LLM-Agent-UMF: LLM-based Agent Unified Modeling Framework for Seamless Integration of Multi Active/Passive Core-Agents

Preprint

Amine B. Hassouna[*][1,2], Hana Chaari[†][§][1,2], and Ines Belhaj[‡][§][1,2]
[1]Mediterranean Institute of Technology, South Mediterranean University, 99628, Tunis, Tunisia
[2]Dracodes, 1006, Tunis, Tunisia

Abstract

The integration of tools in LLM-based agents overcame the difficulties of standalone LLMs and traditional agents' limited capabilities. However, the conjunction of these technologies and the proposed enhancements in several state-of-the-art works followed a non-unified software architecture resulting in a lack of modularity. Indeed, they focused mainly on functionalities and overlooked the definition of the component's boundaries within the agent. This caused terminological and architectural ambiguities between researchers which we addressed in this paper by proposing a unified framework that establishes a clear foundation for LLM-based agents' development from both functional and software architectural perspectives. Our framework, LLM-Agent-UMF (LLM-based Agent Unified Modeling Framework), clearly distinguishes between the different components of an agent, setting LLMs, and tools apart from a newly introduced element: the core-agent, playing the role of the central coordinator of the agent which comprises five modules: planning, memory, profile, action, and security—the latter often neglected in previous works. Differences in the internal structure of core-agents led us to classify them into a taxonomy of passive and active types. Based on this, we proposed different multi-core agent architectures combining unique characteristics of various individual agents. For evaluation purposes, we applied this framework to a selection of state-of-the-art agents, thereby demonstrating its alignment with their functionalities and clarifying the overlooked architectural aspects. Moreover, we thoroughly assessed four of our proposed architectures by integrating distinctive agents into hybrid active/passive core-agents' systems. This analysis provided clear insights into potential improvements and highlighted the challenges involved in the combination of specific agents.

Keywords LLM-based agents · software architecture · modularity · security · classification · taxonomy · core-agent

1 Introduction

Large Language Models (LLMs) excel in tasks like language modeling, text generation, question answering, sentiment analysis, natural language understanding, and commonsense reasoning **[1]**. However, standalone LLMs lacks other skills such as accessing external knowledge, information retrieval, mathematical reasoning, code evaluation, and numerous others. These functional shortcomings can be managed by AI agents by leveraging external tools and human feedback. An autonomous agent is a system interacting with an environment, sensing it, and acting on it over time following a certain agenda **[2]**. There are different classes of agents but in this paper, we will be focusing on LLM-based agents which, by combining the capabilities of LLMs and autonomous agents, can achieve a broad range of tasks **[1]**.

[*]`amine.benhassouna@medtech.tn, amine.benhassouna@dracodes.com (Corresponding author)`
[†]`hana.chaari@medtech.tn, hana.chaari@dracodes.com`
[‡]`ines.bel-hadj@medtech.tn, ines.bel-hadj@dracodes.com`
[§]Equal contribution.

In fact, LLM-based agents have now become dominant in the landscape of AI agents due to several reasons. The first is the previously discussed versatile set of capabilities inherited from LLMs. Secondly, LLMs are regarded as potential steppingstones towards Artificial General Intelligence (AGI), offering hope for the development of general AI agents that can adapt to diverse scenarios **[3]**. Finally, the shift towards more natural conversational interfaces powered by LLMs is transforming the way humans interact with agents, enabling seamless and intuitive interactions.

Understanding the potential of these agents and improving them necessitates a deep understanding of their structure. In LLM-based agents, besides the LLM who handles reasoning, there are other components that oversee the execution of tasks, ensure the security of the agent, and handle its memory **[3]**. But merely pinpointing existing functionalities is insufficient for developers and researchers. To provide a comprehensive overview of these agents, survey **[4]** proposes a comprehensive framework for their construction, consisting of four main modules which will be discussed thoroughly in the next section. Although it provides valuable insights into the functionality of each module in an agent, it overlooks their delineation from a software architecture perspective. This perspective is crucial for developers to establish a common base architecture to build upon and for researchers to improve. Our analysis of the state-of-the-art LLM-based agents reveals common limitations: the complexity of implementation, resulting in unstructured and ambiguous software architecture; lack of modularity and composability, making components non-reusable by other agent-based solutions; and difficulty in maintainability and introducing improvements to existing agents. These issues stem from the absence of a clear framework that emphasizes the architectural perspective of the agent as much as its functionalities.

Our proposed LLM-based Agent Unified Modeling Framework (LLM-Agent-UMF), emphasizes a clear delineation of each component within an LLM-based agent, defining their interactions within specified boundaries from both architectural and functional perspectives. In addition to LLMs, we introduced a new unit within the agent: the core-agent, which we classify into two types: active and passive core-agents. This proposition adds clarity to the capabilities of each component and describes the dynamics of interactions between modules within the agent. As a result, we alleviate the complexity of the architecture and improve the reusability of the components, promoting a shift from multi-agent systems to multi-core agents.

In the evaluation section, we highlighted the advantages offered by our framework, from resolving terminological ambiguities to the introduction of enhancing modules like the security module. To validate the reliability of our framework, we applied it to existing solutions and identified the modules within each agent. This approach enabled us to assess the feasibility and requirements for merging one agent with another, ultimately aiming to create an agent with fully enhanced capabilities.

Compared with previous works, the 5 main contributions of this paper can be summarized as follows:

1. Introducing a new terminology, core-agent, as a structural sub-component in LLM-based agents to improve modularity and promote more effective and precise communication among researchers and contributors in the field of agents and LLM technologies.
2. Modelling the internal structure of a core-agent by adapting the framework suggested by **[4]** that was originally meant to describe the whole agent from an abstract functional perspective.
3. Improving our modeling framework by augmenting the core-agent with a security module and introducing new methods within other modules.
4. Classifying core-agents into active core-agents and passive core-agents, explaining their differences and similarities, and highlighting their unique advantages.
5. Finally, introducing various multi-core agent architectures emphasizing that the hybrid one-active-many-passive core-agent architecture is the optimal setup for LLM-based agents.

The rest of this paper is organized as follows. Section 2 covers a background on LLM-based agents and reviews relevant state-of-the-art works. Section 3 introduces our LLM-based agent unified modeling framework and possible architectures. Section 4 evaluates the efficiency of our proposed agent designs leveraging the capabilities of multiple distinctive agents. Finally, Section 5 summarizes key findings and discusses future challenges.

2 Related Work

We will start off this section by providing a comprehensive overview of the foundational concepts upon which our work is based. First, Tool-augmented LLMs are a major advancement in NLP that combine the language understanding and generation capabilities of LLMs with the ability to interface with external tools and APIs. For instance, TALM **[5]** introduces models which can leverage a wide range of functionalities, from information retrieval to task planning and execution.

By incorporating tool-augmented LLMs, LLM-based autonomous agents exhibit exceptional proficiency in NLP tasks **[6]**, including reasoning **[7]**, programming **[8]**, and text generation, surpassing other types of agents in these areas. Moreover, they address several limitations of standalone LLMs, such as context length constraints and the inability to utilize tools. This development marks a significant breakthrough in the scientific community, explaining the recent surge in the adoption of LLM-based agents.

Researchers and practitioners across various disciplines are leveraging these agents to tackle complex problems and drive innovation in fields such as gaming **[9]** and other professional domains that require specialized expertise **[10]**. Additionally, the trend towards integrating LLMs in scientific research, namely in chemistry **[11]**, highlights their transformative potential, promising to drive forward new discoveries and applications.

Upon examining existing agents, we detected a notable absence of direct security measures or guardrails within the agents to ensure the protection of sensitive information and enhance overall system integrity. Indeed, the level of autonomy in LLM-based agents raises significant concerns regarding ethical use, malicious data, privacy and robustness **[12]**. Notably, one critical risk involves jailbreaks which can be mitigated through the implementation of more robust monitoring and control mechanisms to detect and respond to jailbreaks during deployment **[13]**. Moreover, data privacy remains a persistent challenge in any software system, including agents. To address this, various methods have been developed to safeguard against data extraction attempts from prompts, such as leveraging privacy-preserving algorithms for prompt learning **[14]**. These findings prompted us to place an emphasis on this often-neglected aspect and include security measures in our framework.

Besides security assurance, the development of AI systems, particularly agents, must adhere to fundamental software development principles such as modularity, composability, and maintainability. These principles enhance the flexibility, scalability, and adaptability of AI systems, enabling them to effectively meet the evolving needs of both users and businesses.

Existing agents do not necessarily respect these principles as they do not focus primarily on architectural design. This underscores the importance of architectural frameworks as blueprints for LLM-based agents, highlighting their essential role in releasing the full potential of these systems. Such frameworks enable the development of modular, robust, extensible, and interoperable designs. They provide the necessary scaffolding to build increasingly capable and reliable agents capable of tackling complex real-world problems.

For example, the paper **[6]** exploring LLM-based intelligent agents points out 5 main axes: Planning, Memory, Rethinking, Environment, and Action. Despite their attempt to delineate between LLMs, environment and tools, they did not define the software components of the agent. Likewise, the framework proposed by the survey **[4]** identifies the structure and applications of LLM-powered autonomous agents and highlights four key modules as shown in Figure 1 : the Profile module, the Memory module, the Planning module, and the Action module.

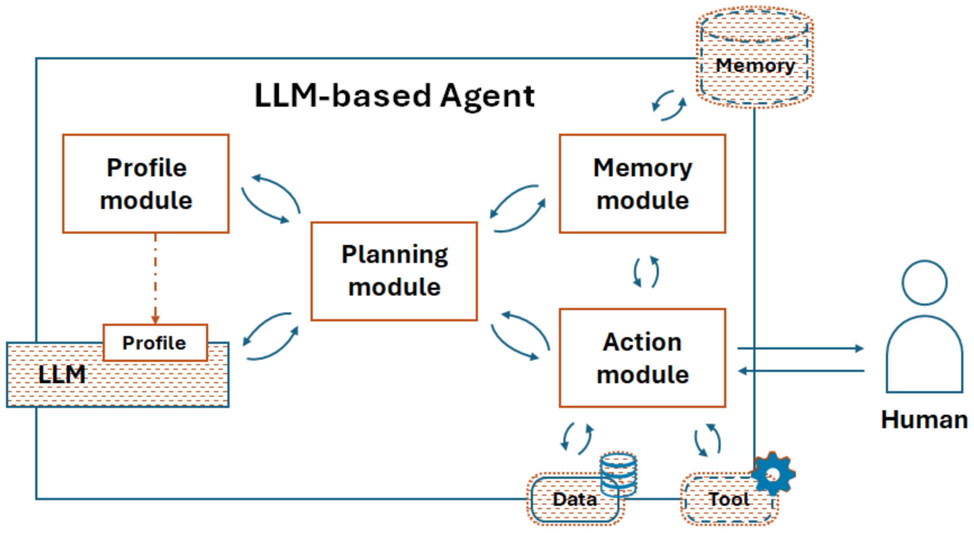


Figure 1: Framework proposed by survey **[4]** for LLM-based agents

The profile module serves to delineate the diverse roles of the LLM. Meanwhile, the memory module retains internal logs encompassing the agent's past thoughts, actions, and observations within its dynamic environment, including interactions with users. The planning module guides the agent in decomposing overarching tasks into manageable steps or subtasks, enhancing responsiveness by leveraging past behaviors in future plans. Together, these modules significantly influence the action module, which translates the agent's decisions into specific outputs harnessing external tools to extend the agent's capabilities **[4]**.

While this framework effectively addresses the functionalities of the agent, it does present some areas for improvement. First, there are functional overlaps in certain modules, particularly the overlap of reflective activities between planning and memory modules. Second, the definition of memory is ambiguous as will be discussed in section 3.1, leading to confusion as the term is used to represent different concepts without clear distinction. Lastly, as illustrated in Figure 1, it is not explicitly stated whether the LLM, tools, data sources, and memory are part of the agent. This fuzzy distinction between the functionalities of each module foster division between software developers and leads to incompatibility and discourages reusability. In the next section, we introduce the main components of our framework, explain the rationale behind their inclusion, and highlight how they solve the limitations present in other works.

3 LLM-based Agent Unified Modeling Framework (LLM-Agent-UMF)

3.1 Core-Agent: Keystone component of an LLM-based agent

Several works have aimed to establish a well-defined framework for building LLM-based agents. For instance, the paper **[15]** presents a framework for designing educational problem-solving simulations using LLM-powered agents. The authors emphasize that separating the AI agent from the environment is important in the design process. However, the paper does not provide a clear framework for the agent's components, making it challenging for future work to identify specific points of modification or reuse. Without a transparent and detailed outline of the agent's architecture and component interactions, it is difficult to pinpoint where changes can be made to alter the agent's behavior or how its components could be reused and integrated in other systems.

This issue arises from a lack of modularity in the software design, which can be mitigated by adhering to the Single Responsibility Principle (SRP). As emphasized in the paper **[16]**, the SRP is crucial in software development because it provides granularity at different levels of the software, both in terms of code and functionalities. This granularity facilitates the implementation of future improvements and enhances reusability opportunities. Additionally, applying this principle prevents various code smells, ensuring the modularity of the code and thus making it easier for practitioners to manage and maintain the software.

Paper **[17]** attempts to establish boundaries between LLM-based agents, the tools they utilize, and their surrounding environment. Although the proposed LLM-based agent system offers an abstract separation of the agent's components, it does not clearly delineate them from a software engineering perspective. Understanding the theoretical contributions of each component within a unified agent is valuable; however, from a practical development standpoint, it is essential to identify the location, functionality, and role of each internal component within the agent.

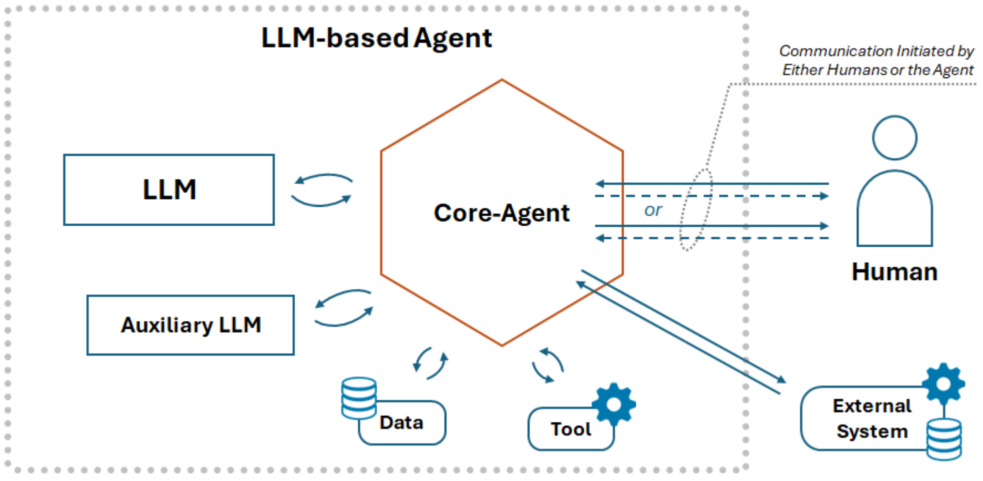


Figure 2: The core-agent as the central component of LLM-based agents

Upon a thorough analysis of LLM-powered agents from a software perspective, we recognize an LLM-based agent as a software system comprising various components including tools, LLMs, and the core-agent, our newly introduced term. While the core-agent is not a newly invented component, it serves as a label to denote an existing functional element that has been previously implicit or unnamed in past frameworks or architectures. Essentially, this label highlights its crucial role within these systems. This terminology helps clarify the structure of an LLM-based agent by identifying its essential parts. As depicted in Figure 2, the core-agent interacts with the environment, collaborates with the language model to make decisions, and translates high-level goals into concrete actions optionally by leveraging the available tools.

Core-Agent:

In this architecture, the core-agent serves as the keystone component, acting as the crucial interface between itself and all the other elements of the agent. It facilitates communication and coordination among these various parts to ensure optimal functionality of the entire system. Functioning as the executor of the agent, the core-agent translates plans developed by the Large Language Model (LLM) into actionable steps potentially leveraging tools and engages with the environment to provide the agent with insights into the external world. These capabilities enable the core-agent to operate as a controller, ensuring precise and complex synchronization of actions that result in flawless interactions and effective information sharing.

The significance of the core-agent is underscored by its symbiotic relationship with the LLM, with each enhancing the other's capabilities. LLMs unveils powerful language understanding, cognitive and reasoning abilities, alongside extensive knowledge. However, it lacks the perception and action components, which are provided by the core-agent, enabling direct environmental interaction. This collaboration expands the scope of problems the agent can address, effectively combining the strengths of both the LLM and the core-agent to tackle a broader range of challenges.

