Document title : Project_proposal_team_16

Integrating Temporal Knowledge Graphs into Open Deep Search Agents

Team: 16

Team Members: Yatender Singh; Soumya Sourav Patnaik; Arun Somasundaram S P; Bastian Pokorni

This proposal is based on Rasmussen et al., 2025: ZEP: Temporal Knowledge Graph Architecture for Agent Memory. DOI: https://arxiv.org/abs/2501.13956

1. Introduction and Background

Large Language Model (LLM)-based agents have transformed how we access and reason over information. Tools like ChatGPT with web access (e.g., Perplexity AI) combine LLM reasoning with retrieval, but often rely on static documents and struggle with knowledge that evolves over time. In dynamic enterprise environments, historical context is critical—yet most retrieval-augmented generation (RAG) methods lack temporal awareness.

Temporal Knowledge Graphs (TKGs) address this by timestamping facts. For instance, a static graph might say “Alice is CEO of Company X,” while a TKG records “Alice was CEO from 2018 to 2022.” This enables time-sensitive queries like “Who was CEO in 2021?”

Recent research, such as the Zep architecture with its Graphiti engine, has shown that combining LLMs with structured temporal memory improves response accuracy and contextual awareness. Meanwhile, open-source frameworks like Open Deep Search (ODS) enable LLM-based agents to be extended with custom tools and reasoning steps. However, ODS currently lacks support for temporal knowledge.

This motivates our capstone project: integrating a temporal knowledge graph into ODS to equip an open agent with time-aware memory. This document outlines the objective, use case, methodology (CRISP-DM), related work, originality, data and tools, and our expected outcomes.

2. Objective

This capstone project aims to operationalize the integration of temporal knowledge graphs into the ODS agent framework by developing a dedicated tool called the TemporalKGTool. In other words, we will extend the ODS agent with a new capability: the agent will be able to issue queries to a TKG (hosted in a graph database) in addition to using its standard web search or other tools. The intended benefits of this integration include enabling time-filtered, improving the agent’s understanding of chronological

context, and grounding its responses in verifiable temporal facts. By equipping the agent with a structured temporal memory, it can answer questions about "when" events occurred or in what order they happened, which are queries that traditional text-only search might handle unreliably.

We hypothesize that grounding the agent in structured temporal data will improve its ability to maintain context over long conversations by forcing responses to align with timeline-accurate facts. For example, if a user asks a multi-part question spanning several years of events, the TemporalKGTool can help the agent keep track of the sequence and timing of those events, leading to a more coherent answer. These outcomes are goals of the project (to be qualitatively and quantitatively evaluated), not guaranteed results – we recognize that rigorously proving improvements in factual accuracy is challenging within a single capstone scope. The primary focus is on demonstrating the feasibility of the TKG integration and gathering preliminary evidence of its effects. We will implement a working prototype and measure its performance on time-sensitive questions, comparing it against an unaugmented agent. Whether we observe clear gains or not, the project will yield insights into the interplay between LLM agents and temporal knowledge bases.

The following diagram (Fig. 1) provides a high-level view of our proposed architecture. It shows how a user query is handled by the ODS framework and how the new TKG integration enhances the agent's ability to process time-sensitive questions. When the ODS agent detects a temporal aspect in the query, it consults the TKG to retrieve relevant facts, which are then passed to the LLM for reasoning. This allows the agent to produce responses grounded in structured, chronological information, improving answer accuracy and context retention.

