Position Paper: Crash-Safe Effects for Agents in Operational Systems

Hannaneh B. Pasandi
University of California, Berkeley
Berkeley, CA, USA
h.pasandi@berkeley.edu

Mohammad Sepahi
Old Dominion University
Norfolk, VA, USA
msepa001@odu.edu

Abstract

Agents have begun to operate computer systems. They migrate databases, roll out configurations, and remediate incidents, and the process that runs them can die mid-plan. When it does, neither obvious recovery is safe. Retry re-executes steps against a world its own earlier actions already changed. Restart begins with no record of which effects became true. Our position is that every side-effecting tool call must carry an *effect contract* made of three fields, an effect identifier, an idempotency key, and a declared reversibility class. The runtime writes the contract to a durable intent log before the call and reconciles that log on restart. We implement the log as EffectLog, 39 lines of Python over SQLite, and evaluate it on a thirteen-step database migration under process-kill injection. Across 320 measured runs, naive recovery leaves duplicate rows in up to 66% of migrations and re-executes a destructive step in up to 59%. Idempotency keys remove the corruption but still take 14.1 relaunches per migration. Reconciling from the log completes in 3.5 relaunches, at 2.8 ms of logging per effect. An agent that cannot show what it did must not be allowed to try again.

CCS Concepts: • **General and reference** → **Reliability**.

Keywords: AI agents, crash recovery, idempotency, reversibility, tool calls

ACM Reference Format:
Hannaneh B. Pasandi and Mohammad Sepahi. 2026. Position Paper: Crash-Safe Effects for Agents in Operational Systems. In *Practical Adoption Challenges of ML for Systems (PACMI '26), September 29–October 2, 2026, Prague, Czech Republic.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3843967.3844688

1 Introduction

An LLM agent is two hours into a thirteen-step production database migration [10]. Steps 1 through 10 analyze the schema and replicate the tables. Steps 11 and 12 are the point of no return, where the old tables are truncated and DNS is cut to the new endpoint. At step 13 the process dies on an API rate limit. Figure 1 draws the run. The top lane is the agent's plan, thirteen chips with twelve done. The bottom lane is the world the plan itself has moved. Tables are truncated, DNS is cut, and live traffic is already on the new stack. What should the agent do next?

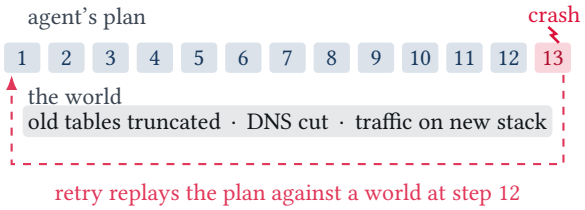

Figure 1. The crash at step thirteen. Retry assumes the world of step 1; the world is at step 12.

Neither move is safe, and for mechanical reasons. Retry assumes the world of step 1, so its early steps re-execute against tables that no longer hold rows and its late steps repeat actions that must not repeat. Restart comes up knowing nothing, since the only record of the run lived in the dead process's context window. This scenario is not exotic. Production has already supplied the incident report of an agent deleting the database it had just been operating [33]. Deployed agents already work across 26 domains with a human kept in the loop, precisely because autonomous action is not trusted [19]. Termination and verification failures lead the multi-agent failure taxonomy [6], and sandboxed testing finds even the safest agents taking severe-outcome actions 23.9% of the time [28]. What breaks is not the model's planning. The execution layer gives the agent no way to make its effects crash-safe, so no operator will hand infrastructure to a process that cannot answer, after a crash, *what did you already do*. Decades of systems machinery, write-ahead logs, sagas, exactly-once execution [12, 16, 17], exist to prevent exactly this. Why has none of it carried over?

