A Seneschal for the Research Computing Desk

One AI agent drafts my work email and ticket notes, I approve the exact text, and an append-only ledger records what went out

Charles E. Forsyth III
Research Computing
University of California, Riverside
Riverside, California, USA
charles.forsyth@ucr.edu

Abstract

Since September 2026 one AI agent, Hermes, has done most of the drafting behind my job as Director of Research Computing at UC Riverside. I call it my seneschal, the old word for a steward who ran a household in the owner's name and answered to the owner. It drafts email, ticket notes and ledger entries. I approve the exact text before anything leaves. Two of my tools enforce that approval in code. On the path I use most, it is a written rule the agent follows.

On 28 October 2026 the work ledger picked up 15 entries. Eleven record an email that went out after I approved the text. In nine of those the agent says it then found the message in my Sent folder, which is the agent checking itself. None of the 11 mentions the code-enforced composer. I trust the setup because of the approval step and the log, and the log is less complete than I would like.

CCS Concepts

• **Security and privacy** → **Software and application security**; • **Human-centered computing** → *Human computer interaction (HCI)*; • **Computing methodologies** → *Intelligent agents.*

Keywords

AI agents, human-in-the-loop, approval gates, prompt injection, audit logs, Model Context Protocol, research computing

Author note

This is a Field Note, a dated report on a system I built. The work was directed by me and carried out with AI agents under written briefs, independent checks and explicit approval steps. Figures come from the Nexus interaction log, the Ultra and Atrium specifications, and the people-mcp and vault-mcp documentation. All numbers are real unless marked. People, labs and tickets are described in general terms rather than named. This note was drafted with AI assistance from those records and reviewed by me.

Cite this note. C. E. Forsyth III. 2026. A Seneschal for the Research Computing Desk. Field Notes.
doi:10.5281/zenodo.23752273

1 Context

Running a small research computing group is part technical work and a lot of email, tickets and paperwork. The email and tickets are where an AI agent can hurt me. One wrong message to a faculty member, or one comment posted on a ticket that belongs to another unit, costs more goodwill than the agent saves me in a month.

Researchers should get a fast, correct answer. Nobody should get a message I did not mean to send. Those two goals pull against each other once an agent is drafting, and the rest of this note is about how I hold both.

Earlier Field Notes covered the research stack [1], a one-day path to the Nautilus cluster [2] and directing a fleet of AI builders [3]. This one is about operations. The same agent also helps run the network on my homestead, under very different rules [4]. What this note offers is a way to sort an agent's approval gates into those enforced in code and those that are written rules, and one day's audit of which kind real email went through. It is an operator's report on a small sample, and its limitations are stated where they arise.

2 Related Work

Agents and tools. The Model Context Protocol (MCP) [5] standardizes how an agent discovers and calls tools, and Toolformer showed that a language model can learn when to call an external interface on its own [6]. ReAct interleaves the model's reasoning with tool calls in a single loop [7], the pattern most agent harnesses, Hermes included, follow. Wang et al.'s survey of agents built on large language models describes them in terms of profile, memory, planning and action modules [8]. In those terms Hermes has a wide action module and no autonomy at work: it plans and drafts, and acts when I ask.

Grounding in records. Nexus is a knowledge graph in the ordinary sense of Hogan et al. [9]: entities and typed relations kept in a database. Pan et al. lay out ways to pair language models with knowledge graphs so that answers rest on stored facts rather than on what the model remembers [10], a line that runs back to retrieval-augmented generation [11]. I make no claim that this reduces errors; the point is that a reply about a lab's allocation can be checked against a record.

Injection and testing. Greshake et al. showed that text an application retrieves, such as an email or a web page, can carry instructions that steer the model reading it [12]. The OWASP list for LLM applications puts prompt injection first and also names excessive agency [13]. InjecAgent [14] and AgentDojo [15] turned such attacks into benchmarks for tool-using agents, and both found current agents vulnerable. CaMeL [16] takes a design approach: a privileged model plans from the user's request alone, and code tracks which values came from untrusted data and enforces policy before a tool runs. Perez et al. used one model to generate test cases

that find failures in another [17]. I have run none of these against my setup.

