Beyond Success Rate: Benchmarking Robustness in Tool-Using Language Agents

Kristina Lewandowska

Gdańsk University of Technology
Gabriela Narutowicza 11/12, 80-222 Gdańsk, Poland

Abstract

The deployment of tool-using language agents in real-world scenarios is hindered by their brittleness in the face of unreliable tools and environments. While current benchmarks focus on task success in idealized conditions, they fail to measure **robustness**—the ability to handle and recover from failures. This paper introduces a novel benchmarking framework that moves *beyond success rate* by systematically injecting a taxonomy of realistic faults into established environments like ToolBench and WebArena. We evaluate state-of-the-art agents on new metrics such as Recovery Rate and Unrecoverable Failure Rate, revealing a significant robustness gap. Our results show that even powerful models like GPT-4 Turbo struggle with basic errors, with recovery rates below 60%. Furthermore, we demonstrate that adversarial fine-tuning on our synthetic failure data can significantly improve robustness, providing a clear pathway for developing more reliable and trustworthy agentic AI systems.

Introduction

The advent of large language models (LLMs) has catalyzed the development of **agentic AI**: systems that transcend static text generation to perform autonomous, multi-step reasoning and action. A cornerstone of this capability is **tool use**, where an LLM leverages external functions, APIs, and resources to execute tasks that lie beyond its internal knowledge or computational boundaries (Qin et al. 2023a; Patil et al. 2023). The potential of these language agents that use tools to transform fields such as software engineering, scientific discovery, and enterprise automation is immense. However, their path to reliable real-world deployment is fraught with a critical, yet under-explored, challenge: **robustness**. Current benchmarks and evaluations predominantly measure performance in idealized, fault-free environments, reporting a single metric of **task success rate**. This myopic focus fails to capture a system's behavior when its tools inevitably fail, return unexpected results, or become unavailable—a common occurrence in production settings. An agent that achieves a 95% success rate on a standard benchmark but catastrophically fails or behaves unpredictably upon encountering a simple API timeout is not a trustworthy agent.

In this paper, we argue that the research community must move *beyond success rate* to properly evaluate and ensure the trustworthiness of agentic systems. We define **robustness** for tool-using agents as the dual capacity for **fault tolerance**—the ability to correctly identify and handle execution errors without crashing or propagating incorrect states—and **effective recovery**—the capability to strategize and execute an alternative path to complete the task after a failure. Without explicit benchmarking and optimization for these properties, agents remain brittle and unsuitable for applications requiring high reliability. To address this gap, we introduce a novel benchmarking framework that systematically injects a taxonomy of realistic failures into established tool-use environments. Our benchmark shifts the evaluation focus from mere capability to reliability, measuring metrics such as **recovery rate**, **error identification accuracy**, and **unrecoverable failure rate**. By applying this benchmark to several state-of-the-art agents, we expose a significant *robustness gap* and provide a foundational tool for building agentic AI that is truly aligned with the principles of trust and control required for real-world deployment.

Literature Review

The foundation of this work rests on three pillars of research: the development of tool-using language agents, the creation of environments to evaluate them, and the nascent study of their robustness and failure modes.

The paradigm of augmenting LLMs with external tools has been rapidly advanced. Early work by (Parisotto et al. 2016) explored neural networks that interact with calculators, while (Yao et al. 2022) popularized the ReAct paradigm, which interleaves reasoning traces with actions, significantly improving task performance. Subsequent research has focused on improving the tool-use capability itself, with (Schick et al. 2023) demonstrating how LLMs can be trained to call APIs in a self-supervised manner, and (Patil et al. 2023) creating a model specifically fine-tuned for generating accurate API calls. The scope has expanded from single tools to orchestrated workflows, as seen in frameworks like (Qin et al. 2023a) and the use of LLMs for planning and executing complex action sequences (Hao, et al. 2024; Liu, et al. 2023).

To evaluate these capabilities, a suite of sophisticated benchmarks has emerged. (Shi et al. 2023) and (Zhou et al.

