A Failure-Mode Taxonomy for LLM-Agent Services: An Empirical Mining Study

Phat T. Tran-Truong[1] ✉⋆, Xuan-Bach Le[1] ⋆, and Xuan Son Ha[2]

[1] Ho Chi Minh City University of Technology (HCMUT), VNU-HCM, Vietnam
{phatttt,lexuanbach}@hcmut.edu.vn
[2] RMIT University, Vietnam
ha.son@rmit.edu.vn

Abstract. LLM-agent frameworks increasingly run as service components that call remote model providers and tool servers, yet service-oriented computing (SOC) lacks an empirically grounded vocabulary for how they fail. We present FMTAX, a taxonomy of 22 failure classes in seven groups, built by coding 720 closed GitHub issues from six repositories of four ecosystems (LangChain, LlamaIndex, MCP, CrewAI) and checked for coverage on 240 held-out issues from four further ecosystems. Labels from two rule coders are calibrated against a human-coded anchor (κ=0.70) and an independent LLM annotator. Up to 40% of the issues are agent-specific; both references indicate that the rule coders over- rather than under-assign such codes. Across all raters, 66–82% of agent-specific failures arise at the agent's service boundaries—the tool contract and the provider and tool-server links—led by tool-argument malformation and provider-error propagation. Ecosystems differ in their agent-specific share and in two classes that survive a like-for-like sampling check. Weighting a mapping from failure classes to SOC resilience patterns by observed prevalence, three boundary patterns (schema validation, error normalisation, timeout with retry) are candidate mitigations for 70% of agent-specific issues—a testable hypothesis for framework design. Corpus, codebook, and pipeline are released.

Keywords: LLM agents · Service-oriented computing · Failure-mode taxonomy · Mining software repositories · Resilience patterns.

1 Introduction

LLM-agent frameworks such as LangChain, LlamaIndex, CrewAI, and servers built on the Model Context Protocol (MCP) have moved within two years from research prototypes [13, 17] to service backbones. An agent is then a composed, request-serving component: it calls remote model providers, invokes tools over network protocols, keeps state, and must meet availability and latency expectations, so its failures are service incidents. Yet how these systems fail in the field

⋆ P. T. Tran-Truong and X.-B. Le contributed equally to this work.
✉ Corresponding author: Phat T. Tran-Truong, phatttt@hcmut.edu.vn.

is documented only in ad-hoc language. Task benchmarks measure what agents fail to do on designed tasks [6, 7, 9, 16], and trace-based taxonomies explain why controlled multi-agent runs break [1]; neither observes the failures practitioners report, nor connects failures to the mitigation vocabulary of SOC. We address this gap with a mining study of framework issue trackers and ask:

RQ1 Which failures are reported against LLM-agent frameworks, how often, and where along the agent's service interactions do they arise?

RQ2 Do failure-class prevalences differ across ecosystems?

RQ3 Which SOC resilience patterns are candidate mitigations for each agent-specific class, and how is the observed failure mass distributed over them?

We contribute (i) **FMTAX**, a 22-class taxonomy with inclusion rules, located on the agent's service interactions; (ii) a labelled 720-issue corpus and a 240-issue held-out corpus with reliability evidence from rule coders, a human anchor, and an independent LLM annotator; (iii) evidence that agent-specific failures concentrate at service boundaries and that ecosystem differences persist under a like-for-like sampling check; and (iv) a prevalence-weighted mapping to SOC resilience patterns. FMTAX is a first, field-grounded version (v1); our statements concern what users *report*, not production failure rates.

