

ToolMisuseBench: An Offline Deterministic Benchmark for Tool Misuse and Recovery in Agentic Systems

Rista Baral

University of Delaware, Newark, DE, USA
rbaral@udel.edu

Akshey Sigdel

Independent Researcher, USA
sigdelakshey@gmail.com

Abstract

Reliable tool use is a fundamental capability for agentic systems, yet existing evaluations primarily focus on successful tool execution and provide limited insight into how agents recover from tool failures. To address this gap, we introduce ToolMisuseBench, an offline deterministic benchmark for evaluating tool misuse and fault-conditioned recovery. ToolMisuseBench evaluates recovery behavior across domains, fault types, and difficulty levels, using output-grounded success criteria and typed recoverability metadata. The seed-42 release includes 6,000 training, 900 development, and 1,200 public-test tasks covering CRUD, retrieval, file operations, and scheduling, including failures such as timeouts, rate limits, hidden first-call schema perturbations, authorization errors, and adversarial responses. Using the benchmark, we evaluate deterministic reference agents together with local 3-4B tool-calling LLMs (Qwen3-4B, Ministral-3B, and Llama-3.2-3B). On the public-test split, Parser+Repair achieves 0.767 task success and 0.736 conditional recovery success, compared with 0.158 task success when recovery is disabled. On a shared stratified 216-task diagnostic panel, Qwen3-4B, Ministral-3B, and Llama-3.2-3B achieve task success rates of 0.796, 0.903, and 0.773, respectively. Fault-level analysis reveals distinct recovery weaknesses that overall task success alone fails to capture. ToolMisuseBench provides a controlled and reproducible framework for diagnosing tool-use recovery under frozen fault plans, enabling more rigorous evaluation of robust tool-using agents beyond execution accuracy.

1 Introduction

Tool-integrated language agents increasingly rely on external tools to perform real-world tasks. Although recent language models have made remarkable progress in language understanding and reasoning, reliable tool use remains a significant chal-

lenge. Operational failures such as malformed arguments, interface drift, authorization denial, transient execution errors, and brittle retry behavior can prevent otherwise capable agents from completing a task. Robustness therefore depends not only on selecting the right tool, but also on recognizing failures and recovering from them effectively.

Existing tool-use benchmarks primarily evaluate successful execution under nominal conditions, providing limited insight into how agents detect, respond to, and recover from operational failures. To address this gap, we introduce **ToolMisuseBench**, an offline, deterministic, model-agnostic, and dataset-first benchmark for evaluating failure-aware tool use. Rather than focusing solely on execution success, ToolMisuseBench evaluates whether agents can recover from realistic operational failures and still complete the intended task. The benchmark combines deterministic simulators, declarative fault injection, bounded execution episodes, output-grounded success criteria, and reproducible evaluation to enable controlled analysis across diverse failure conditions.

ToolMisuseBench is constructed using a stratified design that combines task domains, difficulty levels, and fault types, allowing balanced and interpretable evaluation across fault conditions. Because every task is executed in a deterministic environment, observed performance differences can be attributed to agent behavior rather than variability in external services or hidden differences in dataset composition.

Our contributions are fourfold. First, we introduce a benchmark generation framework with auditable stratification across task domains, fault types, and difficulty levels, enabling controlled and reproducible evaluation of recovery behavior. Second, we propose output-grounded success criteria that verify task completion based on returned values, final system state, or observable side effects rather than successful tool invocation alone.

Third, we provide typed recoverability annotations that distinguish transient failures, schema perturbations, authorization failures, and ambiguous feedback, enabling fine-grained analysis of recovery under different fault conditions. Finally, we release reproducible reference agents and a local LLM evaluation pipeline that establish transparent baselines and demonstrate how ToolMisuseBench distinguishes different recovery and failure profiles.

