The Living Wiki: Schema-Driven LLM Knowledge Bases as Persistent Agent Memory

Anonymous Author(s)

Abstract

We describe the *LLM-maintained wiki*—a pattern in which an agent incrementally builds and sustains a structured, cross-referenced knowledge base between queries—and evaluate it as an instance of the procedural Skill paradigm. The central artifact is the *vault schema*: a `CLAUDE.md` configuration file encoding the agent's read/write protocols as explicit procedural instructions. On a 56-document open-source corpus we find a diagnostic separation: at matched retrieval depth the wiki reaches ground-truth sources at least as often as a strong hybrid RAG baseline (90.0% vs. 87.5% any-source hit at k=10), yet scores lower overall—the bottleneck is not source access but answer synthesis over long wiki contexts. Entity aggregation is the exception, where persistent synthesis yields a clear gain (1.70 vs. 1.30). A consequence-language pilot shows that framing schema steps as obligations ("the vault rots if you skip this") eliminated all observed maintenance deferrals (0/3) compared to reminder framing (4/5 deferred), suggesting that the *linguistic* properties of a Skill file are a first-order reliability lever.

CCS Concepts: • **Computing methodologies** → **Intelligent agents**; *Knowledge representation and reasoning*.

Keywords: LLM agents, personal knowledge management, persistent memory, agent skills, vault schema, compounding knowledge

1 Introduction

RAG is stateless: on every query the agent re-derives knowledge from raw fragments, generates an answer, and discards the synthesis [6]. A year of querying yields no more structured understanding than a day.

We describe and empirically evaluate an alternative—the *LLM-maintained wiki* [5]—whose distinguishing feature is the *vault schema*: a structured `CLAUDE.md` file specifying the agent's read and write protocols in explicit procedural terms. The vault schema is a procedural Skill in the sense of the workshop [1, 4]: it is not trained into the model but configured externally, read at session start, version-controlled, and improvable without touching weights. Our central empirical finding is that the *linguistic form* of this Skill file—consequence framing vs. reminder framing—was the single highest-impact change we made to the system.

The architecture is three-layered: immutable raw sources (S), an LLM-maintained wiki (V), and the vault schema (Σ), forming W=(S, V, Σ). Our methodology is self-referential: the wiki manages the paper's own source material, and the append-only activity log provides our empirical data.

Contributions.

1. A formal characterization of the LLM wiki pattern: its three-layer architecture $W = (S, V, \Sigma)$, three core operations (ingest, query, lint), and the compounding dynamics that distinguish it from stateless RAG.
2. An instrumented implementation with append-only activity logging, a hot-cache acceleration layer, and cross-reference tracking.
3. Empirical measurements on a 56-document open-source corpus (Astral's `uv`): 40 queries across four types reveal that a strong hybrid RAG baseline leads overall while the wiki produces a clear gain on entity aggregation; retrieval diagnostics separate source access from answer synthesis as the binding constraint.
4. Structural measurements documenting synthesized-wiki growth from 21 to 102 pages and a concrete schema failure mode (199 unresolved wikilinks by V3).
5. A consequence-language pilot and six measurement-derived design principles for vault schema construction.

2 Background and Related Work

Retrieval-Augmented Generation. Lewis et al. [6] introduced RAG as a method for grounding LLM generation in non-parametric, updatable knowledge. Its fundamental limitation is statelessness: synthesis is ephemeral, relationships identified in one query are not stored, and the next query re-derives them.

Persistent Agent Memory. MemGPT [9] addresses context-window limits via OS-style memory paging between in-context and external storage. Park et al.'s Generative Agents [10] introduce a natural-language memory stream that periodically synthesizes experiences into higher-level reflections—the closest prior architecture to our V layer. Both differ from our pattern in transparency: MemGPT's storage is opaque, and the Generative Agents stream is behaviorally scoped and simulation-local. The LLM wiki is entirely human-readable markdown stored in version-controlled files. Concurrently, A-MEM [14] applies Zettelkasten-style linking to LLM agent memory; unlike A-MEM, our formalization treats the governing protocol Σ as a separable, inspectable artifact rather than emergent behavior.

