Personal Agents and Conversational Memory

Sungeun An[1], Anna Lisa Gentile[1]

[1]IBM Research

Abstract

As AI agents become a primary interface for digital interaction, managing personal data acquired through conversations is increasingly important. This work focuses on *conversational memory*, the volatile knowledge that agents can accumulate about users over time, addressing challenges in structuring, updating, tracing provenance, and using such knowledge at query/inference time. We present preliminary work exploring structured representations to support the consolidation of conversational knowledge. As a proof of concept, we investigate the use of Knowledge Graphs to capture agentic conversational memory and propose a mechanism that combines imperative and generative computing to produce trustworthy and traceable responses. The approach enables transparent operation, traceable answers, and more deterministic responses grounded in explicitly stored information. A preliminary evaluation on a benchmark for long-term memory retention suggests the benefits of structured knowledge.

Keywords

conversational memory, knowledge graphs, personal agents, SPARQL, long-term memory

1. Introduction

AI agents are increasingly integrated into everyday digital experiences, and interacting with them is rapidly becoming the new normal. This shift brings renewed attention to the importance of personal data management, as AI agents continuously collect, process, and rely on user information to provide personalized assistance. Personal data in this context can have numerous facets, spanning from relatively stable user data stored as persistent or static memory, user data generated via the interaction with numerous applications and services, up to the new type of growing data constituted by the dynamic conversational data from the interactions with the personal agent(s), which we refer to as *conversational memory*. While the management of persistent personal data and the mechanisms governing agent access to such information are critical ongoing challenges [1], this work focuses specifically on the more volatile form of memory, the *conversational memory*, i.e., the information that agents acquire about users during conversations. Understanding and controlling how this conversational memory is formed and used is essential for ensuring responsible and trustworthy agent behavior.

The interaction between AI agents and users' personal data raises a number of challenging issues, particularly because personal information is not inherently available within the agent's underlying model. When users ask personal questions, the agent must rely on information acquired through prior interactions or external sources rather than pre-trained knowledge, making the management of such data both necessary and complex. In this work, we address the problem of how knowledge about a user can be created and maintained over extended interactions with an agent. Specifically, we investigate how this *conversational memory* can be structured in a meaningful and usable way, how the provenance of extracted information can be traced, and how the stored knowledge can be selectively updated as new information emerges. Additionally, we consider how agents can effectively access and utilize this evolving knowledge at query time, enabling coherent and personalized responses while maintaining transparency and control over personal data.

Our approach represents preliminary work toward a systematic framework for managing conversationally acquired personal knowledge in AI agents. We explore how suitable ontologies can support the representation of user-related information, enabling consistent organization and interpretation

ESWC'26: Trust, Autonomy and Accountability in PKG-Based Agentic AI (TAAPAAI) Workshop on May 10, 2026 in Dubrovnik, Croatia

sungeun.an@ibm.com (S. An); annalisa.gentile@ibm.com (A. L. Gentile)

of knowledge extracted from interactions. In particular, we investigate the benefits of structured knowledge representations, which can improve traceability, support controlled updates, and facilitate reliable access and reuse of personal information over time. Rather than treating personal knowledge management as an isolated component, we propose embedding it as a service within agentic paradigms, enabling agents to transparently store, maintain, and retrieve conversational knowledge as part of their normal operation. We implement our ideas within the Mellea framework[1], a library for writing generative programs with structured and maintainable prompts, that allows us to separate the mere generative portion of the approach (prompt-based extraction) from the imperative steps (e.g., format validation).

This work makes three main contributions. First, we present a proof of concept demonstrating how Knowledge Graphs can serve as agentic conversational memory, providing a structured, persistent representation of information acquired during user interactions. Second, we introduce a mechanism that combines imperative and generative computing to support the production of trustworthy and traceable answers, enabling agents to ground their responses in explicitly stored knowledge while preserving the flexibility of generative models. This approach allows for improved transparency and verifiability in the handling of personal information. Finally, we provide a preliminary evaluation of the proposed approach on a standard benchmark for long-term memory assessment, offering initial insights into the effectiveness of structured conversational memory for supporting consistent and reliable agent behavior over extended interactions.

The proposed approach offers several important advantages for the management of conversational personal knowledge in AI agents. First, it is designed to be transparent to the user, operating as a background service that integrates naturally into the interaction without requiring additional effort or explicit intervention. At the same time, the use of structured representations enables clear traceability of the information used to generate answers, allowing the origin and evolution of personal knowledge to be inspected and verified. By grounding responses in explicitly stored data, the approach also supports more deterministic behavior, reducing variability and improving consistency across interactions. More broadly, the adoption of structured knowledge provides a foundation for reliable organization, updating, and retrieval of user-related information, facilitating long-term memory management. Finally, the system is conceived as an embedded service within agentic paradigms, and it is implemented within a standard open-source framework, Mellea[1], demonstrating how structured personal knowledge management can be seamlessly integrated into agent-based architectures.

