

Complex Knowledge Curation using Agentic Ontological Notebook Memory

Gully A. Burns
gullyburns@gmail.com
SciKnow.io
San Carlos, California, USA

Paul Groth
p.t.groth@uva.nl
University of Amsterdam
Amsterdam, The Netherlands

Abstract

In the same way that every software system uses a data model (as database schema, type system, or API contracts), large language model (LLM)-powered agents makes domain-specific semantic distinctions in prompts, tool definitions, data sources, etc. These ontological commitments are implicit, hard to test, and difficult to refine systematically. Furthermore, the way that agents preserve context across sessions (agent memory) usually make few, if any, domain-specific ontological commitments and rely only on the inherent intelligence of LLMs to make those distinctions. We argue that making these commitments explicit improves agent performance for domain-specific work.

To that end, we present *Skillful-Alhazen* as a personal AI research assistant that uses skills to define domain-specific instructions, tools (coded as scripts), and a new kind of ‘ontological notebook’ as memory. Users interact with the assistant in plain language, and the assistant applies the semantic definitions from its memory schema to execute work. It retains structured context based on data in the explicitly defined domain model it can access, query, and modify. We use TypeDB to implement the ontological notebook. TypeDB is a recently-commercialized, closed-world, declarative, logic-based persistent knowledge graph with several desirable properties in this application: including type-safety, strong inference, easy refactoring, and scale.

We demonstrate the use of Skillful-Alhazen in three domains: job hunting, technology investigation, and disease mechanism curation, each based on separate skill definitions. Our tool is open-source available from <https://github.com/sciknow-io/skillful-alhazen>.

CCS Concepts

• **Computing methodologies** → **Knowledge representation and reasoning**; *Natural language processing*; • **Information systems** → *Information extraction*.

Keywords

knowledge graphs, ontological memory, agentic systems, TypeDB, LLM agents, curation, information extraction

ACM Reference Format:

Gully A. Burns and Paul Groth. 2026. Complex Knowledge Curation using Agentic Ontological Notebook Memory. In *Proceedings of the 1st ACM Conference on Agentic and AI Systems (CAIS '26)*, May 26, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3786335.3813226>

1 Introduction

When an agent persists memory, it necessarily commits to some representation of what it knows. Yet current agent memory architectures hide semantic commitments in prompts, class and function code, heuristics, and natural-language instructions. This where they are implicit, hard to test, and challenging to refine systematically. This is sometimes treated as a feature rather than a limitation. After all, Sutton’s “bitter lesson” enjoins us to avoid domain-specific structure because it won’t scale. But for knowledge curation tasks, where the goal is to build a structured representation of a domain, the schema specification is essential. In the current era of frontier coding agents, the engineering burden that once made ontological commitment impractical has collapsed to zero. A human domain expert can generate schema from scratch and query them effectively by chatting to an frontier-level coding agent [22].

We present *Skillful-Alhazen*, an agentic curation system that makes these commitments explicit through an *ontological notebook* powered by TypeDB [19]. Ontological commitments are localized in *skills*: composable modules that each define a TypeDB schema extension alongside the scripts and agent instructions that operationalize it.

The agentic harness provided by Skillful-Alhazen consists of the high-level TypeDB schema that differentiates between *conceptual entities* (the things under study) and the *information content entities* (that describe them). This separation lets the agent curate within a typed conceptual framework while preserving full provenance from source to interpretation.

The key contributions are:

- An **explicit ontological memory** for agents, where each domain skill defines its ontological commitments as a TypeDB schema extension; using TypeDB’s polymorphic type system to support inheritance, role-based relationships, and native inference.
- **Curation-centered schema and processes** keyed to enforcing provenance by separating conceptual entities, raw content, extracted structure, and agent-generated analysis.
- A **domain modeling methodology** keyed to providing iterative support for defining, testing, and refining a domain schema through usage.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CAIS '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-2415-2/26/05

<https://doi.org/10.1145/3786335.3813226>

- Evidence from **three domains** (job hunting, technology investigation, and rare-disease mechanism curation [14]) demonstrating utility across different kinds of curation.

2 Domain-Specific Semantic Modeling for Agentic Memory

A central claim of this work is that using a well-defined domain model provides value to an agent performing a curation-like task. This requires design work to decide what entity types, relationship roles, and inferential rules adequately represent a domain and may be undertaken by an agent.