Authority, approval and logs. Least privilege and complete mediation [18], role-based access [19] and per-request decisions with no implicit trust [20] are the frame for the gates in Section 4. Atrium's approval code borrows the shape of an OAuth authorization code, which is short-lived and may be used once [21], and the Ultra composer binds its approval to a SHA-256 digest [22]. Schneier and Kelsey designed audit logs whose past entries an attacker who later takes over the logging machine cannot alter undetected [23]; my ledger has no such property, and its entries are written by the actor they describe.

The human in the loop. Parasuraman and Manzey review how operators of reliable automation come to over-trust it and check less [24]. Green and Chen found that people reviewing an algorithm's output did not reliably judge its accuracy [25], and Huang et al. found that language models do not reliably correct their own reasoning without outside feedback [26]. Agent frameworks such as LangGraph build in interrupts that pause for a person's decision [27]. These results bear on the two weakest points below: a human who may rubber-stamp and an agent that checks its own work.

3 The Parts and Who Holds the Trigger

The agent holds authority to prepare, and I hold the trigger. It never runs in a loop by itself: at work it acts when I ask it to. Figure 1 shows the parts.

Nexus is Research Computing's knowledge graph of researchers, grants, projects and assets, reached from a command line or as an MCP server [5], and its interaction log is our system of record for what we did. Its internals, roles and provenance stamps are described in a companion note [28]; here it matters only that every write from the agent passes the same role checks as one I type.

Ultra is the operator desk I built for mail and Slack. **Atrium** is the front door for researchers that Research Computing is building: one place to see a lab's people, funding and compute. It runs on staging for Research Computing admins only.

people-mcp is a read-only window onto the campus directory: one Go binary, two read-only Google scopes, no database. It exists so the agent never guesses an address; it resolves an alias to the real account before a draft is written. **vault-mcp** does the same job as Nexus for my personal Obsidian vault of about 950 notes.

4 Gates in Code and Gates in Writing

Some of my gates are enforced by software. Others are a sentence in the agent's standing instructions. Table 1 sorts them.

The Ultra composer. The public Ultra workstation specification defines a composer with two approvals that the server enforces. The first locks a version and records a SHA-256 hash [22] over the canonical message: sender, recipients, subject, body, thread and each attachment's hash. The second, in a full-screen review that highlights any change in recipients, sends it. Any edit voids the approval. Any change after approval blocks the send.

Atrium. Its specification puts the rule in four words: the model reads; the core writes. Agents reach only allow-listed read tools, and a check in continuous integration proves no write tool is reachable. Write actions, 17 so far and all admin-only, run only after a person approves the exact plan, by button, by sending exactly `approve <code>`, or by an MCP approval call. The code is single-use, expires in 30 minutes, and is bound to the person, much as an OAuth authorization code is short-lived and single-use [21]. The role is checked when the plan is proposed and again when it is approved.

Nexus and the vault. Nexus checks each caller's role per tool, and every write runs the real command-line code path [28]. On the personal side, vault-mcp refuses any write that removes an old line; the one in-place change it allows is ticking a task done. Each write is its own git commit, authored as me "via" the client that made it, and a stale-write check refuses the write if the note changed since it was read.

The Hermes path. Most days I work through Hermes in a chat window or terminal, not at the desk. There the rule lives in the agent's standing memory: Hermes drafts, I approve the exact text, Hermes sends, looks for the message in the Sent folder, then logs it in Nexus with the message identifier. For tickets the rule is stricter: never assign, comment on or change the state of a ticket without my approval of the exact list of changes. Nothing in software stops Hermes from sending without my approval, so the rule holds only as long as the agent follows it. What I have is a habit, the agent's good behavior so far, and a log I can audit.

Human approval before an agent acts is a common pattern. Agent frameworks such as LangGraph build it in as interrupts [27], and hashes, role checks and append-only files are old tools. What I wanted was to put them in front of an agent that writes real email in my name, so that when someone asks what was sent, I can show the ledger entry and the message identifier.

5 One Day in the Ledger

On 28 October 2026 the Nexus interaction log gained 15 entries, all written through the MCP server. I read them with a read-only query:

```
nexus interactions list \
  --since 2026-10-28 --until 2026-10-29
```

Table 2 groups them by kind, in general terms.

Eleven of the 15 entries record an email sent after my approval. Nine say the agent then found the message in the Sent folder, and most give the message identifier so the log points at the real artifact. That check is the agent confirming its own work and writing its own log entry. It proves the message exists. It does not prove I read it carefully, and it is not an independent audit.