2. Related Work

As agentic AI transitions from assisted ("Copilot") to autonomous ("Autopilot") models [2], the ability to correctly capture, trace, and utilize long-term knowledge about users becomes increasingly critical [3]. Unlike short-term context, which LLMs can process within a single prompt, long-term memory requires storing and retrieving information over time, often across multiple sessions [4]. We organize existing approaches to long-term memory into three dominant paradigms: context-window extension, retrieval-augmented generation, and agentic memory systems.

Context-window extension. Recent LLMs have demonstrated the ability to process extended context lengths, ranging from 100K to over one million tokens [5, 6, 7]. These capabilities enable a simple yet effective form of memory: storing the entire conversation history directly within the context window. However, this approach faces two fundamental constraints. First, there is a hard limit, and once the context window is exceeded, earlier information must be discarded. Second, even within the allowed length, retrieval accuracy degrades as the volume of context grows, particularly for facts buried in the middle of long sequences. As a result, context-window extension alone cannot serve as a reliable foundation for persistent, long-term personal memory.

[1] https://github.com/generative-computing/mellea

Retrieval-augmented generation. Retrieval-augmented generation (RAG) addresses the context-length constraint by retrieving relevant information from an external store and feeding it to the LLM at inference time. Several approaches enhance retrieval with structural representations such as graphs and trees, including GraphRAG [8], RAPTOR [9], Zep [10], MemoRAG [11], Mem0 [12], and MemoryOS [13]. Wang et al. [14] further propose a Knowledge Graph-based RAG model (KG-RAG) that integrates structured knowledge graphs into traditional RAG architectures to improve semantic and inferential understanding.

Despite these advances, RAG-based methods share a recurring set of limitations. They struggle with *ambiguous queries* that do not map cleanly to retrievable passages, *multi-hop reasoning* that requires chaining evidence across multiple documents, and *long-range comprehension* that demands integrating knowledge distributed throughout an entire session. Because retrieval is typically limited to the top-k most relevant passages, critical information that is semantically distant from the query may never be surfaced. Moreover, these methods predominantly rely on vector-similarity search, which treats memory as an unstructured bag of text fragments and discards the relational structure between facts.

Agentic memory systems. Agentic memory systems attempt to overcome the single-pass limitation of RAG by introducing iterative, decision-driven frameworks. Rather than retrieving once and generating, these agents dynamically process queries, retrieve evidence, reflect on intermediate results, and iterate through multiple retrieval-reasoning cycles. MemGPT [15] draws inspiration from operating-system memory hierarchies, managing different memory tiers to provide the appearance of extended context through data movement between fast and slow memory. A-MEM [16] creates interconnected knowledge networks through dynamic indexing and linking, where new memories trigger updates to the contextual representations and attributes of existing memories, enabling continuous memory evolution. Self-RAG [17] and MemAgent [18] further contribute iterative self-reflection and multi-step retrieval strategies.

While the agentic approach is particularly effective for resolving ambiguous or multi-step queries, these methods remain fundamentally constrained by the same underlying limitation: they still depend on vector-similarity retrieval to access stored information. The iterative framework improves *how* retrieved evidence is processed but does not change *what* can be retrieved. When the relevant knowledge is structurally or temporally distant from the query, even multiple retrieval-reasoning cycles may fail to locate it. Additionally, ACE [19] introduces a framework that treats contexts as evolving playbooks refined through generation, reflection, and curation; however, it focuses on task-strategy accumulation rather than personal factual memory, and its unstructured playbook format lacks the formal semantics needed for precise, traceable retrieval of user-specific information.

Across all three paradigms, we identify three persistent limitations that motivate our work. First, **memory lacks formal structure** is typically stored as unstructured text or embeddings, which limits the representation of relational knowledge and constrains structured reasoning such as temporal queries or aggregation. Second, **stored information rarely includes provenance**, making it difficult to trace responses back to their originating interactions and reducing transparency. Third, **reliance on similarity-based retrieval introduces non-determinism**, as identical queries can yield different results due to model stochasticity, undermining reliability in personal agents.

We propose tackling these gaps with a knowledge graph-based conversational memory that blends structured knowledge with LLM capabilities. Unlike prior approaches, we: (i) complement unstructured text with ontologically grounded knowledge graphs for relational structure and provenance; (ii) foresee the use of deterministic SPARQL queries for reproducible reasoning chains; and (iii) limit LLMs to language understanding and query translation, delegating retrieval and reasoning to symbolic computation.

3. Method

We propose a three-stage pipeline for the effective management and usage of Personal Conversational Memory (PCM): (i) incremental knowledge graph construction from conversational interactions, (ii) natural language to SPARQL query translation, and (iii) deterministic retrieval over the personal knowledge graph. Note that while in this work we focus on the conversational memory, the graph KG nature of the memory accumulation prepares for the seamless integration and utilization of any other ontologically represented personal data. The pipeline decouples knowledge *encoding* (LLM-driven, performed once per interaction) from knowledge *retrieval* (SPARQL-driven, deterministic and repeatable), combining the flexibility of neural language understanding with the precision and transparency of symbolic reasoning.

3.1. Stage 1: Personal Knowledge Graph Construction

As a user interacts with a conversational agent over time, each user message is processed by an LLM to extract structured knowledge in the form of RDF triples. This extraction occurs incrementally: each new message extends the user's personal knowledge graph without requiring reprocessing of prior interactions.

