Short: Plan as the Unit of Accounting in Agentic AI

Hannah B. Pasandi
UC Berkeley
Berkeley, USA
h.pasandi@berkeley.edu

Negar Arabzadeh
UC Berkeley
Berkeley, USA
negara@berkeley.edu

Melissa Pan
UC Berkeley
Berkeley, USA
melissapan@berkeley.edu

Abstract

Operators meter agentic AI one model call at a time, because the call is what the serving stack and the invoice expose. The unit of work is the plan: one request expands into generator, verifier, retry, and judge calls, so the dashboard's numbers are decided one layer above the layer that records them. We present PlanLayer, a thin runtime layer that stamps three fields on every call: a plan identifier, a stage role, and a commit flag set at plan close. Four plan-level accounts follow by aggregation. On a 1,000-task workload calibrated to published benchmarks and a production survey, per-call accounting reports as productive 12.5% of compute the agent later threw away, 19.5% for software-engineering agents; plan-level audits reach 50% of spend against 21% for call-level audits; and a per-plan retry budget of two keeps 98% of task success at 54% of the abandoned work. No telemetry convention or observability platform records any of the three fields today.

CCS Concepts: • **Computer systems organization → Reliability**; • **Computing methodologies → Artificial intelligence**.

Keywords: agentic AI, accounting, observability, agent runtime, ML for systems

ACM Reference Format:
Hannah B. Pasandi, Negar Arabzadeh, and Melissa Pan. 2026. Short: Plan as the Unit of Accounting in Agentic AI. 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/10.1145/3843967.3844679

1 Introduction

A coding agent picks up a ticket. It reads the repository, drafts a patch, and runs the tests. The tests fail. The agent retries three times, throws the attempts away, revises its approach, and calls a second model to confirm the fix. One request has become fourteen calls, and the dashboard shows fourteen rows, each with tokens, latency, and cost, because serving and the invoice report calls [9, 23]. Calls are what teams reason with.

The fourteen calls did not arrive independently. A plan produced them: it decided how many there would be, which verified another's output, and which were thrown away; one request expanding into generator, verifier, retry, and judge calls is the general shape of agentic execution [19, 30, 35]. Two units diverge. The unit of work, what a user requests and waits on, is the plan; the unit of attribution, what cost and reliability get charged to, is the call. Accounting that reads the first through the second cannot say which plan, or which stage of which plan, was expensive or unreliable. We call the mismatch the *single-call fallacy*.

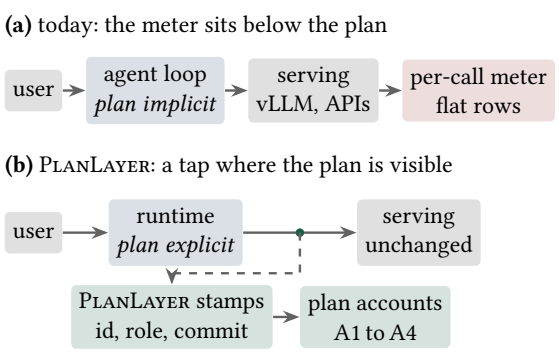

Figure 1. The fix is a tap, not a rebuild.

Is this not just missing metadata? Three routes look like fixes and are not. Serving systems consume plan shape [12, 16, 18] but schedule with it and never record it; the meter below still sees flat rows. The OpenTelemetry GenAI registry names operations such as `invoke_agent` and `execute_tool`,[1] yet no attribute identifies a call's plan, no role separates verification or retry from generation, and nothing records whether a step survived into the final answer; an open RFC on agent lifecycle conventions stops short of all three.[2] Waste studies measure the loss from outside, 21 to 36% of agent tokens [14, 25, 31], without giving the meter a field that sees it in place. The missing piece is the record itself.

PlanLayer is that record: the smallest change that makes the plan visible to accounting. The runtime stamps three fields on every call it issues: a plan identifier; a stage role (generate, tool, retrieve, verify, judge, retry); and a commit flag set at plan close marking survival into the final answer. Each field is something the runtime already knows: it minted the plan, it typed the step, and at close it holds the answer's lineage. Nothing else moves: serving stays as it is, the identifier rides where tracing context rides, and four accounts follow by aggregation (§3): totals (A1), productive versus reliability (A2), fan-out with its cause (A3), and version mix (A4). Figure 1 frames the change: the meter moves up one layer, the plumbing stays.