2023) provide realistic, scalable environments for assessing web-based task completion. For general tool use, (Qin et al. 2023b) created ToolBench, a large-scale collection of real-world APIs and instruction-based queries, which has become a standard for testing functional correctness. Beyond single-task performance, benchmarks like (Kim et al. 2023) and (Zhou, et al. 2024) have begun to explore multi-agent collaboration and long-horizon planning. These benchmarks represent a significant leap forward, but their primary evaluation metric remains the binary success/failure of the end task, operating under the assumption of a perfectly reliable tool environment(Li 2025b).

Acknowledging that agents will encounter errors, a small but critical body of work has started to probe reliability. (Valmeekam et al. 2023) and (Valmeekam, et al. 2024) extensively evaluated the planning capabilities of LLMs, revealing systematic failures even in abstract reasoning. (Wu, et al. 2023) highlighted the challenges of robustness in multi-agent conversations. Most closely related to our work, (Creswell, et al. 2023) and (Yang, et al. 2023) investigated failure modes in tool use, but their analyses were often limited to specific domains or a narrow set of errors. Recent studies have begun to formalize these concerns; for instance, (Wang, et al. 2024) surveyed safety risks, and (He, et al. 2024) provided a high-level overview of trust challenges. However, these are largely taxonomical or conceptual. Crucially, there remains a lack of a *systematic, general-purpose benchmark* for quantifying robustness. Existing evaluations do not proactively and comprehensively inject a wide range of failures (e.g., HTTP errors, malformed outputs, availability shifts) into established testbeds to measure an agent's response. Our work fills this gap by providing a unified framework for *fault injection* into tool-use environments, enabling a rigorous, comparative analysis of agent robustness that moves beyond the limitations of final task success rate and directly addresses the trust and control concerns of deploying agentic AI in the wild.

Methodology

The related work reveals a clear gap: while existing benchmarks like ToolBench (Qin et al. 2023b) and WebArena (Zhou et al. 2023) excel at measuring an agent's functional capability in pristine conditions, they offer no systematic framework for quantifying its **robustness**—its ability to withstand and recover from the unreliable tool interactions endemic to real-world deployment. To bridge this gap, we introduce a comprehensive methodology for benchmarking robustness through proactive fault injection. Our approach is not merely an extension of existing benchmarks but a fundamental re-orientation of the evaluation paradigm from passive observation of success to active stress-testing of failure modes. This section details our systematic process. First, we formalize a **Taxonomy of Failures** to categorize the perturbations we introduce. Second, we describe the **Fault Injection Mechanism** that operationalizes this taxonomy within established agent environments. Third, we define a set of **Robustness Metrics** that move beyond binary task success to capture the quality of an agent's response to adversity. Finally, we outline a **Model Improvement Framework** that leverages our benchmark to train and evaluate more robust agents, demonstrating the practical utility of our methodology for advancing the field of trustworthy agentic AI.

Taxonomy of Failures

A foundational contribution of our work is the establishment of a formal taxonomy that classifies potential failures in tool-using interactions. This taxonomy is crucial for moving beyond the ad-hoc analysis of errors found in prior work (Yang, et al. 2023; Creswell, et al. 2023) and provides a structured, comprehensive basis for our fault injection. We model a tool-use episode as a sequence where an agent, given a state s_t, executes a tool a_t with parameters θ_t, resulting in an observation o_t from the environment. A failure is any deviation from the expected observation $o_t^{expected}$ that challenges the agent's ability to proceed correctly. Our taxonomy categorizes these deviations into three primary classes. **Execution Errors** occur at the interface level and prevent the tool from returning a valid output; these are modeled as observations o_t^{error} belonging to a set of error codes E (e.g., $E = \{404, 500, \text{Timeout}\}$), formally defined as $o_t^{error} \in E$. **Semantic Anomalies** are more insidious, where the tool executes successfully but returns a result $o_t^{anomaly}$ that is malformed or contextually invalid. This includes empty results ($o_t^{anomaly} = \emptyset$), syntactically incorrect data (e.g., non-JSON strings from a JSON API), or semantically implausible values (e.g., a negative price). Finally, **Environmental Non-Stationarity** captures the dynamic nature of real-world systems, where a previously available tool a_t becomes permanently unavailable ($o_t = \text{Deprecated}$) or its functional specification changes. This taxonomy ensures our benchmark covers a wide spectrum of real-world challenges, providing a rigorous testbed for agent robustness.