Furthermore, the presence of the core-agent as a controller mitigates the overall complexity of the agent, especially in a multi-LLM setup where it serves as the primary communication hub within the agent. Consequently, information flow between the LLMs is managed through the core-agent, streamlining interactions and ensuring efficient coordination. In terms of communication, the core-agent is capable of interacting with humans through a well-structured pipeline. However, there are instances where the core-agent may not engage in direct communication with humans. This variability is dependent on the specific type of core-agent, as elucidated in detail in Section 3.3.

LLM:

The LLM acts as a cerebral entity covering cognitive tasks such as natural language understanding and comprehensive text generation based on specific context. Indeed, LLMs can communicate with core-agents which in turn can interact with tools or data sources within the system. Considering that an agent can manifest as unimodal or multimodal **[17]**, the latter can be replicated with different task-specific LLMs, as illustrated in Figure 2. This distinction fosters modularity within our LLM-based agent unified modeling framework, making them open for extension and addition of further domain-specific models without having to introduce changes to the overall system architecture, adhering more to the Open-Close principle (OCP) **[18] [19]** .

Tools:

In LLM-based agents, tools are important assets. They can take various forms such as supplementary systems, software applications or even physical devices that extend and enhance the capabilities of an agent. They span across a spectrum of complexity, ranging from basic API integrations to sophisticated auxiliary systems designed for specific tasks. In this context, tools can be considered external if they are independent systems which achieve a complete objective, or internal if they cooperate with the core-agent to achieve tasks in the scope of the agent goal.

In conclusion, our proposed terminology for an LLM-based agent, which includes a core-agent as its central coordinating component, serves to enhance clarity and consistency in describing these systems from a software perspective. The underscored significance of the core-agent highlights the importance of investigating its internal structure. Section 3.2 focuses on defining the various modules within the core-agent and how they collaborate to facilitate cognitive tasks, decision-making, and action execution leveraging available tools.

3.2 Modeling the Core-Agent Internal Structure

The human brain exhibits remarkable modularity, with distinct regions and functionalities working in coordination to facilitate cognitive processes, decision-making, and behavior. Inspired by this organized design, our proposed framework aims to emulate the brain's modular structure thought the incorporation of five internal modules: the planning module, the memory module, the profile module, the action module and the security module. By integrating the latter concepts, the core-agent serves as the central coordinator, controlling the interactions and information flow between different components, as elaborated in the previous section.

This modular framework effectively addresses the challenges of extendibility and maintainability. Specifically, it allows for independent development and integration of new modules without impacting the entire system. In fact, such modules can be replaced or upgraded separately, facilitating the addition of new capabilities and enabling the framework to adapt more easily to new requirements or technologies. Furthermore, the modular structure promotes code reusability, as individual modules can be shared across different agents or applications, thereby reducing duplication and enhancing consistency.

To provide a robust comparison and underscore the novelty of our framework, it is instructive to examine it alongside existing approaches. A recent survey **[4]** introduced a framework for LLM-based agents, comprising four key modules: profile, memory, planning, and action modules. For a comprehensive overview of this framework, we direct readers to the "Related Work" section. While this survey's proposed framework **[4]** treated the LLM-based agent as a whole system without distinguishing a core component from other entities, our adaptation, tailored specifically for the core-agent, necessitates modifying certain definitions and module structures to accommodate the separation between the LLM and the core-agent. Notably, we had to reposition the four key modules to operate under the core-agent within the framework alongside our newly introduced security-oriented module.

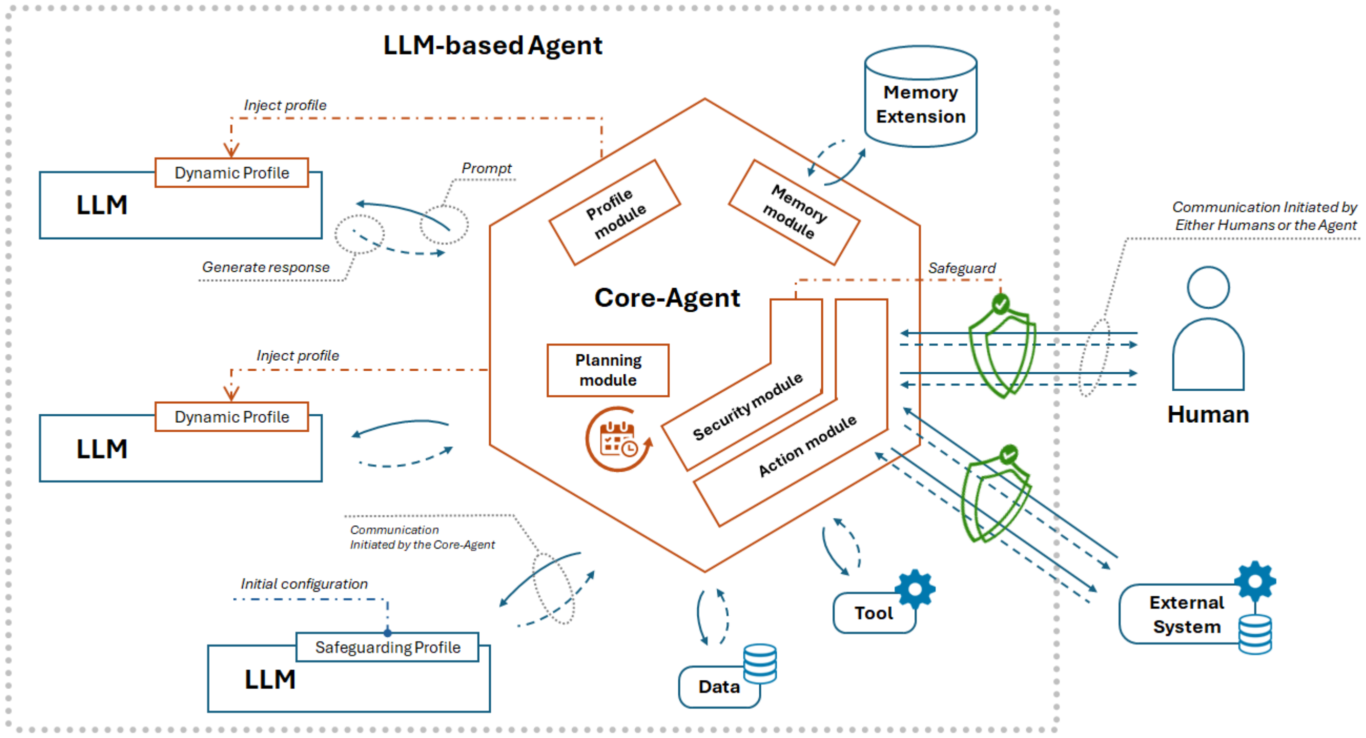


Figure 3: The internal structure of a core-agent

3.2.1 Planning Module

In our proposed framework, the planning module is a pivotal element that enables the agent to break down the complex problems to generate effective plans, the same as in the original framework **[4]**. However, in our solution the planning module becomes part of the core-agent and collaborates with the other sibling modules to empower the agent to achieve specific goals.

The planning module requires complicated understanding and reasoning **[20]** which could leverage the capabilities of an LLM. In fact, the LLM's ability to comprehend nuanced instructions, interpret implicit information, and adapt to various problem domains renders it an invaluable asset in the planning process. Consequently, the planning module can formulate more comprehensive, context-aware, and adaptable plans, significantly enhancing the core-agent's decision-making process. Furthermore, the planning module works in close collaboration with all other modules within the core-agent, including the memory module for Memory-augmented Planning **[21]**. This collaboration enhances planning capabilities by leveraging stored information such as commonsense knowledge, past experiences, and domain-specific knowledge.

This section will detail the functionalities and characteristics of this module from four critical aspects: process, strategies, techniques, and feedback sources. These aspects derive from the clear separation we have established between the core-agent and the LLM, and between our system and external systems Figure 2.

Planning process:

While generating the procedures to undertake, the planning module follows an incremental approach. Therein, the steps dictate how tasks are decomposed, how the planning procedure unfolds, and how alternative solutions are generated and evaluated. Inspired by the human capacity to decompose complex tasks into simpler ones to achieve overarching goals, this process comprises two main steps **[21]**.

- **Task decomposition:**
  This initial phase involves breaking down a complex task into simpler subtasks, thereby establishing a structured hierarchy of intermediate goals. The decomposition of complex tasks can adopt two primary approaches: In the non-iterative decomposition approach, the complex task is broken down into simple subtasks all at once. The planning module defines all subtasks, creating a complete task hierarchy in a single step. This method provides a comprehensive overview of the entire task structure upfront. However, the iterative decomposition approach involves a step-by-step breakdown of the task. In fact, the planning module first defines the initial subtask and goal to reach, establish a proper plan and generates the procedures to undertake. After performing the procedures and completing the plan, the output is considered, and the next subtask is defined. This process is repeated, with each new subtask being planned and executed, contributing to defining the next subtask to achieve. The key advantage of this method is its flexibility and ability to adapt the plan based on the outcomes of each subtask. An exemplary application of the iterative approach is the Decomposition-Alignment-Reasoning Agent (DARA) framework **[22]**. DARA **[22]** demonstrates how iterative decomposition can lead to more precise and context-aware planning, particularly for complex tasks with multiple interdependent subtasks.
- **Plan generation:**
  For each subtask derived from the decomposition step, a specific plan is established outlining the procedures to achieve the task's goal while defining the different tools and parties involved. Depending on the chosen planning strategy, the module can generate either a single plan or multiple candidate-plans for each subtask which will be further detailed in the following section.

Planning strategies:

To guide the planification, organization and execution of complex tasks, the planning module must adhere to a specific strategy when elaborating the procedures during the plan generation step. Selecting a planning strategy can substantially influence the effectiveness, efficiency, and robustness of the resulting plans. Within the LLM-Agent-UMF, we define two primary strategies:

- **Single-path strategy:**
  This approach involves generating a singular path or sequence of procedures to achieve the goal, adhering to the plan step-by-step without exploring alternatives, thereby providing a straightforward, deterministic approach to planning. Chain of thought (CoT) **[23]** is an example that outlines such a strategy. Indeed, it uses sequential reasoning as it involves breaking down complex problems into multiple procedures, each built on the previous ones **[23]**.
- **Multi-path strategy:**
  Consequently, in single path strategy, any error in one procedure can lead the subsequent procedures or the overall plan to be suboptimal or infeasible **[21]**, negatively impacting the entire strategy. A straightforward approach to mitigate such failures is the Multi-path strategy, which involves two major steps: The first phase involves leveraging the LLM to generate multiple plans for the complex task. Indeed, each intermediate step holds multiple subsequent paths **[4]**. As for the second phase, it deals with the evaluation and the selection of the most suitable path.

  As a matter of fact, the Tree of Thoughts (ToT) **[24]** and Graph of Thoughts (GoT) **[24]** are two frameworks that utilize this multi-path approach. They both operate by leveraging an LLM as a thought generator to produce intermediate procedures, that are structured either as a hierarchical tree in case of ToT or as a more complex graph in case of GoT. However, the complexity of managing these structures cannot be solely handled by the LLM. It requires the integration of a specialized software component responsible for orchestrating the process, interacting with the LLM, and organizing the thoughts into the desired structure, whether a tree or a graph. This essential software component is identified as the planning module within the core-agent. The planning module further evaluates different paths within the generated structure and selects an optimal plan **[21]** based on its assessment.

Planning Techniques:

The planning module follows planning techniques as methodological approaches to form executable plans. These techniques are chosen based on criteria such as the complexity of the task, and the need for contextual comprehension. Our framework present two primary techniques:

- **Rule-based technique:**

  Within our architectural framework, rule-based methodologies encompass what is commonly referred to in literature as symbolic planners **[21]**. These techniques proved to be valuable especially in contexts characterized by complex constraints, such as mathematical problem-solving or the generation of plans within highly problematic situations **[21]**.

  Symbolic planners, leveraging frameworks like PDDL (Planning Domain Definition Language), utilize formal reasoning to delineate optimal trajectories from initial states to targeted goal states **[25]**. These methodologies entail formalizing problem scenarios into structured formats, subsequently subjecting them to specialized planning algorithms.

- **Language model powered technique:**

  Language Model powered (LM-powered) methodologies leverage the vast knowledge and reasoning capabilities inherent in LMs to orchestrate planning strategies. Within our framework, this category also encompasses neural planners **[21]**, who are adept at addressing intricate and vague tasks necessitating nuanced comprehension and adaptive problem-solving abilities.

This categorization in our framework provides a clear distinction between approaches primarily driven by LMs and those relying on rule-based methods. It allows for a more streamlined understanding of planning techniques within the context of our core-agent architecture, while still acknowledging the valuable contributions of various planning approaches in **[4]** and **[21]**.

Feedback Sources:

Planning without feedback may pose several challenges as feedback plays a crucial role in optimizing the performance of the planning module within the core-agent. For instance, in iterative task decomposition, feedback has an influential impact on the next generated step and enhances the agent's alignment with the user's expectations.

To effectively address these challenges, the planning module relies on diverse feedback sources. As outlined in Figure 3, the core-agent engages with tools within its system boundaries, as well as entities outside its scope, such as external systems and humans. Consequently, interactions with these components can offer valuable feedback:

- **Human Feedback:**

  Human feedback may be an essential source of information for aligning the planning module with human values and preferences. This feedback results from direct interactions between the core-agent and humans. For example, when the core-agent proposes a plan, humans may provide feedback on its appropriateness, effectiveness, or ethical implications. This feedback could come in various forms, such as ratings, or comments.

- **Tool Feedback:**

  The core-agent often utilizes various tools, which can be internal components of the system or external applications. For instance, an internal calculator tool might raise an exception upon receiving an illegal operation like division by zero. Likewise, external tools provide feedback in the form of error messages, or performance indicators. Indeed, if the core-agent uses a weather prediction remote API, the accuracy of the prediction serves as feedback. This tool-provided feedback helps the core-agent refine its tool selection and usage strategies.

- **Sibling Core-Agent Feedback:**

  In multi-core agent systems, as will be discussed in section 3.4, feedback from sibling core-agents becomes a valuable source of information. This type of feedback results from interactions and information exchanges between different core-agents within the same system. This intra-agent feedback can include shared observations, alternative perspectives on a problem, or evaluations of proposed plans. Such feedback promotes collaborative problem-solving and allows for cross-validation of plans. It enhances the overall robustness of multi-core agent systems by facilitating collective intelligence.

To conclude, the planning module is a critical component of our framework, employing a structured planning process that consists of task decomposition and plan generation steps. This process is distinct from, yet closely intertwined with, the planning strategies—single-path and multi-path—which guide the formulation of plans. These strategies are crucial as they shape the core-agent's approach to problem-solving, influencing both the quality and efficiency of the solutions. By breaking down tasks and generating multiple plans, the core-agent can identify optimal solutions more quickly and effectively.

The planning module's effectiveness is further enhanced by incorporating feedback from various sources, including humans, tools, and sibling core-agents. This feedback loop allows for continuous refinement and adaptation of plans, ensuring that the agent remains responsive and efficient. Moreover, incorporating planning strategies and feedback mechanisms enhances adaptability, enabling the core-agent to address a wide range of scenarios, from simple tasks to complex challenges. Such strategic diversity equips the system with greater intelligence and versatility.

It is important to note that the planning module does not operate in isolation. It collaborates closely with other modules, particularly the memory module, which will be discussed in the next section. This collaboration, especially in the context of memory-augmented planning, further enhances the core-agent's capabilities by leveraging stored information and past experiences.

3.2.2 Memory Module

The memory module is responsible for the storage and retrieval of information pertinent to the core-agent's activities, thereby enhancing the core-agent's decision-making efficiency and task execution capabilities. In **[26]** and **[4]**, the memory module in an LLM-based agent was approached from an abstract functional perspective, neglecting its analysis from a software architectural viewpoint. This led to some overlap between the memory module and the other modules defined in the framework suggested by **[4]**. Consequently, we propose a more comprehensive and well-defined presentation of this module based on three perspectives: Memory Structure, Memory Location, and Memory Format.

The framework presented in the survey **[4]** classified the memory structure into Unified Memory, intended to emulate human short-term memory via in-context learning, and Hybrid Memory, which represents both short-term and long-term memory functionalities. While these names aimed to differentiate types of agent systems, they deviate unnecessarily from the well-established terminology in the field. A more conventional and widely recognized categorization is introduced through the first perspective, Memory Structure, which includes short-term and long-term memory, aligning more with the human memory structure as established by **[27]**.

