LLM-Powered Conversational Multi-Agent Cognitive System for Collaborative Task Solving

Eryck Silva[1], **Frances A. Santos**[1], **Pedro Thompson**[2], **Julio C. dos Reis**[1]

[1]Universidade Estadual de Campinas (UNICAMP)

[2]Petróleo Brasileiro S.A. (PETROBRAS)

eryck@unicamp.br, {frances.santos,jreis}@ic.unicamp.br,
pedrothompson@petrobras.com.br

Abstract. *Multi-agent system architectures powered by Large Language Models (LLMs) present a promising solution for tackling complex problems through autonomous collaboration. However, key challenges persist, including the need for seamless communication and coordination among agents, computational efficiency, behavioral consistency, and effective memory management. This study proposes a novel multi-agent system architecture in which an orchestrator agent dynamically distributes tasks among specialized LLM-powered agents. Our approach enables efficient task execution and intelligent decision support. We present a case application via an early implementation of our solution to identify factors influencing system effectiveness in terms of the agents' memory results.*

1. Introduction

Multi-agent system architectures based on Large Language Models (LLMs) have emerged as a powerful approach for solving complex problems by enabling multiple autonomous agents to operate within a shared environment. These LLM-based agents communicate, collaborate, and coordinate their actions to achieve specific objectives, making such systems particularly advantageous in dynamic and complex domains. Distributing tasks among specialized agents enhances scalability, parallel processing, and adaptability, potentially facilitating decision-making and problem-solving in large-scale systems. LLM-based multi-agent architectures have led to significant advancements across various applications, including software development assistants [Qian et al. 2024] and multi-objective tasks instantiated in various embodied environments [Zhang et al. 2023], underscoring the relevance and potential of such architectures.

LLM-based multi-agent systems still face several key challenges that impact their effectiveness and scalability. One of the main challenges is ensuring seamless communication and coordination among agents, as these systems often involve complex interactions across various tasks and domains. This requires the development of robust mechanisms for synchronization and negotiation, as agents must be able to understand and respond to each other's actions in real-time.

Another challenge is ensuring consistency and coherence in agents' behavior, especially when dealing with dynamic and uncertain environments. Managing memory and context across interactions remains a significant hurdle. This is critical for sustaining meaningful agent behaviors over time, as traditional memory structures may not be well-suited to handling the complexity of episodic memories in multi-agent settings.

Furthermore, integrating LLMs raises ethical concerns and questions about computational efficiency, as these models are resource-intensive and may struggle to maintain high performance when scaled to large numbers of agents. These challenges highlight the need for novel design, coordination, and memory management approaches in LLM-based multi-agent systems.

This article proposes a novel multi-agent system architecture where LLM-powered cognitive agents interact to analyze data effectively. Each agent within the system is designed with specialized roles, such as communication, coordination, and code generation, allowing for task distribution and cooperative problem-solving. Unlike monolithic AI systems, our proposal promotes modularity and scalability, ensuring that individual agents can evolve independently while collaborating with other agents.

The proposed architecture consists of multiple autonomous agents equipped with cognitive modules collaborating to extract insights, enhance operational efficiency, and reduce the cognitive load on human operators. Each agent comprises five main modules: Planning, Hybrid Memory, Execution, Perception, and Reasoning. Our solution considers short-term and long-term types of memory, considering memories for different purposes in the agent, like working memory, semantic memory, and procedural memory. Our solution also employs a shared memory space among all agents playing the role of a blackboard context for the multi-agent system.

We conducted a case application to demonstrate the feasibility of our solution, applying it to the proposed architecture. Our application implemented three Task-driven Agents (Moderator, Financial, and Code Generator) to operate together. The scenario involved fetching and analyzing stock market prices using an API as a tool, along with data manipulation and visualization libraries. The system was built using LangGraph, and memory management was conducted with OpenSearch. Our approach demonstrates the system's feasibility by the promising results obtained while only developing key components (conversation coordination, shared memory, and task-driven agentic approaches).

The remainder of this article is organized as follows. Section 2 reviews related studies. Section 3 presents our proposed architecture. Section 4 presents a case application to demonstrate the benefits of our proposed solution. Section 5 discusses our approach, including open research challenges. Section 6 concludes our study.

