MOSAICO

D3.1: BDI Framework for Agents in AI Communities

November, 2025

Funded by the European Union

Funded by the European Union under the Grant Agreement No 101189664. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

D3.1 BDI Framework for Agents in AI Communities

Work package	WP3 Coordination and collaboration of communities of AIs
Task	T3.1 BDI framework for agents in AI communities
Due date	30/09/2025
Submission date	04/11/2025
Type of deliverable	R - Document, report
Dissemination Level	Public
Deliverable lead	IMT
Version	1.0.0
Authors	IMT, UDA
Reviewers	UY, LIST, QODO
Keywords	LLM-based agent, multi-agent system, belief-desire-intention

Document Revision History

Version	Date	Description of change	List of contributor(s)
0.0.1	02/09/2025	Initial population of template	IMT
0.1.0	20/09/2025	Interim version before internal review	IMT, UDA
0.2.0	06/10/2025	First version for internal review	IMT, UDA
0.3.0	23/10/2025	Revision of Section 2	UDA
0.4.0	30/10/2025	Revision of Section 3	IMT
1.0.0	31/10/2025	Final version	IMT

Partners

Funding

Funded by
the European Union

Call	Digital and emerging technologies for competitiveness and fit for the Green Deal (HORIZON-CL4-2024-DIGITAL-EMERGING-01)
Topic	HORIZON-CL4-2024-DIGITAL-EMERGING-01-22: Fundamentals of Software Engineering
Type of action	Research and Innovation Action (RIA)

Disclaimer

Copyright Notice

Executive Summary

In this deliverable, we investigate the integration of the Belief-Desire-Intention (BDI) framework from the classical literature of multi-agent systems into the new paradigm of LLM-based agents. The MOSAICO framework will need to reason on beliefs, desires, and intentions of its agents. This will be required, e.g., for selecting the agent whose objectives (desires) best match the global task, warning the user of the intention to perform a modification, evaluating that the beliefs of the agent are not faulty. Here we study how users may program the agent's BDI cycle in LLM-based agentic frameworks, and provide concrete code to implement BDI in an existing LLM-based agentic framework.

We start by presenting a multivocal literature review that provides a comprehensive state-of-the-art on LLM-based multi-agent systems (LLM-MAS), considering both academic and most recent open-source frameworks. The survey covers especially the related work in LLM-MAS from the submission of the MOSAICO proposal up to now. It allows us to introduce pivotal concepts that are needed to design effective agents, and to identify existing frameworks for experimenting in MOSAICO before the first version of the MOSAICO orchestrator (M15).

In the second part, we provide two alternative integration mechanisms for BDI in MOSAICO: 1) we extend LLM-based agents with semantic actions representing the evaluation steps of the BDI reasoning cycle, 2) we integrate an existing domain-specific language (DSL) for describing BDI behavior into LLM-based agents. Such mechanisms are accompanied by executable code that extends an pre-existing open-source agentic framework, Microsoft AutoGen. The integration patterns can be replicated for other LLM-based agentic frameworks and applied in diverse use cases. We provide examples of such use cases in the final section.

Table of Contents

1 Introduction **8**

2 State of the art on LLM-based Multi-Agent Systems **9**
2.1 Autonomous Agents 9
2.2 LLM-based agents 10
2.2.1 Tool-use 12
2.2.2 Memory 13
2.2.3 Decision-making Thought 14
2.3 Multi-vocal Literature review on MAS 15
2.3.1 Planning 17
2.3.2 Conducting 20
2.4 Conclusion 29

3 BDI for AI Agents **29**
3.1 Belief-Desire-Intention 30
3.1.1 AgentSpeak 31
3.1.2 BDI and LLMs 32
3.2 Ad-hoc BDI on AutoGen 32
3.3 AgentSpeak on AutoGen 37
3.4 Example scenarios 40
3.4.1 Prompting an LLM for missing beliefs 41
3.4.2 Requirement Manager (cooperation between AgentSpeak and AutoGen agents) . . 43
3.4.3 Translation from Natural Language to AgentSpeak by an LLM 44

4 Conclusion **47**

List of Figures

1 Overview of the MOSAICO framework 9
2 Self-attention in Transformer neural network architectures 11
3 Overview of an LLM-powered agent (Weng, 2023) 12
4 The RAG workflow 13
5 Overview of LLM-MAS 16
6 Overview of the conducted process. 17
7 Roles of LLM inference in BDI agents 33
8 Class diagram of our ad-hoc extension of AutoGen RoutedAgent. Map<String,String> 34
9 Communication diagram for the requirement manager example 35
10 Class diagram of AgentSpeak on AutoGen 38
11 Additional classes 39
12 Communication diagram for the prompt example 41
13 Class diagram for the prompt example 42
14 Communication diagram for the manager example of one iteration of the evaluation and generation cycle 45
15 Communication diagram for the translation example 45

List of Listings

1 Typical use of the RoutedAgent class from AutoGen 33
2 Implementation of `bdi_select_intention` in `RequirementManagerAgent` 36
3 Constructor of `RequirementManagerAgent` 36
4 Implementation of `bdi_act` in `RequirementManagerAgent` 36
5 Implementation of `bdi_select_intention` in `ConsensusAgent` 37
6 `Creation of a class deriving from BDIAgent` 40
7 Defining the `.prompt` action 41
8 AgentSpeak code of Requirement Manager 43

List of Tables

1 Keywords groups and final boolean search query 18
2 Inclusion and exclusion criteria for the literature analysis. 19
3 Inclusion and exclusion criteria for MAS frameworks. 19
4 Elicited papers from the literature analysis. 21
5 Contributions of the elicited papers 22
6 Overview of MAS frameworks. 25
7 Comparison of MAS frameworks in terms of covered features. 28

Abbreviations

BDI	Belief-Desire-Intention
CoT	Chain of Thought
DSL	Domain-Specific Language
LLM	Large-Language Model
LLM-MAS	LLM-Based Multi-Agent System
MAS	Multi-agent system
MLR	Multi-vocal literature review
SLR	Systematic literature review

1 Introduction

The recent emergence of agentic AI has opened new research directions in the field of multi-agent systems (MAS). These agents leverage the reasoning and communication capabilities of large-language models (LLMs) to perform complex collaborative and autonomous tasks. Within this evolving landscape, the MOSAICO framework aims to orchestrate and coordinate LLM-based agents for problem-solving in software engineering and modeling. To achieve this, MOSAICO requires agents capable of reasoning about their own beliefs, desires, and intentions—the foundational constructs of the classical Belief-Desire-Intention (BDI) model in multi-agent systems.

Figure 1 shows an overview of the integrated solution that is to be provided by MOSAICO. The user communicates their task to a reference agent (in red) through their client (in this case, a Microsoft Visual Studio Code extension). The reference agent then discovers the most appropriate task by querying the MOSAICO repository: for complex tasks, this may be a collaboration agent (in green) which will orchestrate other MOSAICO agents to solve the task in a reliable way. These may be solution agents which propose solutions to the given task (in blue), supervision agents which evaluate the proposed solutions (in yellow), consensus agents (which decide the final solution among several), or even other collaboration agents. In parallel, the right-hand section (Background Processes) represents the continuous maintenance of the MOSAICO Repository, where agents are benchmarked, classified, and managed to ensure optimal performance. This repository supports dynamic agent discovery, selection, and deployment during runtime.

Integrating BDI reasoning into such MOSAICO agents is crucial for enabling high-level cognitive behaviors such as: selecting the agent whose objectives best align with a global task, notifying users about upcoming actions or intentions, and verifying the consistency of an agent's beliefs. This deliverable investigates how the BDI framework can be effectively incorporated into modern LLM-based agentic architectures. It also explores how users can program and control the agent's BDI reasoning cycle in such frameworks, providing concrete implementation examples.

The deliverable is structured in two main sections. Section 2 presents a multivocal literature review offering a comprehensive state-of-the-art on LLM-based multi-agent systems (LLM-MAS), encompassing both academic research and the most recent open-source frameworks. This review establishes the conceptual background necessary to design effective agents within MOSAICO and identifies candidate frameworks suitable for early experimentation prior to the release of the first version of the MOSAICO orchestrator (M15).

Section 3 introduces two complementary integration mechanisms for embedding BDI reasoning into MOSAICO. The first extends LLM-based agents with semantic actions that correspond to the evaluation steps of the BDI reasoning cycle. The second incorporates an existing domain-specific language (DSL) for describing BDI behavior into LLM-based agents. Both approaches are demonstrated through executable implementations built upon the open-source framework Microsoft AutoGen. These integration patterns are designed to be reusable across other LLM-based agentic frameworks and adaptable to a variety of use cases, some of which are illustrated in the final section of the deliverable.

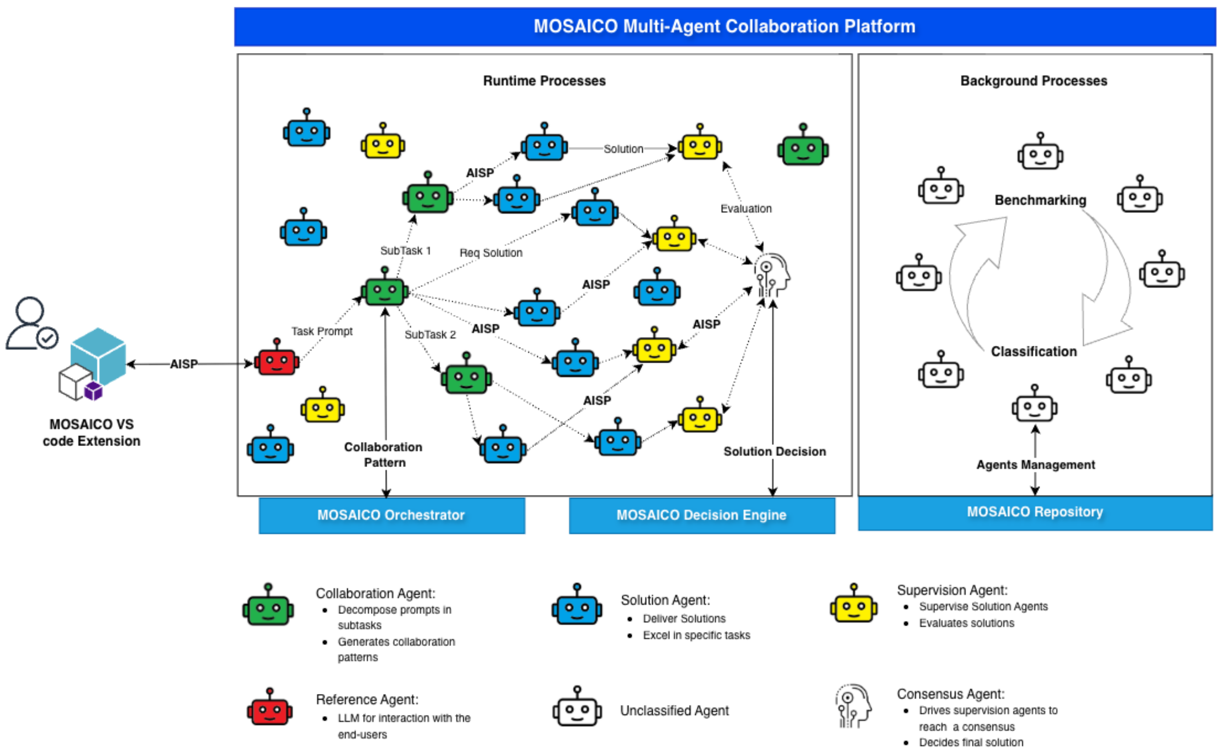


Figure 1: Overview of the MOSAICO framework

2 State of the art on LLM-based Multi-Agent Systems

The objective of this section is twofold. First, we provide an overview of autonomous agents and multi-agent systems (MAS) by analyzing the evolution from traditional agents to LLM-based multi-agent systems (LLM-MAS). In particular, we introduce pivotal concepts that are needed to design effective agents in Section 2.1 while Section 2.2 describes the peculiar features of LLM-based agents. Second, in Section 2.3 we present a multivocal literature review (MLR) that provides a comprehensive state-of-the-art on LLM-based multi-agent systems, considering both academic and most recent open-source frameworks. We finally summarize the main takeaways of our survey in Section 2.4.

2.1 Autonomous Agents

According to (Wooldridge et al., 1995), autonomous agents are pieces of software or hardware capable of acting independently, with or without human intervention. These agents can interact socially, perceive their environment, and act toward specific goals. Extending this definition, (Franklin et al., 1996) adds that the actions of an autonomous agent can affect its future perceptions. In particular, a software system that scans the environment through inputs and outputs cannot be considered an agent since its actions do not influence its future perceptions. In contrast, a thermostat can be considered an autonomous agent because adjusting the temperature is an action that will impact the environment and the agent's actions (Franklin et al., 1996) subsequently. Based on these core definitions, He et al. (He et al., 2025) formulate the following fundamental attributes:

1. **Autonomy**: It can supervise its actions and internal states without external intervention.
2. **Perception**: It has the ability to sense the environment through its sensors.
3. **Intelligence and Goal-Driven Behavior**: The agent has specific goals to achieve using domain knowledge and problem-solving abilities.
4. **Social Ability**: It can interact with other agents and humans to achieve these goals.
5. **Levels of Learning Capabilities**: According to the underpinning capabilities, an agent can continuously evolve and learn through its interactions with the environment.

Recently, different agents have been developed according to their capabilities, underlying architecture, design principles, and operating modes (Krishnan, 2025). Therefore, they can be classified into different categories as follows:

➢*Reactive agents.* These agents rely on a predefined set of condition–action rules, responding immediately to perceived stimuli. However, they lack the ability to react to situations beyond the predetermined rules, making them suitable only for stable and fully observable contexts.

➢*Model-based reflex agents.* Unlike reactive agents, model-based agents integrate an internal representation of the environment which allows to keep track of the current state, even when some information is not immediately perceptible. Thus, the decision-making ability is improved compared to reactive agents in partially observable environments, since the agent can evaluate the potential outcomes of a given action.

➢*Goal-based agents.* Goal-based agents further enhance environment perception by explicitly representing goal states. Such agents use reasoning to compare different strategies and select the most effective one to achieve the intended goal.

➢*Utility-based agents.* These agents extend the goal-based approach by introducing a utility function that assigns different values to possible outcomes, ensuring more consistent decisions when conflicts between goals arise. By comparing scenarios and values associated with potential benefits, they choose the actions that maximize the overall reward.

➢*Learning agents.* Compared to the others, learning agents have the ability to continuously improve performance through the processing of past experiences. Using the feedback reward mechanism, they can dynamically adapt the feedback received, resulting in optimized internal models, decision rules, and utility functions.

In addition to the aforementioned categories, agents can be designed to handle tasks within a specific application domain, e.g., healthcare, cyber-physical systems, or software development. These agents integrate knowledge, strategies, and operational capabilities closely tied to the context in which they operate. They are modeled to maximize efficiency and reliability in carrying out particular activities.

We acknowledge that the classification provided for intelligent agents is not exhaustive due to the fast-evolving nature of the field. However, this overview provides the concepts needed to understand LLM-based agents and how they can be combined to develop MAS.

2.2 LLM-based agents

Large Language Models (LLMs) are a specific instance of foundation models characterized by more than hundreds of billions parameters (W. X. Zhao et al., 2023) and trained on extensive datasets (Naveed et al., 2023). In particular, they leverage the *transformer* neural network architectures (Vaswani et al., 2017) to embed long-range dependencies within input sequences and their multilayer processing structure, thus allowing parallel training of complex models and enhancing language processing. In particular, the key mechanism is the *self-attention*, which allows each token in the sequence to attend to all others, weighting their relevance for meaning as shown in Figure 2.

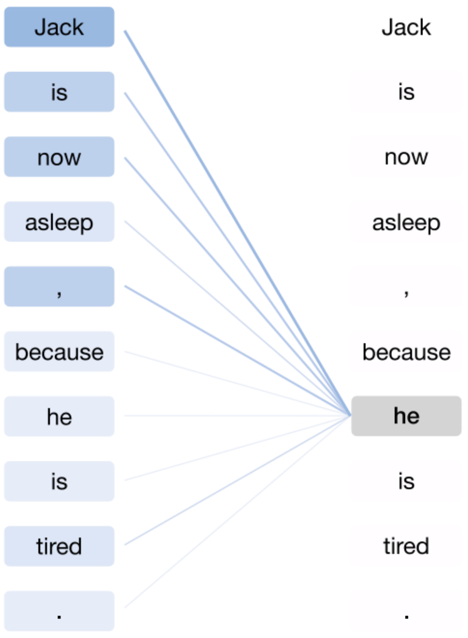


Figure 2: Self-attention in Transformer neural network architectures

The Transformer quickly became the standard architecture for natural language understanding and generation tasks.

To guide the inference phase, users can define a *prompt*, a structured textual input consisting of instructions to obtain a desired and coherent answer. Since the launch of GPT-3.5 in November 2022, several prompt engineering strategies have been proposed to optimize prompts (Amatriain, 2024).

Zero-shot prompt is the basic technique in which the model is provided with a set of instructions without explicit training on a task. In constrast, the few-shot technique (Brown et al., 2020) involves a limited number of examples or instructions in a given context to guide the model in generating relevant output. Obviously, the quality of the examples must be such that the model can produce consistent results (Xiaojun Chen et al., 2024).

Recent examples of LLMs include GPT-4 (Achiam et al., 2023), LLaMa (Touvron et al., 2023), Gemini (Gemini Team et al., 2023), and DeepSeek (A. Liu et al., 2024). All of these models share the capability to solve complex tasks with performance that resembles human-level intelligence. Given their advanced capabilities, LLMs have found applications in various fields, ranging from text-generation to multimodal processing.

In the context of autonomous agent development, LLMs can provide extensive knowledge and the capability to be fine-tuned for specific domains (Naveed et al., 2023). For example, the models in the GPT-family (Radford et al., 2018) have been trained in two stages, i.e., unsupervised pretraining and supervised fine-tuning, yielding specialization in natural language generation and strong performance in specific applications, e.g., downstream coding tasks (Mastropaolo et al., 2021), summarization (Doan et al., 2023), or multimodal content generation. Although the process can be performed once, it requires huge computational resources to obtain significant accuracy improvements. Among the most notable fine-tuning techniques, the LoRa approach (Hu et al., 2022) is a parameter-efficient fine-tuning technique that adjusts specific layers while freezing most of the original model parameters. In such a way, it reduces the hardware barrier to entry by up to 3 times compared to the most common techniques.

Figure 3 depicts the architecture of LLM-powered agents and the corresponding components as proposed by (Weng, 2023). Similarly to traditional autonomous agents presented in Section 2.1, the agent can make a set of *actions*, execute them, and trigger one or more reasoning steps. Unlike traditional agents, the LLM-based ones can leverage the tool-use capabilities to exploit a plethora of tools, including custom functions or other AI models. Afterward, the results obtained can be stored in the *Memory* component that can be used to trigger the *Decision-making Thought*, thus influencing the future behavior of the agent.

We provide a detailed description of each component in the next paragraphs.