Fault Injection Mechanism

To operationalize our taxonomy, we design a fault injection mechanism that acts as a proxy layer between the agent and its environment. This mechanism is a significant advancement over static benchmarks, as it programmatically induces failures to simulate a non-ideal world. The injector is parameterized by a fault profile $\mathcal{P} = (F, \lambda, \delta)$, where F is the set of failure types from our taxonomy, $\lambda : F \to [0, 1]$ is a probability distribution over failure types, and δ is the global fault injection rate. For each tool-call a_t made by the agent, with probability δ, the injector intercepts the call and, instead of executing it, returns a synthesized observation $o_t^{injected}$ sampled according to λ. For example, if $\lambda(\text{Timeout}) = 0.3$, there is a 30% chance that an injected fault will be a timeout error. This approach allows us to simulate complex scenarios, such as a specific tool being "flaky" by setting a high δ for that tool alone. The mathematical core of this process is a conditional observation function: $O(a_t, \theta_t, \mathcal{P}) = \begin{cases} \text{sample}(F, \lambda) & \text{with probability } \delta \\ \text{Env}(a_t, \theta_t) & \text{otherwise} \end{cases}$, where Env is the original environment's execution function. By integrating this mechanism into platforms like ToolBench and WebArena, we create a robustness-adjusted version of these

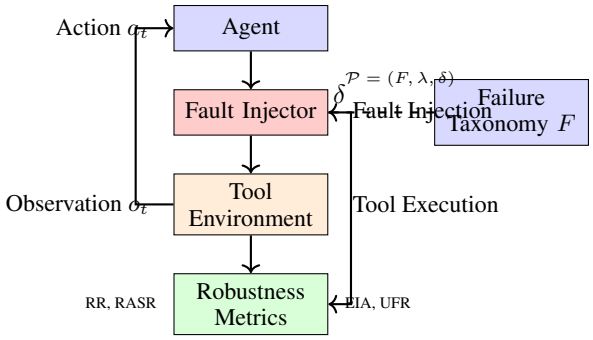

Figure 1: Proposed robustness benchmarking framewor

benchmarks, enabling reproducible and controlled stress-testing that was previously absent from the literature.

Figure 1 provides a high-level overview of our robustness benchmarking framework. The system operates as a modified agent-environment loop where a **Fault Injector** module sits between the agent and the tool environment. During execution, the agent selects an action a_t (e.g., a tool call), which is first processed by the fault injector. This module, parameterized by $\mathcal{P} = (F, \lambda, \delta)$, uses the failure taxonomy F and probability distribution λ to decide whether to inject a fault with probability δ. If a fault is injected, a synthetic failure observation is generated and sent directly to the robustness metrics evaluator; otherwise, the action is passed to the tool environment for normal execution, which returns an observation o_t. This observation is also fed back to the agent to continue its task and is simultaneously evaluated by the metrics module. The four core robustness metrics: Recovery Rate (RR), Robustness-Adjusted Success Rate (RASR), Error Identification Accuracy (EIA), and Unrecoverable Failure Rate (UFR), are computed based on the agent's responses to both normal and fault-injected observations. This closed-loop design enables systematic stress-testing of agents under controlled failure conditions, providing a comprehensive assessment of their resilience.

To operationalize the conceptual framework, Algorithm 1 provides the precise computational procedure for our benchmarking methodology. The algorithm takes as input an agent $\mathcal{A}$, an environment $\mathcal{E}$, and the fault profile $\mathcal{P}$ that defines the failure taxonomy F, probability distribution λ over failure types, and global injection rate δ. The core loop (lines 4-14) executes the agent-environment interaction for a maximum of T steps. For each step, after the agent selects an action a_t, the algorithm probabilistically decides whether to inject a fault (line 6). If injection occurs, a specific failure type f is sampled from the taxonomy according to λ, and a corresponding synthetic observation o_t is generated (lines 7-9); otherwise, the action is executed normally in the environment (line 11). The trajectory and injected fault records are maintained throughout execution. Finally, the COMPUTEMETRICS function (line 16) analyzes the complete interaction history to calculate the robustness metrics defined in Section 3.3, including Recovery Rate by correlating agent responses with injected faults, and Robustness-Adjusted Success Rate by evaluating task completion under

