ReproRepo: Scaling Reproducibility Audits with GitHub Repository Issues

Shanda Li[♠], **Qiuhong Anna Wei**[♠], **Jingwu Tang**[♠], **Valerie Chen**[♠], **Nihar B Shah**[♠], **Tim Dettmers**[♠], **Yiming Yang**[♠], **Ameet Talwalkar**[♠♡]

[♠]School of Computer Science, Carnegie Mellon University [♡]Datadog Correspondence to shandal@cs.cmu.edu.

Reproducing research results from papers and released code is central to scientific progress. Existing works have introduced benchmarks to evaluate whether LLM agents can assist with reproducibility, but they are difficult to scale due to their reliance on substantial manual effort for data curation and evaluation. We introduce ReproRepo, a scalable framework for reproducibility evaluation that leverages human-raised GitHub issues as naturally occurring supervision on realistic reproduction blockers. We instantiate ReproRepo on 1,149 recent machine learning papers from major conferences and evaluate four frontier model-agent configurations. Our results show that LLM agents, even without executing code, can identify many real-world reproducibility problems from paper-repository pairs: the best agent in our study, namely Codex with GPT-5.5, surfaces at least one semantically related human-reported blocker for $\sim 90\%$ of papers in the study. Further analysis shows that agents are particularly effective for surfacing visible failures and identifying the right semantic region, but may still be insufficient in exact localization. ReproRepo can serve as a reusable, scalable framework for future evaluations of LLM agents on real-world reproducibility auditing. Our code is released at https://github.com/LithiumDA/ReproRepo.

1. Introduction

Reproducing previous research is essential for scientific progress, enabling researchers to verify findings and confidently build on prior work [1, 2]. As modern research increasingly involves navigating papers, code, data, and other experimental artifacts, there is growing interest in whether large language model (LLM) agents can help automate aspects of reproducibility given their strong coding and reasoning capabilities. While some recent works challenge agents to produce results from a paper alone [3, 4], the practical utility of reproducibility is in settings where one reproduces results using provided code [5, 6]. This more closely reflects everyday research practice, where researchers build on prior work by rerunning and extending existing codebases and experiments.

A major bottleneck faced by prior work on reproducibility evaluation is that of *scalability*. These approaches require substantial human effort, including expert-curated questions [5], artifact preparation [7], manually-injected errors [8], expert rubrics [3, 4], reference implementation verification [9], and manual evaluation [10, 11]. As a result, these datasets typically cover only a limited number of papers—usually fewer than 100—and tend to feature only papers with high-quality artifacts. Moreover, when these benchmarks quickly become outdated, the manual work required for update makes it difficult to maintain relevance in the fast-changing research landscape.

To address this limitation, we introduce ReproRepo, a framework for scalable, realistic, and easily updatable evaluation of reproducibility with LLM agents. Our key idea is to **use real reproduction pain as supervision**: instead of relying on experts to manually design tasks, inject errors, or write evaluation rubrics, ReproRepo leverages GitHub issues as evidence of reproduction blockers encountered by real users. More specifically, it first builds task instances by collecting conference papers, linked GitHub repositories, and

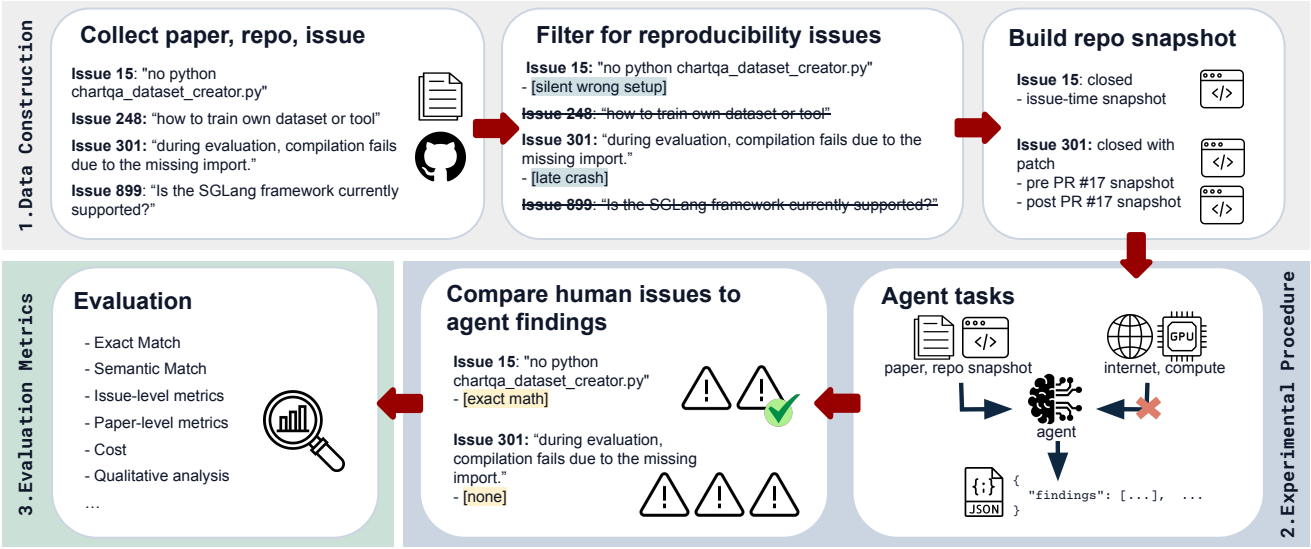

Figure 1: **Overview of ReproRepo**. We build an issue-grounded benchmark from paper-repository pairs and human-reported GitHub issues, ask agents to statically audit the paper and fixed repository snapshots, and evaluate whether their findings recover hidden human-reported reproducibility blockers.

user-reported reproducibility issues. Then LLM agents are tasked with predicting potential reproducibility blockers given papers and codebases. Finally, ReproRepo validates agent findings by comparing against human-reported ones and computes match rates as the main evaluation metric. In our study, since fully executing modern repositories often requires GPUs, large datasets, and external services such as LLM API calls, we focus on a compute-light static inspection setting for experimentation: agents inspect only the paper and repository snapshot, without code execution.

Using ReproRepo, we evaluate Claude Code and Codex paired with four frontier LLMs on 1149 papers from 3 major machine learning (ML) conferences. We find that, surprisingly, static agents (i.e., agents who do not actually execute any code) recover a substantial fraction of human-reported reproduction blockers: Codex with GPT-5.5 identifies at least one semantically related hidden issue for 93.3% of NeurIPS 2024 main-conference papers and 89.7% of ICLR 2026 papers. Further analysis shows that agents have low false positive rates, are best at surfacing visible failures, and often identify the right semantic region. However, they often still struggle with exact localization of errors and identification of less visible failure types.

Overall, the contributions of our work are three-fold:

- We introduce **ReproRepo**, a scalable framework for auditing research reproducibility with LLM agents. It uses GitHub issues as naturally occurring human supervision, enabling diverse and continuously updatable benchmarks with minimal manual annotation.
- We instantiate ReproRepo on recent machine learning papers from major conferences, constructing a large-scale benchmark of 1,149 papers and 7,553 human-reported reproduction blockers.
- We systematically evaluate frontier LLM agents under ReproRepo. Our results show that static inspection can surface human-reported blockers for most papers, but agents still struggle with precise localization despite strong semantic coverage of visible failures.

Framework	Given Paper	Given Code	# Papers	Supervision Source
PaperBench [3]	✓	–	20	Conference papers, expert rubrics
Paper2Code [4]	✓	–	90	Conference papers, expert eval
AutoReproduce [9]	✓	–	13	Papers w/ verified implementation
ReplicationBench [7]	✓	–	31	Papers, expert tasks
Read Paper, Write Code [12]	∼	–	48	Expert-verified reproducible papers
CORE-Bench [5]	–	✓	90	Papers w/ repos, expert-curated questions
R-based social science study [8]	–	✓	5	Reproducible studies, expert-injected failures
Scaling Reproducibility [11]	✓	✓	384	Journal papers, expert review
REPRO-Bench [6]	✓	✓	112	Papers w/ expert reproducibility report
ReplicatorBench [13]	✓	⋆	19	Papers w/ expert reproducibility report
Ours (ReproRepo)	✓	✓	**1149**	Conference papers, GitHub issues

Table 1: **Comparing ReproRepo with prior reproducibility evaluations.** The works are ordered by input setting: paper-only, code-only, and paper-plus-code. Checkmarks indicate whether the task includes the listed input; "∼" indicates methods description of papers; "⋆" indicates task-dependent code access.

2. Related Work

LLM agents for scientific workflows Before LLM agents, reproducibility evaluation relied on reporting checklists, submission policies, and independent execution or reanalysis [1, 14, 15, 2]. These efforts clarified review standards but depended on scarce expert labor.

Recent advances of LLM agents have led to a range of works exploring their potential in resolving repository issues [16], performing research implementations [17–19], assisting checklist review [20], verifying scientific manuscripts [21–23], assessing research proposal soundness [24], supporting data-driven discovery [25], assessing soundness of automated research [26] and even rediscovering established findings [27].

These agent capabilities made it possible and easier to automate reproducibility evaluation workflows.