Because each classical remedy assumes something the agent setting removes. Transactions make the world hold still inside one transaction domain, even a planet-spanning one [8]. An agent's calls cross the database here, DNS there, and a ticket system elsewhere, and no manager spans them. Sagas run forward with a compensating action per step [12], but assume compensators authored in advance for an enumerated workflow, and nobody wrote the compensator for step 11 because nobody knew step 11 would exist. Durable

PACMI '26, Prague, Czech Republic

ACM ISBN 979-8-4007-2998-0/26/09
https://doi.org/10.1145/3843967.3844688

workflow engines event-source every step and replay deterministically [3, 32]. An LLM agent is not a pure function of its history, because its next step depends on sampled output and fresh observation of a changed world. What is missing is not a smarter policy. It is a durable record for one to read.

Our position is that agentic automation of operations will not cross the adoption line until every side-effecting tool call carries an *effect contract*. The contract names three fields, an effect identifier, an idempotency key, and a declared reversibility class. The runtime writes them to a durable intent log before the call and reconciles that log on restart. The lever already exists. The tool boundary is the one place where the runtime mediates every effect, survives the crash, and owes no allegiance to any tool's administrative domain. A log written there turns recovery from replay into reconciliation. Recovery probes what became true, then resumes against the world as it is, never the world of step 1. The contract is deliberately mechanism, not policy, in the sense the exokernel line gave those words [11]. The durable layer records intents and outcomes. Deciding whether to retry, compensate, or page a human stays above it, in code an operator can read.

We built it. EffectLog is a durable intent log in 39 lines of Python over SQLite, driven by a 132-line harness that runs the migration of Figure 1 against on-disk state and kills the worker mid-plan. Across 320 measured runs, naive recovery leaves duplicate rows in up to 66% of migrations and re-executes a destructive step in up to 59%. Idempotency keys eliminate both but still relaunch 14.1 times per migration at 20% acknowledgment loss, since a lost key reads as work not done. Reconciliation from the log completes the same migration in 3.5 relaunches, 6.2× fewer, at 2.8 ms of logging per effect. Knowing what became true turned out to be the fast path, not only the safe one. We make three contributions.

- We name the execution gap precisely along three failure axes, duplicate effects on retry, stale-world assumptions on replay, and unknown state on restart, and we show why transactions, sagas, and durable workflow engines each fail to transfer to agents (§3).
- We implement EffectLog and evaluate it on a live thirteen-step migration under process-kill injection, 80 trials per operating point, with one measured takeaway per failure axis (§4).
- We specify the effect contract, its fields, its intent-log protocol, and a reversibility taxonomy that determines which actions an agent may retry alone and which require a human gate (§5).

Scope and broader impact. We claim the shape of the contract, not the optimality of any policy above it. The prototype is single-agent and single-host, the injector kills the process only between atomic effects, and compensator drafting remains a hypothesis (§6). Typing effects by reversibility lets autonomy grow class by class. The human gates the irreversible class, not every step, which bounds blast radius as agents take on more. The same log that makes recovery safe is also the ledger an operator audits, so accountability and autonomy share one artifact. The risk is misplaced trust. A mislabeled class turns a gate into a hole, so the taxonomy must degrade to safety, and in our design it does. If deployed agents cross the adoption line on workflow-engine wrapping alone, this position is wrong, and that is measurable.

2 Why Now

Three lines are converging on this gap. Agents have entered operations, with two thirds of surveyed agents tolerating minute-scale latency, exactly the long-running, effectful regime where crashes are a matter of when [19, 20], on benchmarks that already pose long-horizon effectful tasks [15, 37]. The frameworks that carry them [26, 29, 30, 34, 36, 38] treat a tool call as a stateless function invocation, and nothing distinguishes *the request failed* from *the request succeeded and the acknowledgment was lost*, the oldest distinction in distributed computing. And the agent-runtime community is converging on durable, rewindable state for the agent's own computation [10, 35], while OS-side work meters its resource footprint [7, 40, 41] and our companion work prices what a replayed plan costs in carbon [21, 22, 24]. None of it covers the agent's effects on the world outside the harness. Rewind restores the agent's memory. It does not restore the database.