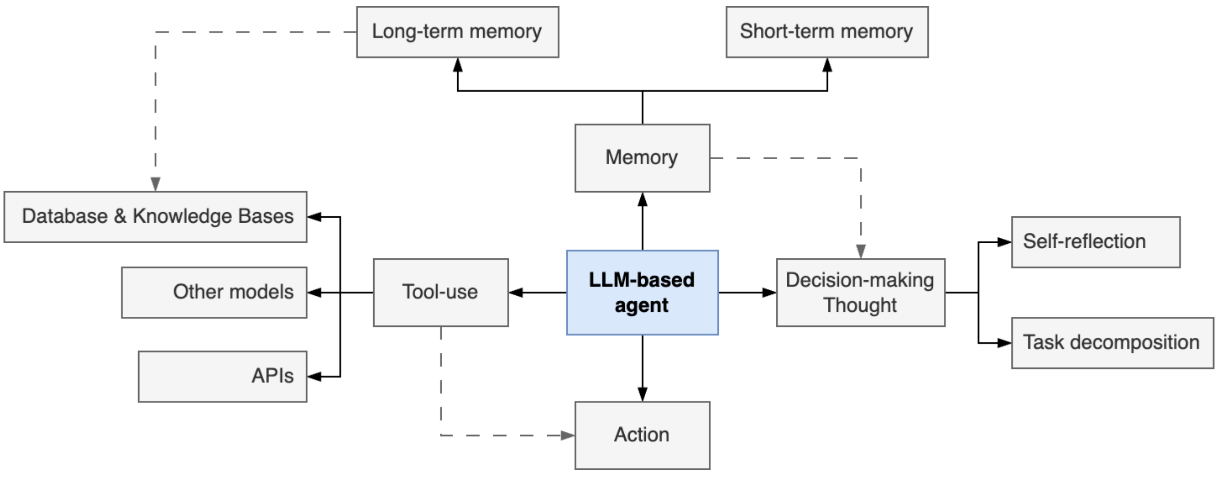


Figure 3: Overview of an LLM-powered agent (Weng, 2023)

2.2.1 Tool-use

To perform their tasks effectively, LLM-based agents can make use of external tools and resources (Guo et al., 2024), a concept also known as function calling (Huyen, 2025). This functionality comes in handy in diverse and dynamic environments, especially when domain knowledge is required (L. Wang et al., 2024).

The most common way to access external services is to relying on specific APIs. For example, HuggingGPT (Shen et al., 2023) utilizes the HuggingFace ecosystem to solve complex tasks. In addition to reusable off-the-shelf components, custom tools or functions can be conceived to support a specific task. In addition, databases and external knowledge bases can provide domain-specific knowledge sources, enabling them to perform more grounded and meaningful actions. For example, MRKL (Karpas et al., 2022) employs various expert systems to retrieve specialized information. Finally, it is possible to access external models to expand the agent's capabilities. In contrast to APIs, external models can address more complex tasks. For example, MM-REACT (Z. Yang et al., 2023) uses several external models for video summarization.

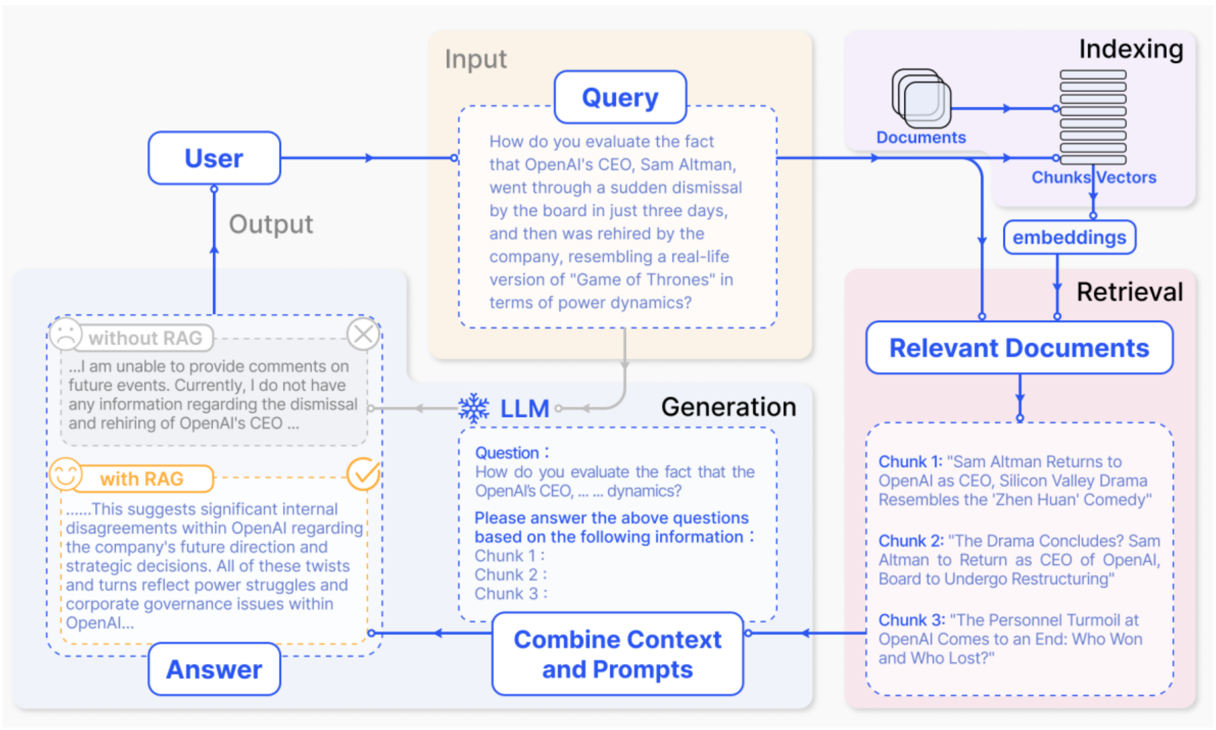
2.2.2 Memory

Memory is an essential component that allows the agent to acquire, store, and retrieve knowledge for interaction with its environment (L. Wang et al., 2024). By analyzing the literature, agents can be equipped with Short-Term, Long-Term, or a combination of both. The Short-Term Memory refers to the storage of agents' last conversation with built-in functions. This strategy is employed to handle the length of the model's context window. Meanwhile, the Long-Term memory preserves information over time and can include external sources of knowledge. implemented via vector databases that use embedding representations to support the retrieval augmented generation (RAG) technique (Gao et al., 2024; Lewis et al., 2021). In particular, this strategy focuses on enriching the original prompt by retrieving additional knowledge from long-term memory. In particular, there are three main phases, as shown in Figure 4:

– *Indexing:* First, retrieval tools are set up to extract and encode relevant information from external databases or remote resources by using well-founded algorithms, e.g., TF-IDF or BM25.

– *Retrieval:* Then, contextual information is integrated to enrich the input prompt and pass it to the model for inference.

– *Generation:* The model eventually benefits from the information, thus improving the final output with contextual information.



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question : How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

Figure 4: The RAG workflow

Although this strategy can increase the LLM-based agent's capabilities, the whole process may require time and a huge computational effort.

To address this issue, several strategies have been proposed, such as Maximum Inner Product Search (MIPS) (Weng, 2023), LSH (Lv et al., 2007), ANNOY (Bernhardsson, 2013), HNSW (Malkov et al., 2018), FAISS (Jégou et al., 2025), and ScaNN (P. Sun et al., 2020).

2.2.3 Decision-making Thought

This component is devoted to supporting two different features, i.e., Task Decomposition and Self-Reflection. The former is the LLM's ability to decompose a task into smaller yet easier subtasks. The latter is the capability of adding reasoning steps based on feedback received from an action (Guo et al., 2024; Weng, 2023).

In particular, the main task decomposition strategies are listed below:

- **Chain of Thought (CoT)** (Zhuosheng Zhang, A. Zhang, M. Li, and Smola, 2022): Unlike zero and few-shot prompting, this technique introduces reasoning steps helping the model solve complex problems step-by-step while making the reasoning process transparent. In addition to traditional CoT, the multimodal CoT architecture has been proposed (Zhuosheng Zhang, A. Zhang, M. Li, H. Zhao, et al., 2024), to combine textual files with contextual visual information such as images, charts, or tables.

- **Tree of Thoughts** (Yao, D. Yu, et al., 2023): An extension of CoT, where the model generates multiple possibilities at each step, forming a tree structure. Plans can then be selected using BFS (breadth-first search) or DFS (depth-first search), with states evaluated by classifiers or majority voting.

- **LLM+P** (B. Liu et al., 2023): Delegates planning to an external tool using a Planning Domain Definition Language (PDDL), which defines a plan and returns it to the LLM for execution.

- **Self-consistency** This technique is based on the principle that a complex reasoning problem involves multiple reasoning strategies that lead to a single correct answer. This approach has been shown to significantly enhance the CoT mechanism. The approach, unsupervised at inference time, involves sampling the outputs of the decoder to capture, from various reasoning processes, the most coherent path, selecting the final answer through voting or probabilistic mechanisms (Xuezhi Wang et al., 2023).

Once the tasks have been decomposed, a first iteration of the whole process is triggered and executed. Depending on the domain, the agent can *re-think* the performed actions, based on previous outputs or predefined external metrics. Thus, different self-reflection strategies (Weng, 2023) can be devised according to the objective, including:

- **ReAct** This paradigm (Reasoning + Acting) (Yao, J. Zhao, et al., 2023) describes a mechanism for prompt construction that combines explicit reasoning in natural language with operational actions through external tools. ReAct prompts define a structured format in which each inference follows an iterative sequence: *Thought* $\rightarrow$ *Action* $\rightarrow$ *Observation* $\rightarrow$ ... The action space is extended from A to $\hat{A} = A \cup L$, where L represents linguistic actions, such as thoughts or reflections. Thoughts do not affect the environment but enrich the context $\mathbf{c}_t$ of the agent or system, which then adopts a policy $\pi(a_t \mid \mathbf{c}_t)$ conditioned by the historical cognitive process. This structure allows the model to reason step by step and has demonstrated greater robustness, transparency, and adaptability compared to traditional prompts based only on chain-of-thought reasoning.

- **Chain of Hindsight** (H. Liu et al., 2023): Improves outputs by incorporating a list of previous outputs annotated with human feedback.

- **Reflection** (Shinn et al., 2023): In this approach, the model performs an action based on its memory, and then a separate LLM model serves as an evaluator of the model's actions. The advantage of this method is that the feedback is delivered in verbal form, allowing for more elaborate and detailed responses from the evaluator, who is also an LLM.

Although LLM-based agents can achieve a better performance compared to traditional agents, they can suffer from different limitations. In particular, the *hallucination* phenomenon (Y. Zhang et al., 2023) occurs when the underpinning LLM generates out-of-context, inaccurate, or completely wrong answers. Hallucinations might be caused by different causes.

- *Contextual Understanding:* LLMs are trained on large amounts of text data and learn to generate responses based on patterns and associations present in the training data;
- *Lack of Grounding:* Hallucination can occur when the model generates responses that are not grounded in the input context or are disconnected from the topic or subject being discussed;
- *Complexity of Language Understanding:* Natural languages are inherently complex, hallucinations may arise when the model misinterprets the meaning of the input, or fails to recognize important contextual cues;
- *Bias and Error Propagation:* Hallucinations can be triggered by biases contained in the training data.

Although we acknowledge that hallucination can be reduced by relying on advanced techniques such as RAG or fine-tuning, these require additional efforts to be developed. In particular, RAG involves the collection, storage, and curation of external data sources to augment the original prompt. Similarly, the fine-tuning requires high-quality datasets that need to be collected, pre-processed, and encoded to train the model, plus high computational resources. Moreover, fine-tuned models can suffer from the so-called *catastrophic forgetting* (Kirkpatrick et al., 2016), i.e., a significant drop in performance on previously learned tasks.

In this respect, LLM-MAS could offer a viable solution. LLM-MAS architectures employ multiple specialized agents that collaborate through activities like debate and discussion. This collaborative setup promotes divergent thinking, enhances reasoning and relevance, and ensures more rigorous validation. The following section will characterize LLM-MAS systems and their operation.

2.3 Multi-vocal Literature review on MAS

Unlike a single-agent architecture, LLM-MAS are able to orchestrate and manage a group of AI agents, each designed for specific tasks, working collaboratively to overcome the limitations of individual models. For example, in the context of software development, one agent may specialize in code generation, while another focuses on validation, ensuring that the output is both functional and secure.

In the case of MAS, these agents possess specific abilities that enhance their effectiveness and efficiency in completing tasks. These abilities can either be *predefined*, meaning their profiles remain unchanged during execution, or *dynamically* generated based on execution requirements (He et al., 2025). In addition, the agents can be *homogeneous*, where all agents perform the same task, or *heterogeneous*, where each agent has a different profile (He et al., 2025).

The second key component, the **Orchestration Platform**, serves as the central infrastructure that enables interaction between agents and manages the flow of their activities. It

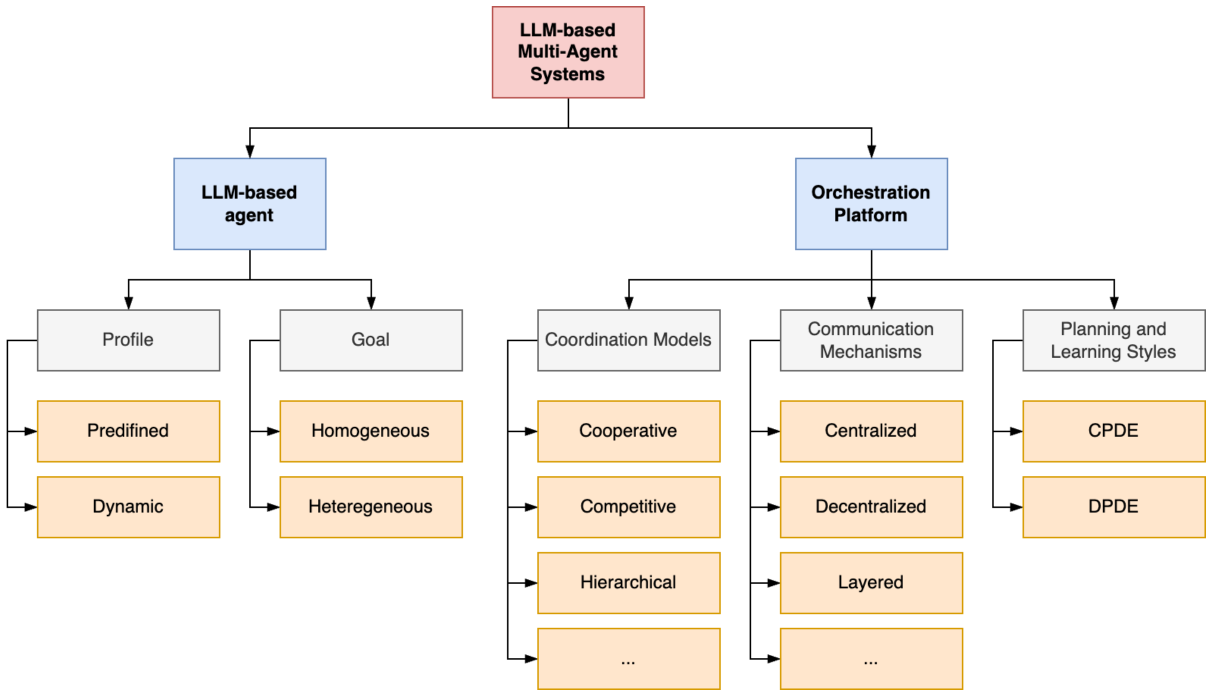


Figure 5: Overview of LLM-MAS

facilitates coordination, communication, planning, and learning. The key characteristics of an orchestration platform include *Coordination models*, *communication mechanisms*, and *Planning and Learning Styles* (He et al., 2025). In particular, the coordination models describe how agents interact with each other. They can cooperate toward a common goal by exchanging information (*cooperative*), pursue their own goals that might interfere with one another (*competitive*), follow a leader (*hierarchical*), or use a mixed approach (Guo et al., 2024; He et al., 2025). To support these interactions, communication mechanisms define how information is exchanged between agents within the same system (He et al., 2025). There are several approaches to communication among agents: *Layered Communication* involves agents communicating hierarchically, interacting only with adjacent layers. *Decentralized Communication* allows all agents to intercommunicate directly without a central authority. In *Centralized Communication*, a single agent is responsible for handling all communication, serving as a central node through which all agents interact. Finally, *Shared Message Pool* enables agents to publish and subscribe to messages only in areas of interest (Guo et al., 2024). These mechanisms collectively determine how effectively agents can collaborate and share information (He et al., 2025). Finally, planning and learning styles manage the learning phase and how tasks are distributed between agents. There are approaches such as *Centralized Planning, Decentralized Execution* (CPDE), where a plan is centrally created, and each agent executes it independently, and *Decentralized Planning, Decentralized Execution* (DPDE), where both planning and execution are distributed between agents (He et al., 2025).

To analyze more in-depth LLM-MAS pivotal components, we conduct a multivocal literature review (MLR) that involves both academic peer-reviewed works and gray literature, including tools and frameworks.

Figure 6 depicts the overall methodology composed of three main steps: *Planning*, *Conducting*, and *Mapping* according to (Kitchenham et al., 2010; Neto et al., 2019). It is worth mentioning that this deliverable will focus on the first two phases of the MLR, i.e., *Planning* and *Conducting* phases. Meanwhile, we report the *Mapping* phase in the Deliverable D2.1 Section

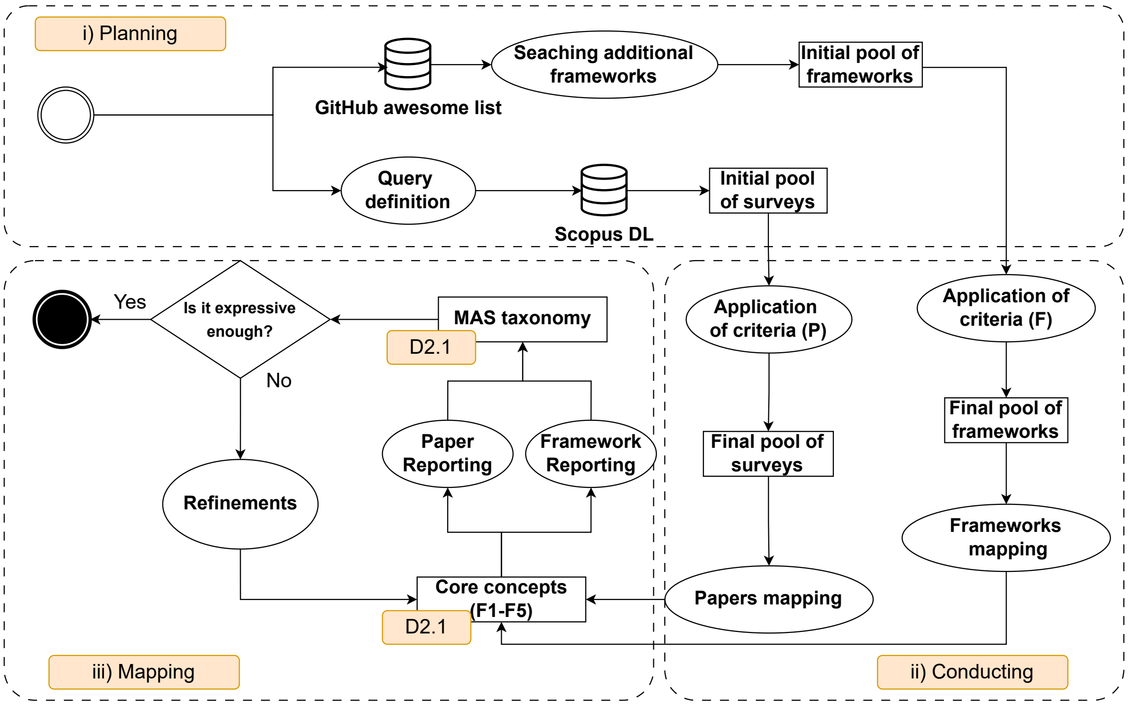


Figure 6: Overview of the conducted process.