2. Related Work

Tavares and Ralha [Tavares and Ralha 2024] developed a multi-agent system for multi-robot environments that require physical interaction and observation, thereby adding complexity. They implemented an adaptive automated planning strategy that reacts to unexpected changes, such as blocked paths. A central agent, the "Coordinator" is responsible for generating, updating, and distributing plans to other agents in response to such events. Using a space resource gathering simulation as a benchmark, the authors found that while replanning may increase execution time, it improves the overall success of the mission. Similarly, Cardoso *et al.* [Cardoso et al. 2021] applied automated planning strategies based on the Belief-Desire-Intention (BDI) model [Georgeff et al. 1999] in a multi-agent setting to improve simulated grounded tasks. They used PDDL [Aeronautiques et al. 1998] with a multi-agent programming language to implement the system, which operated in a block-structure composition task within a grid.

In Wang *et al.* [Wang et al. 2024a], a systematic review explores LLM-based autonomous agents, where Large Language Models (LLMs) act as central controllers to achieve human-like decision-making capabilities. Their study introduced a unified framework comprising four interconnected modules: Profile, Memory, Planning, and Action. The Profile module defines the agent's role (e.g., coder, teacher, domain expert) using strategies such as handcrafting, LLM-generation, or dataset alignment, influencing subsequent modules. The Memory module enables agents to accumulate experiences and evolve, integrating short-term and long-term memory systems with storage formats like natural language, embeddings, databases, and structured lists.

Similarly, Sumers *et al.* [Sumers et al. 2024] introduced the Cognitive Architectures for Language Agents (CoALA) framework, which structures language agents using principles from cognitive architectures and production systems. CoALA defines agents along three key dimensions: memory, action space, and decision-making. Memory is categorized into working, episodic, semantic, and procedural, while the action space encompasses internal reasoning, retrieval, learning, and external grounding.

Both frameworks contribute valuable theoretical foundations for advancing language agents toward more adaptive, memory-efficient, and decision-aware architectures. While Wang *et al.* [Wang et al. 2024a] focus on modular interactions between memory, planning, and actions, Sumers *et al.* [Sumers et al. 2024] emphasized cognitive principles and structured retrieval mechanisms. Such findings have motivated a new generation of language agent architectures that aims for more sophisticated memory integration, decision-making, and standardized evaluation methodologies.

Also inspired by cognitive architectures, Zhang *et al.* [Zhang et al. 2023] proposed the *CoELA* (Cooperative Embodied Language Agent) framework, designed for embodied agents and integrating LLMs for reasoning and language generation. Qian *et al.* [Qian et al. 2024] proposed *ChatDev*, a chat-powered software development framework that employs multiple LLM-driven agents with predefined roles to collaboratively complete software engineering tasks. Their framework structures communication using a chat chain, which organizes interactions into sequential phases: design, coding, and testing, allowing agents to refine solutions iteratively.

Such examples (*ChatDev* and *CoELA*) provide a clear perspective on the potential of integrating LLMs with cognitive components, such as memory, planning, and actions, to enhance agent collaboration and task execution. Their limitations underscore the need for further advancements in areas such as low-level action control and reasoning stability to ensure these agents perform optimally in dynamic and complex environments. Current approaches often focus on grouping agents by common behaviors (*e.g.*, a Planner agent), while a more effective strategy might involve eliciting agents that excel in specific tasks.

Our approach employs task-driven agents with similar cognitive components but distinct expertise. We propose a multi-agent system, where specialized agents collaborate to solve complex problems. Unlike prior studies, our architecture is explicitly designed for robust deployment across varying scenarios, integrates both agent–user and inter-agent communication layers, and enables seamless access to external tools and databases. It builds upon CoALA's principles [Sumers et al. 2024], but extends them into a scalable and task-oriented system.