This is in contrast to the dominant paradigm in LLM-based agent memory, where schema design is either absent (unstructured vector stores, flat key-value memory) or fully automated (LLM-extracted entity types in GraphRAG pipelines [2]). Automated extraction has real value for breadth, but it produces a *discovered* ontology shaped by corpus statistics rather than a *designed* ontology shaped by domain understanding.

TypeDB's type system supports the modeling decisions that domain experts need to make:

- **Type hierarchies.** Entities inherit properties from super-types so queries over the supertype automatically include instances of all subtypes.
- **Role-based relationships.** Relationships in TypeDB are not binary edges between nodes but typed *relations* with named *roles* that restrict which entity types may participate. A causal-association relation, for example, might require a gene to play the cause role and a phenotype to play the effect role: a constraint that a labeled property graph cannot express at the schema level.
- **Schema-level logic.** TypeDB's function system allows domain based derivations and inference. This provides a layer of semantic-aware computation that we have yet to explore but is a powerful, deterministic approach to domain-specific reasoning.

These capabilities mean that the domain modeling activity produces infrastructure the agent actively uses during reasoning.

Iterative schema refinement. A key challenge is that it is highly likely that the schema will need to evolve as edge-cases are considered, errors are revealed, or if the scope of the data being considered changes. We are developing a feedback loop for iterative refinement. Agent-generated notes and extraction failures serve as loss signals: when the agent cannot cleanly represent an encountered entity within the current schema, or usage of the system reveals a schema derived error, the agent records the deficit / error as a note in its memory, and subsequently treats it as a github issue to be addressed under normal coding agent practices. This closes the loop between curation experience and ontological design - so that our tools can learn from experience in a systematic, documented way in order to optimize the conceptual vocabulary framework being developed.

3 System Architecture

Skillful-Alhazen consists of three layers (Figure 1): an *agent layer* that handles natural-language interaction; a *skill layer* that provides domain-specific capabilities; an *ontological memory layer* backed by

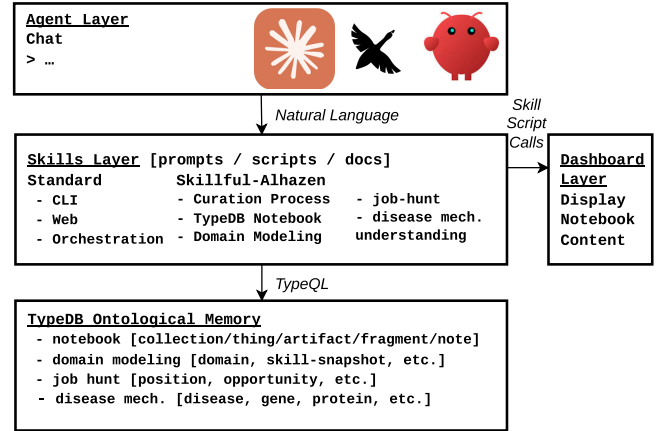


Figure 1: Three-layer architecture of Skillful-Alhazen. The agent interacts with users in natural language, selects and executes skills, which in turn read and write to the TypeDB ontological memory through typed schemas.

TypeDB. An additional *dashboard layer* provides navigable views over structured notebook content as a webapp.

3.1 Agent Layer

The agent layer provides LLM-powered reasoning that drives all interactions. Skillful-Alhazen supports two execution modes: *Interactive Mode* where the user interacts with Claude Code directly using natural language to interact with their data, and *Agentic Mode* where an OpenClaw deployment acts as an always-on service with scheduled searching, messaging integration, and cron-driven knowledge accumulation.

In both modes, the agent operates through natural language. The user never writes queries or call APIs directly. Instead, the agent selects appropriate skills based on the user's intent and executes them autonomously.

3.2 Skill Architecture

Skills are self-contained, composable modules that extend both the ontological schema and the agent's behavioral repertoire. Each skill consists of:

- (1) A **TypeDB schema extension** defining domain-specific entity types and relationships.
- (2) **Python scripts** for external API access, data transformation, complex queries, or any other complex reproducible function.
- (3) **Agent instructions** in two tiers: a slim `SKILL.md` (~30 lines) loaded at startup for skill selection, and a full `USAGE.md` read on demand for execution details. Additional context may be included as needed.
- (4) **Human-readable instructions**: a standard `README.md` file.

Most skills follow a common *curation pattern*: **searching** (identifying items of interest from external sources), **ingestion** (fetching and storing raw content as artifacts), **sensemaking** (the agent analyzing content and generating fragments and notes), **analysis**

(querying across accumulated knowledge), and **reporting** (dashboard views and summaries).