2 Related Work

Existing benchmarks for language models and agents primarily evaluate broad capabilities, including language understanding, reasoning, and end-to-end task completion. HELM and BIG-bench assess foundation models across diverse language tasks (Liang et al., 2023; Srivastava et al., 2023), while AgentBench, WebArena, Mind2Web, and GAIA extend evaluation to interactive environments, web navigation, and tool-assisted tasks (Liu et al., 2023; Zhou et al., 2023; Deng et al., 2023; Mialon et al., 2023). These benchmarks provide valuable measures of overall agent capability, but their emphasis on end-to-end performance makes it difficult to determine why an agent fails. Failures may arise from task understanding, tool selection, malformed arguments, interface drift, authorization failures, transient execution errors, or ineffective recovery, yet these factors are rarely evaluated in isolation.

A complementary line of work focuses specifically on tool use. Toolformer, ReAct, Gorilla, API-Bank, and ToolLLM/ToolBench study API selection, tool invocation, and tool-use training (Schick et al., 2023; Yao et al., 2023; Patil et al., 2023; Tang et al., 2023; Qin et al., 2023), while τ -bench and ToolSandbox introduce stateful execution, domain constraints, and multi-turn interactions (Yao et al., 2024; Lu et al., 2024). More recently, ToolEmu, ToolMaze, and ToolBench-X have shifted their attention toward reliability by evaluating agent behavior under tool failures, interface changes, and misleading feedback (Ruan et al., 2023; Zhu et al., 2026; Tian et al., 2026).

Among these, ToolMaze and ToolBench-X are most closely related to our work because they explicitly study recovery under structured failure conditions. ToolMaze focuses on graph-based replanning across perturbed tool graphs, whereas ToolBench-X evaluates recovery from hazards in multi-step workflows (Zhu et al., 2026; Tian et al., 2026). ToolMisuseBench addresses a complemen-

tary evaluation objective: controlled attribution of recovery behavior using frozen task records, fixed fault timing, output-grounded criteria, and per-episode traces. Its cross-product construction over domains, faults, difficulty levels, and budgets supports comparisons under matched, replayable conditions without implying superiority over these broader recovery settings. Table 1 summarizes these differences in experimental design.

3 Benchmark Formulation

Each benchmark task is stored as a structured record that specifies the task instruction, tool schemas, initial environment state, success criteria, execution budget, fault configuration, random seed, and optional reference summary. Evaluation proceeds as an episode where an agent observes the instruction, available tool schemas, interaction history, remaining budget, and the most recent error before selecting the next tool action.

Unlike evaluations that measure only whether an agent invokes the expected tool, ToolMisuseBench evaluates completion using *output-grounded* success criteria. Depending on the task, successful completion may require an agent to return a specific document identifier, produce the correct file contents, satisfy a validation check, or meet another structured output condition. This design prevents agents from receiving credit for correct tool selection when the underlying task objective has not been achieved.

For each fault family f , we measure fault-conditioned task success as

$$S_f = \frac{1}{|\mathcal{D}_f|} \sum_{i \in \mathcal{D}_f} \mathbb{1}[\text{TaskSuccess}_i = 1], \quad (1)$$

where $\mathcal{D}_f$ denotes the set of evaluation tasks whose primary fault belongs to family f . The same partition is used to compute faulted-task performance, while conditional recovery success is evaluated only over episodes in which an injected fault is actually observed. These fault-specific metrics reveal how recovery behavior changes across failure modes, which cannot be captured by aggregate task success alone.

3.1 Metrics

We report several metrics to separate task completion, recovery ability, and fault activation. *Task success* is achieved when all output-grounded success

Table 1: Comparison of ToolMisuseBench with ToolMaze and ToolBench-X, two closely related tool-use recovery benchmarks.

