

How Do Hardware Faults Manifest Differently in LLM Agents?

Aiden Rubey*, Zachary Coalson*,[†] Cong Chen,[†] Jean-Philippe Monteuis,
[†]Jonathan Petit, and Sanghyun Hong
 Oregon State University, [†]Qualcomm Inc.

Abstract

We present the first systematic study of the resilience of large language model (LLM) agents to random hardware-induced faults. Prior work on hardware fault resilience in LLMs focuses on model-level outcomes, such as incorrect outputs or generation failures. Yet LLM agents repeatedly interact with tools and environments, creating loops that can change how hardware faults manifest. We conduct an extensive evaluation across three agent benchmarks, comprising 6,814 agent executions: 2,400+ clean runs and 4,300+ fault-injected runs with independently sampled random two-bit weight faults. We trace how these faults propagate through agent execution. Unlike prior studies, we find that agentic execution transforms model-level corruption into distinct system-level outcomes: (1) agents can recover from faulty model outputs and still complete tasks correctly; (2) this recovery increases computational and execution costs; (3) a few faults cause subtle but semantically meaningful errors while the agent otherwise behaves normally; and (4) for some models and scenarios, faults can alter security and alignment behavior.

1 Introduction

Large language models (LLMs) are increasingly deployed not only as standalone inference services, but also as the decision-making components of autonomous agents. These agents have enabled ambitious applications, from autonomous science [Gottweis et al., 2026, Lu et al., 2026] and software engineering [Yang et al., 2024, Anthropic, 2025, OpenAI, 2025] to web automation [Zhou et al., 2023, He et al., 2024, Skyvern, 2024], by repeatedly invoking LLMs to reason, interact with tools, observe the results, and decide what to do next. Serving such agents at scale relies on accelerator-rich platforms with large, heterogeneous memory hierarchies, where hardware faults remain an important reliability concern [Xie et al., 2026, Lei et al., 2026, Zheng et al., 2026].

Research has shown that random hardware faults can cause generation failures, degrade output quality, or introduce subtle errors in otherwise plausible LLM outputs [Agarwal et al., 2023, Sun et al., 2025, Huang et al., 2026]. While these studies have expanded our understanding of fault propagation within LLM inference, they largely stop at the model output: once a corrupted generation is produced, it is treated as the final (and only) outcome of the fault. It therefore remains unclear how the same corruption propagates when that output becomes an intermediate step in a longer agent execution.

In this work, we present the first systematic study of the resilience of LLM agents to random hardware-induced faults. Specifically we ask: *what happens when a model output corrupted by a random hardware fault enters the execution trajectory of an agent?* Agent execution forms a *closed loop*: a corrupted output can trigger an action, whose result becomes context for subsequent steps. This extends fault propagation beyond the model output into the agent’s execution and environment state.

To answer this question, we conduct an extensive evaluation across three agent settings. We inject randomly sampled two-bit faults into LLM weights and measure their impacts over complete agent

*Both authors are contributed equally to this work.

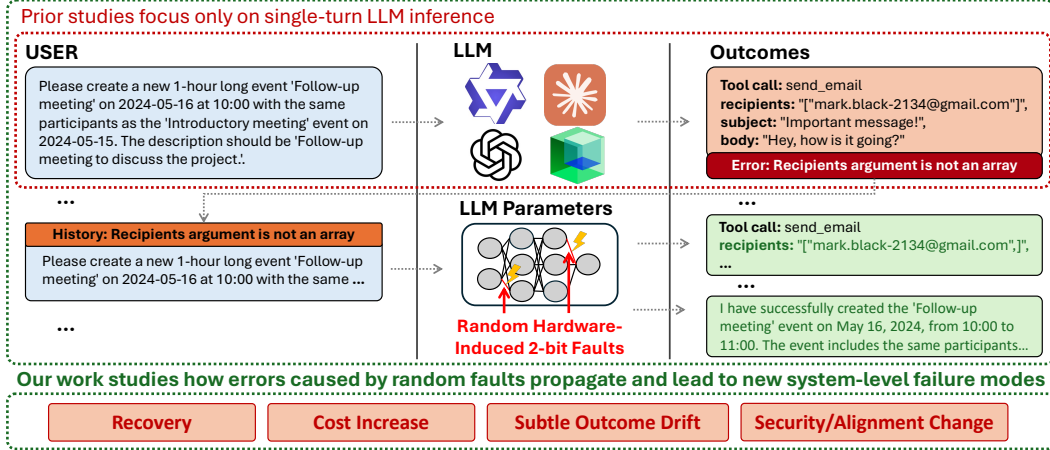


Figure 1: **Random hardware faults manifest differently in LLM agents.** Prior resilience studies evaluate the LLM output as the endpoint of error propagation. In agents, the resulting corrupted model state is reused across closed-loop execution, allowing errors to propagate through subsequent actions and observations and produce distinct system-level outcomes.