Agentic reproducibility evaluation. Recent works applying LLM agents to reproducibility evaluation can largely be categorized by input type: paper-only, code-only, paper-plus-code. Paper-only works ask agents to recreate results or implementations from a manuscript or paper-derived description [3, 4, 28, 9, 7, 12]. These tasks often focus more on code generation and measure scientific understanding. Code-only and paper-plus-code works utilize the released code and ask agents to answer curated questions [5], repair errors in social science research codebases [8], conduct reanalysis of data [11], detect fabrications [29], score paper-code reproducibility [6, 30], and evaluate claim replicability [13]. Table 1 summarizes how our work compares with the prior benchmark works.

In particular, our work focuses on the paper-plus-code setting and leverages GitHub issues for scalability.

3. ReproRepo

In this section, we describe our proposed framework, ReproRepo, for evaluating LLM agents on reproducibility auditing (Figure 1).

3.1. Data construction

Our data curation strategy is motivated by how reproducibility problems arise in ordinary research practice. Researchers commonly release code on GitHub, and follow-up users often open issues when they encounter blockers while trying to reproduce the paper. We treat these issues as user-reported records of reproduction experiences: even if some recent reports are drafted with AI assistance, opening an issue still reflects a user's judgment that the blocker is real and relevant. These issues therefore provide naturally occurring records of reproduction experiences and can serve as free supervision signals. Moreover, GitHub issues come with useful structure, such as timestamps, open/closed status, discussion threads, and, in some cases, linked commits or pull requests that document how a problem was resolved. Our data construction pipeline is built upon these properties.

Data collection & filtering. We construct evaluation data from conference papers, their associated GitHub repositories, and user-reported GitHub issues. Starting from a complete set of accepted or published conference papers, we collect paper metadata and GitHub repository URLs from Paper Copilot [31] and conference metadata. Non-GitHub links and papers without a usable repository are filtered out. For each repository, we collect both open and closed GitHub issues. Open issues provide real user reports. Closed issues provide historical reports that may have been clarified or fixed after the original failure. Then an LLM judge is employed to filter for reproducibility-related issues.

Repository snapshot construction. To prevent information leakage and post hoc reasoning during reproducibility evaluation, we build snapshots differently for open and closed issues. For open issues, we collect the repository's default-branch state at collection time as the codebase snapshot. For closed issues, we collect the issue-time codebase snapshot, i.e., the repository state immediately before the issue creation, to make sure the evaluation presents to agents a pre-fix code state that a user would have seen when reporting the problem. For closed issues with patch evidence, we additionally recover a post-fix snapshot from the human resolution. This will be used as an additional validation set, i.e., a target problem should disappear after the human fix, to evaluate the false positive rate of predicted issues from agents.

3.2. Reproducibility Audit Protocol

Given the constructed paper-repository snapshots and hidden human issues, we next describe how agents perform static audits and how their outputs are assessed.

Agent tasks. We task LLM agents with identifying potential reproducibility blockers given papers and codebases as input. The curated human-reported issues are hidden from the agents. Agents are expected to output a list of predicted reproduction blockers, ordered by the estimated likelihood of the blocker encountered by real users. Each finding should describe one concrete, GitHub-issue-style problem that a real reproducer would plausibly report. For each run, we prepare an isolated workspace that contains the code repository snapshot and the corresponding paper file. To prevent information leakage and post hoc reasoning, we remove all git commit histories and deny the agent internet access. Our evaluation particularly considers a compute-light static inspection setting. The agent is instructed to perform a static inspection without environment setup and code execution. GPU compute is withheld from the workspace. The full prompt is provided in Section E.1.

Agent output assessment. After prediction, we evaluate the agent output with a separate LLM judge. The judge receives the hidden human issues for the same paper-snapshot pair as well as the agent's blind findings. For each hidden human issue, the judge compares it against the agent findings and assigns a label among *exact match* (EM), *semantic match* (SM), and *none* based on the following criteria:

- **EM**: The finding identifies substantially the same practical reproduction problem and the same trigger, even if the wording might differ.
- **SM**: The finding identifies a semantically related blocker with concrete overlap yet incomplete specificity, e.g., the same evaluation pipeline but different failure points, the same missing/released artifact family but different files or checkpoints, etc.
- **None**: No agent finding would be useful evidence that the agent discovered this issue.

Concrete examples of the three labels, including verbatim agent-output excerpts, are provided in Section F. The full judge prompt is provided in Section E.2.

3.3. Evaluation Metrics

We evaluate agent predictions based on the assessment outcome described above. The main metrics are EM rate and SM rate, detailed below.

Top-k reporting budget. The agent outputs an ordered list of findings for each paper/repository snapshot. To make scores comparable across agents and to avoid rewarding arbitrarily long reports, we evaluate matches under a fixed reporting budget. For a budget k, only the top k findings are eligible to match hidden issues. We write EM@k and SM@k for exact and semantic matching under this budget. We use $k = 10$ in the main experimental results. We also report curves over k to show how performance changes as the reviewer-facing budget increases in Section 5.2.

Result aggregation level. Exact match and semantic match can be aggregated across different granularities. Let $F_p^{(k)}$ denote the top k agent findings for paper or snapshot group p. A hidden issue i associated with p is counted as matched at k only if its best-aligned finding lies in $F_p^{(k)}$. The *issue-level metric* treats each hidden issue as one evaluation item. Let I be the set of hidden issues, and let $y_i^{(k)}$ be the best label for issue i under the top-k budget. We define:

$$\text{Issue}_{\text{EM}}@k = \frac{|\{i \in I : y_i^{(k)} = \text{EM}\}|}{|I|}; \qquad \text{Issue}_{\text{SM}}@k = \frac{|\{i \in I : y_i^{(k)} \in \{\text{EM}, \text{SM}\}\}|}{|I|}.$$

Issue-level match rate measures how much of the hidden issue evidence is recovered by the agent. It is the most direct issue-retrieval metric, but it can be strict for papers with many related issues, since each issue is counted separately. The *paper-level metric* characterizes whether the static audit surfaces one or all human-observed reproduction blockers for a paper. Let P be the set of evaluated papers, and let I_p be the set of hidden issues associated with paper p. The *any-issue* paper match rate counts a paper as match if at least one of its hidden issues is matched, while the *all-issue* paper match rate counts a paper only if all the hidden issues are matched.

$$\text{Paper}_{\text{EM}}^{\text{any}}@k = \frac{|\{p \in P : \exists\, i \in I_p \text{ s.t. } i \text{ is EM@}k\}|}{|P|}; \quad \text{Paper}_{\text{EM}}^{\text{all}}@k = \frac{|\{p \in P : \forall\, i \in I_p,\ i \text{ is EM@}k\}|}{|P|}.$$

Conference	# accepted papers	# selected papers	# open issues	# closed issues	# closed issues w/ patch
NeurIPS 2022 main	2905	264	579	974	22
NeurIPS 2022 DB	163	88	342	524	32
NeurIPS 2024 main	4037	410	1642	1518	40
NeurIPS 2024 DB	460	107	299	678	44
ICLR 2026	5355	280	405	592	12
All datasets	12686	1149	3267	4286	150

Table 2: **Dataset Statistics.** Accepted-paper counts are conference- or track-level public counts. Selected papers are papers retained after issue filtering, i.e., papers with at least one accepted benchmark issue. Open and closed issue counts refer to accepted benchmark cases. DB denotes datasets and benchmarks track.

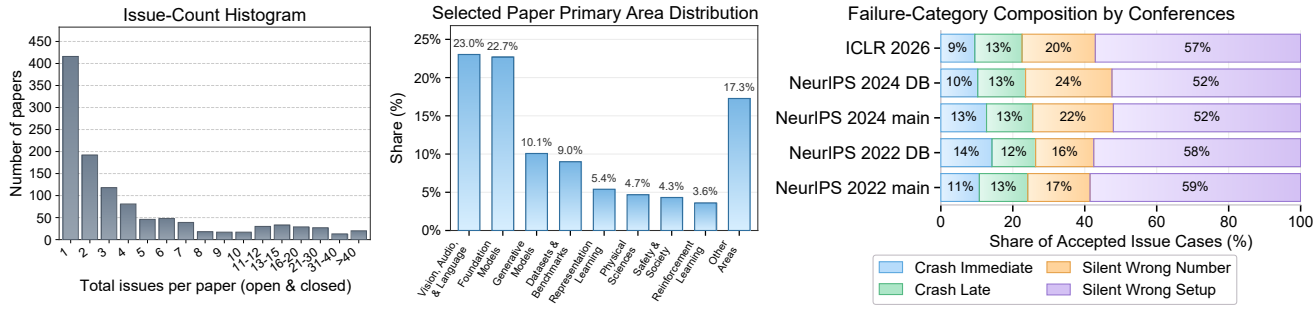

Figure 2: **Dissections on the 1149 selected papers and 7553 issues. Left**: distribution of selected issue counts per paper, aggregated across all five conferences. **Middle**: primary research area distribution of selected ICLR 2026 papers where area metadata is available. **Right**: composition of failure categories in the selected issues. DB denotes datasets and benchmarks track.

$\text{Paper}^{\text{any}}_{\text{SM}}@k$ and $\text{Paper}^{\text{all}}_{\text{SM}}@k$ are defined similarly. This paper-level coverage metric is practically relevant. For example, identifying one real blocker can already direct reviewer or author attention to a problematic artifact in paper triage.