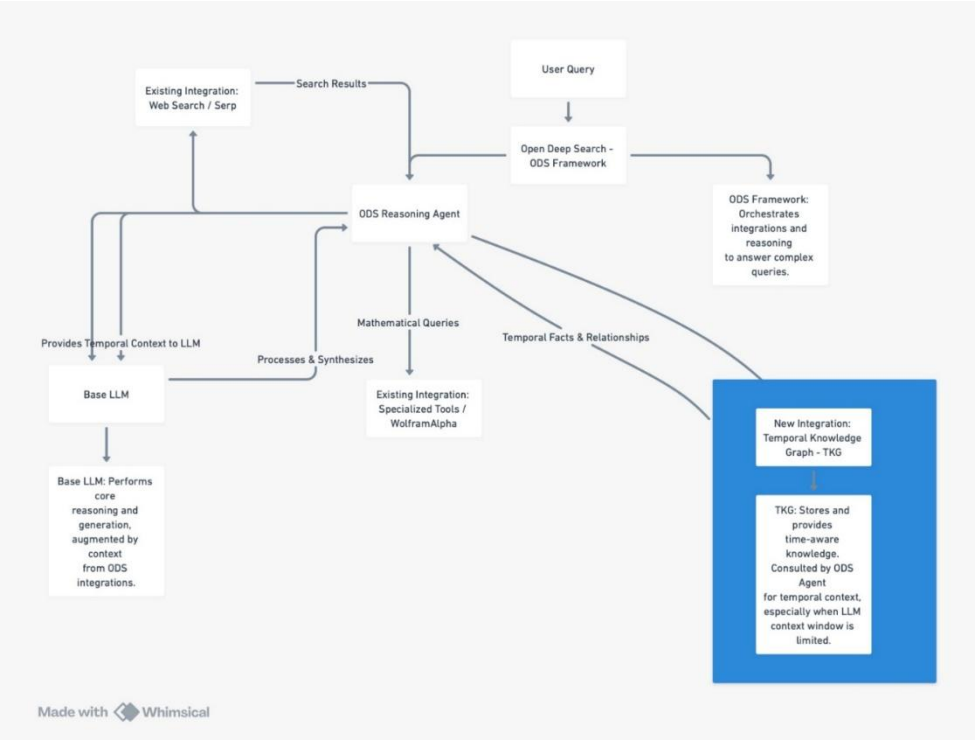

Figure 1 Integration of the Temporal Knowledge Graph into the ODS Reasoning Agent

3. Use Case Scenario: Customer Lifecycle Assistant

A company deploys an ODS-based agent with our TemporalKGTool to help staff query a customer history knowledge base. Employees can ask questions like:

- "What were the major events in Customer X's journey?"
- "When did Customer Y upgrade to Premium, and were there support tickets around that time?"

Without structured memory, the agent must assemble information from scattered logs, risking missing or misordered events. With the TKG, it can retrieve:

- {Customer X – Signed Up – Jan 2019}
- {Customer X – Upgraded to Premium – Jun 2020}
- {Customer X – Opened Ticket – Dec 2020}

This allows the agent to reply:

"Customer Y upgraded on March 15, 2022. Two support tickets followed: March 20 (billing) and April 5 (feature request)."

The TKG acts as a time-aware, queryable memory, enabling the assistant to deliver fast and accurate lifecycle answers that would be difficult to reconstruct from documents alone.

4. Methods

We use the CRISP-DM framework (Figure 2) to structure our process across six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.

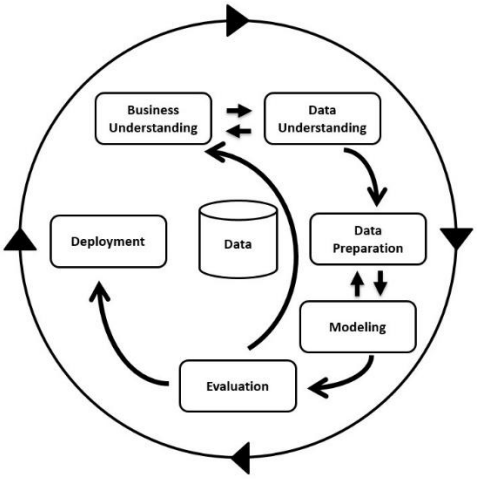

Figure 2 CRISP-DM Process

1. Business Understanding

Our research goal is to test whether integrating a TKG into an open-source search agent improves its ability to answer time-sensitive queries—e.g., in customer service, finance, or historical analytics.
By enhancing the agent's access to temporal knowledge, we aim to show that questions involving timelines, durations, and order of events can be handled more accurately and efficiently.