Ontology. The knowledge graph is grounded in the *Personal Conversational Memory* (PCM) ontology, a multi-vocabulary schema that reuses nine established vocabularies and defines custom extensions only where no standard covers a concept:

- **Schema.org** (`schema`)[2] [20] models core entities (`Person`, `Vehicle`, `Place`, `Product`, `Organization`, `Event`) and actions. We use over 20 action types (e.g., `BuyAction`, `TravelAction`, `RepairAction`, `CommunicateAction`,
  `SearchAction`) with properties capturing agents, objects, temporal extent, location, and status. Recurrence patterns use `schema:Schedule` with ISO 8601 durations.
- **OntoBio**(`ob`)[3] [21] provides biographical modeling: family relationships with property chains (30+ properties such as `hasFather`, `hasSibling`, `hasSpouse`), habits (`Habit`, `DailyRoutine`, `FoodHabit`), activities (`PhysicalActivity`, `CreativeActivity`), food and meals, physical traits, education, employment, residence, and causality.
- **PROV-O** (`prov`)[4] [22] provides provenance tracking, linking every extracted triple to the source utterance via `prov:wasDerivedFrom`.
- **FOAF** (`foaf`)[5] and the **Relationship vocabulary** (`rel`)[6] represent people and social connections (`friendOf`, `colleagueOf`, `mentorOf`).
- **OpenCare** (`opencare`)[7] models health records, diseases, symptoms, and treatment procedures.
- **SKOS**[8] [23] provides concept hierarchies for categorization, while **BIO**[9] and **CV/ResumeRDF**[10] cover employment and education history.
- **PCM custom extensions** (`pcm`) fill gaps not covered in a simple way by any standard ontology: personal preferences (`Preference`), problem events (`ProblemEvent`), service events (`ServiceEvent`), pets (`Pet`), clothing (`Clothing`), plants (`Plant`), collections (`Collection`), and social media accounts (`SocialMediaAccount`).

[2]]https://schema.org/
[3]https://w3id.org/ontobio
[4]https://www.w3.org/TR/prov-o/
[5]https://xmlns.com/foaf/spec/
[6]https://vocab.org/relationship/
[7]https://w3id.org/opencare
[8]https://www.w3.org/TR/skos-reference/
[9]http://purl.org/vocab/bio/0.1/
[10]http://rdfs.org/resume-rdf/

LLM-Based Extraction. Each user message is independently processed by an LLM that receives the ontology schema as a system prompt and produces valid Turtle (TTL) RDF triples as output. The system prompt instructs the model to: (i) identify all entities, events, preferences, and relationships mentioned in the message; (ii) assign appropriate ontology types; (iii) resolve temporal expressions to `xsd:date` format; and (iv) link actions to their agents, objects, and locations. The LLM operates at temperature T=0.0 to ensure deterministic extraction. Extracted triples are parsed using `rdflib`[11] and merged into the user's persistent knowledge graph.

Provenance. Every extracted entity is linked to its source message through a unique provenance identifier encoding the conversation session and message position. The original message text is preserved (as a *sioc:Post*) and linked to the extracted entity (with *prov:wasDerivedFrom*), enabling full traceability from any entity back to the utterance from which it was extracted. This supports explainability: when the system retrieves an answer, the user can trace it to a specific conversation. Provenance is at the entity level, not the triple level. Although the latter would be more precise, it would bring the method outside the realm of RDF 1.1 tooling and add significant complexity for the LLM.

3.2. Stage 2: Natural Language to SPARQL Translation

When the user poses a question, the system translates it into a SPARQL query over their personal knowledge graph. An LLM receives three inputs:

1. The **ontology reference**, listing all available types and properties, enabling the model to construct well-formed graph patterns.
2. A **grounding sample** of entities from the user's knowledge graph (up to 30 entity URIs with their types and names), anchoring the generated query in the actual contents of the graph.
3. The user's **natural language question**.

The model generates a SPARQL `SELECT` query with appropriate graph patterns, `FILTER` clauses for text matching, `OPTIONAL` blocks for properties that may be absent, and temporal ordering where relevant. By grounding the query in a sample of the graph's entities, the model can generate queries that reference specific URIs and predicates present in the user's data, rather than relying solely on generic patterns.

3.3. Stage 3: Deterministic Retrieval

The generated SPARQL query is executed against the user's personal knowledge graph. Results are returned as structured variable bindings: entity names, dates, computed durations (via XSD date arithmetic), literal values, and provenance links. No LLM is involved in this stage: retrieval is purely deterministic graph pattern matching and traversal.

This separation is a key design choice. The LLM is used where its strengths lie, i.e., understanding natural language and mapping it to structured representations, while the actual retrieval is performed by a symbolic engine that is transparent, reproducible, and auditable. If a query returns incorrect results, one can inspect the generated SPARQL and the underlying triples (with provenance) to diagnose whether the issue lies in extraction, query generation, or the data itself.