3. LLM-Based Multi-Agent Architecture for Task-Solving

3.1. System Architecture

Figure 1 presents both the designed architecture and the internal modules of a Task-driven Agent (explained in the next subsection). Our proposal based upon the guidelines provided by CoALA [Sumers et al. 2024] while aiming for applying it to a robust and scalable system, thus the presence of many of its components in Task-driven Agents' modules. Moreover, as system's architecture is guided towards distinct scenarios, elaborating multi-agent approaches elicited needs for both conducting interactions with the end-user and between agents. In light of this, the architecture was elaborated to allow for communication with end-users, receiving, processing, and transforming queries; and a set of Task-driven Agents that can act together to answer user-generated queries. The Task-driven Agents can access possible external modules, such as databases and tools.

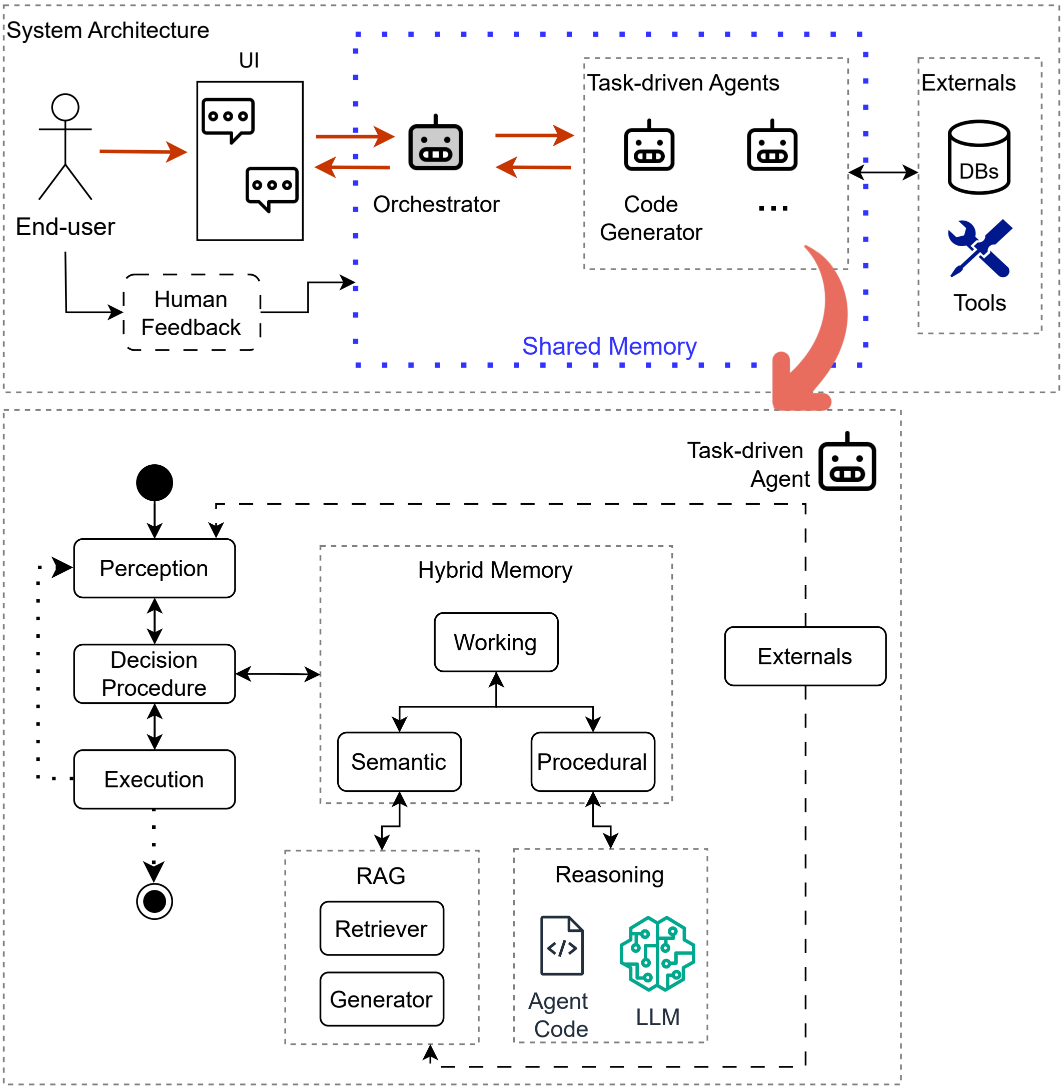

Figure 1. Our proposed Architecture System and Task-driven Agent model.