False positive rate. High match rates are useful only if agent findings are not dominated by generic or persistent warnings that remain even after a real problem has been fixed. We therefore evaluate false positives using closed issues with clear code patches. For each such issue, we run the agent on the *post-fix* repository snapshot, where the corresponding historical blocker should no longer be present. If the agent still reports a finding that matches the fixed issue, we count it as a false positive. In Section 5, we aggregate FPR at the paper level: a paper is counted as false positive if any predicted finding matches an already-fixed issue. This is a conservative criterion and provides an upper bound on issue-level false positives.

4. Evaluation Setup

4.1. Data Overview

In our study, we focus on machine learning conferences because code release is common in this community and paper metadata is readily available from OpenReview. To cover artifacts from both earlier and recent venues, we include NeurIPS 2022 and NeurIPS 2024, each separated into the main conference and the

Model	Agent	Issue Match		Paper Match (Any)		Paper Match (All)		FPR	Cost
		EM@10	SM@10	EM@10	SM@10	EM@10	SM@10		
DeepSeek-V4-Pro	Claude Code	19.3%	53.1%	47.6%	89.0%	25.6%	55.3%	2.80%	$0.29
Claude Opus 4.7	Claude Code	21.7%	54.1%	52.0%	86.5%	27.8%	57.1%	2.10%	$4.07
GPT-5.4-Mini	Codex	13.4%	43.4%	33.0%	82.1%	18.3%	52.0%	**1.40%**	$0.22
GPT-5.5	Codex	**25.4%**	**58.1%**	**59.7%**	**89.7%**	**34.8%**	**65.6%**	3.50%	$1.26

Table 3: **Comparing Model Performances.** EM@10 and SM@10 denote exact match rate and semantic match rate among the first 10 agent findings. All match metrics are computed on the ICLR 2026 paper set. FPR is computed on the full post-fix validation subset across all five conferences/tracks. Issue match is computed over hidden issues. Paper Match (Any) counts a unique paper as match if at least one hidden issue for that paper is matched, while Paper Match (All) requires all hidden issues for that paper to be matched. FPR denotes false positive rate on the post-fix snapshots of all closed issues with patches. Cost denotes the average cost per run.

datasets & benchmarks track, as well as ICLR 2026. Starting from accepted papers with public GitHub repository links, we collect candidate open and closed GitHub issues and retain papers with at least one reproducibility-related issue. This collection and filtering procedure is venue-agnostic and can be readily extended to additional conferences or journals.

Dataset statistics. Table 2 provides an overview of the data scale in our evaluation. 1149 papers and 7553 human-reported reproducibility issues are selected from a set of 12686 papers, making the evaluation significantly larger than all the prior studies. In particular, there are 150 closed issues with clear code patches that address the issue, which will be used to examine false positive rates of the agent prediction.

Dataset dissection. Figure 2 summarizes three views of the resulting benchmark. The left panel shows that the number of selected issues per paper follows a long-tail distribution: most papers contribute only a handful of reproducibility issues, while a small set of heavily-used repositories contributes many. Per-conference histograms are deferred to Figure 5. The middle panel shows that selected papers span a diverse range of primary research areas, with Vision, Audio & Language Applications (23.0%) and Foundation Models (22.7%) as the largest groups, followed by a long tail of other areas. The right panel shows that the four failure categories, namely crash immediate, crash late, silent wrong setup, and silent wrong number, detailed in Section B.2, appear in stable proportions across all five conferences/tracks, with *silent wrong setup* as the dominant category (52–59%). The results suggest that ReproRepo captures a general distribution of diverse real-world reproducibility blockers.

4.2. Models & Agents

Our evaluation covers two popular coding agents, Claude Code and Codex, paired with four frontier open and closed models: DeepSeek-V4-Pro [32], Claude Opus 4.7 [33], GPT-5.4-Mini [34], and GPT-5.5 [35]. DeepSeek-V4-Pro and Claude Opus 4.7 are evaluated with Claude Code, and GPT-5.4-Mini and GPT-5.5 with Codex.

Conference	Issue Match		Paper Match (Any)		Paper Match (All)		Token Usage	Cost
	EM@10	SM@10	EM@10	SM@10	EM@10	SM@10		
NeurIPS 2022 main	23.8%	57.4%	61.2%	89.7%	15.2%	41.4%	966,794	$1.28
NeurIPS 2022 DB	18.2%	51.0%	60.2%	96.6%	10.2%	31.8%	1,225,375	$1.50
NeurIPS 2024 main	23.9%	56.3%	61.0%	93.3%	22.7%	51.1%	925,068	$1.30
NeurIPS 2024 DB	12.2%	47.3%	46.7%	87.6%	10.5%	33.3%	1,159,698	$1.53
ICLR 2026	25.4%	58.1%	59.7%	89.7%	34.8%	65.6%	943,011	$1.26

Table 4: **Comparing Results on Different Conferences.** EM@10 and SM@10 denote exact match rate and semantic match rate among the first 10 agent findings, and the results are based on Codex with GPT-5.5. Issue match is computed over hidden issues. Paper Match (Any) counts a unique paper as match if at least one hidden issue for that paper is matched, while Paper Match (All) requires all hidden issues for that paper to be matched. Token usages refers to the average number of total tokens per run, including both input and output. Cost denotes the average cost per run.

5. Experimental Results

In this section, we present our experimental results and discuss the key findings.

5.1. Main Evaluation Result

Table 3 compares the four models in our study on the ICLR 2026 paper set. Table 4 further reports the performance of Codex with GPT-5.5, the best-performing configuration in Table 3, across five conferences/tracks. We make the following observations based on the results:

Surprisingly, agents can consistently identify reproducibility blockers through static inspection. Table 3 shows that all four models recover at least one human-reported issue for a large fraction of papers without executing the code. Under semantic matching, the paper-level any-issue match rate ranges from 82.1% to 89.7%; even under the stricter exact match criterion, the state-of-the-art GPT-5.5 model achieves a rate of 59.7%. Table 4 further shows that this trend holds across all five conference/track splits. These results suggest that many real reproducibility blockers leave visible traces in the paper-repository pair. Meanwhile, we note that while DeepSeek-V4-Pro is less accurate under exact matching, it reaches a similar paper-level semantic match rate to GPT-5.5 (89.0% vs. 89.7%) while costing only 23% as much per run. This makes it a strong cost-quality trade-off when broad triage coverage is the main goal.

Agents often identify the right semantic region, but exact localization remains challenging. Across both Table 3 and Table 4, semantic match rates are substantially higher than exact match rates. For example, in the model comparison, issue-level SM@10 ranges from 43.4% to 58.1%, while issue-level EM@10 ranges only from 13.4% to 25.4%. A similar gap appears at the paper level. This indicates that agents can often identify the relevant workflow, artifact, or failure region, but do not always recover the precise trigger or root cause reported by users. Therefore, semantic matches should be interpreted as useful triage signals rather than confirmed localization of exact issues.

The results remain stable across venues and years. Table 4 shows no clear evidence that the agent performs better on older or later venues. The issue-level semantic match rates are close for NeurIPS 2022 main, NeurIPS 2024 main, and ICLR 2026, and the paper-level any-issue semantic match rate remains

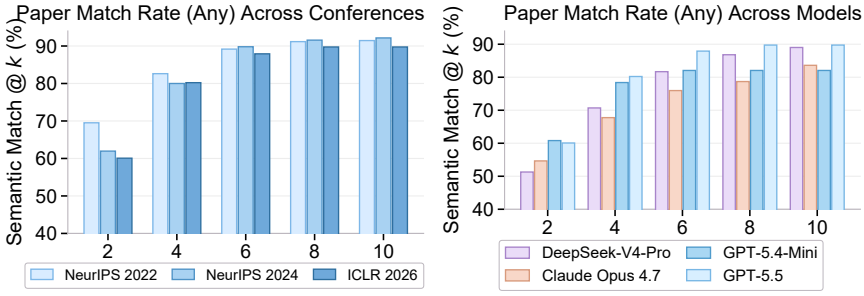

Figure 3: **Semantic match rates with varying reporting budget** k. **Left**: conference comparison for GPT 5.5 with Codex, with the two NeurIPS tracks combined by year. **Right**: model comparison on ICLR 2026.

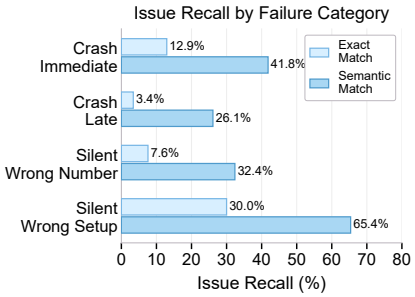

Figure 4: **Exact and semantic match rates** of GPT-5.5 on different failure categories.

around 90%. This stability suggests that the task is unlikely to be solved mainly by memorization. One notable trend is that Datasets and Benchmarks tracks tend to have lower issue-level and all-issue match rates than the corresponding main conferences, indicating that their reproduction blockers may be more subtle or harder to identify. They also require more tokens on average, consistent with the larger or more complex artifact structure in these papers.

The false positive rate is very low. In Table 3, the false positive rate is at most 3.50% across models, much lower compared with paper-level match rates. This is encouraging because the false-positive evaluation is conducted on post-fix snapshots, where the matched historical issue should no longer be present. The result indicates that the agents' findings are not dominated by hallucinated blockers and that most reported issues correspond to plausible static evidence.

5.2. Ablation Study

Effect of the reporting budget. A natural concern is that the paper-level match rate may be driven mainly by allowing the agent to report many candidate findings.