3.3 Ontological Memory

The core of Skillful-Alhazen is its TypeDB-backed ontological memory. Unlike vector databases that store embeddings without type information, TypeDB enforces a *schema* that defines the conceptual vocabulary available to the agent.

The schema is rooted in a distinction between *domain things* (real-world entities such as systems, diseases, genes, companies, and job postings) and *information content entities* (ICEs) that carry textual or structured content. This mirrors the Information Artifact Ontology (IAO) [1] distinction between entities that exist in the world and representations of those entities.

The ICE schema has three high-level types:

Artifacts are raw captured content — API responses, HTML pages, PDF documents. They represent the unprocessed source material.

Fragments are structured extractions from artifacts — a sentence or section from a paper, or a list of job requirements extracted from a posting.

Notes are the agent’s own analysis — fit scores, mechanism hypotheses, comparative syntheses. Notes are always attributed to the agent and are linked to ICEs. A key aspect of this design is that notes can be linked to notes.

This approach enables provenance: every notes are linked to concrete information sources. When the agent generates a synthesis about, say, the therapeutic landscape for a rare disease, the user can follow the chain back to the original database records.

4 Demonstration

We demonstrate Skillful-Alhazen through three skills that showcase ontological memory across different domains.

4.1 Job Market Analysis

The job-hunt skill tracks job applications by ingesting postings from URLs, extracting structured requirements (technology stacks, experience levels, domain expertise), performing fit analysis against a user-maintained skill profile, and identifying skill gaps across a portfolio of prospects.

A typical interaction begins with the user providing a job posting URL. The agent fetches the page (creating an *artifact*), extracts structured requirements as *fragments*, and generates a fit analysis *note* scoring the match. Over time, the knowledge graph accumulates a structured landscape of the job market, enabling queries like “What skills appear most frequently in my top-ranked positions?” that would be impossible with flat document storage.

4.2 Technology Investigation

The tech-recon skill supports systematic, goal-driven investigation of competing computational systems in order to address a predetermined goal. A structured interview elicits user-defined success criteria; the agent then discovers candidate systems, ingests documentation and source repositories (by creating local copies of codebases where possible), generates structured comparison notes,

runs structured analysis workflows on demand and produces Observable Plot visualizations mapped to those criteria — all surfaced through a live dashboard.

The key modeling decisions are (a) promoting *success criteria* to first-class entities so that every note and analysis is anchored to an explicit evaluation dimension, and (b) tracking candidate systems through a typed pipeline (candidate → confirmed → ingested → analyzed), so that investigation progress is itself a queryable graph property.

4.3 Disease Mechanism Curation - Leveraging the Monarch Initiative’s DisMech system

DisMech [14] is a pre-alpha curation pipeline from the Monarch Initiative that encodes rare-disease pathophysiology as structured YAML files, one per disorder, curated and reviewed through GitHub pull requests. As of April 2026 the corpus spans 797 disorders.¹ Each file records pathophysiology mechanisms, causal genes, phenotypes (HPO terms), treatments (MAXO codes), and supporting evidence (PMIDs) in a LinkML-defined schema.

The pipeline is narrow by design: a single-purpose CI workflow produces YAML outputs rendered as static web pages. There is no database backend, so aggregate queries require custom Python scripts that scan every YAML file on every query.

We implemented two DisMech skills for Skillful-Alhazen: First, we used DisMech’s comprehensive LinkML schema to create a TypeDB schema into which we ingested the curated YAML corpus into TypeDB, making the full knowledge graph that we used to develop a type-enabled memory evaluation. Second, we implemented cross-database mapping from this externally-defined system into a restricted subset of Skillful-Alhazen’s notebook memory system.

A comparative agentic memory benchmark for structured knowledge. To measure the benefit concretely, we ran a controlled benchmark of thirteen questions spanning three analytically distinct categories: pathway aggregation (e.g. “how many diseases involve WNT/ β -catenin mechanisms?”), absence detection (e.g. “which diseases have HPO phenotypes but no genetic entries?”), and global ranking (e.g. “which five diseases have the most documented pathophysiology mechanisms?”). Each question was answered under two conditions: RAG over a 1,024-dimensional Qdrant vector index and structured TypeDB traversal. Ground truth was computed deterministically by scanning all YAML files; no LLM was involved in scoring.²

