Memory Matters: The Need To Improve Long-Term Memory In LLM-Agents

Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, Dustin Dannenhauer

Abstract

In this paper, we provide a review of the current efforts to develop LLM agents, which are autonomous agents that leverage large language models. We examine the memory management approaches used in these agents. One crucial aspect of these agents is their long-term memory, which is often implemented using vector databases. We describe how vector databases are utilized to store and retrieve information in LLM agents. Moreover we highlight open problems, such as the separation of different types of memories and the management of memory over the agent's lifetime. Lastly, we propose several topics for future research to address these challenges and further enhance the capabilities of LLM agents, including the use of metadata in procedural and semantic memory and the integration of external knowledge sources with vector databases.

Introduction

Over the last year, there has been considerable interest in autonomous agents capable of leveraging large language models (LLMs). LLMs are evolving at a rapid pace, each with their own trade-off between size, training cost, and latent representation thanks to the availability of large corpuses of text and the development of the attention mechanism. Their large knowledge bases and general reasoning are bootstrapping the deployment of specialized chat agents, coding assistants (e.g., copilot (GitHub 2023)), and technical question-answering (e.g., AskYourPDF (ask 2023)). We refer to any agent system that relies on an LLM, in capacities beyond human-machine interfaces (such as task decomposition, planning, or task execution), as an LLM agent.

Pairing LLMs with autonomy requires memory systems. These memory systems ensure that interactions are coherent, contextual, and efficient and that the system can learn and adapt over time. Efforts like Auto-GPT (aut 2023) ask a human user to describe a goal in natural language and proceed to decompose the task into subtasks that are then executed via terminal commands and API calls. Information from earlier executions of subtasks is stored as memories to accomplish future subtasks. Short-term memory was implemented by prepending previous subtask prompts and results to subsequent subtask calls. Since LLMs have fixed-size context windows, complex tasks quickly generate more information than can fit within these limits. Long-term memory solutions currently implemented via vector databases have significant limitations. We review the current efforts to develop LLM agents, describe their use of vector databases for long-term memory, identify open problems in using vector databases as long-term memory, and propose topics for future work.

Recent LLM Agents

LLM agents are a recent but rapidly evolving trend. These agents independently perform tasks, create new ones, prioritize them, and adapt to changing requirements to achieve a specific objective. LLMs, such as GPT-4, serve as the core controller for these agents. Most LLM agents utilize a framework containing at least the following components, all implemented as LLM API calls with different prompts: planning, task execution, memory management, and tool use.

Planning decomposes complex high-level tasks into smaller, simpler sub-tasks recursively until sub-tasks are executable. Once the original high-level task is fully decomposed into executable sub-tasks, task execution proceeds sequentially. Each task attempt is checked for success and is re-tried until it succeeds, enabling limited adaptation capabilities. Memory management addresses short-term and long-term information storage outside of LLM's context window, allowing agents to retain context-specific knowledge and recall past experiences. API calls to external tools enable LLM agents to access real-time information, execute code, and leverage proprietary databases.

A formal example of such a framework is MRKL (Modular Reasoning, Knowledge, and Language) (Karpas et al. 2022), a neuro-symbolic architecture for autonomous agents that combines the power of LLMs with expert modules. The LLM acts as a central controller, routing inquiries to the most suitable expert module based on its understanding of the task and the capabilities of the modules. These modules can be neural or symbolic, providing the agent with diverse tools for handling different tasks. MRKL enables agents to leverage both neural and symbolic reasoning, allowing them to tackle complex problems effectively.

Several examples of autonomous agents with LLMs have garnered noteworthy attention. One of the most well-known examples is Auto-GPT (aut 2023). Auto-GPT leverages GPT-4 as its controller and operates with minimal user input. It employs a self-assigned goal-oriented approach to break down objectives into sub-tasks and uses various tools to achieve them. Noteworthy capabilities include writing, debugging, testing, and editing code. However, challenges persist, such as confabulatory tendencies and difficulty in task focus and contextual understanding.