To address this concern, we plot the match rate under top-k reporting budget for varying k in Figure 3. The result shows that the match rates always rise quickly at small k. Across conferences, semantic paper-level match is already above 60% at $k = 2$ and around 80% or higher at $k = 4$, before plateauing at roughly 90% by $k = 6$ to 10. The model comparison shows the same pattern: most of the gain appears in the first few ranked findings, while later findings provide smaller incremental coverage. This suggests that the high match rate does not come from long issue lists. Our ranked static audit requirement encourages the model to place real reproduction blockers near the top of the report. The property is practically useful for reviewer or author triage when humans only have limited bandwidth to check a few potential issues.

Ablation on paper input. One important aspect of reproducibility is paper-code consistency. Simply inspecting codebases without access to the paper may not be sufficient for identifying all reproduction blockers, since many failures are defined only relative to the paper's claimed datasets, checkpoints, evaluation metrics, etc. We validate this hypothesis by comparing a code-only setting with the default paper-and-code setting in Table 6. Adding the paper improves performance across all metrics, with the largest gains appearing under exact matching: EM@10 increases by 8.98 percentage points at the paper-

Finding type	Count
Documentation-repository mismatch	27
Dependency or environment gap	23
Missing or under-specified artifacts	23
Missing evaluation/reproduction workflow	13
Paper-code protocol mismatch	3

Table 5: **Manual categorization** of 89 top-ranked unmatched Codex findings from papers with one collected human issue.

Agent input	Issue Match		Paper Match (Any)	
	EM@10	SM@10	EM@10	SM@10
Code only	20.9%	57.0%	50.7%	87.6%
Paper & Code	**25.4%**	**58.1%**	**59.7%**	**89.7%**

Table 6: **Ablation on agent input.** Experiment is conducted on ICLR 2026 paper set with GPT-5.5.

level any-issue metric. This suggests that repository inspection alone can often identify the broad failure region, as reflected by the strong code-only semantic match rates, but access to the paper helps the agent determine whether a repository artifact, command, configuration, or released resource actually supports the paper's intended reproduction target. Therefore, paper input is most valuable for turning general repository-risk detection into more precise paper-grounded reproducibility auditing.

5.3. Issue Taxonomy

We next examine which kinds of reproducibility issues appear in the benchmark and which of them are visible to static auditing. Figure 2 (right) summarizes the distribution of the four human issue categories, and Figure 4 reports category-level recall. The category definitions are given in Section B.2. We defer the corresponding agent-prediction category distributions to Figures 7 to 9.

We make two observations:

The issue taxonomy is consistent across conferences. Figure 2 (right) shows that the 5 conferences/-tracks share highly similar patterns in their failure-category composition. Specifically, *Silent Wrong Setup* is consistently the largest failure category. It covers more than half of the issues, including mismatch of the paper's intended configuration, inconsistency in the artifact, etc. This suggests that the benchmark is not driven by a venue-specific artifact norm; across datasets, many reproduction blockers arise before long training runs or final metric comparison.

Static auditing works better for visible repository failures. Figure 4 shows that the agent recovers *silent wrong setup* most reliably, followed by *crash immediate*. This pattern is consistent with static inspection: missing checkpoints, undocumented data preparation, broken imports, mismatched scripts, absent evaluation commands, and paper-repository scope gaps often leave evidence in READMEs, configs, file structure, or paper claims. By contrast, *crash late* and *silent wrong number* are harder to recover statically because they often require execution, downloaded artifacts, hardware-specific paths, long-horizon training, or comparison against reported metrics. Taken together, these results suggest that no-execution auditing is useful for early triage of visible setup, artifact, and evaluation-path risks, but cannot replace execution or output comparison.

5.4. Additional Analyses

We further examine whether unmatched top-ranked findings are unsupported predictions or parallel reproducibility concerns not reported by humans. For this analysis, we focus on 89 ICLR 2026 papers with exactly one human-reported issue, where the top-ranked Codex finding does not match that issue. We

manually categorize each finding using static repository evidence, including documentation, scripts, configs, dependency files, file inventory, and paper-claim alignment. Manual review shows that **most unmatched findings reflect parallel reproducibility risks.** All 89 findings had concrete static evidence for a plausible issue in the released artifact. The type breakdown is presented in Table 5. These findings can fail to match the single collected human issue for several reasons: a user may report only the first blocker encountered, some gaps can be repaired locally without filing an issue, and paper-level reproduction gaps such as missing aggregation scripts may not appear during ordinary package use. Thus, in this low-issue-count subset, non-matching top-ranked agent findings more often reflect sparse human supervision and parallel reproduction risks than unsupported predictions.

6. Conclusion

We introduce ReproRepo, a scalable framework for evaluating LLM agents on issue-grounded static reproducibility auditing. By using human-reported GitHub issues as naturally occurring supervision, ReproRepo enables large, realistic, and easily updatable evaluation over paper-repository pairs. Across 1,149 machine learning papers and 7,553 reproducibility blockers curated from recent major conferences, we find that static agents can surface many real user-facing failures without executing code, identifying at least one semantically related issue for roughly 90% of papers with the best configuration, i.e., Codex with GPT-5.5. However, currently, these agents still struggle with the exact localization of the blockers. In the future, ReproRepo can serve as a reusable pipeline for evaluating new models' and agents' capabilities to audit the reproducibility of newly released papers.

Limitations

ReproRepo is based on GitHub issues. While those human-reported issues provide evidence from real users and make the benchmark substantially easier to scale, they are noisier than expert-curated labels. Users may report only the first blocker they encounter, some valid reproduction risks may never be filed as issues, and a small number of issues may come from user mistakes or reflect user-specific environments or transient external states. This limitation can be mitigated through data filtering. Another limitation is that the issues do not necessarily exhaust all reproducibility blockers, especially for repositories with few users. Our experiments focus on static, no-execution auditing. This setting is intentionally compute-light and useful for early triage. However, it does not cover all reproducibility failures, such as long-running crashes, hardware-specific errors, etc. Future work can combine issue-grounded static auditing with execution-based validation for a more complete view of reproducibility.

Ethics Statement

This study uses publicly available repository metadata and GitHub issue discussions as evidence of reproducibility barriers. Our analysis does not require identifying individual issue authors, and any released data should avoid unnecessary personal information such as user handles. Automated audit findings should be treated as triage signals rather than definitive judgments about a paper or repository. To reduce the risk of over-interpretation, we evaluate false positives and ground findings in repository evidence. Repository maintainers should have opportunities to correct false positives, clarify documentation, and update artifacts.

Acknowledgments

NBS was supported in part by grants NSF 1942124 and ONR N000142512346. TD was supported in part by Schmidt Sciences. AT was supported in part by gift funding from Datadog.

References

[1] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). *Journal of Machine Learning Research*, 22(164):1–20, 2021. URL https://www.jmlr.org/papers/v22/20-303.html.

[2] Daniel Nüst and Stephen J Eglen. CODECHECK: an open science initiative for the independent execution of computations underlying research articles during peer review to improve reproducibility. *F1000Research*, 10:253, 2021. doi: 10.12688/f1000research.51738.2. URL https://f1000research.com/articles/10-253/v2. [version 2; peer review: 2 approved].

[3] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 56843–56873. PMLR, 2025. URL https://proceedings.mlr.press/v267/starace25a.html.

[4] Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2Code: Automating code generation from scientific papers in machine learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3DcaUTjdKc.

[5] Zachary S Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=BsMMc4MEGS.

[6] Chuxuan Hu, Liyun Zhang, Yeji Lim, Aum Wadhwani, Austin Peters, and Daniel Kang. REPRO-bench: Can agentic AI systems assess the reproducibility of social science research? In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 23616–23626, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1210. URL https://aclanthology.org/2025.findings-acl.1210/.

[7] Christine Ye, Sihan Yuan, Suchetha Cooray, Steven Dillmann, Ian LV Roque, Dalya Baron, Philipp Frank, Sergio Martin-Alvarez, Nolan Koblischke, Frank J Qu, et al. Replicationbench: Can AI agents replicate astrophysics research papers? *arXiv preprint arXiv:2510.24591*, 2025.

[8] Syed Mehtab Hussain Shah, Frank Hopfgartner, and Arnim Bleier. Automating computational reproducibility in social science: Comparing prompt-based and agent-based approaches. *arXiv preprint arXiv:2602.08561*, 2026. doi: 10.48550/arXiv.2602.08561. URL https://arxiv.org/abs/2602.08561.

[9] Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. AutoReproduce: Automatic AI experiment reproduction with paper lineage. *arXiv preprint arXiv:2505.20662*, 2025. doi: 10.48550/arXiv.2505.20662. URL https://arxiv.org/abs/2505.20662. Accepted by ACL 2026 Main.

[10] Xiaoyan Bai, Alexander Baumgartner, Haojia Sun, Ari Holtzman, and Chenhao Tan. The story is not the science: Execution-grounded evaluation of mechanistic interpretability research. *arXiv preprint arXiv:2602.18458*, 2026.

[11] Yiqing Xu and Leo Yang Yang. Scaling reproducibility: An AI-assisted workflow for large-scale replication and reanalysis. *arXiv preprint arXiv:2602.16733*, 2026. doi: 10.48550/arXiv.2602.16733. URL https://arxiv.org/abs/2602.16733.