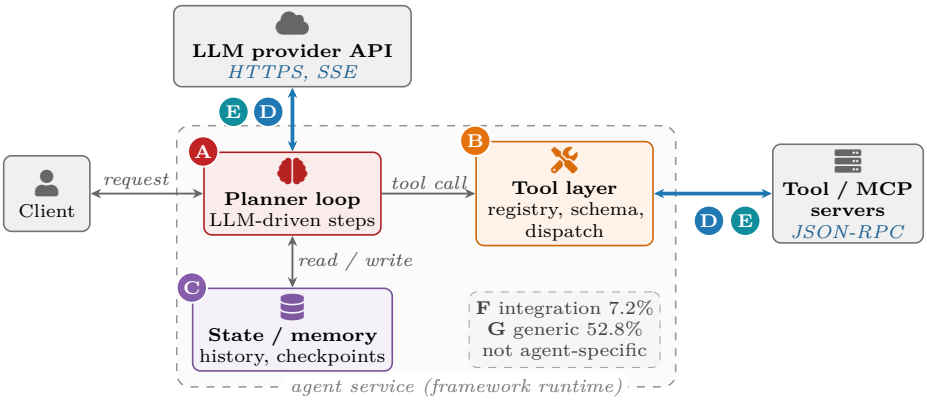
2 LLM-Agent Services and Related Work

2.1 LLM-Agent Services

An LLM-agent framework turns a model into a request-serving service (Fig. 1). A request enters a *planner loop* that repeatedly calls a remote *LLM provider* over HTTPS, often streamed via server-sent events. When the model emits a tool call, the *tool layer* validates its arguments against the tool's schema and dispatches it, increasingly through an MCP client that discovers and invokes remote *tool servers* over JSON-RPC [10]; results flow into conversation *state* for the next step. Each hop is a service interaction with a contract, a transport, and failure semantics, and our corpus shows each hop failing: a provider rejects a structured-output request because the framework serialises `tools: []` (`langchain#34907`, argument malformation); an embeddings response of unexpected shape surfaces as an opaque error (`langchain#35204`, provider-error propagation); an MCP SDK silently drops a tool's input schema at registration, so clients see an empty contract (`typescript-sdk#1643`, discovery/registration); an MCP server deadlocks on an oversized JSON-RPC request id (`typescript-sdk#1765`, race/async); and a chat assistant repeats its previous answer (`crewAI#3819`, state leakage).

2.2 Related Work

Empirical bug studies. Coded-sample bug studies are a long-standing MSR genre, e.g., for deep-learning systems [4, 5, 18] and concurrency [8]. We follow their template (codebook, calibration, exemplars) for a new population: agent frameworks at the service boundary.



A control 2.2%: iteration timeout, circuit breaker; B tool 15.6%: schema-validation gateway, tool registry; C state 1.2%: bulkhead isolation, bounded buffer; D service 17.4%: timeout + retry, error normalisation, credential broker; E reliability 3.6%: back-pressure, idempotency key.

Fig. 1. One request through an LLM-agent service. Tags mark where each agent-specific FMTAX group arises; thick blue links cross the agent's service boundary. The key lists each group's share of the 720-issue corpus and its candidate SOC resilience patterns (per class: Table 2).

LLM-agent reliability. Agent research centres on planning, tool use, and multi-agent coordination [15], assessed through task benchmarks. Closest to our intent, MAST [1] derives a failure taxonomy from traces of controlled multi-agent runs. FMTAX differs in its evidence (field issues, which include provider, transport, and packaging failures that designed runs exclude), its service framing (failures located on service interactions and mapped to SOC patterns), and its cross-ecosystem prevalence. The views are complementary: MAST's inter-agent categories align with our control and state classes, which are rare in our data.

SOC resilience patterns. The SOC and microservice literature catalogues resilience patterns such as timeouts, circuit breakers, bulkheads, sagas, idempotency keys, dead-letter queues, transactional outboxes, API-gateway request validation, service registries, and back-pressure [11, 12]; empirical work on microservices observes failures at service boundaries, from trace-based fault localisation to resilience testing by fault injection [2, 19]. We use this catalogue as the target vocabulary of RQ3—as candidate mitigations, not validated remedies.

3 Study Design

3.1 Repositories and Sampling

We mined six GitHub repositories spanning four ecosystems and distinct service roles (Table 1), chosen as production-grade, widely used, and actively maintained; AutoGen, whose issue process and labels changed during our window, serves in the

Table 1. Construction corpus: 120 most recent closed issues per repository (GitHub, snapshot 2026-04-26).

Repository	Ecosystem	Service role	Query filter
`langchain-ai/langchain`	LangChain	agent framework	`label:bug`
`langchain-ai/langgraph`	LangChain	graph orchestration	`label:bug`
`run-llama/llama_index`	LlamaIndex	RAG / data agents	`label:bug`
`modelcontextprotocol/python-sdk`	MCP	tool protocol (Py)	none
`modelcontextprotocol/typescript-sdk`	MCP	tool protocol (TS)	none
`crewAIInc/crewAI`	CrewAI	multi-agent teams	`label:bug`

held-out check instead. We took the 120 most recent closed issues per repository (720 in total; snapshot 2026-04-26), a balanced frame that keeps any one project from dominating comparisons at the cost of population-proportional prevalences. For each issue we parse title, labels, timestamps, and the first 3,500 characters of the body from cached HTML, the unit of reproducibility. Inclusion criteria differ: four repositories were queried with `is:issue is:closed label:bug`, but the two MCP SDKs without the label filter, so their sample also contains feature requests and questions; 82 of the 240 MCP issues carry the `bug` label. Section 4.3 therefore repeats RQ2 on a like-for-like frame.