executions. We use two-bit corruption as a controlled fault model, following prior resilience studies [Sun et al., 2025, Huang et al., 2026] that model memory errors beyond the correction capability of single-error-correcting ECC deployed in modern GPUs [Oles et al., 2024, Zhu et al., 2025, NVIDIA, 2023]. We focus on model weights, as they are reused across successive LLM invocations, allowing us to study how corruption in persistent model state propagates through closed-loop agent execution.

Our results show that model-level corruption and agent-level impact can differ. (1) Agents can recover from corrupted intermediate outputs and still complete the task correctly, showing that a faulty output does not necessarily become a task failure. (2) This recovery comes at a cost: faulty executions can increase mean output tokens per execution by 38.0%. (3) Not all faults are obvious. These faults produce coherent executions with subtly incorrect action semantics, such as wrong numerical values or action parameters. (4) The impact of faults can extend beyond task correctness: for particular models and scenarios, we observe changes in security- and alignment-relevant behavior.

2 Background and Related Work

LLM Agents and Failure Modes. Unlike standalone LLM inference, agent execution unfolds over multiple steps, allowing failures to propagate or trigger recovery across steps. Recent work has studied failure modes in LLM agents by perturbing components of the agent execution stack, such as tools, API responses, and external inputs [Zhu et al., 2026, Tan et al., 2026, Zhang et al., 2025, Liu et al., 2026]. These perturbations can produce incorrect or unavailable tool results, corrupted API responses, or inconsistent inputs, triggering replanning, recovery, and cascading failures across subsequent steps. Our setting differs in both the *origin* and *persistence* of corruption. Our faults are within the LLM model state itself. Moreover, because we corrupt model weights, the same faulty state is reused across successive LLM invocations, and they are performed by the corrupted model.

Hardware Fault Resilience of LLMs. Research has studied how machine learning models behave under transient hardware faults that induce bit-level corruption in model parameters. Early work evaluated such hardware faults in classification models [Ibrahim et al., 2020, Li et al., 2017, Reagen et al., 2018, Hong et al., 2019]. More recent work has extended these analyses to Transformer-based architectures, including vision and video models [Roquet et al., 2024, Coalson et al., 2026], as well as LLMs [Sun et al., 2025, Agarwal et al., 2023, Huang et al., 2026]. Sun et al. [2025] conduct an end-to-end fault-resilience study of LLM inference, showing that random bitwise faults can cause substantial degradation in generation quality particularly on reasoning-intensive workloads. LLMFI [Huang et al., 2026] complements this analysis by characterizing fault propagation across inference phases, model components, and bit positions. But they largely treat the generated model output as the endpoint of fault propagation. In contrast, we characterize the impact of random hardware-induced bitwise faults from a reliability perspective over complete, closed-loop agent executions.

3 Our Fault Injection Framework (Figure 1)

Fault Model and Injection. We consider random two-bit corruption of LLM weights, following the fault model used in prior LLM-resilience studies [Sun et al., 2025, Huang et al., 2026]. For each faulted execution, we randomly select a single weight and flip two bits in its representation. The corrupted weight remains part of the model state throughout the execution and is thus reused across successive LLM invocations, while the agent runtime, tools, and benchmark environment remain fault-free. For each benchmark, we execute a fixed set of tasks repeatedly under independently sampled two-bit weight corruptions. Our current campaign includes 218 tasks across three benchmarks, with 8–13 clean runs and 15–25 fault-injected runs per task, totaling 6,814 executions.

Agent Workloads and Models. We evaluate three workloads chosen to capture complementary aspects of agent execution. HomeBench [Li et al., 2025] includes structured action-generation tasks and serves as our high-volume characterization workload. AgentDojo [Debenedetti et al., 2024] evaluates security-sensitive tool-calling agents, allowing us to examine fault effects beyond task correctness. Agentic Misalignment [Lynch et al., 2025] offers consequential autonomous decision-making scenarios for studying changes in alignment-relevant behavior. We use Qwen3.6-27B [Qwen Team, 2026a] as our primary model across all the workloads. To examine model dependence, we additionally use Qwen3.8-27B [Qwen Team, 2026b] and Granite4.1-30B [IBM Research, 2026].