Other examples are BabyAGI (Nakajima 2023) and SmartGPT (Corman 2023), which demonstrate AI-powered task management systems. BabyAGI utilizes LLMs like OpenAI and vector databases like Chroma or Weaviate to generate, prioritize, and execute tasks. By leveraging LLMs' natural language processing capabilities, BabyAGI creates new tasks based on an objective and the results of previous tasks, enabling it to adapt and evolve based on changing requirements. SmartGPT, on the other hand, focuses on enabling LLMs to complete complex tasks without user input. It achieves this by breaking tasks into minor problems, collecting information from external sources, and leveraging the internet.

GPT-Engineer (Osika 2023) is an open-platform project that aims to make it easy for developers to build and extend their own autonomous agents. By generating an entire codebase based on a user-defined prompt, GPT-Engineer enables developers to customize the behavior of their agents according to their specific needs. This project highlights the potential of LLMs in code generation and the creation of personalized code-generation toolboxes.

The agents mentioned above are considered generalist agents to be used to solve open ended problems. ChemCrow (Bran et al. 2023) is an example of a domain-specific LLM agent. It utilizes LLMs augmented with expert-designed tools to accomplish organic synthesis, drug discovery, and materials design tasks. By combining LLMs with specialized tools, ChemCrow demonstrates the potential of autonomous agents in specific domains and showcases how LLMs can be integrated with existing expert knowledge.

Generative Agents (Park et al. 2023) takes a unique approach by creating a sandbox environment where multiple virtual characters, controlled by LLM-powered agents, interact with each other. Inspired by The Sims, this experiment combines LLMs with memory, planning, and reflection mechanisms to simulate believable human behavior. Generative Agents highlights the potential of autonomous agents in interactive applications and the integration of LLMs in creating realistic virtual environments.

Long-Term Memory via Vector Databases

The common model of cognition (Laird, Lebiere, and Rosenbloom 2017) distinguishes between long-term memory and working memory and is depicted in Figure 1. Working memory represents the current situation, including the state of current reasoning and abstractions of the ongoing physical situation of the agent's embodiment. Long-term memory (LTM) systems alter an agent's behavior by changing the contents of working memory, which can lead to changes to the abstract representations that control embodiment. Long-term memory is subdivided into procedural, semantic, and episodic memory. These memory systems must operate incrementally and in real time, maintaining reactivity to environmental changes. We propose that LLM agents can potentially align with the functionality described in the common model of cognition, then focus on potential implementations of long-term memory for these agents.

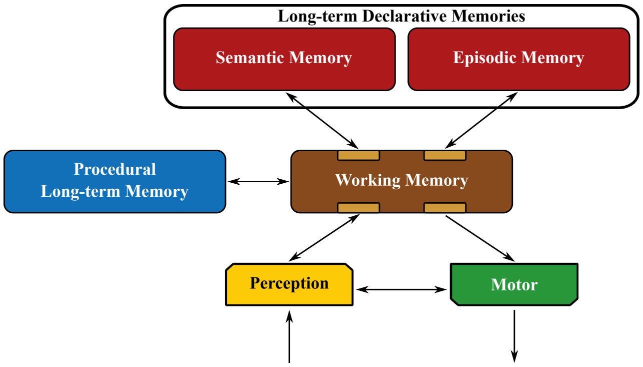

Figure 1: The common model of cognition, altered to distinguish between episodic and semantic memory. (Jones and Laird 2023)

An LLM context window can abstractly function similarly to working memory. LLM agents can use a description of the current situation and retrieve contents of a long-term memory store to populate a prompt template. Then, a prompt template program and LLM collectively function similarly to procedural memory by creating prompts and responses to alter the contents of a context window over time. Storage of these contents over time (especially to timescales beyond what a context window can support) to an external database can provide episodic memory storage. Additionally, derived facts (either mined from episodic memory or reasoned) can be stored separately as semantic memory. We now describe how vector databases in implemented LLM agents support long-term memory capabilities, followed by risks and limitations.