Structured TypeDB queries scored 0.75 / 0.68 / 0.77 on the three categories; RAG scored 0.00 / 0.00 / 0.03. The structured scores are conservative underestimates: ground truth is derived from YAML file scanning, while the structured condition queries TypeDB ‘exclusively, and small synchronization differences between the two representations (e.g. 798 diseases in TypeDB vs. 797 YAML files, regex PMID counting vs. typed evidence-item aggregation) reduce several exact-count scores from 1.0 to 0.5. On questions where TypeDB and YAML agree (Q2, Q4, Q7, Q8, Q11), the structured condition scores 1.0. The failure modes for RAG are architectural: aggregation requires corpus-wide counting that a similarity search cannot

¹<https://dismech.monarchinitiative.org/>

²<https://github.com/sciknow-io/alhazen-skill-dismech>

Table 1: Ontological memory structure across demonstration skills. All three skills share the core ICE hierarchy while extending it with domain-specific entity types and relationships.

	Job Hunt	Tech-Recon	DisMech
Entities	Companies, Positions	Systems, Criteria	Diseases, Mechanisms
Design Artifacts	Opportunity types Job postings, pages	Criteria as entities Docs, repos	YAML → TypeDB DisMech YAML files
Fragments	Requirements, quals	Feature evidence	Mechanisms, phenotypes
Notes	Fit analysis, gaps	Comparison notes	Evidence links, gaps
APIs	Greenhouse, LinkedIn	GitHub, Hugging-Face	Monarch, DisMech

verify for completeness; absence detection requires evidence of non-existence that no positive-assertion corpus can supply; global ranking requires ordering by a structural property (phenotype cardinality) rather than semantic relevance. These are the relational primitives (textscount, NOT EXISTS, ORDER BY) that a typed knowledge graph provides natively and retrieval-based methods cannot.

This benchmark deliberately isolates query types where structure matters; it does not capture the full space of curator questions. Retrieval-augmented generation remains the appropriate tool for narrative queries where semantic similarity over embedded text is exactly the right primitive. The Skillful Alhazen notebook memory model is designed to provide both. Each ingested entity is simultaneously represented as a typed node in the TypeDB graph and as embedded text fragments in the vector index: structured fields (gene identifiers, HPO codes, MONDO terms) are queryable as typed attributes, while free-text mechanism descriptions and PubMed abstracts are embedded for semantic retrieval. The two representations are complementary by design (structural queries answer questions of completeness and provenance; semantic queries answer questions of meaning and similarity) and a production curation assistant would route to whichever is appropriate for each question.

4.4 Demo Scenario

Attendees will interact with Skillful-Alhazen in a live terminal session. We will demonstrate: (1) ingesting a job posting URL and watching the knowledge graph grow in real time; (2) querying across accumulated knowledge (e.g., “What skill gaps appear across my top 5 prospects?”); (3) switching to the Tech-Recon skill and developing a personalized investigation around some technical question of interest (e.g., “Map the current state of agentic AI in pharmaceutical drug development, using Rentosertib (ISM001-055, Insilico Medicine) as a concrete case study of end-to-end AI-assisted pipeline execution. How do existing agentic memory systems (MemPalace, Mem0, Zep, Mastra, Supermemory ASMR) evaluate their own performance?”); and (4) inspecting the TypeDB graph to show provenance chains from notes through fragments to artifacts.

5 Related Work

Agent memory architectures. MemGPT [16], the precursor to the Letta platform [7], introduced an *LLM-as-operating-system* paradigm in which agents actively manage their own memory hierarchy. MemGPT’s memory tiers are unstructured text: the agent can persist and retrieve content, but imposes no schema on *what kinds of things* it is remembering. LangGraph [6] and similar orchestration frameworks provide state management and checkpointing for multi-step agent workflows, but treat persisted state as opaque key-value pairs rather than typed ontological structures. Skillful-Alhazen’s distinctive contribution is using an *editable schema-driven typed knowledge graph* as the memory substrate, where the schema itself provides structure and power for well-defined, logical reasoning algorithms.

Knowledge graph memory for agents. Zep [17] introduced Graphiti, a temporally-aware knowledge graph engine stored in Neo4j that synthesizes both unstructured conversational data and structured data while maintaining historical relationships via a bi-temporal model. On the LongMemEval benchmark, Zep improves accuracy of up to 18.5% while reducing retrieval latency by 90% [17]. Mem0 [10] takes a complementary approach, storing memories across three parallel backends: a vector store for semantic search, a key-value store for fast lookup, and a property graph for relational queries. On the LOCOMO benchmark, Mem0 improves accuracy by 26% over OpenAI’s built-in memory feature while reducing p95 latency by 91% [10]. Cognee [9] adopts an architecture that combines embeddings with knowledge graph extraction (subject–relation–object triplets), performing well on HotPotQA multi-hop reasoning benchmarks. Neo4j Labs’ neo4j-agent-memory [15] distinguishes three memory types: short-term memory for conversation history and session state, long-term memory for entities and relationships, and reasoning memory for decision traces, tool usage audits, and provenance.

