Toward Systems Foundations for Agentic Exploration

Jiakai Xu[*]
Columbia University
ax2155@columbia.edu

Tianle Zhou[*]
Columbia University
mz2998@columbia.edu

Eugene Wu
Columbia University
ewu@cs.columbia.edu

Kostis Kaffes
Columbia University
kkaffes@cs.columbia.edu

Abstract

Agentic exploration, letting LLM-powered agents branch, backtrack, and search across many execution paths, demands systems support well beyond today's pass-@-k resets. Our benchmark of six snapshot/restore mechanisms shows that generic tools such as CRIU or container commits are not fast enough even in isolated testbeds, and they crumble entirely in real deployments where agents share files, sockets, and cloud APIs with other agents and human users. In this talk, we pinpoint three open fundamental challenges: fork semantics, which concerns how branches reveal or hide tentative updates; external side-effects, where fork awareness must be added to services or their calls intercepted; and native forking, which requires cloning databases and runtimes in microseconds without bulk copying.

1 Agentic Exploration

Large Language Models (LLMs) under an interaction–feedback paradigm have demonstrated strong performance on everyday tasks, where prior interactions determine following actions [8, 15]. More recently, LLM-powered agents functioning as system agents are used to interact directly with real computing environments, such as operating systems and development toolkits [4, 7, 11, 13, 16]. These tasks often require actions that can alter the state, e.g., an application, a database, a language runtime, or an operating system, making the problem a partially observable, multi-step decision process. Consequently, effective exploration—where an agent actively interacts with a stateful environment, observes the outcome of its actions, and making better multi-step strategies—becomes critical. On Terminal-Bench's [11] command-line tasks, disabling exploration reduces accuracy by 27.2 percentage points (30.6% → 3.4%).

From pass@k to real exploration. Most exploration-based agent frameworks implicitly assume that the environment can be restored to an initial reference state such that applying the same set of actions leads to identical observations [1, 12]. This guarantees that exploration outcomes on alternative branches are valid and reproducible. In practice, this is satisfied by benchmark harnesses that deliver deterministic initial states and allow programmatic resets. For example, WebArena[16] constructs each website as a self-contained Docker image and provides scripts to reset to the initial state, and OSWorld[13] offers task-specific initial-state setup and uses VM images to recover the initial state. In these scenarios, the baseline is always to pass@k from a clean state: for each trial the harness resets to a pristine snapshot and allows the agent to act until success/failure.

While the pass@k method is simple and works well when per-step overheads are small or tasks are short, it performs poorly on more complex, long-horizon, and realistic tasks because agents tend to make mistakes due to losing sight of ultimate goals and cumulative errors, i.e., non-observable state changes, on long-horizon tasks [5]. The most common solution to this problem is taking into account intermediate states and doing exploration over them. For instance, Reflective Monte Carlo Tree Search (MCTS) [14] augments tree search with budgeted rollouts and reward backtracking, and HiAgent[6] uses hierarchical decomposition with task-level checkpointing, both obtained impressive performance gains in long-horizon tasks. On terminal-bench, simply allowing for searching from intermediate states significantly improves agent performance: We observe a 20 percentage point increase in success rate on a selected subset of tasks when applying MCTS instead of the baseline pass@2 method with claude 3.5 sonnet model.

2 Exploration as State Restoration

Instead of always resetting to a clean initial state, supporting exploration from intermediate states requires digital environments to resume execution from that point onward. The system infrastructure must ensure that replaying or branching from this state produces observations consistent with the original execution. Systems can support such agentic exploration with three different primitives (as displayed in Figure1):

(1) *Replay-to-node (prefix replay).* For every explored search-tree node, the runtime records the command prefix needed to reach the node from the initial state. Revisiting the node is achieved by re-executing the command prefix, obviating any explicit state capture but incurring replay cost proportional to the length of the command prefix and the overhead of the individual commands.

[*] Both authors contributed equally to this research.

(2) *Snapshot/Restore.* Alternatively, the system can materialize a snapshot at each node and reload it on demand, trading storage overhead for $O(1)$ restoration latency.

(3) *Backtracking.* For any operation o_i made that shifts the environment state $S_i \longrightarrow S_i'$, a compensation operation $c_i = reverse(o_i)$ is pre-defined that shifts $S_i' \longrightarrow S_i$. Restoration amounts to reversing all of the intermediate nodes, but relies on pre-defined logic [2].