Algorithm 1: Fault Injection for Robustness Benchmarking

Require: Agent $\mathcal{A}$, Environment $\mathcal{E}$, Fault Profile $\mathcal{P} = (F, \lambda, \delta)$, Task τ, Max Steps T
Ensure: Robustness Metrics $\mathcal{M}$

```
0: trajectory ← ∅
0: injected_faults ← ∅
0: s_0 ← initial state of τ
0: for t = 0 to T − 1 do
0:    a_t ← A(s_t) {Agent selects action}
0:    inject_fault ∼ Bernoulli(δ)
0:    if inject_fault then
0:       f ∼ SampleFailure(F, λ) {Sample from taxonomy}
0:       o_t ← SynthesizeObservation(f)
0:       injected_faults.append((t, f, o_t))
0:    else
0:       o_t ← E(s_t, a_t) {Normal execution}
0:    end if
0:    trajectory.append((s_t, a_t, o_t))
0:    s_{t+1} ← UpdateState(s_t, a_t, o_t)
0:    if TaskComplete(s_{t+1}) then
0:       break
0:    end if
0: end for
0: M ← ComputeMetrics(trajectory, injected_faults)
0: return M =0
```

fault conditions (similar to Li (2025a)). This algorithm ensures reproducible and controlled evaluation of agent robustness, transforming our conceptual framework into a practical benchmarking tool.

Robustness Metrics

Moving beyond the singular metric of task success rate, we propose a multi-faceted set of robustness metrics designed to provide a nuanced evaluation of an agent's behavior under failure. Let T be a set of tasks, and for a given task $\tau \in T$, let S_τ be a binary indicator of its ultimate success. The standard **Robustness-Adjusted Success Rate (RASR)** is simply the average of S_τ under fault injection: $\text{RASR} = \frac{1}{|T|} \sum_{\tau \in T} S_\tau$. While informative, this metric alone is insufficient, as it does not capture the *quality* of the recovery attempt. To address this, we introduce the core metric of **Recovery Rate (RR)**. For a task τ with a set of N injected faults, let R_i be a binary indicator that the agent successfully recovered from the i-th fault (e.g., by correctly identifying it and taking a plausible subsequent action). The Recovery Rate is then $\text{RR} = \frac{1}{N} \sum_{i=1}^{N} R_i$. This metric directly measures fault tolerance. Furthermore, we measure **Error Identification Accuracy (EIA)**, the proportion of times an agent's reasoning correctly names the type of error encountered. Finally, to quantify catastrophic failures, we track the **Unrecoverable Failure Rate (UFR)**, the proportion of tasks where the agent enters an infinite loop, crashes, or produces nonsensical output after a fault. These metrics, used in concert, provide a far more complete picture of agent robustness than was previously available, directly addressing the deficiencies of a

success-rate-only evaluation.

Model Improvement Framework

The ultimate goal of our benchmark is not just to evaluate but to foster the development of more robust agents. To this end, we propose a model improvement framework that leverages our fault injection system for training and fine-tuning. Current fine-tuning paradigms for tool use (Patil et al. 2023; Qin et al. 2023b) primarily use successful execution traces, which do not teach an agent how to handle failure. Our framework introduces **Adversarial Fine-Tuning Data**, which consists of tuples $(s_t, a_t, o_t^{injected}, a_{t+1}^{recovery})$, where $a_{t+1}^{recovery}$ is an expert-demonstrated or synthetically generated optimal action to take after observing the injected fault $o_t^{injected}$. We can then fine-tune a base model M_ϕ with parameters ϕ by minimizing a loss function $\mathcal{L}$ that combines standard cross-entropy on successful trajectories with a new robustness-specific term: $\mathcal{L}_{total} = \mathcal{L}_{standard} + \alpha \mathbb{E}_{(s,a,o^{inj})}[\mathcal{L}_{CE}(M_\phi(s, a, o^{inj}), a^{recovery})]$, where α is a weighting hyperparameter. This explicitly trains the model on recovery strategies. Furthermore, we can use reinforcement learning with a reward function r_t that incorporates our robustness metrics, for instance, providing a positive reward for a high Recovery Rate. By applying this framework and then re-evaluating the fine-tuned model on our benchmark, we can quantitatively demonstrate an improvement in robustness, thereby closing the loop from diagnosis to cure and providing a clear pathway for building more trustworthy, resilient agentic systems.