Skillful-Alhazen’s ontologically-grounded approach using ICEs (artifact / fragment / note) that can be anchored to domain-specific entities and their relations provides semantic rigor and provenance to knowledge graph development from first principles. The use of TypeDB naturally enables access to graph algorithms through integration libraries [20].

Memory Benchmarks. The benchmarks currently used to evaluate memory systems (LoCoMo [8], LongMemEval [23], BEAM [18], and others) all test *conversational recall*. This is a necessary capability, but it is not sufficient for curation workflows that require structured data operations. The systems described above (Mem0 [10], Zep [17], Letta [7], Cognee [9], and Neo4j’s agent memory [15]) all report results on conversational benchmarks, yet none evaluates domain-specific curation tasks such as schema stability under evolving data, relational reasoning across typed entities, or longitudinal consistency of a growing knowledge base. These dimensions are simply not addressed by schema-free architectures.

Karpathy’s influential “LLM Wiki” proposal [5] argues that current RAG systems force “the LLM [to rediscover] knowledge from scratch on every question” and advocates a three-layer architecture of raw sources, a persistent LLM-maintained wiki with cross-references, and a governing schema; essentially a notebook whose

maintenance cost approaches zero because the agent handles the bookkeeping. This vision aligns closely with our ontological notebook, though we replace the flat-file wiki with a typed knowledge graph that can enforce constraints and support relational queries.

The gap is architectural. Our DisMech benchmark (Section 4.3) was designed to isolate the aggregation, absence-detection, and relational-ranking primitives, and the results confirm the structural hypothesis: RAG scores near zero on all three categories while typed graph traversal scores 0.68–0.77. The benchmarks that the field currently relies on would not detect this difference, because they do not test for it.

Retrieval over knowledge graphs. Microsoft’s GraphRAG [2] demonstrated that building a knowledge graph from a corpus substantially improves question-answering compared to baseline vector RAG. GraphRAG has since been integrated into Microsoft’s Discovery platform for agentic scientific research with performance gains (e.g., 63% reduction in ticket-resolution time at LinkedIn [11]). LightRAG [4] addresses GraphRAG’s computational cost, achieving comparable retrieval quality with reduction in token usage.

These GraphRAG approaches share an important limitation relative to Skillful-Alhazen: they treat the knowledge graph as a retrieval index over a fixed corpus, constructed once or in batch. Skillful-Alhazen’s ontological memory is instead a *live, agent-maintained notebook*: the schema defines what concepts the agent can reason about, and the agent continuously extends the graph as it encounters new material, generating typed extractions and attributed analysis notes rather than community summaries.

Ontologies in scientific knowledge management. The use of ontologies to structure biomedical knowledge has a long history predating LLM-based agents. Much of this work centered on the use of the Ontology Web Language (OWL) [21], with a strong reliance on the use of description logics as a primary reasoning methodology [13]. The DisMech project showcases a pragmatic approach to the use of existing ontologies [14], that also leverages other graph based methods. Matt Might’s precision medicine framework [12] also demonstrates how ontological discipline in knowledge curation enables mechanistic reasoning about rare disease etiology that purely statistical approaches cannot support. Although the framework is expressed in Natural Language for use by human experts, parts of it were also implemented in the MediKanren system [3] that uses a non-description-logic engine for graph-based inference.

6 Conclusion & Future Work

In this work, we demonstrate how the use of ontological notebook memory for agent systems can allow agents to work with humans to help perform sense making tasks like job searching, technology investigation, and disease mechanism curation. Semi-formal expert human understanding can be rendered into a computationally viable reasoning engine when frontier coding agents translate an expression of that expertise from natural language to TypeDB schema and Python code. This is a powerful pragmatic strategy for building knowledge-enabled tools. Our future work focuses on developing the virtuous cycle of using experience of the systems’ use curation work to improve schema + code design as an automated agentic loop.

Acknowledgments

Skillful-Alhazen is a fork of CZI’s Alhazen project, originally released under the MIT License. We thank Callum Hemsley, Joshua Send, and Krishnan Govindraj from the TypeDB team for their support and feedback for this work.