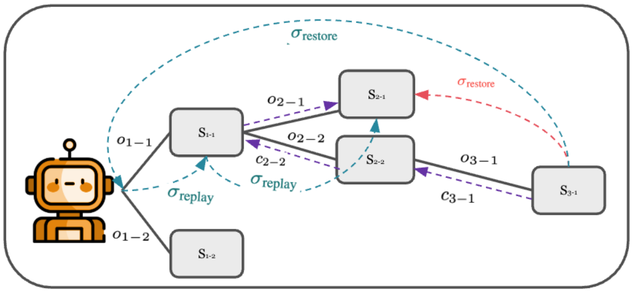

Figure 1. An LLM agent (orange) explores by taking different actions, creating a branching tree in which every node represents a distinct state of the environment. A prefix replay (teal) starts from root and replay all commands on record; A snapshot/restore (red) checkpoint method restores the state directly to the target; a backtracking method (purple) goes through all intermediate nodes to the target node.

Unlike humans, agent exploration involves frequent state switching, which requires high-fidelity state restoration. Backtracking method, in practice, is therefore challenging because many system-level operations are inherently irreversible (e.g., file deletion, network I/O, time-sensitive actions). Therefore, the backtracking method is not good for general-purpose exploration. Thus, a reliable agent system framework must at least provide a minimal form of snapshot/restoration, ensuring that distinct exploration operations can be conducted with consistent observations.

In the simplest agent settings—such as a dialog agent whose entire world state is the conversation log—snapshotting reduces to persisting that log, so recording the full state is nearly trivial. In richer cases where the agent manipulates a stateful environment, e.g., external software or operating-system resources, snapshotting must encompass the full execution context. At minimum, this includes:

- **Filesystem** — to preserve file modifications, e.g., installed packages and intermediate artifacts in long-running tasks.
- **Memory** — to retain application and kernel state, e.g., heap memory.

Thus, a full system snapshot is necessary to enable seamless, multi-path exploration in realistic tasks. This insight motivates the benchmark study that follows.

2.1 Existing Technologies and Benchmark

We measured snapshot and restore latency as the amount of modified application memory or filesystem contents are independently varied from $0GB$ to $2GB$. We compared six commonly-used mechanisms: **CRIU**[3], **Docker**, **Podman**, **Hybrid** (Podman + CRIU checkpoint), **AWS VM snapshots**, and our prototype **checkpoint-lite** (CRIU + OverlayFS). Table 1 summarizes the results, and five trends stand out:

1. **AWS VMs**, used by OS-World [13], are extremely slow to re-instantiate as they are not built for this purpose.
2. **Docker/Podman commits**, used by WebArena [16], AgentBench [9], and Terminal-Bench [11], rebuild containers from image layers, *i.e.*, filesystem state only and therefore lose live memory. Startup latencies can exceed 10 s, making them unsuitable for fine-grained agentic exploration.
3. **CRIU** offers fast memory snapshots by dumping process memory and metadata to a file, snapshotting and restoring 2 GiB process in 1.445 s, but snapshot cost rises linearly with memory and it still ignores files.
4. **Hybrid (Podman checkpoint)** integrates CRIU with container runtimes to capture memory and network, but restore times remain high (up to 12 s for 2 GiB).
5. Our Go-based **checkpoint-lite** prototype orchestrates CRIU dumps alongside OverlayFS layer snapshots, achieving near-CRIU times (1.757 s for 2 GiB state) while also preserving filesystem state.

However, even before factoring in storage costs, existing checkpoint/restore tools impose second-scale overheads, making them unsuitable for rapid agentic exploration. Worse, they are missing critical features that we show next are essential for environment-agnostic agentic exploration.

3 The Missing Pieces

From snapshot/restore to native forking. What agentic exploration really demands is not generic *snapshot/restore* but a lightweight, *native fork* primitive: the ability to spin off multiple live logical copies of a running application or system without duplicating unchanged data. Conceptually, it resembles `fork()` in Unix—copy-on-write pages, lazy duplication—but extended to encompass higher-level resources. Unlike traditional OS forks, an agent-targeted fork must duplicate open file descriptors *semantically*: a child's write to a socket should not corrupt the parent's stream, and diverging file writes should land in per-branch overlays that can later merge or discard cleanly. Achieving this requires tighter integration between the OS, the storage stack, and the language runtime so that forking incurs microseconds of latency rather than the milliseconds or seconds we observe with coarse snapshotting.

Generic, system-wide forking is useful, but some subsystems benefit from domain-specific support. Databases are a prime example: Neon's "branching" Postgres [10] clones let developers fork a live logical database, yet each branch takes

Table 1. Snapshot and restore time (in seconds) across different tools and configurations.