3, as this final step was used to devise the initial architecture of the MOSAICO repository.

2.3.1 Planning

The first step involves the definition of the query and the source of knowledge, i.e., the selected digital library. In the scope of the paper, we consider the Scopus digital library[1] as it is one of the largest and most comprehensive databases of peer-reviewed literature in the field of computer science. In addition, the platform allows searching for gray literature, e.g., preprints and conference papers, that are not indexed in other digital libraries. Scopus covers preprints from 2017 onwards from arXiv, ChemRxiv, bioRxiv, medRxiv, SSRN, TechRxiv, and Research Square, thus granting a wide coverage of the most relevant preprint services. Although we acknowledge that the search is not exhaustive in terms of the identified papers, we aim to identify a "quasi-gold" set of academic works that can be used to elicit the foundational concepts of a MAS. However, we adhere to formal guidelines proposed for conducting systematic studies that also involve gray literature (Garousi et al., 2019) without running additional steps, e.g., snowballing.

The search string is composed of two groups of keywords related to LLM-MAS systems and systematic studies in software engineering, as shown in Table 1. By composing the two groups using AND and OR clauses, we obtained the Boolean query that has been applied to the title and abstract of the papers. We also limit the search to the Computer Science domain and to articles published in well-ranked venues according to the CORE ranking[2] and Scimago Journal Rank (SJR).[3] Concerning the second group of keywords, we followed the guidelines proposed

[1]https://www.scopus.com/
[2]https://portal.core.edu.au/conf-ranks/
[3]https://www.scimagojr.com/

Groups	Keywords
MAS-related keywords	MAS, multi-agent system, LLM, large-language models, pre-trained language model
Systematic studies related keywords	evidence-based software engineering survey, structured review, systematic review, literature review, literature analysis, in-depth survey, literature survey, meta-analysis, past studies, subject matter expert, analysis of research, empirical body of knowledge
Final boolean query (Title and abstract search)	(mas OR multi-agent AND system AND llm OR large-language AND models OR pre-trained AND language AND model AND "evidence-based software engineering" OR survey OR "structured review" OR "systematic review" OR "literature review" OR "literature analysis" OR "in-depth survey" OR "literature survey" OR "meta-analysis" OR "past studies" OR "subject matter expert" OR "analysis of research" OR "empirical body of knowledge")

Table 1: Keywords groups and final boolean search query

by (Kitchenham et al., 2010) for conducting *tertiary studies* in which the subject of the study is systematic reviews. Although the search string is not exhaustive, it represents a starting point to give an overview of the state-of-the-art.

We then defined a set of inclusion and exclusion criteria to filter the results as summarized in Table 2. In particular, we did not consider papers that propose a new MAS. Instead, we focused on studies that provide foundational concepts for building MAS. In addition, we imposed the following criteria: *(i)* the search string is composed of keywords related to MAS and LLMs; *(ii)* the search was limited to Computer Science; *(iii)* only articles published in well-ranked venues according to CORE ranking and Scimago Journal Rank (SJR) were selected according to notable guidelines for conducting systematic studies (Kitchenham et al., 2010; Wohlin, 2014); *(iv)* the articles must be written in English; and finally *(v)* the search was confined to the two most recent years, i.e., from January 2023 to January 2025.

For the framework selection, our primary source of knowledge is `GitHub`, as it provides a wide range of open-source projects. The inclusion and exclusion criteria that have been applied to select MAS frameworks are summarized in Table 3. In particular, we first identified a `GitHub` awesome list[4] of tools. We considered open-source frameworks that provide a proper documentation and/or a supporting `GitHub` repository. In addition, we excluded tools with less than 10,000 stars in `GitHub`. Although the number of stars might not be a decent indicator of the quality of the framework, it can be used as a proxy to evaluate the popularity of the tool (Borges et al., 2018) together with the number of forks. In addition, we checked tech blog posts, forums, and other sources to identify additional relevant tools and frameworks as they fall within the definition of gray literature provided in (Garousi et al., 2019). For the literature analysis, authors independently analyzed the tools and then discussed the results to reach a consensus on the

[4]`https://github.com/kaushikb11/awesome-llm-agents`

Table 2: Inclusion and exclusion criteria for the literature analysis.

Inclusion criteria
✓ Surveys, literature reviews, or similar that focus on MAS based on LLMs and/or pre-trained models.
✓ Articles that provide governance rules, guidelines,and other foundational concepts for building MAS.
✓ Conference and journal paper published in well-ranked venues according to CORE ranking and Scimago Journal Rank (SJR). In addition, we consider relevant papers that have not been published but available on preprint services, e.g., arXiv.
✓ Articles written in English and published in the Computer Science domain.
✓ Studies published from January 2023 to January 2025
Exclusion criteria
✖ Studies that focus on multi-agent systems but not large language models.
✖ Papers that propose a new MAS but do not provide a systematic study or comparison of them.
✖ Workshops, vision/short papers, and posters.

final set of tools.

Table 3: Inclusion and exclusion criteria for MAS frameworks.

Inclusion criteria
✓ Frameworks with supporting `GitHub` repositories.
✓ Frameworks that provide proper documentation, tutorials, or similar to extract foundational concepts of MAS.
✓ Open-source frameworks or that provide permissive licenses.
Exclusion criteria
✖ Frameworks to develop single LLM-based systems or that do not allow orchestration among agents.
✖ Frameworks that have less than 10,000 `GitHub` stars.

2.3.2 Conducting

Once we defined the search string and the inclusion and exclusion criteria, we obtained a total number of 47 papers that match the identified keywords. Then, we applied the defined criteria, ending up with 18 papers that are relevant to the scope of the paper. In particular, five of them are classified as *white literature* as they have been peer-reviewed in high-ranked journals and conferences, while the remaining papers fall within the *gray literature* category, i.e., they are available on preprint services, e.g., arxiv.

Elicited papers

The selected papers are summarized in Table 4, where we report the ID, title, source, and type of paper. They are divided into two main categories, i.e., white and gray literature (Soldani, 2020). The first category includes peer-reviewed papers published in high-ranked journals and conferences, while the second category involves preprints available on arXiv and other preprint services. The selected papers are sorted by publication date, and the ID is used to refer to them in the rest of the deliverable. In the following, we discuss the pivotal contributions of elicited works summarized in Table 5.

➤ **Foundational analysis** (He et al., 2025) envisioned a research agenda for MAS in software engineering. After collecting relevant work with an SLR, they develop two different games using MAS, i.e., Snake and Tetris, to assess the performance of MAS in solving complex programming tasks. Afterward, they identify a set of challenges and future opportunities in the field, which have been categorized into two main dimensions, i.e., enhancing individual agent capabilities and optimizing agent synergy. For single agents, it is crucial to refine role-playing capabilities to build specialized agents by following three main steps, namely *(i)* identifying key SE roles using market analysis and involving stakeholder; *(ii)* understanding the limitations of LLM-based agents such as hallucination and performance degradation; and *(iii)* tailoring AI agents with specialized knowledge, prompts, and continuous learning. For the optimization of the agents' synergy, the interaction with human experts and handling privacy concerns are the main challenges. In addition, specialized coordination mechanisms are required to ensure the agents' synergy, e.g., using a centralized coordination mechanism to manage the agents' interactions.

Similarly, (Guo et al., 2024) conducted a comprehensive overview of MAS, focusing on fundamental concepts, technical frameworks, and domain applications. In particular, they identify four main features of AI agents, i.e., environment interface, profiling, communication, and acquisition of capabilities. In addition, nine different application domains have been analyzed in terms of implementation tools, datasets employed, and open challenges.

(M. Sun et al., 2024) investigated MAS for a specific code development tasks, i.e., data science. First, a list of 20 agents has been reviewed and compared to extract specific agent roles, i.e., Agent Manager, Prompt Agent, Data Agent, Model Agent, and Operation Agent. Afterward, the authors discuss MAS concepts, in particular planning, reasoning, and reflection processes for decision-making. To highlight the benefits of the cooperation, three different case studies have been presented in which the authors tested conversational, end-to-end, and tool-based agents are employed. Although the usage of MAS is promising, several challenges are still open, e.g., improving domain-specific knowledge and multi-modal handling.

(Du et al., 2024) defined a conceptual framework for designing context-aware multi-agent systems (CA-MAS). The proposed framework is composed of five different steps, i.e., Sense,

Table 4: Elicited papers from the literature analysis.

ID	Title	Year	Venue/Source	Type
P1	Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead	2025	TOSEM	White
P2	Large Language Model Based Multi-agents: A Survey of Progress and Challenges	2024	IJCAI	White
P3	The Tyranny of Possibilities in the Design of Task-Oriented LLM Systems: A Scoping Survey	2023	arXiv	Gray
P4	The rise and potential of large language model based agents: a survey	2025	Science China Inf. Sciences	White
P5	Multi-agent, human–agent and beyond: A survey on cooperation in social dilemmas. (Mu et al., 2024)	2024	Neurocomputing	White
P6	LLM-based Multi-Agent Reinforcement Learning: Current and Future Directions	2024	arXiv	Gray
P7	Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects	2024	arXiv	Gray
P8	A Survey on Large Language Model-based Agents for Statistics and Data Science (M. Sun et al., 2024)	2024	arXiv	Gray
P9	A Survey on Context-Aware Multi- Agent Systems: Techniques, Challenges and Future Directions	2024	arXiv	Gray
P10	A Comprehensive Survey on Multi-Agent Cooperative Decision-Making: Scenarios, Approaches, Challenges and Perspectives	2025	arXiv	Gray
P11	A Survey on Trustworthy LLM Agents: Threats and Countermeasures	2025	arXiv	Gray
P12	Multi-Agent Autonomous Driving Systems with Large Language Models: A Survey of Recent Advances	2025	arXiv	Gray
P13	Multi-Agent Coordination across Diverse Applications: A Survey	2025	arXiv	Gray
P14	Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems	2025	arXiv	Gray
P15	Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG	2025	arXiv	Gray
P16	LLM-Based Multi-Agent Decision- Making: Challenges and Future Directions	2025	Robotics and Aut. Letters	White
P17	LMs Working in Harmony: A Survey on th e Technological Aspects of Building Effective LLM-Based Multi Agent Systems	2025	arXiv	Gray
P18	Multi-Agent Collaboration Mechanisms: A Survey of LLMs	2025	arXiv	Gray

Learn, Reason, Predict, and Act. In addition, context-awareness process is split into three subtasks, i.e., context acquisition, abstraction and comprehension, and utilization. The survey also explores the organizational structure of CA-MAS and how agents reach the stage of consensus

Contribution	Description	Provided In
Foundational analysis	It refers to surveys that discuss pivotal components of MAS broadly, including a comparison in different application domains	P1, P2, P8, P9, P17, P15
Behavioral analysis	Studies that focus on coordination patterns, orchestration, and agent profiling	P4, P13, P14, P16, P18
Reinforcement learning MAS	This category represents a special class of MAS based on RL models and architectures	P6, P7, P10
Quality aspects	It refers to studies that investigate qualitative aspects such as trustworthiness, fairness, and ethical concerns	P3, P5, P11, P12

Table 5: Contributions of the elicited papers

by following different consensus strategies, spanning from leader-follower to group consensus. Finally, challenges and potential application domains like IoT, energy systems, smart cities, disaster relief, autonomous navigation, and supply chains are outlined.

(Aratchige et al., 2025) proposed a survey on LLM-MAS, focusing on four critical areas, i.e., architecture, memory, planning, and frameworks. Each area is then further divided into sub-areas, spanning from planning patterns to memory management. In addition, five prominent MAS frameworks have been discussed, highlighting their strengths and weaknesses. As key findings, the authors report that the Mixture of Agents architecture and ReAct planning framework as highly effective strategies for designing and managing LLM-based multi-agent systems. In addition, memory and technology choices remain largely application-specific, underscoring the importance of aligning system components with the desired outcomes.

(Singh et al., 2025) analyzed the Agentic Retrieval-Augmented Generation (Agentic RAG) to overcome the limitation of traditional RAG systems. In particular, five types of RAG have been identified, spanning from naive RAG to advanced techniques. Afterward, a detailed taxonomy of Agentic RAG architectures, highlights key applications in different domains, e.g., healthcare, finance, and education, and examines practical implementation strategies by inspecting existing frameworks that natively support Agentic RAG, i.e., CrewAI, LangChain, and AutoGPT. Although using MAS equipped with RAG capabilities can improve traditional approaches, peculiar aspects such as multi-agent collaboration and dynamic adaptability need to be carefully designed.

➤ **Behavioral analysis** (Xi et al., 2025) proposed a general conceptual framework for LLM-based agents composed of three key parts, i.e., brain, perception, and action. In particular, the brain is composed of knowledge and decision patterns that drive the actions that are taken in a defined environment. In addition, they defined the so-called “agent society”, a conceptual framework inspired by human society, where agents are able to interact with each other and with the environment.

(L. Sun et al., 2025) analyzed the coordination mechanisms in MAS and how they can be applied to well-known application domains, i.e., search and rescue, warehouse automation and logistics, and transportation systems. In particular, three main patterns have been identified, i.e., coordinated learning, communication and cooperation, and Conflict-of-Interest resolution, to answer four guiding questions: *(i) "What is coordination?" (ii) "Why is coordination needed?" (iii) "Who to coordinate with?"* and *(iv) "How to coordinate?"* Then, the three coordination aspects and questions have been used to analyze the collected studies in the four different domains.

(Yan et al., 2025) presented a communication-centric perspective on LLM-based multi-agent systems, examining key aspects spanning from system-level features to communication goals and strategies. Similarly to our approach, the authors select previous studies on MAS to investigate four main aspects, i.e., strategies, paradigms, object, and content of the communication process.

Sun et al. (C. Sun, Huang, and Pompili, 2025) investigated MAS applied to decision-making (DM) systems by surveying both single and multi-agent systems. In particular, they reviewed ten different frameworks, analyzing them in terms of the employed datasets, the type of LLM architecture, and the roles related to the DM process, i.e., communication, planning, and decision-making. The analysis of the selected frameworks is complemented with a discussion on the challenges and future opportunities in the field, including the need of a proper evaluation framework to assess safety and security, integrating human feedback, and integration with traditional MARL systems.

(Tran et al., 2025) conducted a survey on multi-agent collaboration mechanisms by identifying eight relevant surveys and four key aspects related to the collaboration process, i.e., multi-agent collaboration, collaborative aspects, general framework for MAS, and analysis of real-world applications. The paper eventually provides a systematic approach to analyze and design collaborative interactions within MASs empowered by LLMs, highlighting a set of challenges such as the need for a unified governance and shared decision-making process.

➤ **Reinforcement learning MAS** Focusing on reinforcement learning (RL) processes, (C. Sun, Huang, and Pompili, 2024) presented a comparison of ten different LLM-based multi-agent frameworks based on RL, the so-called MARL systems. In particular, the analysis involves cooperative tasks of multiple agents with a common goal and communication among them. Each framework has been analyzed in terms of application domain, the exploited datasets, if they require additional training, and the role of the LLM agents, e.g., decision, planning, or communication. The authors provide a discussion, highlighting the challenges and future opportunities in the field, including the need for a proper evaluation framework to assess safety and security, integrating human feedback, and integration with traditional MARL systems.

Similarly, (Cheng et al., 2024) reviewed LLM-MAS based on RL, focusing on the memory handling, communication mechanisms and different application domains. In particular, the authors first analyzed 34 papers that propose a single agent system in various application domains, i.e., Social Sciences, Code development, Games, Healthcare and Sciences, and Law. This preliminary analysis has been conducted to elicit peculiar single-agent features. Afterward, they considered 11 MAS to cover all the analyzed domains. The framework has been analyzed in terms of exploited data, communication mechanism, support of external tools, and type of environment, i.e., text-based or multi-modal.

(Jin et al., 2025) explored multi-agent cooperative decision-making (MACD) systems, analyzing the interplay between multi-agent reinforcement learning (MARL) systems and large language models (LLMs). The paper first categorizes MACD into five different categories, i.e., Rule-Based Systems, Game Theory-Based, Evolutionary Algorithms-based, MARL-based, and LLM-based. Starting from this initial category, the authors explore MARL-based system in terms of governance rules, environments, learning methodologies, and communication protocols. The MACD systems are eventually investigated in three different main application domains, i.e., social science, engineering, and natural science.

➤ **Quality aspects** Starting from the definition of a minimal task-oriented LLM system, Dhamani and Maher (Dhamani et al., 2023) proposed a survey focused on prompting and uncertainty estimation. To this end, they presented a *thought experiment* based on hypothesis and assumptions where single and multi-agent systems have been compared in terms of different design parameters, e.g., training data, context size, and number of agents. Based on this analysis, a set of conjectures has been formulated to guide future research in the field.

(Mu et al., 2024) examined two main kinds of cooperation in MAS system, i.e., multi-agent cooperation and human-agent cooperation from the social dilemma (SD) perspective. Concerning the first type, AI agents have been analyzed using the concept of intrinsic and external motivation. Concerning the second type of cooperation, the authors' overview current AI algorithms for cooperating with humans, shedding light on the peculiar phenomenon of the "bias towards AI agents", i.e., the situation where humans do not trust AI agents due to their black-box nature. The two cooperation models have been used to envision a set of future research directions, including the need for a proper evaluation framework to assess the design of MAS in terms of SD theorems. Even if this analysis does not directly involve software development, it provides a set of guidelines to design MAS that can be reused in different domains.

(M. Yu et al., 2025) focused on trustworthy in LLM-based agents and MAS, by considering five key aspects, i.e., safety, privacy, truthfulness, fairness, and robustness. In particular, the authors classify existing MAS in terms of brain, memory, tool, and agent-to-agent communication by mimicking the attack-defense mechanism widely adopted in cybersecurity. The findings on the study reveal that the MAS system can be a promising solution to improve current approaches, maintaining trust requires innovative even though monitoring–using supervisor agents needs to be properly defined.

(Y. Wu et al., 2025) selected, reviewed, and categorized both single LLM-based approaches and MAS for Autonomous Driving Systems (ADS). The authors first derived a taxonomy based on three main concepts, i.e., Multi-agent communications, Human-agent, and Applications, focusing on the interaction paradigms between the agents. The results shed light on three main open challenges, including hallucination propagation, the need for multi-modal capabilities, and the scalability issues in MAS.

Takeaways from papers: Overall, the elicited papers cover foundational aspects of MAS, such as orchestration, profiling, and perception. Interestingly, three papers focus on a special class of MAS, namely those combined with RL architectures, while only four surveys explicitly mention ethical concerns.

Elicited framework

Concerning the MAS frameworks, we started from a pool of 20 different frameworks that are publicly available on `GitHub`. After applying the criteria, we excluded three frameworks that have fewer than 10,000 stars, while three of them require a subscription. Therefore, we ended up with a final set of 14 frameworks that are relevant to the scope of the deliverable. The selected frameworks are summarized in Table 6, where we report the ID, the name, the first release date, the license type, and the number of stars, forks, contributors, and *dependants*, i.e., other GitHub projects that use the framework[5]. It is worth mentioning that evaluating and comparing each framework goes beyond the scope of this document. Instead, we aim to elicit

[5]All the repositories statistics are updated at 22-09-2025

concepts that can allow researchers and practitioners to build their own multi-agent systems. In what follows, we provide a brief description of the selected frameworks.