Agent Skill Libraries. Voyager [13] demonstrates lifelong skill acquisition: a growing library of executable code snippets, each encoding a solved task. Our vault schema is structurally analogous—the governing procedural Skill for knowledge management—and the wiki pages are the compound artifact analogous to Voyager's skill entries.

LLM-Maintained Wikis. Karpathy's `llm-wiki` idea file [5] crystallizes the pattern we evaluate: raw files remain immutable, an LLM compiles them into an evolving markdown wiki, and a schema tells the agent how to ingest, query, and lint that wiki. Our work is an empirical stress test and formalization of this pattern.

Structured Agent Instructions. Santos et al. [11] analyzed 328 real-world `CLAUDE.md` files, finding substantial heterogeneity and no shared design vocabulary. Lulla et al. [8] showed that well-structured `AGENTS.md` files reduce agent runtime by 28.64% and token consumption by 16.58%. The vault schema contributes a domain-specific design vocabulary for knowledge management tasks and, via the consequence-language pilot, evidence that *how* a Skill step is framed—not only whether it is present—predicts agent adherence.

SkillsBench [7] benchmarks this design space directly: curated, human-authored Skills raise pass rates by 16.2 percentage points on average, while self-generated Skills provide no benefit on average—models struggle to author the procedural knowledge they benefit from consuming. The vault schema occupies a middle position: the protocol (Σ) is human-authored and human-maintained, while the content V it governs is entirely agent-generated. This separation—curated procedure over agent-generated content—is the design choice we argue for.

Historical Context. Vannevar Bush's Memex [2] first articulated the vision of a personal, associative knowledge machine organized by cross-reference rather than index. The LLM wiki is a practical realization of this vision.

3 The LLM Wiki Architecture

3.1 Overview

The LLM wiki is a three-layer architecture for personal knowledge management. Formally, the system $W = (S, V, \Sigma)$ consists of:

- S: the *source corpus*—immutable raw documents (articles, transcripts, notes)
- V: the *wiki*—a typed, cross-linked graph of markdown pages, LLM-maintained
- Σ: the *vault schema*—a structured `CLAUDE.md` file encoding the agent's read/write protocols as procedural instructions

The key invariant: S is append-only; V is LLM-maintained; Σ is human-maintained. Separation of responsibilities across layers makes the system maintainable as the vault grows.

3.2 Layer 1: Raw Sources (S)

Sources are stored in a flat `sources/` directory with minimal YAML frontmatter (title, type, date, tags) and are never modified after ingest. The immutability invariant serves two purposes: provenance (every wiki page's lineage is traceable) and re-synthesis (the wiki can be rebuilt from sources if the schema changes, without data loss).

3.3 Layer 2: The Wiki (V)

The wiki consists of typed markdown pages in three directories. *Entities* (`entities/`) are factual records of people, companies, and projects, updated when new information arrives. *Concepts* (`concepts/`) are synthesized pages covering ideas and frameworks, cross-referenced from multiple sources. *Analyses* (`analyses/`) are time-stamped writeups authored by agent and user.

Every page uses wikilink syntax (`[[page-name]]`) to cross-reference other pages. Wikilinks form the associative graph the agent traverses without reading the full corpus.

Navigation is supported by three root-level files: `index.md` (a map of every page), `hot-cache.md` (a summary of the most recent ingest session, enabling warm starts), and `log.md` (an append-only record of every ingest and query—our primary measurement instrument).

3.4 Layer 3: The Vault Schema (Σ)

The vault schema encodes the agent's operating protocols as explicit procedural instructions. It specifies a six-step **write order** for ingest: (1) write raw to `sources/`, (2) update/create entity and concept pages, (3) write analysis if applicable, (4) update `index.md`, (5) append to `log.md`, (6) update `hot-cache.md`.

Critically, the schema uses **consequence language** rather than reminders: "if you skip step 4 or 6, the vault rots." Section 4 shows this language is a likely adherence driver. The schema also specifies read order (hot-cache first, then index, then targeted pages), page format requirements, and a no-duplicate rule.