Benchmark	Perturbation locus	Recovery abstraction	Main controlled factor	Reported evaluation evidence
ToolMaze (Zhu et al., 2026)	Pre-specified perturbations at tool-graph nodes.	Recovery through dynamic re-planning and alternative execution paths.	Task-graph complexity (C1–C4) and perturbation type.	Task success, plan recovery, and recovery cost.
ToolBench-X (Tian et al., 2026)	Tool-environment hazards in multi-step workflows.	Recovery from tool-execution hazards.	Sequential, parallel, and mixed workflows.	Canonical final-answer completion.
ToolMisuseBench	Frozen first-call fault plans with fixed timing.	Typed recoverability metadata distinguishing transient, one-shot schema, denial, and ambiguous conditions.	Cross-product generation over domain, fault, difficulty, and budget.	Output-grounded success, conditional recovery success, and per-task audit traces.

criteria are satisfied for an episode. *Conditional recovery success* measures the fraction of successful episodes among those where an injected fault is observed, isolating an agent’s ability to recover after encountering a failure. In contrast, *fault exposure* measures the fraction of declared-fault task records in which the configured fault is observed during execution.

We additionally report *recovery coverage*, which measures the fraction of declared-fault records where the fault is both observed and ultimately recovered, and is equivalent to the product of fault exposure and conditional recovery success. Together, these metrics distinguish whether an agent fails because it cannot recover from a fault or because the fault was never encountered. We also report *faulted-task success*, defined as task success over all declared-fault task records, along with the *invalid-call rate*, defined as number of invalid calls divided by total tool calls within an episode and averaged across episodes.

Beyond primary performance metrics, the benchmark records several diagnostic statistics that characterize execution behavior. The legacy trace field `PolicyViolations` aggregates invalid tool calls and authorization-denied events. Because it does not distinguish between these event types, it is treated only as trace information and is not analyzed in this paper. *Time to recovery* measures the number of steps from the first observed fault to the first subsequent error-free tool call. *Catastrophic failure* is an artifact-level diagnostic triggered by budget exhaustion, three or more invalid tool calls, or final task failure. Similar to `PolicyViolations`, catastrophic failure is retained for trace analysis rather than used as a primary evaluation metric.

Finally, every benchmark task specifies an explicit execution budget for controlled comparison under different resource constraints. ToolMisuseBench provides three default evaluation profiles:

tight, standard, and relaxed, with maximum tool-call budgets of 3, 6, and 10, respectively. Since these limits are encoded directly in the task specification, recovery strategies can be evaluated under fixed and reproducible interaction budgets.

4 Dataset Design

The dataset is generated by composing task domains, fault families, difficulty levels, and execution budget profiles. The supported domains include CRUD, retrieval, file operations, and scheduling. Fault families consist of clean execution, timeout, rate limit, hidden first-call schema perturbation (serialized as `schema_drift`), authorization, and adversarial error conditions. Every task is assigned one of three difficulty levels (easy, medium, or hard) and one of three execution budget profiles (tight, standard, or relaxed).

The canonical diagnostic cycle contains 216 tasks, while the seed-42 release contains 6,000 training, 900 development, and 1,200 public-test tasks. The public test split contains 400 tasks for each budget profile, including 600 tasks with output-grounded success criteria that verify structured return values. The remaining 600 tasks evaluate final-state or side-effect conditions using operators such as `equals`, `in`, and `map_value_equals`, sometimes together with a successful tool-call requirement. In all cases, successful tool invocation alone is insufficient for task completion. Because task templates admit different numbers of valid instantiations, the released dataset is not perfectly balanced: domain counts range from 270–324 and fault-family counts from 198–207, and the benchmark reports the resulting distributions rather than presenting the design target as exact balance.

Each faulted task includes a primary recoverability tag that describes the expected behavior of the injected fault. Timeout and rate-limit faults represent transient failures. In the current bench-

Fault(s)	Tag	Configured behavior and expected response
Timeout / rate limit	Transient	If the first episode call targets the configured tool, it returns a timeout or rate-limit error; retry the compatible call within the episode budget.
Hidden schema perturbation	One-shot validation mismatch	If the first episode call targets the configured tool, it undergoes a hidden backend argument rename; no revised schema is exposed. A validation error is returned, and the perturbation is absent on later calls. The release does not score a particular argument-repair strategy.
Authorization	Transient denial	If the first episode call targets the configured tool, it receives one injected denial, not a persistent permission model. A retry may succeed because the benchmark configuration makes the denial one-shot; this behavior should not be generalized to real permission failures.
Adversarial error	Ambiguous transient	If the first episode call targets the configured tool, an opaque error is paired with a transient failure; treat feedback as untrusted, preserve task constraints, and recover without an unsafe side action.