[1] https://opentelemetry.io/docs/specs/semconv/registry/attributes/gen-ai/, attribute registry, stability *Development*, August 2026.
[2] https://github.com/open-telemetry/semantic-conventions-genai/issues/35 and https://github.com/traceloop/openllmetry/issues/3460.

We evaluate PlanLayer on a 1,000-task simulated workload calibrated to AgencyBench and to a production survey of deployed agents [11, 19]. Per-call accounting reports as productive 12.5% of compute the agent later threw away, 19.5% for software-engineering agents. Cost-ranked call audits reach 21% of total spend; the same effort at plan level reaches 50%. And the record is a control: a per-plan retry budget of two keeps 98% of task success at 54% of the abandoned work.

We make four contributions:

- We name the single-call fallacy and place the plan against call, trace, and task granularities (§2).
- We specify PlanLayer: the plan object with append-only revision, a precise commit rule, the three-field record, and the four accounts (§3).
- We quantify what the per-call view misses, what the plan view recovers, and why removing any field blinds one account (§4).
- We release the workload generator, the accounts library, and every script behind §4, so the accounting claims here can be replayed rather than trusted: https://github.com/hanabhp/PlanLayer.

Broader impact and scope. A correct unit of accounting sits upstream of every policy: retry budgets, plan-aware billing, and carbon attribution inherit what the meter can see, and today it sees calls. Visibility cuts both ways, and a crude hand on the new knob trades success for cost in ways a record cannot adjudicate, so PlanLayer is deliberately policy-free: it reports lineage and leaves the decision with the deployment. The same fields would let carbon accounts charge reasoning and rollback to the plans that pay off [22].

2 The Single-Call Fallacy

Per-call accounting attributes cost, latency, and reliability to individual model invocations. Invoices report it, dashboards aggregate it, and energy and carbon for the same spend are profiled with it [6, 13, 22]. The key tension: the view is exactly right for chat, where call and unit of work coincide, and quietly wrong for agents. What breaks is fourfold. Fan-out is set by the plan, so two requests with identical intent differ by an order of magnitude in calls [1] and the per-call view loses the cause. Verification, retry, and judging exist to make the answer correct [2, 19], yet a cost-ranked view reads a retry as a duplicate to suppress. A spike has no owner without a propagated identifier; it lands on whatever calls happened to be running. And the user waits on the plan's completion time, a quantity no per-call latency captures [19]. Table 1 places the granularities side by side: traces (spans without semantics) and tasks (outcomes without shape) each recover part of the picture, and only the plan carries roles and commit state, the two attributes the four failures share.

Table 1. Accounting granularities compared.

Can it attribute…	Call	Trace	Task	Plan
calls to their workflow	×	✓	✓	✓
spend to stage roles	×	×	×	✓
discarded (uncommitted) work	×	×	×	✓
latency the user waits on	×	part.	✓	✓
cost to a planning decision	×	×	×	✓

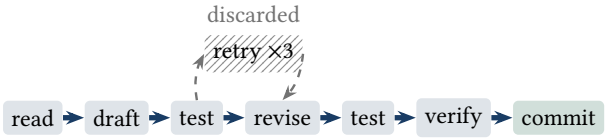

Figure 2. The plan for the ticket of §1; thick edges committed, hatched uncommitted.

3 PlanLayer

The plan object. A plan is a directed acyclic graph of calls, produced and revealed incrementally by the runtime [26, 27, 35]. Nodes carry a role; edges carry dependency or fallback. Real agents loop, revise, and abandon branches mid-flight, so the object is append-only: a revision adds nodes and edges under the same identifier, and a loop unrolls into repeated role-typed steps. Because accounts are computed at close, a plan that changed shape five times yields the same four accounts as one that did not; superseded prefixes simply close uncommitted, and per-call rows never change. The rollbacks in §4 are exactly such mid-flight revisions, so every headline number is measured over plans modified during execution.