Vector Databases

Vector embeddings have become a popular representation for memories because transformer neural network LLMs use vector embeddings natively. Input data as raw text is run through an encoder network that embeds the data in a fixed-dimensional space. These embeddings represent data such that distance-based measures can be used to retrieve similar data.

A vector database is designed to support data storage and retrieval in an embedding representation rather than more structured, scalar data in traditional relational databases. Queries for vector databases rely on similarity matching to return the top-k matches, using algorithms such as cosine similarity, euclidean distance, or dot product. Metadata can be added as an additional filtering step[1] that offers conditional logic filters on top of similarity measure-based retrieval (e.g., the *year* metadata field must be *greater than or equal to* '2020').

[1] https://docs.pinecone.io/docs/metadata-filtering

One instance of a vector database being used to augment an LLM is Teenage-AGI (tee 2023). In this implementation, each ”thought” and memory that the LLM outputs or perceives is stored in a respective Pinecone vector DB. Whenever a new thought or perception is processed, both databases are queried for the top-k matches, and these are added to the context for the LLM prompt (aka the LLM agent's working memory). This allows the LLM to attend to previous experiences related to the current thought or memory, which often improves the accuracy and meaningfulness of the output as seen in research on few-shot prompting.

Another instance of an LLM agent utilizing a vector database is Voyager, an “LLM-powered embodied lifelong learning agent” (Wang et al. 2023), which acts in Minecraft. Descriptions of skills (computer programs that interact with the Minecraft engine) that an agent has learned are retained in a vector database and a GPT model is used to generate plans. Plans and a description of the environment are embedded and the closest skills in the db are retrieved. The skills are refined through iterative prompting until they are correct, and plans are generated by chain-of-thought prompting (Zhang et al. 2022) with the agent's current state as context.

The two previous approaches to enhancing LLM agent performance with vectors databases add different types of memory to the agent. In the case of Teenage-AGI, the database contains declarative and episodic memories (embeddings of previous experiences and thoughts) while in the case of Voyager, the database contains procedural memories (embeddings of descriptions of skills).

In both cases, the database quickly retrieves relevant material to the current state and task even though what the information is used for is different (informing context vs deciding what skills to use). These approaches are similar to Kahneman's System 1 (Kahneman 2011) – the fast, associative, and intuitive part of human consciousness. However, without a proper analog to the logical and slower System 2, underlying issues with LLM agents such as hallucinations and irrational planning will continue to plague both the data stored in long-term memory and the executive functioning and planning of the agent.

Fundamental Problems

Separation between types of memories: Poor retrieval may occur if all types of long-term memory (procedural, episodic, and semantic) are stored in the same vector database. For example, an agent could have the knowledge that the Earth is a globe stored in its semantic memory but have recently concluded a debate on the flatness of the Earth. The episodic memories of an argument that the Earth is flat are directly inconsistent with the agent's semantic memory. Therefore memories must be segregated based on type.

Infinite subtask loop: Current LLM-agents can become stuck trying to revisit the same sub-task over and over. This usually happens when a subtask fails and the LLM suggests one or more alternative solutions that also fail. With each failure, the LLM keeps suggesting from a fixed set of solutions without trying something new. Episodic memory offers a way to detect if failures are reoccuring, since each failure is stored in the long-term memory. Simpler solutions, such as dev-GPT [2] count the number of failures per subtask strategy and try a new strategy after 10 failures.

Lifelong Memory Management: LLM-agents will accumulate many memories as they solve tasks over their lifetime. Vector databases offer performance features such as indexing, scaling, and metadata-filtered queries, but don't address issues such as how the agent's information flow is segmented or how memories are organized under metadata categories. Continuously stored memories will result in a more complete agent history at the cost of space and time for retrieval. Solutions to forgetting such as (Martínez-Plumed et al. 2015) may be required to maintain performance over many tasks.