3 The Execution Gap

The gap has three axes, and each defeats a different classical remedy.

Duplicate effects on retry. If step 7, replicate table T, runs twice, the damage ranges from wasted work to corruption. The classical answer is exactly-once execution, meaning at-least-once delivery plus deduplication by a request identifier. RIFL built this mechanism for datacenter RPC [16], and payment APIs expose it as idempotency keys [14, 31]. It does not transfer as-is. The agent's calls cross administrative boundaries, the database here, DNS there, the ticket system elsewhere, and no shared component deduplicates across them. The identifier must travel with the call, minted by the runtime and honored, or at least logged, at each boundary. Configuration management learned the same lesson as convergence to desired state [4].

Stale-world assumptions on replay. Even with deduplication, replaying the plan is unsafe, since its preconditions were computed against the world of step 1. Transactions solve this inside one system by making the world hold still [8], but an agent's world spans systems that share no transaction manager. Sagas [12] answered this for long-lived workflows, running forward with a compensating action per step, and Helland's argument [13] explained why beyond a

single system this is the only workable shape. But sagas assume compensators authored in advance for an enumerated workflow, and an agent's plan is generated at run time.

Unknown state on restart. The restarted agent must answer *what became true*, and today the answer lives only in the dead process's context window. Durable workflow engines [32], durable functions [3], and transactional serverless workflows [39] solve exactly this for code, event-sourcing every step so replay reconstructs state deterministically. They are the closest existing answer, so we take them head on. Determinism is the load-bearing assumption. The workflow must be a pure function of its history, and an LLM agent is not, because its next step depends on sampled output and fresh observation of a changed world. Wrapping the agent in a workflow engine preserves the log but not the semantics; process checkpointing [9] restores memory, not the world, the same limit as harness rewind. Crash-only software [5] and operator-facing undo [2] anticipated the stance, but manage a system's own state, not a world of foreign side effects. Journaling before acting is itself old, from write-ahead logging [17] to log-structured filesystems [27]; what has no prior home is a record of intents about a *foreign* world, typed by reversibility and recovered by probing rather than replaying (§5).

4 Prototype and Evaluation

We built it, then we crashed it, 320 times per recovery mode.

Setup. EffectLog is a durable intent log in 39 lines of Python over SQLite in write-ahead mode.[1] It writes four records per effect, *intend*, *attempt*, *observe*, *resolve*, plus a recovery call returning unresolved intents. The 132-line harness runs the thirteen-step migration of Figure 1 against on-disk state rather than mocked stubs, an application database, a source table, a cutover file, and a notification log. Crash injection SIGKILLs the worker after a step's effect commits and, in the acknowledgment-loss case, before its observe record persists, so the effect happened but no record of it survived. A parent process relaunches until the migration completes, then audits the disk for duplicate rows and destructive re-executions. Three modes run under identical injection. Naive restart replays from step 1. Idempotency keys are checked at the tool boundary. EffectLog reconciliation probes each unresolved intent against the world and resumes from the frontier, the first step whose effect is not yet known to have committed. Four operating points pair a per-step crash rate c with an acknowledgment-loss rate q, namely c=2%+q=5%, c=5%, c=10%, q=20%, at 80 trials each. Scripts and raw runs are at https://github.com/hanabhp/CrashSafe.

[1]Line counts exclude blank lines and comments; the counting script ships with the artifact. The log runs with `PRAGMA synchronous=FULL`, an fsync at every commit, so its durability rests on the same assumption every write-ahead system makes.