The proposed architecture offers several advantages over purely neural approaches to long-term memory:

Transparency Every retrieved fact is traceable to a specific source utterance via provenance links, and the retrieval logic is expressed as an inspectable SPARQL query.

Incrementality The knowledge graph grows with each interaction without requiring reprocessing of prior data or retraining of any model.

[11] https://github.com/RDFLib/rdflib

Structured reasoning SPARQL supports temporal arithmetic, aggregation, and multi-hop graph traversal, which are operations that are difficult for systems relying on vector similarity alone.

Determinism Given the same knowledge graph and the same question, the retrieval stage always produces the same result.

The primary limitation is the reliance on LLM accuracy at triple extraction (Stage 1) and query generation (Stage 2). Extraction errors, such as missing triples or incorrect property assignments, propagate to retrieval failures. Similarly, the generated SPARQL may use predicates or patterns that do not match the graph's structure, yielding empty results even when the relevant knowledge exists. We discuss these failure modes in Section 4.3.

4. Preliminary Experiments

4.1. Dataset

We test our proposed approach using the LongMemEval benchmark [24]. Unlike earlier long-context benchmarks that primarily measure retrieval from static contexts [25, 26], LongMemEval is designed to assess five core long-term memory abilities of LLMs: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention. LongMemEval consists of 500 entries, each containing up to 53 multi-turn conversation sessions (on average ~491 messages and ~123 K tokens). *LongMemEval-S* (full haystack) includes all sessions (relevant + distractors), primarily testing retrieval, while *LongMemEval-Oracle* includes only ground-truth relevant sessions, focusing on reasoning and execution. ***Single-Session User*** **(SS-U)** (n=70) and ***Single-Session Assistant*** **(SS-A)** (n=56) test whether the model can recall information mentioned by the user or the assistant within a single session, respectively. ***Single-Session Preference*** **(SS-P)** (n=30) evaluates whether the model can leverage remembered user information to generate a personalized response. ***Multi-Session*** **(MS)** (n=133) requires aggregating user information across two or more sessions. ***Knowledge Update*** **(KU)** (n=78) assesses the ability to recognize changes in a user's life and update memory accordingly. ***Temporal Reasoning*** **(TR)** (n=133) tests reasoning over both metadata timestamps and explicit time references in conversation.

4.2. Experimental Setup

Baseline For the baseline setup, we use a zero-shot setting. We create an LLM prompt that contains the full conversation history, plus the question to be answered, without any external memory system (actual prompt in Figure 1). All conversation sessions are concatenated as alternating user/assistant message turns. A system prompt instructs the model to treat the provided conversation history as its own prior interactions and to use it when answering the final question. Evaluation follows an LLM-as-judge approach, where the judge model scores whether the generated answer matches the gold standard reference answer. For inference, we use Claude Sonnet 4.5 via the Anthropic Messages API [27], setting a greedy decoding (temperature 0.0) with a max output length of 1,000 tokens. We also use Claude Sonnet 4.5 as the judge model for all the experiments.

PCM-approach For our proposed method, we use Claude Sonnet 4.5 [27] for all the stages. In the current pipeline, the extraction at stage 1 is done with a prompt that asks the LLM to extract triples from user messages, specifying a few extraction rules, including the ontologies, classes, and properties to use. The generation of SPARQL queries at stage 2 is done with a prompt that asks the LLM to convert a natural language question into a SPARQL query, and provides the current types and properties available in the graph. Prompts are provided in Appendix A.

Baseline Prompt

```
The following messages are from your previous conversations with this user across
multiple sessions. Treat them as your own conversation history. Use the information
from these past sessions to answer the user's final question. If the answer is
contained in the conversation history, provide it directly.
[user]: ⟨session 1, turn 1⟩
[assistant]: ⟨session 1, turn 2⟩
...
[user]: ⟨session N, turn 1⟩
[assistant]: ⟨session N, turn 2⟩
...
{question}
```

Figure 1: Baseline prompt for the zero-shot execution.

4.3. Preliminary Results and Current Limitations

The baseline experiment (Table 1) shows that even when LLMs are capable of really long context handling, when given the whole conversations as a long context, the efficacy of answering questions is at an overall accuracy of 32.6%.

Table 1
Baseline results on LongMemEval.

Metric	LongMemEval-S	LongMemEval-Oracle
Single Session User	0.429	0.942
Single Session Assistant	0.696	1.000
Single Session Preference	0.033	0.466
Multi Session	0.188	0.812
Knowledge Update	0.526	0.846
Temporal Reasoning	0.203	0.541
Overall	**0.326**	**0.764**