As opposite to how it was defined in framework **[4]**, our short-term memory definition diverges to focus more on the impact of the core-agent rather than on the LLM. From a content perspective, various sources of information contribute to short-term memory. As pointed out in paper **[26]**, data can be derived from one trial of a given task or from previous trials of the same task, which is referred to as short-term memory. This data has a narrow scope that focuses on a specific task and primarily communicated to the LLM for the purpose of in-context learning and enhancing the LLM capability with more information related to the task at hand. Conversely, long-term memory refers to the ability to store and recall information over extended periods, beyond the scope of a specific task. This type of memory enables the core-agent to maintain coherence and context over prolonged interactions, learning and adapting from past experiences. Namely, MemoryBank stores all interactions with user in a large symbolic memory and processes experiences into high-level summaries to reflect upon future similar tasks **[28]**.

However, there is always a limit to the amount of memory a core-agent can retain. Therefore, techniques such as forgetting mechanisms must be implemented to decide which memories to discard and which to retain **[26]**. As mentioned earlier, survey **[4]** treats memory from an abstract functional aspect, which is inconsistent with our framework that emphasizes a clear delineation of distinct modules. Our approach aims to provide a more precise and structured understanding of core-agent components from a software perspective. For instance, according to survey **[4]**, the memory module includes a "reflection" operation with a cognitive aspect that we believe belongs to the planning module. The planning module focuses on achieving the agent's goals and is responsible for decision-making and synchronizing among all other modules. Thus, it makes it more suitable for handling reflections and memory optimization by collaborating with the memory module to access stored data. Likewise, extracting useful information from memory and reflecting upon it to produce a solid plan belongs to planning module responsibilities. For example, Voyager **[29]** considers environmental feedback, handled as short-term information, and leverages the LLM capability to adjust its plan and make more efficient and rational decisions in the scope of planning module.

Furthermore, the writing operation is an essential aspect of a memory, distinguishing it from data repositories. Therefore, we exclude knowledge repositories from the memory category. As a matter of fact, the retrieval of knowledge from databanks, in a read-only way, falls under the responsibilities of the action module, which will be discussed in section 3.2.4. As we have set writing and reading as essential operations for the memory module and excluded reflection from the possible set of operations, we decided to omit the Memory Operation perspective discussed in survey **[4]** from our framework.

Beyond focusing on the scope of the data, inside trial and across trials, as stated in **[26]**, we emphasize the location of the memory because it is more relevant from a software architecture point of view. Hence, we introduce the third perspective, Memory Location, which encompass two categories: Embedded Memory, internal to the core-agent boundaries, and Memory Extension, outside the core-agent boundaries, yet still within the boundaries of the agent system as depicted in Figure 3.

Finally, the last perspective, Memory Format, already present in both works **[4]** and **[26]**, focuses on the shape and the representation of the memory that could have diverse manifestations: natural language, embeddings, SQL databases, or structured lists. In fact, memory information such as agent behaviors and observations could be directly described using raw natural language, providing flexibility and retaining rich semantic information. Another solution would be to encode memory information into embedding vectors to enhance retrieval and reading efficiency. In addition, as exemplified by ChatDB **[30]**, databases could be used as memory holders to store memory information in a structured representation, allowing agents to manipulate memories efficiently and comprehensively using SQL queries. And finally, structured lists, as used in GITM **[31]** let core-agents save the sequential actions of subgoals in a structured way and organize memory information into hierarchical lists to convey semantic information concisely.

As an observation, the most common format is textual, nevertheless **[4]** notes that these formats are not mutually exclusive, so one core-agent can handle multiple formats, such as GITM's key-value list structure that combines embedding vectors and raw natural language, to harness the respective benefits of each approach.

3.2.3 Profile Module

Similarly to the approach presented by the paper **[4]**, our framework defines the function of the profile module as to establish the role of the LLM and adopts more diverse methods. This template explicitly separates the roles of both the core-agent and the LLM. Indeed, the profile module facilitates the dynamic adaptation of various profiles tailored to specific use cases and strategies employed by the planning module.

The Profile module features four methods for defining profiles: the Handcrafted In-Context Learning Method, the LLM-generation method, the Dataset Alignment Method, and the newly introduced Fine-tuned Pluggable Modules method.

The Handcrafted In-Context Learning Method, previously referred to as the Handcrafting Method in the survey **[4]**, involves deputing the core-agent to set the LLMs profiles through in-context learning techniques that employ pre-configured prompts. This method allows for fine-grained control over the LLM's personality and behavior. While it is a straightforward method to implement, it necessitates the use of LLMs that are well-suited for in-context learning and often possess a cumbersome size.

The LLM-Generation Method facilitates the automatic creation of profiles for agents using LLMs. This method begins by specifying the profile's characteristics which include detailed information such as age, gender, and interests. Optionally, several seed agent profiles can be selected to serve as few-shot examples, as outlined in the paper **[4]**. Once these seed profiles are established, LLMs are employed to generate additional profiles by referring to the initial seed examples. For instance, RecAgent **[32]** suggests designing appropriate prompts that encourage a GPT model to generate comprehensive profile descriptions by referring to a table of attributes corresponding to various samples and generate additional profiles. The LLM-Generation Method, while offering significant time-saving advantages, is constrained by its reliance on LLMs. This dependence can lead to potential biases **[33]** or inaccuracies in generated agent profiles.

Additionally, **the Dataset Alignment Method** derives profiles from real-world datasets **[4]**, which consists of data about actual individuals. This approach starts by organizing the information in these datasets into natural language prompts that describe the characteristics of the role of the LLM. These structured data are then used to create the profiles. In the study conducted by **[34]**, researchers utilized GPT-3 alongside real world demographic data from ANES to assign roles based on characteristics such as state of residence. Subsequently, they evaluated whether GPT-3 could mimic real human behavior reliably. The Dataset Alignment Method ensures that the LLM profile accurately reflects real-world attributes and behaviors, thereby making it meaningful and realistic. However, the effectiveness of this method relies heavily on the accuracy and representativeness of the underlying real-world datasets.

Finally, we introduced **the Fine-tuned Pluggable Modules Method**, a pioneering solution to set LLM's profile leveraging several state-of-the-art techniques, aiming to provide an efficient customization and adaptation of the LLM's profile. This approach defines the LLM profile by injecting a pluggable module fine-tuned to influence the language model behavior. Indeed, that module is a set of additional tunable parameters that must be previously trained using Parameter-Efficient Fine-Tuning (PEFT) techniques, such as Sequential adapter (AdapterS), Prompt-tuning, or LoRA **[35]**.

This process ensures precise and effective customization to achieve the desired profile by eliminating the need for in-context learning, which traditionally involves injecting extensive prompt information into each query. By omitting this step, the required context size is significantly reduced, as the adapters directly encode the desired behaviors and profiles into the model parameters. This reduction in context size and memory footprint allows the LLM to operate more efficiently, using less memory during inference without compromising performance.

3.2.4 Action Module

While we have enriched the approaches applied in this module, we define it within the LLM-Agent-UMF similarly to the methodology described in **[4]**. This module serves as the convergence point where contributions from all other modules and their latent inputs are visualized as outcomes. It translates the agent's decisions into executable actions, conceptualized from four perspectives: Action Goal, Action Production, Action Space, and Action Impact.

The first perspective, the **Action Goal**, represents what the core-agent intends to accomplish by performing actions based on various objectives such as Task Completion, Communication, or Environment Exploration. The Action module maintains this Action Goal and guides the core-agent in selecting appropriate actions accordingly. The second aspect, **Action Production**, focuses on how the actions are produced and the catalysts behind them. In fact, the core-agent can produce actions through three primary methods: Action via Memory Recollection, Action via Plan Following, and Action via API Call Request, newly introduced in our framework.

In the case of **Action via Memory Recollection**, the core-agent generates actions by extracting relevant information from its memory based on the current task. For instance, in GITM, if the agent needs to achieve a sub-goal, it checks its memory for similar past experiences and uses successful actions from those experiences to handle the current task **[31]**.

Similarly, through the **Action via Plan Following** method, the core-agent can execute actions by adhering to pre-generated plans. In GITM and DEPS (Describe, Explain, Plan, and Select) **[36]**, the agent breaks down the task into subtasks and sub-procedures, acting sequentially to achieve each procedure and complete the overall task. In case of plan failure, both agents have replanning capabilities, so the action module executes the new plan. These two methods, Action via Memory Recollection and Action via Plan Following, align with the framework described in **[4]**.

By exploring other state-of-the-art techniques like TALM **[5]** and ToolFormer **[37]** that exemplify LLMs' capacity to improve performance across diverse tasks through incorporating external tools, we introduce a new method named **Action via API Call Request**. This novel approach empowers the core-agent to execute actions in response to API call requests initiated by the LLM, facilitating smooth integration and effective utilization of external resources.

The third perspective, **Action Space**, defines the set of possible actions that can be performed by the core-agent, remaining consistent with the original framework **[4]**. These actions can leverage internal knowledge, where the core-agent acts on internal information, and/or utilizes tools when comprehensive expert knowledge is required. These tools can encompass APIs such as HuggingGPT, which leverages AI models via the HuggingFace API to accomplish complex user tasks **[38]**. Alternatively, the core-agent can rely not just on its memory but extend its operational scope and knowledge by communicating with other read-only data sources like databases and knowledge repositories. It is useful to note here that data repositories can have the same formats discussed in the memory module (section 3.2.2). For instance, LLamaIndex **[39]** stores data as vector embeddings at the indexing stage to leverage semantic search, where the similarity between embeddings is used to rank documents by their relevance to a query. In contrast, ReAct **[40]** uses a textual data repository, like Wikipedia, to mitigate error propagation in chain-of-thought reasoning.

Finally, **Action impact** refers to the consequences resulting from an action. Numerous impacts can be cited, such as changing the environment by moving to different locations, gathering resources, or constructing buildings **[41]**. Actions can also alter the internal state of the core-agent by updating its memory or acquiring new knowledge which may affect the planning module. Additionally, impacts can trigger new actions, creating a chain of actions during task completion.

3.2.5 Security Module

As LLMs continue to advance and being implemented in widespread application, addressing the potential risks and unintended consequences associated with their use becomes imperative. These concerns are mainly around unauthorized and unethical use, data biases and privacy **[3]** . This has led to the introduction of guardrails recently in LLMs field as algorithms to identify and prevent the misuse of LLMs **[12]**. To enhance our framework, we propose a fifth module: the Security Module. This module aims to provide a more capable and responsible core-agent. Its role is monitoring the action module specifically in production environments to ensure the safety and responsible use of LLMs.

The Security Module operates within the parameters of the Confidentiality, Integrity, Availability (CIA) triad **[42]**, a crucial model which encompasses three pivotal principles in the security field. Confidentiality is centered around protecting sensitive information from unauthorized access or disclosure. Within LLMs-based agents, this principle is critical in safeguarding user data and ensuring the non-divulgence of sensitive information. Integrity is concerned with the accuracy, consistency, and trustworthiness of data throughout its lifecycle. For agents, this principle involves maintaining the reliability of the model's outputs and preventing any unauthorized modifications to the system or its data. Lastly, availability focuses on ensuring that information and resources are accessible to authorized users whenever required. In the context of LLM applications, this entails maintaining system uptime and providing safe responses to user inquiries. By adhering to these principles, the Security Module aims to establish a robust and trustworthy environment for the operation of the agent, effectively addressing critical concerns related to their deployment and usage.

While conducting a thorough research around the Security Module, our approach will involve exploring multiple facets: Identifying and securing critical assets and data within the core-agent modeling framework, implementing strategies and mechanisms to protect these assets from potential threats, ensuring the core-agent's ability to effectively respond to and mitigate any security incidents, and maintaining the privacy of user data while adhering to relevant data protection regulations.

Security measures:

Regardless of the guardrail type deployed, the Security Module encompasses three fundamental axes: Prompt safeguarding, response safeguarding, and data privacy safeguarding.

- **Prompt Safeguarding:**
  Prompt safeguarding necessitates employing measures to detect and mitigate unauthorized access to Large Language Models through prompt injection attacks **[43]**. Techniques for enhancing the security of LLM-based agents can be integrated directly into the LLM itself, with Adversarial Training (AT) being a prominent example **[44]** AT enhances an LLM's defense mechanisms by fine-tuning it with augmented training data containing adversarial examples, thereby increasing the model's ability to safeguard against malicious prompts and improving its robustness. However, AT faces significant limitations, such as the challenges in efficiently selecting adversarial examples and the model's exposure to adversarial perturbations such as HOUYI **[45]**, a black-box prompt injection attack. Moreover, such training-based security techniques may impact the generative performance of the LLM which necessitates additional evaluation steps.

  Other techniques address these limitations by decoupling the security measures from the LLM and delegating them to a distinct entity that we identify as a core-agent supplemented with a security module as illustrated in Figure 3. This decoupling is essential for improved protection and enables the implementation of more advanced security protocols on the LLM input, which can evolve independently of the LLM's training process. An example of this approach is Nvidia NeMo **[46]**, which functions as an intermediary layer between users and LLMs, employing advanced techniques such as vector databases and comparison with stored canonical forms to filter and process user inputs before they reach the model, thereby providing robust prompt safeguarding without directly modifying the underlying LLM.

  These approaches are essential to address scalability challenges, enable proactive defense, and facilitate continuous learning in LLM security, ensuring that protection mechanisms can adapt to new threats and maintain the integrity of LLM interactions.
- **Response safeguarding:**
  The recent survey **[47]** has demonstrated that despite the implementation of prompt safeguarding techniques, the overall resilience of Large Language Models (LLMs) against advanced attacks, known as jailbreaks, may not experience significant improvement. These jailbreaks are designed to exploit biases or vulnerabilities within language models by manipulating their responses. Notable examples include white-box attacks AutoDAN-Zhu **[48]**, which generate stealthy prompts to avoid triggering the model protective mechanisms. Additionally, black-box attacks leverage manually crafted prompts to deceive the LLM **[47]**. The effectiveness of these

jailbreak techniques is further illustrated in **[12]**, where researchers successfully achieved a jailbreak attack on ChatGPT 3.5 by framing potentially harmful query, "how to hotwire a car" as a hypothetical scenario. The existence of these jailbreak methods highlights the urgent necessity for continuous and rigorous monitoring of LLM outputs to detect and mitigate potential breaches.

Indeed, guarding the agent outputs involves ensuring the security and integrity of the text generated by LLMs. This includes detecting and redacting harmful content while maintaining coherence and relevance. Two notable examples of such techniques are Guardrails AI **[47]** and LLMSafeGuard **[49]**, discussed further in section 4.

- **Data Privacy Safeguarding:**

  Lastly, safeguarding data privacy is pivotal, especially when handling sensitive or personal information. It is essential to ensure that LLMs are protected against sensitive data breaches **[47]**. Existing research has predominantly focused on securing training data through traditional techniques such as Differential Privacy **[50]** and watermarking **[51]**. As a matter of fact, Differential Privacy tuned models add noise to the data, making it difficult to identify individual data points. Similarly, watermarking techniques embed identifiable markers into LLM outputs, allowing for the tracing of data origin and preventing unauthorized use.

  However, under our proposed framework, the primary objective shifts from solely protecting the privacy of training data to ensuring that the LLM does not divulge sensitive information to external tools. This approach aims to maintain the privacy and security of data while interacting with other systems. Indeed, as previously mentioned, the core-agent can leverage external tools, APIs, and knowledge repositories to augment the capabilities of the LLM. Being part of the agent as a whole system, internal tools are part of the privacy circle, thereby, there is no need to apply security measures on communication procedures. However, the use of external resources introduces potential risks that require specific attention. These risks include a lack of robust data privacy measures, potentially leading to data leaks or unauthorized access. For example, when the core-agent utilizes third-party services to access additional information or perform specific tasks, the data transmitted may contain sensitive details that specific systems are not authorized to access. Additionally, such sensitive information could be intercepted or mishandled if proper secure channels are not leveraged.

  To mitigate these risks, the security module within the core-agent must implement a range of powerful techniques such as access control mechanisms, and data encryption. Therefore, the framework ensures that interactions with external resources maintain the highest standards of security and data privacy.

Thus, the security module operates along three fundamental axes: Prompt Safeguarding, Response Safeguarding, and Data Privacy Safeguarding. Their integration forms a robust defense mechanism capable of mitigating diverse threats, ranging from prompt injection attacks to potential data breaches. Through this approach, the security module shapes the behavior of the core-agent, prioritizing security in every aspect, from data retrieval to external communication.