[12] Benjamin Kohler, David Zollikofer, Johanna Einsiedler, Alexander Hoyle, and Elliott Ash. Read the paper, write the code: Agentic reproduction of social-science results. *arXiv preprint arXiv:2604.21965*, 2026.

[13] Bang Nguyen, Dominik Soós, Qian Ma, Rochana R Obadage, Zack Ranjan, Sai Koneru, Anna Szabelska, Adam Gill, Timothy M. Errington, Shakhlo Nematova, Sarah Rajtmajer, Jian Wu, and Meng Jiang. ReplicatorBench: Benchmarking LLM agents for replicability in social and behavioral sciences. *arXiv preprint arXiv:2602.11354*, 2026. doi: 10.48550/arXiv.2602.11354. URL https://arxiv.org/abs/2602.11354.

[14] Ian Magnusson, Noah A Smith, and Jesse Dodge. Reproducibility in NLP: What have we learned from the checklist? In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12789–12811, 2023. doi: 10.18653/v1/2023.findings-acl.809. URL https://aclanthology.org/2023.findings-acl.809/.

[15] Robert Stojnic. ML code completeness checklist. Papers with Code Blog, 2020. URL https://medium.com/paperswithcode/ml-code-completeness-checklist-e9127b168501.

[16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024. doi: 10.48550/arXiv.2310.06770. URL https://arxiv.org/abs/2310.06770.

[17] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

[18] Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A research coding benchmark curated by scientists. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 30624–30650. Curran Associates, Inc., 2024. doi: 10.52202/079017-0963. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/36850592258c8c41cecdaa3dea5ff7de-Paper-Datasets_and_Benchmarks_Track.pdf.

[19] Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, Juntong Ni, George Michalopoulos, Yue Zhang, Ziyang Zhang, Mian Zhang, Zhiyu Chen, and Xinya Du. LMR-BENCH: Evaluating LLM agent's ability on reproducing language modeling research. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6164–6186, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.314. URL https://aclanthology.org/2025.emnlp-main.314/.

[20] Alexander Goldberg, Ihsan Ullah, Thanh Gia Hieu Khuong, Benedictus Kent Rachmat, Zhen Xu, Isabelle Guyon, and Nihar B Shah. Usefulness of LLMs as an author checklist assistant for scientific papers: NeurIPS'24 experiment. *arXiv preprint arXiv:2411.03417*, 2024. doi: 10.48550/arXiv.2411.03417. URL https://arxiv.org/abs/2411.03417.

[21] Ryan Liu and Nihar Shah. ReviewerGPT? An exploratory study on using large language models for paper reviewing. *arXiv preprint 2306.00622*, 2023. AAAI 2024 Workshop on Scientific Document Understanding.

[22] Guijin Son, Jiwoo Hong, Honglu Fan, Heejeong Nam, Hyunwoo Ko, Seungwon Lim, Jinyeop Song, Jinha Choi, Gonçalo Paulo, Youngjae Yu, and Stella Biderman. When AI co-scientists fail: SPOT-a benchmark for automated verification of scientific research. *arXiv preprint arXiv:2505.11855*, 2025. doi: 10.48550/arXiv.2505.11855. URL https://arxiv.org/abs/2505.11855.

[23] Sarina Xi, Vishisht Rao, Justin Payan, and Nihar B Shah. FLAWS: A benchmark for error identification and localization in scientific papers. *arXiv preprint arXiv:2511.21843*, 2025. doi: 10.48550/arXiv.2511.21843. URL https://arxiv.org/abs/2511.21843.

[24] Sy-Tuyen Ho, Minghui Liu, Huy Nghiem, and Furong Huang. Soundnessbench: Can your AI scientist really tell good research ideas from bad ones?, 2026. URL https://arxiv.org/abs/2605.30329.

[25] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6z4YKr0GK6.

[26] Ziming Luo, Atoosa Kasirzadeh, and Nihar B Shah. The more you automate, the less you see: The hidden pitfalls of AI scientist systems. In *NeurIPS 2025 AI for Science Workshop*, 2025. URL https://openreview.net/forum?id=7Sndugns1l.

[27] Zhen Wang, Fan Bai, Zhongyan Luo, Jinyan Su, Kaiser Sun, Xinle Yu, Jieyuan Liu, Kun Zhou, Claire Cardie, Mark Dredze, Eric P. Xing, and Zhiting Hu. FIRE-bench: Evaluating agents on the rediscovery of scientific insights. *arXiv preprint arXiv:2602.02905*, 2026. doi: 10.48550/arXiv.2602.02905. URL https://arxiv.org/abs/2602.02905.

[28] Mingyang Zhou, Quanming Yao, Lun Du, Lanning Wei, and Da Zheng. Reflective paper-to-code reproduction enabled by fine-grained verification. *arXiv preprint arXiv:2508.16671*, 2025. doi: 10.48550/arXiv.2508.16671. URL https://arxiv.org/abs/2508.16671.

[29] Hui Chen, James Xu Zhao, Dongfu Jiang, Qianyun Guo, Jiefeng Chen, Yiwei Wang, Muhao Chen, See-Kiong Ng, Pang Wei Koh, and Bryan Hooi. FabScore: Fine-grained evaluation of fabrications in

automated AI research. In *ICML 2026 AI for Science Workshop*, 2026. URL https://openreview.net/forum?id=6gYWcYQcWM.

[30] Linhao Zhang, Tong Xia, Jinghua Piao, Lizhen Cui, and Yong Li. PaperRepro: Automated computational reproducibility assessment for social science papers. *arXiv preprint arXiv:2603.00058*, 2026. doi: 10.48550/arXiv.2603.00058. URL https://arxiv.org/abs/2603.00058.

[31] Jing Yang, Qiyao Wei, and Jiaxin Pei. Paper Copilot: Tracking the evolution of peer review in AI conferences. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CyKVrhNABo.

[32] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026.

[33] Anthropic. System Card: Claude Opus 4.7. https://www.anthropic.com/system-cards, April 2026. Accessed 2026-05-25.

[34] OpenAI. Introducing GPT-5.4 mini and nano. https://openai.com/index/introducing-gpt-5-4-mini-and-nano/, March 2026. Accessed 2026-05-25.

[35] OpenAI. GPT-5.5 System Card. https://openai.com/index/gpt-5-5-system-card/, April 2026. Accessed 2026-05-25.

A. Artifact Use, Licenses, & Intended Use

Our study builds on existing public artifacts, including conference paper metadata, public GitHub repositories, GitHub issue threads, and repository links discovered from Paper Copilot and conference metadata. We use these artifacts only for research purposes: constructing an issue-grounded benchmark for evaluating static reproducibility auditing, measuring whether LLM agents can identify human-reported reproduction blockers, and supporting aggregate analyses of reproducibility failures.

Paper Copilot [31] is used as a discovery source for paper metadata and repository links, rather than as a source of benchmark labels. We cite Paper Copilot and do not claim ownership over its content. Our released artifacts will not relicense or redistribute Paper Copilot content beyond the minimal metadata needed to identify papers and repositories, subject to the applicable terms of use.

GitHub repositories and issues remain governed by their original repository licenses, GitHub terms, and any other applicable upstream conditions. Our work does not relicense repository code, issue text, papers, or third-party metadata. When releasing artifacts, we will release only derived annotations, evaluation scripts, paper/repository identifiers, issue identifiers, commit hashes, and instructions for reconstructing snapshots where permitted. Users of the released artifacts are responsible for complying with the original licenses and access conditions of the corresponding papers, repositories, GitHub content, and metadata sources.

The intended use of the artifacts created by our work is research on reproducibility auditing, benchmark construction, and evaluation of LLM agents. The artifacts are not intended for ranking, shaming, or making definitive claims about individual authors, maintainers, papers, or repositories. Agent-generated audit findings should be treated as triage signals that require human verification. Use of derived artifacts outside research contexts, or in ways that conflict with the original access conditions of the underlying sources, is not intended.

B. Dataset Details

B.1. Per-Conference Issue Counts

Figure 5 breaks down the issue-count distribution from Figure 2 by conference and track. During data preprocessing, we deliberately excluded repositories with more than 100 issues, as collecting issues from such repositories would require excessive GitHub API requests. It shows that the long-tailed pattern is not driven by a single split: in every venue, most selected papers have a few human-reported reproducibility issues, while a small number of repositories attract many reports.

The skew in Figure 5 motivates reporting both issue-level and paper-level metrics. Issue-level match rates measure how much human issue evidence is recovered, while paper-level match rates prevent a small number of heavily discussed repositories from dominating the interpretation.