3.2 Taxonomy Construction

FMTAX was built in four steps of open and axial coding [14]. (1) *Seed:* candidate classes were derived deductively from fault taxonomies of deep-learning systems [4, 5], concurrency-bug studies [8], and the failure modes that SOC stability patterns address, such as timeouts, cascading failures, unsafe retries, and overload [11, 12]. (2) *Open coding:* issues of the construction sample were read and coded against the seed; an issue that fitted no candidate opened a new one, and candidates that could not be stated as an inclusion rule with an observable textual cue were merged into a neighbour or dropped. (3) *Axial grouping:* classes were grouped by the locus of the failure on the agent's service interactions (Fig. 1): planner loop (`A`, control), tool layer and contract (`B`, tool), memory (`C`, state), provider and server links (`D`, service), and concurrent calls and retries (`E`, reliability), plus integration (`F`) and generic (`G`) groups for causes that are not agent-specific. (4) *Operationalisation and freeze:* each class received an inclusion rule, an exclusion rule, an exemplar, and a place in the tie-break order `B`$\succ$`C`$\succ$`D`$\succ$`A`$\succ$`E`$\succ$`F`$\succ$`G`. Codebook v1 (Table 2; full rules in the supplement, Sect. A) was frozen with the 2026-04-26 snapshot and has not changed since; all coders, the anchor, and the held-out check apply it unchanged. Four seed classes (`B3`, `C3`, `E1`, `E2`) were kept although no issue was coded to them, as documented failure modes (Sect. 5).

3.3 Labelling and Reliability

Two deterministic rule coders apply codebook v1 under opposite decision orders: Coder-A consults repository labels first, then title/body keyword rules; Coder-B

applies keyword rules first, then labels. Each issue receives one primary code. As both share keyword tables, their agreement measures the robustness of the rules, not human reliability, so we calibrate them against two references. (i) A *human anchor* of 30 issues hand-coded under v1 by one author (the lead investigator), stratified 10 per ecosystem under the three-way grouping used when it was drawn, which pooled LlamaIndex and MCP; in the four-ecosystem grouping it holds 10 LangChain, 4 LlamaIndex, 6 MCP, and 10 CrewAI issues. (ii) An *independent LLM annotator* (Claude, disclosed as a model, not a human) coded a blind stratified sample of 90 issues (28/16/24/22 per ecosystem, including the anchor) from title and body alone, given the codebook but no rule or anchor labels; it returned labels for 89. We report Cohen's κ with bootstrap 95% CIs.

3.4 Validation and Analysis

For held-out coverage we mined 240 closed issues (60 each) from AutoGen, Semantic Kernel, Haystack, and DSPy (snapshot 2026-07-19) and labelled them with the unchanged coders. Semantic Kernel and DSPy were queried with the bug filter, AutoGen and Haystack without it (fewer than 60 closed `bug`-labelled issues were retrievable), and only 11 DSPy bodies could be fetched, so 49 issues without text are coded `G3`. We also truncate each construction repository to 40–120 issues. Prevalences carry bootstrap 95% CIs (B=5000). For RQ2 we run pooled two-proportion z-tests for every code and ecosystem pair (132 tests) with Holm correction [3], and a χ^2 test of the agent-specific share; resolution times are compared with Kruskal–Wallis and Mann–Whitney tests.

4 Results

4.1 Reliability

The rule variants agree at κ=0.897 (code) and 0.877 (group), an upper bound on consistency. Against the human anchor both reach κ=0.700 (95% CI $[0.51, 0.85]$) while agreeing perfectly with each other, i.e., they share blind spots. The LLM annotator agrees moderately with the anchor (κ=0.54, $[0.34, 0.73]$) and weakly with the rules (κ=0.32, $[0.21, 0.43]$; 70% agreement on whether an issue is agent-specific). Disagreements have a direction: the rules mark 18 of the 30 anchor issues as agent-specific against 15 for the human (recall 1.00, precision 0.83), and 42 of the 89 blind issues against 29 for the LLM annotator. The agent-specific share below is thus an upper estimate; scaled by the anchor's precision it is about one third.