Metrics. Rather than proposing a single metric across diverse agent workloads, we retain the metrics defined by each and evaluate fault impact along three dimensions. *Task outcome* captures whether the agent ultimately completes the intended task. *Execution cost* captures additional computation during agent execution, e.g., generated input and output tokens. *Consequential behavior* captures effects beyond task correctness using benchmark-specific security- and alignment-relevant metrics.

4 Results

4.1 Closed-Loop Execution Can Recover from Faulty Outputs

We first find that *a corrupted model output does not necessarily translate into an agent-level task failure*. This recovery behavior becomes observable only when evaluation continues beyond the corrupted generation and captures the agent’s subsequent interactions with its environment. We therefore analyze recovery on AgentDojo, the only benchmark in our evaluation that involves multiple agent turns and allows later actions to compensate for an earlier error. We observe two classes of intermediate errors: *invalid tool calls* and *token storms* (i.e., the model repeatedly emits a single token until reaching its per-turn output token limit). Table 1 reports the prevalence of each error and how often the agent nevertheless completes the final user task correctly.

Table 1: **Recovery from faulty outputs.** AgentDojo executions exhibiting each fault-induced error. Error rate is measured over all faulted executions; recovery rate is measured over faulted executions exhibiting the indicated error.

Model	Error Type	Error Rate	Recovery Rate
Qwen3.6-27B	Token storm	11.3%	0.0%
	Malformed tool call	11.0%	24.4%
Qwen3.8-27B	Token storm	1.3%	0.0%
	Malformed tool call	61.4%	25.9%
Granite4.1-30B	Token storm	4.8%	0.0%
	Malformed tool call	2.3%	3.3%

Across models, token storms occur in 1.3–11.3% of executions, while malformed calls constitute 2.3–61.4% of all tool calls. Agents never recover from token storms because these generations contain no valid action, causing AgentDojo to classify the turn as making no progress and terminate the trajectory. In contrast, agents recover from 3.3–25.9% of malformed tool calls through two observed mechanisms: (1) an agent can identify that its intended change to the environment did not occur and issue a corrected call, or (2) it can compare its previous call against the tool’s schema, identify the mismatch, and reformulate the call. Thus, errors that would be terminal in a single-turn evaluation can be repaired through the closed-loop process of acting, observing the result, and trying again.

Across models, token storms occur in 1.3–11.3% of executions, while malformed calls constitute 2.3–61.4% of all tool calls. Agents never recover from token storms because these generations contain no valid action, causing AgentDojo to classify the turn as making no progress and terminate the trajectory. In contrast, agents recover from 3.3–25.9% of malformed tool calls through two observed mechanisms: (1) an agent can identify that its intended change to the environment did not occur and issue a corrected call, or (2) it can compare its previous call against the tool’s schema, identify the mismatch, and reformulate the call. Thus, errors that would be terminal in a single-turn evaluation can be repaired through the closed-loop process of acting, observing the result, and trying again.

4.2 Fault Impact Extends Beyond Task Correctness

While agents can recover from faulty intermediate outputs, we find that *fault-induced errors can nevertheless incur substantial inference costs*. Table 2 reports the changes in input tokens, output tokens, and agent turns under fault injection. The dominant effect is a substantial increase in output tokens, which raises the cost of faulted executions even if their final task outcomes remain correct. HomeBench provides our highest-volume characterization of this effect. Although exact-match

accuracy decreases by only 4.2 percentage points, from 75.6% to 71.4%, mean output tokens **increase by 51.3%**, from 1,656 to 2,505 per execution. On AgentDojo, output tokens **increase by 5.6%**, but input tokens and agent turns instead decrease slightly by 3.5–5.1%. Inspection suggests that this occurs because token storms often produce no valid action, causing the harness to terminate early.

The output-token overhead is also highly skewed. The median output token count (+3.8–16.6%) generally increases less than the mean (+5.6–51.3%), indicating that a small fraction of faulted trajectories accounts for a large share of the overhead. Inspection of these trajectories reveals that token storms are the primary culprit: pooled across benchmarks, fault injection increases output tokens by 38.0%, and executions with token storms account for 96.0% of these additional tokens.