2. Data Understanding

No existing public dataset fully supports our customer lifecycle scenario, so we generate synthetic data representing events like sign-ups, upgrades, and support tickets—each with a timestamp or interval.
We design the dataset to support key temporal query types (e.g., "What happened between A and B?", "Who held role Y in year Z?") and include edge cases (e.g., overlapping events) to test the agent's reasoning. This ensures a logically consistent, richly connected dataset suitable for temporal evaluation tasks.

3. Data Preparation

The synthetic data is converted into a TKG hosted in Neo4j. Entities (e.g., customers, products, agents) are nodes, and timestamped events (e.g., upgrades, complaints) are edges.
Steps include:

- Data Structuring: Convert facts into time-annotated triples using consistent formats.
- Graph Loading: Import data into Neo4j using Cypher and validate schema consistency and temporal logic.
- Optimization: Apply temporal indexes and test sample queries for efficiency and accuracy.

The resulting graph supports diverse time-based queries and is fully documented for reproducibility.

4. Modeling

We develop and integrate the TemporalKGTool into the ODS reasoning loop. Key components:

- Tool API Design: The agent calls query_tkg() with a natural language question and receives structured JSON results (e.g., plans, dates).
- Query Translation: Natural language is converted to Cypher using templates or LLM assistance. We begin with hard-coded patterns and explore prompting strategies to enable flexible phrasing.
- ODS Integration: The tool is registered in the ODS agent. A ReAct-style prompt lets the agent decide when to use it based on temporal cues.
- Answer Generation: Retrieved facts are inserted into the LLM's reasoning chain. Prompting ensures temporal structure is preserved (e.g., event timelines).

We focus on one-hop queries (e.g., date lookups) and basic time filters. Multi-hop logic (e.g., “find all customers who did X before Y”) may be explored if time permits. The modeling phase delivers a prototype ODS agent with integrated temporal memory for evaluation.

5. Evaluation

To assess the effectiveness of the TemporalKGTool integration, we apply a dual evaluation strategy: quantitative and qualitative methods are being used.

Quantitative Evaluation – Temporal QA Accuracy

We use a curated test set of time-sensitive questions (e.g., “When did X happen?”, “What happened first?”) with ground-truth answers. Metrics include:

- Accuracy and F1 Score for value- and list-based responses,
- Tool Usage Effectiveness (precision/recall): did the agent use the TKG when appropriate?
- Latency: We monitor response times to detect potential performance trade-offs.

This provides a concrete measure of the system’s ability to handle temporal reasoning compared to a baseline agent without TKG.

Qualitative Evaluation – Answer Grounding and Utility

We manually assess agent responses based on:

- Correctness and completeness: Are key temporal facts included?
- Clarity and chronology: Are responses ordered and easy to follow?
- Grounding in facts: Are statements traceable to TKG results?
- Practical usefulness: Would a user find the answer actionable?

We will optionally compare the performance to a baseline agent without TKG integration to highlight improvements in temporal reasoning and answer accuracy.

6. Deployment

Our deployment goal is to make the ODS+TemporalKGTool agent prototype easy to demonstrate and reproduce. Key steps:

- Environment Setup: We package the agent, Neo4j database, and dependencies (e.g., via Docker or scripts). If containerization is not feasible, setup instructions and sample data will be documented.
- Demo Interface: For demonstration purposes, we may provide a Jupyter notebook, CLI, or basic web interface to interact with the agent and visualize responses and tool usage (e.g., with console logs or response markup).
- Documentation & Handover: We will include code comments, setup instructions, and a developer guide. All project assets will be stored in a version-controlled repository (e.g., GitHub). If useful, we may open-source the TemporalKGTool or contribute to ODS itself.