Operation	Memory	Disk	criu	Docker	Podman	checkpoint-lite	Hybrid	AWS-VM
Snapshot + Restore	0 GB	0 GB	0.060	0.416	0.835	0.418	1.657	353
Snapshot + Restore	1 GB	0 GB	0.760	/	/	1.079	9.921	-
Snapshot + Restore	2 GB	0 GB	1.445	/	/	1.757	18.154	-
Snapshot + Restore	0 GB	1 GB	/	5.097	7.935	2.499	14.735	-
Snapshot + Restore	0 GB	2 GB	/	6.915	12.914	4.622	26.648	-

Tested on a Linux server with 56-core Intel® Xeon® Gold 5512U CPU, 128GB RAM, running Ubuntu 24.04.2 LTS with Linux kernel 6.8.0.
Tool versions: CRIU 4.1, Docker 27.5.1, Podman 4.9.3, runc 1.2.5. checkpoint-lite is our own Go-based tool using CRIU + OverlayFS.

seconds to materialize––far slower than the sub-millisecond forks agents would need for interactive branching. Similar gaps appear in language runtimes: Python's multiprocessing fork inherits bytecode and heap, but extension modules holding GPU tensors or open sockets do not survive, forcing full re-initialization. Bridging this gap calls for *native fork hooks* inside components––database engines that version page caches in micro-seconds and runtimes that expose copy-on-write heaps. Building such primitives pushes the responsibility down to where the semantics are understood, yielding fork operations that are both correct and fast enough to unlock large-scale, multi-path agentic exploration.

From benchmarks to the real world. The snapshot mechanisms described above suffice only for *isolated* benchmarking environments. Real deployments couple agents to databases, browsers, and cloud APIs whose state lives beyond the local filesystem or RAM. The simplest example is a live socket: restoring a checkpoint invalidates the TCP sequence numbers, auth tokens, or DOM tree held by the remote peer. Thus, we need to develop methods to enable general-purpose agentic exploration without sacrificing correctness at scale. One such example could be to expose *fork-aware APIs* whose side effects are intrinsically versioned—much like S3's object-versioning, where each branch writes to an immutable commit rather than mutating shared state.

Semantics of multi-agent exploration. In production settings, multiple autonomous agents—and often live human users—operate on the same resources at once, so the key semantic question is what those other actors should observe while one agent is branching speculatively. A conservative design might reveal only *committed* trajectories, hiding tentative side effects until they are finalized; this preserves serial consistency but can cause costly merge conflicts. A more optimistic design could let agents fork atop one another's in-flight trajectories, promoting richer collaboration yet exploding the state space combinatorially.

Acknowledgments

This work was supported by the National Science Foundation (1845638, 1740305, 2008295, 2106197, 2103794, 2312991) and DAPLab funders (Amazon, Google, Intellect Design, Tidalwave, Veris).

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[2] Edward Y. Chang and Longling Geng. 2025. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. arXiv:2503.11951 [cs.AI] https://arxiv.org/abs/2503.11951

[3] CRIU Project. 2012. Checkpoint/Restore In Userspace (CRIU). https://criu.org/. Accessed: 2025-08-06.

[4] Aleksandra Eliseeva, Alexander Kovrigin, Ilia Kholkin, Egor Bogomolov, and Yaroslav Zharov. 2025. EnvBench: A Benchmark for Automated Environment Setup. arXiv:2503.14443 [cs.LG] https://arxiv.org/abs/2503.14443

[5] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks. arXiv:2503.09572 [cs.CL] https://arxiv.org/abs/2503.09572

[6] Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024. HiAgent: Hierarchical Working Memory Management for Solving Long-Horizon Agent Tasks with Large Language Model. arXiv:2408.09559 [cs.CL] https://arxiv.org/abs/2408.09559

[7] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

[8] Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. AlpacaEval: An Automatic Evaluator of Instruction-following Models. https://github.com/tatsu-lab/alpaca_eval.

[9] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688 [cs.AI] https://arxiv.org/abs/2308.03688

[10] Neon Database. 2025. *Neon: Serverless Postgres.* https://github.com/neondatabase/neon GitHub repository.

[11] The Terminal-Bench Team. 2025. Terminal-Bench: A Benchmark for AI Agents in Terminal Environments. https://github.com/laude-institute/terminal-bench

[12] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032 [cs.LG] https://arxiv.org/abs/2407.17032

[13] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. arXiv:2404.07972 [cs.AI] https://arxiv.org/abs/2404.07972

[14] Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2025. ExACT: Teaching AI Agents to Explore with Reflective-MCTS and Exploratory Learning. arXiv:2410.02052 [cs.CL] https://arxiv.org/abs/2410.02052

[15] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL] https://arxiv.org/abs/2306.05685

[16] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854* (2023). https://webarena.dev