Experiments and Results

To empirically validate the proposed robustness benchmarking framework, we conduct a comprehensive experimental analysis. This section is structured to first establish the experimental setup by detailing the benchmarks, baseline models, and implementation specifics derived from our methodology. We then present a multi-faceted results analysis, where each subsection corresponds to a key component of our methodological framework: the evaluation of baseline robustness using our new metrics, a deep dive into failure-type analysis, an ablation study on fault injection parameters, a comparison of model improvement strategies, a robustness-efficiency trade-off analysis, and finally, a qualitative case study of failure and recovery modes. This structured approach allows us to systematically dissect the performance of tool-using agents under stress, providing clear answers to our research questions and demonstrating the critical necessity of moving beyond success rate.

Experimental Setup

Benchmarks and Datasets Our experiments leverage two established benchmarks, modified with our fault injection mechanism. The first is **ToolBench** (Qin et al. 2023b), a large-scale collection of over 16,000 real-world APIs across 49 categories. We utilize its instruction-following subset, which contains 2,347 complex tasks that require multi-step tool use. The second is **WebArena** (Zhou et al. 2023), a reproducible web environment for benchmarking autonomous agents. It provides 812 realistic tasks across four websites (shopping, content management, forum, and map) that require understanding and interacting with a live web interface. We selected these benchmarks for their realism, complexity, and representativeness of different tool-use paradigms (REST APIs and web navigation). Our fault injector was integrated into both platforms, allowing us to perform controlled experiments on the same underlying tasks while introducing the failures defined in our taxonomy.

Table 1: Overall Robustness Performance ($\delta = 0.3$)

Model	Task Success Rate	**RASR**	**Recovery Rate**	**UFR**
Gorilla-7B	71.2	45.3	28.1	41.5
ToolLLaMA-7B	75.8	52.1	35.7	34.2
GPT-4 Turbo	**84.5**	**68.9**	**59.4**	**18.3**

Baseline Agents We evaluate three state-of-the-art tool-using agents to ensure a comprehensive comparison. The first is **Gorilla-7B** (Patil et al. 2023), an open-source LLM specifically fine-tuned for generating accurate API calls, representing a model optimized for tool invocation in a static context. The second is **ToolLLaMA-7B** (Qin et al. 2023b), another open-source model trained on a massive corpus of tool-use instructions and execution traces, which incorporates planning and reasoning. The third baseline is a proprietary model, **GPT-4 Turbo with function calling** (OpenAI 2023), accessed via the official API. This model represents the current commercial state-of-the-art in tool use and provides a strong performance ceiling for our experiments. All models were evaluated using their recommended prompting strategies and tool-calling frameworks in a zero-shot setting to assess their inherent robustness without specialized training on our fault-injected data.

Overall Robustness Performance

Table 1 presents the core finding of our study, comparing the three baseline agents under a standard fault injection rate ($\delta = 0.3$). The results reveal a dramatic **robustness gap** that is entirely obscured by the standard Task Success Rate. While GPT-4 Turbo maintains a respectable lead, its Robustness-Adjusted Success Rate (RASR) drops by over 15 percentage points, indicating that a significant portion of its capability is brittle and fails under non-ideal conditions. The open-source models suffer even more severely, with Gorilla-7B's performance being nearly halved. More critically, the Recovery Rate (RR) metrics are alarmingly low across the board; even the best model only successfully recovers from roughly 60% of injected faults. The Unrecoverable Failure Rate (UFR) tells a complementary story, with Gorilla-7B entering catastrophic states in over 40% of tasks. This data unequivocally demonstrates that high performance on standard benchmarks is not a reliable indicator of real-world reliability. The significant dispersion between Task Success Rate and RASR underscores the necessity of our