End-users interact with the system via a designated User Interface (UI) to make requests and obtain the associated results. Here, a specific agent, denominated **Orchestrator**, is instantiated. Based upon the requirements, the Orchestrator is prompted to decide which Task-driven Agent(s) are necessary to answer the end-user's query. At any given moment, the Orchestrator can access detailed information about all Task-driven Agents implemented in the system. This provides a configurable and expandable multi-agent environment. The Orchestrator is also responsible for mediating the communication flow between other agents.

Shared Memory is a workspace for all agents to interact with. It persists data such as retrieved elements from vector stores, resulting objects from initial queries, and other relevant data that can assist agents during the interaction with the end-user.

Human Feedback provides more contextual information about the conversations, especially whether the end-user's queries have been successfully answered. This information is stored in Task-driven Agents' memory, aiming to empower agents' reasoning in their plan elaboration and decision-making.

The set of Task-driven Agents is pictured in Figure 1 as an expandable module. A Code Generator was chosen to represent a specific task, but many others can be leveraged. Another example would be a Data Analyst that is specialized in identifying which visualization or validation methods would be best applied to answer a question from a database.

We assume that using personas can significantly leverage the agents' behavior [Becker 2024] and enhance their utility and effectiveness. The Task-driven Agents are prompted to be experts in their respective specific task areas. Our design decisions are in line with recent studies. For instance, the finding that expert agents can aid in complex tasks, such as ethical and strategic question-answering [Becker 2024].

3.2. Task-driven Agents

Our approach involves eliciting agents that excel in unique tasks, rather than grouping them by a common behavior (*e.g.*, a Planner agent). We named these agents as Task-driven. Each Task-driven Agent comprises a set of modules, designed to perform both internal and external actions that agents can undertake. Figure 1 presents the cognitive components for our Task-driven Agents. These components are linked by arrows, indicating how each module is connected and the types of data flux. The orange arrows represent the components used whenever a user requests a new query or message, meaning that these are always activated in this scenario. Black arrows indicate connections that might be requested by the different modules at any given time, but they are not necessarily triggered with each user message. Each agent is comprised of five main modules: Hybrid Memory, Planning, Execution, Perception, and Reasoning. Figure 1 presents the "hybrid memory" as the 'central hub' that allows communication among all components of the Task-Driven Agent. Dashed lines illustrate which agent's modules communicate with external tools and environments.

Hybrid Memory. It comprises two main memories: a short-term, focused on managing limited and in-context information pertinent to the current iteration; and a long-term, dedicated to the history of agents' actions, typically registered and retrieved from vector stores [Wang et al. 2024b]. **Working memory** is the only short-term memory component dedicated to retrieving and passing information to all other modules. As the name suggests, it is the working table for the agent to engage in planning, reasoning, and acting. The **Semantic Memory** is dedicated to retrieving and reflecting upon context-specific information from databases via Retrieval Augmented Generation (RAG) [Lewis et al. 2021] methods; and the **Procedural Memory** encompasses data used by the agent's interactions with an LLM to perform reasoning.

Decision Procedure (Planning). This module encapsulates both the planning and decision-making of the Task-driven Agent. It is configurable and modifiable depending on the agent's specific task. This decision was based on recent studies noting that different tasks perform well under different paradigms of decision-making. For instance, Becker *et al.* [Becker 2024] argued that single-agent Chain of Thought [Wei et al. 2022] outperforms multi-agent systems in basic tasks, such as translation. In our use-case scenario, this promotes ease of testing different configurations among planning abilities [Huang et al. 2024], thereby achieving better effectiveness across various domains.

Execution. When the plan is generated, the Task-driven Agent refers to the Executor module to transform these plans into action, using any tools it can access and deems necessary. This can result in an object-type response, such as a produced report or an image representing a plotted graph. The Executor module is responsible for delivering the results obtained. Depending on the executed task, these results can be stored in a database, potentially uploading existing data that can be used in future interactions.