Table 6: Overview of MAS frameworks.

ID	Name	First release (Total)	License	# Stars	# Forks	# Contrib.	# Dep.
F1	AutoGPT	Apr 12, 2023 (53)	MIT	174,000	45,400	781	1
F2	LangChain[a]	Jan 16, 2023 (879)	MIT	105,000	17,000	3,784	270,534
F3	Dify	May 15, 2023 (130)	Apache 2.0	85,700	12,700	993	0
F4	MetaGPT	Jul 15, 2023 (22)	MIT	53,500	6,300	113	118
F5	AutoGen	Sep 19, 2023 (87)	CC-BY-4.0, MIT	42,100	6,300	558	3,822
F6	Llama Index	Jan 29, 2023 (440)	MIT	40,400	5,700	1,679	22,801
F7	Flowise	May 30, 2023 (67)	Apache 2.0	36,500	19,100	256	0
F8	CrewAi	Nov 14, 2023 (82)	MIT	29,000	3,900	273	17,032
F9	Semantic Kernel	Apr 25, 2023 (203)	MIT	23,700	3,600	411	2,177
F10	Agno	Jan 30, 2025 (54)	MPL 2.0	21,900	2,900	297	0
F11	Haystack	Nov 28, 2019 (165)	Apache 2.0	20,000	2,100	309	1,181
F12	OpenAI SDK[b]	Mar 11, 2025 (13)	MIT	19,400	2,100	15	0
F13	Smolagents	Dec 27, 2024 (20)	Apache 2.0	15,700	1,400	183	0
F14	Letta[c]	Oct 27, 2023 (163)	Apache 2.0	18,482	1,922	142	0

[a] Langchain is the backend of LangGraph `https://www.langchain.com/langgraph`
[b] OpenAI SDK has replaced SwarmAI
[c] Letta is based on top of the MemGPT(Packer et al., 2024) research paper

AutoGPT (*Autonomous-GPT* 2024) is an open source autonomous AI agent that aims to achieve user-defined goals by autonomously breaking them down into smaller, manageable sub-tasks and leveraging the Internet and various other tools to accomplish them. To facilitate the development of LLM-MAS, AutoGPT provides an intuitive interface for users to create workflows tailored to specific tasks, utilizing one or more agents. This makes it a valuable tool for non-programmers and for rapid prototyping.

LangChain (LangChain Team, 2024) stands as a comprehensive framework for constructing applications driven by LLMs, with significant support for multi-agent workflows facilitated by its LangGraph component. LangGraph conceptualizes multi-agent workflows as graphs, where individual agents are represented as subgraphs, and the connections between these nodes define the flow of control and communication within the system. A fundamental aspect of this approach involves the definition of independent agents, each potentially equipped with its own unique prompt, LLM, set of tools, and custom code tailored for its specific role and collaborative interactions.

The Dify tool (Dify Contributors, 2024) supports the creation of custom agents that possess the ability to independently utilize a variety of tools to handle intricate tasks autonomously. A key component within the Dify workflow system is the Agent Node, which enables autonomous reasoning by integrating pluggable “Agent Strategies” such as ReAct and Function Calling. To facilitate the development of these custom reasoning strategies, Dify provides a Standardized Development Kit.

Similarly to AutoGPT, MetaGPT (MetaGPT Team, 2024) exploits Standardized Operating Procedures (SOPs) to instruct agents with human-like domain expertise, thus efficiently breaking down complex tasks into subtasks and reducing errors.

AutoGen (*Microsoft AutoGen* 2024) supports the development of conversable agents, which are capable of sending and receiving messages to initiate and continue dialogues, thus facilitating collaborative task completion. Furthermore, the platform offers an intuitive visual Orchestration Studio that allows users to design AI Applications and complex workflows with ease.

LlamaIndex (*Llama Index* 2024) facilitates the creation of LLM-MAS through its AgentWorkflow class, enabling the generation of systems where multiple agents can collaborate and seamlessly hand off control to one another as needed. This allows for the development of sophisticated systems in which individual agents with specialized roles can work together in a predefined sequence or based on specific conditions. However, it does not provide a visual interface for agent orchestration even though it is possible to generate a visual representation of the workflow using the Graphviz library.[6]

Similarly to Dify, Flowise (Flowise Contributors, 2024) is a low-code framework that allows users to create and deploy AI agents by means of a visual interface. It supports the integration of various LLMs and tools, enabling users to build complex workflows without extensive programming knowledge. Flowise also provides a marketplace for sharing and discovering pre-built agents and workflows.

CrewAI (CrewAI Team, 2024) focuses on the creation of teams of AI agents, each with clearly defined roles, access to specific tools, and distinct goals, thereby optimizing both autonomy and collaborative intelligence. The architecture of CrewAI comprises several key components, including Crews, which manage the overall team of AI agents; AI Agents, the specialized team members with specific roles and expertise; Process, the system that manages the workflow and collaboration patterns; and Tasks, the individual assignments given to the agents.

Semantic Kernel (SK) (*Microsoft Semantic Kernel* 2024), an open-source toolkit developed by Microsoft, serves as a versatile platform for integrating cutting-edge LLM technology into a wide range of applications, including the construction and orchestration of both individual AI agents and complex LLM-MAS. The framework adopts a model-agnostic design, ensuring compatibility with various prominent LLMs such as OpenAI, Azure OpenAI, and Hugging Face, providing developers with the freedom to select the most suitable model.

Agno (Agno Contributors, 2025), formerly known as Phidata, is presented as a lightweight library for constructing AI agents that possess memory, knowledge, tools, and reasoning capabilities. The framework places a strong emphasis on performance and minimalism, boasting rapid agent instantiation times and a small memory footprint. Agno adopts a model-agnostic approach, offering compatibility with more than 23 different model providers, thus providing developers with significant flexibility.

Haystack (Deepset.ai, 2024) stands out for its rich set of pre-built components, including retrievers, generators, evaluators, and document stores, making it easy to build sophisticated pipelines out of the box. Its component-based architecture supports modular and flexible design, enabling rapid prototyping and customization. Haystack also provides a visual interface for building and managing pipelines, making it accessible to users with varying levels of technical expertise even if the orchestration must be handled manually.

OpenAI agents, formerly known as Swarm (*OpenAgents* 2024), is an experimental framework developed by OpenAI, designed for building, orchestrating, and deploying multi-agent systems, with a focus on being lightweight, highly controllable, and easily testable. The framework's core abstractions are Agents and handoffs. An Agent in Swarm is defined by its instructions, which serve as the system prompt, and its tools, which are Python functions that the agent

[6] `https://graphviz.org/`

can call to perform specific actions. These instructions can be either static text or dynamically generated text based on context variables, allowing flexible agent behavior. Handoffs represent the mechanism for transferring control of a conversation from one agent to another.

Smolagents (Smolagents Team, 2024) is hosted on Hugging Face and provides predefined classes and functions to integrate MAS with the platform services, e.g., datasets, models, and spaces. In addition to the core architecture capabilities, Smolagents grants access to different LLM providers, integration with external telemetry tools, and open-source infrastructure that allows us to run AI-generated code in secure isolated sandboxes, e.g., E2B[7] or Docker.[8]

Different from the other frameworks, Letta (Letta Contributors, 2024) allows a fine-grained memory handling, thus supporting self-improvement of agents. In particular, agents have a self-editing memory that is split between in-context memory and out-of-context memory, organized into memory blocks. Furthermore, the framework supports the MCP protocol and the creation of local agents.

To further analyze the elicited frameworks, we define a set of features (FW1-FW10) that have been extracted from the documentation. In particular, we aim at understanding how a notable framework can support developers in specifying MAS according to:

FW1 - MAS core architecture We analyze if the tool allows for the definition of the core features of a multi-agent system identified in the literature, i.e., definition of agents, orchestration, concept of memory and external tools as defined in existing studies.

FW2 - MAS type Whether the tool enables integration of custom agents or agents from different architectures (e.g., AI models, transformers). If not provided, the system is considered Homogeneous; otherwise Heterogeneous.

FW3 - Role specification Checks if the platform allows defining agent roles, e.g., PromptAgent, SupervisorAgent, or other role types.

FW4 - Tool support Indicates whether the tool permits usage of external tools, APIs, or other services.

FW5 - Remote access to agents Evaluates if the tool facilitates remote access to agents, e.g., through a web interface, or if agents are available only locally.

FW6 - Agent monitoring Assesses whether the platform provides monitoring facilities such as evaluation metrics, logging, or telemetry.

FW7 - Human feedback integration Determines if it is possible to embed human feedback in the agent decision-making process. It is worth noting that chat interaction alone is not considered human feedback).

FW8 - Agent comparison Relates to functionality that allows comparing agents, e.g., via a web interface or other mechanisms.

FW9 - Reuse of benchmark datasets Whether the platform enables reuse of benchmark datasets for agent evaluation through a specific interface or API.

FW10 - Discovery capabilities Checks for discovery features such as a store or marketplace for agents or tools, enabling discovery of new agents or functionalities.

[7] `https://e2b.dev/`

[8] `https://www.docker.com/`

Table 7 provides an overview of the selected MAS frameworks and their support for the ten features. In particular, we checked the documentation and similar documents to indicate if a feature is fully supported ●, partially supported ◐, or not supported ○. In addition, we grouped them according to the underpinning paradigm, i.e., low-code or high-code. The former characterizes framework that facilitate the MAS development by providing graphical capabilities for newcomer developers. The latter category represents traditional frameworks that require developers to write code to implement the MAS even though they may provide some built-in functions and classes to facilitate the development.

From our analysis, we found that the most complete framework is LangChain F2, i.e., it provides support (full and partial) for all the elicited features. This is quite expected as it was one of the first MAS frameworks to be developed, and it has been continuously improved since its release. Similarly, Dify F3 supports all the identified features except for FW6, i.e., Agent Monitoring. Contrariwise, MetaGPT F4, CrewAI F8, and Semantic Kernel F9 focus on core MAS features, neglecting the most advanced ones like benchmarking and monitoring.

Overall, the low-code frameworks cover more features compared to high-code, especially qualitative aspects like benchmarking, monitoring, and evaluation. On the one hand, low-code frameworks are designed to be more user-friendly and accessible, often providing built-in tools for these aspects. On the other hand, high-code frameworks focus more on flexibility and extensibility, leaving qualitative aspects to be implemented by developers as needed.

Table 7: Comparison of MAS frameworks in terms of covered features.

	FW1	FW2	FW3	FW4	FW5	FW6	FW7	FW8	FW9	FW10
					Low-code frameworks					
F1	●	●	●	●	●	○	○	○	○	●
F3	●	●	●	●	●	○	●	●	●	●
F7	●	●	●	●	●	●	○	◐	○	●
F11	●	●	●	●	●	●	○	●	◐	○
					High-code frameworks					
F2	●	●	●	●	●	◐	●	●	◐	●
F4	◐	●	●	●	○	○	◐	○	○	○
F5	●	◐	●	◐	◐	○	●	○	●	○
F6	●	●	●	●	●	◐	●	●	○	○
F8	●	●	●	●	●	○	●	○	○	○
F9	●	●	●	●	●	○	●	○	○	○
F10	●	●	●	●	●	○	○	○	○	●
F12	●	●	●	●	○	○	○	◐	○	●
F13	●	●	●	●	●	●	○	◐	○	○
F14	●	●	●	●	●	●	●	○	○	○

Takeaway from frameworks: Most of the popular frameworks support the majority of the identified features, with LangChain being the most complete one. However, there are still some gaps in terms of agent monitoring and evaluation, which are not fully supported by few frameworks. In particular, the low-code platforms tend to cover more qualitative aspects compared to high-code ones, even though the integration between the two paradigms is supported.

2.4 Conclusion

In this section we conducted a multivocal study to analyze both academic surveys and well-founded GitHub projects related to LLM-MAS. Although this study is not exhaustive, we adopted notable guidelines in software engineering to cover the most popular and established studies.

In particular, our findings are summarized as follows:

- Existing literature has started to analyze and compare different MAS, focusing mostly on core functionalities and testing them in miscellaneous application domains. However, advanced qualitative aspects like trustworthiness or monitoring are still at an early stage.
- Dedicated MAS frameworks are flourishing, even though most of them focus on core features rather than more advanced ones, confirming the outcomes of the academic literature. In this respect, we foresee a rapid evolution of those technologies that will address the gap, given the fast-evolving nature of the field

In the context of MOSAICO, this snapshot will also give foundational concepts for elaborating on the internal repository in terms of needed metadata and recent technologies. In addition, the project's activities will focus on creating a dedicated language to facilitate the re-use, comparison, and monitoring of existing LLM-based agents, inspired by existing well-founded frameworks.

3 BDI for AI Agents

In this section, we investigate the integration of the BDI framework from the classical literature of multi-agent systems into the new paradigm of LLM-based agents.

The MOSAICO framework will need to reason about beliefs, desires, and intentions of its agents. This is useful, e.g., for selecting the agent whose objectives best match the global task, warning the user of the intention to perform a modification, evaluating that the beliefs of the agent are not faulty. For example, several reasons may cause two MOSAICO supervision agents to give different assessments: 1) they may have different assumptions, i.e. beliefs (e.g. they analyze slightly different parts of code, or different states of the system), 2) they may want to assess w.r.t. different criteria, i.e. desires (e.g. a requirement is not correct because is it ambiguous, redundant, or conflicting with other requirements), or 3) they may have followed a different process, i.e. intentions (e.g. one has used a reliable external tool, another LLM inference).

We provide two integration mechanisms for BDI in MOSAICO: 1) we extend LLM-based agents with semantic actions representing the evaluation steps of the BDI reasoning cycle (Section 3.2), 2) we integrate an existing domain-specific language (DSL) for describing BDI behavior into LLM-based agents (Section 3.3). Users can choose the first mechanism if they do not want to commit to coding agents in a dedicated programming language, or the second for exploiting behavior analysis or reusing (or integrating with) existing BDI behaviors.

Such mechanisms are accompanied by actual working code. This code has the two-fold objective of illustrating the concrete semantics of the integration pattern, and providing executable prototypes for other work-packages. Concrete experimentation on BDI agents requires a working agent orchestrator, but the first version of the MOSAICO orchestrator will be available only at M15 within D3.2 (Orchestrator Framework for AI Communities - Initial Version). As a consequence, we decide in this first deliverable to reuse an existing open-source orchestrator. We opt for Microsoft AutoGen (listed in Section 2.3), due to its documentation quality, supporting community, and usable tooling. For these reasons, in the rest of the section we illustrate each solution as an extension of base agents from AutoGen. The integration patterns can be replicated for other LLM-based agentic frameworks (see Section 2.3), and will be featured in the final MOSAICO implementation.

LLM-based BDI agents can be applied in diverse use cases, where LLMs are used either 1) as an integral part of the BDI cycle (i.e. to analyze the environment, derive beliefs, choose plans, commit to intentions) or 2) to provide natural language interfaces to agents that have a deterministic BDI behavior. We provide examples of such use cases in the final section.

The section is structured as follows. In Section 3.1 we introduce the basic concepts of BDI. The two solutions are described in Section 3.2 and Section 3.3. The use cases are finally described in Section 3.4.

3.1 Belief-Desire-Intention

Inspired by philosophical and psychological theories about human cognition (Bratman, 1987), the Belief-Desire-Intention (BDI) model is a theoretical framework used to explain and design rational agents (Georgeff et al., 1999), i.e., entities that can reason, plan, and act autonomously and purposefully in their environment. It is built on three foundational mental attitudes:

- *Beliefs:* Represent the **information** an agent has about its environment, itself, and other agents, e.g. a robot might believe that a door is locked based on sensor data. This results in an agent's knowledge base, which can be updated as new information is perceived, e.g. the door is now unlocked.
- *Desires:* Represent the **motivations** or objectives that the agents aim to achieve. These can range from simple tasks to complex and long-term goals that may conflict, thus requiring the agent to prioritize or reconcile them.
- *Intentions:* Represent the **commitments** an agent makes to act on specific desires given its beliefs. Unlike desires, which are potential goals, intentions represent the agent's chosen course of actions, e.g. a robot might intend to "unlock the door" to fulfill its desire to "enter the room" when it believes that "the door is locked".

The BDI model operates as a cycle of reasoning and action. At first, the agent perceives its environment to update its beliefs. It then deliberates over its desires, selecting which to pursue based on its current beliefs. It commits to these chosen goals and forms a plan to achieve them. Finally, it acts in the environment, executing its plan and adjusting as needed. This cycle repeats continuously as the environment changes and thus allows BDI agents to be **reactive**, i.e. responding to changes in real-time, and **proactive**, i.e. pursuing long-term goals.

Before presenting our implementations (Section 3.2 to 3.4), we present AgentSpeak (Section 3.1.1), a specialized programming language designed for implementing BDI agents, then we analyze how the integration of LLMs with BDI frameworks can significantly enhance agents' adaptability and reasoning capabilities (Section 3.1.2).

3.1.1 AgentSpeak

The AgentSpeak language (Rao, 1996) was introduced to allow the formalization of agent behavior based on BDI concepts. The language is declarative and event-driven. An AgentSpeak program is a set of rules called *plans* that describe how an agent reacts to events (coming from the environment) in the context of the beliefs it has. Here is an example of a rule that describes a robot behavior:

```
+location(waste,X) : location(robot,X) & location(bin,Y) <-
    pick(waste);
    !location(robot,Y);
    drop(waste).
```

A plan is made of three parts:

- The *triggering event* on the left hand side of the plan (before the colon). Events can be the addition of a belief, starting with +, such as `+location(...)`; the addition of a goal, with a `!`, such as `+!location(...)`; the deletion of a belief or a goal, starting with `-`. Variables, starting with an uppercase letter, are bound by triggering events (`X` in the example).
- Between the colon and the arrow is the *context*, that specifies beliefs that must hold to trigger the plan. Variables can be bound here too (`Y` in the example).
- The *body*, after the arrow, is a sequence of *actions*, such as `pick(...)`, and new goals, such as `!location(...)`. When a plan is triggered, that sequence becomes the current *intention* of the agent, and when a sub-goal of that sequence is being processed, it becomes its current *desire*.[9]

The rule above specifies that when the robot gets the information that there is waste at a position X, given that its own position is also X and that there is a bin at position Y, then it should pick the waste, try to go to the position Y, and then drop the waste.

Other rules can describe the behavior intended to achieve the goal `!location(robot,Y)`. The language is inspired by the logic programming paradigm: beliefs are implemented by literals, behaviors are described by rules, and the rule selection is powered by unification and backtracking. Thus, the behavior of AgentSpeak agents is formally defined by the semantics of the AgentSpeak language (see (Rao, 1996)).

Multi-agent systems generally support communication between agents. AgentSpeak agents in the Jason[10] platform can send messages to each other following so-called *speech-act based inter-agent communication*. Each message contains an *illocution* to specify whether the message is belief sharing (`tell`), a new goal (`achieve`) or even an update in an agent's plans (`tellHow`). Here is an example of plan with an action to send a message.

```
+!share_secret : secret(X) <-
    .send(receiver, tell, secret(X)).
```

Here, `.send` is not a keyword of the AgentSpeak language but an action provided by the Jason platform. The reception of such a `tell` message managed by the agent's environment which triggers the addition of a new belief in that agent's state. Next, that agent can react to that event, or use that information later, according to its programmed behavior.

[9]More precisely, since the execution of a plan can activate other plans, the current intention of an agent results from the stack of plans which are currently activated, and several desires can be active simultaneously.