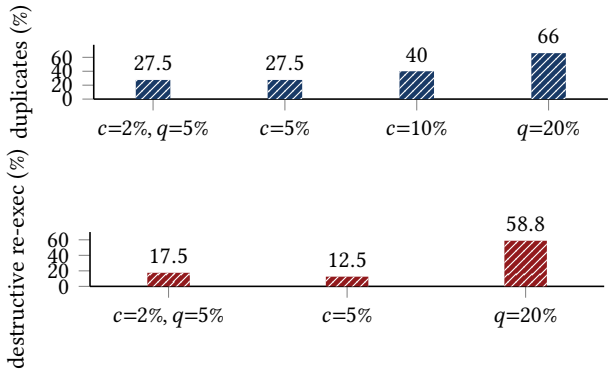

Figure 2. Naive restart, measured. Duplicate rows (top) and destructive re-execution (bottom) across operating points. Keys and EffectLog sit at zero in every run of both.

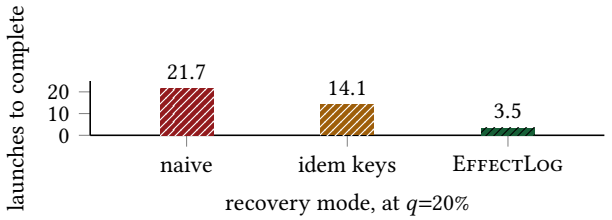

Figure 3. Mean launches per completed migration at 20% acknowledgment loss.

Naive restart, measured. Figure 2 shows the share of completed migrations with duplicate rows (top) and destructive re-executions (bottom) under naive restart. Neither failure is confined to extreme settings. Even a modest 2% per-step crash rate with 5% acknowledgment loss leaves 27.5% of migrations with duplicated rows, rising to 66% (mean 49.8 duplicate rows) at 20% acknowledgment loss, where destructive re-execution also peaks at 58.8%, the source table deleted a second time while the cutover is already live. With idempotency keys or EffectLog the count is zero in all 320 runs of each. The boundary refuses what the identifier marks as done, and EffectLog's recovery probes the world before it acts.

Takeaway 1. *Naive restart leaves duplicate rows in up to 66% of runs and re-executes a destructive step in up to 58.8%; an idempotency key or EffectLog drops both to zero in every one of 320 runs.*

Recovery cost, measured, and a result we did not expect. Correctness was the design goal; the launches column delivered a second finding. At 20% acknowledgment loss, naive restart needs 21.7 launches per completed migration and idempotency keys still need 14.1, while EffectLog completes in 3.5, a 6.2× reduction (Figure 3). Logging costs 2.8 ms per effect for four durable records, three orders of magnitude below the tool latencies of a production migration. The log is not only the safety mechanism; it is the cheaper way to finish.

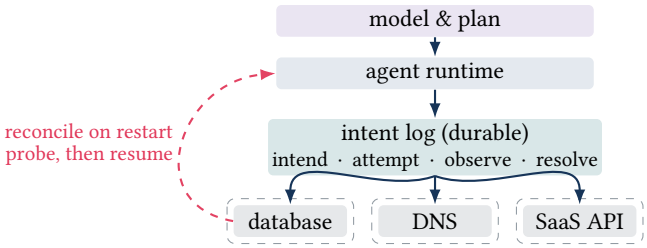

Figure 4. The contract lives at the tool boundary; recovery is reconciliation, not replay. Dashed outlines mark separate administrative domains.

Takeaway 2. *Keys buy safety, not recovery. At 20% acknowledgment loss naive restart relaunches 21.7 times and keyed restart 14.1, while probe-then-resume completes in 3.5, a 6.2× reduction.*

Assumptions and limitations. Four points, stated plainly. *Crash model.* The injector kills the process after a step's effect commits, or before its acknowledgment persists; it never tears a write inside an effect, since every effect here is one atomic operation, and a tool without that property must wrap its effect or declare it FENCED. *Key placement, a bug we hit.* An early version committed the key and the effect in separate transactions, so a crash could record the key without the effect and deduplication then skipped real work; the hardened version commits both in one transaction for database steps, and for file steps writes the key just after, at-least-once, where a duplicate is possible but a lost effect is not. *Log durability.* If the log is unavailable, corrupted, or lost with the process, recovery degrades to naive restart, never below it. *The open window.* The world can change between a probe and the resumed action; a fencing token is the classical answer, and stays future work. At 80 trials per point, binomial 95% confidence intervals stay within eleven percentage points.