Guardrail types:

To implement these security axes effectively, various guardrail methodologies have been developed. These guardrails act as the operational layer of the security module, translating high-level security objectives into actionable safeguards. The paper **[47]** delves into diverse guardrail methodologies and solutions offered by LLM service providers and the open-source community. Through meticulous analysis of these methodologies, two primary types emerge, rule-based guardrails and LLM-based guardrails.

- **Rule-based guardrails:**

  These guardrails operate based on a predetermined set of rules and regulations aimed at screening and preventing potentially detrimental or undesirable inputs/outputs from LLMs. To elucidate the process, users define the content necessitating protection. Subsequently, the guardrails assess the inputs/outputs against these predefined regulations, and custom rules **[47]**, to ascertain compliance. In instances where the content is deemed unsafe, it may be obstructed, or a cautionary alert may be issued. For instance, the Adversarial Robustness Toolbox (ART) **[52]** is specifically designed to bolster the security and robustness of models against adversarial attacks. It offers tools and methods to defend against and adapt to malicious inputs, thereby safeguarding AI applications from potential vulnerabilities.

- **LLM-powered guardrails:**

  While rule-based guardrails provide a solid foundation for safeguarding LLM operations, they face limitations in adaptability and maintenance. The need for manual, continuous improvement and intervention to upgrade rules can be time-consuming and may struggle to keep pace with rapidly evolving threats and diverse use cases. LLM-powered guardrails offer a compelling solution to these challenges. A prevalent design approach for constructing these guardrails involves the usage of neural-symbolic agents **[47]**. These agents, functioning akin to core-agents from a security standpoint, undertake the critical task of analyzing input and output, ensuring their adherence to a predefined set of requirements. By leveraging the inherent learning and adaptability capabilities of language models, these guardrails can evolve and respond to new situations in a faster and more automated manner. They can understand context, nuance, and intent more effectively than rigid rule sets, allowing more sophisticated and flexible protection mechanisms.

Moreover, neural-symbolic agents resolve conflicts that may arise between requirements, leverage historical data to reason symbolically and possess the capability to collaborate with other AI systems **[47]**. While LLM-based solutions may introduce computational overhead, this potential drawback can be mitigated by adopting lightweight models specifically designed for guardrail tasks. These optimized models can provide the benefits of LLM-powered security with reduced resource demands, striking a balance between robust protection and operational efficiency.

In fact, within the general scope of the agent, the core-agent can communicate with an auxiliary LLM, which is finetuned on specialized dataset to set the acceptability guidelines of the response generated by the main LLM. The core-agent allows for customization of guardrail rules, including monitoring and enforcement protocols **[12]**. These customized rules are then passed to the auxiliary LLM to classify the nature of the input. Such classification helps the core-agent to decide whether the requirements are fulfilled or not. A leading example of this approach is LLaMA Guard **[12]**. Introduced by Meta (Facebook), it was designed specifically to guarantee the security and reasonable utilization of LLaMA models and used to analyze both input and output data. It employs predictive classification techniques to assess and improve security across user-specified categories. This implementation underscores the critical role of LLM-based guardrails in fortifying the integrity and reliability of AI systems. By leveraging the LLM's capabilities to understand and enforce complex security rules, Llama Guard provides a flexible and powerful mechanism for ensuring safe and responsible AI operation, particularly in next-generation LLM models like Llama 3.1 **[53]**.

To wrap up the LLM-based agent unified modeling framework, our proposed solution emulates the modular architecture of the human brain by introducing the core-agent component, which encompasses five internal modules: planning, memory, profile, action, and security. This modular design introduces enhancements on the security level and addresses challenges related to extendibility and maintainability, effectively separating the core-agent functionalities from the LLM. Consequently, this structured aspect highlights the role of each module and the implication of their integration, especially the planning module as it gives core-agents adopting it an authoritative aspect. The latter perspective leads to the classification of core-agents that will be discussed in Section 3.3 .

3.3 Active/Passive Core-Agent Classification

As discussed in the preceding sections, the core-agent represents a distinct entity within the LLM-Agent-UMF. While the LLM excels in cognitive tasks such as understanding, reasoning, and generating responses, it lacks the capability to directly interact with the environment or external tools. This is where the core-agent plays a crucial role. It bridges the gap between the LLM's cognitive abilities and the need to engage with external sources, enabling seamless integration with various tools and systems. The core-agent is thus characterized by its action capabilities and its ability to respond to user requests through interaction with these diverse tools. In fact, ToolLLM **[54]** is a general tool-use framework that enhances LLMs capabilities enabling agents to use external tools and APIs. It uses a neural API retriever to recommend appropriate APIs for each instruction. Then they employ a depth-first search-based decision tree algorithm to evaluate multiple reasoning traces and expand the search space. Consequently, it enhances the planning ability of the retriever and empowers the finetuned LLM, ToolLlaMA, to generate adequate instructions. The retriever here, in association with the search-based decision tree algorithm, satisfies our definition of a core-agent. In this case where the LLM-based agent conducts cognitive tasks, memory and planning modules are essential in the core-agent to ensure reasoning capabilities because they enable the agent to retain and recall past experiences, plan and synchronize actions, reason and make decisions **[26] [21]**.

In other cases, such as Toolformer **[37]**, we identify entities that fit our definition of a core-agent but lack both planning and memory modules. Indeed, Toolformer fine-tunes its LLM on function calling, enabling it to generate API requests within natural language as needed. Consequently, the LLM determines when to make an API call, which API to use, and how to integrate the results, while the actual execution of the API request is delegated to an entity that we identify as a core-agent. In this case, the planning module of the core-agent is obsolete because planning is handled solely by the LLM. Yet, its action module is present because it is still responsible for executing API calls systematically. For example, if the model suggests using a calculator API, the core-agent retrieves the arguments for the mathematical operation, performs the calculation, and returns the computed result to the LLM.

The inspection of the state-of-the-art led to the conclusion that the action module is always indispensable in a core-agent as it is responsible for producing the executive steps to achieve their goals. However, the architectural disparities in core-agents and the absence of some modules in some proposed agent systems highlight the need to introduce a new taxonomy classifying core-agents into two distinct categories: Active core-agents and passive core-agents. The following sections analyze and pinpoint the main differences and similarities between active and passive core-agents in their structural alignment with our framework.

3.3.1 Active Core-Agents

Active core-agents encompass all five modules described in section 3.2 and illustrated in Figure 3, but what differentiates an active from a passive core-agent is its managerial aspect. An active core-agent is characterized by its leading position in the agent as the orchestrator of other components, so naturally, it requires a planning module to divide tasks into subtasks and collaborates with the memory module to provide the necessary context, analyze information, and make decisions. Consequently, we consider an active core-agent to be stateful, meaning it can maintains information about its past interactions and states over time. This is facilitated by an adaptive memory that captures and stores various aspects of the agent's lifecycle, allowing it to use this historical data to inform future actions and decisions. The profile module role is emphasized in the active core-agent category, because it guides the LLM's behavior in a certain direction. Furthermore, the security module plays a prominent role in safeguarding the communication between the LLM and the human, ensuring a reliable exchange; Acting as an intermediary, the core-agent safeguards the LLM from threats such as jailbreak attempts and protects user data privacy by implementing safety measures as outlined in Table 1.

Throughout our research on the state of the art, we observed that LLM-based agents are recently built upon active core-agents performing tasks from planning to execution **[4]**. As highlighted in **[17]**, active core-agents are more effective because they incorporate planning and memory modules, which enable them to reason, plan, and execute tasks efficiently. This structure allows the agent to adapt to changing situations and make informed decisions, making the system more robust and capable.

However, relying solely on active core-agents would increase the complexity of the agent, which can lead to scalability issues and negatively impacts the maintainability of the agent as it will hinder and complicate future improvement efforts. As noted in **[17]**, "the complexity of the agent system grows exponentially with the number of tasks it needs to perform". Therefore, rather than centralizing responsibilities on one entity, it would be more beneficial and adhering to the Single Responsibility Principle, if we leverage other core-agents to granulate task execution and reduce the complexity of the agent system. This approach is supported by the concept of "separation of concerns" in software engineering, which emphasizes the importance of dividing responsibilities among multiple components to improve system modularity and maintainability. By distributing tasks among multiple core-agents, we can reduce the cognitive load on individual core-agents, improve system efficiency, and enhance overall performance.

3.3.2 Passive Core-Agents

Passive core-agents are employed when LLMs cover all cognitive tasks of the agent such as planning and taking decisions, while passive core-agent's role is mainly to execute specific procedures. As a direct consequence, the planning module becomes unnecessary and likewise the memory needed in reasoning. Unlike active core-agents, passive core-agents are stateless, and the short-term memory is handled by the LLM, covering only the current task's state. In LLM-based agents, passive core-agents, which always follow instructions from domain-specific LLMs, lack the ability to control the profile of the LLM thus do not possess a Profile module. The LLM profile may be statically defined during the system setup or dynamically defined by another entity, which will be discussed in the next section.

The most essential module in a passive core-agent is the action module. Our framework posits that the function of a passive core-agent is limited to specific task execution. Actions are often triggered by API call requests, which are not decision-based nor self-generated by the passive core-agent but provided by another entity (e.g., LLM or an active core-agent) as shown in Figure 4. The actions do not alter the internal state of the agent or change a predetermined plan. This again points to the absence of a planning module in passive core-agents. Furthermore, we introduce another distinction between passive and active core-agents: the communication between humans and core-agents is interactive and bidirectional in both categories, aiming to gather information and/or feedback. However, as pointed out by Figure 4, the communication between passive core-agents and humans can only be initiated from the passive core-agent part, unlike active core-agents, where communication can be initiated by either party.

Despite not being directly responsible for handling prompts from humans and providing generated text responses, passive core-agents should still possess a robust security module. This component is crucial in ensuring privacy during their interactions with other humans or third-party systems by preventing leakage of sensitive data while minimizing potential threats and breaches. Consequently, this bolsters the overall trustworthiness and reliability of LLM-based agent applications. It is also important to note that in this setup, as illustrated in Figure 4, the LLM ensures by itself the safety of the prompts by implementing one of the mechanisms previously discussed in section 3.2.5 such as adversarial training.

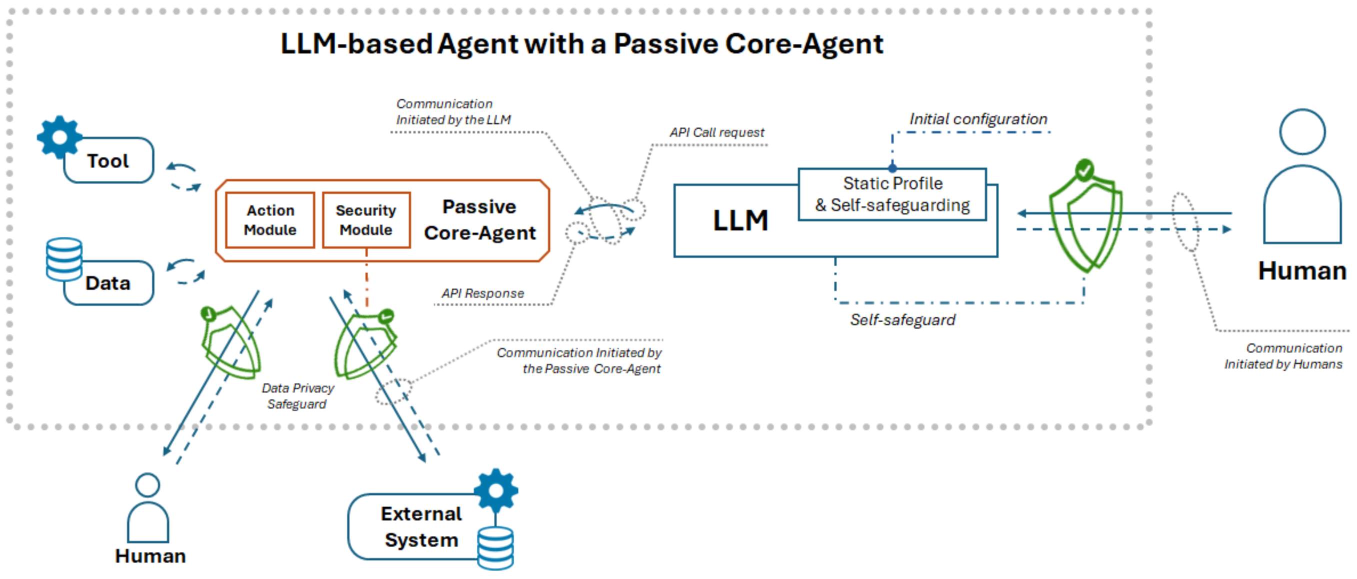

LLM-based Agent with a Passive Core-Agent
Communication Initiated by the LLM
API Call request
Initial configuration
Tool
Action Module
Security Module
Passive Core-Agent
Static Profile & Self-safeguarding
LLM
Data
API Response
Self-safeguard
Human
Communication Initiated by the Passive Core-Agent
Communication Initiated by Humans
Data Privacy Safeguard
Human
External System

Figure 4: LLM-based agent architecture including a passive core-agent

Describing the characteristics of both active and passive core-agent classes results in constructing a well-structured summary, table 1, that encapsulates the distinct modules, their respective sub-modules, and the underlying methods for each category. This enables a comprehensive understanding of their functional differences and similarities within the context of LLM-based agents.

In this section, we detailed the construction of passive and active core-agents, emphasizing their architectural design. This analysis allowed us to identify their utilities and limitations, which are detailed in table 2. Both categories improve modularity, reduce system complexity, and possess multi-tasking capabilities. Passive core-agents reinforce the single responsibility principle and imply simple implementation based only on action and security modules, enhancing reusability and offering straightforward integration into multi core-agents' setups with minimal synchronization requirements which will be discussed in the section 3.4. However, their simplicity limits their ability to handle complex tasks, precludes human-initiated communication, and lacks memory. Thus, it restricts visibility into the agent's overall status and contextual data access, which is only available via API call requests. Additionally, they have no control over the LLM profile.

In contrast, active core-agents enhance LLM planning and memory capabilities, making them suitable for handling complex tasks. They can access memory and contextual data, control dynamically LLM's profile, and break down complex tasks into manageable subtasks. Despite these advantages, active core-agents require complex implementation involving extra modules compared to passive core-agent and intricate synchronization in multi core-agent' setups, which will be detailed in section 3.4.

Table 1: Active and passive core-agent internal structure

Core-Agent Structure			Active	Passive
Modules	Sub-modules / Methods		Core-Agent	Core-Agent
Profile	Handcrafted in-context learning method		X	[*]
	Fine-tuned pluggable modules method		X	[*]
	LLM-generation method		X	
	Dataset alignment method		X	
Memory	Memory location	Embedded memory	X	
		Memory extension		
	Memory structure	Short-term		
		Long-term		
	Memory format	Natural language		
		SQL database		
		Embeddings		
		Structured list		
Planning	Planning techniques	Rule-based	X	
		LM-powered		
	Planning process	Task decomposition		
		Plan generation		
	Planning strategies	Single-path strategy		
		Multi-path strategy		
	Feedback sources	Sibling core-agent feedback		
		Human feedback		
		Tools feedback		
Action	Action goals	Task completion	X	X
		Communication	X [**]	X [***]
		Environment Exploration	X	
	Action production	Action via Memory Recollection	X	
		Action via Plan Following	X	X
		Action via API Call Request	X	X
	Action space	Internal knowledge	X	X
		Tools (APIs, Data repositories, External systems, etc)	X	X
	Action impact	Change environment	X	X
		Alter internal state	X	
		Trigger new actions	X	X
Security	Safeguarding measures	Prompt Safeguarding	X	
		Response Safeguarding	X	
		Data Privacy Safeguarding	X	X
	Guardrail types	Rule-based guardrail	X	X
		LLM-powered guardrail		

[*] Passive core-agents do not have a profile module. Depending on the architecture of the whole agent, the LLM' s profile will be set either statically or dynamically, but not by the passive core-agent as it has no control over it.
[**] Communication can be initiated by either Humans or the active core-agent.
[***] Communication can only be initiated by the passive core-agent.