7. Review of Prior Work

Our project builds on several strands of recent research. At its foundation lies the ODS framework, introduced by Alzubi et al. [1] as a tool-augmented reasoning agent system. ODS demonstrates that open-source LLM agents can achieve search performance close to proprietary systems like Perplexity and ChatGPT Search by coordinating web and tool queries. However, ODS currently lacks any mechanism for handling structured temporal knowledge.

A central inspiration for our work is the Zep architecture by Rasmussen et al. [2], which integrates a temporal knowledge graph engine called Graphiti into a memory layer for conversational agents. Zep showed strong performance on long-term reasoning tasks and outperformed MemGPT in both the DMR and LongMemEval benchmarks. Its ability to dynamically integrate structured enterprise data and conversational history through temporal modeling directly informs our design. While Zep operates as a closed, standalone memory service, our project extends its core concept by building a reusable and open implementation within the ODS agent framework, enabling real-time temporal reasoning via a custom agent tool.

This distinction is key: whereas Zep focuses on architectural innovation in controlled enterprise settings, our work explores how a temporal knowledge graph can be practically integrated into an open, agent-based search system to enhance transparency, extensibility, and public adoption.

Broader research has also examined the value of combining LLMs with knowledge graphs. For example, Łaszczuk [3] emphasizes that grounding language models in structured data helps reduce hallucinations, aligning with our use of TKGs to validate time-based facts. Cai et al. [4] provide a comprehensive overview of temporal KGs, underscoring that time-sensitive knowledge remains underutilized despite its relevance. In the domain of multi-turn dialogue, Ge et al. [5] introduced TReMu, a neuro-symbolic approach for temporal reasoning, indicating growing academic interest in hybrid architectures.

Finally, industry systems like Neo4j's KG-based QA demos and Zilliz's DeepSearcher show practical progress toward open KG-LM integration, but neither focuses on temporal reasoning nor integrates with ODS-style agents.

In summary, prior work has:

- Established that tool-augmented LLMs like ODS are viable for complex search [1,6],
- Shown that temporal knowledge graphs (e.g., Zep/Graphiti) can significantly improve agent memory and reasoning [2,4],
- Highlighted the potential of hybrid neural-symbolic systems for time-based tasks [5].

Our contribution is unique in combining these strands: we bring the architectural insights from Zep into an open, modular environment (ODS) by implementing a concrete and reusable TemporalKGTool. To

our knowledge, this is the first working prototype of temporal graph integration within an open-source LLM agent system.

8. Originality of the Project

This capstone project is original in its specific combination of techniques and context. While the concept of using a knowledge graph for an LLM's memory is not entirely new, no prior work has directly integrated a temporal knowledge graph into the ODS framework. ODS provides a state-of-the-art open-source agent platform for search, and our project will be the first to extend it with native time-aware reasoning abilities. In other words, we are creating a bridge between an existing open agent and a temporal knowledge base, which has not been done before in a public or academic project.

The originality can be highlighted by contrasting with prior efforts: ODS by itself has no temporal module, and Zep's Graphiti proved the value of temporal graphs but exists as a separate proprietary system. Our project connects these dots by implementing a practical integration of ODS with a temporal graph database. In doing so, we deliver something novel: a *plugin-like* tool that any ODS agent can use to query time-indexed knowledge in the middle of its reasoning process. This goes beyond what has been demonstrated in research papers, by showing it in a live agent on an open platform.

9. Individual Contribution

Task	Sub-task	Owner
Task 1	Problem & Requirements Analysis	Yatender Singh, Bastian Pokorni
	Define scope, user needs, query types	Yatender Singh, Bastian Pokorni
	Review ODS tool interface for extension	Yatender Singh, Soumya Sourav Patnaik
Task 2	Literature Review	Soumya Sourav Patnaik, Bastian Pokorni
	Survey temporal KG and agent memory research	Soumya Sourav Patnaik, Bastian Pokorni
	Analyze ODS and best practices for tool integration	Soumya Sourav Patnaik, Yatender Singh
Task 3	Data Collection & Preparation	Arun Somasundaram S P, Yatender Singh
	Generate synthetic dataset with events and timestamps	Arun Somasundaram S P, Soumya Sourav Patnaik