B.2. Failure Taxonomy

Typical accepted problems include environment failures, missing artifacts, broken scripts or configs, paper-repository mismatches, training or inference blockers, evaluation failures, and result discrepancies. We categorize the human issue evidence along two axes: (1) whether the failure is visible as a crash or silent because the code runs but yields the wrong setup or result, and (2) whether it affects startup/setup or a

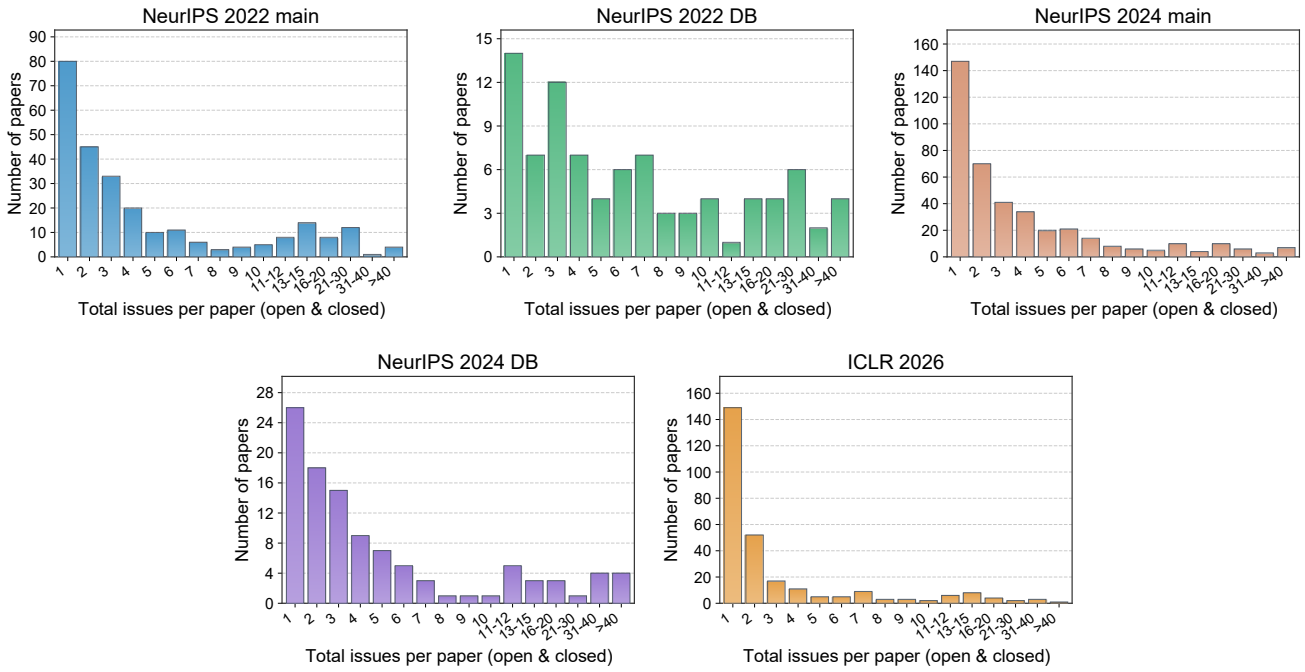

Figure 5: **Issue-Count Skew.** Per-split frequency distributions of hidden issue counts per paper, computed as the sum of open and closed reproduction issues for each issue-bearing paper. Zero-issue papers are omitted, and papers with more than 100 hidden issues are filtered before plotting.

later training/evaluation stage. This yields four categories:

- *Crash immediate*: the code fails at startup, e.g., broken imports, missing modules, or environment errors that prevent any run.
- *Crash late*: the code crashes mid-execution, e.g., shape mismatches, OOM, or runtime errors during training or evaluation.
- *Silent wrong setup*: the released setup does not match the paper's intended configuration, artifact scope, checkpoint, data provenance, or evaluation path, even when individual scripts execute without error.
- *Silent wrong number*: the code runs to completion but produces results that disagree with the paper's reported numbers.

C. Additional Evaluation Results

C.1. All-Finding Conference Comparison

Table 7 repeats the conference comparison in Table 4 after removing the top-10 reporting budget. It communicates that the main top-10 results capture nearly all of the matched issues: allowing all findings changes the reported rates only minimally.

Conference	Issue Match		Paper Match (Any)		Paper Match (All)		Avg. Cost
	EM	SM	EM	SM	EM	SM	
NeurIPS 2022 main	23.8%	57.4%	61.2%	89.7%	15.2%	41.4%	$1.28
NeurIPS 2022 DB	18.2%	51.0%	60.2%	96.6%	10.2%	31.8%	$1.50
NeurIPS 2024 main	24.0%	56.4%	61.0%	93.3%	22.7%	51.1%	$1.30
NeurIPS 2024 DB	12.2%	47.3%	46.7%	87.6%	10.5%	33.3%	$1.53
ICLR 2026	25.4%	58.1%	59.7%	89.7%	34.8%	65.6%	$1.26

Table 7: **All-Finding Conference Comparison.** Diagnostic all-finding conference comparison for Codex with GPT-5.5, using the same metrics as in Table 4 but without the top-10 cutoff.

Conference	Uncached Input	Cached Input	Output	Total Tokens	Cost
NeurIPS 2022 main	103,922	851,657	11,215	966,794	$1.28
NeurIPS 2022 DB	118,001	1,095,238	12,136	1,225,375	$1.50
NeurIPS 2024 main	112,126	801,607	11,335	925,068	$1.30
NeurIPS 2024 DB	126,678	1,019,991	13,029	1,159,698	$1.53
ICLR 2026	98,834	832,592	11,586	943,011	$1.26

Table 8: **Token Usage and Cost.** Codex with GPT-5.5 token usage and estimated cost per evaluated paper-snapshot pair, averaged within each conference. Total Tokens sums uncached input, cached input, and output tokens.

C.2. Token Usage and Cost

Table 8 expands the token and cost columns in Table 4. The table shows that Datasets and Benchmarks tracks require more tokens and cost more per run than the corresponding main-conference splits, consistent with larger or more complex artifact structures.

C.3. Ablation on Output Ordering

We instruct the model to output potential issues in descending order of the estimated likelihood that the issue would be encountered and reported by human users. In Figure 6, we ablate this design choice by comparing the top-k paper-level any-issue match rate with and without the ordering instruction. The ordering instruction **mainly improves the ranking quality under small reporting budgets**. For both exact and semantic matching, the model with the ordering instruction generally achieves higher match rates when only the first few findings are considered, indicating that it places human-reported blockers earlier in the report. As k increases, the two curves become close, especially when $k \geq 8$. This suggests that the instruction does not substantially change the total set of issues the agent is able to identify; rather, it helps prioritize the findings that are more likely to correspond to real user-reported reproduction blockers. This makes the ordered report more suitable for human triage under a limited inspection budget.

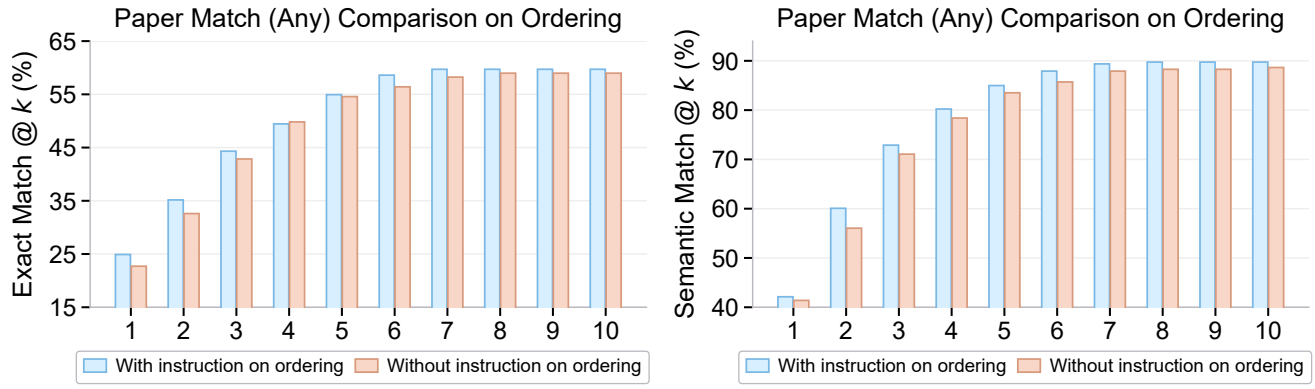

Figure 6: Ablation on the ordering instruction. We compare top-k paper-level any-issue match rates with and without instructing the agent to order findings by the estimated likelihood of being encountered and reported by human users. Left: exact match. Right: semantic match.

Configuration	Run	Issue Match		Paper Match (Any)		Paper Match (All)	
		EM@10	SM@10	EM@10	SM@10	EM@10	SM@10
Claude Opus 4.7 (Claude Code)	Run 1	21.7%	54.1%	52.0%	86.5%	27.8%	57.1%
	Run 2	23.3%	55.4%	54.6%	88.3%	30.0%	56.8%
	Run 3	23.1%	55.5%	53.5%	89.0%	28.6%	57.5%
	Mean±std	22.7%±0.85%	55.0%±0.76%	53.4%±1.29%	87.9%±1.32%	28.8%±1.12%	57.1%±0.37%
GPT-5.5 (Codex)	Run 1	25.4%	58.1%	59.7%	89.7%	34.8%	65.6%
	Run 2	24.5%	56.4%	58.2%	89.0%	33.0%	61.5%
	Run 3	25.2%	59.9%	57.1%	90.5%	33.3%	66.7%
	Mean±std	25.0%±0.50%	58.1%±1.78%	58.4%±1.29%	89.7%±0.73%	33.7%±0.97%	64.6%±2.70%

Table 9: **Run-to-run variance.** Three independent runs on the ICLR 2026 set, using the same metrics as Table 3. Run 1 of each configuration is the run reported in Table 3. The small standard deviations indicate that the reported match rates are stable across runs.