3.5 Operations

Three operations are defined by Σ. Ingest takes a raw source and produces all six artifacts in the write order above. Query reads hot-cache → index → targeted pages → synthesized answer. Lint validates that all indexed pages exist, all wikilinks resolve, and hot-cache reflects the most recent log entry.

3.6 Compounding Dynamics

The wiki's value can compound with ingestion volume, but compounding should be treated as an empirical property rather than an architectural guarantee. Cross-references have a larger candidate set as the wiki grows, concept pages can be refined rather than duplicated, and the hot-cache can

reduce cold-start cost for recent content. Section 4.2 shows this dynamic only partially: link density and clustering rose quickly from V1 to V3, while both curves saturated and unresolved links accumulated.

4 Empirical Evaluation

4.1 Setup

We built a wiki using the vault schema on a corpus of documents from Astral's open-source uv Python package manager[1]: 24 GitHub issues (mix of bugs, features, and discussions), 21 pull requests with descriptions and review metadata, 9 changelog entries, and 2 blog posts. This corpus was chosen because graph-structured synthesis does genuine load-bearing work on software project knowledge: contributor–module relationships, design evolution, security decisions, and cross-cutting trajectories are patterns that exist in the source documents but are not explicitly stated in any single one. The vault schema was used unchanged; we tested the schema, not a redesigned one.

Across the final V3 snapshot, the activity log records all 56 corpus entries (55 ingests and one strategy-driven skip). The synthesized wiki graph contained 102 pages: 46 entity pages, 46 concept pages, and 10 analysis pages. The structural experiment excludes raw source files and root operational files, so graph metrics refer only to the LLM-maintained wiki layer V.

Experiment 1 (RAG vs. wiki quality). We designed 40 queries (10 per type) grounded in the source corpus. *Type A* (single-document factual): answers exist in one document. *Type B* (entity aggregation): answers require collecting information about one contributor or module across multiple documents—entity pages aggregate this. *Type C* (temporal trajectory): answers require ordering events over time—analysis pages encode evolution; RAG retrieves unordered chunks. *Type D* (implicit relationship): answers require traversing relationships not stated in any single document—e.g., "Which contributors worked on both the index module and security features?"—the wikilink graph surfaces these intersections. All 40 answers are verifiable in the source documents; none require wiki-authored content unavailable in raw sources.

Both systems ran over the same source files. The final RAG baseline uses an enhanced source-level index with hybrid BM25+dense retrieval and source-context assembly. The wiki navigator uses hybrid BM25+dense traversal over the wiki index, treats the hot-cache as nonterminal diagnostic context, and expands selected wiki pages into linked source pages. We sweep matched budgets k={3, 5, 10}: RAG receives top-k source contexts, while the wiki receives the same budget for targeted wiki pages and linked source pages. Retrieved contexts were scored by a scripted, LLM-assisted scorer: GPT-4o-mini first generated an answer from the retrieved context only, then verified the answer against author-specified required facts and citation checks. Scores are 0–3 (0 = unanswerable, 3 = at least 80% required-fact coverage with a ground-truth citation) with a binary can-answer flag. A blind human-rating study and Cohen's κ remain pending.

Experiment 2 (graph structure). We measured the induced wikilink graph over `entities/`, `concepts/`, and `analyses/` at V1, V2, and V3. Metrics include page count, unique directed edges, raw wikilink occurrences, mean links/page, clustering coefficient, unresolved wikilinks, and top hubs by in-degree.

Experiment 3 (schema adherence). Three ingest batches and a consequence-language pilot (second vault with "vault rots" replaced by "remember to update", 5 ingests) were logged from a prior self-referential pilot on academic source material.

4.2 Results

Graph structure and schema adherence. Table 1 reports the structural trajectory. From V1 to V3, synthesized pages grew 4.9× (21 to 102) while unique in-wiki edges grew 11.1× (59 to 656). Mean unique links/page rose from 2.81 to 6.43 and the clustering coefficient rose from 0.262 to 0.494. This supports the weaker, measurable claim that later ingests touched a denser associative structure. It does *not* support an unqualified superlinear growth claim: the curves saturate, and unresolved wikilinks grew from 0 to 199 by V3.