Table 2: Recovery Rate by Failure Type

Model	Execution Errors	Semantic Anomalies	Env. Non-Stationarity
Gorilla-7B	31.5	15.2	12.8
ToolLLaMA-7B	40.3	25.7	21.4
GPT-4 Turbo	**65.8**	**52.1**	**43.5**

Table 3: Performance Degradation vs. Fault Injection Rate (δ)

Model	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.5$
Gorilla-7B	58.7	51.2	45.3	32.1
ToolLLaMA-7B	65.4	58.9	52.1	40.5
GPT-4 Turbo	**78.2**	**73.1**	**68.9**	**58.7**

proposed evaluation paradigm. For deployment in safety-critical or user-facing applications, an agent's RR and UFR may be more informative metrics of true operational reliability than its top-line success rate.

Failure-Type Analysis

To understand the nature of the robustness gap, we dissect agent performance by the failure categories from our taxonomy. Table 2 shows the Recovery Rate for each model across the three primary failure types. A clear hierarchy of difficulty emerges: all models handle Execution Errors (e.g., HTTP 404) best, likely because these are explicit and easily identifiable in the observation stream. Semantic Anomalies (e.g., malformed JSON) prove more challenging, as they require the agent to not only detect an issue but also to parse and validate the structure and content of the output, a reasoning step that current models struggle with. Environmental Non-Stationarity, representing the most complex failure mode, is the most difficult, with even GPT-4 Turbo recovering less than half the time. This suggests that agents lack a persistent, updable world model; they cannot seamlessly adapt when a fundamental premise of their toolset changes. The performance gap between GPT-4 and the open-source models is most pronounced for Semantic Anomalies, indicating that the proprietary model's stronger reasoning capabilities are a significant factor in robustness. This analysis provides crucial guidance for future research: improving robustness is not a monolithic task. Specific interventions, such as better output parsers, are needed for semantic issues, while architectural changes like dynamic tool registries may be required to handle non-stationarity.

Impact of Fault Injection Rate

The fault injection rate δ is a key parameter in our framework that controls the "stress level" of the environment. Table 3 shows how the Robustness-Adjusted Success Rate (RASR) degrades for each model as δ increases. The results demonstrate a nearly linear negative correlation between fault rate and performance for all models, confirming that tool-using agents lack inherent resilience. GPT-4 Turbo exhibits the most graceful degradation, with a 19.5 percentage point drop from $\delta = 0.1$ to $\delta = 0.5$, whereas Gorilla-7B drops 26.6 points over the same range, indicating its higher brittleness. This has profound implications for deployment. In a real-world system where the underlying tools have a known error rate, this data can be used to predict the agent's effective reliability. For instance, if a critical application requires a success rate above 60%, our data suggests that Gorilla-7B could not be used if the operational environment has a fault rate exceeding 10%, whereas GPT-4 Turbo could potentially tolerate a fault rate of 30% or more. This provides a quantitative, data-driven method for model selection based on operational requirements and expected environmental reliability, moving beyond vendor claims of capability to empirical measures of robustness.

Model Improvement Strategies

Table 4: Effectiveness of Robustness Fine-Tuning on ToolLLaMA-7B

Method	RASR	Recovery Rate	UFR	EIA
Baseline	52.1	35.7	34.2	40.1
+ CoT Prompting	55.8 (+3.7)	41.2 (+5.5)	30.5 (-3.7)	55.3 (+15.2)
+ Adv. Fine-Tuning	**62.3 (+10.2)**	**51.8 (+16.1)**	**22.1 (-12.1)**	**68.9 (+28.8)**