Table 2: Advantages and disadvantages of Active and Passive Core-Agents

Core-agents' advantages and disadvantages		Active Core-Agent	Passive Core-Agent
Advantages	Reinforce Single Responsibility Principle (SRP).		X
	Imply simple implementation based on two modules.		X
	Improve modularity and reduce complexity of the system.	X	X
	Improve component reusability.	X	X
	Improve composability and integration in multi core-agents' setups without any (or with minor) synchronization.		X
	Enhance LLM planning and memory capabilities.	X	
	Handle complex tasks.	X	
	Access to memory and contextual data.	X	
	Possess flexible profile that can be adapted dynamically.	X	
	Break down complex tasks into subtasks.	X	
	Possess multi-tasking capabilities.	X	X
	Protects against adversarial attacks.	X	
Disadvantages	Imply complex implementation with multiple modules.	X	
	Synchronization is needed in multi core-agents' setups.	X	
	Handle tasks with limited complexity.		X
	Preclude human-initiated communication.		X
	Lacks memory, limiting visibility into the agent's overall status.		X

3.4 Multi Active/Passive Core-Agent Architecture

Handling complex tasks often necessitates the use of multiple agents, as a single agent may not possess the requisite capabilities or expertise to tackle diverse domains. However, LLM-based multi-agent systems face considerable challenges, including scalability, integration, management of inter-agent relationships, and ensuring interpretability in managing intricate tasks **[55]**. In some instances, implementing a multi-agent system may be unnecessary, as their aforementioned complexities and drawbacks can be circumvented with a multi-core agent system. A single-agent system can potentially accommodate multiple core-agents, each dedicated to distinct tasks such as systematic execution or complex management. This idea leads us to propose a pioneering Multi Active/Passive Core-Agent Architecture.

To achieve the effective distribution of responsibilities and manage the workload within the agent system, we must propose an efficient classification of multi-core agent architectures. Our framework classifies multi-core agents into two primary categories: uniform and hybrid.

3.4.1 Uniform multi-core agent

Uniform multi-core agents are exclusively based either on active core-agents or passive core-agents, unlike hybrid multi-core agents that integrate both active and passive core-agents within a single system.

- **Uniform multi-passive core-agent architecture** leverages passive core-agents' capabilities of handling low-level operations and executing specific tasks. The configuration shown in Figure 5 is an example where the LLM communicates with multiple passive core-agents and harnesses strategically their singular strengths to retrieve diverse information or perform specialized functions to generate a comprehensive final output. In fact, the language model assumes leadership and complete control over the ensemble of passive core-agents.

  In essence, uniform multi-passive core-agent systems are distinguished by the ease of integration of new passive core-agents, thereby extending their functionality without the need for complex synchronization. Consequently, the only modification resulting from the introduction of new passive core-agents involves adapting the LLM profile, either statically at setup/configuration time or dynamically via an active core-agent, as will be discussed in the context of the hybrid setup.

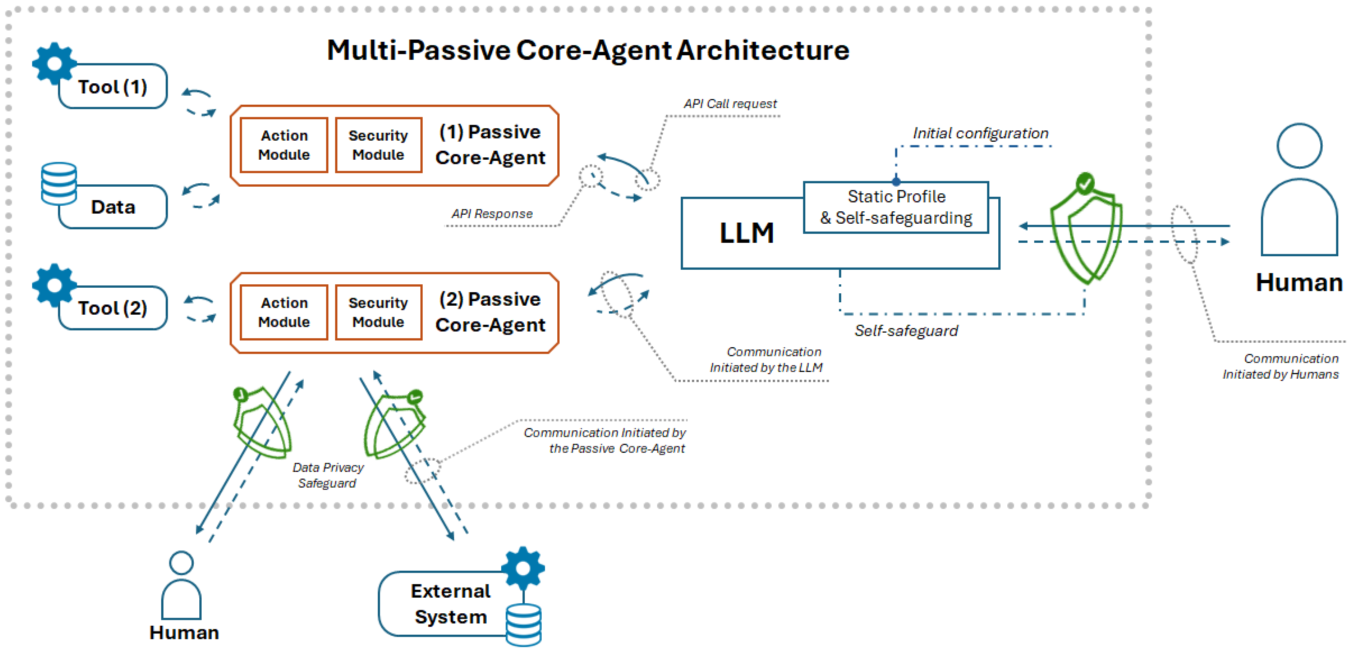


Figure 5: Multi-Passive Core-Agent Architecture

- **Uniform active core-agents architecture** deals with the interaction of a group of active core-agents in one system as illustrated in Figure 6. As opposite to passive core-agents that operate solely with action and security modules, active core-agents possess all five modules (Planning, Memory, Profile, Action and Security) enabling them to manage complex cognitive tasks. This architecture may be seen as a better alternative to uniform multi-passive core-agents' design due to its wider range of capabilities and functionalities. However, due to the authoritative nature of the active entities, the multi-active core-agent design is more complex than the one exclusively based on passive core-agents. The inclusion of multiple active elements in one system introduces challenges similar to those in multi-agent systems. For instance, effective communication among active core-agents is paramount; given their dynamic nature, timely and accurate exchange of information _such as inter-sibling core-agent feedback and status updates_ is crucial to ensure cohesive operation of the agent. As the number of active core-agents grows, managing intra-communication becomes increasingly complex resulting in frequently emerging synchronization issues. This is why multi-active core-agent systems potentially necessitate consensus algorithms, such as Raft **[56] [57]**, to elect a leader.

  Therefore, we deduce that the independent implementation of either of these architectures is limited and may be problematic. While the multi-passive core-agents architecture is efficient in granular task execution and low-level operations, it lacks a component for handling high-level tasks such as decision-making, task planning, and resource allocation which are intrinsic to multi-active core-agent systems. However, the latter introduces synchronization issues and increases system complexity. This dilemma compels us to introduce the hybrid approach.

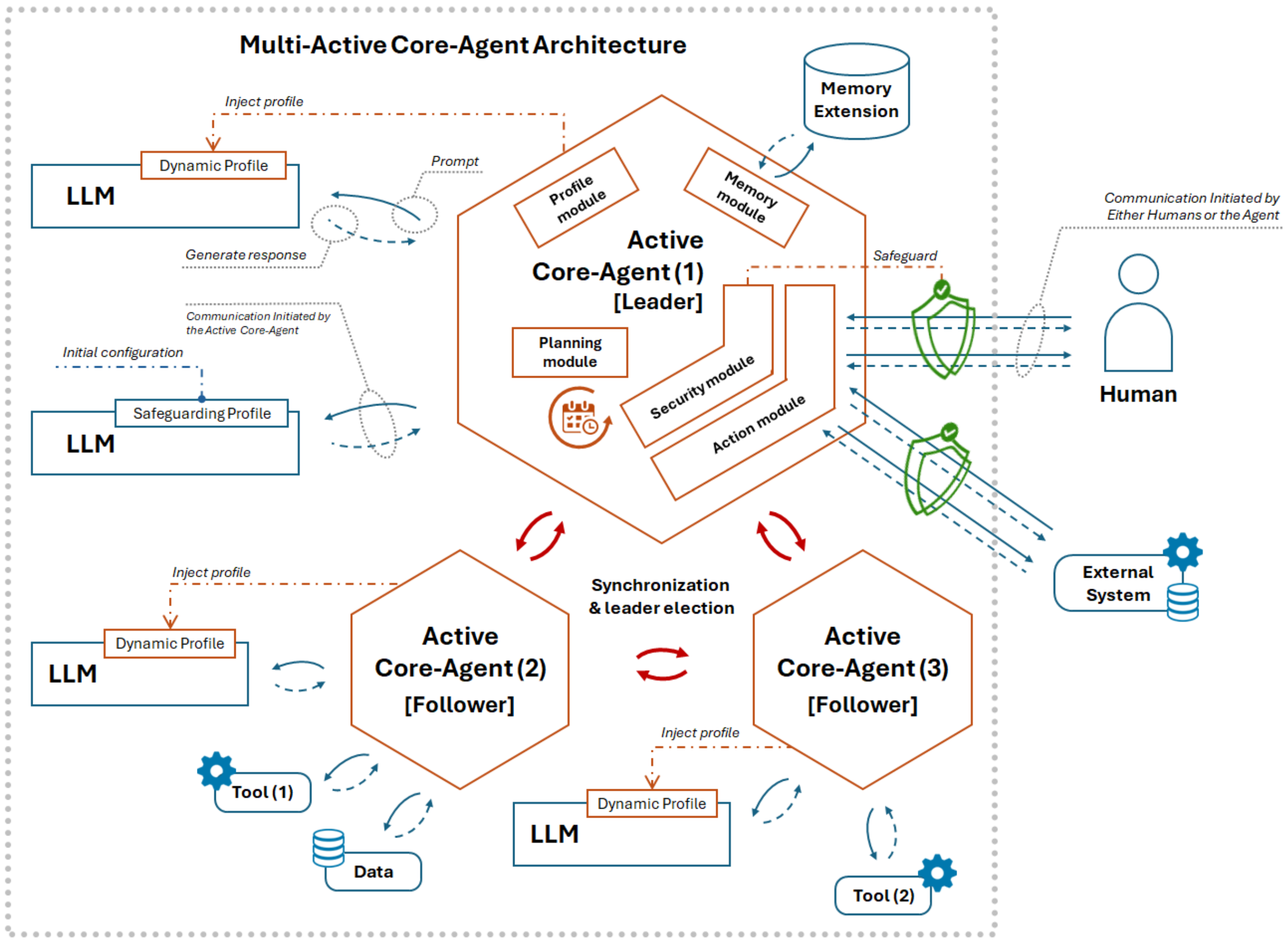


Figure 6: Multi-Active Core-Agent Architecture

3.4.2 Hybrid multi-core agent

To leverage the strength of both passive and active core-agent architectures, we propose an optimal system depicted in Figure 7. It integrates one active entity as the manager with multiple passive entities functioning as workers within a unified system. One managerial aspect of the active core-agent is its ability to configure dynamically the profile of LLMs, enabling them to effectively utilize passive core-agents for handling specific tasks. This configuration leverages the parallel execution capabilities of numerous passive core-agents under the guidance and leadership of an active core-agent, empowering the system to handle wider range of tasks, while preserving a comfortable level of flexibility, extendibility and scalability.

This hybrid design realizes the full potential of the multi-core architecture: by uniting the strengths of the uniform passive core-agent architecture with the capabilities of the active core-agent, the system can dynamically allocate resources and adjust its configuration based on the specific requirements of the task at hand. Our proposed architecture of one-active-many-passive strikes a balance between the intricate nature of multi-active core-agent architectures and the practicality offered by passive core-agents.

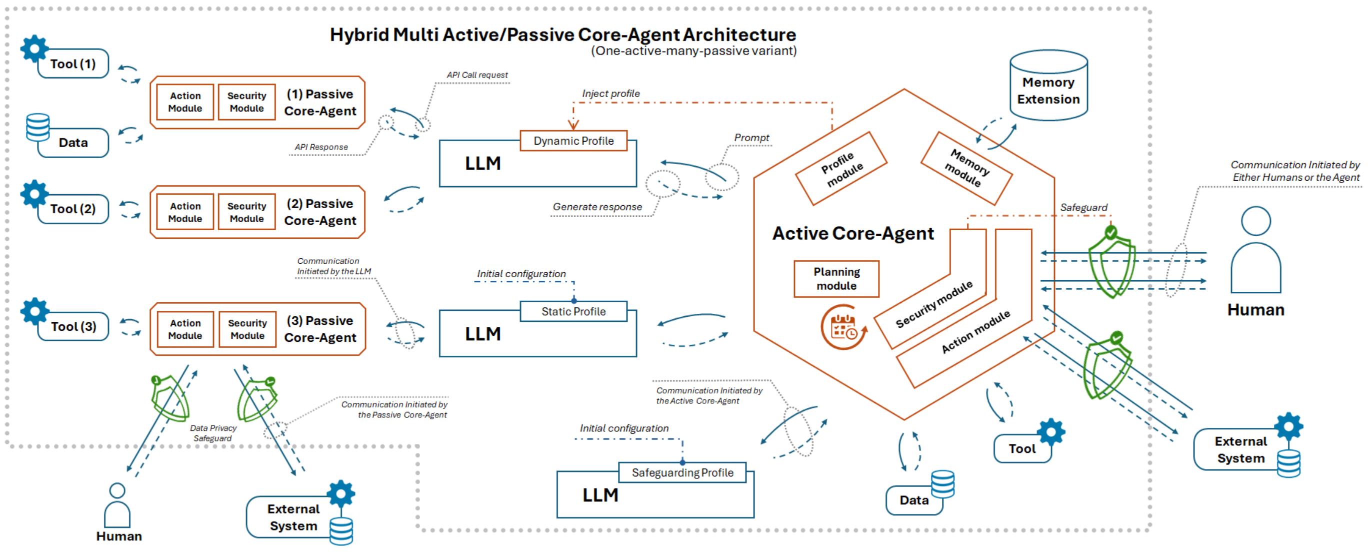


Figure 7: One-active-many-passive core-agent hybrid architecture

As a matter of fact, in scenarios characterized by dynamic environmental changes, the inclusion of multiple active core-agents becomes essential to uphold the resilience and adaptability of the agent. Naturally, given the complexities outlined earlier, the implementation of an agent based on a many-active-many-passive architecture, as illustrated in Figure 8, would be intricate especially on the level of synchronization between active core-agents. Such a system impels a meticulous design, emphasizing intra-agent interactions, adherence to communication protocols, delineation of tasks for each active core-agent, and error-handling strategies. Clearly, these challenges underscore the simplicity of our proposed one-active-many-passive architecture. Nevertheless, there remains a promising opportunity for further research, as the challenges posed by multi-active core-agents pave the way for advancing and refining our framework.

In conclusion, the modularity of core-agents contributes to guaranteeing the composability within the agent architecture from a software perspective. It facilitates the seamless integration of new passive core-agents within a single agent system as the system scales, obviating the need for a transition to a multi-agent system. Furthermore, this architecture tackles scalability and adaptability challenges by adhering to the Open/Closed Principle (OCP), enhancing core-agents' integration across evolving systems, and fostering robustness and flexibility. As outlined in the paper **[6]**, the expansion of the system necessitates dynamic scaling to accommodate growing demands and ensure optimal performance. This entails adaptive capabilities such as increasing the number of agents or utilizing larger LLMs. These challenges are effectively addressed by architectures based on multiple core-agents primarily due to the unitary role a core-agent plays within the agent system. In fact, our framework allows the active core-agent to dynamically incorporate or detach passive core-agents, as illustrated by the switch linking the leader active core-agent and the passive core-agent (2) in Figure 8. In the following section, we discuss the results of our work.