Perception. The Perception module is used to enhance the knowledge of the Task-driven Agent based on the environment. It can either be previously stored information in a specific database, the Shared Memory, or in the contextual memory of the iteration, including what other Agents have previously done while working on the same user query.

Reasoning. This module leverages the interactions that the Agent has with the LLM. It is composed of both LLM calls and the Agent Code. It calls to the LLM to balance a more extensible and determined set of rules (defined in the Agent Code) with the power of stochasticity produced by the LLM's many configurable parameters. Again, this balance can be leveraged depending on the use-case scenario in which the architecture is employed.

4. Case Application

We provide a proof of concept for our proposed architecture, having developed a small-scale version to simulate an example. We replicate a conversation between three Task-driven Agents to perform a web scrapping search and retrieve stock prices for companies within a given period. We implemented agents that leverage conversation coordination mechanism, task-driven approaches, and shared memory management techniques, key-components of our proposed architecture.

4.1. Procedures

Our architecture was developed in Python 3.10 using LangGraph[1] as a framework basis. A Task-driven Agent is composed of four nodes: one for observation, one for asking for feedback, one for planning, and one for execution, following the logic described in Figure

---

[1] `https://langchain-ai.github.io/langgraph/`

1. As mentioned, three Task-driven Agents were designed for this case application: a Moderator Agent (MA), a Financial Agent (FA), and a Coder Agent (CA).

MA is designed as the **Orchestrator** to conduct the conversation among agents to best respond to the user's request. On parallel, FA's purpose is to scrap the web using an API to fetch and persist financial and market data, while also being responsible for answering questions about the data it obtains. The CA's objective is to generate Python code that manipulates and visualizes already obtained financial data.

The Report conversation paradigm [Becker 2024] was chosen for the communication between agents. It starts with MA, which then selects the best agent to act next. After the next agent finishes, the word is returned to MA, which, based on the response, determines whether the goal was achieved. If it was, results are sent back to the user; if not, MA passes the result to another agent, and the cycle continues. Figure 2 presents the schema for this conversation.

Memory modules employ a persistent mechanism that combines the *Pickle*[2] library with data storage in *OpenSearch*[3]. We implemented a basic RAG approach to conduct data retrieval. As for the Shared Memory, the library *NetworkX*[4] was used to establish a timeline of the created objects, which any agent could access.

We used *yfinance*[5] as the external tool calling to fetch financial and market data. This API was important so the Financial Agent could handle both fetching and analyzing retrieved data based upon the user's request. Task-driven Agents' modules (observe, ask for feedback, plan, and execute) were developed with the same LLM: `GPT-4o`. In parallel, we have used OpenAI's `embedding-openai-ada-002` for the RAG system.

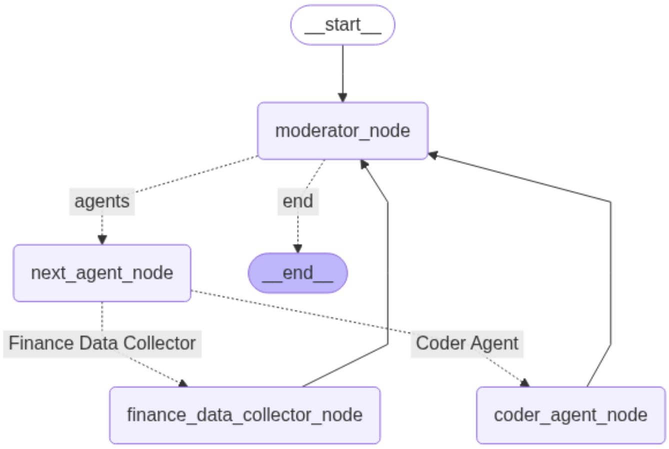

Figure 2. Report conversation schema instantiated in the case application.