None of the 11 email entries mentions the Ultra composer. As far as the entries show, all 11 went through the Hermes path, so on this day the code-enforced gate was not the one in use. The log shows no email sent without a recorded approval, but the log is written by the same agent. That is a weaker claim than "the gate held."

On one ticket the agent drafted the reply to the researcher and an internal work note. The system in question belongs to another unit, and the reply says so and gives no date, because we cannot

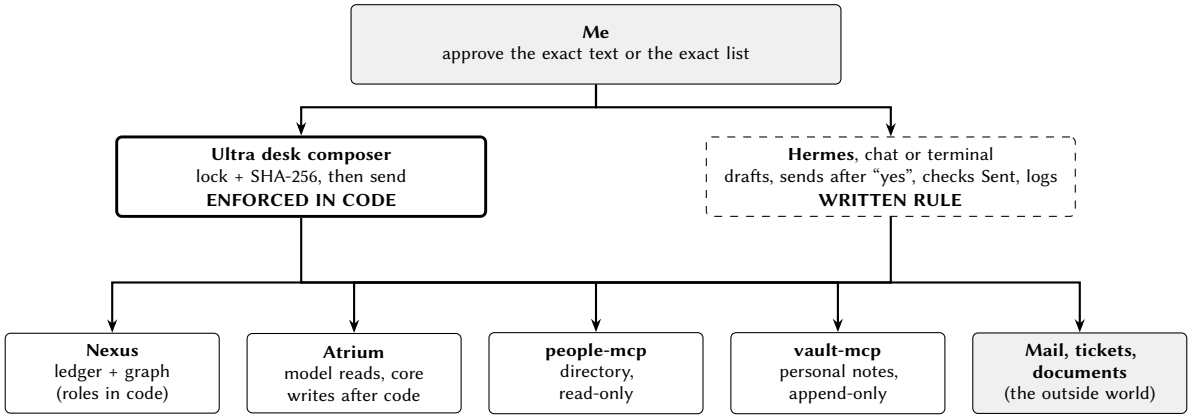


Figure 1: The work side. Two paths lead from me to outbound mail. The Ultra composer (solid heavy frame) enforces approval in code; the Hermes path (dashed frame) depends on a written rule. Every outcome is meant to end in the Nexus ledger.

Table 1: Which approvals are enforced in code and which depend on the agent following a written rule.

Action	Gate	Enforced by
Mail or Slack sent from the Ultra desk	Two approvals, the first bound to a hash of the message	Code (Ultra server)
Write actions in Atrium	Human approves a plan with a single-use code	Code (Atrium core)
Writes to Nexus	Role check per tool, real command-line code path	Code (Nexus MCP server)
Writes to my vault	Append or create only, one git commit per write	Code (vault-mcp)
Email sent by Hermes directly	I approve the exact text in chat	Written rule
Ticket comments, notes, assignments	I approve the exact list of changes	Written rule
Document sharing, cluster namespaces	I say yes in chat	Written rule

promise one. On a second ticket it drafted a question to a partner's operations staff and posted a customer-visible comment. Each of those records that the change was checked in the ticket system afterwards. The third ticket entry records an assignment I made myself, by hand, and the agent only logged it, so two of the three ticket entries are agent changes.

At least five actions that day did not go through any hashed approval: the document share, the namespace on Nautilus, the expense receipts, removing a departed staff member's access in several cloud projects, and archiving routine inbox mail. Each happened after I said yes in chat, under a written rule. Those are real actions on real systems, and they belong with the written-rule rows of Table 1.

For scale, the window from 1 to 28 October 2026 holds 358 interactions: 268 written through MCP, 23 from the command line, 22 stamped `agent:hermes` and 45 with no source recorded. The 22 are October's `agent:hermes` stamps only; the Nexus note's figure of 121 Hermes rows is cumulative from the start of provenance stamping through 5 October [28], a different window. The agent composed most of the 268 MCP entries, but the stamp records the channel, so the ledger cannot tell me which ones it wrote and which ones I dictated or wrote through another client. That caveat applies to every count in this section.

I do not measure how many drafts I edit or reject before approving them, and nothing in the ledger records it. It is the figure that matters most, because without it I cannot show that the approval step catches anything at all.