The commit rule. The commit flag needs a rule, not a feeling. At close, the runtime marks committed every call on a data path into the final answer, every retrieval or tool call whose output such a call consumed, and every verify or judge call whose verdict admitted a committed step; everything else closes uncommitted. In Figure 2, the revision, its passing test, and the verifier are committed, and so are the first draft and failing test, because the revision consumed them; the three retries are not. Uncommitted is lineage, not blame: a discarded attempt may be why the committed one exists, and §4 prices that exchange.

The record. PlanLayer stamps three fields on every call: the plan identifier, the stage role, and, at plan close, the commit flag. Four accounts follow. A1: per-plan cost, completion time, and outcome, split committed versus abandoned. A2: generator spend against verification, retry, and judging, the bottleneck recent work identifies for useful agents [20]. A3: calls per plan, so a spike is charged to the planning decision that produced it. A4: which model served each stage, so cross-version retries stop smearing into an average. Three fields are stamped; the version in A4 is not a fourth, because the model string arrives with every serving response and A4 is a join at close.

Placement. The record is mechanism; what to do with it is policy. Following the exokernel split [5], the three fields

belong in the durable layer of the runtime, and the decisions built on them stay outside it, a separation just argued for agent interfaces as well [28]. Stamping needs no new infrastructure: one identifier and two enumerations per call plus a join at close, with roles taken from the runtime's own step types, so no model output is parsed. The carrier exists too: the identifier and role travel as span attributes beside today's `gen_ai` fields, while the commit flag cannot: spans are immutable once ended and commit is known only at close, so that field lives in the runtime. Multi-agent plans need no special case: the identifier crosses with a delegated request, and the delegating step becomes a tool node whose subtree is the callee's plan, and the failure modes that motivate verification in these systems are exactly the ones a role-labeled account localizes [2]. The accounts feed directly into systems that already assume plan shape exists but stop short of recording it, among them precedence-aware schedulers [10], agentic exploration [32], and counterfactual performance models [3].

Deployment path. PlanLayer does not require schema agreement before it is useful. A framework can stamp the fields for its own plans and gain A1 to A4 locally; a gateway can stamp the identifier and infer coarse roles from its own routing, recovering A1 and A3 fleet-wide while A2 waits on cooperation. The commit flag alone needs the runtime, because only the runtime knows which steps survived.

4 Evaluation

Methodology. No production trace with per-call plan membership is public, so we evaluate on a generated workload calibrated to published measurements. Plan lengths follow the AgencyBench mixture [11]: 70% short plans (median 6 calls), 25% research plans (median 33), 5% long autonomous plans (median 120), matching the survey where most deployments run few steps and a minority run for hours [19]. Plans revise mid-flight with 25% probability, calibrated to reports of 21 to 36% reducible agent work [14, 25, 31]; a revision discards 30 to 70% of executed steps and appends a fresh branch under the same identifier, exercising §3's append-only semantics. The result: 1,000 tasks, 1,246 plans, 21,587 calls over a simulated month. All numbers are properties of this simulation.

RQ1: How much compute is invisible per call? Of the compute per-call accounting reports as productive, 12.5% sits in steps the agent later abandoned; no per-call field marks them. Figure 3(a) shows why the blind spot concentrates: calls per plan span two orders of magnitude, and the 17.5% of plans with 33 or more calls carry 66% of traffic. Figure 3(b) breaks the share out by agent type: chat plans are too short to revise and lose 1.8%; research and tool-use agents lose 11.6% and 12.7%; software-engineering agents lose 19.5%. The high-value frontier of deployment is where the per-call view is most wrong.

RQ2: Does per-call ranking find the expensive plans? No. Plan totals are set mainly by fan-out; per-call cost varies far less, and the rank correlation between a call's cost and its plan's total is 0.07. Figure 3(c) shows the consequence: the top 10% of calls by cost reaches 21% of spend while the top 10% of plans reaches 50%. The expensive calls are large-context chat; the expensive plans are long chains of cheap calls.