4.2 RQ1: Prevalence and Locus of Failures

Generic software bugs (`G1`) are the largest class (38.3%, CI $[34.7, 41.9]$; Table 2), followed by argument malformation (`B2`, 13.1%, $[10.6, 15.6]$), feature requests and questions (`G2`, 12.6%; 85 of 91 from MCP, Sect. 4.3), and provider-error

Table 2. FMTAX codebook v1: prevalence (% of 720 issues, Coder-A) and the proximal candidate SOC pattern of each agent-specific class (groups `A`–`E`). Full rules, exemplars, and the complete mapping with rationale: supplement.

Code	Class	Inclusion rule (abridged)	%	Proximal SOC pattern
A1	Non-terminating planning	loops; iteration limit hit	1.4	max-iteration timeout
A2	Premature termination	stops before requested output	0.8	saga / compensation
B1	Tool mis-selection	wrong tool, or none when needed	0.6	typed tool registry
B2	Argument malformation	arguments fail schema checks	13.1	schema-validation gateway
B3	Output mishandling	tool result misread/fabricated	0.0	output contract validator
B4	Discovery/registration	tool or server not discoverable	1.9	registry + health probe
C1	Context overflow	context or memory exhausted	0.1	bounded buffer
C2	State leakage	state leaks across sessions	1.1	bulkhead isolation
C3	Hallucinated state	acts on non-existent state	0.0	state probe before acting
D1	Auth/credentials	missing/expired key or scope	1.9	credential broker
D2	Rate limit/quota	429 or quota hit, no back-off	0.3	token bucket
D3	Network/transport	timeout, stalled stream, abort	5.4	timeout + retry/backoff
D4	Provider-error propagation	provider error misreported/lost	9.7	error normalisation
E1	Unsafe retry	retry repeats a side effect	0.0	idempotency key
E2	Cascading tool errors	tool error corrupts later calls	0.0	bulkhead; circuit breaker
E3	Race/async ordering	race in async calls or streams	3.6	back-pressure
F1	Dependency/version	version change breaks code	4.2	–
F2	Build/install/config	install or configuration fails	2.9	–
F3	Doc/example mismatch	documented usage fails	0.1	–
G1	Generic software bug	cause incidental to agent logic	38.3	–
G2	Feature request/question	not a failure report	12.6	–
G3	Uncodable	too little information to code	1.8	–

propagation (`D4`, 9.7%, [7.6, 11.9]). The agent-specific groups `A`–`E` account for 288 issues (40.0%).

Where they arise is the clearest result (Fig. 1): 82.3% of agent-specific issues occur at the agent's service boundaries—the tool contract (`B`, 15.6% of the corpus) and the provider and server links (`D`, 17.4%)—whereas control-flow (`A`, 2.2%), state (`C`, 1.2%), and concurrency (`E`, 3.6%) failures are rare. The concentration holds across raters and samples: 80.0% of the human anchor's agent-specific issues, 65.5% of the LLM annotator's, and 82.0% in the held-out corpus. Four classes (`B3`, `C3`, `E1`, `E2`) are never observed. Within `G1`, rule-identifiable sub-themes (runtime errors, configuration, serialisation, dependencies, documentation) cover 27.9%; the rest is a diffuse tail of ordinary defects, so much tracker traffic is conventional software breakage. All 18 observed classes appear within the first 76.2% of the corpus; as the coders can only emit FMTAX classes, this shows a stable class set, not a complete one.

Resolution time. Closure was part of the sampling query, so time-to-resolution (TTR) is right-censored with respect to the open backlog; we report it descriptively and derive no priorities from it. The median TTR of 6.1 days depends more on the ecosystem (2.1 d LlamaIndex, 4.5 d LangChain, 8.0 d MCP, 35.3 d CrewAI) than on the failure group (Kruskal–Wallis over the seven groups, p=0.053). `B2` (12.4 d) and `E3` (27.9 d) have long medians, but neither differs significantly from the other issues (Mann–Whitney p=0.39, p=0.10), and `G2`'s long median (23.6 d) reflects MCP feature requests.

Table 3. Ecosystem comparison (%). "MCP (`bug`)" keeps only MCP's `bug`-labelled issues, matching the other repositories' inclusion criterion.