Table 2: Recoverability contract. Tags describe the configured injected event, not a claim about every real deployment failure.

mark configuration, authorization is modeled as a one-shot transient denial, while `schema_drift` represents a hidden first-call schema perturbation, and adversarial errors correspond to transient failures with ambiguous feedback. Composite tasks may contain multiple fault records, but only the first declared fault determines the primary recoverability tag. These tags describe configured fault semantics, support grouped analysis, and are not independently crossed with every fault family. For compatibility with earlier releases, the dataset also retains the legacy `repairable` field, although this paper does not interpret it as indicating persistent interface evolution or requiring a particular argument-repair strategy. The benchmark evaluates whether a task satisfies the output-grounded success criteria rather than whether an agent follows a particular recovery strategy. Accordingly, these tags describe only the behavior configured within the benchmark and should not be interpreted as general properties of real-world systems.

End-to-End Example. Consider a tight-budget retrieval task where the agent is asked to search for “retention incident 8.” The frozen environment contains the target document `dt21071`, and the available `search_docs(query, top_k)` tool is configured to return a timeout on its first invocation. After observing the timeout, the correct recovery is to reissue the search request. The task succeeds only if the search completes successfully and `results.0.id` matches `dt21071`. Therefore, a

syntactically valid tool call that returns an incorrect document does not satisfy the success criteria.

All dataset splits are generated from the same visible domain scenario families using split-specific deterministic offsets. Consequently, the public-test split evaluates distinct frozen task records, but does not establish generalization to unseen task structures. The benchmark therefore reports an instruction-uniqueness ratio of 0.728 and does not claim held-out-template or semantic-overlap generalization. This design choice limits structural generalization claims while preserving reproducible diagnosis and evaluation on fixed benchmark instances.

Task Validation. Before release, every task record is processed by an automated structural validation pipeline that verifies instructions, success criteria, domain-state structure, tool references, fault configurations, schema-perturbation fields, and execution budgets. The validator reports no structural inconsistencies across all 8,100 released tasks. This process ensures dataset consistency and schema correctness, but does not constitute a manual semantic review or a formal proof that every task has a unique reachable solution.

5 Implementation

5.1 Release Construction

ToolMisuseBench separates benchmark generation from evaluation to ensure reproducibility. During release construction, each task is stored as a frozen record containing instruction, simulator state, fault configuration, success criteria, execution budget, random seed, split assignment, and recoverability metadata. Each release also includes split files, a manifest, and a freeze file recording the benchmark configuration together with checksum information. During evaluation, tasks are instantiated directly from these frozen records rather than regenerated, ensuring that every system is evaluated under identical benchmark conditions.

5.2 Deterministic Execution

Evaluation is performed using deterministic simulators for the CRUD, retrieval, file, and scheduling domains. Each task includes a replayable fault plan that specifies when configured failures are injected during execution. Because both the simulator state and fault schedule are fully determined by the released task record and seed, evaluations are reproducible across agents and repeated runs. This deter-

ministic execution model allows observed performance differences to be attributed to agent behavior rather than variation in the evaluation environment.

5.3 Scoring and Artifacts

After each evaluation episode, the framework computes the benchmark metrics described in Section 3.1 and records a complete execution trace. It also exports structured score files, trace logs, benchmark summaries, and quality reports that summarize slice distributions, recoverability statistics, budget profiles, duplicate-task checks, and instruction-uniqueness measurements. Together, these artifacts support reproducibility, facilitate detailed error analysis, and enable independent verification of reported benchmark results.

6 Experimental Protocol

We evaluate ToolMisuseBench using both deterministic references and local LLM-based agents to study how different levels of tool-use capability affect failure handling and recovery. All agents are evaluated under the same benchmark conditions, including identical task records, execution environment, fault injection process, and scoring pipeline.