5 The Effect Contract

Every side-effecting tool call carries three fields. `effect_id` names this intended effect, `idem_key` is the deduplication token the tool boundary honors, and `reversibility` is one of IDEMPOTENT, REVERSIBLE, FENCED, or IRREVERSIBLE. The runtime keeps one log (Figure 4).

The protocol. The runtime writes four log records per effect to durable storage. *Intend* is written before the call and carries the three fields and the assumed precondition. *Attempt* marks that the call was issued with the idempotency key. *Observe* records the outcome as witnessed, including an unknown fate. *Resolve* closes the effect as committed, compensated, or escalated. Recovery is the reconciliation loop that §4 prices. It reads intents without resolutions, probes the world for each, and then resumes planning against the reconciled world, never the world of step 1. This is a write-ahead log for reality, the oldest idea in the room [17], applied one layer higher.

The reversibility taxonomy is where policy attaches, typing effects the way dependability engineering types faults and failures [1]. IDEMPOTENT effects retry freely. REVERSIBLE effects carry a compensator reference the planning model can draft at plan time, validated like any verification step [21, 22], giving GoEX's undo posture [25] a durable record. FENCED effects are safe only under an ordering token the boundary checks, the truncate-then-cutover pair of steps 11 and 12 being the case in point. IRREVERSIBLE effects, and only they, hard-require a human gate, turning the vague instinct of keeping a human in the loop [19] into a typed rule.

Placement. In the exokernel split between mechanism and policy [11], the contract is mechanism. It belongs in the durable layer beneath the model and above the tools, the only component that sees every effect, survives the crash, and owes allegiance to no tool's domain. A boundary that honors keys gives exactly-once. One that does not still gets intent-logged and reconciled by observation.

6 Discussion and Future Outlook

The workflow-engine objection. The most natural pushback is to wrap the agent in a durable workflow engine. Section 3 answers it. Determinism fails for sampled, world-observing agents, and engine-level exactly-once covers invocation, not undoability. The contract borrows the engines' durability discipline [3, 32] and drops the replay semantics they cannot keep. A recent startup applies the same instinct to agent actions [18]. The difference is scope. The contract types reversibility and travels with the call across administrative boundaries that a transactional shell cannot span.

Open hypothesis. Plan-time compensator drafting (§5) is a hypothesis. A draft that passes the tool's schema and the verifier machinery agents already run stands. Where none validates, the effect is gated as IRREVERSIBLE, so the taxonomy degrades to safety, not optimism. Two findings would refute us, workflow-engine wrapping alone carrying agents across the adoption line, or compensators that rarely validate, which would empty the REVERSIBLE class. Measuring per-domain effect-class mixes in deployed harnesses [22, 23, 34] is first on our list.

7 Conclusion

In this work, we argued that operational agents need a crash-safe effect contract, an effect identifier, an idempotency key, and a reversibility class, logged before each call and reconciled on restart. We implemented it as EffectLog, a 39-line intent log, and evaluated it under process-kill injection on a thirteen-step migration. We showed that naive recovery corrupts up to 66% of runs, while reconciling from the log eliminates corruption and completes in 3.5 relaunches.

References

[1] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (2004), 11–33. doi:10.1109/TDSC.2004.2

[2] Aaron B. Brown and David A. Patterson. 2003. Undo for Operators: Building an Undoable E-mail Store. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '03)*. USENIX Association. https://www.usenix.org/legacy/events/usenix03/tech/brown.html

[3] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S. Meiklejohn. 2021. Durable Functions: Semantics for Stateful Serverless. *Proceedings of the ACM on Programming Languages* 5, OOPSLA, Article 133 (2021). doi:10.1145/3485510

[4] Mark Burgess. 1995. Cfengine: A Site Configuration Engine. *USENIX Computing Systems* 8, 3 (1995). https://www.usenix.org/publications/compsystems/1995/sum_burgess.pdf