[10]`https://jason-lang.github.io`

This kind of messaging is also supported by other platforms for MAS with AgentSpeak support. Jason is the most popular AgentSpeak platform, but other platforms have been developed concurrently. Jason is implemented in Java, and the user can add actions implemented in Java. Spade,[11] a MAS developed in Python, also has an extension for AgentSpeak called Spade-bdi.[12] The language is intended to be compatible with the Jason variant of AgentSpeak. Internal actions can be implemented in Python. AgentSpeak support is also available in a few other platforms, such as LightJason[13] or SuperAGI,[14] but those are less relevant to MOSAIC.

Thanks to the formal semantics of AgentSpeak, several works provide formal analysis of agent behavior (Bordini, Fisher, Pardavila, et al., 2003; Bordini, Fisher, Visser, et al., 2004; Bordini, Fisher, Wooldridge, et al., 2009; Guerra-Hernández et al., 2009), which can be used to assess MOSAICO agents.

3.1.2 BDI and LLMs

Very little research has been published on the integration of BDI in LLM-based agents. To the best of our knowledge, only three articles are available, presenting initial work and a vision for future research (Frering et al., 2025; Gatti et al., 2025; Ichida et al., 2024). The approaches can be divided into two groups, depending on the role of LLM inference, w.r.t. the behavior of the agent. We represent the two approaches in Figure 7.

The first approach (Ichida et al., 2024) is shown on the left-hand side of Figure 7. The LLM is used within the reasoning cycle of BDI, either to analyze environment and events to derive knowledge in terms of beliefs, or to match beliefs against plans to derive intentions, or to apply complex intentions to the environment and generate new events. This aims to enhance the reasoning capabilities of BDI agents, to provide them with general knowledge, and to increase their flexibility in addressing unknown situations.

The second approach (Frering et al., 2025; Gatti et al., 2025) is shown on the right-hand side of Figure 7. In this case, the LLM is used to interface the mental model of the agent with the user, by translating beliefs, desires, intentions, and events into (and from) natural language. This may greatly enhance the ability of the user to understand and control the agent behavior. The agent behavior itself is not impacted by the LLM: it is defined programmatically by one of the traditional methods in the multi-agent systems literature.

In MOSAICO, we want to support these two approaches. We show in Section 3.4 that we can apply our BDI-augmented LLM agents to use cases that fit into the first (Section 3.4.1 and 3.4.2) and second (Section 3.4.3) category shown in Figure 7.

3.2 Ad-hoc BDI on AutoGen

BDI Components in AutoGen Agents

We first extend AutoGen agents with BDI services.[15] AutoGen is a multi-agent platform developed in Python that provides an uniform support for dealing with LLMs (see Section 2.3). Communication between agents is based on a publish/subscribe model. Agents are described

[11] `https://spadeagents.eu`
[12] `https://github.com/javipalanca/spade_bdi`
[13] `https://lightjason.org`
[14] `https://superagi.com/superagi_apps/agent-speak-toolkit/`
[15] Code available at `https://gitlab.eclipse.org/eclipse-research-labs/mosaico-project/bdi-on-autogen`

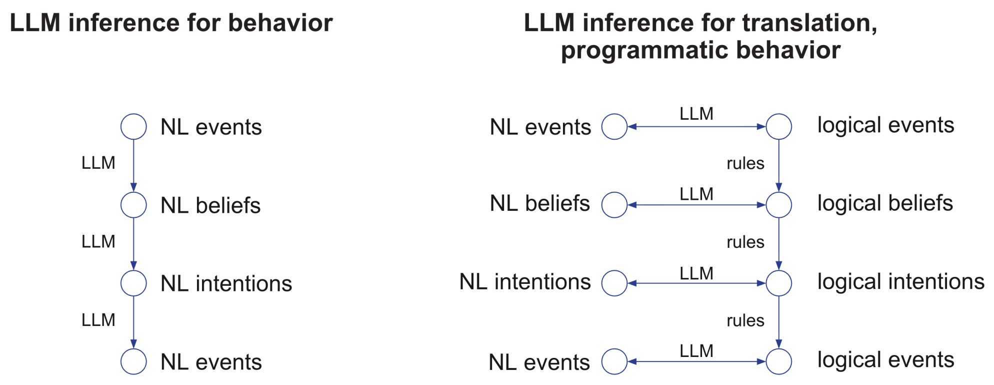


Figure 7: Roles of LLM inference in BDI agents

```
@type_subscription(topic_type="ATopic")
class MyAgent(RoutedAgent):
    def __init__(self) -> None:
        super().__init__("An agent that does something.")

    @message_handler
    async def handle_message(self, message: MyMessage, ctx: MessageContext) -> None:
        print(f"{self.id.type} received :\n{message.content}")
```

Listing 1: Typical use of the RoutedAgent class from AutoGen

by classes and are instantiated or run when they need to respond to an event (according to their subscription). While a set of typical agents is provided in the `autogen-agentchat` library, user can define their specific agents by sub-classing the `RoutedAgent` class from the `autogen-core` package. A typical example of such a sub-class in AutoGen is pictured in Lst. 1. This example shows how the subscription is done with the `@type_subscription` annotation, how the super-constructor is called in the constructor, and how the message handler is declared with the `@message_handler` annotation. Note that the type for messages, `MyMessage` here, is user defined while the `MessageContext` type is defined by AutoGen and is used to carry routing information.

Our BDI extension is provided by the class `BDIRoutedAgent` in Fig. 8, that users can sub-class instead of sub-classing the standard `RoutedAgent` from AutoGen.

In this setting, the reception of an AutoGen message by a `BDIRoutedAgent` agent triggers the `bdi_loop` method, which triggers a sequence of `bdi_observe`, `bdi_select_intention`, `bdi_act` method invocations, following the classical BDI interaction loop described in Section 3.1. The specific code for a new agent is given by implementing those three methods which are abstract in the `BDIComponent` class. Those methods can refer to standardized beliefs, intentions and desires objects (classes `BeliefStore`, `IntentionStore` and `DesireStore` in the diagram) to support each step.

To illustrate that hierarchy, we implement below a multi-agent program to build a list of requirements from a given specification.

Example

In this section, we describe the collaboration of 6 basic agents to produce a list of atomic requirements from a specification given by the user. Those 6 agents are displayed in the

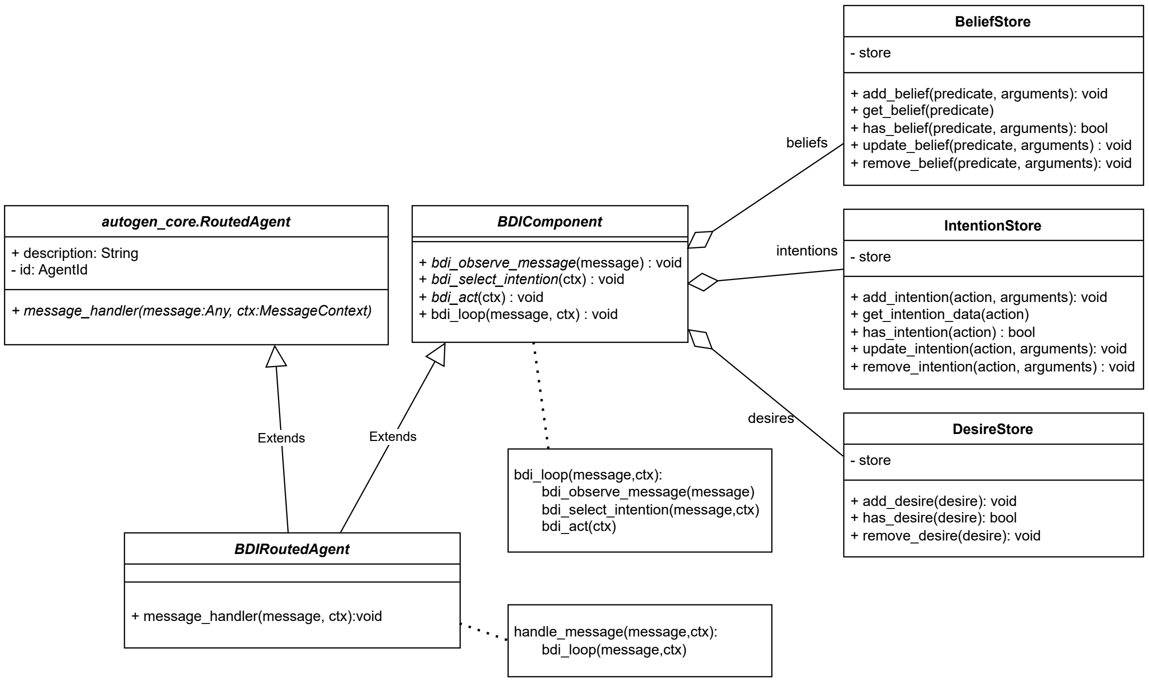

BeliefStore
- store
+ add_belief(predicate, arguments): void
+ get_belief(predicate)
+ has_belief(predicate, arguments): bool
+ update_belief(predicate, arguments) : void
+ remove_belief(predicate, arguments): void
beliefs
autogen_core.RoutedAgent
+ description: String
- id: AgentId
+ message_handler(message:Any, ctx:MessageContext)
BDIComponent
+ bdi_observe_message(message) : void
+ bdi_select_intention(ctx) : void
+ bdi_act(ctx) : void
+ bdi_loop(message, ctx) : void
intentions
IntentionStore
- store
+ add_intention(action, arguments): void
+ get_intention_data(action)
+ has_intention(action) : bool
+ update_intention(action, arguments): void
+ remove_intention(action, arguments) : void
desires
DesireStore
- store
+ add_desire(desire): void
+ has_desire(desire): bool
+ remove_desire(desire): void
Extends
Extends
bdi_loop(message,ctx):
bdi_observe_message(message)
bdi_select_intention(message,ctx)
bdi_act(ctx)
BDIRoutedAgent
+ message_handler(message, ctx):void
handle_message(message,ctx):
bdi_loop(message,ctx)

Figure 8: Class diagram of our ad-hoc extension of AutoGen RoutedAgent. Map<String,String>

communication diagram in Figure 9. Each agent is specified by a different class inheriting from `BDIAgent`.

The `RequirementManagerAgent` agent receives a specification (message n° 1) and manages a loop where a list of atomic requirements is built iteratively until a decision to stop is taken. This decision is taken based on an interaction with an LLM, which is given the specification, the current list of requirements, and which is asked if the requirements cover completely the specification. If the answer is yes, the list of requirements is returned to the user (message n° 6 in the diagram). Otherwise, the specification and current requirements are given to the next agent: `RequirementDecomposerAgent` (message n° 2).

The `bdi_observe_message` method implementation is just a call to a function that populates the belief base with the incoming specification:

```
def bdi_observe_message(self, message):
    message__bdi_observe_message(self, message)
```

That function, `message__bdi_observe_message` (below), is shared between all the agents from this example since many messages contain the same information: the specification and a list of requirements.

```
def message__bdi_observe_message(d: BDIComponent, m: Message):
    d.update_belief(m.initial_description, spec_tag)
    d.update_belief(m.current_list, req_list_tag)
```

The `bdi_select_intention` method implementation (Lst. 2) builds a prompt with the specification, the requirements, and a request to evaluate the completeness of the requirements, then submits it to an LLM. if the LLM answers that the requirements are complete ("COMPLETE") then the agent selects the intention "STOP". Otherwise, it selects the intention "PASS".

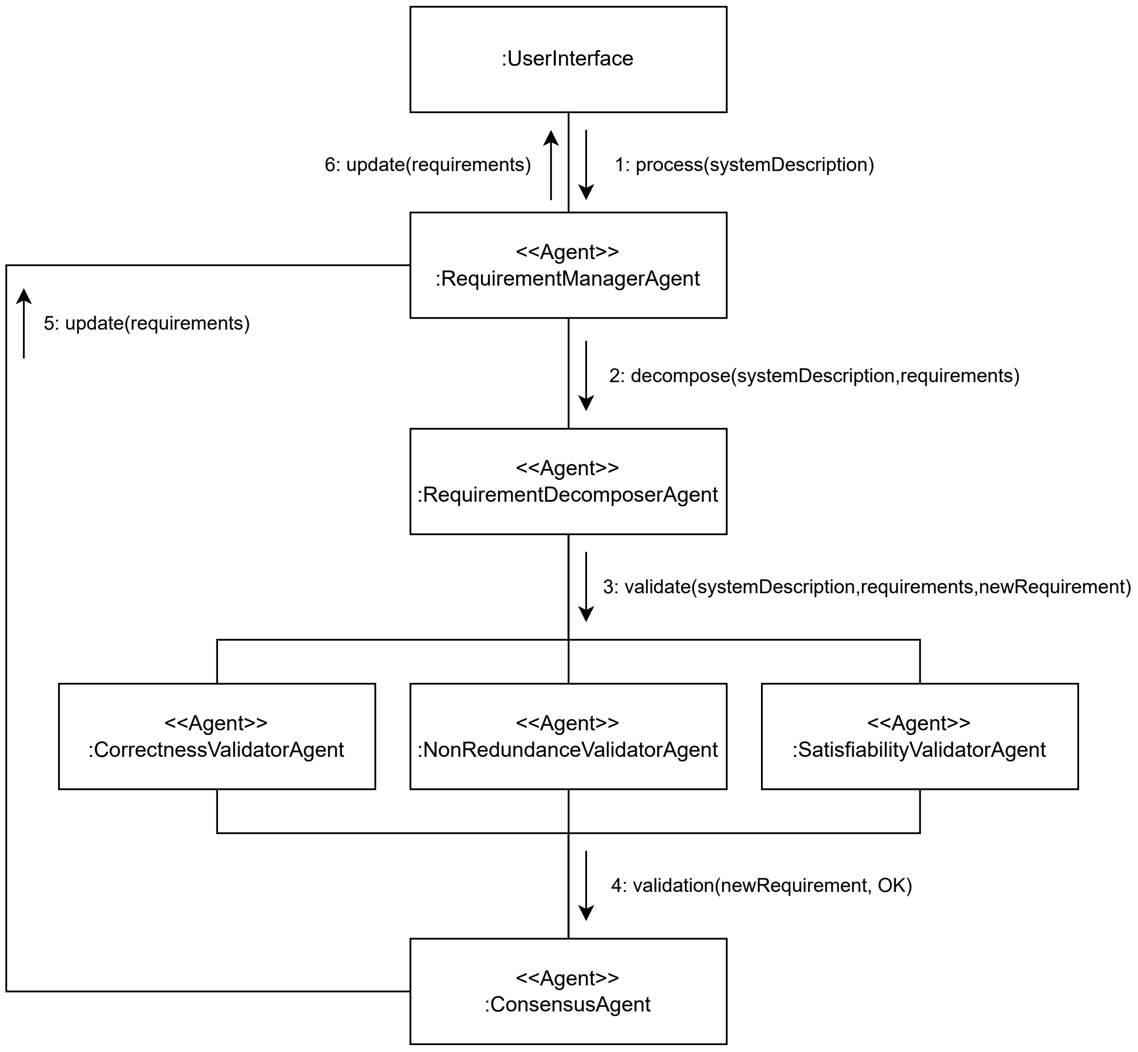

:UserInterface
6: update(requirements)
1: process(systemDescription)
<<Agent>>
:RequirementManagerAgent
5: update(requirements)
2: decompose(systemDescription,requirements)
<<Agent>>
:RequirementDecomposerAgent
3: validate(systemDescription,requirements,newRequirement)
<<Agent>>
:CorrectnessValidatorAgent
<<Agent>>
:NonRedundanceValidatorAgent
<<Agent>>
:SatisfiabilityValidatorAgent
4: validation(newRequirement, OK)
<<Agent>>
:ConsensusAgent

Figure 9: Communication diagram for the requirement manager example

Note that the role given to the LLM is passed in the system part of the prompt, built on information given in the constructor of the class (Lst. 3). The desire of the agent is also initialized in that constructor (used only for log purposes in this example).[16] This specific agent has only one fixed desire (i.e. building a suitable list of requirements), that it will try to achieve by committing, at each given time, to one intention, "PASS" or "STOP".

The implementation of the `bdi_act` (cf. Lst. 4), checks if the current intention is "PASS". If so, it sends a message to the new agent with all relevant beliefs. Otherwise, it stops, and the generated list of requirements is sent to the user interface.

The next 4 agents (`RequirementDecomposerAgent`, `CorrectnessValidatorAgent`, `NonRedundanceValidatorAgent`, `SatisfiabilityValidatorAgent`) follow a similar structure: when they receive a message, they consult an LLM and send a message to the next agent taking the response from the LLM into account. Their role is to generate a new require-

[16]The `super()` here refers to a subclass of `BDIAgent` used to share some code between `BDIAgent` agents that consult an LLM; the reader may refer to it in the source code for this experiment.

```
    async def bdi_select_intention(self, ctx):

        l = self.get_belief(req_list_tag)
        the_list = l if l != "" else "EMPTY"

        prompt = (
            f"This is the specification of the system: {self.get_belief(spec_tag)}"
            f"This is the list of atomic requirements: {the_list}"
            + self.llm_explicit_directive
        )
        llm_result = await self._model_client.create(
            messages=[
                self._system_message,
                UserMessage(content=prompt, source=self.id.key),
            ],
            cancellation_token=ctx.cancellation_token,
        )
        response = llm_result.content

        if response.startswith("COMPLETE"):
            self.add_intention("STOP", l)
        else:
            self.add_intention("PASS", l)
```

Listing 2: Implementation of `bdi_select_intention` in `RequirementManagerAgent`

```
    def __init__(self, model_client: ChatCompletionClient) -> None:
        super().__init__(
            model_client=model_client,
            description="Requirement Manager agent (with LLM).",
            llm_role="You are a requirement manager.",
            llm_job_description=(
                "Given a specification of a system, and a list of atomic requirements,
                 tell if that list of atomic requirements covers well that specification."
                + " Answer COMPLETE is the specification is well covered."
                + " Answer PARTIAL otherwise."
            ),
        )

        self.llm_explicit_directive = "Do you think the specification is well covered ?"

        self.add_desire(
            "Build a list of atomic requirements that cover the given specification."
        )
```

Listing 3: Constructor of `RequirementManagerAgent`

```
    async def bdi_act(self, ctx):
        if self.has_intention("PASS"):
            self.remove_intention("PASS")
            await self.publish_message(
                Message(
                    initial_description=self.get_belief_by_tag(spec_tag),
                    current_list=self.get_belief_by_tag(req_list_tag),
                ),
                topic_id=TopicId(cut_request_topic_type, source=self.id.key),
            )
        else:
           assert self.has_intention("STOP")
            requirements = self.get_intention("STOP")
            self.remove_intention("STOP", requirements)
            update_user(requirements)
```

Listing 4: Implementation of `bdi_act` in `RequirementManagerAgent`

```
# ConsensusAgent
async def bdi_select_intention(self, ctx):

    old_list = self.get_belief(req_list_tag)

    if (
        (self.get_belief(self.validation_correctness) is False)
        or (self.get_belief(self.validation_non_redundancy) is False)
        or (self.get_belief(self.validation_satisfiability) is False)
    ):
        self.add_intention("PASS", old_list)

    elif (
        (self.get_belief(self.validation_correctness) is None)
        or (self.get_belief(self.validation_non_redundancy) is None)
        or (self.get_belief(self.validation_satisfiability) is None)
    ):
        self.add_intention("WAIT", "-")

    else:
        assert (
            (self.get_belief(self.validation_correctness) is True)
            and (self.get_belief(self.validation_non_redundancy) is True)
            and (self.get_belief(self.validation_satisfiability) is True)
        )

        new_list = old_list + " \n * " + self.candidate
        self.add_intention("ADD", new_list)
```

Listing 5: Implementation of `bdi_select_intention` in `ConsensusAgent`

ment (`RequirementDecomposerAgent`) and to evaluate the new requirement with respect to those previously generated (`CorrectnessValidatorAgent`, `NonRedundanceValidatorAgent`, `SatisfiabilityValidatorAgent`).