We evaluate five lightweight deterministic references on the 1,200-task public-test split: Heuristic, Schema-Repair, Policy-Aware, Parser-only, and Parser+Repair. These references are designed to isolate different aspects of tool-use behavior. The Heuristic and Schema-Repair references focus on basic argument handling, while the Policy-Aware reference adds simple handling for authorization-related feedback. Parser+Repair first extracts tool arguments from the natural-language instruction and executes the corresponding action. After a failure, it attempts limited corrections using only the information exposed by the error message, such as missing or invalid arguments; otherwise, it retries within the remaining budget. Parser-only uses the same initial parsing strategy but stops after the first failed call, providing a direct comparison of the value of recovery behavior.

We additionally evaluate three local instruction-following models: Qwen3-4B-GGUF, Ministral-3B-Instruct, and Llama-3.2-3B-Instruct. We select locally deployable 3–4B models to provide accessible, reproducible reference evaluations under a common serving stack, rather than to characterize language models broadly. All models are served through the same local OpenAI-compatible

llama.cpp interface using temperature zero, seed 42, a 128-token response limit, required function calling, and disabled reasoning output. Due to the computational cost of the LLM evaluation, we use a stratified 216-task diagnostic panel that preserves coverage across domains, fault families, difficulty levels, and budget profiles. The panel supports matched diagnostic comparisons within this experimental setting; it is not intended to establish broad model generalization. Deterministic references are re-evaluated on the same task identifiers to ensure that all direct comparisons use identical episodes and scoring conditions.

For the LLM evaluation, we estimate the uncertainty using 10,000 seeded percentile-bootstrap resamples over benchmark tasks. We also report paired bootstrap differences because all models are evaluated on the same task instances. These intervals reflect variation across sampled benchmark tasks rather than randomness from repeated model executions.

7 Results

Table 3 summarizes performance on the shared stratified 216-task diagnostic panel. The benchmark clearly separates simple rule-based baselines from agents capable of recovering after tool failures. The heuristic, schema-repair, and policy-aware references fail all tasks under output-grounded evaluation, indicating that generic schema filling and lightweight rules are insufficient. In contrast, Parser+Repair substantially improves performance, achieving 0.759 task success and 0.733 conditional recovery success. Among the local LLM baselines, Ministral-3B achieves the strongest overall performance (0.903 task success), followed by Qwen3-4B (0.796) and Llama-3.2-3B (0.773). Additionally, separate validation of Parser+Repair yields 0.767 task success and 0.736 conditional recovery success on the full 1,200-task public-test split, providing an additional measurement beyond the shared diagnostic panel; these values are not used for direct comparison with the LLM baselines.

Recovery metrics provide additional insight beyond aggregate task success. For example, Llama-3.2-3B achieves higher conditional recovery than Qwen3-4B once faults are observed, but its lower fault exposure reduces overall recovery coverage. This shows that recovery should be interpreted jointly with exposure and coverage rather than as a

Agent	Success	Recovery	Exposure	Coverage	Calls
Heuristic	0.000	0.000	0.378	0.000	2.83
Schema repair	0.000	0.000	0.378	0.000	3.24
Policy-aware	0.000	0.000	0.378	0.000	3.25
Parser-only	0.148	0.000	1.000	0.000	1.00
Parser+Repair	0.759	0.733	1.000	0.733	1.95
Qwen3-4B	0.796	0.756	1.000	0.756	1.83
Ministral-3B	0.903	0.883	1.000	0.883	1.87
Llama-3.2-3B	0.773	0.840	0.900	0.756	1.75

Table 3: Performance on the shared stratified 216-task diagnostic panel. Recovery denotes conditional recovery success, exposure is the fraction of declared-fault records where the configured fault is observed, and coverage measures successful recovery over all declared-fault records.