Having established the robustness gap, we evaluate two strategies from our improvement framework to close it, using ToolLLaMA-7B as our base model. Table 4 compares the baseline against two interventions: Chain-of-Thought (CoT) Prompting, which instructs the model to reason explicitly about the tool's output before proceeding, and our proposed Adversarial Fine-Tuning on the synthetic failure-recovery data. The results are striking. While CoT prompting provides a modest boost, particularly in Error Identification Accuracy (EIA), it is our Adversarial Fine-Tuning that leads to a dramatic improvement across all metrics. The RASR increases by over 10 points, the Recovery Rate jumps by 16.1 points, and the Unrecoverable Failure Rate is cut by more than a third. The massive gain in EIA suggests the fine-tuned model has learned to accurately diagnose problems, which is a prerequisite for effective recovery. This experiment provides a clear pathway forward: simply scaling up model size or training data on successful trajectories is insufficient. To build truly robust agents, the research community must invest in creating and leveraging adversarial training datasets that explicitly teach recovery behavior. The significant performance lift achieved here validates the utility of our entire benchmarking framework, as it provides both the means to identify weaknesses and the data to fix them.

Robustness-Efficiency Trade-off

A common concern with improved robustness is a potential loss of efficiency. Table 5 quantifies this trade-off by

Table 5: Robustness vs. Efficiency (Avg. Steps per Task)

Model	Steps (No Fault)	Steps ($\delta = 0.3$)	% Increase
Gorilla-7B	4.2	6.8	61.9%
ToolLLaMA-7B	4.5	6.5	44.4%
GPT-4 Turbo	3.8	5.1	**34.2%**
ToolLLaMA-7B (Fine-Tuned)	4.7	5.9	**25.5%**

comparing the average number of steps required to complete a task in pristine versus fault-injected conditions. All models require more steps when faults are present, as they must spend computational effort on recovery. However, the magnitude of this increase varies significantly. Gorilla-7B suffers the worst efficiency degradation, requiring over 60% more steps, which aligns with its high Unrecoverable Failure Rate, as getting stuck in loops inflates the step count. Crucially, our adversarially fine-tuned ToolLLaMA-7B shows the most efficient recovery behavior. While its baseline step count is slightly higher, its step increase under fault is the lowest at only 25.5%, significantly better than its original version (44.4%). This indicates that the fine-tuned model not only recovers more often but does so more directly, having learned effective recovery strategies rather than fumbling through random retries. This finding is critical for practical deployment: investing in robustness training does not merely make an agent more reliable; it can also make it more computationally efficient and cost-effective in unpredictable environments, as it reduces wasted cycles on failed actions and unproductive recovery attempts.

Qualitative Failure Mode Analysis

Beyond quantitative metrics, a qualitative analysis of failure modes provides deep insight into the behavioral flaws of current agents. Table 6 categorizes the most frequent failure behaviors observed across all models. The high frequency of "Ignore & Proceed" (18.3%) and "Hallucinated Success" (12.7%) is particularly alarming from a trust perspective, as the agent provides no indication to the user that it is operating on incorrect or fabricated information. The "Infinite Loop" (15.4%) and "Cascading Failure" (21.8%) modes highlight a lack of meta-cognition and state management, where the agent fails to recognize a dead-end or how one failure corrupts its entire plan. The most common mode, "Incorrect Diagnosis" (24.1%), underscores that while models are often aware that something is wrong, their reasoning about the root cause is flawed, preventing effective recovery. Notably, "Graceful Failure"—the desired behavior when recovery is impossible—is the rarest outcome. This taxonomy of failure modes, made possible by our fine-grained benchmarking, offers a clear roadmap for targeted improvements. For instance, to combat "Ignore & Proceed," architectures could enforce explicit error-checking modules. To reduce "Incorrect Diagnosis," training could include more diverse fault explanation data. This analysis confirms that robustness is not a single capability but a suite of behaviors that must be individually designed and evaluated.