The `ConsensusAgent` also reacts to individual messages coming from the 3 validator agents but, instead of acting immediately, it waits ("WAIT") for the recept of the three answers before taking a decision ("PASS" or "ADD") and sending it to the `RequirementManagerAgent` (message n° 5). This behavior is specified in its implementation of the `bdi_select_intention` method (Lst. 5).

An intention is made of two strings: one that describes the action, and one for the data associated for that action. For instance we have `self.add_intention("PASS", old_list)` in the code above. Since the WAIT action does not need any data, we use the string "-" to denote that in `self.add_intention("WAIT", "-")`. We use the "-" as a dummy parameter because it is more suitable for experimenting than an empty string or null/None.

3.3 AgentSpeak on AutoGen

As a second integration mechanism, we add AgentSpeak capabilities on top of AutoGen.[17]

The development team of Spade-bdi provides a Python AgentSpeak runtime as an autonomous package: `python-agentspeak`.[18] That runtime handles agent states and behaviors according to input AgentSpeak programs (in `.asl` files). The communication system is provided separately, in the Spade-BDI package, since it relies on the multi-agent platform (Spade here).

[17] Code available at `https://gitlab.eclipse.org/eclipse-research-labs/mosaico-project/autogen-agentspeak`

[18] `https://github.com/niklasf/python-agentspeak`

We use the Python-AgentSpeak runtime and we follow the design choices of Spade-BDI which extends Spade with AgentSpeak capabilities[19] by providing a `BDIAgent` class to support agents defined by an AgentSpeak program.

The class diagram in Figure 10 shows two classes we provide, `BDIAgent` and `BDITalker`, that are sub-classes of the `RoutedAgent` class from AutoGen. We first explain the `BDIAgent` class below.

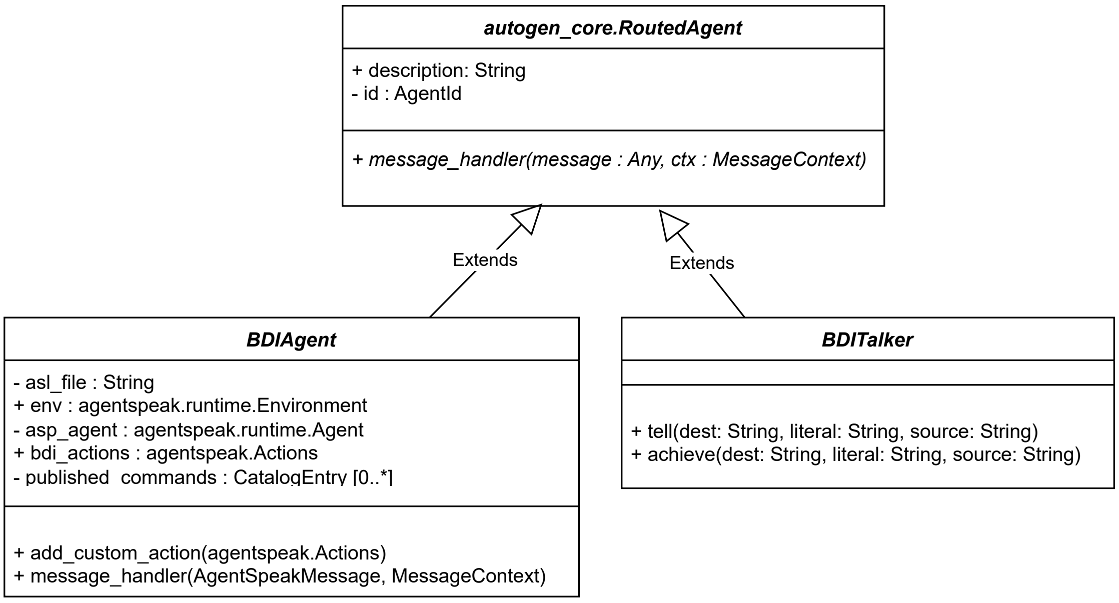

autogen_core.RoutedAgent
+ description: String
- id : AgentId
+ message_handler(message : Any, ctx : MessageContext)
Extends
Extends
BDIAgent
- asl_file : String
+ env : agentspeak.runtime.Environment
- asp_agent : agentspeak.runtime.Agent
+ bdi_actions : agentspeak.Actions
- published_commands : CatalogEntry [0..*]
+ add_custom_action(agentspeak.Actions)
+ message_handler(AgentSpeakMessage, MessageContext)
BDITalker
+ tell(dest: String, literal: String, source: String)
+ achieve(dest: String, literal: String, source: String)

Figure 10: Class diagram of AgentSpeak on AutoGen

Structure of the BDIAgent class

At run time, each `BDIAgent` object is based on an AgentSpeak program (`asl_file`) and runs its own AgentSpeak state (`asp_agent`) in its own AgentSpeak environment (`env`). The environment is not shared because two AutoGen agents can run on different machines.

Each BDIAgent object also has a list of actions (`bdi_actions`) it can perform within the `asl` program. That list is initialized with standard actions and the user can add some actions defined in Python in that list (c.f. the example from Section 3.4.1).

The set of standard actions is fully described later. For now, note that it contains the message sending primitive.

Message structure

AutoGen does not enforce the content of messages to follow any pattern. On the other hand, AgentSpeak messages are structured on an illocution (tell, achieve, etc.), a literal (representing a belief or a desire), and a sender. We define the type `AgentSpeakMessage` in Fig. 11 that will be used as the content of AutoGen messages that can be handled by AgentSpeak agents.

[19]`https://github.com/javipalanca/spade_bdi`

Note that the recipient is not part of that data structure because in AutoGen a message has a topic (or recipient) and a content, and the `AgentSpeakMessage` type takes place as the content.

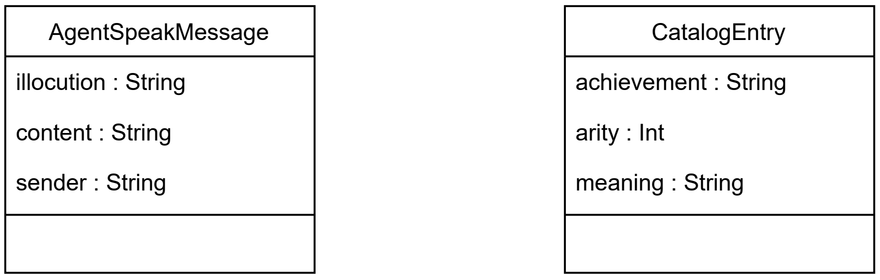


Figure 11: Additional classes

Additional services in `BDIAgent`

In addition to the basic actions provided in the Python-AgentSpeak package (such as `.print`), we add two actions for sending messages (`.send` and `.send_plan`), and two new actions to allow introspection/reflexivity (`.set_public` and `.send_catalog`).

`.send(topic, illoc, lit)`: Replace the standard `.send` action. Indeed, in our setting, each AgentSpeak agent has its own environment and runtime and the messages must be routed by the AutoGen services and not by the AgentSpeak runtime.

From the AgentSpeak point of view, the `topic` parameter is the recipient of the message, whereas from the AutoGen point of view, it is the topic of the message (routed to subscribers).

The two other parameters, the illocution and the literal are packed into an `AgentSpeakMessage` together with the identity of the sender agent and that message is routed by AutoGen.

`.send_plan(topic, illoc, plan)`: This is a variant of `.send` for the illocution `tellHow`, where the third parameter is not a literal but a string.

`.set_public(command, arity, doc)`: To address the need for dealing with a constantly evolving set of available agents, we add the concept of visibility to AgentSpeak agents. Defining an achievement (or a goal) public makes it available in a catalog that can be consulted by other agents. That achievement comes with its arity and a natural language documentation (semantics) that can be understood by a LLM.

Example (from Sec. 3.4.3): `.set_public(do_move, 0, "Move the robot.")`

The catalog of public achievements is encoded by a list of entries of type `CatalogEntry` (Fig. 11).

`.send_catalog(topic)`: Send the list of achievements declared with `.set_public`. While `.set_public` has an internal effect, `.send_catalog` is dedicated to effectively give the corresponding information on request. See the example in Sec. 3.4.3.

```
@type_subscription(topic_type="sender")
class SenderAgent(BDIAgent):

    def __init__(self, descr):
        super().__init__(descr, "sender.asl")

    @message_handler
    async def handle_message(self,
            message: AgentSpeakMessage,
            ctx: MessageContext) -> None:
        self.on_receive(message, ctx)
```

Listing 6: `Creation of a class deriving from BDIAgent`

The reaction to incoming messages is also managed in the `BDIAgent` class, by the `on_receive` method. It deals with `tell` and `untell` illocutions to modify the beliefs, `achieve` and `unachieve` to modify the goals, and `tellHow` and `untellHow` to modify the plans.[20]

The BDITalker class

The `BDIAgent` class allows MOSAICO users to program AgentSpeak agents on AutoGen. This involves representing the agent behavior as AgentSpeak rules. However, we want users also to be able to program plain AutoGen agents with behavior described in Python, and to make them interact with AgentSpeak agents.

We provide the `BDITalker` class that contains services aimed at simplifying such development. That class just adds the `tell` and `achieve` methods to the base `RoutedAgent` class.[21] Those methods are used to send `AgentSpeakMessage` messages without having to build those messages manually. Users can extend `BDITalker` and implement their behaviors in Python, relying on such services for translating messages before sending them to other MOSAICO agents. Their use can be seen in the example in Section 3.4.2.

Use of the BDIAgent class to build AgentSpeak agents

To define a new AgentSpeak agent class, the user defines a class inheriting from `BDIAgent` as in Lst. 6.

This code is constrained by the way AutoGen classes are defined by specifying their subscription (`type_subscription` annotation) and a message handler (`message_handler` annotation). In this example, the new class of agents is named `SenderAgentClass`, the message topic to be subscribed is "sender", and the AgentSpeak program is in the `sender.asl` file.

Further explanations on the use of our mechanisms are given through the examples in the next section.

3.4 Example scenarios

[20] The `askHow` illocution is not supported yet.

[21] Remark: we provide those services with methods in a subclass instead of simple function definitions because message sending in AutoGen relies on message sending capabilities of `RoutedAgent` objects.

3.4.1 Prompting an LLM for missing beliefs

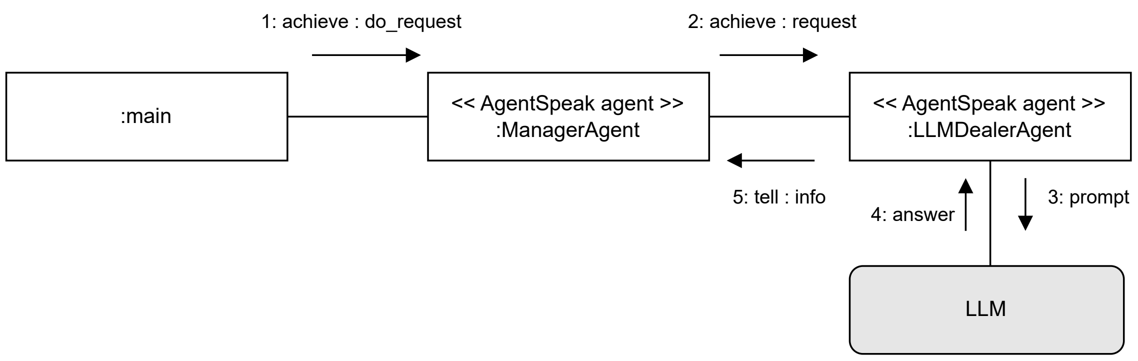


Figure 12: Communication diagram for the prompt example

In this example,[22] we add to an AgentSpeak agent the capacity to connect to an LLM to get some missing information.

For the sake of illustration, we consider a context where a Java program is being built, and the requested information is the possibility to use pattern matching in the `instanceof` construct in the version of Java used for that project.

We have two AgentSpeak agents, the first one (`ManagerAgent`) asks that information to the second one (`LLMDealerAgent`) with an *achieve* message (message n° 2 in Figure 12). The AgentSpeak program in `LLMDealerAgent` checks if the information is available in its beliefs, then it replies directly to the manager. This is described by the following rule:

```
+!request("has_pattern_matching_for_instanceof", V)[source(X)] :
    has_pattern_matching_for_instanceof(V, B) <-
        .send(X, tell,
            info("has_pattern_matching_for_instanceof", V,B)).
```

When the information is not in the beliefs, the agent triggers a prompt to an LLM (`.prompt` action, message n° 3 in the diagram), and saves the originator of the request for later (`respond_to` belief) when the LLM answer will arrive (dialogs with the LLM are asynchronous):

```
+!request("has_pattern_matching_for_instanceof", V)[source(X)] :
    not has_pattern_matching_for_instanceof(V,_) <-
        .prompt(has_pattern_matching_for_instanceof(V));
        +respond_to(X).
```

Note that in AgentSpeak language, the `not` operator describes the absence of belief.

To add the prompting capacity to the agent, a `run_prompt(...)` method is added in the `LLMDealerAgent` class at the Python level (see Figure 13) and that method is triggered at the AgentSpeak level by the `.prompt` action which is added to the list of available actions for that class of agents by the code in Fig. 7.

```
style=agentspeakstyle,basicstyle=\ttfamily\small]
def add_custom_actions(self, actions):
    super().add_custom_actions(actions)
```

[22]Source code for this example available at https://gitlab.eclipse.org/eclipse-research-labs/mosaico-project/autogen-agentspeak/-/tree/main/examples/llm_auto_completion_SE

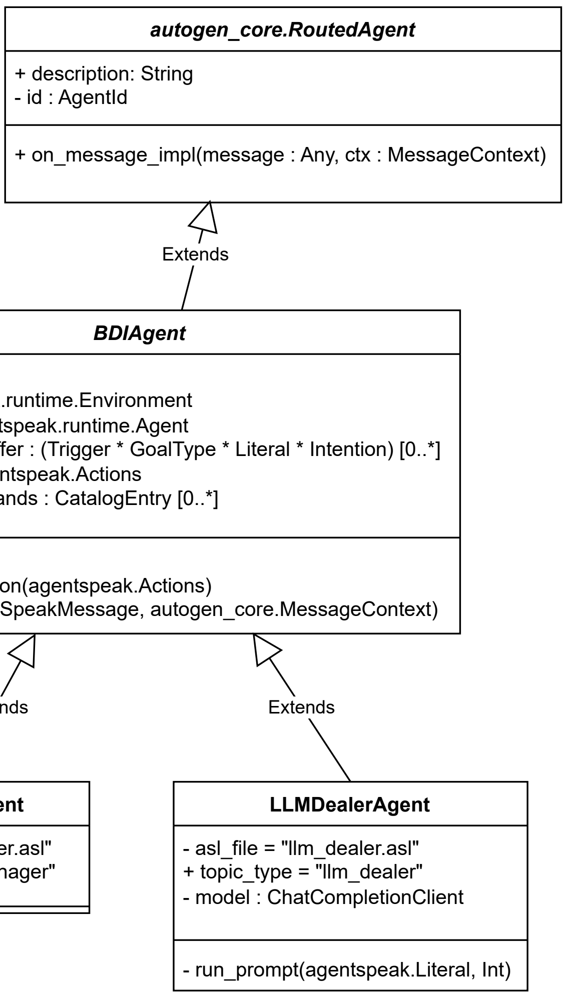

autogen_core.RoutedAgent
+ description: String
- id : AgentId
+ on_message_impl(message : Any, ctx : MessageContext)
Extends
BDIAgent
- asl_file : String
+ env : agentspeak.runtime.Environment
- asp_agent : agentspeak.runtime.Agent
+ bdi_intention_buffer : (Trigger * GoalType * Literal * Intention) [0..*]
+ bdi_actions : agentspeak.Actions
- published_commands : CatalogEntry [0..*]
+ add_custom_action(agentspeak.Actions)
- on_receive(AgentSpeakMessage, autogen_core.MessageContext)
Extends
Extends
ManagerAgent
- asl_file = "manager.asl"
+ topic_type = "manager"
LLMDealerAgent
- asl_file = "llm_dealer.asl"
+ topic_type = "llm_dealer"
- model : ChatCompletionClient
- run_prompt(agentspeak.Literal, Int)

Figure 13: Class diagram for the prompt example

```
    @actions.add_procedure(
            functor=".prompt",
            arg_specs=(
                agentspeak.Literal,
            ),
        )
    def _prompt(subject: agentspeak.Literal):
        task = asyncio.create_task(
            self.run_prompt(subject.functor, subject.args[0]))
```

Listing 7: Defining the `.prompt` action

The Python code for building the prompt in `run_prompt(...)` is simple prompt engineering based on AutoGen services for connection to LLMs. That function is tailored for the use case, so that we can simply match the received literal with a predefined list of possibilities as follows:

```
async def run_prompt(self, subject: agentspeak.Literal, v:int):
    if str(subject) == "has_pattern_matching_for_instanceof" :
        prompt = "In Java version " + str(v) + ", is there pattern \
                matching for instanceof? Answer with a single boolean \
                True or False."
        [...]
    [...]
```

In our scenario, when the answer from the LLM arrives as a new belief event (message n° 4 in the diagram), the `LLMDealerAgent` sends a tell message to the originator of the request (message n° 5).

```
+has_pattern_matching_for_instanceof(V, B) : respond_to(D) <-
  -respond_to(D) ;
  .send(D,tell, info("has_pattern_matching_for_instanceof",V, B)).
```

3.4.2 Requirement Manager (cooperation between AgentSpeak and AutoGen agents)

The following example[23] illustrates a hybrid multi-agent system, combining AutoGen and AgentSpeak agents, designed to automatically generate and refine a list of software requirements.

Design and Agent Interaction

This example consists of three specialized agents:

- **The Requirement Manager (AgentSpeak):** Serves as the central coordinator of the process. It maintains the current state of the requirements list and orchestrates communication with the other agents. The manager operates using the BDI model: it *believes* in the current specification, *desires* to achieve a complete and coherent list of requirements for this specification, and *intends* to either generate new ones or evaluate their completeness. Its AgentSpeak code is shown in Listing 8.
- **The Completeness Evaluator (AutoGen):** Evaluates wether the current list of requirements is complete. Unlike the BDI-driven manager, this agent uses conversational reasoning to assess the requirements against the specification, determining if further refinement is necessary. The agent is developed as a standard AutoGen agent, with Python behavior, extending the `BDITalker` class.
- **The Requirement Generator (AutoGen):** Interprets the task specification and the current list of requirements, then adds one requirement to the list by leveraging an LLM's capabilities to address identified gaps. The agent is developed as a standard AutoGen agent, with Python behavior, extending the `BDITalker` class.

```
+spec(S)[source(F)] <-
    +from(F).
```

[23]Source code for this example available at `https://gitlab.eclipse.org/eclipse-research-labs/mosaico-project/autogen-agentspeak/-/tree/main/examples/requirement_manager_with_llm`

```
+!build : spec(S) & not req(_) <-
    +req([]);
    !build.

+!build : spec(S) & req(L) <-
    .send(to_completeness_evaluator, tell, spec(S));
    .send(to_completeness_evaluator, tell, req(L));
    .send(to_completeness_evaluator, achieve, evaluate).

+completeness(true) : req(L) & from(F) <-
    .send(F, tell, req(L)).

+completeness(false) : spec(S) & req(L) <-
    .send(to_generator, tell, spec(S));
    .send(to_generator, tell, req(L));
    .send(to_generator, achieve, generate).

+new_req(N) : req(L) <-
    .length(L,RES);
    -new_req(N);
    -req(L);
    +req([N|L]);
    !build.
```

Listing 8: AgentSpeak code of Requirement Manager

Step-by-Step Process

A visualization of one iteration of the process is shown in Figure 14. It begins when the Requirement Manager receives the task's specification: *"A function to compare two words."* and is then tasked to build a set of requirements for it. Then comes the *Completeness Evaluator* which receives at first the list of current requirements and the specification, and then is tasked to evaluate wether the list is complete or not. In the first iteration of this process, the list is empty so the evaluator tells the manager it is not complete.