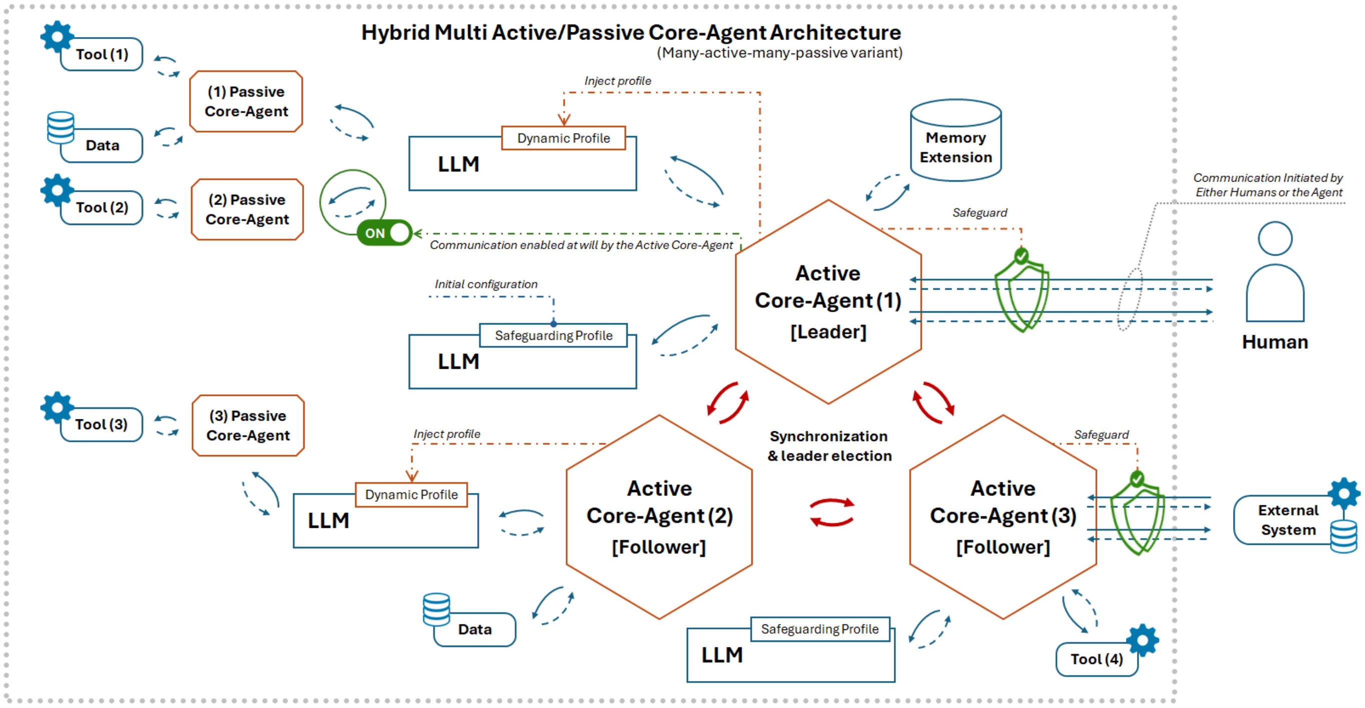

Hybrid Multi Active/Passive Core-Agent Architecture
(Many-active-many-passive variant)
Tool (1)
Data
Tool (2)
(1) Passive Core-Agent
(2) Passive Core-Agent
Inject profile
Dynamic Profile
LLM
Memory Extension
ON
Communication enabled at will by the Active Core-Agent
Initial configuration
Safeguarding Profile
LLM
Safeguard
Active Core-Agent (1) [Leader]
Communication Initiated by Either Humans or the Agent
Human
Tool (3)
(3) Passive Core-Agent
Inject profile
Dynamic Profile
LLM
Active Core-Agent (2) [Follower]
Synchronization & leader election
Safeguard
Active Core-Agent (3) [Follower]
External System
Data
Safeguarding Profile
LLM
Tool (4)

Figure 8: Many-active-many-passive core-agent hybrid architecture

4 Results and discussion

4.1 Evaluation of the new core-agent terminology

Agents have traditionally been discussed as a single monolithic unit, with different inner functional entities, such as LLMs and complementary software components, treated in an intertwined manner, lacking clear boundaries definition. This approach can lead to confusion when attempting to analyze or compare various systems that exhibit agent-like behaviors. One of the main reasons why agent definition remains misunderstood lies in the disparity between how researchers and practitioners use terminology to describe their work. For instance, some studies might not explicitly refer to their LLM-based system as an "agent", even though it exhibits characteristics commonly associated with agents such as autonomy, adaptability, and goal-directed behavior. The ToolLLM **[54]** research paper provides a prime example of this discrepancy. While the authors indeed utilize API retriever component in conjunction with their LLM, they refrain from using the term "agent." The same can be observed in the Toolformer **[37]** work, which implicitly incorporates software to execute API call requests generated by LLMs but similarly avoids referencing the system as an agent.

Introducing the core-agent term, to denote the central component within LLM-powered agents, and defining the role it plays, address these terminological ambiguities and facilitate more transparent discussions about such systems. For clear communication, as explained in **[58]**, one should avoid the use of different terms for one thing or a single term for different things. They suggest that establishing a well-defined language can help reduce confusion, promote consistency, and enhance communication among researchers and educators. These improvements will ultimately facilitate better collaboration, leading to accelerated development and optimization of these systems for a wide array of applications.

One terminological ambiguity we resolve in this work pertains to the concept of “memory”. For instance, ChatDB **[30]** illustrates a dual usage by defining memory in one context as the data retrieved by the agent from knowledge sources, while in other contexts, it refers to the space for storing processed data and information essential to the core-agent’s operations. To distinguish clearly between these usages, we clarify that the former definition pertains to a "data repository," whereas the latter accurately corresponds to "memory."

In addition to resolving ambiguities, based on the definition that we established for the “core-agent” component, we successfully identified core-agents within multiple LLM-based systems, mainly Toolformer and ToolLLM. On the one hand, Toolformer incorporates a simple-structured core-agent that includes only the action module responsible for API execution.

On the other hand, ToolLLM adopted a more sophisticated approach. In fact, we recognize the use of a core-agent encompassing two modules: an action module represented by the neural API retriever and an implicate planning module responsible of managing the flow of information within the agent: The core-agent intercept user instructions, leverage the API retriever to gather relevant APIs, relays the APIs to the LLM for response generation, executes the requested APIs, and finally returns the outcome back to the LLM for final user response formulation. Thus, the identification of core-agents in these systems, accompanied with the presence of LLMs elucidates that Toolformer and ToolLLM are indeed LLM-based agents.

Furthermore, **[59]** improves planning process by leveraging both LLMs and Planning Domain Definition Language (PDDL) based planners. In fact, the LLM generates a PDDL-based description of the problem, which is then evaluated by a PDDL-based planner to elaborate the optimal plan, and finally translated back into natural language. We observe that our framework aligns with the aforementioned architecture; The discussed technique describes indeed a core-agent containing a planning module powered with a PDDL interpreter and utilizing an LLM to translate from and to natural language.

The integration of the security module into the core-agent is substantiated by the presence of security concerns in LLMs and the imperative to address them comprehensively. While some studies focus solely on guardrail techniques or algorithms without tying them to an agent, the application of guardrails leveraging our framework occurs within the frame of a core-agent, particularly within its security module.

For example, LLMSafeGuard **[49]** introduces a lightweight framework to protect in real-time the LLM text generation. This study opted for the usage of the beam search algorithm to generate candidate responses. Leveraging an external validator, it rejects candidates that violate security constraints and proceeds with valid ones. Projecting this work on our framework, we concluded that the external validator is indeed a core-agent whose planning module assesses the alignment of a candidate with the agent's security constraints to determine whether to generate alternative candidates in case of rejection or proceed with sentence completion in case of acceptance. Evidently, this module communicates with the security module which implements a real-time validation approach to decide which path to take in the LLM's generation process.

Despite paper **[60]** emphasizes on the necessity of achieving reliability, confidentiality, and integrity in LLM-based agents by proposing methods to attain each attribute, it does not explicitly discuss the implementation of these mechanisms architecturally. Additionally, all discussed techniques serve the diverse security goals outlined in our framework, reinforcing the rationale for relocating guardrails to the security module within the core-agent.

4.2 Evaluation of active/passive core-agent concept

Our introduced framework, LLM-Agent-UMF, provides a valuable tool for comparing several state-of-the-art LLM-based agents as represented in Table 3. In this section, we evaluate our LLM-Agent-UMF by projecting it onto existing agent implementations and highlighting the presence or absence of essential modules such as planning, memory, profile, action, and security modules. This analysis allows us to classify the identified core-agents into active and passive. Besides, it provides insights into existing agents structures facilitating opportunities to merge functionalities from different agents within one system for enhanced capabilities.

Since some solutions such as ChatGPT are proprietary and closed-source, we cannot analyze its structure and capabilities assertively or project them transparently onto our own framework. Based on the observed behavior and public information about ChatGPT, we can make plausible hypotheses regarding its features and capabilities, but we cannot state them definitively.

To guide our hypothesis, we assessed ChatGPT 4o mini's response to illegal prompts by demanding the steps to hotwire a car. Our investigation aimed to verify the system's behavior when confronted with malicious queries. Matching our expectation, ChatGPT 4o mini declined to respond to direct illegal prompts, as illustrated in Appendix A, Figure 14. Consequently, we formulated two hypotheses on how the security measures are implemented:

- **Hypothesis 1**: The LLM itself is trained to monitor the input and prevent generation of undesirable output. This can be achieved by adhering to Adversarial Training (AT) techniques **[61]** and implementing input validation and sanitization to prevent malicious prompts from being used. Furthermore, if we consider ChatGPT 4o mini as an agent, there is no clear evidence of its interaction with external tools, which would necessitate functionalities of an action module in a core-agent. Based on this argument, we hypothesize that ChatGPT 4o mini is not an agent in the first place.

- **Hypothesis 2**: Security measures are implemented independently outside the scope of the main LLM. The input is handled before being forwarded to the LLM, and the output is monitored after its generation and before being presented to the user. According to this flow of events, we observe a minimum level of algorithmic planning thus the need for an active core-agent responsible for managing the guardrails of ChatGPT. However, such simple planning designed for a specific goal _ensuring safeguarding workflow_ does not require a memory module nor a profile module.

Similarly, we consider the capability of ChatGPT 4o on code execution and suppose the presence of an active core-agent. Indeed, we constructed the **third hypothesis**, around that process as follows: the core-agent checks if the prompt requires coding operations. If it is the case, it changes the profile of the LLM according to the task at hand, then leverages the LLM capability to generate code which is later executed in an isolated environment. Afterwards it reset the profile of the LLM to explain the results in a suitable textual representation. This hypothesis could be further enriched with assumptions from the ChatGPT 4o mini second hypothesis about the security measures and leads us to the conclusion that ChatGPT 4o is an LLM-based agent powered with a fully featured active core-agent rather than a standalone LLM.

In table 3, we categorize a group of LLM-based agents. Our selection includes diverse open-source and proprietary agents.

Table 3: Classification of state-of-the-art agents using LLM-Agent-UMF

		Core-Agent Modules					**Core-Agent Category**
		Planning	**Profile**	**Memory**	**Action**	**Security**	
ToolLLM [54]		X [*]	X	X	X	-	Active
Gorilla [62] [**]	Zero-shot	-	-	-	X	-	Passive
	With retriever	X [*]	X	X	X	-	Active
Toolformer [37]		-	-	-	X	-	Passive
Confucius [63]		-	-	-	X	-	Passive
ToolAlpaca[64]		-	-	-	X	-	Passive
GPT4Tools [65] [***]		X	X	X	X	-	Active
Chameleon [7]		X	X	X	X	-	Active
ChatDB [30]		X	X	X	X	-	Active
LLM+P [59]		X	X	X	X	-	Active
LLMSafeGuard [49]		X	-	X	X	X	Active
ChatGPT 4o mini	Hypothesis 1 [****]	-	-	-	-	-	N/A
	Hypothesis 2 [****]	X	-	-	X	X	Active
ChatGPT 4o	Hypothesis 3 [****]	X	X	X	X	X	Active

[*] Both ToolLLM and Gorilla (with retriever mode) use a retriever that recommends APIs and fetch their documentation.
[**] Gorilla was implemented in 2 versions: the first version is "zero-shot" without prompt tuning, the second is with a retriever which retrieves up-to-date API documentation to relay back to the LLM.
[***] GPT4Tools framework leverages multi-modal tool usage. In case of a multi-step subtask, any procedure resulting in an intermediary output is stored by the memory module and employed by the planning module to reach the final output.
[****] ChatGPT is a proprietary software owned by OpenAI, so we have formulated hypotheses regarding its structure, as detailed in the paragraph preceding Table 3.

By examining the core-agents within each agent independently, developers can deduce the compatibility of core-agents and identify challenges that may arise when integrating multiple agents into one system such as the necessity of synchronization, potential conflicts between core-agents or functional redundancy.

4.3 Evaluation of multi-core agent architectures

Thanks to the classification done in table 3, we can now effortlessly merge various aspects from existing agents into a single entity. To demonstrate this capability, we propose four representative scenarios that highlight the potential of LLM-Agent-UMF for designing multi core-agent systems by combining distinctive features from state-of-the-art agents. Each of these scenarios utilizes Llama 3.1 8B **[53]**, the newest state-of-the-art LLM from Meta AI team, for its remarkable performance and optimized memory footprint.

4.3.1 Toolformer and Confucius as a multi passive core-agent system

As both Toolformer **[37]** and Confucius **[63]** agents incorporate only passive core-agents, it becomes evident that integrating their capabilities within one agent is viable. As illustrated in Figure 9, the new agent, named LA1 (LLM-based Agent 1), encompasses two passive core-agents. On one hand, the Toolformer passive core-agent empowers the agent with the ability to utilize specialized tools such as a calculator, a calendar, a knowledge retrieval LM, a machine translation system and the Wikipedia search engine, ensuring that LA1 can effectively handle these tools in an accurate manner. On the other hand, the Confucius passive core-agent acts as a complementary second core-agent, enabling LA1 to manage unseen tools and work alongside Toolformer to tackle tools not previously evaluated or encountered during the testing phase. This versatile design makes LA1 capable of dealing with new challenges in real world scenarios while maximizing efficiency.

Nevertheless, it is crucial to ensure that the agent LLM, Llama 3.1 8B, is aligned with relevant regulation datasets. As elucidated by the LLM-Agent-UMF, techniques such as LoRA can be used to create pluggable modules to define the profile of the LLM. In fact, Toolformer's modified version of CCNet augmented with API calls should be used to teach the LLM how to communicate appropriately with the Toolformer passive core-agent. Similarly, Confucius, defining itself as a tool learning framework, should also be leveraged to train the LLM to master various external tools.

The integration of these two passive core-agents within LA1 showcases the effectiveness of LLM-Agent-UMF in designing multi passive core-agent systems and highlights its flexibility as well as potential for combining multiple existing agents' capabilities. Furthermore, it is important to acknowledge that during this process, LLM-Agent-UMF led us to identify weaknesses in the architectural design such as the absence of a privacy safeguarding mechanism to monitor data transfers between the Toolformer and Confucius core-agents and external service providers. This emphasizes the significance of ongoing research aimed at addressing these concerns.

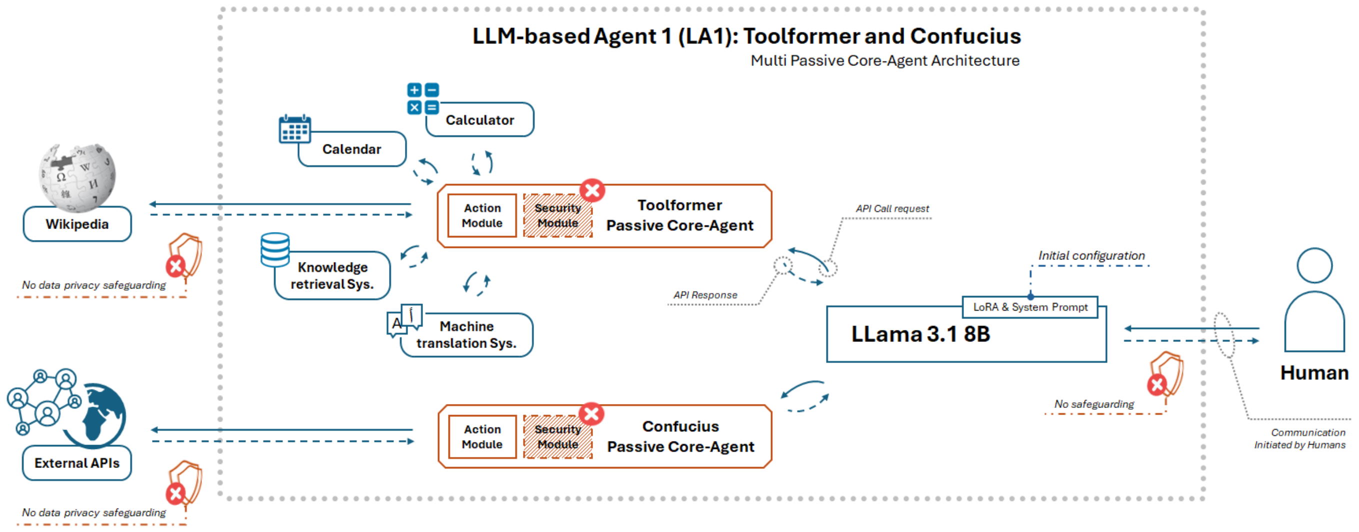

LLM-based Agent 1 (LA1): Toolformer and Confucius
Multi Passive Core-Agent Architecture
Calculator
Calendar
Wikipedia
Action Module
Security Module
Toolformer Passive Core-Agent
No data privacy safeguarding
Knowledge retrieval Sys.
Machine translation Sys.
API Call request
API Response
Initial configuration
LoRA & System Prompt
LLama 3.1 8B
Human
No safeguarding
Communication Initiated by Humans
External APIs
Confucius Passive Core-Agent
No data privacy safeguarding

Figure 9: LLM-based Agent 1 (LA1): Toolformer and Confucius – Multi Passive Core-Agent Architecture

4.3.2 ToolLLM and ChatDB as a multi active core-agent system

The second scenario explores a new agent design that integrates the ToolLLM **[54]** and ChatDB **[30]** capabilities. While both agents possess unique strengths, their combined functionality offers an advantageous synergy. Equipped with a neural API retriever, ToolLLM is capable of leveraging the appropriate external API to fulfill human instructions. On the other hand, ChatDB incorporates an SQL-based symbolic memory framework that enables LLMs to perform complex multi-hop reasoning.