[5] George Candea and Armando Fox. 2003. Crash-Only Software. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*. USENIX Association. https://www.usenix.org/legacy/events/hotos03/tech/candea.html

[6] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why Do Multi-Agent LLM Systems Fail? *arXiv preprint arXiv:2503.13657* (2025). doi:10.48550/arXiv.2503.13657

[7] Jingyuan Chen, Gongqi Huang, and Amit Levy. 2026. Towards Agentic Performance Management. In *1st Workshop on Agentic Operating Systems (AgenticOS '26)*.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems* 31, 3, Article 8 (2013). doi:10.1145/2491245

[9] CRIU Project. 2026. CRIU: Checkpoint/Restore In Userspace. https://criu.org. Accessed July 2026.

[10] Guanlan Dai. 2026. Agents Are Not Just a Model Problem. They Are an Execution Problem. Keynote, 1st Workshop on Agentic Operating Systems (AgenticOS '26).

[11] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole, Jr. 1995. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*. Association for Computing Machinery, 251–266. doi:10.1145/224056.224076

[12] Hector Garcia-Molina and Kenneth Salem. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 249–259. doi:10.1145/38713.38742

[13] Pat Helland. 2007. Life Beyond Distributed Transactions: An Apostate's Opinion. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR '07)*. https://www.cidrdb.org/cidr2007/papers/cidr07p15.pdf

[14] Pat Helland. 2012. Idempotence Is Not a Medical Condition. *Commun. ACM* 55, 5 (2012), 56–65. doi:10.1145/2160718.2160734

[15] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *Proceedings of the 12th International Conference on Learning Representations (ICLR '24)*. doi:10.48550/arXiv.2310.06770

[16] Collin Lee, Seo Jin Park, Ankita Kejriwal, Satoshi Matsushita, and John Ousterhout. 2015. Implementing Linearizability at Large Scale and Low Latency. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*. Association for Computing Machinery, 71–86. doi:10.1145/2815400.2815416

[17] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. 1992. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Transactions on Database Systems* 17, 1 (1992), 94–162. doi:10.1145/128765.128770

[18] OneWill. 2026. Stealing 50 Years of Database Ideas for AI Agents. https://onewill.ai/blog/2026/stealing-50-years-of-database-ideas-for-ai-agents/. Accessed July 2026.

[19] Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A. Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, Shu Liu, Tianneng Shi, Xiaoyuan Liu, Jared Quincy Davis, Emmanuele Lacavalla, Alessandro Basile, Shuyi Yang, Paul Castro, Daniel Kang, Koushik Sen, Dawn Song, Joseph E. Gonzalez, Ion Stoica, Matei Zaharia, and Marquita Ellis. 2026. Measuring Agents in Production. *arXiv preprint arXiv:2512.04123* (2026). doi:10.48550/arXiv.2512.04123

[20] Melissa Z. Pan, Yuxuan Zhu, Jared Quincy Davis, Riccardo Cogo, Lakshya A. Agrawal, Negar Arabzadeh, Xiaoyuan Liu, Huanzhi Mao, Sid Pallerla, Tianneng Shi, Alexander Xiong, Alessandro Basile, Emmanuele Lacavalla, Shuyi Yang, Diana Arroyo, Paul Castro, Daniel Kang, Joseph E. Gonzalez, Koushik Sen, Dawn Song, Ion Stoica, Matei Zaharia, and Marquita Ellis. 2025. Useful Agentic AI: A Systems Outlook. In *Proceedings of the 1st Workshop on Systems for Agentic AI (SAA '25), co-located with SOSP '25*. Seoul, Republic of Korea.

[21] Hannaneh B. Pasandi. 2026. Patient Packets. (2026). Companion paper, under submission.