From there, the Generation phase begins. Similarly, the manager tasks the *Requirement Generator* to generate one new requirement for the specification and the current list of requirement. The generator leverages an LLM to identify the gaps and provide one.

With the added requirement to the list, the Completeness Evaluator checks once again if the list is satisfactory. If it is not, then another iteration of the Generation/Evaluation Cycle is activated. If it is, the manager returns the list to the main/user.

3.4.3 Translation from Natural Language to AgentSpeak by an LLM

In this example,[24] we illustrate how an LLM can interact with an AgentSpeak agent with little integration effort. We consider a robot driven by an AgentSpeak program (asl_robot in Figure 15), and a human that wants to send queries to that robot (User in Figure 15). When the human

[24]Source code for this example available at `https://gitlab.eclipse.org/eclipse-research-labs/mosaico-project/autogen-agentspeak/-/tree/main/examples/llm_translation`

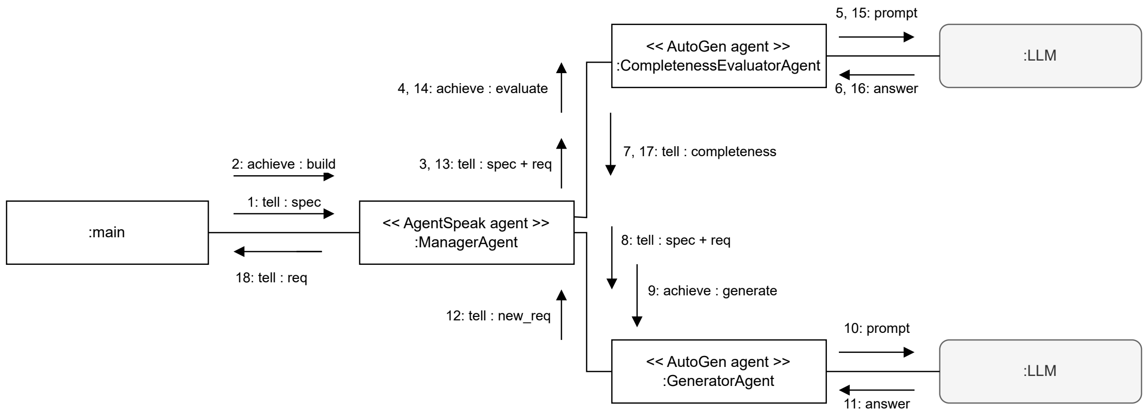


Figure 14: Communication diagram for the manager example of one iteration of the evaluation and generation cycle

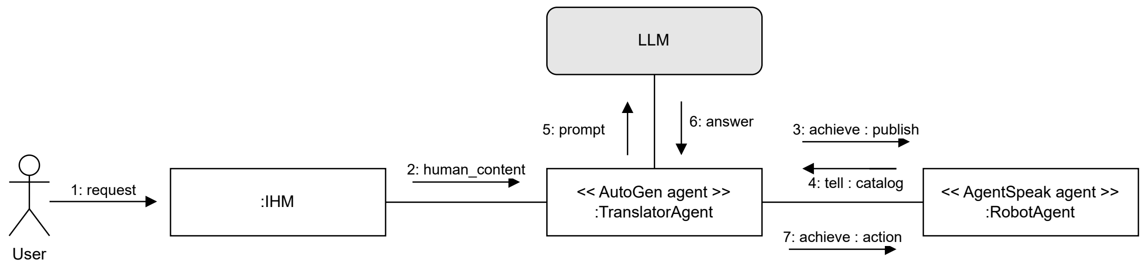


Figure 15: Communication diagram for the translation example

expresses a query (message n° 1), the corresponding sentence (a string) is sent to the translator agent (message n° 2). To be able to translate that sentence into an AgentSpeak query, the translator has to ask to the AgentSpeak robot the set of queries it can handle (message n° 3). The AgentSpeak program contains a set of `.set_public` actions triggered on startup:

```
+!start <-
    .set_public(do_move, 0, "Move the robot.") ;
    .set_public(do_jump, 0, "Make the robot jump.") ;
    .set_public(move_by, 1, "Move the robot by the distance
            given as parameter (in cm).").
```

Each `.set_public` action feeds the catalog of achievements that an agent can perform with their documentation in natural language (that the LLM can understand). Note that the `.set_public` action is part of our BDI agent class described in Section 3.3.

When the robot agent receives the publish request (message n° 3) it sends its catalog to the translator agent (message n° 4) accordingly to the following rule:

```
+!publish[source(S)] <-
    .send_catalog(S).
```

In response, the translator agent can send a prompt to the LLM containing its role (produce an achievement literal), the catalog of possible literals, and the sentence coming from the human user (message n° 5). The LLM then answers to the agent (message n° 6) which can send the relevant achieve request to the AgentSpeak robot (message n° 7).

Here is an example of prompt produced in our scenario.

```
Your task is to translate a human request into an AgentSpeak goal for a robot.
The possible AgentSpeak achievements are described in the following list.
[BEGIN LIST OF ACHIEVEMENTS]
[CatalogEntry(achievement='do_move', arity=0, meaning='Move the robot.'),
CatalogEntry(achievement='do_jump', arity=0, meaning='Make the robot jump.'),
CatalogEntry(achievement='move_by', arity=1, meaning='Move the robot by the
distance given as parameter (in cm).')]
[END OF LIST OF ACHIEVEMENTS]
Here is the sentence to translate.
[BEGIN] Please jump now.[END]
Respond with only one achievement.
For example you can answer 'do_dig' if that achievement has arity 0,
or 'do_wait(300)' if arity is 1.
```

4 Conclusion

In this deliverable, we first presented an overview of traditional autonomous agents and how large language models (LLMs) have empowered their reasoning capabilities using cutting-edge strategies, e.g., prompt engineering, RAG, and fine-tuning. However, recent research highlights the limitations of single-agent architectures, such as catastrophic forgetting or hallucinations, therefore suggesting the need for conceiving LLM-MAS. We conducted a multivocal study to analyze both academic surveys and well-founded GitHub projects related to LLM-MAS.

We then explored how the classical BDI framework can be integrated into LLM-based agents within the MOSAICO project. We highlighted the importance of reasoning about agents' beliefs, desires, and intentions—for instance, to select the most suitable agent for a task, to notify users of planned actions, or to evaluate the consistency of agents' beliefs. Differences in assessments between agents can arise from varying beliefs, desires, or intentions, illustrating the need for explicit BDI reasoning.

Finally, we presented two concrete integration mechanisms: first, extending LLM-based agents with semantic actions that implement the evaluation steps of the BDI reasoning cycle; second, embedding a domain-specific language to describe BDI behaviors, enabling behavior analysis and reuse of existing BDI logic. Both approaches are accompanied by working code that demonstrates the semantics of the integration and provides prototypes for other MOSAICO work-packages. As the first version of the MOSAICO orchestrator will be available only in M15, we used Microsoft AutoGen to implement these prototypes, showing that the integration patterns are transferable to other LLM-based agent frameworks.

In future work within the MOSAICO project, we will explore the applications of LLM-based BDI agents that we outlined here. Agents can either participate directly in the reasoning cycle — analyzing the environment, forming beliefs, selecting plans, and committing to intentions — or serve as natural language interfaces to deterministic BDI agents. These examples illustrate the flexibility and applicability of our integration approaches, laying the groundwork for their future use in the final MOSAICO implementation.

References

Achiam, Josh, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. (2023), "GPT-4 technical report", *in*: *arXiv preprint arXiv:2303.08774*. [Cited on page 11]

Agno Contributors (2025), *Agno*, `https://github.com/agno-agi/agno`, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Amatriain, Xavier (2024), *Prompt Design and Engineering: Introduction and Advanced Methods*, arXiv: `2401.14423 [cs.SE]`, URL: `https://arxiv.org/abs/2401.14423`. [Cited on page 11]

Aratchige, R. M. and W. M. K. S. Ilmini (2025), *LLMs Working in Harmony: A Survey on the Technological Aspects of Building Effective LLM-Based Multi Agent Systems*, arXiv: `2504.01963 [cs.MA]`, URL: `https://arxiv.org/abs/2504.01963`. [Cited on page 22]

Autonomous-GPT (2024), `https://github.com/Significant-Gravitas/Auto-GPT`, GitHub repository, last accessed 11-02-2025. [Cited on page 25]

Bernhardsson, Erik (2013), *Annoy: Approximate Nearest Neighbors Oh Yeah*, GitHub repository, C++ library with Python bindings for high-dimensional vector similarity search, URL: `https://github.com/spotify/annoy` (visited on 06/09/2025). [Cited on page 13]

Bordini, Rafael H., Michael Fisher, Carmen Pardavila, and Michael Wooldridge (2003), "Model checking agentspeak", *in*: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, Melbourne, Australia: Association for Computing Machinery, pp. 409–416, ISBN: 1581136838, DOI: `10.1145/860575.860641`, URL: `https://doi.org/10.1145/860575.860641`. [Cited on page 32]

Bordini, Rafael H., Michael Fisher, Willem Visser, and Michael Wooldridge (2004), "Verifiable Multi-agent Programs", *in*: *Programming Multi-Agent Systems*, ed. by Mehdi M. Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 72–89, ISBN: 978-3-540-25936-7. [Cited on page 32]

Bordini, Rafael H., Michael Fisher, Michael Wooldridge, and Willem Visser (June 2009), "Property-based Slicing for Agent Verification", *in*: *Journal of Logic and Computation* 19.6, pp. 1385–1425, ISSN: 0955-792X, DOI: `10.1093/logcom/exp029`, eprint: `https://academic.oup.com/logcom/article-pdf/19/6/1385/2811334/exp029.pdf`, URL: `https://doi.org/10.1093/logcom/exp029`. [Cited on page 32]

Borges, Hudson and Marco Tulio Valente (2018), "What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform", *in*: *Journal of Systems and Software* 146, pp. 112–129, ISSN: 0164-1212, DOI: `https://doi.org/10.1016/j.jss.2018.09.016`, URL: `https://www.sciencedirect.com/science/article/pii/S0164121218301961`. [Cited on page 18]

Bratman, Michael (1987), "Intention, plans, and practical reason", *in*: [cited on page 30]

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020), *Language Models are Few-Shot Learners*, arXiv: `2005.14165 [cs.CL]`, URL: `https://arxiv.org/abs/2005.14165`. [Cited on page 11]

Chen, Xiaojun, Ting Liu, Philippe Fournier Viger, Bowen Zhang, Guodong Long, and Qin Zhang (June 2024), "A fine-grained self-adapting prompt learning approach for few-shot learning with pre-trained language models”, *in*: *Knowledge-Based Systems* 299, p. 111968, DOI: 10.1016/j.knosys.2024.111968. [Cited on page 11]

Cheng, Yuheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He (Jan. 2024), *Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects*, arXiv:2401.03428 [cs] version: 1 Read_Status: New Read_Status_Date: 2025-02-10T21:50:41.077Z, DOI: 10.48550/arXiv.2401.03428, URL: http://arxiv.org/abs/2401.03428 (visited on 02/07/2025). [Cited on page 23]

CrewAI Team (2024), *CrewAI*, https://github.com/crewAIInc/crewAI, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Deepset.ai (2024), *Haystack*, https://github.com/deepset-ai/haystack, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Dhamani, Dhruv and Mary Lou Maher (Dec. 2023), *The Tyranny of Possibilities in the Design of Task-Oriented LLM Systems: A Scoping Survey*, arXiv:2312.17601 [cs] version: 1 Read_Status: New Read_Status_Date: 2025-02-10T21:50:40.142Z, DOI: 10.48550/arXiv.2312.17601, URL: http://arxiv.org/abs/2312.17601 (visited on 02/07/2025). [Cited on page 24]

Dify Contributors (2024), *Dify*, https://github.com/langgenius/dify, GitHub repository, last accessed 11-02-2025. [Cited on page 25]

Doan, Thu T. H., Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio (June 2023), “Too long; didn’t read: Automatic summarization of GitHub README.MD with Transformers”, *in*: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE ’23, Oulu, Finland: Association for Computing Machinery, pp. 267–272, ISBN: 9798400700446, DOI: 10.1145/3593434.3593448, URL: https://dl.acm.org/doi/10.1145/3593434.3593448 (visited on 02/01/2024). [Cited on page 11]

Du, Hung, Srikanth Thudumu, Rajesh Vasa, and Kon Mouzakis (Feb. 2024), *A Survey on Context-Aware Multi-Agent Systems: Techniques, Challenges and Future Directions*, arXiv:2402.01968 [cs] version: 1 Read_Status: New Read_Status_Date: 2025-02-10T21:50:43.239Z, DOI: 10.48550/arXiv.2402.01968, URL: http://arxiv.org/abs/2402.01968 (visited on 02/07/2025). [Cited on page 20]

Flowise Contributors (2024), *Flowise: Drag & drop UI framework for building LLM flows*, https://github.com/FlowiseAI/Flowise, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Franklin, Stan and Art Graesser (1996), “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”, *in*: *International workshop on agent theories, architectures, and languages*, ECAI ’96, Springer, Berlin, Heidelberg: Springer-Verlag, pp. 21–35, ISBN: 3540625070. [Cited on page 9]

Frering, Laurent, Gerald Steinbauer-Wagner, and Andreas Holzinger (Feb. 2025), “Integrating Belief-Desire-Intention agents with large language models for reliable human–robot interaction and explainable Artificial Intelligence”, *in*: *Eng. Appl. Artif. Intell.* 141.C, ISSN: 0952-1976, DOI: 10.1016/j.engappai.2024.109771, URL: https://doi.org/10.1016/j.engappai.2024.109771. [Cited on page 32]

Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang (2024), *Retrieval-Augmented Generation for Large Language Models: A Survey*, arXiv: 2312.10997 [cs.CL], URL: https://arxiv.org/abs/2312.10997. [Cited on page 13]

Garousi, Vahid, Michael Felderer, and Mika V. Mäntylä (Feb. 2019), “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering”, *in*: *Information and Software Technology* 106, Read_Status: New Read_Status_Date: 2025-02-

04T10:31:39.920Z, pp. 101–121, ISSN: 0950-5849, DOI: 10.1016/j.infsof.2018.09.006, URL: https://www.sciencedirect.com/science/article/pii/S0950584918301939 (visited on 01/21/2025). [Cited on pages 17, 18]

Gatti, Andrea, Viviana Mascardi, and Angelo Ferrando (2025), "ChatBDI: Think BDI, Talk LLM", *in*: *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '25, Detroit, MI, USA: International Foundation for Autonomous Agents and Multiagent Systems, pp. 2541–2543, ISBN: 9798400714269. [Cited on page 32]

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. (2023), "Gemini: a family of highly capable multimodal models", *in*: *arXiv preprint arXiv:2312.11805*. [Cited on page 11]

Georgeff, Michael, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge (1999), "The Belief-Desire-Intention Model of Agency", *in*: *Intelligent Agents V: Agents Theories, Architectures, and Languages*, ed. by Jörg P. Müller, Anand S. Rao, and Munindar P. Singh, Berlin, Heidelberg: Springer, pp. 1–10, DOI: 10.1007/3-540-49057-4_1. [Cited on page 30]

Guerra-Hernández, Alejandro, José Martín Castro-Manzano, and Amal El Fallah-Seghrouchni (Jan. 2009), "CTL AgentSpeak(L): a Specification Language for Agent Programs", *in*: *Journal of Algorithms in Cognition, Informatics and Logic* 64.1, pp. 31–40, DOI: 10.1016/j.jalgor.2009.02.003, URL: https://hal.science/hal-01169912. [Cited on page 32]

Guo, Taicheng, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang (Aug. 2024), "Large Language Model Based Multi-agents: A Survey of Progress and Challenges", en, *in*: vol. 9, Read_Status: New Read_Status_Date: 2025-02-10T21:50:52.078Z tex.ids= guoLargeLanguageModel2024a ISSN: 1045-0823, arXiv, pp. 8048–8057, DOI: 10.24963/ijcai.2024/890, arXiv: 2402.01680 [cs.CL], URL: https://www.ijcai.org/proceedings/2024/890 (visited on 02/07/2025). [Cited on pages 12, 14, 16, 20]

He, Junda, Christoph Treude, and David Lo (Jan. 2025), "LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead", *in*: *ACM Trans. Softw. Eng. Methodol.*, Just Accepted, ISSN: 1049-331X, DOI: 10.1145/3712003, arXiv: 2404.04834 [cs.SE], URL: https://doi.org/10.1145/3712003. [Cited on pages 9, 15, 16, 20]

Hu, Edward J, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2022), "LoRA: Low-Rank Adaptation of Large Language Models", *in*: *International Conference on Learning Representations*, URL: https://openreview.net/forum?id=nZeVKeeFYf9. [Cited on page 12]

Huyen, Chip (Jan. 2025), *Agents*, Blog post on Huyenchip.com, URL: https://huyenchip.com/2025/01/07/agents.html. [Cited on page 12]

Ichida, Alexandre Yukio, Felipe Meneguzzi, and Rafael C. Cardoso (2024), "BDI Agents in Natural Language Environments", *in*: *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, pp. 880–888, ISBN: 9798400704864. [Cited on page 32]

Jégou, Hervé, Matthijs Douze, and Jeff Johnson (2025), *FAISS: A library for efficient similarity search of dense vectors*, GitHub repository, Facebook AI Research, URL: https://github.com/facebookresearch/faiss (visited on 06/09/2025). [Cited on page 13]

Jin, Weiqiang, Hongyang Du, Biao Zhao, Xingwu Tian, Bohang Shi, and Guan Yang (Jan. 2025), *A Comprehensive Survey on Multi-Agent Cooperative Decision-Making: Scenarios, Approaches, Challenges and Perspectives*, en, SSRN Scholarly Paper, Read_Status: New Read_Status_Date: 2025-02-10T21:50:46.778Z tex.ids= jinComprehensiveSurveyMultiAgent2025, Rochester, NY, DOI: 10.2139/ssrn.5106265, URL: https://papers.ssrn.com/abstract=5106265 (visited on 02/07/2025). [Cited on page 23]

Karpas, Ehud, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. (2022), "MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning", *in*: *arXiv preprint arXiv:2205.00445*. [Cited on page 12]

Kirkpatrick, James, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell (2016), "Overcoming catastrophic forgetting in neural networks", *in*: *CoRR* abs/1612.00796, arXiv: 1612.00796, URL: http://arxiv.org/abs/1612.00796. [Cited on page 15]