Table 2. Drop one field; which accounts go blind.

	removed field			
Account	id	role	commit	ver.[†]
A1 totals with provenance	×	✓	part.	✓
A2 productive vs. reliability	×	×	✓	✓
A3 fan-out with cause	×	✓	✓	✓
A4 version mix per stage	×	✓	✓	×

[†]Not stamped: the model string serving returns, joined at close. "part.": totals survive without the committed split.

RQ3: What does the record enable? Once the commit flag exists, discarded work becomes controllable. Figure 3(d) sweeps a per-plan retry budget enforced from the recorded lineage. Unlimited retries give the 12.5% baseline at 71.4% success; a budget of two keeps 98% of that success (70.1%) at 54% of the waste (6.8%); a budget of zero saves most and costs most, 1.8% waste at 62.3% success. Here is the surprising bit: budgets above zero buy success, not just spend. The 9.1 points of success between β=0 and β=∞ are success that discarded attempts purchased, priced at 10.7 points of abandoned share; the flag turns that exchange rate from folklore into a measurable property. The account also guards the guard: at a 10% verifier false-positive rate, the reliability share A2 reports inflates from 23% to 31%, a drift only a role-labeled account can flag.

RQ4: Is every field necessary? Table 2 removes one field at a time. Each removal blinds a distinct account: without the identifier nothing aggregates; without roles reliability spend disappears into generation; without the commit flag abandoned work returns to invisibility inside A1; without the version join cross-version retries smear into an average. Nothing in the record is redundant.

5 Discussion and Threats

The 12.5% headline moves with the calibration. Sweeping the revision rate and abandoned-prefix length jointly gives 0.9% at the most conservative corner (5% rate, short prefixes), 10.7% at the survey-aligned point, and 24.0% at the worst (40% rate, long prefixes). Production-hardened deployments sit low in the range. What does not move is visibility: at every operating point the abandoned work is invisible to a unit with no field for whether a step survived. MAST-style production traces [2] with plan membership would pin the share without changing which unit can see it.

Roles are only as honest as the runtime that assigns them; we first tried inferring roles from model outputs, and it failed because frameworks phrase retries as fresh generations, so

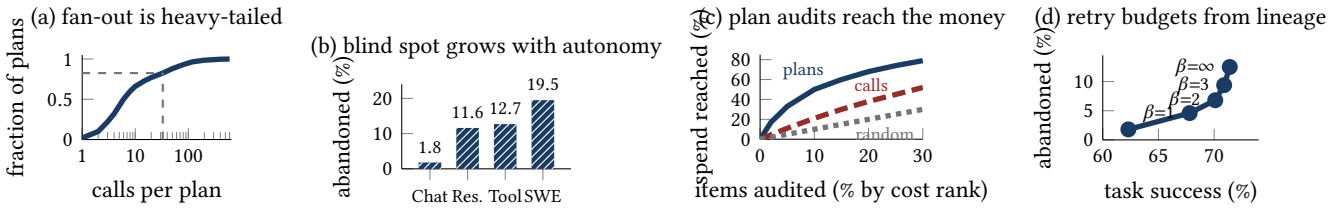

Figure 3. The fallacy, quantified (1,246 plans; 21,587 calls).

the enumeration moved into the trace format. And the layer is not free, though its cost (§3) is small against auditing the wrong plans.

Interaction with serving optimizations. Prefix caching makes the per-call number less meaningful still: calls with identical token counts differ several-fold in served cost with cache hits, and the discount belongs to the plan, whose steps share prefixes, not to any call [9, 23]; plan totals absorb the sharing because it happens inside the unit.

Billing, SLOs, and incentives. What would it mean to bill by plan? Every provider meters per call and per token; usage exports group by project, key, workspace, user, or model, and none exposes a workflow unit,[3] so a plan-level bill is assembled client-side, which the identifier makes mechanical. For the user, A1 binds a service objective: the completion time and outcome of what they waited on. For whoever runs the agent, the accounts reprice model selection: a cheaper planner that doubles fan-out shows up as a more expensive plan, and cross-version retries stop hiding in averages. The key tension is incentive: A2 protects the user, and the operator stamps it, so a runtime that mislabels retries flatters its own split. The role enumeration therefore belongs in a shared trace format, and white-box benchmarks emitting the three fields [29] would let accounting claims be replayed rather than trusted.