C.4. Run-to-Run Variance

Agent outputs are stochastic, so a natural question is whether the match rates in Table 3 are stable across repeated runs or reflect the luck of a single sample. To quantify this, we repeat the full ReproRepo pipeline three times, independently, for the two Claude Code and Codex configurations with the strongest model in each agent family, namely Claude Opus 4.7 and GPT-5.5. The first run of each configuration corresponds to the numbers reported in Table 3. Table 9 lists each of the three runs separately together with their mean and sample standard deviation.

Run-to-run variance is small across every metric. The standard deviation is at most 2.7 percentage points and is typically below 1.5 points. Crucially, the run-to-run spread is much smaller than the gaps between configurations and between EM and SM in Table 3. Therefore, none of the qualitative conclusions in Section 5, including the large EM–SM gap, the high paper-level any-issue coverage near 90% under semantic matching, and the ranking of configurations, depend on the particular sampled run, verifying the robustness of our claims.

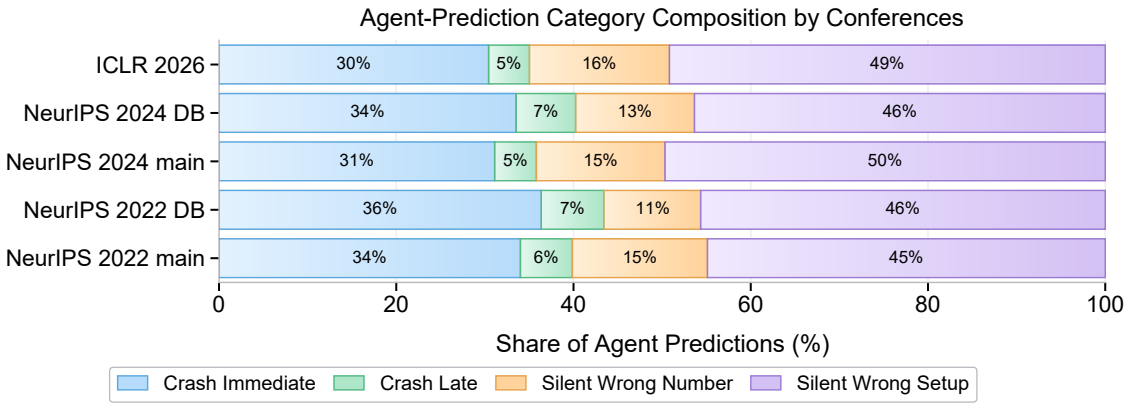

Figure 7: **Agent Prediction Taxonomy Analysis.** The distribution of agent-predicted issue categories across conference splits.

C.5. Human Validation of Alignment Judge Labels

Because ReproRepo uses an LLM judge to compare agent findings against real GitHub issues, we additionally perform a human validation study to check the quality of the judge. We sample 150 finding-issue pairs where the findings are generated by Codex with GPT-5.5, with 50 examples from each automatic label: exact match, semantic match, and none. For each pair, a human annotator is shown the agent-predicted issue, the candidate GitHub issue, and the judge rationale with the automatic label hidden, and assigns one of the same three labels.

Among the 50 pairs labeled EM by the judge, the human annotator labels 44 as EM, indicating high agreement under the strictest label. Among the 100 broadly matched pairs (including both EM and SM), the human annotator labels 84 as matched and 16 as none. Conversely, among the 50 pairs labeled None by the judge, the human annotator labels 44 as None and 6 as SM. For the matched-vs.-unmatched decision, the human annotation agrees with the LLM judge on 128 of 150 examples (85.3%). In addition, among the 90 examples that the human annotator labels as matched, the judge recovers 84. Thus, the match decision has 84.0% precision and 93.3% recall within the validation sample. Therefore, the human labels support the reliability of the LLM judge.

D. Agent Prediction Taxonomy

The following figures apply the same taxonomy breakdown applied to human issues to agent predictions. Together, they show how the breakdown of agent findings compares with that of the human-raised GitHub issues, and how this distribution changes when predictions are restricted to exact or semantic matches.

Figure 7 shows the category distribution across all agent predictions: from static inspection, the agent most commonly surfaces issues related to *silent wrong setup*, followed by *crash immediate*, whereas both silent wrong number and *crash late* are less commonly predicted. This highlights that along the two axes, the crash vs. silent distinction is much less salient in agent predictions than the setup vs. later-stage distinction. While it consistently identifies slightly more silent failures than crash failures, it predominantly predicts setup/immediate failures compared to number/late failures.

Figure 8 restricts the distribution to predictions that exactly match a hidden human issue, and Figure 9

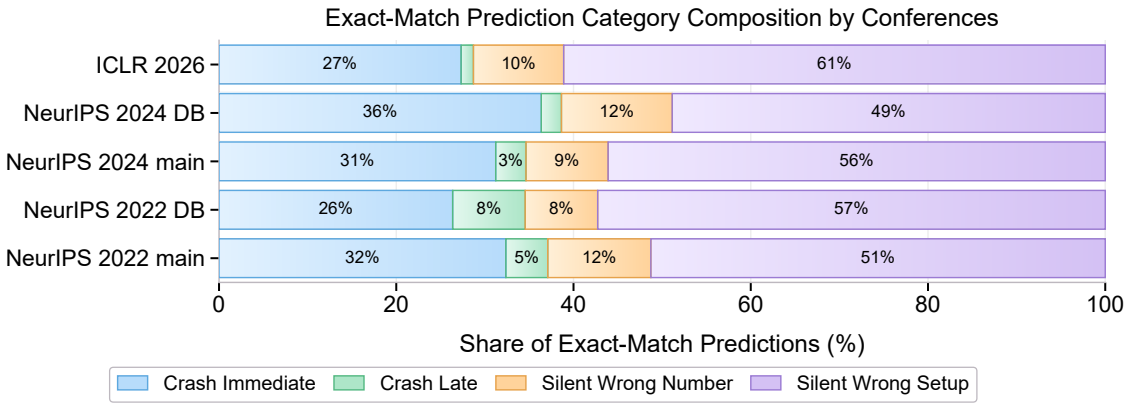

Figure 8: **Exact-Match Agent Prediction Taxonomy Analysis.** The distribution of exact-match agent-predicted issue categories across conference splits.

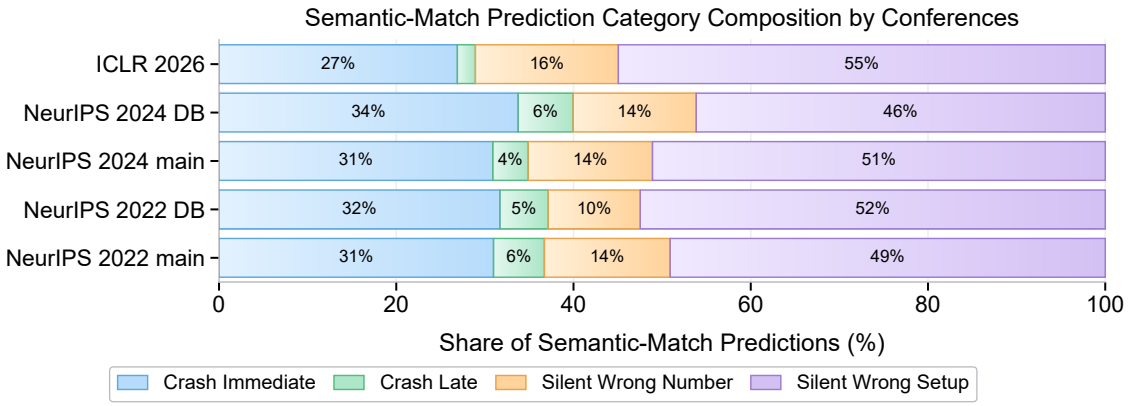

Figure 9: **Semantic-Match Agent Prediction Taxonomy Analysis.** The distribution of semantic-match agent-predicted issue categories across conference splits.

restricts to semantically matched predictions. In general, these show a similar pattern as Figure 7.