Kitchenham, Barbara, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman (Aug. 2010), "Systematic literature reviews in software engineering – A tertiary study", en, *in*: *Information and Software Technology* 52.8, Read_Status: New Read_Status_Date: 2025-02-10T21:50:52.079Z, pp. 792–805, ISSN: 09505849, DOI: 10.1016/j.infsof.2010.03.006, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950584910000467 (visited on 02/07/2025). [Cited on pages 16, 18]

Krishnan, Naveen (2025), *AI Agents: Evolution, Architecture, and Real-World Applications*, arXiv: 2503.12687 [cs.AI], URL: https://arxiv.org/abs/2503.12687. [Cited on page 10]

LangChain Team (2024), *Langchain*, https://github.com/langchain-ai/langchain, GitHub repository, last accessed 11-02-2025. [Cited on page 25]

Letta Contributors (2024), *Letta AI*, https://github.com/letta-ai/letta, GitHub repository, last accessed 23-09-2025. [Cited on page 27]

Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela (2021), *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, arXiv: 2005.11401 [cs.CL], URL: https://arxiv.org/abs/2005.11401. [Cited on page 13]

Liu, Aixin, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. (2024), "Deepseek-v3 technical report", *in*: *arXiv preprint arXiv:2412.19437*. [Cited on page 11]

Liu, Bo, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone (2023), "Llm+ p: Empowering large language models with optimal planning proficiency", *in*: *arXiv preprint arXiv:2304.11477*. [Cited on page 14]

Liu, Hao, Carmelo Sferrazza, and Pieter Abbeel (2023), "Chain of hindsight aligns language models with feedback", *in*: *arXiv preprint arXiv:2302.02676*. [Cited on page 14]

Llama Index (2024), https://github.com/run-llama/llama_index, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Lv, Qin, William Josephson, Zhe Wang, Moses Charikar, and Kai Li (2007), "Multi-probe LSH: efficient indexing for high-dimensional similarity search", *in*: *Proceedings of the 33rd international conference on Very large data bases*, pp. 950–961. [Cited on page 13]

Malkov, Yu A and Dmitry A Yashunin (2018), "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs", *in*: *IEEE transactions on pattern analysis and machine intelligence* 42.4, pp. 824–836. [Cited on page 13]

Mastropaolo, Antonio, Simone Scalabrino, Nathan Cooper, et al. (May 2021), "Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks", en, *in*: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, Madrid, ES: IEEE, pp. 336–347, ISBN: 978-1-66540-296-5, DOI: 10.1109/ICSE43902.2021.00041, URL:

https://ieeexplore.ieee.org/document/9401982/ (visited on 01/30/2024). [Cited on page 11]

MetaGPT Team (2024), *MetaGPT*, https://github.com/geekan/MetaGPT, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Microsoft AutoGen (2024), https://github.com/microsoft/autogen, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Microsoft Semantic Kernel (2024), https://github.com/microsoft/semantic-kernel, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Mu, Chunjiang, Hao Guo, Yang Chen, Chen Shen, Die Hu, Shuyue Hu, and Zhen Wang (2024), "Multi-agent, human–agent and beyond: A survey on cooperation in social dilemmas", *in*: *Neurocomputing* 610, p. 128514, ISSN: 0925-2312, DOI: https://doi.org/10.1016/j.neucom.2024.128514, URL: https://www.sciencedirect.com/science/article/pii/S0925231224012852. [Cited on pages 21, 24]

Naveed, Humza, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian (2023), "A comprehensive overview of large language models", *in*: *arXiv preprint arXiv:2307.06435*. [Cited on page 11]

Neto, Geraldo Torres G., Wylliams B. Santos, Patricia Takako Endo, and Roberta A.A. Fagundes (Sept. 2019), "Multivocal literature reviews in software engineering: Preliminary findings from a tertiary study", *in*: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ISSN: 1949-3789 Read_Status: New Read_Status_Date: 2025-02-04T10:31:28.372Z, pp. 1–6, DOI: 10.1109/ESEM.2019.8870142, URL: https://ieeexplore.ieee.org/document/8870142 (visited on 01/21/2025). [Cited on page 16]

OpenAgents (2024), https://github.com/xlang-ai/OpenAgents, GitHub repository, last accessed 11-02-2025. [Cited on page 26]

Packer, Charles, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez (2024), *MemGPT: Towards LLMs as Operating Systems*, arXiv: 2310.08560 [cs.AI], URL: https://arxiv.org/abs/2310.08560. [Cited on page 25]

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018), *Improving Language Understanding by Generative Pre-Training*, https://openai.com/research/language-unsupervised, OpenAI Technical Report. [Cited on page 11]

Rao, Anand S (1996), "AgentSpeak (L): BDI agents speak out in a logical computable language", *in*: *European workshop on modelling autonomous agents in a multi-agent world*, Springer, pp. 42–55. [Cited on page 31]

Shen, Yongliang, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang (2023), "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face", *in*: *Advances in Neural Information Processing Systems* 36, pp. 38154–38180. [Cited on page 12]

Shinn, Noah, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao (2023), "Reflexion: Language agents with verbal reinforcement learning, 2023", *in*: *URL https://arxiv. org/abs/2303.11366*. [Cited on page 14]

Singh, Aditi, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei (2025), *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG*, arXiv: 2501.09136 [cs.AI], URL: https://arxiv.org/abs/2501.09136. [Cited on page 22]

Smolagents Team (2024), *Smolagents: Minimalist framework for building powerful agents*, https://github.com/huggingface/smolagents, GitHub repository, last accessed 11-02-2025. [Cited on page 27]

Soldani, Jacopo (Oct. 2020), "Grey Literature: A Safe Bridge Between Academy and Industry?", *in*: *SIGSOFT Softw. Eng. Notes* 44.3, pp. 11–12, ISSN: 0163-5948, DOI: 10.1145/3356773.3356776, URL: https://doi.org/10.1145/3356773.3356776. [Cited on page 20]

Sun, Chuanneng, Songjun Huang, and Dario Pompili (May 2024), *LLM-based Multi-Agent Reinforcement Learning: Current and Future Directions*, arXiv:2405.11106 [cs] version: 1 Read_Status: New Read_Status_Date: 2025-02-10T21:50:45.311Z, DOI: 10.48550/arXiv.2405.11106, URL: http://arxiv.org/abs/2405.11106 (visited on 02/07/2025). [Cited on page 23]

Sun, Chuanneng, Songjun Huang, and Dario Pompili (June 2025), "LLM-Based Multi-Agent Decision-Making: Challenges and Future Directions", *in*: *IEEE Robotics and Automation Letters* 10.6, pp. 5681–5688, ISSN: 2377-3766, DOI: 10.1109/LRA.2025.3562371, URL: https://ieeexplore.ieee.org/document/10970024/ (visited on 05/12/2025). [Cited on page 23]

Sun, Lijun, Yijun Yang, Qiqi Duan, Yuhui Shi, Chao Lyu, Yu-Cheng Chang, Chin-Teng Lin, and Yang Shen (2025), *Multi-Agent Coordination across Diverse Applications: A Survey*, arXiv: 2502.14743 [cs.MA], URL: https://arxiv.org/abs/2502.14743. [Cited on page 22]

Sun, Maojun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang (Dec. 2024), *A Survey on Large Language Model-based Agents for Statistics and Data Science*, arXiv:2412.14222 [cs] version: 1 Read_Status: New Read_Status_Date: 2025-02-10T21:50:46.777Z, DOI: 10.48550/arXiv.2412.14222, URL: http://arxiv.org/abs/2412.14222 (visited on 02/07/2025). [Cited on pages 20, 21]

Sun, Philip and Google Research (July 2020), *ScaNN: Scalable Nearest Neighbors*, GitHub repository and research announcement, Initial release and research blog post, URL: https://github.com/google-research/google-research/tree/master/scann (visited on 06/09/2025). [Cited on page 13]

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. (2023), "Llama: Open and efficient foundation language models", *in*: *arXiv preprint arXiv:2302.13971*, arXiv: 2302.13971 [cs.CL], URL: https://arxiv.org/abs/2302.13971. [Cited on page 11]

Tran, Khanh-Tung, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen (2025), *Multi-Agent Collaboration Mechanisms: A Survey of LLMs*, arXiv: 2501.06322 [cs.AI], URL: https://arxiv.org/abs/2501.06322. [Cited on page 23]

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin (2017), "Attention is All you Need", *in*: *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., arXiv: 1706.03762 [cs.CL], URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 03/31/2025). [Cited on page 11]

Wang, Lei, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. (Mar. 2024), "A survey on large language model based autonomous agents", *in*: *Frontiers of Computer Science* 18.6, p. 186345, ISSN: 2095-2236, DOI: 10.1007/s11704-024-40231-1, URL: http://dx.doi.org/10.1007/s11704-024-40231-1. [Cited on pages 12, 13]

Wang, Xuezhi, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou (2023), *Self-Consistency Improves Chain of Thought Reasoning in Language Models*, arXiv: 2203.11171 [cs.CL], URL: https://arxiv.org/abs/2203.11171. [Cited on page 14]

Weng, Lilian (June 2023), *LLM-Powered Autonomous Agents*, Lil'Log blog, Accessed via Lilian Weng's blog, URL: https://lilianweng.github.io/posts/2023-06-23-agent/. [Cited on pages 12–14]

Wohlin, Claes (May 2014), “Guidelines for snowballing in systematic literature studies and a replication in software engineering”, en, *in*: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, London England United Kingdom: ACM, pp. 1–10, ISBN: 978-1-4503-2476-2, DOI: 10.1145/2601248.2601268, URL: https://dl.acm.org/doi/10.1145/2601248.2601268 (visited on 02/13/2025). [Cited on page 18]

Wooldridge, Michael and Nicholas R Jennings (1995), “Intelligent agents: Theory and practice”, *in*: *The knowledge engineering review* 10.2, pp. 115–152. [Cited on page 9]

Wu, Yaozu, Dongyuan Li, Yankai Chen, Renhe Jiang, Henry Peng Zou, Liancheng Fang, Zhen Wang, and Philip S. Yu (2025), *Multi-Agent Autonomous Driving Systems with Large Language Models: A Survey of Recent Advances*, arXiv: 2502.16804 [cs.MA], URL: https://arxiv.org/abs/2502.16804. [Cited on page 24]

Xi, Zhiheng, 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, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui (Jan. 2025), “The rise and potential of large language model based agents: a survey”, *in*: *Science China Information Sciences* 68.2, Read_Status: New Read_Status_Date: 2025-05-12T08:57:00.797Z, p. 121101, ISSN: 1869-1919, DOI: 10.1007/s11432-024-4222-0, URL: https://doi.org/10.1007/s11432-024-4222-0. [Cited on page 22]

Yan, Bingyu, Xiaoming Zhang, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, and Chaozhuo Li (2025), *Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems*, arXiv: 2502.14321 [cs.MA], URL: https://arxiv.org/abs/2502.14321. [Cited on page 23]

Yang, Zhengyuan, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang (2023), “Mm-react: Prompting chatgpt for multimodal reasoning and action”, *in*: *arXiv preprint arXiv:2303.11381*. [Cited on page 12]

Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan (2023), “Tree of thoughts: Deliberate problem solving with large language models”, *in*: *Advances in neural information processing systems* 36, pp. 11809–11822, arXiv: 2305.10601 [cs.CL], URL: https://arxiv.org/abs/2305.10601. [Cited on page 14]

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao (2023), *ReAct: Synergizing Reasoning and Acting in Language Models*, arXiv: 2210.03629 [cs.CL], URL: https://arxiv.org/abs/2210.03629. [Cited on page 14]

Yu, Miao, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen (2025), *A Survey on Trustworthy LLM Agents: Threats and Countermeasures*, arXiv: 2503.09648 [cs.MA], URL: https://arxiv.org/abs/2503.09648. [Cited on page 24]

Zhang, Yue, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi (2023), *Siren’s Song in the AI Ocean: A Survey on Hallucination in Large Language Models*, arXiv: 2309.01219 [cs.CL], URL: https://arxiv.org/abs/2309.01219. [Cited on page 15]

Zhang, Zhuosheng, Aston Zhang, Mu Li, and Alex Smola (2022), “Automatic chain of thought prompting in large language models”, *in*: *arXiv preprint arXiv:2210.03493*. [Cited on page 14]

Zhang, Zhuosheng, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola (2024), *Multimodal Chain-of-Thought Reasoning in Language Models*, arXiv: `2302.00923 [cs.CL]`, URL: `https://arxiv.org/abs/2302.00923`. [Cited on page 14]

Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. (2023), "A survey of large language models", *in*: *arXiv preprint arXiv:2303.18223* 1.2. [Cited on page 10]

List of selected papers

[P1] J. He, C. Treude, D. Lo, Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead, ACM Trans. Softw. Eng. Methodol. (2025). DOI: doi:10.1145/3712003.

[P2] Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., and Zhang, X. (2024). Large Language Model Based Multi-agents: A Survey of Progress and Challenges. 9, 8048–8057. 10.24963/ijcai.2024/890.

[P3] Dhamani, D., and Maher, M. L. (2023). The Tyranny of Possibilities in the Design of Task-Oriented LLM Systems: A Scoping Survey (arXiv:2312.17601; Versione 1). arXiv. 10.48550/arXiv.2312.17601.

[P4] Xi, Z., Chen, W., Guo, X. et al. The rise and potential of large language model based agents: a survey. Sci. China Inf. Sci. 68, 121101 (2025). https://doi-org.univaq.idm.oclc.org/10.1007/s11432-024-4222-0

[P5] Mu, C., Guo, H., Chen, Y., Shen, C., Hu, D., Hu, S., and Wang, Z. (2024). Multi-agent, human–agent and beyond: A survey on cooperation in social dilemmas. Neurocomputing, 610, 128514. 10.1016/j.neucom.2024.128514.

[P6] Sun, C., Huang, S., and Pompili, D. (2024). LLM-based Multi-Agent Reinforcement Learning: Current and Future Directions (arXiv:2405.11106; Versione 1). arXiv. 10.48550/arXiv.2405.11106.

[P7] Cheng, Y., Zhang, C., Zhang, Z., Meng, X., Hong, S., Li, W., Wang, Z., Wang, Z., Yin, F., Zhao, J., and He, X. (2024). Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects (arXiv:2401.03428; Versione 1). arXiv. 10.48550/arXiv.2401.03428.

[P8] Sun, M., Han, R., Jiang, B., Qi, H., Sun, D., Yuan, Y., and Huang, J. (2024). A Survey on Large Language Model-based Agents for Statistics and Data Science (arXiv:2412.14222; Versione 1). arXiv. 10.48550/arXiv.2412.14222.

[P9] H. Du, S. Thudumu, R. Vasa, K. Mouzakis, A Survey on Context-Aware Multi-Agent Systems: Techniques, Challenges and Future Directions, 2024. DOI: 10.48550/arXiv.2402.01968.

[P10] Jin, W., Du, H., Zhao, B., Tian, X., Shi, B., and Yang, G. (2025). A Comprehensive Survey on Multi-Agent Cooperative Decision-Making: Scenarios, Approaches, Challenges and Perspectives (SSRN Scholarly Paper 5106265). Social Science Research Network. 10.2139/ssrn.5106265.

[P11] Yu, M., Meng, F., Zhou, X., Wang, S., Mao, J., Pang, L., Chen, T., Wang, K., Li, X., Zhang, Y., An, B., and Wen, Q. (2025). A Survey on Trustworthy LLM Agents: Threats and Countermeasures (arXiv:2503.09648). arXiv. 10.48550/arXiv.2503.09648.

[P12] Wu, Y., Li, D., Chen, Y., Jiang, R., Zou, H. P., Fang, L., Wang, Z., and Yu, P. S. (2025). Multi-Agent Autonomous Driving Systems with Large Language Models: A Survey of Recent Advances (arXiv:2502.16804; Versione 1). arXiv. 10.48550/arXiv.2502.16804.

[P13] Sun, L., Yang, Y., Duan, Q., Shi, Y., Lyu, C., Chang, Y.-C., Lin, C.-T., and Shen, Y. (2025). Multi-Agent Coordination across Diverse Applications: A Survey (arXiv:2502.14743; Versione 2). arXiv. 10.48550/arXiv.2502.14743.

[P14] Yan, B., Zhang, X., Zhang, L., Zhang, L., Zhou, Z., Miao, D., and Li, C. (2025). Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems (arXiv:2502.14321; Versione 1). arXiv. 10.48550/arXiv.2502.14321.

[P15] Singh, A., Ehtesham, A., Kumar, S., and Khoei, T. T. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG (arXiv:2501.09136; Versione 3). arXiv. 10.48550/arXiv.2501.09136.

[P16] Sun, C., Huang, S., and Pompili, D. (2025). LLM-Based Multi-Agent Decision-Making: Challenges and Future Directions. IEEE Robotics and Automation Letters, 10(6), 5681–5688. 10.1109/LRA.2025.3562371

[P17] Aratchige, R. M., and Ilmini, W. M. K. S. (2025). LLMs Working in Harmony: A Survey on the Technological Aspects of Building Effective LLM-Based Multi Agent Systems (arXiv:2504.01963). arXiv. 10.48550/arXiv.2504.01963

[P18] Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., and Nguyen, H. D. (2025). Multi-Agent Collaboration Mechanisms: A Survey of LLMs (arXiv:2501.06322). arXiv. 10.48550/arXiv.2501.06322

List of selected frameworks

[F1] **AutoGPT**. (2023). *AutoGPT*. `https://github.com/Significant-Gravitas/AutoGPT`. Last accessed 2025-05-05.

[F2] **LangChain**. (2023). *LangChain*. `https://github.com/langchain-ai/langchain`. Last accessed 2025-05-05.

[F3] **Dify**. (2023). *Dify*. `https://github.com/langgenius/dify`. Last accessed 2025-05-05.

[F4] **MetaGPT**. (2023). *MetaGPT*. `https://github.com/geekan/MetaGPT`. Last accessed 2025-05-05.

[F5] **AutoGen**. (2023). *AutoGen*. `https://github.com/microsoft/autogen`. Last accessed 2025-05-05.

[F6] **Llama Index**. (2023). *Llama Index*. `https://github.com/run-llama/llama_index`. Last accessed 2025-05-05.

[F7] **Flowise**. (2023). *Flowise*. `https://github.com/FlowiseAI/Flowise`. Last accessed 2025-05-05.

[F8] **CrewAI**. (2023). *CrewAI*. `https://github.com/joaomdmoura/crewai`. Last accessed 2025-05-05.

[F9] **Semantic Kernel**. (2023). *Semantic Kernel*. `https://github.com/microsoft/semantic-kernel`. Last accessed 2025-05-05.

[F10] **Agno**. (2025). *Agno*. `https://github.com/agno-agi/agno`. Last accessed 2025-05-05.

[F11] **Haystack**. (2019). *Haystack*. `https://github.com/deepset-ai/haystack`. Last accessed 2025-05-05.

[F12] **OpenAI Agents**. (2025). *OpenAI Agents*.[25] `https://github.com/openai/openai-agents`. Last accessed 2025-05-05.

[F13] **Smolagents**. (2024). *Smolagents*. `https://github.com/mihailo-mak/smol-agents`. Last accessed 2025-05-05.

[F14] **Letta AI**. (2023). *Letta*. `https://github.com/letta-ai/letta`.
Last accessed 2025-09-23.

[25] Formerly referred to as *SwarmAI*.