The initial State of the graph is instantiated directly via code, simulating the initial request from the user (here called *Sheep of Gate Avenue — SoGA)* to MA. The user's request is: "*Make a chart of the evolution of prices of companies in the telecommunication and the oil and gas sector in the last 2 months*". MA is instantiated with a basic

---

[2] `https://docs.python.org/3/library/pickle.html`

[3] `https://opensearch.org/`

[4] `https://networkx.org/`

[5] `https://pypi.org/project/yfinance/`

prompt description as an assistant. After receiving the request, MA passes the word to FA. This agent was also built with a special memory that leverages the RAG system to answer questions by retrieval. Since this memory is initially empty, FA uses *yfinance*'s tool to fetch and store the data. We implemented a relevant guardrail to prevent hallucination: if no data was found about a company, FA was instructed to explicitly state this (instead of making up information). After condensing the answer about the companies as a DataFrame, the word was sent back to MA and then forwarded to CA. This agent's coding also presents a necessary guardrail, only permitting to use the libraries *pandas*, *numpy*, and *plotly*. Additionally, CA was also requested to never generate code for downloading or generating synthetic data.

4.2. Results

Figure 3 presents an abstracted timeline of the execution of the case application. From left to right, the figure illustrates the sequence of agents' actions. Moreover, stores in the Shared Memory also follow an analogous paradigm, indicating which objects are first stored in it. Dark green arrows indicate input, whereas light blue arrows indicate output. The purple dashed lines within the Shared Memory indicate that an update has occurred.

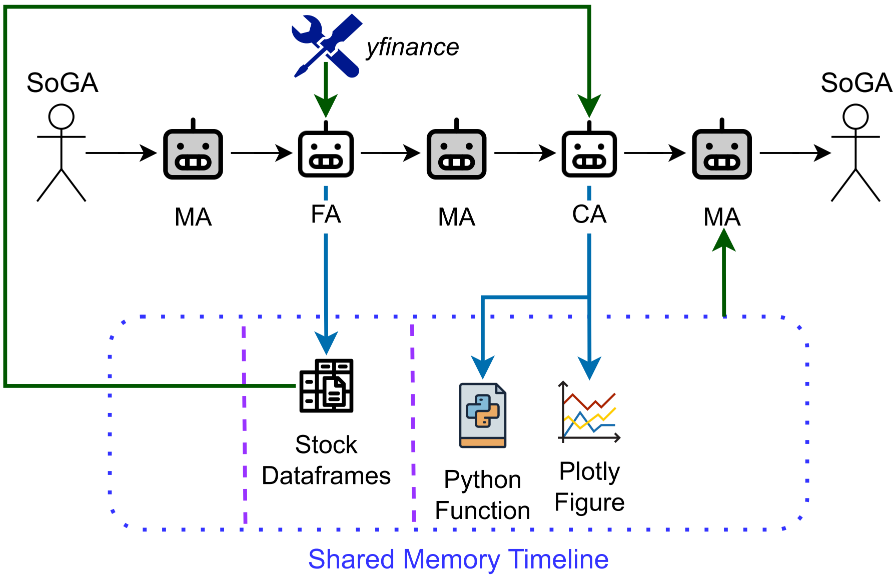

Figure 3. Abstracted timeline of the running case application.

The conversation among agents is displayed in Chat 1. It complements Figure 3 by specifying message exchange and a general overview of what each Task-driven Agent has done and how the results are organized. At the end, MA returns with the chart and a description how it should be read.

Chat 1. Conversation between user and Task-driven Agents.

MA: Hi! How can I assist you today?
SoGA: Make a chart of the evolution of prices of companies in the telecommuni-

cation and the oil and gas sector in the last 2 months.

—

MA: Please download the price data for companies in the telecommunication and oil and gas sectors for the last 2 months.

FA: The price data for companies in the telecommunication and oil and gas sectors for the last 2 months has been successfully downloaded. Here are the details:

1. *Telecommunication Sector*: Qualcomm (QCOM): Data ID - ee072f76-4d73-41e4-9047-b4d0a585fc19; T-Mobile US, Inc. (TMUS): Data ID - 4796c275-c1eb-4cc0-8fff-3f0ee031d6f4; Comcast Corporation (CMCSA): Data ID - 9aadfec1-553c-4f52-abad-6fa7883bb1c1
2. *Oil and Gas Sector*: Exxon Mobil Corporation (XOM): Data ID - 534dd2ce-c09f-4ddd-902a-3bc047b1a413

These DataFrames contain daily stock price data including ’Open’, ’High’, ’Low’, ’Close’, ’Volume’, ’Dividends’, and ’Stock Splits’ for each company over the specified period. Please let me know if you need further assistance.