References

- [1] W. Ceusters. 2013. The Information Artifact Ontology. *Proceedings of the International Conference on Biomedical Ontology* (2013). IAO: <https://github.com/information-artifact-ontology/IAO>.
- [2] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. 2024. From Local to Global: A GraphRAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130* (2024).
- [3] A. Foksinska, C. M. Crowder, A. B. Crouse, J. Henrikson, W. E. Byrd, G. Rosenblatt, M. J. Patton, K. He, T. K. Tran-Nguyen, M. Zheng, S. A. Ramsey, N. Amin, J. Osborne, UAB Precision Medicine Institute, and M. Might. 2022. The precision medicine process for treating rare disease using the artificial intelligence tool mediKanren. *Front Artif Intell* (2022). <https://doi.org/10.3389/frai.2022.910216>
- [4] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang. 2024. LightRAG: Simple and Fast Retrieval-Augmented Generation. *arXiv preprint arXiv:2410.05779* (2024).
- [5] A. Karpathy. 2025. LLM Wiki. <https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f>. GitHub Gist proposing a three-layer LLM-maintained knowledge base architecture.
- [6] LangChain Team. 2024. LangGraph: Building Stateful, Multi-Actor Applications with LLMs. <https://github.com/langchain-ai/langgraph>. Accessed: 2026-03-01.
- [7] Letta Team. 2026. *Letta Research: Building AI that learns like humans*. <https://www.letta.com/research>
- [8] A. Maharana, D-H. Lee, S. Tuber, M. Choudhury, and M. Bansal. 2024. Lo-CoMo: Long-Context Conversational Memory Benchmark. *arXiv preprint arXiv:2402.17753* (2024).
- [9] V. Markovic et al. 2025. *Cognee: Memory Engine for AI Agents*. <https://cognee.ai>
- [10] Mem0 Team. 2025. *Mem0: Scalable Long-Term Memory for AI Agents*. <https://mem0.ai>
- [11] Microsoft Research. 2025. GraphRAG in Production: Enterprise Deployments. <https://www.microsoft.com/en-us/research/project/graphrag/>
- [12] M. Might. 2015. The Algorithm for Precision Medicine. <https://bertrand.might.net/articles/algorithm-for-precision-medicine/>
- [13] C. Mungall. 2023. Strange Loops: Journeys in Declarative Logic Programming in Genomics and Beyond. <https://zenodo.org/records/19598332>. doi:10.5281/zenodo.19598332 Presentation at Declarative Programming in Biology and Medicine workshop, ICFP 2023, Seattle, WA.
- [14] C. Mungall, H. Caufield, and Monarch Initiative Team. 2026. DisMech: A Disease Mechanism Knowledge Base. <https://github.com/monarch-initiative/dismech>. Accessed: 2026-04-24.
- [15] Neo4j Labs. 2026. *neo4j-agent-memory: Complete Memory for AI Agents*. <https://github.com/neo4j-labs/neo4j-agent-memory>
- [16] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez. 2024. MemGPT: Towards LLMs as Operating Systems. In *Proceedings of the International Conference on Learning Representations (ICLR)*. <https://doi.org/10.48550/arXiv.2310.08560>
- [17] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, and D. C. Rasmussen. 2025. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv preprint arXiv:2501.13956* (2025). <https://doi.org/10.48550/arXiv.2501.13956>
- [18] M. Tavakoli, A. Salemi, C. Ye, M. Abdalla, H. Zamani, and J. R. Mitchell. 2026. Beyond a Million Tokens: Benchmarking and Enhancing Long-Term Memory in LLMs. In *Proceedings of the International Conference on Learning Representations (ICLR)*. <https://arxiv.org/abs/2510.27246>
- [19] TypeDB Team. 2024. TypeDB: The Polymorphic Database. <https://typedb.com>. Accessed: 2026-03-01.
- [20] TypeDB Team. 2026. *typedb-graph-utils*. <https://github.com/typedb-osi/typedb-graph-utils>
- [21] W3C OWL Working Group. 2012. OWL 2 Web Ontology Language Document Overview. <https://www.w3.org/TR/owl2-overview/>. W3C Recommendation, 11 December 2012.
- [22] A. Walker. 2026. Vibe Querying with TypeDB Studio. <https://typedb.com/blog/vibe-querying-with-typedb-studio>. Accessed: 2026-03-12.
- [23] D. Wu, H. Wang, W. Peng, J. Jiang, and M. Yu. 2024. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *arXiv preprint arXiv:2410.10813* (2024).