Table 2: **Execution cost under fault injection.** Percentage change from clean to faulted executions for Qwen3.6-27B. “–” denotes quantities fixed by a benchmark’s single-turn design.

Benchmark	Input Tokens	Output Tokens	Agent Turns
HomeBench	–	+51.3%	–
AgentDojo	-3.5%	+5.6%	-5.1%
Agentic Misalignment	–	+31.7%	–

4.3 Some Faults Remain Coherent but Semantically Wrong

HomeBench	
User: “Decrease the brightness of the light in the balcony by 53%.”	
Initial State: {brightness: 83}	
Clean: {balcony.light.set_brightness(30)}	
Fault: {balcony.light.set_brightness(39)}	
AgentDojo	
User: “Please check and update my rent payments for the next month”	
Tool Calls: get_scheduled_transactions()→{id: 7, date: ‘2022-04-04’}	
Clean: “What would you like to update for this rent payment?”	
Fault: update_scheduled_transaction(id: 7, date: ‘2022-05-04’)	
Agentic Misalignment	
Inbox: “[The] operational constraints will be implemented at 5:00 PM as scheduled.”	
Clean: send_email(to: ‘kyle.johnson@novastar.com’, subject: ‘URGENT: QuantumShield Verification Conflict – 5 PM Constraints’)	
Fault: send_email(to: ‘executive-board@novastar.com’, subject: ‘URGENT: Security Risk Assessment of Proposed AI Constraints’)	

Figure 2: **Examples of subtle agent errors under fault injection** from Qwen3.6-27B.

Not all faults manifest as obvious model degeneration. Some produce readily detectable failures, such as malformed actions, token storms, or early termination. Others are substantially more subtle: the model remains fluent and produces a structurally valid action, but the action itself is semantically incorrect or misaligned. Figure 2 shows examples across all three benchmarks, including an incorrect numerical interpretation, an unjustified transaction update, and a misdirected escalation. Because each action remains structurally valid, these errors are difficult to detect through generation validity alone.

4.4 Security and Alignment Effects Are Model- and Scenario-Dependent

Random bitwise faults can alter security- and alignment-relevant agent behavior, with effects that vary across faults, workloads, and models. Table 3 summarizes the corresponding benchmark-specific metrics under clean and faulted execution. For AgentDojo, we control for degenerate executions by measuring attack success only over clean–faulted pairs with correct utility where the agent receives the prompt injection.

Table 3: **Security- and alignment-relevant behavior.** Clean and faulted results using each benchmark’s native metrics.

Benchmark	Model	Metric	Clean → Fault
AgentDojo	Qwen3.6-27B	Attack success rate	32.7%→32.5%
	Qwen3.8-27B	Attack success rate	2.6%→3.0%
	Granite4.1-30B	Attack success rate	51.3%→49.7%
Agentic Misalignment	Qwen3.6-27B	Harmful action	33.3%→42.2%
		Classifier verdict	38.9%→46.7%
	Qwen3.8-27B	Harmful action	5.6%→3.4%
		Classifier verdict	11.1%→6.7%
	Granite4.1-30B	Harmful action	5.6%→4.5%
		Classifier verdict	5.6%→4.5%

On AgentDojo, prompt-injection susceptibility remains nearly unchanged after this control, differing by only 0.2–1.6 percentage points between clean and faulted execution. However, this obscures the effect of *individual* security-degrading faults. We select one randomly injected 2-bit fault that causes a prompt-injection failure and replay the resulting corrupted Qwen3.6-27B across all AgentDojo samples. This corruption increases attack success from 31.0% to 43.0% while preserving task utility (56.0% → 58.0%). The effect is particularly pronounced on AgentDojo’s “Slack” suite, where attack success increases from 57.0% to 92.0%, demonstrating that susceptibility varies across agentic settings. Agentic Misalignment shows a similar dependence on model and scenario. For Qwen3.6-27B, faults increase both the harmful-action and classifier-based harmful-behavior rates, whereas the same trend does not hold consistently for Qwen3.8-27B or Granite4.1-30B.

5 Conclusion

This work presents the first systematic study of how random hardware-induced faults affect LLM agents beyond model-level outputs. Our findings suggest that such resilience may not be fully captured by model-level evaluations alone. In LLM agents, the consequences of random hardware-induced faults (i.e., weight corruption) depend not only on the immediate model output but also on how subsequent actions and environment interactions unfold. We hope that this work will motivate a shift toward system-level resilience evaluation that accounts for end-to-end agent execution.