E. Prompt Templates

E.1. Agent Prompt

```
Generate a blind, static, no-execution reproducibility issue discovery result
for the provided research paper PDF and repository snapshot.

Blind audit rules:
- Use only the provided paper PDF and the local repository files in this
  workspace.
- Do not inspect GitHub issues, pull requests, discussions, comments, commit
  history, remote branches/tags, internet search, external websites, hidden
  benchmark metadata, or any repository state outside this snapshot.
- Do not run setup, tests, scripts, notebooks, downloads, training, inference,
  evaluation, or benchmark commands.

Goal:
Identify GitHub-issue-style reproduction blockers that a real user would be
likely to report when trying to reproduce the paper from this repository
snapshot.

Focus on concrete static evidence:
- missing or ambiguous data/checkpoint/model artifacts
- broken or incomplete installation/dependency instructions
- missing scripts/configs/paths referenced by the paper or README
- paper/repository mismatches
- evaluation/training workflow gaps
- result/metric provenance gaps
- code/config patterns that would cause reproduction-time crashes or silent
  wrong numbers

Prefer atomic findings: each finding should describe one user-facing blocker,
its trigger, root cause, static evidence, and impact on reproducing the paper.

Order findings by expected human-report likelihood, highest first. Prioritize
issues a normal reproducer would actually hit and open as a GitHub issue:
immediate blockers, documented workflow failures, missing required artifacts,
misleading instructions, common setup/data/model paths, and result mismatches.
Put broad audit observations, speculative risks, and low-actionability issues
later or omit them.

Use these finding categories when possible:
- environment_setup
- dependency_version
- data_access
- model_weights
- training_reproduction
- evaluation_reproduction
- runtime_error
- hardware_backend
- api_service
- result_mismatch
- documentation_gap
- paper_repo_mismatch

Write exactly one required artifact:
- ./findings.json

Required ./findings.json schema:

{{
  "reproducibility_assessment": "...",
  "findings": [
    {{
      "finding_id": "F01",
```

```
      "rank_reason": "Why this issue is likely to be reported by a human reproducer before lower-ranked
      ↪  findings.",
      "severity": "critical|high|medium|low",
      "stage": "install/environment|data/model
      ↪  acquisition|training/fine-tuning|inference/demo|evaluation/metrics|result
      ↪  comparison/provenance",
      "category": "environment_setup|dependency_version|data_access|model_weights|training_reproduction|e
      ↪  valuation_reproduction|runtime_error|hardware_backend|api_service|result_mismatch|documentation
      ↪  _gap|paper_repo_mismatch",
      "issue_title": "...",
      "user_symptom": "...",
      "trigger_context": "...",
      "root_cause": "...",
      "evidence": [
        {{
          "path": "...",
          "line_refs": "...",
          "evidence_type": "file|command|missing_artifact|documentation|config|paper_repo_mismatch",
          "summary": "..."
        }}
      ],
      "impact": "...",
      "affected_claims_or_metrics": ["..."],
      "reproducibility_blocker": true,
      "confidence": "high|medium|low"
    }}
  ],
  "missing_or_ambiguous_artifacts": ["..."]
}}

The JSON finding fields should contain enough information for a separate judge
to compare them against hidden GitHub issues by symptom, trigger, root cause,
evidence, and impact. If you find no concrete reproduction blockers, write an
empty findings array.
```

E.2. Judge Prompt

```
You are an alignment judge for a reproducibility benchmark.

You will receive exactly one file: ./task.json. It contains hidden GitHub issue
benchmark cases and blind agent findings for the same paper/repository snapshot.

Allowed information:
- The hidden issue cases in ./task.json.
- The blind agent findings in ./task.json.

Forbidden information:
- Do not browse the web.
- Do not inspect GitHub issues, pull requests, commits, discussions, release
  pages, external websites, or repository files.
- Do not run commands other than reading ./task.json and writing outputs.
- Do not infer from issue numbers, URLs, PR numbers, or commit hashes.

Leakage policy:
- The discovery agent did not see hidden issue text, comments, PRs, commits, or
  gold answers. Judge only after discovery is complete.
- For closed_gold cases, any provenance-only fix evidence is optional
  adjudication support. Do not require the blind finding to mention patch files
  or post-fix implementation details unless the hidden issue itself identifies
  those details as the user-facing reproduction problem.

Your job:
For each hidden issue, decide whether the best matching agent finding is:
- "exact": substantially the same user-facing reproduction problem.
  It should align on the main symptom and at least most of trigger/workflow,
  root cause, involved artifact, and impact. Wording can differ.
- "semantic": a related blocker or the same workflow area is identified, but
  with incomplete specificity. The Codex finding and GitHub issue share a
  concrete reproduction context, but do not describe the exact same blocker.
- "none": no agent finding would be useful evidence that the agent discovered
  this hidden issue.

Comparison dimensions:
- user-facing symptom
- trigger path / workflow
- root cause
- involved artifact, file, command, config, dataset, model, or metric
- impact on reproducing the paper

Also judge each agent finding against all hidden issues so downstream summaries
can estimate over-reporting. A finding may match zero, one, or multiple hidden
issues if it genuinely covers multiple human reports.

Write exactly one required artifact:
- ./result.json

Required JSON schema:
{{
  "task_id": "{task['task_id']}",
  "snapshot_group_id": "{task['snapshot_group_id']}",
  "issue_alignments": [
    {{
      "hidden_issue_id": "string from task.hidden_issues[].hidden_issue_id",
      "target": "open|closed_gold|closed_silver",
      "issue_number": 0,
      "best_label": "exact|semantic|none",
      "best_finding_id": "string or null",
      "best_finding_index": 0,
      "dimension_labels": {{
        "user_facing_symptom": "exact|semantic|none",
        "trigger_path_or_workflow": "exact|semantic|none",
        "root_cause": "exact|semantic|none",
```

```
        "involved_artifact": "exact|semantic|none",
        "impact": "exact|semantic|none"
      }},
      "rationale": "short explanation grounded in both the hidden issue and finding",
      "matched_evidence": ["short strings"]
    }}
  ],
  "finding_alignments": [
    {{
      "finding_id": "string",
      "finding_index": 0,
      "best_label": "exact|semantic|none",
      "matched_hidden_issue_ids": ["zero or more hidden issue ids"],
      "matched_targets": ["open|closed_gold|closed_silver"],
      "rationale": "short explanation"
    }}
  ],
  "notes": ["optional short notes about ambiguity"]
}}

Use only the three allowed labels exactly as written. If there are no hidden
issues or no findings, still write valid arrays.
```

F. Agent Output Examples

In this section, we provide concrete examples on exact match, semantic match, and no match.

F.1. Exact match example

Human GitHub issue

real data test human report

GitHub issue

Is the test command wrong?

GitHub discussion

`@dongruoyu1` I believe so ("test in 5.2 is incorrect"). At least the script didn't work. 5.2 Real Test script looks like a direct copy of the simulation one but doesn't work on real data.
e.g.

```
python test.py -template mst_s -outf ./exp/mst_s/
-method mst_s -pretrained_model_path ./model_zoo/mst/mst_s.pth
```

But the **format** of real measurements is different from simulation so the real test code could not load it. When I ran the aforementioned code, I got this error: #36

Agent output

Documented real-test commands do not match `real/test_code/test.py` and use an unreachable hard-coded checkpoint. agent finding

User symptom

Running the README's real testing command with `-template`, `-outf`, `-method`, and `-pretrained_model_path` is rejected by `argparse`, and editing around that still leads to missing `./Data` inputs and a hard-coded `/data/lj/...` checkpoint path.

Trigger context

Follow README section "4. Real Experiement" and run a listed command from `repo/real/test_code/`, for example:

```
python test.py -template mst_s -outf ./exp/mst_s/
-method mst_s -pretrained_model_path ./model_zoo/mst/mst_s.pth
```

Root cause

`repo/real/test_code/test.py` is a standalone stale script. It defines a different argument interface, hard-codes CUDA device 0, hard-codes default data under `./Data`, ignores the README `model_zoo` path, and loads a developer-local absolute checkpoint path.

Impact

The documented real-data inference path cannot be used to reproduce the paper's real reconstruction results without rewriting the script and supplying undocumented local artifacts.

This is an exact match: the LLM predicts **exactly the same trigger command** as the one discussed in the GitHub issue, highlighted in blue above, and recovers the same user-facing workflow and practical blocker.

F.2. Semantic match example

Human GitHub issue

Inference of DAT model human report

GitHub issue

Hi, thank you for your work of 'Discrete Adversarial Training'
I was trying to use the pre-tained model for inference. The [ViT-B/16] works pretty well. However, the [MAE-ViT-H] doesn't work. Could you tell the difference between this two model when testing them on the custom images?
For the [MAE-ViT-H], I used `vit_huge_patch14_224` instead of `mae_vit_huge_patch14` to initialize the model in `timm`. (the later can not be found in `timm` model zoo.)
I normalize the input image with `T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))`
Thank you !

Agent output

No training recipe is provided for the headline MAE-H DAT fine-tuning result. agent finding

User symptom

A user trying to reproduce the reported 31.40 ImageNet-C mCE / 32.77 Stylized-ImageNet MAE-H result can only download an already trained checkpoint; they cannot reproduce the fine-tuning run from documented commands or local configs.

Trigger context

Use the paper abstract/Table 1 or DAT README pretrained-weights table and look for the corresponding MAE-H fine-tuning command/config.

Root cause

The only DAT training command in the README trains `vit_base_patch16_224` and does not specify `mae_vit_huge_patch14`, a MAE initialization checkpoint, MAE-specific input size/crop settings, or fine-tuning hyperparameters. The MAE-H section is evaluation-only.

Impact

The paper's most prominent reproducibility target cannot be regenerated from the repository snapshot; users can only verify a released final checkpoint if the external URL is available.

This is a semantic match: both sides identify the same MAE-H DAT model artifact as a reproduction-critical object, but the human issue is an inference-time model-name/import failure whereas the agent finding is about the missing fine-tuning recipe.

F.3. No match example

Agent output

DANN, ADDA, and SGADA training scripts import `wandb` but requirements omit it. agent finding

User symptom

After installing the documented requirements for UFLD-DANN, UFLD-ADDA, or UFLD-SGADA, running `train.py` would fail at startup with `ModuleNotFoundError: No module named 'wandb'` unless `wandb` was installed separately.

Trigger context

Follow any of the DANN/ADDA/SGADA READMEs: `pip install -r requirements.txt`, then `python train.py configs/...`.

Root cause

The train scripts import and use Weights & Biases, but the corresponding `requirements.txt` files list only `opencv-python`, `tqdm`, `tensorboard`, `addict`, `sklearn`, and `pathspec`.

Impact

The installation instructions are incomplete for three reported UDA baselines, blocking reproduction before any data or model logic runs.

The agent finding is still a plausible reproducibility issue. However, this kind of missing-dependency problem is often easy for users to resolve locally without opening a GitHub issue. No existing GitHub issues match this agent finding.