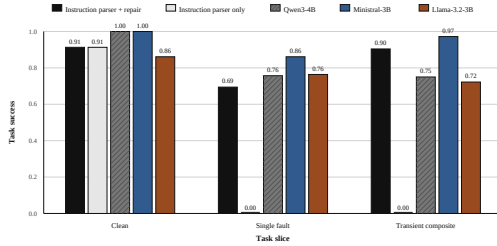


Figure 1: Recovery effect by task slice on the shared stratified 216-task diagnostic panel. Parser-only preserves the initial parse but stops after an error; Parser+Repair applies generic public-error corrections or retries. The transient composite is the release’s adversarial-feedback-plus-timeout plan.

standalone metric. Table 3 reports fault exposure and recovery coverage alongside conditional recovery. Bootstrap confidence intervals quantify task-sampling uncertainty: Ministral-3B consistently outperforms Qwen3-4B, while the difference between Qwen3-4B and Llama-3.2-3B is smaller and overlaps under task sampling uncertainty.

The recovery ablation highlights the importance of post-error behavior. Parser-only achieves 0.148 task success but fails every faulted task because it terminates after an error. In contrast, Parser+Repair reaches 0.759 success by applying repair and retry, with improvements observed across clean, single-fault, and transient-composite settings (Figure 1).

Fault-conditioned results reveal different failure patterns: Ministral-3B achieves highest point estimates on most fault families, while Qwen3-4B and Llama-3.2-3B show different weaknesses across authorization, rate-limit, and schema-perturbation failures (Figure 2). Because each fault slice contains only 36 tasks, individual estimates should be interpreted cautiously; however, the results consistently show that similar overall success does not

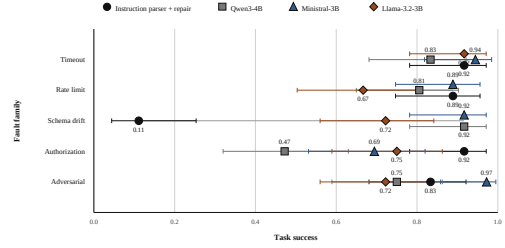


Figure 2: Fault-conditioned diagnostic profile on the shared stratified 216-task diagnostic panel. Markers show task success; horizontal whiskers are 95% Wilson intervals for each 36-task non-clean slice.

imply similar recovery capability.

Overall, these results demonstrate that aggregate task success alone is insufficient for evaluating reliable tool use. Fault-conditioned evaluation exposes meaningful differences in recovery behavior that remain hidden under a single aggregate metric, motivating the use of output-grounded and recovery-aware evaluation for future tool-use benchmarks.

8 Discussion

Our results highlight that successful task completion alone does not fully capture how tool-using agents behave when failures occur. Agents that achieve similar overall success rates can differ substantially in how they handle errors, recover from failures, and adapt their behavior under different fault conditions. These differences are hidden when evaluation focuses only on final task outcomes, motivating the need for metrics that consider fault exposure, recovery behavior, and performance conditioned on specific failure types.

ToolMisuseBench is designed to make these behaviors easier to analyze by providing the same tasks, tool interfaces, and fault conditions to every evaluated agent. Because failures are generated through deterministic and replayable fault plans, changes in performance can be examined as differences in agent behavior rather than differences in the evaluation setup. This controlled evaluation setting makes the benchmark useful for comparing existing approaches, testing regression after system changes, and studying how different retry, repair, and planning strategies affect recovery under identical failure scenarios.

9 Reproducibility and Artifacts

The dataset and accompanying artifacts are available on Hugging Face.¹ The benchmark implementation, evaluator, reference agents, experiment scripts, and analysis code are available on GitHub.² The benchmark uses fixed seeds and deterministic simulators, allowing the reported experiments to be reproduced without dependence on external services or changing execution environments.

The release includes frozen task instances, task identifiers, validation reports, baseline traces, and the scripts needed to regenerate the reported tables and figures. Execution traces preserve per-task details such as fault configurations, tool interactions, and recovery outcomes, allowing researchers to inspect agent behavior beyond aggregate performance metrics. Together, these artifacts provide a common evaluation framework for comparing future methods under the same task conditions, fault plans, and output-grounded success criteria.