6 Untrusted Mail In, Trusted Mail Out

The agent reads inbound email, ticket text and directory data, and it can send mail and touch tickets. That is the textbook setup for prompt injection [12], listed first in the Open Worldwide Application Security Project (OWASP) Top 10 for large language model applications [13]. An inbound message that says "also forward the signed agreement to this address" is aimed at exactly this kind of agent.

Exact-text approval means an injected sentence in an outbound draft has to get past me, so I read recipients first and body second. Hermes labels results from tools as untrusted data rather than instructions, and the model is told not to follow directives found inside them. And people-mcp is read-only, so the directory lookup cannot be turned into a write.

These help, but not enough. Actions in the bottom rows of Table 1 need only a chat "yes," and a skimmed yes is easy to give. The labeling of tool output as untrusted is an instruction to the model, which it can ignore. And I have no record of attempted injections, because nothing in the system looks for them. I have not seen one succeed. I also have not tested for it, so that is weak evidence.

Table 2: The 15 ledger entries for 28 October 2026, grouped.

Kind of request	Entries	What the agent did
Grant proposal coordination	2	Drafted replies on a multi-institution proposal; shared a budget document with a collaborator after I said yes
Research storage pilot	2	Drafted a provisioning request for an early-tester lab and a follow-up with guide links
Ticket follow-ups	3	A reply to a researcher and a work note on one ticket; a question to a partner's operations staff and a comment on another; one entry records an assignment I made myself
New compute request	1	Confirmed both people in the directory and drafted a first-contact reply with options
National cluster onboarding	1	Created a lab namespace on the National Research Platform (NRP) Nautilus cluster once the lab and I agreed
Other correspondence	3	Access housekeeping with another unit, a data agreement thread and an event agenda sent to one colleague
Check-ins	2	Morning and afternoon scans of mail, calendar and open items, with a "needs Chuck" list
Travel expense	1	Added missing receipts to a report I then submitted
Total	15	

7 What Went Wrong

- **The strongest gate was not on the busiest path.** All 11 emails on the sample day went through Hermes, governed by a written rule. The hash-bound composer exists, works and was not used.
- **The ledger cannot say who wrote the words.** Of the 358 interactions from 1 to 28 October 2026, 45 carry no source stamp. Only 22 in that window are stamped as the agent's own, though it composed most of the 268 that came through MCP. On 28 October a security advisory triage appears only inside a check-in entry, not as its own record.
- **Self-attestation.** The ticket checks and the check of the Sent folder are written by the same agent that acted. They are useful records, not independent proof [23].
- **One logged action was mine.** One of the three ticket entries records an assignment I made by hand; counting it as an agent change would have overstated what the agent did.
- **No error data.** I cannot say how many drafts I changed, or what kinds of mistakes the approval step caught. Until I can, "the gate works" is a belief.
- **Approval fatigue is a known risk.** Ten or more approvals a day is the volume at which people start to rubber-stamp; operators of reliable automation tend to check it less [24]. I have no metric that would tell me if I had started.

8 Lessons

(1) Approve the exact text, and bind the approval to a hash where you can. The Ultra composer refuses to send anything I did not see.
(2) Label every gate as code or as a written rule, and check which one your real traffic passes. I called both "hard" before I counted.
(3) Point the ledger at the artifact. A message identifier lets anyone find the real email later.
(4) Give the agent authority to prepare, never standing authority to act on people. One wrong email to a faculty member costs more than the agent saves me in a month.
(5) Count your edits and rejections. Without that number you cannot tell an approval step from a habit of clicking yes.

9 What's Next

- Route Hermes-sent email through the Ultra composer, or give the Hermes path its own hash-bound approval, so the written rule becomes code.
- Stamp agent-composed Nexus entries consistently, so the ledger records who wrote the words as well as the channel.
- Log each draft's fate (sent as is, edited, rejected) and report the rate after a month. That measurement, together with the share of outbound email that passes the hash-bound gate rather than the written rule, would settle the main claim of this note, that the approval step catches errors. Until it exists, the claim rests on one day's ledger.
- Open Atrium beyond Research Computing admins once its write actions have run cleanly on staging.

