

TTTR: Record, Replay, and Steer AI (Extended Abstract)

Cheng Tan
Northeastern University

Abstract

AI agent behaviors are difficult to reproduce because nondeterminism enters at multiple layers of the AI software stack, from tensor computation and token sampling to the external environment. TTTR is a record-replay framework that captures these sources and enables selective replay, allowing one layer to vary while others remain fixed. This supports cross-layer fault localization, counterfactual analysis, and reproducible evaluation of agent behavior.

ACM Reference Format:

Cheng Tan. 2026. TTTR: Record, Replay, and Steer AI (Extended Abstract). In *Rigorous LLM Systems (RigoLLM '26)*, November 15–18, 2026, Hong Kong, China. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3858610.3859269>

1 Introduction

An agent may complete a task successfully one day and, given the identical task the next day, fail or take an entirely different sequence of actions. No tooling tells us which run to trust or what differed. Prior work [6] finds that across 17 models and 10 system-administration tasks, agents routinely exceed 80% pass rates, yet two runs of the same task agree on their sequence of state-modifying actions only about 39% of the time—sometimes diverging solely because the environment’s timestamp differed.

Nondeterminism is an old concern in systems, but in the AI stack its effects are amplified: rather than being contained within one abstraction layer, it propagates from the GPU upward until it surfaces as a different shell command. We argue that the missing primitive is record-and-replay spanning all three layers (§2), and outline TTTR, Tensor-Token-Trace-level Replay (§3). *Replay* here means faithfully re-executing one specific past run in order to investigate it—instead of, as in recent agent systems [3, 10], reusing a recorded workflow on new tasks. TTTR’s principal value lies in *selective* replay: holding two layers fixed while releasing the third turns “what changed?” into a controlled experiment (§4).

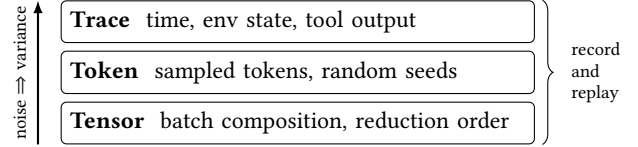


Figure 1. Non-determinism enters at three layers and propagates upward.

2 Three layers of non-determinism

Tensor. The arithmetic itself is nondeterministic. Most inference kernels lack *batch invariance*: reduction order depends on how many requests are batched together, so a request’s output depends on server load at arrival. Batch-invariant kernels remove this, at roughly 1.6× the latency [5]. Beneath lies slower drift: cuBLAS guarantees bitwise reproducibility only on the same architecture and SM count, and not across toolkit versions [7, 13]. Different hardware generations can also introduce nondeterminism into the same computation.

Token. Sampling during decoding introduces nondeterminism in next-token selection. This is the most visible source and the only one with a user-facing control, but that control is weak: the seed parameter is best-effort (“determinism is not guaranteed”) [9].

Trace. The environment contributes timestamps, PIDs, file listings, network responses, tool outputs, and concurrent modifications by other processes. These lie outside the model, yet they enter its context and so steer later tokens. Since such fields vary by task, TTTR defines this layer by an interface rather than a list: every value crossing the agent–environment boundary.

The layers compose upward (Figure 1): a tie resolved differently in a reduction yields a different token, a different command, then a different machine state: so a debugger for agents cannot stop at the framework boundary.

3 TTTR: record, replay, and steer

Record boundaries, not state. A failure is not known when recording starts, so recording must be always-on, and hence cheap. Like R2 and RR [4, 8], TTTR logs only the nondeterministic inputs crossing each layer boundary and recomputes the rest. At the tensor layer this is batch composition and kernel/library identity, not activations; at the token layer, sampled token IDs and sampler state; at the trace layer, every tool result and, beneath it, every syscall return, as in ReVirt and RR [2, 8]. Two naive baselines fail. Recording



This work is licensed under a Creative Commons Attribution 4.0 International License.

RigoLLM '26, Hong Kong, China

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3090-0/2026/11

<https://doi.org/10.1145/3858610.3859269>

everything is impractical: FP32 logits over a 150K-entry vocabulary alone cost 0.6 MB per generated token—gigabytes per session. A transcript, as kept by agent tracing tools, is cheap but not replayable: it shows what happened, not what would have happened had one value differed.

Replay decisions, not bits. ODR abandons exact-schedule reproduction and guarantees only identical *output* [1]; likewise, the right fidelity target for agents is the same *decisions*, not bit-identical tensors.

Selective replay is TTTR’s central capability: hold two layers fixed while releasing the third, or perturb one recorded value—advance the clock, substitute a tool result, alter a sampled token—and observe where execution first diverges. This generalizes mutable replay [12] to arbitrary sources of nondeterminism, and delta debugging [14, 15] from one input to a search *across* layers.

Challenges. (1) After a perturbation causes divergence, execution leaves the log and continues live, needing per-step environment checkpoints (e.g., container snapshots). (2) The agent writes to its environment, so replay must sandbox side effects. (3) Hosted models hide the tensor layer, limiting them to token- and trace-level replay.