MA: Please create a chart showing the evolution of prices for the companies in the telecommunication sector (Qualcomm, T-Mobile US, Inc., and Comcast Corporation) and the oil and gas sector (Exxon Mobil Corporation) over the last 2 months using the provided data.

CA: I generated a function to create a line chart showing the evolution of stock prices for the specified companies in the telecommunication and oil and gas sectors over the last 2 months. The function processes the provided DataFrames, extracts the ’Close’ prices, and plots them using Plotly. The resulting figure object is returned.

—

MA: The chart showing the evolution of prices for companies in the telecommunication sector (Qualcomm, T-Mobile US, Inc., and Comcast Corporation) and the oil and gas sector (Exxon Mobil Corporation) over the last 2 months has been successfully created. You can view the line chart, which visualizes the closing stock prices over this period. The chart includes lines for each company, with the x-axis representing dates and the y-axis representing closing prices in USD. If you need further assistance or additional data, feel free to ask. Here is the chart: [fig (id: 49d85463-72b1-4b46-bd16-2a39785dfd3d)].

We have created an anonymous repository[6] to contain parts of the case application, such as agents’ and system prompts, the whole conversation messages, and the retrieved items from the Shared Memory.

5. Discussion

We describe the challenges in developing our proposed architecture, which extends beyond code implementation. They are instead deepened in theoretical inquiries, such as component integration, memory management policies, communication paradigms among the Task-driven Agents, and their emergent effects, as well as different forms of the architecture’s evaluation.

---

[6] `https://anonymous.4open.science/r/wesaac-financial-agent-4F74/`

Our case application revealed the potential of memories in leveraging a Task-driven Agent's information retrieval and, therefore, its reasoning process. It demonstrated the agent's ability to fetch information from the web and store it using embedding mechanisms for later retrieval. Moreover, the Shared Memory's potential was leveraged by all agents having access to previously stored data, thus facilitating its use for reasoning and acting. In our case application, if the user requests further chart customization, only the Coder Agent needs to modify the previously created function. Our Shared Memory paves the way for understanding the potential of what can be achieved when multiple agents interact with each other and the users themselves.

The Report [Becker 2024] conversational paradigm implemented offers a simple to coordinate mechanism, while it promotes agents' individuality and positively affects their direct collaboration. Shared Memory is like a typical working table that only stores results. Should the agents converse directly, they would know each other and directly express their opinions alongside their results. This can be achieved through other conversational paradigms that will be implemented in further studies.

Single-agent systems are better suited to solve simple tasks, whereas complex tasks can benefit from multi-agent collaboration [Becker 2024]. Our Task-driven agent, equipped with its cognitive components, is flexible enough to leverage scenarios where, depending on the task's complexity, some modules, such as Semantic Memory, may be unnecessary for the agent. Further ablation studies are necessary to assess these hypotheses under different tasks with different levels of complexity.

The decision procedure of the Task-driven Agents is highly related to the memory modules [Wang et al. 2024a, Huang et al. 2024] and the collaboration among other agents [Becker 2024, Yin et al. 2023]. A series of challenges arise from this fact. For instance, [Becker 2024] argued that agents should go beyond information retrieval regarding long-term memory. This requires investigating different RAG approaches to evaluate their effectiveness in supporting planning and decision-making. For example, could a Knowledge Graph [Ehrlinger and Wöß 2016, da Costa et al. 2022], constructed upon an ontology for a specific domain, enhance the agents' capabilities of understanding underlying conceptual terms that average LLMs might misunderstand due to not having been trained upon? This question raises further discussions between fine-tuning and RAG, with the literature generally pointing out that both techniques should be used to achieve better results [Pan et al. 2024].