Table 1. Structural growth of the synthesized wiki layer V across sequential uv snapshots. Raw source files are excluded.

Metric	V1	V2	V3
Pages	21	75	102
Entities	3	31	46
Concepts	18	36	46
Analyses	0	8	10
Unique in-wiki edges	59	456	656
Mean unique links/page	2.81	6.08	6.43
Clustering coefficient	0.262	0.478	0.494
Unresolved wikilinks	0	47	199

Core load-bearing adherence—updating `index.md` and `hot-cache.md` for observed ingests—held at 100% across all three uv batches. The number of pages updated per ingest rose from 3.53 to 6.89 to 7.61, consistent with the agent reusing and refining existing pages as the wiki grew. In the consequence-language pilot (academic corpus), maintenance steps were systematically deferred in 4 of 5 ingests under reminder language vs. 0 of 3 under consequence language. This *deferral pattern* is consistent with Lulla et al. [8] and

[1] https://github.com/astral-sh/uv

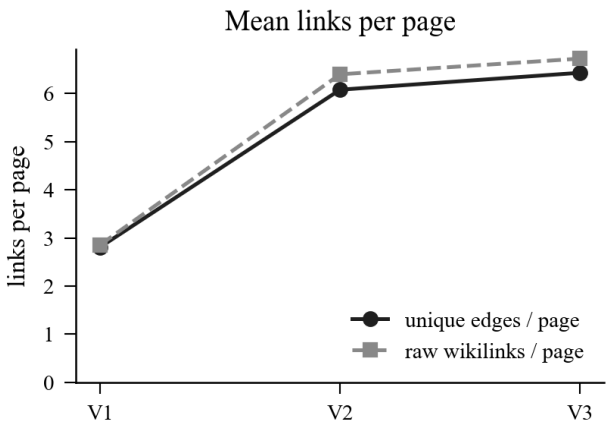

Figure 1. Mean in-wiki links/page rises sharply after the first batch and then begins to saturate, matching the table-level graph analysis.

Chatlatanagulchai et al. [3]; our finding extends this: consequence framing appears to convert advisory steps into load-bearing obligations, although the pilot is too small for statistical significance.

RAG vs. wiki quality (uv corpus, N=40). Table 2 reports the final depth-matched results. The stronger RAG baseline improves monotonically with depth and beats the current wiki navigator overall. At k=10, RAG scores 1.32 vs. the wiki's 0.95, with 42.5% vs. 30.0% can-answer. Bootstrap 95% CIs for mean score are [1.00, 1.68] for RAG and [0.68, 1.23] for the wiki; the paired wiki-minus-RAG score difference is −0.38 [−0.72, −0.03].

Table 2. Final context quality: enhanced hybrid RAG vs. LLM-maintained wiki (uv corpus, N=40, scripted LLM-assisted scorer). Score: 0–3. Can-ans: fraction of queries with sufficient context to answer. Source-hit reports any/all ground-truth source access.

Depth	System	Score	Can-ans	Source-hit any/all
k=3	RAG	1.10	32.5%	67.5% / 37.5%
k=3	Wiki	0.85	22.5%	85.0% / 40.0%
k=5	RAG	1.25	40.0%	75.0% / 40.0%
k=5	Wiki	0.88	22.5%	90.0% / 47.5%
k=10	RAG	1.32	42.5%	87.5% / 52.5%
k=10	Wiki	0.95	30.0%	90.0% / 55.0%

The by-type picture at k=10 is narrower than the original hypothesis. RAG dominates Type A single-document factual queries (2.40 vs. 0.80; 90% vs. 30% can-answer), where direct source retrieval is enough. The wiki's clearest advantage is Type B entity aggregation (1.70 vs. 1.30; 60% vs. 40% can-answer), consistent with entity pages preserving contributor–module synthesis across documents. Type C is essentially tied on score (1.00 wiki vs. 1.10 RAG) with low can-answer rates for both systems. Type D remains weak for both (0.30 wiki vs. 0.50 RAG), despite high wiki source-hit rates, indicating that the current wiki traversal reaches relevant material but does not yet assemble enough relationship evidence into answerable context.