	LangChain	LlamaIndex	MCP (all)	MCP (`bug`)	CrewAI
Issues (n)	240	120	240	82	120
Agent-specific (%)	50.4	40.8	23.3	34.1	51.7
B2 argument malformation (%)	17.5	13.3	3.8	3.7	22.5
B4 discovery / registration (%)	0.4	0.0	4.6	8.5	1.7
D4 provider-error propagation (%)	16.7	10.0	3.3	3.7	8.3
G2 feature request / question (%)	2.5	0.0	35.4	4.9	0.0

4.3 RQ2: Ecosystem Differences

Table 3 contrasts the original frame with a like-for-like frame that restricts MCP to its 82 `bug`-labelled issues. The agent-specific share differs in both ($\chi^2(3)$=45.5, $p<10^{-9}$; like-for-like $\chi^2(3)$=9.3, p=0.025): about half of LangChain and CrewAI issues are agent-specific, against 41% for LlamaIndex and 23–34% for MCP. Per-code tests show that the original frame overstates structure. Three of its six Holm-significant contrasts concern `G2` and vanish like-for-like, where MCP's `G2` share falls from 35.4% to 4.9%: the prominence of feature discussion in MCP is an artefact of its broader query. Two others (`B2`, `D4`: LangChain vs. MCP) keep their direction but lose significance. Two contrasts hold like-for-like, without `G2`, and under Fisher's exact test (supplement): argument malformation is more frequent in CrewAI than in MCP (22.5% vs. 3.7%, p_{Holm}=0.028), and discovery/registration failures are more frequent in MCP than in LangChain (8.5% vs. 0.4%, p_{Holm}=0.006), a contrast the original frame masks. Both have a service reading: MCP's distinctive failures concern tool discovery and schema registration, the protocol's registry function (e.g., `typescript-sdk#1643`); CrewAI's concern tool payloads.

4.4 RQ3: Candidate SOC Resilience Patterns

Table 2 maps each agent-specific class to the pattern that standard SOC practice would name as its proximal mitigation, in the vocabulary of Nygard and Richardson [11, 12]; a "schema-validation gateway" is API-gateway request validation applied to tool schemas. For the dominant classes the rationale is direct: a gateway rejects malformed payloads before they reach a tool (`B2`); error normalisation turns raw provider faults into typed, inspectable events, with dead-lettering for unhandled ones (`D4`); timeouts bound waits and retries with backoff absorb transient faults (`D3`). The mapping is ours and is not validated against remediation outcomes, and since every agent-specific class received a pattern, its coverage is complete by construction and carries no evidence. What the data add is how the observed failures distribute over the patterns: three boundary patterns are the candidate mitigations for 70.5% of agent-specific issues—schema validation (32.6%), error normalisation (24.3%), and timeout with retry (13.5%)—and backpressure for races adds 9.0%, whereas the patterns for `B3`, `C3`, `E1`, and `E2` match

no observed issue. This shows where a pattern would have to act; whether it prevents the failures requires intervention studies.

4.5 Held-out Coverage and Sampling Sensitivity

Every held-out issue received an FMTAX code (the 49 without text `G3`); 16 of the 22 classes recur across all seven groups, and the rule variants stay consistent (κ=0.975). Of the 191 issues with text, 26.2% are agent-specific, 82.0% of them at service boundaries. The generic share is high (68.1%) and may hide failure vocabulary the rules miss, so we read this as a coverage check, not a prevalence estimate. Truncating the construction repositories to 40–120 issues keeps the agent-specific share within 37.8–41.0% with the same dominant classes.

5 Discussion

Agent failures are service-boundary failures. The dominant agent-specific failures are contract and interaction faults where the agent exchanges structured data with other services: malformed tool payloads, mis-propagated provider errors, transport faults, and, in MCP, broken tool discovery and registration. Planning failures, which benchmark and trace studies emphasise [1, 7], are rare in field reports, and even tool *selection* (`B1`, 0.6%) is far rarer than argument malformation. This places agent reliability on familiar SOC ground—gateway validation, typed error normalisation, bounded timeouts with retry, registries with health probes—which frameworks today leave to application code. Making these mechanisms default-on at the agent–service boundary is a concrete, testable design hypothesis, not a demonstrated remedy. The four unobserved classes (output mishandling, hallucinated state, unsafe retry, cascading errors) produce wrong answers or duplicated side effects rather than exceptions and are likely under-reported in issue text; telling rarity from silence needs runtime telemetry. The released rule coder can serve as a triage labeller.