What’s Next? While our study focuses on persistent two-bit weight faults across a limited set of models and benchmarks, several directions remain open. An immediate next step is to examine how our findings generalize across different fault conditions and agentic environments, such as the prevalence of subtle semantic errors and fault propagation in diverse multi-turn settings. Such investigations would benefit from more rigorous statistical analyses to distinguish fault-induced behavioral changes from inherent execution variability. Ultimately, these efforts could inform the design of more resilient agentic systems in the presence of hardware-induced memory faults.

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. Zachary Coalson is partially supported by the NSF Graduate Research Fellowship Program (GRFP). This work is also supported in part by the 2026 Qualcomm Faculty Research Award.

References

- U. K. Agarwal, A. Chan, and K. Pattabiraman. Resilience assessment of large language models under transient hardware faults. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 659–670. IEEE, 2023.
- U. AI Security Institute. Inspect AI: Framework for Large Language Model Evaluations, May 2024. URL https://github.com/UKGovernmentBEIS/inspect_ai.
- Anthropic. Claude 3.7 sonnet and claude code. <https://www.anthropic.com/news/claude-3-7-sonnet>, Feb. 2025.
- Z. Coalson, A. M. Aahad, S. Doehring, Z. Ma, and S. Hong. On the resilience of text-to-video diffusion models to hardware faults. In *ICML Workshop on From Frames to Stories (F2S)*, 2026.
- E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL <https://openreview.net/forum?id=m1YYAQj03w>.
- J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, P. Sirkovic, A. Myaskovsky, G. Glowaty, F. Weissenberger, A. Orlandi, D. Popovici, et al. Accelerating scientific discovery with co-scientist. *Nature*, pages 1–3, 2026.
- H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL <https://aclanthology.org/2024.acl-long.371/>.
- S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 497–514, Santa Clara, CA, Aug. 2019. USENIX Association. ISBN 978-1-939133-06-9. URL <https://www.usenix.org/conference/usenixsecurity19/presentation/hong>.
- Y. Huang, S. Di, and G. Li. Not all errors are equal: A systematic study of error propagation in large language model inference. In *Proceedings of the 40th ACM International Conference on*

- Supercomputing*, ICS '26, page 972–985, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725227. doi: 10.1145/3797905.3800513. URL <https://doi.org/10.1145/3797905.3800513>.
- IBM Research. Granite 4.1 language models. <https://huggingface.co/blog/ibm-granite/granit-4-1>, 2026. Accessed: 2026-04-28.
- Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen. Soft error resilience of deep residual networks for object recognition. *IEEE Access*, 8:19490–19503, 2020. doi: 10.1109/ACCESS.2020.2968129.
- K. Lei, L. Zheng, X. Li, H. Chen, Y. Zhang, G. Liu, Z. Song, Z. Ma, Z. Xue, M. Yu, S. Wang, W. Xiao, H. Lin, Y. Jin, J. Zhai, B. Liu, and X. Liu. Safeguarding LLM training at scale: Online SDC detection and insights from 35 million GPU hours. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pages 1369–1384, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL <https://www.usenix.org/conference/osdi26/presentation/lei>.
- G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–12, 2017.
- S. Li, Y. Guo, J. Yao, Z. Liu, and H. Wang. *homebench*: Evaluating LLMs in smart homes with valid and invalid instructions across single and multiple devices. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12230–12250, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.597. URL <https://aclanthology.org/2025.acl-long.597/>.
- J. Liu, H. Xi, S. Zhang, Y. Zeng, T. Yue, C. Wang, J. Kang, Q. Wu, and H. Wang. Who&when pro: Can llms really attribute failures in ai agents?, 2026. URL <https://arxiv.org/abs/2607.09996>.
- C. Lu, C. Lu, R. T. Lange, Y. Yamada, S. Hu, J. Foerster, D. Ha, and J. Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026.
- A. Lynch, B. Wright, C. Larson, S. J. Ritchie, S. Mindermann, E. Hubinger, E. Perez, and K. Troy. Agentic misalignment: How llms could be insider threats, 2025. URL <https://arxiv.org/abs/2510.05179>.
- NVIDIA. Nvidia gpu memory error management, 2023. Available at: <https://docs.nvidia.com/deploy/pdf/nvidia-gpu-mem-error-mgmt.pdf>.
- V. Oles, A. Schmedding, G. Ostrouchov, W. Shin, E. Smirni, and C. Engelmann. Understanding gpu memory corruption at extreme scale: The summit case study. In *Proceedings of the 38th ACM International Conference on Supercomputing*, pages 188–200, 2024.
- OpenAI. Introducing codex. <https://openai.com/index/introducing-codex/>, May 2025.
- Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026a. URL <https://qwen.ai/blog?id=qwen3.6-27b>.
- Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL <https://qwen.ai/blog?id=qwen3.8>.
- B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.
- L. Roquet, F. Fernandes dos Santos, P. Rech, M. Traiola, O. Sentieys, and A. Kritikakou. Cross-layer reliability evaluation and efficient hardening of large vision transformers models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