4.3 Failure Modes

Index drift: without explicit enforcement, `index.md` fell out of sync within two sessions. Consequence language and the lint operation prevented recurrence. *Hot-cache staleness*: the cache was updated on ingest but not on lint, leaving it stale. We updated the schema to require refresh after any vault-mutating operation. *Hot-cache over-triggering*: earlier runs allowed the hot-cache to terminate retrieval, which hid relevant index pages on complex synthesis questions. The final evaluation treats hot-cache as nonterminal and matches wiki depth to the RAG budget. *Dangling wikilinks*: by V3, the graph contained 199 unresolved wikilinks. The most likely cause is that the schema rewarded aggressive linking but did not require a post-ingest link-resolution pass. *Source access without answerability*: even after source expansion, the wiki often touches a ground-truth source without assembling the required facts into a concise answerable context. This is the main remaining failure mode in the final runs.

5 Design Principles

Each principle below is derived from a specific measurement or failure mode in our evaluation.

P1 — Schema first. In an unstructured pilot, the vault degraded into a flat file dump within two ingest sessions once no schema governed page creation. Define the vault schema before ingesting anything; retrofitting it requires a full re-synthesis pass.

P2 — Use consequence language. Replacing "remember to update the index" with "if you skip the index update, the vault rots" eliminated maintenance deferrals entirely in our pilot (4/5 deferred under reminder framing; 0/3 under consequence framing). This was the single highest-impact schema change we made, and it costs nothing but word choice.

P3 — Keep sources immutable. Never modify a source file after ingest. Synthesis lives in wiki pages that cite sources, preserving provenance and enabling re-synthesis if the schema evolves.

P4 — Treat the hot-cache as a guide, not a gate. In early runs, allowing the hot-cache to terminate retrieval masked relevant index pages; complex synthesis queries were cut off before reaching the required evidence. Use hot-cache as

nonterminal diagnostic context—it narrows the search but must not foreclose it.

P5 — Make the log append-only and structured. The append-only `log.md` was simultaneously the audit trail, the primary empirical data source for this paper, and the operational debugging tool when adherence degraded. Without it the deferral pattern in Experiment 3 would have been unobservable. Structure enables parsing; append-only guarantees integrity.

P6 — Lint links as a first-class operation. By V3, the wiki graph contained 199 unresolved wikilinks—the schema had rewarded aggressive cross-referencing but required no post-ingest resolution pass. High link density is only useful when the links resolve; linting should be a mandatory step in the write order, not an optional cleanup.

6 Discussion

Access vs. answerability. Table 2 separates two mechanisms easy to conflate. At k=10, the wiki reaches at least one ground-truth source slightly more often than RAG (90.0% vs. 87.5%) and all ground-truth sources slightly more often (55.0% vs. 52.5%), yet it still scores lower overall. Graph traversal can improve access to relevant material; answerability still depends on whether the retrieved wiki and source pages expose the exact required facts in a compact form. Entity aggregation is the one query type where persistent synthesis currently produces a clear gain; for implicit relationship queries (Type D), source access is high but answer assembly remains the binding constraint. Figure 1 adds the structural view: density grew substantially, but the plateau and dangling-link count show that graph growth alone is not sufficient.

The vault schema in the Skills ecosystem. The vault schema's position in the broader Skills landscape is worth naming precisely. Santos et al. [11] found that real-world `CLAUDE.md` files are highly heterogeneous with no shared design vocabulary. Lulla et al. [8] showed that structured instruction files reduce runtime and token use, but their analysis treats presence and structure as the primary variables. Our finding extends this: *how* a step is framed within the Skill file—whether as an obligation with named consequences or as an advisory reminder—appears to predict agent adherence independently of whether the step is present at all. Consequence language may be a generalizable Skill-authoring primitive, not merely a quirk of our domain.

SkillsBench [7] benchmarks this design space across 86 tasks: curated Skills raise pass rates by 16.2 percentage points on average while self-generated Skills provide no benefit on average. The vault schema occupies a deliberately middle position: the protocol Σ is human-authored and human-maintained, while the content V it governs is entirely agent-generated. This separation—curated procedure over agent-generated content—avoids the opacity of fully automated memory systems while sidestepping the self-generation failure mode SkillsBench documents.