Threats to validity. Construct: one primary code per issue; the boundary between agent-specific and generic causes is a judgement; the pattern mapping is the authors' (RQ3 shows where patterns would act, not that they work). *Internal:* labels come from rule coders whose mutual agreement overstates reliability; the only human reference is a 30-issue anchor coded by one author, who is not independent of the study; the LLM annotator is independent of the rules but not human. Both references indicate that the rules over-assign agent-specific codes, and neither the class-emergence curve nor the held-out check can reveal missing classes; independent human double-coding on a larger anchor is the most important next step. *External:* six repositories, one snapshot of recent closed issues, and inclusion criteria that differ across repositories (addressed for RQ2 by the like-for-like frame); only GitHub is covered, so other forges such as GitLab, further projects, and unreported failures remain outside our scope. *Conclusion:*

sparse classes have wide CIs; per-code tests are Holm-corrected; TTR is right-censored and confounded by ecosystem. Analyses are deterministic given the cached HTML; the data are public issue text, cited by repository and number.

6 Conclusion

FMTAX is a first, field-grounded failure taxonomy for LLM-agent services. Up to 40% of reported problems are agent-specific, and two-thirds to four-fifths of these arise at the agent's service boundaries rather than in its reasoning. Ecosystem differences are real but narrower than a naive comparison suggests, and three boundary patterns are candidate mitigations for 70% of agent-specific issues. Next steps are independent human double-coding, a survival analysis including open issues, intervention studies of default-on boundary patterns, and further ecosystems and forges.

Data and Code Availability

The corpora (720 construction and 240 held-out issues, with cached HTML), codebook v1, rule coders, anchor and LLM labels, analysis scripts, and the supplement are available at `https://github.com/LeoTTTPhat/FMTAX`.

Acknowledgments. We acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Cemri, M., Pan, M.Z., Yang, S., Agrawal, L.A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J.E., Stoica, I.: Why do multi-agent LLM systems fail? arXiv preprint arXiv:2503.13657 (2025)
2. Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V.: Gremlin: Systematic resilience testing of microservices. In: Proc. 36th IEEE Int. Conf. on Distributed Computing Systems (ICDCS). pp. 57–66 (2016). `https://doi.org/10.1109/ICDCS.2016.11`
3. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics **6**(2), 65–70 (1979)
4. Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P.: Taxonomy of real faults in deep learning systems. In: Proc. 42nd Int. Conf. on Software Engineering (ICSE). pp. 1110–1121 (2020). `https://doi.org/10.1145/3377811.3380395`
5. Islam, M.J., Nguyen, G., Pan, R., Rajan, H.: A comprehensive study on deep learning bug characteristics. In: Proc. 27th ACM Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 510–520. ACM (2019). `https://doi.org/10.1145/3338906.3338955`

6. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: SWE-bench: Can language models resolve real-world GitHub issues? In: Proc. Int. Conf. on Learning Representations (ICLR) (2024), arXiv:2310.06770
7. Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., et al.: AgentBench: Evaluating LLMs as agents. In: Proc. Int. Conf. on Learning Representations (ICLR) (2024), arXiv:2308.03688
8. Lu, S., Park, S., Seo, E., Zhou, Y.: Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In: Proc. 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). pp. 329–339. ACM (2008). https://doi.org/10.1145/1346281.1346323
9. Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., Scialom, T.: GAIA: a benchmark for general AI assistants. In: Proc. Int. Conf. on Learning Representations (ICLR) (2024), arXiv:2311.12983
10. Model Context Protocol: Model context protocol specification. https://modelcontextprotocol.io/specification (2025), accessed 26 September 2026
11. Nygard, M.T.: Release It!: Design and Deploy Production-Ready Software. Pragmatic Bookshelf, 2nd edn. (2018)
12. Richardson, C.: Microservices Patterns: With Examples in Java. Manning Publications (2018)
13. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761 (2023)
14. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: A critical review and guidelines. In: Proc. 38th Int. Conf. on Software Engineering (ICSE). pp. 120–131. ACM (2016). https://doi.org/10.1145/2884781.2884833
15. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D., Wang, C.: AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155 (2023)
16. Yao, S., Shinn, N., Razavi, P., Narasimhan, K.: τ-bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv preprint arXiv:2406.12045 (2024)
17. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2022)
18. Zhang, R., Xiao, W., Zhang, H., Liu, Y., Lin, H., Yang, M.: An empirical study on program failures of deep learning jobs. In: Proc. 42nd Int. Conf. on Software Engineering (ICSE). pp. 1159–1170 (2020). https://doi.org/10.1145/3377811.3380362
19. Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q., He, C.: Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proc. 27th ACM Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 683–694 (2019). https://doi.org/10.1145/3338906.3338961