Nevertheless, at the current state, our end-to-end approach cannot yet produce satisfying results, with the accuracy of our approach only between 11% and 15.6%. To disentangle *extraction failures* (the answer is absent from the knowledge graph) from *query failures* (the answer is present but the SPARQL query fails to retrieve it) in our approach, we perform a coverage analysis on each per-question KG. For every benchmark question, we search the KG for the expected answer string using a fuzzy matching strategy applied to all knowledge-bearing literals (excluding provenance, conversation structure, and ontology metadata): (i) *exact* normalized match, (ii) *substring* containment in either direction, (iii) *numeric* match on extracted numbers, and (iv) *keyword* overlap requiring at least 50% of content words to appear in a single literal. A question is marked as *covered* if any of i-iv applies. 66.2% of the questions are found in the extracted KGs from the oracle dataset, and 76.0% from the one extracted from the full haystack. We note that our coverage check is a *lower-bound estimate* of the information actually available in the KG, not a precise measure. This is because certain answer types require computation or inference that cannot be detected by string matching alone. For instance, questions expecting a *count* (e.g., "How many charity events did I participate in?") require aggregating multiple entities via `COUNT`, and the answer "4" will only be found if that number happens to appear verbatim in some literal, not because the KG contains four distinct event instances that a SPARQL query could enumerate. Similarly, *duration* answers (e.g., "7 days") require computing the difference between two dates stored as separate `schema:startDate` values. Conversely, *entity* answers (proper nouns, product names, place names) and *preference* answers ("Italian food", "Shell gas station") are well-suited to literal matching

and yield the most reliable coverage estimates. As a result, the overall coverage figures (66.2% for oracle, 76% for the full haystack) likely *underestimate* the true extractive recall of the KG, particularly for the 196 questions (39.2%) that expect numeric, duration, or temporal-ordering answers where the required information may be present as structured data but not as a matchable string. Despite KG coverage rates, the end-to-end correctness remains very low (15.6% on oracle and 11% on the full haystack) The primary bottleneck is the NL→SPARQL generation stage. Analysis of incorrect results reveals three recurring failure modes: (i) the LLM retrieves a *related but wrong entity* (e.g., returning a `pcm:Preference` node instead of comparing `schema:startDate` values when asked "which came first?"), (ii) the generated SPARQL *fails silently*, producing empty results (iii) temporal reasoning queries require *multi-step subqueries* (find a reference date, then filter by it) that the LLM frequently fails to compose correctly.

5. Conclusion & Future Work

We presented a three-stage pipeline for personal converational memory management based on knowledge graphs, and we performed a preliminary evaluation using a standard benchmark for long conversation management on AI agents.

Our experiments reveal both the promise and the limitations of structured knowledge graph approaches to long-term conversational memory management. The obvious clear advantage of a structured approach is that it offers transparency, as every retrieved answer can be traced to specific source utterances via provenance links, and the retrieval logic is expressed as an inspectable SPARQL query. This transparency is a significant advantage over black-box approaches, but comes with lower recall than systems that search over unstructured text. The bottlenecks of our current pipeline are extraction incompleteness and query imprecision. Analysis of failure cases shows that many expected answer values (specific prices, brand names, quantities) are never extracted into the knowledge graph in the first place. When the relevant information *is* present, SPARQL queries often match too broadly or fail on aggregation tasks (counting, summing). Nonetheless, the performance varies significantly by question type: Knowledge-update questions and preference questions are well-suited to structured retrieval, as they involve specific facts with clear entity-property mappings. Temporal reasoning and multi-session questions are harder, requiring date arithmetic and cross-session aggregation that stress both extraction and query generation. Questions about assistant-generated content perform worst, as our current pipeline only extracts knowledge from user messages.

We foresee a few directions for improvement. First, the per-message extraction strategy processes each utterance independently, losing cross-message context that would help resolve coreferences and aggregate quantities. Second, the current ontology does not capture all information types present in natural conversation, notably implicit knowledge, and fine-grained numerical facts. Third, the NL-to-SPARQL translation is currently rudimentary and relies on a sample of graph entities for grounding, which may miss relevant entities in large graphs. The most impactful improvement would be enriching the knowledge extraction stage to capture more complete property values rather than just entity-level facts. A hybrid approach that combines structured KG retrieval for factual questions with unstructured retrieval for open-ended questions could leverage the strengths of both paradigms. Additionally, allowing iterative query refinement when initial queries return empty results could improve query precision without sacrificing the transparency and determinism that distinguish this approach.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude Sonnet 4.5 in order to: LLM inference for knowledge extraction and baseline experiments as described in the paper. The author(s) reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. A. Bonatti, J. Domingue, A. L. Gentile, A. Harth, O. Hartig, A. Hogan, K. Hose, E. Jiménez-Ruiz, D. L. McGuinness, C. Sun, R. Verborgh, J. Wright, Towards Computer-Using Personal Agents, CoRR abs/2503.15515 (2025). URL: https://doi.org/10.48550/arXiv.2503.15515. doi:10.48550/ARXIV.2503.15515. arXiv:2503.15515.

[2] S. Hosseini, H. Seilani, The role of agentic AI in shaping a smart future: A systematic review, Array 26 (2025) 100399. URL: https://doi.org/10.1016/j.array.2025.100399. doi:10.1016/J.ARRAY.2025.100399.

[3] S. Murugesan, The Rise of Agentic AI: Implications, Concerns, and the Path Forward, IEEE Intell. Syst. 40 (2025) 8–14. URL: https://doi.org/10.1109/MIS.2025.3544940. doi:10.1109/MIS.2025.3544940.

