing assets.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Our released code are well documented with READMEs.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Our paper does not involve crowdsourcing nor human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Our paper does not involve crowdsourcing nor human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.