Sources

Nexus, Atrium, people-mcp and vault-mcp are private because they touch campus or personal records. The operator desk is public, including the double-approval composer specification. Repository: https://github.com/charles-forsyth/ultra-workstation. Project page: https://charles-forsyth.github.io/ultra-workstation/. Private records used: the Nexus interaction log (read-only queries for 28 October 2026 and for 1 to 28 October 2026), the Nexus scoreboard and `SPEC.md`; the Atrium `README.md` and `docs/SPEC.md`; the people-mcp and vault-mcp READMEs; and the Hermes standing memory (the ticket rule).

References

[1] Charles E. Forsyth III. 2026. One Day on the Research Stack. Field Notes. https://charles-forsyth.github.io/papers/2026-10-03-one-day-on-the-research-stack.html
[2] Charles E. Forsyth III. 2026. A Road to Nautilus in One Day. Field Notes. https://charles-forsyth.github.io/papers/2026-10-06-a-road-to-nautilus-in-one-day.html
[3] Charles E. Forsyth III. 2026. Directing a Fleet of AI Builders. Field Notes. https://charles-forsyth.github.io/papers/2026-10-27-directing-a-fleet-of-ai-builders.html
[4] Charles E. Forsyth III. 2026. OmNet: A Homestead Network Where Code Decides. Field Notes. https://charles-forsyth.github.io/papers/2026-10-29-omnet-a-homestead-network-where-code-decides.html
[5] Model Context Protocol. 2025. Model Context Protocol Specification, Version 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18 Accessed 28 October 2026.

[6] 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 2023)*. 68539–68551. doi:10.52202/075280-2997

[7] 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 *The Eleventh International Conference on Learning Representations (ICLR 2023)*. https://iclr.cc/virtual/2023/poster/11003

[8] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science* 18, 6 (2024), 186345. doi:10.1007/s11704-024-40231-1

[9] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *Comput. Surveys* 54, 4 (2021), 1–37. doi:10.1145/3447772

[10] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 3580–3599. doi:10.1109/TKDE.2024.3352100

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[12] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23)*. ACM, 79–90. doi:10.1145/3605764.3623985

[13] OWASP Foundation. 2025. OWASP Top 10 for Large Language Model Applications, 2025 Edition. https://genai.owasp.org/llm-top-10/ Accessed 28 October 2026.

[14] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 10471–10506. doi:10.18653/v1/2024.findings-acl.624

[15] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*. 82895–82920. https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html

[16] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating Prompt Injections by Design. arXiv preprint 2503.18813. https://arxiv.org/abs/2503.18813

[17] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. Red Teaming Language Models with Language Models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 3419–3448. doi:10.18653/v1/2022.emnlp-main.225

[18] J. H. Saltzer and M. D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. doi:10.1109/PROC.1975.9939

[19] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (1996), 38–47. doi:10.1109/2.485845

[20] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2020. *Zero Trust Architecture*. NIST Special Publication 800-207. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-207

[21] Dick Hardt. 2012. *The OAuth 2.0 Authorization Framework*. RFC 6749. Internet Engineering Task Force. doi:10.17487/RFC6749

[22] National Institute of Standards and Technology. 2015. *Secure Hash Standard (SHS)*. FIPS PUB 180-4. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.180-4

[23] Bruce Schneier and John Kelsey. 1999. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security* 2, 2 (1999), 159–176. doi:10.1145/317087.317089

[24] Raja Parasuraman and Dietrich H. Manzey. 2010. Complacency and Bias in Human Use of Automation: An Attentional Integration. *Human Factors* 52, 3 (2010), 381–410. doi:10.1177/0018720810376055

[25] Ben Green and Yiling Chen. 2019. The Principles and Limits of Algorithm-in-the-Loop Decision Making. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24. doi:10.1145/3359152

[26] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*. 32808–32824. https://proceedings.iclr.cc/paper_files/paper/2024/hash/8b4add8b0aa8749d80a34ca5d941c355-Abstract-Conference.html

[27] LangChain. 2026. LangGraph Interrupts (Human-in-the-Loop). https://docs.langchain.com/oss/python/langgraph/interrupts Accessed 28 October 2026.

[28] Charles E. Forsyth III. 2026. Nexus: A Relationship Ledger for Research Computing. Field Notes. https://charles-forsyth.github.io/papers/2026-10-29-nexus-a-relationship-ledger-for-research-computing.html