Connection to Reflexion and open problems. Our finding resonates with Reflexion [12]: both systems demonstrate that LLMs can accumulate structured knowledge through linguistic artifacts rather than weight updates. Reflexion targets task-specific episodic traces; our wiki targets persistent declarative synthesis. Open problems include schema versioning as the vault evolves, multi-user conflict resolution for shared vaults, and automated hot-cache threshold tuning.

Limitations. All 40 queries were scored by an LLM-assisted pipeline using author-defined required facts, without a blind human review procedure. A second independent human rater and Cohen's κ remain pending before results can be treated as robust human judgments. The consequence-language pilot (N=8) is insufficient for a statistically significant claim and has an experimenter-awareness confound. The final RAG baseline is one implementation: different chunking, reranking, or answer-generation prompts could change the absolute scores. The wiki navigator does not yet perform explicit multi-hop evidence assembly; the low Type D result reflects a navigator limitation, not evidence that graph-structured memory cannot support implicit relationship queries.

7 Conclusion

The LLM wiki offers a practical path from stateless retrieval to compounding, inspectable agent memory, enabled by the vault schema—a procedural Skill governing knowledge management across sessions. Our evaluation reveals a diagnostic separation: the wiki reaches ground-truth sources as well as a strong RAG baseline, but answer synthesis over long wiki contexts remains the binding constraint. The exception is entity aggregation, where persistent synthesis produces a clear gain. The consequence-language pilot suggests that Skill-file authoring choices—specifically the difference between obligation framing and reminder framing—are a first-order reliability lever, potentially generalizable beyond knowledge management. The persistent wiki is not a feature of the LLM; it is a feature of the schema.

References

[1] John R. Anderson. 1982. Acquisition of cognitive skill. *Psychological Review* 89, 4 (1982), 369–406. doi:10.1037/0033-295X.89.4.369

[2] Vannevar Bush. 1945. As We May Think. *The Atlantic* (July 1945). https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

[3] Worawalan Chatlatanagulchai, Kundjanasith Thonglek, Brittany Reid, Yutaro Kashiwa, Pattara Leelaprute, Arnon Rungsawang, Bundit Manaskasemsak, and Hajimu Iida. 2025. On the Use of Agentic Coding Manifests: An Empirical Study of Claude Code. arXiv preprint arXiv:2509.14744. https://arxiv.org/abs/2509.14744

[4] Vincent Hsiao, Mark Roberts, and Leslie Smith. 2025. Procedural Knowledge Improves Agentic LLM Workflows. arXiv preprint arXiv:2511.07568. https://arxiv.org/abs/2511.07568

[5] Andrej Karpathy. 2026. LLM Wiki: A Pattern for Building Personal Knowledge Bases Using LLMs. GitHub Gist. https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f Created April 4, 2026.

[6] 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 (NeurIPS)*. https://arxiv.org/abs/2005.11401

[7] Xiangyi Li, Wenbo Chen, Yimin Liu, et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv preprint arXiv:2602.12670. https://arxiv.org/abs/2602.12670

[8] Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents. arXiv preprint arXiv:2601.20404. https://arxiv.org/abs/2601.20404

[9] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. arXiv preprint arXiv:2310.08560. https://arxiv.org/abs/2310.08560

[10] 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 *ACM Symposium on User Interface Software and Technology (UIST)*. https://arxiv.org/abs/2304.03442

[11] Helio Victor F. Santos, Vitor Costa, Joao Eduardo Montandon, and Marco Tulio Valente. 2025. Decoding the Configuration of AI Coding Agents: Insights from Claude Code Projects. In *1st International Workshop on Agentic Engineering (AGENT), co-located with ICSE 2026*. https://arxiv.org/abs/2511.09268

[12] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/2303.11366

[13] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research (TMLR)* (2024). https://arxiv.org/abs/2305.16291

[14] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic Memory for LLM Agents. arXiv preprint arXiv:2502.12110. https://arxiv.org/abs/2502.12110