[4] Z. Zhang, Q. Dai, X. Bo, C. Ma, R. Li, X. Chen, J. Zhu, Z. Dong, J. Wen, A Survey on the Memory Mechanism of Large Language Model-based Agents, ACM Trans. Inf. Syst. 43 (2025) 155:1–155:47. URL: https://doi.org/10.1145/3748302. doi:10.1145/3748302.

[5] OpenAI, GPT-4o System Card, CoRR abs/2410.21276 (2024). URL: https://doi.org/10.48550/arXiv.2410.21276. doi:10.48550/ARXIV.2410.21276. arXiv:2410.21276.

[6] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. P. Lillicrap, J. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser, I. Antonoglou, R. Anil, S. Borgeaud, A. M. Dai, K. Millican, et al., Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, CoRR abs/2403.05530 (2024). URL: https://doi.org/10.48550/arXiv.2403.05530. doi:10.48550/ARXIV.2403.05530. arXiv:2403.05530.

[7] Anthropic, Claude 4.6 Opus, https://www.anthropic.com/news/claude-opus-4-6, 2026. Announcement introducing Claude 4.6 Opus.

[8] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, J. Larson, From Local to Global: A Graph RAG Approach to Query-Focused Summarization, CoRR abs/2404.16130 (2024). URL: https://doi.org/10.48550/arXiv.2404.16130. doi:10.48550/ARXIV.2404.16130. arXiv:2404.16130.

[9] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, C. D. Manning, RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval, in: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, 2024. URL: https://openreview.net/forum?id=GN921JHCRw.

[10] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, D. Chalef, Zep: A Temporal Knowledge Graph Architecture for Agent Memory, CoRR abs/2501.13956 (2025). URL: https://doi.org/10.48550/arXiv.2501.13956. doi:10.48550/ARXIV.2501.13956. arXiv:2501.13956.

[11] H. Qian, Z. Liu, P. Zhang, K. Mao, D. Lian, Z. Dou, T. Huang, MemoRAG: Boosting Long Context Processing with Global Memory-Enhanced Retrieval Augmentation, in: G. Long, M. Blumestein, Y. Chang, L. Lewin-Eytan, Z. H. Huang, E. Yom-Tov (Eds.), Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025, ACM, 2025, pp. 2366–2377. URL: https://doi.org/10.1145/3696410.3714805. doi:10.1145/3696410.3714805.

[12] P. Chhikara, D. Khant, S. Aryan, T. Singh, D. Yadav, Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory, in: ECAI 2025, Frontiers in Artificial Intelligence and Applications, IOS Press, 2025, pp. 2993–3000. URL: https://doi.org/10.3233/FAIA251160. doi:10.3233/FAIA251160.

[13] J. Kang, M. Ji, Z. Zhao, T. Bai, Memory OS of AI Agent, in: C. Christodoulopoulos, T. Chakraborty, C. Rose, V. Peng (Eds.), Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025, Association for Computational Linguistics, 2025, pp. 25961–25970. URL: https://doi.org/10.18653/v1/2025.emnlp-main.1318. doi:10.18653/V1/2025.EMNLP-MAIN.1318.

[14] S. Wang, H. Yang, W. Liu, Research on the construction and application of retrieval enhanced generation (rag) model based on knowledge graph, Scientific Reports 15 (2025) 40425. URL: https://doi.org/10.1038/s41598-025-21222-z. doi:10.1038/s41598-025-21222-z.

[15] C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, J. E. Gonzalez, MemGPT: Towards LLMs as

Operating Systems, CoRR abs/2310.08560 (2023). URL: https://doi.org/10.48550/arXiv.2310.08560. doi:10.48550/ARXIV.2310.08560. arXiv:2310.08560.
[16] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, Y. Zhang, A-mem: Agentic memory for LLM agents, in: The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2026. URL: https://openreview.net/forum?id=FiM0M8gcct.
[17] A. Asai, Z. Wu, Y. Wang, A. Sil, H. Hajishirzi, Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection, in: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, 2024. URL: https://openreview.net/forum?id=hSyW5go0v8.
[18] H. Yu, T. Chen, J. Feng, J. Chen, W. Dai, Q. Yu, Y.-Q. Zhang, W.-Y. Ma, J. Liu, M. Wang, H. Zhou, Memagent: Reshaping long-context LLM with multi-conv RL-based memory agent, in: The Fourteenth International Conference on Learning Representations, 2026. URL: https://openreview.net/forum?id=k5nIOvYGCL.
[19] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li, U. Thakker, J. Zou, K. Olukotun, Agentic context engineering: Evolving contexts for self-improving language models, in: The Fourteenth International Conference on Learning Representations, 2026. URL: https://openreview.net/forum?id=eC4ygDs02R.
[20] R. V. Guha, D. Brickley, S. Macbeth, Schema.org: evolution of structured data on the web, Commun. ACM 59 (2016) 44–51. URL: https://doi.org/10.1145/2844544. doi:10.1145/2844544.
[21] B. Dutta, S. Arzoo, Towards a Biographical Ontology: The OntoBio Framework and Its Applications, Knowledge Organization 53 (2026). doi:10.31083/KO45190.
[22] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, J. Zhao, Prov-o: The prov ontology, 2013. URL: https://www.w3.org/TR/prov-o/.
[23] A. Miles, S. Bechhofer, SKOS simple knowledge organization system reference, 2009. URL: https://www.w3.org/TR/skos-reference/.
[24] D. Wu, H. Wang, W. Yu, Y. Zhang, K. Chang, D. Yu, LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory, in: The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025, OpenReview.net, 2025. URL: https://openreview.net/forum?id=pZiyCaVuti.
[25] A. Maharana, D. Lee, S. Tulyakov, M. Bansal, F. Barbieri, Y. Fang, Evaluating Very Long-Term Conversational Memory of LLM Agents, in: Proceedings of ACL 2024, Association for Computational Linguistics, 2024, pp. 13851–13870. URL: https://doi.org/10.18653/v1/2024.acl-long.747. doi:10.18653/V1/2024.ACL-LONG.747.
[26] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, J. Li, LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding, in: Proceedings of ACL 2024, Association for Computational Linguistics, 2024, pp. 3119–3137. URL: https://doi.org/10.18653/v1/2024.acl-long.172. doi:10.18653/V1/2024.ACL-LONG.172.
[27] Anthropic, Claude Sonnet 4.5 System Card, 2025. URL: https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-4-5-sonnet-system-card.pdf.