Table 6: Frequency of Common Failure Modes (% of Injected Faults)

Category	Failure Mode	Frequency
Ignore & Proceed	Agent treats an error code (e.g., 404) as valid data and continues execution.	18.3%
Hallucinated Success	Agent claims a failed tool execution was successful and invents a plausible result.	12.7%
Infinite Loop	Agent retries the exact same failed action repeatedly without altering parameters.	15.4%
Cascading Failure	A single unhandled fault leads to a sequence of logically subsequent errors.	21.8%
Incorrect Diagnosis	Agent identifies an error but misattributes its cause, leading to an invalid recovery.	24.1%
Graceful Failure	Agent correctly identifies the error, explains why the task cannot proceed, and stops.	7.7%

Conclusion

This paper identified and addressed a critical gap in the evaluation of tool-using language agents: the lack of systematic robustness measurement. We proposed a comprehensive framework that shifts the paradigm from passive success-rate tracking to active stress-testing via fault injection. Our experiments, conducted on major benchmarks, quantitatively exposed a severe robustness gap in current state-of-the-art agents, proving that high task success does not equate to operational reliability. The introduction of metrics like Recovery Rate and Unrecoverable Failure Rate provides a nuanced view of agent behavior under failure. Crucially, we demonstrated that our benchmark is not just a diagnostic tool but a catalyst for improvement, showing that adversarial fine-tuning can significantly enhance an agent's resilience. For the future of trustworthy agentic AI, we argue that robustness must become a first-class evaluation criterion, and our framework provides the necessary tools and metrics to make this a reality.

References

Creswell, A.; ; et al. 2023. Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. *arXiv preprint arXiv:2205.09712*.

Hao, S.; ; et al. 2024. Reasoning with Language Model Planning: A Survey. *arXiv preprint arXiv:2402.00702*.

He, Y.; ; et al. 2024. TrustAgent: Towards Safe and Trustworthy Large Language Model based Agents. *arXiv preprint arXiv:2405.07647*.

Kim, B.; Xiong, H.; ; and Doshi-Velez, F. 2023. Evaluating Cognitive Maps and Planning in Large Language Models with CogEval. *arXiv preprint arXiv:2309.15129*.

Li, Z. 2025a. MCL for MLLMs: Benchmarking Forgetting in Task-Incremental Multimodal Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2760–2766.

Li, Z. 2025b. Retrieval-augmented forecasting with tabular time series data. In *Proceedings of the 4th Table Representation Learning Workshop*, 192–199.

Liu, Y.; ; et al. 2023. LLM+: A New Language Model Framework for Tool Use and Data Analysis. *arXiv preprint arXiv:2310.04515*.

OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

Parisotto, E.; Mohamed, A.-r.; Singh, R.; Li, L.; Zhou, D.; and Kohli, P. 2016. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*.

Qin, Y.; Hu, S.; Lin, Y.; Chen, W.; Ding, N.; Cui, G.; Zeng, Z.; Huang, Y.; Xiao, C.; Han, C.; et al. 2023a. Tool Learning with Foundation Models. *arXiv preprint arXiv:2304.08354*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Liu, X.; Wang, Y.; et al. 2023b. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint arXiv:2307.16789*.

Schick, T.; Dwivedi-Yu, J.; Dessi, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint arXiv:2302.04761*.

Shi, F.; Chen, X.; Misra, K.; Scales, N.; Duan, H.; Srinivasan, P.; Levine, Y.; and Zhou, D. 2023. Mind2Web: Towards a Generalist Agent for the Web. *arXiv preprint arXiv:2306.06070*.

Valmeekam, K.; ; et al. 2024. An Empirical Investigation of the Planning Capabilities of Large Language Models in Sequential Decision Making Tasks. *Workshop on Agent Learning in Open-Endedness @ ICLR 2024*.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023. On the Planning Abilities of Large Language Models–A Critical Investigation. *Advances in Neural Information Processing Systems*, 36.

Wang, Y.; ; et al. 2024. Large Language Model Safety: A Survey. *arXiv preprint arXiv:2405.18148*.

Wu, Q.; ; et al. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*.

Yang, S.; ; et al. 2023. Delve: A Tool for Evaluating Logical Reasoning Errors in Language Models. *arXiv preprint arXiv:2310.10310*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Bisk, Y.; Fried, D.; Alon, U.; et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854*.

Zhou, Y.; ; et al. 2024. AgentTown: A Dynamic Environment for Complex Multi-Agent Interactions. *arXiv preprint arXiv:2402.10160*.