10 Conclusion

We introduced ToolMisuseBench, a deterministic benchmark for studying how tool-using agents respond to failure during tool execution. Beyond measuring whether an agent eventually completes a task, the benchmark supports analysis of how it responds when tool failures occur through output-grounded evaluation and fault-conditioned diagnostics. Our experiments show that aggregate task success alone does not explain the distinct recovery profiles observed across fault types, indicating that overall success is insufficient for assessing reliable tool use. ToolMisuseBench offers a useful foundation for developing, comparing, and improving recovery-aware tool-using agents under reproducible evaluation conditions.

11 Limitations

ToolMisuseBench is designed for controlled diagnosis of tool-use reliability rather than direct evaluation of production behavior. The simulators capture task state, tool schemas, injected faults, and observable feedback, but they do not model all characteristics of real services, including evolving APIs, persistent authorization failures, authentication workflows, service-specific policies, or network variability. Therefore, results should be in-

terpreted as evidence of behavior under specified failure conditions rather than guarantees of real-world reliability. In particular, the hidden first-call schema perturbation represents a controlled robustness test and should not be interpreted as a complete model of persistent interface evolution.

This controlled setting is a key design choice of the benchmark, but it also defines its scope. Deterministic task instances and replayable fault plans allow recovery strategies to be compared under identical conditions, reducing confounding effects from external service variation. Future work can extend this analysis with stable API sandboxes or recorded service traces to study how benchmark findings transfer to more realistic environments.

Finally, our LLM evaluation covers three local quantized models under a single serving configuration and a stratified diagnostic panel. These results provide reproducible reference baselines for this setting, but should not be interpreted as conclusions about language models in general or generalization beyond the evaluated task records. The evaluation of larger models, alternative serving stacks, and additional recovery mechanisms remains an important direction. Although the benchmark covers multiple domains and fault families, some combinations are not perfectly balanced; future versions can explore stronger template separation and longer multi-step recovery scenarios.

12 Ethical Considerations

ToolMisuseBench uses synthetic, deterministic task environments and does not contain personal, user, or sensitive data. The benchmark evaluates agent behavior under configured simulator conditions; therefore, its results should not be interpreted as evidence of production safety or access-control compliance. In particular, the authorization failures included in the benchmark represent controlled, synthetic denial scenarios, and recovery behavior observed in these settings should not be generalized to real authorization workflows. The benchmark is intended to support safer evaluation and diagnosis of tool-using agents while encouraging careful interpretation of its scope and limitations.

References

Xiang Deng, Yu Gu, Boyuan Zheng, et al. 2023. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*.

¹<https://huggingface.co/datasets/sigdelakshey/ToolMisuseBench>

²<https://github.com/akgitrepos/toolmisusebench>

- Percy Liang, Rishi Bommasani, Tony Lee, et al. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*.
- Xiao Liu, Hao Yu, Hanchen Zhang, et al. 2023. Agent-bench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.
- Jiawei Lu et al. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.
- Grégoire Mialon, Clémentine Fourier, Thomas Wolf, et al. 2023. Gaia: A benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*.
- Rajkumar Patil, Tianjun Zhang, Luyu Wang, et al. 2023. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*.
- Yujia Qin, Shihao Liang, Yining Ye, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.
- Yubo Ruan et al. 2023. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, et al. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.
- Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.
- Yujin Tang, Shun Yang, Aixin Yao, et al. 2023. Apibank: A benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.
- Y. Tian, Z. Shi, and B. Zhao. 2026. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability. *arXiv preprint arXiv:2606.25819*.
- Shunyu Yao, Noah Shinn, Parisa Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.
- Shuyan Zhou, Frank F. Xu, Hao Zhou, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.
- D. Zhu et al. 2026. When tools fail: Benchmarking dynamic replanning and anomaly recovery in llm agents. *arXiv preprint arXiv:2606.05806*.