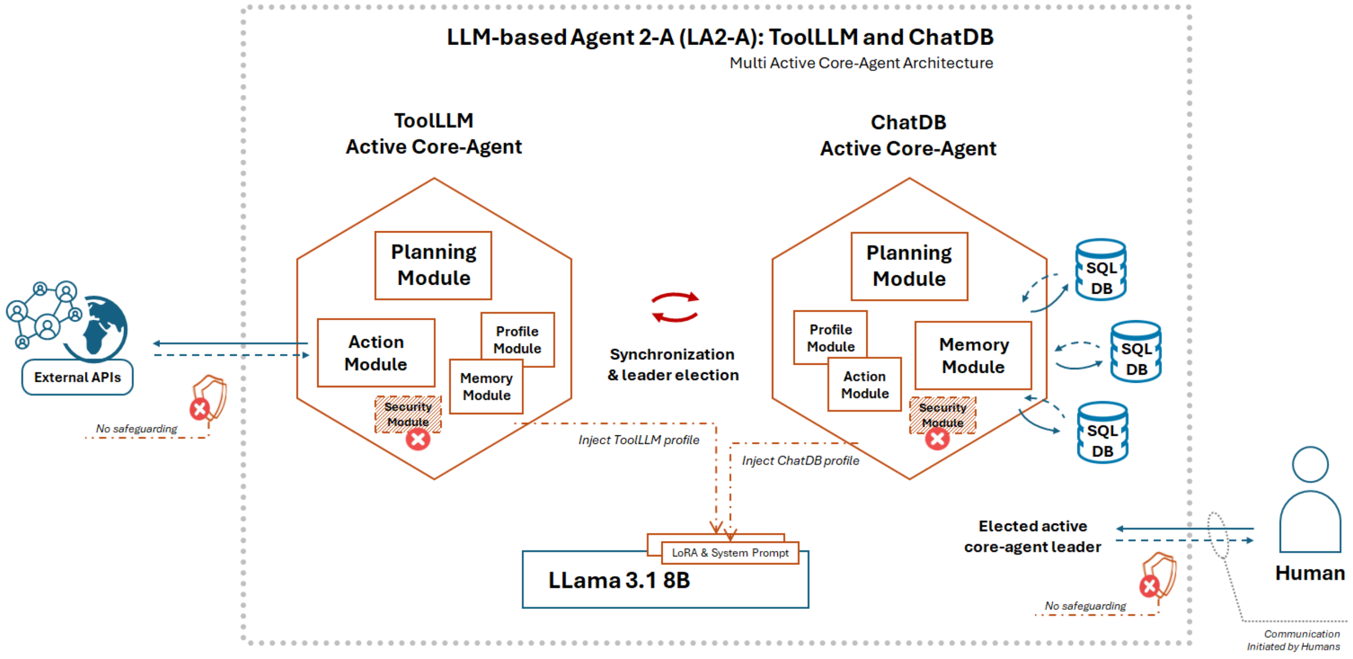


Figure 10: LLM-based Agent 2-A (LA2-A): ToolLLM and ChatDB – Multi Active Core-Agent Architecture

As mentioned in section 3.4.2, incorporating multiple active core-agents within one system could pose challenges. To evaluate this scenario, two architectural variants were explored: LA2-A and LA2-B. In LA2-A, Figure 10, both ToolLLM and ChatDB retained their individual functionalities as distinct active core-agents. This case requires synchronization between the two active core-agents and opting for a consensus algorithm like Raft **[56]** would be a knowledgeable choice. To further optimize the memory footprint of the system, there will be one unique instance of Llama 3.1 8B shared between the two active core-agents and each one of them must inject the adequate profile dynamically either as a pluggable trained module like LoRA or using a system prompt.

However, for LA2-B, illustrated in Figure 11, the unique capabilities of the two core-agents were merged into a single monolithic active core-agent. In this case, it is essential to identify specific modules where conflicts may arise. Being the key element in an active core-agent, the planning module must be thoroughly analyzed. Indeed, it should be designed to optimally handle external API calling through the integration of ToolLLM's API retriever while also seamlessly communicating with ChatDB's memory module specialized in SQL-based database handling. Furthermore, the profile module must be able to select the appropriate profile for the LLM depending on the task at hand. By addressing potential conflicts within these modules, LA2-B can effectively leverage both agents' strengths and achieve a synergistic advantage by leveraging only one monolithic active core-agent.

These two scenarios highlight the versatility of the LLM-Agent-UMF for designing novel LLM-based agents combining multiple complex state-of-the-art agents supplemented with active core-agents, while also identify and addresses the challenges associated with such integration efforts.

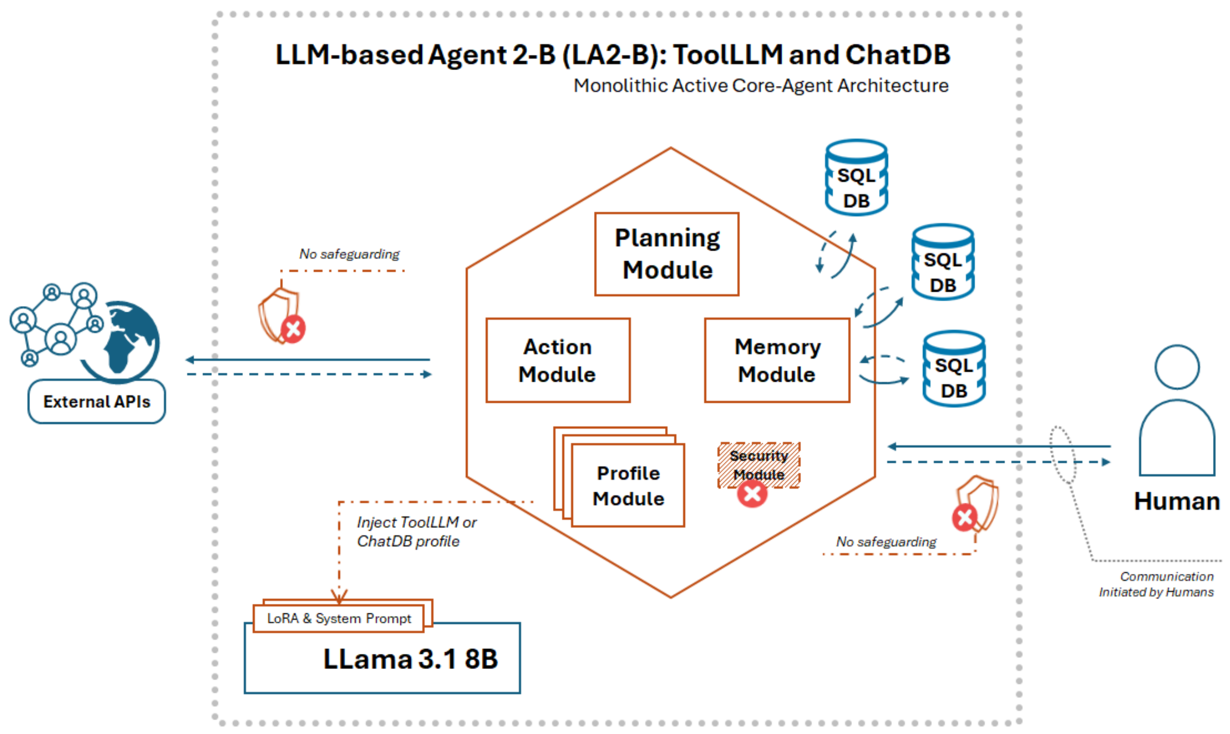


Figure 11: LLM-based Agent 2-A (LA2-A): ToolLLM and ChatDB – Multi Active Core-Agent Architecture

4.3.3 Implanting LLMSafeGuard security module in ToolLLM

A third observation is that any active core-agent can be taken as a base to be expanded with other modules as depicted in Figure 12. Taking the example of ToolLLM, we can implant the security module of another agent selected based on specific security objectives. Namely, if the goal is to augment ToolLLM's capabilities **[54]** with real-time safeguarding of the generated text, we can incorporate the security module of LLMSafeGuard **[49]** resulting in a newly designed agent, LA3. This example underscores the simplicity of such integration from a software architectural viewpoint.

Indeed, the LLM-Agent-UMF enables us to easily identify and incorporate missing modules without causing functional conflicts or challenges. Structurally, LA3 inherits the four primary modules of ToolLLM, along with the security module from LLMSafeGuard, seamlessly expanding its capabilities while maintaining compatibility and coherence between components.

4.3.4 Hybrid active/passive core-agent system

The last proposed agent design, LA4, is the most broad-based integration proposition representing the one-active-many-passive architecture outlined in section 3.4.2. As illustrated in Figure 13, LA4 architecture incorporates the active core-agent from LLM+P **[59]**, implants the security module from LLMSafeGuard **[49]**, and integrates the two passive core-agents from Toolformer and Confucius.

The selection of the LLM+P active core-agent as our central active entity was motivated by its cutting-edge planning module. As delineated in section 4.1, it seamlessly makes use of an LLM to generate a PDDL-based description from natural language inputs, which is subsequently evaluated by the integrated PDDL planner to establish an optimal plan. Unfortunately, as identified in table 3, LLM+P does not possess a security module making the overall agent vulnerable to security threats such as adversarial attacks that easily circumvent basic protection mechanisms such as implemented using adversarial training. This led us to leverage the same solution proposed in section 4.3.3. and implant the LLMSafeGuard security module in the LLM+P active core-agent. As a result, this procedure yields an optimized active core-agent that incorporates all five necessary modules for efficient functioning while ensuring robustness against potential security threats.

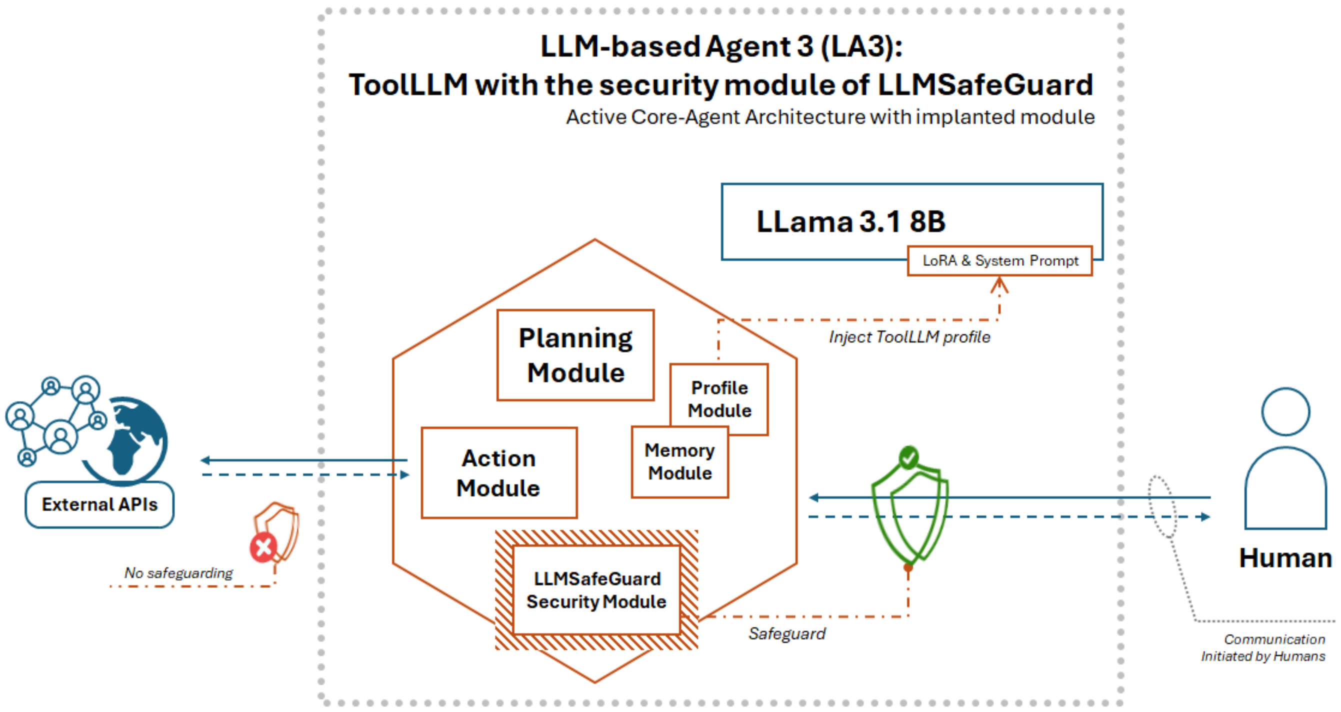


Figure 12: LLM-based Agent 3 (LA3): ToolLLM with the security module of LLMSafeGuard

To further enhance LA4's capabilities and performance, it would be advantageous to empower the agent with the skill of effectively utilizing available tools and APIs. In section 4.3.1, Toolformer and Confucius were proposed as suitable candidates due to their complementary features. Indeed, their passive core-agents nature makes the integration straightforward and does not necessitate advanced synchronization mechanisms since both will be controlled by the LLM which itself is managed by the active core-agent. The primary consideration is the incorporation of both Toolformer and Confucius profiles into the profile module of LA4's active core-agent, allowing it to adapt the LLM behavior dynamically for its specific needs. By combining these various technologies within LA4, it would become a comprehensive agent capable of elaborating an optimal planning strategy leveraging tools and external APIs.

Nonetheless, it is crucial to address the security vulnerability uncovered through the application of LLM-Agent-UMF within LA4's design. Indeed, the external API calling mechanisms utilized by Toolformer and Confucius passive core-agents are not monitored or safeguarded against potential information leakage. Following the guidance provided by LLM-Agent-UMF, this issue can be resolved either by implementing a dedicated security module on each of these passive core-agents or as supplementary measures within the active core-agent's security module, thus centralizing the management of information safeguarding processes. The optimal selection of technologies and implementation details will be explored in future work.

The four previous scenarios exemplify how researchers and developers can make informed decisions about the design of their LLM-based agents prior to the development process, grounded in clear architectural reasoning. This systematic approach will enhance the robustness and functionality of LLM-based agents. In the next section, we will discuss the limitations and the future work to enhance our framework.