[22] Hannah B. Pasandi, Negar Arabzadeh, and Melissa Pan. 2026. PlanLayer: Making the Plan the Unit of Accounting for Agentic AI. (2026). Under submission, 5th Workshop on Practical Adoption Challenges of ML for Systems (PACMI '26).

[23] Hannah B. Pasandi, Mohammad Hosseini, and Sina Darabi. 2026. Learning on the GPUs the World Owns: A Fleet Contract for Learned Systems on Commodity Hardware. (2026). Under submission, 5th Workshop on Practical Adoption Challenges of ML for Systems (PACMI '26).

[24] Hannaneh B. Pasandi and Tamer Nadeem. 2026. The Reasoning Tax: Profiling Test-Time Compute Carbon in Agentic AI Workloads. *ACM SIGENERGY Energy Informatics Review* 6, 2 (June 2026).

[25] Shishir G. Patil, Tianjun Zhang, Vivian Fang, Noppapon C., Roy Huang, Aaron Hao, Martin Casado, Joseph E. Gonzalez, Raluca Ada Popa, and Ion Stoica. 2024. GoEX: Perspectives and Designs Towards a Runtime for Autonomous LLM Applications. *arXiv preprint arXiv:2404.06921* (2024). doi:10.48550/arXiv.2404.06921

[26] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems 37 (NeurIPS '24)*. doi:10.48550/arXiv.2305.15334

[27] Mendel Rosenblum and John K. Ousterhout. 1992. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems* 10, 1 (1992), 26–52. doi:10.1145/146941.146943

[28] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *Proceedings of the 12th International Conference on Learning Representations (ICLR '24)*. doi:10.48550/arXiv.2309.15817

[29] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems 36*

(NeurIPS '23). doi:10.48550/arXiv.2302.04761

[30] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS '23)*. doi:10.48550/arXiv.2303.11366

[31] Stripe, Inc. 2026. Idempotent Requests. Stripe API Documentation, https://stripe.com/docs/api/idempotent_requests. Accessed July 2026.

[32] Temporal Technologies. 2026. Temporal: Durable Execution Platform. https://temporal.io. Accessed July 2026.

[33] Tom's Hardware. 2026. Claude-Powered AI Coding Agent Deletes Entire Company Database in 9 Seconds. https://www.tomshardware.com/tech-industry/artificial-intelligence/claude-powered-ai-coding-agent-deletes-entire-company-database-in-9-seconds-backups-zapped-after-cursor-tool-powered-by-anthropics-claude-goes-rogue.

[34] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, et al. 2024. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *arXiv preprint arXiv:2407.16741* (2024). doi:10.48550/arXiv.2407.16741

[35] Yuan Wang, Mingyu Li, and Haibo Chen. 2026. Rethinking OS Interfaces for LLM Agents. In *1st Workshop on Agentic Operating Systems (AgenticOS '26)*.

[36] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155* (2023). doi:10.48550/arXiv.2308.08155

[37] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *Proceedings of the 13th International Conference on Learning Representations (ICLR '25)*. doi:10.48550/arXiv.2406.12045

[38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR '23)*. doi:10.48550/arXiv.2210.03629

[39] Haoran Zhang, Adney Cardoza, Peter Baile Chen, Sebastian Angel, and Vincent Liu. 2020. Fault-Tolerant and Transactional Stateful Serverless Workflows. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association. https://www.usenix.org/conference/osdi20/presentation/zhang-haoran

[40] Yusheng Zheng, Jiakun Fan, Quanzhi Fu, Yiwei Yang, Wei Zhang, and Andi Quinn. 2026. AgentCgroup: Understanding and Controlling OS Resources of AI Agents. In *1st Workshop on Agentic Operating Systems (AgenticOS '26)*.

[41] Yusheng Zheng, Yanpeng Hu, Tong Yu, and Andi Quinn. 2025. AgentSight: System-Level Observability for AI Agents Using eBPF. In *Proceedings of the 4th Workshop on Practical Adoption Challenges of ML for Systems (PACMI '25)*. Association for Computing Machinery.