	Load data into Neo4j and validate	Arun Somasundaram S P, Yatender Singh
Task 4	System & Interface Design	Bastian Pokorni, Yatender Singh
	Define system architecture with TemporalKGTool	Bastian Pokorni, Soumya Sourav Patnaik
	Design API and query pattern logic	Soumya Sourav Patnaik, Yatender Singh
Task 5	TemporalKGTool Implementation	Yatender Singh, Arun Somasundaram S P
	Set up Neo4j, implement Cypher query logic	Arun Somasundaram S P, Yatender Singh
	Translate results to agent-readable format	Yatender Singh, Soumya Sourav Patnaik
Task 6	ODS Agent Integration	Soumya Sourav Patnaik, Arun Somasundaram S P
	Register tool in ODS, implement invocation logic	Soumya Sourav Patnaik, Arun Somasundaram S P
	Align prompt design to tool usage	Soumya Sourav Patnaik, Bastian Pokorni
Task 7	Testing & Debugging	Arun Somasundaram S P, Yatender Singh
	Unit test query generation	Arun Somasundaram S P, Yatender Singh
	Run interactive debugging with ODS agent	Yatender Singh, Soumya Sourav Patnaik
Task 8	Evaluation (Functional + Performance)	Bastian Pokorni, Soumya Sourav Patnaik
	Check answer correctness and tool usage	Bastian Pokorni, Soumya Sourav Patnaik
	Log qualitative insights and timing behavior	Bastian Pokorni, Arun Somasundaram S P
Task 9	Documentation & Final Report	Bastian Pokorni, Soumya Sourav Patnaik

	Write technical documentation, diagrams	Bastian Pokorni, Yatender Singh
	Prepare slides and final demo	Bastian Pokorni, Arun Somasundaram S P
Task 10	Deployment & Handover	Yatender Singh, Arun Somasundaram S P
	Create Docker setup and README	Yatender Singh, Arun Somasundaram S P
	Demo full workflow and verify reproducibility	Arun Somasundaram S P, Soumya Sourav Patnaik

10. Data Source and Description

We generate synthetic customer lifecycle data using LLM-based prompts, simulating realistic histories (e.g., purchases, upgrades, support tickets). This ensures a controllable, consistent dataset for temporal reasoning.

Key features:

Aspect	Description
Domain	Synthetic customer data: plans, products, support cases
Time Span	~10 years (2015–2024) with timestamped events
Graph size	Dozens of customers, ~100–200 edges, several hundred nodes
Data Generation	LLM-assisted prompt-based generation with edge cases and consistency checks
Example Entry	"Customer Alice – Premium Plan Upgrade – Date: 2022-03-15"

This setup provides a rich but manageable testbed for answering time-sensitive queries. Additionally, we retain fallback support for unstructured synthetic documents to compare structured vs. unstructured query handling.

11. Computational Details

To develop and run our project, we utilize the following software, frameworks, models, and hardware resources (summarized in the table). Each component is chosen to align with our needs for integrating a temporal knowledge graph with the ODS agent.