6 Related Work

Serving already consumes plan shape. Parrot exposes application shape to the scheduler since request-level APIs strip it [12]; Autellix schedules agent programs as wholes [16]; Cortex pools resources by workflow [18]; precedence-aware schedulers exploit the dependency graph [10]. Each assumes the shape is recorded somewhere; PlanLayer is the somewhere.

Agent efficiency measures the waste. Token studies find 59% of tokens in review loops [25], 21 to 36% reducible by trajectory reduction [31], and large savings under runtime supervision [14]; coding-agent cost is predictable from plan shape [1], and the carbon of agentic test-time compute has been profiled per step [22], a tax the commit flag lets operators charge to the plans that pay off. These treat waste as an agent-side target; we treat its invisibility as an accounting-side defect.

Observability and control. AgentSight traces agents at kernel and network boundaries [37], unified interfaces propose a shared observability spine [7, 13], AgentCgroup finds the same mismatch one layer down [36], and agentic performance management needs the counterfactual inputs A1 to A4 provide [3].

Semantic conventions and platforms. The OpenTelemetry GenAI conventions standardize per-call span attributes (operation names, token usage, agent and conversation identifiers, a workflow name); proposals add lifecycle spans (§1). LangSmith, Langfuse, Arize Phoenix, Traceloop, Helicone, Weave, and Datadog reconstruct the trace tree and cost it. None carries a durable plan identifier, a reliability role, or commit state, and none can carry the last as a span attribute, since spans are immutable once ended while commit is known only at plan close. The conventions are the carrier our fields extend, not a substitute.

Characterization. Production surveys establish the verify-retry-judge shape [4, 19, 20]; MAST taxonomizes the failures verification catches [2]; benchmarks supply the workload shape we calibrate to [8, 11, 15, 17, 34]; agent designs span tool use [24, 26, 33] and multi-agent societies [21]. None can be charged for what it discards until the discarding is recorded.

7 Conclusion

Return to the ticket. Fourteen rows sat on the dashboard; three fields turn them back into one plan with a committed spine and a hatched branch that finally has a price. PlanLayer moves the tap, not the plumbing: stamp the fields where the plan is visible, derive the accounts, and cost becomes auditable, attributable, and controllable. A public trace format carrying the three fields can replace the 12.5% we simulate with deployment numbers. Everything follows from three fields the runtime already knows.

References

[1] Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei. 2026. How Do Coding Agents Spend Your Money? Analyzing and Predicting Token Consumptions in Agentic Coding Tasks. OpenReview, ICLR 2026 submission. https://openreview.net/forum?id=1bUeVB3fov

[3] OpenAI groups usage by project, key, user, and model; Anthropic by workspace, key, and model; Google Vertex accepts per-request billing labels. None defines a plan unit.

[2] 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. https://doi.org/10.48550/arXiv.2503.13657

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

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

[5] 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. https://doi.org/10.1145/224056.224076

[6] Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. 2023. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon '23)*. Association for Computing Machinery. https://doi.org/10.1145/3604930.3605716

[7] Yanpeng Hu and Yusheng Zheng. 2025. Unified Agentic Interfaces is All You Need for AI Agent Observability. In *Proceedings of the 1st Workshop on Systems for Agentic AI (SAA '25), co-located with SOSP '25*. Seoul, Republic of Korea.