A. Appendix

Stage 1 System Prompt: LLM-Based Triple Extraction

You are a personal knowledge graph extraction engine.
Given a natural language message from a user named {agent_name}, extract structured knowledge as RDF triples in Turtle format. **Rules**

1. The primary agent is ALWAYS "{agent_name}". Use URI pcm:{agent_id} for them.
2. Output ONLY valid Turtle (TTL) — no explanation, no markdown fences.
3. Always start with these prefixes: `@prefix ...` *(omitted for brevity)*
4. All resource URIs MUST use the pcm: namespace.
5. Use the most specific type available. Dual-type with Schema.org actions and domain-specific classes where applicable (e.g. a repair is schema:RepairAction + pcm:ServiceEvent; a trip is schema:TravelAction + ob:Travel).
6. Extract dates from text. Partial dates like "3/22" or "March 15th" should be resolved to the current year. Format as "YYYY-MM-DD"^^xsd:date.
7. Every event/action MUST include schema:agent. If the message is in first person, the agent is pcm:{agent_id}.
8. If an event is caused by another, link them with ob:causalContext.
9. Extract ALL meaningful knowledge: events, objects, preferences, habits, plans, relationships, facts, family/social ties, health, travel, food, traits.
10. For completed actions use schema:CompletedActionStatus. For planned actions use schema:PotentialActionStatus.
11. Every pcm:ProblemEvent MUST include pcm:affects. Every pcm:ServiceEvent and schema:RepairAction MUST include schema:object.
12. NO ISOLATED NODES: Every entity MUST connect to at least one other entity via a relationship property.
13. LABELS: Every individual MUST have an rdfs:label with a short, human-readable name.
14. ENTITY UPDATES: When multiple messages refer to the same real-world entity, reuse the SAME URI. Do NOT create duplicates.
15. RELATIONSHIPS: Use rel: for social relationships (rel:friendOf, rel:colleagueOf) and ob: for family (ob:hasFather, ob:hasSibling, ob:hasSpouse).
16. HEALTH: Use opencare: for health records (opencare:HealthRecord, opencare:diagnosedWith, opencare:hasSymptom).
17. TRAVEL: Use ob:Travel with ob:travelFrom, ob:travelTo, ob:modeOfTransport. Also dual-type as schema:TravelAction.
18. HABITS & FREQUENCY: Use ob:Habit and subclasses. Attach schema:Schedule via schema:eventSchedule with schema:repeatFrequency (ISO 8601 duration).
19. PETS: Use pcm:Pet linked via pcm:hasPet with pcm:petSpecies.
20. SOCIAL MEDIA: Use pcm:SocialMediaAccount linked via pcm:hasSocialAccount.
21. CLOTHING: Use pcm:Clothing linked via pcm:wears.
22. PLANTS: Use pcm:Plant linked via pcm:grows with pcm:plantSpecies.
23. COLLECTIONS: Use pcm:Collection linked via pcm:hasCollection.

Available Types and Properties
[Ontology reference catalog from PCM ontology] (omitted for brevity)

Example

```
Input: "I just bought a silver Honda Civic on February 10th."

Output:

pcm:annalisa a schema:Person, foaf:Person ;
rdfs:label "Annalisa"@en ;
schema:name "Annalisa" .

pcm:silver-honda-civic a schema:Vehicle ;
rdfs:label "Silver Honda Civic"@en ;
schema:name "Silver Honda Civic" ;
schema:color "silver" ; schema:brand "Honda" ; schema:model "Civic" .

pcm:buy-honda-civic-2026-02-10 a schema:BuyAction ;
rdfs:label "Purchase of silver Honda Civic"@en ;
schema:name "Purchase of silver Honda Civic" ;
schema:startDate "2026-02-10"^^xsd:date ;
schema:agent pcm:annalisa ;
schema:object pcm:silver-honda-civic ;
schema:actionStatus schema:CompletedActionStatus .
```

Now extract knowledge from the following message(s). Output ONLY valid Turtle.