Prior agent record-and-replay. AgentRR [3] distills recorded traces into reusable experience so that later, similar tasks run cheaper and safer; Codex Record & Replay [10] turns a user’s demonstration into a reusable skill. Both replay a *workflow* for reuse; neither reproduces a given execution, pins its nondeterminism, or reaches below the API. TTTR is complementary: a debugging primitive.

4 Applications

Fault localization across layers. Pinning one layer to a prior run isolates the source of divergence, and the answer dictates the fix. If pinning tensors removes a failure, the agent sits on a decision boundary that arithmetic noise can tip: no environment fix will help, and the remedy lies in the prompt or model. If pinning the trace removes it, minimizing over trace values names the culprit (say, the timestamp) and the tool or environment to harden.

Counterfactuals, regression, evaluation. Replay varies one factor with the rest fixed: would the agent still kill the process if the port were free? Unlike re-execution, which changes many factors at once, this is a controlled intervention [11]. Replaying recorded failures against a patched prompt or model yields regression tests, and pinning lower layers lets evaluation separate model behavior from lower-layer noise, which pass rate hides.

References

- [1] Gautam Altekar and Ion Stoica. 2009. ODR: Output-Deterministic Replay for Multicore Debugging. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, Big Sky, MT, USA. doi:10.1145/1629575.1629594
- [2] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. 2002. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI '02)*. USENIX Association, Boston, MA, USA. <https://www.usenix.org/conference/osdi-02/revirt-enabling-intrusion-analysis-through-virtual-machine-logging-and-replay>
- [3] Erhu Feng, Wenbo Zhou, Zibin Liu, Le Chen, Yunpeng Dong, Cheng Zhang, Yisheng Zhao, Dong Du, Zhichao Hua, Yubin Xia, and Haibo Chen. 2025. Get Experience from Practice: LLM Agents with Record & Replay. arXiv preprint arXiv:2505.17716. doi:10.48550/arXiv.2505.17716
- [4] Zhenyu Guo, Xi Wang, Jian Tang, Xuezheng Liu, Zhilei Xu, Ming Wu, M. Frans Kaashoek, and Zheng Zhang. 2008. R2: An Application-Level Kernel for Record and Replay. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*. USENIX Association, San Diego, CA, USA.
- [5] Horace He and Thinking Machines Lab. 2025. Defeating Nondeterminism in LLM Inference. Thinking Machines Lab: Connectionism. <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>
- [6] Tian Lu, Zikai Wang, and Cheng Tan. 2026. Beyond Pass Rate: A Hierarchy of Behavioral Consistency for LLM Agents. In *Proceedings of the 1st Symposium on Consistency Checking Principles (SCCP '26)*. Boston, MA, USA. <https://sccp-workshop.github.io/sccp26/papers/sccp26-behavioral-consistency-llm-agents.pdf> Co-located with VLDB 2026.
- [7] NVIDIA Corporation. 2026. cuBLAS Library User Guide: Results Reproducibility. NVIDIA CUDA Toolkit Documentation. <https://docs.nvidia.com/cuda/cublas/index.html#results-reproducibility>
- [8] Robert O’Callahan, Chris Jones, Nathan Froyd, Kyle Huey, Albert Noll, and Nimrod Partush. 2017. Engineering Record and Replay for Deployability. In *Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC '17)*. USENIX Association, Santa Clara, CA, USA, 377–389.
- [9] OpenAI. 2026. Create Chat Completion: the seed Parameter. OpenAI API Reference. <https://developers.openai.com/api/reference/> Marked deprecated; “Determinism is not guaranteed”.
- [10] OpenAI. 2026. Introducing Record & Replay (Codex). OpenAI Developer Community. <https://community.openai.com/t/introducing-record-replay/1384088>
- [11] Jainet Shah. 2026. Causal Agent Replay: Counterfactual Attribution for LLM-Agent Failures. arXiv preprint arXiv:2606.08275. doi:10.48550/arXiv.2606.08275
- [12] Nicolas Viennot, Siddharth Nair, and Jason Nieh. 2013. Transparent Mutable Replay for Multicore Debugging and Patch Validation. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. ACM, Houston, TX, USA, 127–138. doi:10.1145/2451116.2451130
- [13] Nathan Whitehead and Alex Fit-Florea. 2011. *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs*. Technical Report. NVIDIA Corporation. <https://developer.download.nvidia.com/assets/cuda/files/NVIDIA-CUDA-Floating-Point.pdf>
- [14] Andreas Zeller. 1999. Yesterday, My Program Worked. Today, It Does Not. Why?. In *Software Engineering — ESEC/FSE '99 (Lecture Notes in Computer Science, Vol. 1687)*. Springer, Toulouse, France, 253–267. doi:10.1007/3-540-48166-4_16
- [15] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Transactions on Software Engineering* 28, 2 (Feb. 2002), 183–200. doi:10.1109/32.988498