[8] 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)*. https://openreview.net/forum?id=VTF8yNQM66

[9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*. Association for Computing Machinery, 611–626. https://doi.org/10.1145/3600006.3613165

[10] Adam Lechowicz, Rohan Shenoy, Noman Bashir, Mohammad Hajiesmaili, Adam Wierman, and Christina Delimitrou. 2025. Carbon- and Precedence-Aware Scheduling for Data Processing Clusters. In *Proceedings of the ACM SIGCOMM 2025 Conference (SIGCOMM '25)*. Association for Computing Machinery, 1241–1244. https://doi.org/10.1145/3718958.3750478

[11] Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, Weiye Si, Wenjie Li, Dequan Wang, and Pengfei Liu. 2026. AgencyBench: Benchmarking the Frontiers of Autonomous Agents in 1M-Token Real-World Contexts. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 7422–7440. https://doi.org/10.18653/v1/2026.acl-long.337

[12] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. 2024. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI '24)*. USENIX Association, 929–945.

[13] Changyuan Lin and Mohammad Shahrad. 2024. Bridging the Sustainability Gap in Serverless through Observability and Carbon-Aware Pricing. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems (HotCarbon '24)*. Association for Computing Machinery. https://doi.org/10.1145/3679240.3681000

[14] Fulin Lin, Shaowen Chen, Ruishan Fang, Hongwei Wang, and Tao Lin. 2026. Stop Wasting Your Tokens: Towards Efficient Runtime Multi-Agent Systems. In *Proceedings of the 14th International Conference on Learning Representations (ICLR '26)*. https://doi.org/10.48550/arXiv.2510.26585

[15] 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. 2024. AgentBench: Evaluating LLMs as Agents. In *Proceedings of the 12th International Conference on Learning Representations (ICLR '24)*. https://openreview.net/forum?id=zAdUB0aCTQ

[16] Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. 2025. Autellix: An Efficient Serving Engine for LLM Agents as General Programs. arXiv preprint arXiv:2502.13965. https://doi.org/10.48550/arXiv.2502.13965

[17] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A Benchmark for General AI Assistants. In *Proceedings of the 12th International Conference on Learning Representations (ICLR '24)*. https://openreview.net/forum?id=fibxvahvs3

[18] Nikos Pagonas, Yeounoh Chung, Kostis Kaffes, and Arvind Krishnamurthy. 2025. Cortex: Workflow-Aware Resource Pooling and Scheduling for Agentic Serving. In *Proceedings of the 1st Workshop on Systems for Agentic AI (SAA '25), co-located with SOSP '25*. Seoul, Republic of Korea. https://doi.org/10.48550/arXiv.2510.14126

[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. https://doi.org/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] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery. https://doi.org/10.1145/3586183.3606763

[22] 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). HotCarbon '26.

[23] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Inigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st ACM/IEEE Annual International Symposium on Computer Architecture (ISCA '24)*. IEEE, 118–132. https://doi.org/10.1109/ISCA59077.2024.00019

[24] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS '24)*.

[25] Mohamad Salim, Jasmine Latendresse, SayedHassan Khatoonabadi, and Emad Shihab. 2026. Tokenomics: Quantifying Where Tokens Are Used in Agentic Software Engineering. In *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR '26)*. https://doi.org/10.48550/arXiv.2601.14470

[26] 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 *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS '23)*.

[27] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS '23)*.

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

[29] Zirui Wang, Guangba Yu, and Michael R. Lyu. 2026. AI-NativeBench: An Open-Source White-Box Agentic Benchmark Suite for AI-Native Systems. arXiv preprint arXiv:2601.09393. https://doi.org/10.48550/arXiv.2601.09393

[30] 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. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *Proceedings of the 1st Conference on Language Modeling (COLM '24)*.

[31] Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. 2026. Reducing Cost of LLM Agents with Trajectory Reduction. *Proceedings of the ACM on Software Engineering* 3, FSE, Article FSE056 (2026). https://doi.org/10.1145/3797084

[32] Jiakai Xu, Tianle Zhou, Eugene Wu, and Kostis Kaffes. 2025. Toward Systems Foundations for Agentic Exploration. In *Proceedings of the 1st Workshop on Systems for Agentic AI (SAA '25), co-located with SOSP '25*. Seoul, Republic of Korea.

[33] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS '24)*. https://openreview.net/forum?id=mXpq6ut8J3

[34] 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)*. https://openreview.net/forum?id=roNSXZpUDN

[35] 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)*.

[36] 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)*.

[37] 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, 110–115. https://doi.org/10.1145/3766882.3767169