Figure 2: System prompt used in Stage 1 for LLM-based triple extraction. The model receives this prompt concatenated with each user message. The ontology reference catalog (omitted) lists all available types and properties from the PCM multi-vocabulary schema (Schema.org, OntoBio, FOAF, PROV-O, Relationship, OpenCare, SKOS, BIO, CV/ResumeRDF, and custom PCM extensions).

Stage 2 System Prompt: NL → SPARQL Generation

```
You are a SPARQL query generator for a personal knowledge graph.
Given a natural language question, generate a SPARQL SELECT query that retrieves the
SPECIFIC answer from the graph.
```

Rules

1. Output ONLY the SPARQL query — no explanation, no markdown fences.
2. Always use these prefixes: `schema:`, `ob:`, `foaf:`, `prov:`, `xsd:`, `pcm:`, `rdfs:`, `rel:`, `opencare:`, `skos:`, `bio:`, `cv:`, `sioc:`.
3. The primary person in the graph is `pcm:user` (a `schema:Person`).
4. Use `OPTIONAL` for properties that may not exist.
5. Return the SPECIFIC value that answers the question — not just the entity.
6. Order results by date when temporal information is relevant.
7. Limit results to 20 unless the question asks for all.

CRITICAL: Type-first querying strategy

- Map natural language concepts to ontology types — do NOT rely on keyword matching in `schema:name`:
  `"issue/problem"` → `schema:Event`
  `"bought/purchased"` → `schema:BuyAction`
  `"trip/travel"` → `ob:Travel / schema:TravelAction`
  `"habit/routine"` → `ob:Habit (ob:DailyRoutine, ob:FoodHabit)`
  `"health/doctor"` → `opencare:HealthRecord / opencare:Disease`
  `"friend/colleague"` → `rel:friendOf, rel:colleagueOf`
  `"family/parent/sibling"` → `ob:hasParent, ob:hasSibling, ob:hasSpouse`
  `"food/meal"` → `ob:Meal, ob:FoodItem, ob:FoodHabit`
  `"lives/resided"` → `ob:Residence, ob:residedIn`
- Use `rdfs:subClassOf*` to match all subtypes:
  `?entity rdf:type ?type . ?type rdfs:subClassOf* schema:Event .`
- Use subqueries for temporal references:
  `{{ SELECT (MIN(?d) AS ?refDate) WHERE {{ ... }} }}`
  `FILTER(?date > ?refDate)`
- Only fall back to `FILTER(CONTAINS(...))` on `schema:name` to distinguish entities of the *same* type.

Common mistakes to avoid

- Do NOT search for events by keyword when you can match by type.
- "how many X?" → use `COUNT(DISTINCT ?entity)`.
- "how many days/weeks?" → return the relevant dates; post-processing computes the difference.
- "who did I X with?" → return `schema:participant`, not `schema:agent`.
- For price questions, `SELECT schema:price`.
- For preferences, check `foaf:interest` or keyword matches in `schema:name`/`schema:description`.

Available Types and Properties

[Ontology reference catalog from PCM ontology v7] (omitted for brevity)

Examples

Example 1: `"What was the first issue I had with my new car after its first service?"`
Reasoning: "issue" → `pcm:ProblemEvent`, "service" → `pcm:ServiceEvent`. Find earliest service date, then first problem after it.

```
SELECT ?issue ?issueName ?issueDate WHERE {
{{ SELECT (MIN(?svcDate) AS ?earliestService) WHERE {
?svc rdf:type ?svcType . ?svcType rdfs:subClassOf* pcm:ServiceEvent .
?svc schema:startDate ?svcDate . }} }
?issue rdf:type ?issueType . ?issueType rdfs:subClassOf* pcm:ProblemEvent .
?issue schema:startDate ?issueDate ; schema:name ?issueName .
FILTER(?issueDate > ?earliestService)
} ORDER BY ?issueDate LIMIT 1
```

Example 2: `"How often do I attend yoga classes?"`
Reasoning: "yoga" → `ob:Habit` or `ob:PhysicalActivity`. "how often" → look for `schema:Schedule`.

```
SELECT ?name ?description ?freq WHERE {
{{ ?habit rdf:type ?hType . ?hType rdfs:subClassOf* ob:Habit .
?habit schema:name ?name . FILTER(CONTAINS(LCASE(str(?name)), "yoga"))
OPTIONAL { ?habit schema:eventSchedule ?sched .
?sched schema:repeatFrequency ?freq . }
} UNION {
?activity rdf:type ?aType . ?aType rdfs:subClassOf* ob:Activity .
?activity schema:name ?name . FILTER(CONTAINS(LCASE(str(?name)), "yoga"))
}} } LIMIT 5
```

```
Now generate a SPARQL query for the following question.
```

Figure 3: System prompt used in Stage 2 for NL→SPARQL generation. The model receives this prompt along with a graph summary (entity types and counts) and the user's natural language question. The type-first strategy directs the LLM to query by ontology type rather than string matching, using `rdfs:subClassOf*` for type hierarchy traversal.