An open challenge is related to performing a coherent evaluation of general LLM-based multi-agent architectures, such as the one proposed in our investigation. While there are many established benchmarks for running tests on specific scenarios [de Souza Soares et al. 2021, Zhuo et al. 2024], a lack of similar methods exists to evaluate general architectures. The investigation and development of general benchmarks that can instantiate different scenarios applicable to cognitive single- or multi-agent frameworks would undoubtedly be a valuable asset for comparison in the literature.

While we have not fully implemented our proposed architecture, the provided case study employed a general, but real-life, purpose application. The obtained results can pave the way for building other modules and enriching the cognition of our Task-driven Agents to reach a robust and scalable multi-agent framework that can be applied in scenarios of varying dimensions, from small or medium-sized education applications to being part of complex industry systems.

6. Conclusion

The design and development of LLM-based multi-agent systems still suffer from the scientific challenge of enabling coherent, goal-aligned collaboration and communication among autonomous agents while maintaining interpretability, scalability, and adaptability in dynamic environments. Our investigation presented an LLM-based multi-agent architecture designed to conduct conversational cognitive-enhanced tasks driven by expert agents. The architecture's structure envisions the creation of multiple LLM-based cognitive agents that are experts in a particular area and discuss among themselves — and the user, via direct feedback — to leverage reasoning. Our case study revealed that a Shared Memory that stores middle or final constructed objects can potentially increase the reasoning process among agents. We plan to investigate further the extent to which multiple agents' conversational task-solving, having access to this memory, can demonstrate their full potential.

Acknowledgments

This study was sponsored by Petróleo Brasileiro S.A. (PETROBRAS) within the project "*Application of Large Language Models (LLMs) for online monitoring of industrial processes*" conducted in partnership with the University of Campinas.

References

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., Sri, D. W., Barrett, A., Christianson, D., et al. (1998). Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*

Becker, J. (2024). Multi-Agent Large Language Models for Conversational Task-Solving. *arXiv preprint arXiv:2410.22932*, (arXiv:2410.22932).

Cardoso, R. C., Ferrando, A., and Papacchini, F. (2021). Automated Planning and BDI Agents: A Case Study. In *Workshop-Escola de Sistemas de Agentes, Seus Ambientes e Aplicações (WESAAC)*, pages 131–142. SBC.

da Costa, F. J., Avila, C. V. S., Rolim, T. V., de Castro Andrade, R. M., and Vidal, V. M. P. (2022). Dikw4iot: Uma abordagem baseada na hierarquia dikw para a construção de grafos de conhecimento para integração de dados de iot. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 190–202. SBC.

de Souza Soares, E. F., Souza, R., Thiago, R. M., Machado, M. d. O. C., and Azevedo, L. G. (2021). A recommender for choosing data systems based on application profiling and benchmarking. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 265–270. SBC.

Ehrlinger, L. and Wöß, W. (2016). Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48(1-4):2.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (1999). The belief-desire-intention model of agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages: 5th International Workshop, ATAL'98 Paris, France, July 4–7, 1998 Proceedings 5*, pages 1–10. Springer.

Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., and Chen, E. (2024). Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716.*

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401*, (arXiv:2005.11401).

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., and Wu, X. (2024). Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., et al. (2024). Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.

Sumers, T., Yao, S., Narasimhan, K., and Griffiths, T. (2024). Cognitive architectures for language agents. *Transactions on Machine Learning Research.* Survey Certification.

Tavares, C. J. and Ralha, C. G. (2024). Multi-agent System Architectural Aspects for Continuous Replanning. In *Workshop-Escola de Sistemas de Agentes, Seus Ambientes e Aplicações (WESAAC)*, pages 39–50. SBC.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. (2024a). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Wang, Y., Pan, Y., Zhao, Q., Deng, Y., Su, Z., Du, L., and Luan, T. H. (2024b). Large Model Agents: State-of-the-Art, Cooperation Paradigms, Security and Privacy, and Future Trends. *arXiv preprint arXiv:2409.14457.*

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yin, Z., Sun, Q., Chang, C., Guo, Q., Dai, J., Huang, X., and Qiu, X. (2023). Exchange-of-Thought: Enhancing Large Language Model Capabilities through Cross-Model Communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153, Singapore. Association for Computational Linguistics.

Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., and Gan, C. (2023). Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485.*

Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., et al. (2024). Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877.*