Conclusions and Future Work

We review recent efforts to develop LLM agents: agents that use LLMs in capacities beyond human-machine interfaces such as problem solving. Agents rely on information over various time horizons to accomplish complex, multi-step tasks. We discuss how short-term memory and long-term memory are currently being used in LLM-agents by referencing the common model of cognition (Laird, Lebiere, and Rosenbloom 2017), and elaborate on the risks with using vector-databases for long-term memory.

In future work, exploring the development of learning mechanisms for metadata in both procedural and semantic memory would be valuable. This would involve investigating how the agent can autonomously learn and update metadata attributes based on experiences and interactions with the environment. For procedural memory, the agent could learn metadata attributes such as the success rates, reliability scores, or preferences of different actions, allowing it to prioritize and adapt its decision-making process. Similarly, for semantic memory, the agent could learn metadata attributes that capture the occurrence frequencies, probabilistic information, or temporal context of concepts and relationships, enabling more efficient retrieval and utilization of memories.

It would also be beneficial to investigate the integration of metadata attributes with external knowledge sources, such as ontologies or knowledge graphs. An agent could leverage these sources' rich semantic relationships and structured information by linking the LTM with external knowledge bases. This integration could enable more comprehensive and accurate metadata attributes, as well as facilitate the reasoning and inference capabilities of the LLM. For instance, an agent could use the metadata attributes to perform semantic searches or traverse the knowledge graph to retrieve related concepts and relationships. This would enhance an agent's ability to contextualize and interpret memories within a broader knowledge framework, leading to more robust decision-making and problem-solving capabilities. Recent approaches such as OntoGPT and graphGPT look to achieve semantic parsing and node generation through the use of LLMs to create the Knowledge base/ontology from

[2] https://github.com/jina-ai/dev-gpt

which an episodic or declarative LTM grounding can take place.

References

2023. AskYourPDF. *https://askyourpdf.com/*.

2023. Auto-GPT: An Autonomous GPT-4 Experiment. *https://github.com/Significant-Gravitas/Auto-GPT*.

2023. Teenage-AGI. *https://github.com/seanpixel/Teenage-AGI*.

Bran, A. M.; Cox, S.; White, A. D.; and Schwaller, P. 2023. ChemCrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*.

Corman. 2023. SmartGPT. *https://github.com/Cormanz/smartgpt*.

GitHub. 2023. GitHub Copilot. *https://github.com/features/copilot*.

Jones, S. J.; and Laird, J. E. 2023. A cognitive architecture theory of anticipatory thinking. *AI Magazine*.

Kahneman, D. 2011. *Thinking, Fast and Slow.* Macmillan. ISBN 978-1-4299-6935-2.

Karpas, E.; Abend, O.; Belinkov, Y.; Lenz, B.; Lieber, O.; Ratner, N.; Shoham, Y.; Bata, H.; Levine, Y.; Leyton-Brown, K.; et al. 2022. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Laird, J. E.; Lebiere, C.; and Rosenbloom, P. S. 2017. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *Ai Magazine*, 38(4): 13–26.

Martínez-Plumed, F.; Ferri, C.; Hernández-Orallo, J.; and Ramirez-Quintana, M. J. 2015. Knowledge acquisition with forgetting: an incremental and developmental setting. *Adaptive Behavior*, 23(5): 283–299.

Nakajima, Y. 2023. BabyAGI. *https://github.com/yoheinakajima/babyagi*.

Osika, A. 2023. GPT Engineer. *https://github.com/AntonOsika/gpt-engineer*.

Park, J. S.; O'Brien, J. C.; Cai, C. J.; Morris, M. R.; Liang, P.; and Bernstein, M. S. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Zhang, Z.; Zhang, A.; Li, M.; and Smola, A. 2022. Automatic Chain of Thought Prompting in Large Language Models. In *The Eleventh International Conference on Learning Representations*.