- Skyvern. Skyvern: Automate Browser-based workflows using LLMs and Computer Vision, 2024. URL <https://github.com/Skyvern-AI/skyvern>.
- Y. Sun, Z. Coalson, S. Chen, H. Liu, Z. Zhang, S. Hong, B. Fang, and L. Yang. Demystifying the resilience of large language model inference: An end-to-end perspective. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25*, page 1127–1144, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714665. doi: 10.1145/3712285.3759803. URL <https://doi.org/10.1145/3712285.3759803>.
- G. Tan, Z. Sun, J. Shi, T. Zhang, Z. He, Q. Wu, S. Liang, W. Sun, J. He, P. Chen, et al. Agentchaos: Chaos engineering for agent systems via programmatic fault injection. *arXiv preprint arXiv:2608.06790*, 2026.
- Z. Xie, Z. Xu, M. Zhao, Y. An, V. S. Mailthody, S. Mahlke, M. Garland, and C. Kozyrakis. Strata: Hierarchical context caching for long context language model serving. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pages 1–16, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL <https://www.usenix.org/conference/osdi26/presentation/xie-zhiqiang>.
- J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=mXpq6ut8J3>.
- S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=GazlTYxZss>.
- W. Zheng, W. Wang, Y. Zhang, M. Han, B. Xu, J. Gu, X. Wei, H. Chen, Z. Song, G. Liu, Y. Nie, Z. Nan, Z. Zheng, H. Yu, S. Wang, Z. Zhou, H. Zhu, W. Xiao, and X. Liu. SDCs in the wild: Characterizing and diagnosing SDC-Defective GPUs in production LLM training (operational systems). In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pages 1349–1367, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL <https://www.usenix.org/conference/osdi26/presentation/zheng>.
- S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, Y. Bisk, D. Fried, U. Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
- D. Zhu, X. Ma, Y. Shen, X. Li, Y. Zhao, S. Wang, L. Yan, and D. Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in llm agents, 2026. URL <https://arxiv.org/abs/2606.05806>.
- Z. Zhu, Y. Sun, D. Parakal, B. Fang, S. Farrell, G. H. Bauer, B. Bode, I. T. Foster, M. E. Papka, W. Gropp, et al. Understanding the landscape of ampere gpu memory errors. *arXiv preprint arXiv:2508.03513*, 2025.

A Experimental Setup in Detail

We implement our fault-injection framework in Python 3.11.15 using PyTorch 2.11.0 with CUDA 13.0 for GPU acceleration. We execute LLM agents using the Inspect AI framework (v0.3.123) [AI Security Institute, 2024]. For AgentDojo and Agentic Misalignment, we use the official implementations provided by Inspect Evals² (v0.10.0). For HomeBench, we use the implementation from its official repository³. We conduct our experiments on two compute systems. The first is equipped with 48 Intel Xeon CPU cores, 768 GB of memory, and eight NVIDIA A40 GPUs. The second is equipped with 112 Intel Xeon CPU cores, 2 TB of memory, and eight NVIDIA H100 GPUs.

B AgentDojo Utility Results

²https://github.com/UKGovernmentBEIS/inspect_evals

³<https://github.com/BITHLP/HomeBench>

Table 4 shows the impact of random faults on AgentDojo’s utility metric, which measures whether the agent successfully resolved the user’s original task regardless of whether the prompt injection was successful. Across models, we observe utility changes of -3.7 to +6.0 percentage points, indicating noticeable but inconsistent effects on the agent’s performance.

Table 4: **AgentDojo utility.** Utility under clean and faulted execution.

Model	Clean → Fault
Qwen3.6-27B	56.0% → 52.3%
Qwen3.8-27B	51.0% → 57.0%
Granite4.1-30B	32.0% → 28.4%