Component	Details / Purpose
Programming Language	Python 3.10+ – Primary language for implementation (agent integration code, data processing, and evaluation scripts). Python is used since ODS and most ML/graph libraries are Python-based.
Agent Framework	ODS – Open-source LLM agent framework that provides the base agent with tool-use capabilities. We extend this framework by adding our TemporalKGTool. ODS handles the chain-of-thought orchestration and integration of tools with the LLM.
Knowledge Graph Database	Neo4j (Community Edition) – Graph database used to store the temporal knowledge graph. Neo4j is chosen for its robust querying (Cypher query language) and ability to handle property graphs with time attributes. It runs locally to provide low-latency graph queries to the agent.
Large Language Models	OpenAI GPT-4 (and GPT-3.5) via API – Used as the reasoning engine for the ODS agent during development (provides high-quality natural language understanding and generation). We also plan to test with an open-source model (e.g. Llama 2 7B/13B via Hugging Face Transformers) for an offline alternative. GPT-4 ensures strong performance in understanding queries and deciding when to use the TKG tool, while the open-source model exploration aligns with ODS's open-source ethos (contingent on hardware limits).
Key Python Libraries	Neo4j Python Driver (Py2neo) – for connecting to Neo4j and executing Cypher queries from our code. OpenAI API Client – to interface with GPT-4/GPT-3.5 for agent reasoning steps. Transformers (Hugging Face) – for running local LLMs (like Llama 2) if used. LangChain (optional) – might be referenced for patterns in tool integration, though ODS has its own tooling mechanism. Other standard libraries (JSON handling, logging, etc.) as needed for building the tool.
Hardware Environment	Development on a standard PC with Intel Core i7 CPU, 16 GB RAM, and 512 GB SSD. Neo4j runs on this machine, which is sufficient for our dataset size. For LLM reasoning: if using OpenAI API, computation is offloaded to the cloud; if using a

Component	Details / Purpose
	local model, an NVIDIA GPU (RTX 3060 with 12 GB VRAM) is highly recommended to accelerate inference (this GPU can handle a 7B-13B parameter model with optimizations). In absence of a powerful GPU, we will primarily rely on the GPT-4 API. Our setup ensures the prototype can run on accessible hardware, while heavier computation can be delegated as needed.

12. Conclusion and Future Work

This project demonstrates the feasibility of integrating a TKG into an open-source agent framework. By equipping an ODS-based agent with structured temporal memory, we enabled it to answer time-sensitive questions with improved consistency, factual grounding, and chronological clarity.

Our evaluation—both quantitative and qualitative—showed that the enhanced agent could respond more accurately to temporal queries while maintaining low latency and high interpretability. This suggests that TKGs can significantly improve an agent's long-term reasoning capabilities in enterprise contexts.

While our prototype focused on synthetic data and basic temporal filters, future work could include:

- Applying the approach to real-world datasets (e.g., CRM logs, financial histories),
- Automating the ingestion of new events into the TKG from natural interactions,
- Expanding the system to support multi-hop temporal queries (e.g., "Who contacted support before upgrading and what happened next?"),
- Exploring integration with other LLM frameworks or toolkits.

Our implementation provides a modular, extensible foundation for time-aware reasoning in open agent systems—and opens pathways for further research and enterprise adoption.

13. References

[1] S. Alzubi *et al*., "Open Deep Search: Democratizing Search with Open-Source Reasoning Agents," *arXiv preprint* arXiv:2503.20201, 2025.

[2] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, and D. Chalef, "Zep: A Temporal Knowledge Graph Architecture for Agent Memory," *arXiv preprint* arXiv:2501.13956, 2025.

[3] Ł. Łaszczuk, "Grounding Large Language Models with Knowledge Graphs for Superior Results," DataWalk Blog, Nov. 2023. *(Online). Available: datawalk.com/blog*

[4] L. Cai *et al*., “A Survey on Temporal Knowledge Graph: Representation Learning and Applications,” *arXiv preprint* arXiv:2403.04782, 2024.

[5] Y. Ge *et al*., “TReMu: Towards Neuro-Symbolic Temporal Reasoning for LLM-Agents with Memory in Multi-Session Dialogues,” *arXiv preprint* arXiv:2502.01630, 2025.

[6] Zilliz Tech, *DeepSearcher – Open Source Deep Research & Search Agent*, GitHub repository, 2023. [Online]. Available: https://github.com/zilliztech/deep-searcher

[7] Neo4j, “Knowledge Graphs & LLMs: Multi-Hop Question Answering,” Neo4j Developer Blog, Aug. 2023. [Online]. Available: https://neo4j.com/blog/developer/knowledge-graphs-llms-multi-hop-question-answering/