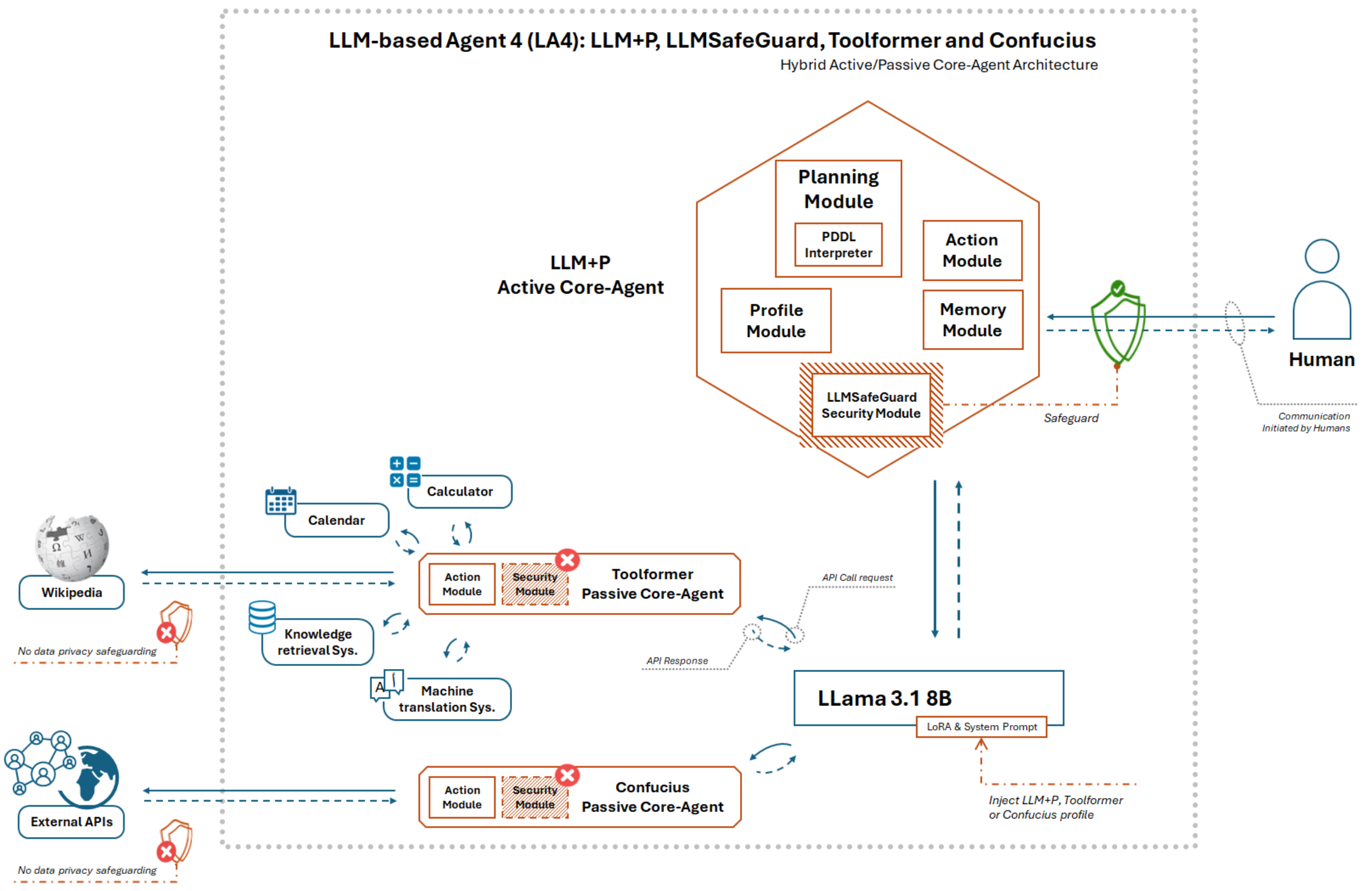

LLM-based Agent 4 (LA4): LLM+P, LLMSafeGuard, Toolformer and Confucius
Hybrid Active/Passive Core-Agent Architecture
LLM+P
Active Core-Agent
Planning Module
PDDL Interpreter
Action Module
Profile Module
Memory Module
LLMSafeGuard Security Module
Safeguard
Human
Communication Initiated by Humans
Calculator
Calendar
Wikipedia
No data privacy safeguarding
Action Module
Security Module
Toolformer Passive Core-Agent
Knowledge retrieval Sys.
Machine translation Sys.
API Call request
API Response
LLama 3.1 8B
LoRA & System Prompt
Inject LLM+P, Toolformer or Confucius profile
External APIs
No data privacy safeguarding
Action Module
Security Module
Confucius Passive Core-Agent

Figure 13: LLM-based Agent 4 (LA4): LLM+P, LLMSafeGuard, Toolformer and Confucius – Hybrid Active/Passive Core-Agent

5 Conclusion and future work

In this paper, we introduce a structural component within LLM-based agents named the core-agent. This component is engineered to address the architectural ambiguities that software developers encounter and enhance modularity. We propose LLM-Agent-UMF, a comprehensive framework for modeling the structure of the agent, explicating each of core-agent's five modules: planning, memory, profile, action, and security. Subsequently, we classified core-agents into passive and active and highlighted their structural and functional differences. Based on this classification, we designed uniform and hybrid multi-core agent architectures. Most prominently, the one-active-many-passive architecture exploits the full potential of both active and passive core-agents, striking a balance between easiness of development and the power of hybrid architectures. By applying our framework to state-of-the-art agents, we identified within their structures core-agents and their constituting internal modules which assisted us in the classification process. This allowed us to recognize the individual characteristics of each agent and discover potential prospects of merging different functionalities into a single multi-core agent.

Our work prepares the foundation for the development of LLM-based agents with a clear delineated structure that leverages the power of core-agents. The progressive adoption of LLM-Agent-UMF will further attest to its efficiency. An interesting future direction to improve it involves finding solutions to simplify the implementation of multi-active core-agent architectures. In fact, they suffer from challenges related to synchronization that necessitate further investigation. In this context, we see two promising avenues: Integrating a consensus algorithms like Raft to elect a leader responsible for the coordination; Otherwise, integrating a central gateway that is solely responsible of selecting the most suitable active core-agent to handle the user request based on factors like load, availability, and domain. Each active core-agent shall register with the gateway, providing information about their capabilities and status. The selected core-agent processes the task and sends the response back to the user through the gateway.

In conclusion, the ultimate purpose of this framework, which is predicated on the core-agent, is to reformulate the conception of LLM-based agents. By basing their development on this unit rather than addressing it in a monolithic manner, researchers and practitioners can use a unified terminology to refer to different modules within a common architecture. This shared foundation facilitates uniformity in research and enhancements. Additionally, developers can commence implementing consistent solutions that are easily maintainable and highly adaptable for future improvements.

References

[1] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3), mar 2024. ISSN 2157-6904. doi:10.1145/3641289. URL `https://doi.org/10.1145/3641289`.

[2] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings, editors, *Intelligent Agents III Agent Theories, Architectures, and Languages*, pages 21–35, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-68057-4. doi:10.1007/BFb0013570.

[3] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023.

[4] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024. doi:10.1007/s11704-024-40231-1. URL `https://doi.org/10.1007/s11704-024-40231-1`.

[5] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models, 2022. URL `https://arxiv.org/abs/2205.12255`.

[6] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He. Exploring large language model based intelligent agents: Definitions, methods, and prospects, 2024. URL `https://arxiv.org/abs/2401.03428`.

[7] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 43447–43478. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/871ed095b734818cfba48db6aeb25a62-Paper-Conference.pdf`.

[8] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024. URL `https://arxiv.org/abs/2402.01030`.

[9] Sihao Hu, Tiansheng Huang, Fatih Ilhan, Selim Tekin, Gaowen Liu, Ramana Kompella, and Ling Liu. A survey on large language model-based game agents, 2024. URL `https://arxiv.org/abs/2404.02039`.

[10] Zhixuan Chu, Yan Wang, Feng Zhu, Lu Yu, Longfei Li, and Jinjie Gu. Professional agents – evolving large language models into autonomous experts with human-level competencies, 2024. URL `https://arxiv.org/abs/2402.03628`.

[11] M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, May 2024. doi:10.1038/s42256-024-00832-8. URL `https://doi.org/10.1038/s42256-024-00832-8`.

[12] Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie Ruan, and Xiaowei Huang. Building guardrails for large language models, 2024. URL `https://arxiv.org/abs/2402.01822`.

[13] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 80079–80110. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/fd6613131889a4b656206c50a8bd7790-Paper-Conference.pdf`.

[14] Haonan Duan, Adam Dziedzic, Nicolas Papernot, and Franziska Boenisch. Flocks of stochastic parrots: Differentially private prompt learning for large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko,

M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 76852–76871. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f26119b4ffe38c24d97e4c49d334b99e-Paper-Conference.pdf.

[15] Unggi Lee, Sanghyeok Lee, Junbo Koh, Yeil Jeong, Haewon Jung, Gyuri Byun, Yunseo Lee, Jewoong Moon, Jieun Lim, and Hyeoncheol Kim. Generative agent for teacher training: Designing educational problem-solving simulations with large language model-based agents for pre-service teachers, 2023.

[16] Apostolos Ampatzoglou, Angeliki-Agathi Tsintzira, Elvira-Maria Arvanitou, Alexander Chatzigeorgiou, Ioannis Stamelos, Alexandru Moga, Robert Heb, Oliviu Matei, Nikolaos Tsiridis, and Dionisis Kehagias. Applying the single responsibility principle in industry: Modularity benefits and trade-offs. In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*, EASE '19, page 347–352, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450371452. doi:10.1145/3319008.3320125. URL https://doi.org/10.1145/3319008.3320125.

[17] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He. Exploring large language model based intelligent agents: Definitions, methods, and prospects, 2024. URL https://arxiv.org/abs/2401.03428.

[18] O. Turan and Ö. Ö. Tanrıöver. An experimental evaluation of the effect of solid principles to microsoft vs code metrics. *AJIT-E: Academic Journal of Information Technology*, 9(34):7–24, 2018. doi:10.5824/1309-1581.2018.4.001.x.

[19] Miguel A. Laguna, José M. Marqués, and Yania Crespo. On the semantics of the extend relationship in use case models: Open-closed principle or clairvoyance? In Barbara Pernici, editor, *Advanced Information Systems Engineering*, pages 409–423, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13094-6.

[20] Malik Ghallab. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[21] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024. URL https://arxiv.org/abs/2402.02716.

[22] Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. Dara: Decomposition-alignment-reasoning autonomous language agent for question answering over knowledge graphs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 3406–3432, Bangkok, Thailand and virtual meeting, aug 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.203.

[23] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1173–1203, Bangkok, Thailand, aug 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.65.

[24] Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwaśniewski, Jürgen Müller, Lukas Gianinazzi, Ales Kubicek, Hubert Niewiadomski, Aidan O'Mahony, Onur Mutlu, and Torsten Hoefler. Demystifying chains, trees, and graphs of thoughts, 2024. URL https://arxiv.org/abs/2401.14295.

[25] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. Pddl - the planning domain definition language, 08 1998.

[26] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

[27] R.C. Atkinson and R.M. Shiffrin. Human memory: A proposed system and its control processes. In Kenneth W. Spence and Janet Taylor Spence, editors, *Psychology of Learning and Motivation*, volume 2, pages 89–195. Academic Press, 1968. doi:10.1016/S0079-7421(08)60422-3. URL https://www.sciencedirect.com/science/article/pii/S0079742108604223.

[28] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17): 19724–19731, Mar. 2024. doi:10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

[29] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=ehfRiF0R3a`.

[30] Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory, 2023. URL `https://arxiv.org/abs/2306.03901`.

[31] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023. URL `https://arxiv.org/abs/2305.17144`.

[32] Aniket Kumar Singh, Bishal Lamichhane, Suman Devkota, Uttam Dhakal, and Chandra Dhakal. Do large language models show human-like biases? exploring confidence—competence gap in ai. *Information*, 15(2), 2024. ISSN 2078-2489. doi:10.3390/info15020092. URL `https://www.mdpi.com/2078-2489/15/2/92`.

[33] Aniket Kumar Singh, Bishal Lamichhane, Suman Devkota, Uttam Dhakal, and Chandra Dhakal. Do large language models show human-like biases? exploring confidence - competence gap in ai. *Inf.*, 15:92, 2024. URL `https://api.semanticscholar.org/CorpusID:267539494`.

[34] Lisa P. Argyle, Ethan C. Busby, Nancy Fulda, Joshua R. Gubler, Christopher Rytting, and David Wingate. Out of one, many: Using language models to simulate human samples. *Political Analysis*, 31(3):337–351, 2023. doi:10.1017/pan.2023.2.

[35] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023. URL `https://arxiv.org/abs/2312.12148`.

[36] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2024. URL `https://arxiv.org/abs/2302.01560`.

[37] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf`.

[38] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 38154–38180. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/77c33e6a367922d003ff102ffb92b658-Paper-Conference.pdf`.

[39] Jerry Liu. LlamaIndex, 11 2022. URL `https://github.com/jerryjliu/llama_index`.

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

[41] Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents, 2024. URL `https://arxiv.org/abs/2403.12014`.

[42] MD Minhaz Chowdhury, Nafiz Rifat, Mostofa Ahsan, Shadman Latif, Rahul Gomes, and Md Saifur Rahman. Chatgpt: A threat against the cia triad of cyber security. In *2023 IEEE International Conference on Electro Information Technology (eIT)*, pages 1–6, 2023. doi:10.1109/eIT57321.2023.10187355.

[43] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online, nov 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.301. URL `https://aclanthology.org/2020.findings-emnlp.301`.

[44] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. URL `https://arxiv.org/abs/1412.6572`.

[45] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. Prompt injection attack against llm-integrated applications, 2024. URL `https://arxiv.org/abs/2306.05499`.

[46] Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In Yansong Feng and Els Lefever, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 431–445, Singapore, dec 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-demo.40. URL `https://aclanthology.org/2023.emnlp-demo.40`.

[47] Yi Dong, Ronghui Mu, Yanghao Zhang, Siqi Sun, Tianle Zhang, Changshun Wu, Gaojie Jin, Yi Qi, Jinwei Hu, Jie Meng, Saddek Bensalem, and Xiaowei Huang. Safeguarding large language models: A survey, 2024. URL `https://arxiv.org/abs/2406.02622`.

[48] Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. Autodan: Interpretable gradient-based adversarial attacks on large language models, 2023. URL `https://arxiv.org/abs/2310.15140`.

[49] Ximing Dong, Dayi Lin, Shaowei Wang, and Ahmed E. Hassan. A framework for real-time safeguarding the text generation of large language model, 2024. URL `https://arxiv.org/abs/2404.19048`.

[50] Haoran Li, Yulin Chen, Jinglong Luo, Yan Kang, Xiaojin Zhang, Qi Hu, Chunkit Chan, and Yangqiu Song. Privacy in large language models: Attacks, defenses and future directions, 2023. URL `https://arxiv.org/abs/2310.10383`.

[51] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models, 2024. URL `https://arxiv.org/abs/2301.10226`.

[52] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, and Ben Edwards. Adversarial robustness toolbox v1.0.0, 2019. URL `https://arxiv.org/abs/1807.01069`.

[53] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and Aiesha Letman et al. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

[54] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL `https://arxiv.org/abs/2307.16789`.

[55] Thorsten Händler. A taxonomy for autonomous llm-powered multi-agent architectures. In *KMIS*, pages 85–98, 2023.

[56] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(1):172–181, 2020. doi:10.1109/TSMC.2019.2895471.

[57] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, jun 2014. USENIX Association. ISBN 978-1-931971-10-2. URL `https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro`.

[58] Josip Slisko and Dewey I Dykstra Jr. The role of scientific terminology in research and teaching: is something important missing? *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, 34(6):655–660, 1997.

[59] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023. URL `https://arxiv.org/abs/2304.11477`.

[60] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal llm agents: Insights and survey about the capability, efficiency and security, 2024. URL `https://arxiv.org/abs/2401.05459`.

[61] Aleksander Mądry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *stat*, 1050(9), 2017.

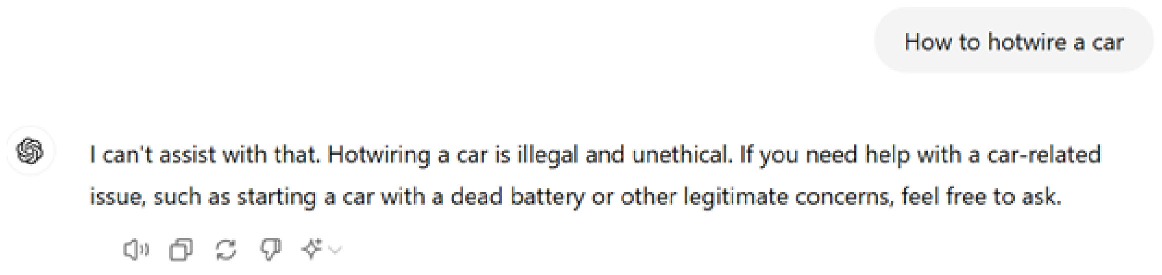
[62] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL `https://arxiv.org/abs/2305.15334`.

[63] Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18030–18038, Mar. 2024. doi:10.1609/aaai.v38i16.29759. URL `https://ojs.aaai.org/index.php/AAAI/article/view/29759`.

[64] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases, 2023. URL `https://arxiv.org/abs/2306.05301`.

[65] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 71995–72007. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/e393677793767624f2821cec8bdd02f1-Paper-Conference.pdf`.

A Screenshot of ChatGPT 4o mini refusing to explain how to hot-wire a car



How to hotwire a car

I can't assist with that. Hotwiring a car is illegal and unethical. If you need help with a car-related issue, such as starting a car with a dead battery or other legitimate concerns, feel free to ask.

Figure 14: ChatGPT 4o mini refusing to explain how to hot-wire a car