DOI: 10.15514/ISPRAS-2025-37(4)-28

Combining Logical Reasoning and LLMs Toward Creating Multi-Agent Smart Home Systems

L. Rezunik, ORCID: 0009-0000-9428-4718 <lrezunik@hse.ru >
M.A. Prozorskiy, ORCID: 0009-0006-9250-7280 <mprozorskiy@hse.ru>
D.V. Alexandrov, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>

HSE University,
11, Pokrovsky boulevard, Moscow, 109028, Russia.

Abstract. The rapid advancement of AI technologies, particularly Large Language Models (LLMs), has sparked interest in their integration into Multi-Agent Systems (MAS). This holds substantial promise for applications such as smart homes, where it can significantly enhance user experience by optimizing comfort, energy efficiency, and security. Despite the potential benefits, the implementation of MAS based on LLMs faces several challenges, including the risks of hallucinations, scalability issues, and concerns about the reliability of these systems in real-world applications. This study explores the development of MAS incorporating LLMs, with a focus on mitigating hallucinations through the integration of formal logical models for knowledge representation and decision-making, along with other machine learning methods. To demonstrate the efficacy of this approach, we conducted experiments with a plant care module within a smart home system. The results show that our approach can significantly reduce hallucinations and enhance the overall reliability of the system. Further research will focus on refining these methods to enhance adaptability and scalability to ensure system's functionality in real-world environments.

Keywords: multi-agent system; LLM-MA; first order logic; smart home.

For citation: Rezunik L., Prozorskiy M.A., Alexandrov D. V. Combining Logical Reasoning and LLMs Toward Creating Multi-Agent Smart Home Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, paet 2, 2025, pp. 219-234. DOI: 10.15514/ISPRAS-2025-37(4)-28.

Acknowledgements. This work is an output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE University).

Комбинирование логических рассуждений и LLM на пути к созданию мультиагентных систем умного дома

Л. Резуник, ORCID: 0009-0000-9428-4718 <lrezunik@hse.ru >
М.А. Прозорский, ORCID: 0009-0006-9250-7280 <mprozorskiy@hse.ru>
Д.В. Александров, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>

Национальный исследовательский университет «Высшая школа экономики», Россия, 109028, г. Москва, Покровский бул., 11, стр. 10.

Аннотация. Стремительное развитие технологий искусственного интеллекта, в частности больших языковых моделей (LLM), вызвало интерес к их интеграции в мультиагентные системы (МАС). Это открывает широкие перспективы в том числе для приложений умного дома, где они могут значительно улучшить пользовательский опыт за счет комфорта, энергоэффективности и безопасности. Несмотря на потенциальные преимущества, реализация МАС на основе LLM сталкивается с рядом проблем, включая риск возникновения галлюцинаций, проблемы масштабируемости и опасения по поводу надежности этих систем в реальных приложениях. В данном исследовании рассматривается разработка МАС, включающих LLM, с акцентом на уменьшение галлюцинаций путем интеграции формальных логических моделей для представления знаний и принятия решений, а также методов машинного обучения. Чтобы продемонстрировать эффективность этого подхода, были проведены эксперименты с симуляцией системы ухода за растениями в контексте умного дома. Результаты показали, что наш подход позволяет значительно уменьшить количество галлюцинаций и повысить общую надежность системы. Дальнейшие исследования будут направлены на доработку этих методов с целью повышения адаптивности и масштабируемости для обеспечения функциональности системы в реальных условиях.

Ключевые слова: мультиагентные системы; большие мультиагентные языковые модели LLM-MA; логика первого порядка; умный дом.

Для цитирования: Резуник Л., Прозорский М.А., Александров Д.В. Комбинирование логических рассуждений и LLM на пути к созданию мультиагентных систем умного дома. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 219–234 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–28.

Благодарности. Данная работа является результатом исследовательского проекта, реализованного в рамках Программы фундаментальных исследований НИУ ВШЭ.

1. Introduction

Multi-Agent Systems (MAS) are a well-established concept in the field of distributed artificial intelligence, developed to enable multiple autonomous agents to operate, interact, and make decisions within a shared environment. These systems are designed to tackle complex problems that are difficult or inefficient for a single agent to solve alone [1]. MAS have been applied across various domains, including smart homes, industrial automation, robotics, and environmental simulation.

With the emergence of large language models (LLMs) and ongoing advancements in artificial intelligence, a trend has emerged toward integrating AI with MAS to enhance decision-making. As a result, the concept of LLM-MA (LLM Multi-Agent) has appeared – MAS that are either entirely based on LLM agents or incorporate LLMs. The relevance of such systems is supported by various studies showcasing their application in problem solving, world simulation, knowledge acquisition, and more [2].

However, since LLM-MA systems are based on LLMs, they inherit many of the same challenges: scalability issues, limited collaboration capabilities, and difficulty in assessing accuracy and reliability. The most significant problem is hallucination, which in MAS can lead to unpredictable behavior. Therefore, although LLMs can have a transformative impact on MAS development, many problems remain unsolved or only partially addressed, offering substantial opportunities for research.

In this paper, we focus on the integration of LLMs into MAS and the development of a hardware-software system for automated plant care as part of a smart home MAS. The goals of this research are: (1) to describe a novel approach aimed at reducing the effect of LLM hallucinations in MAS, (2) to propose a MAS architecture designed for the same purpose, (3) to demonstrate how this approach can be applied in a real-world smart home scenario focused on plant care.

The remainder of the paper is organized as follows: Section 2 introduces the methods underlying our approach. Section 3 and 4 detail the developed algorithms and the approach itself, showcasing a smart home plant-watering agent, its implementation, and experimental results. The final sections discuss the related work, results, and directions for future research.

2. Key Concepts and Methods

As mentioned previously, Multi-Agent Systems (MAS) are systems composed of agents – computational units designed to operate independently and efficiently in dynamic environments [3].

We can formalize the concept of a MAS and define it as a tuple:

$$MAS = (A, E, I)$$

Here *A* represents a set of agents, *E* – the environment, in which the system operates and *I* – a set of relationships between agents of MAS.

An agent in a multi-agent system can be described in terms of its actions, perceived environmental changes, and its responses – i.e., actions performed in reaction to those changes:

$$A = (Env, Act, R)$$

- *Env* – the set of observations the agent can receive from the environment.
- *Act* – the set of possible actions.
- *R* – the reaction function $R: Env \rightarrow Act$, which determines the action the agent takes based on the environment.

Agents may also possess internal states, and their actions can depend on these states. However, agents are generally considered reactive.

When discussing LLM agents in MAS, the environment of such an agent is defined by its inputs: user prompt, system prompt (hidden prompt), embeddings, and configuration settings (e.g., temperature, max tokens). All these elements can be controlled. However, it is difficult to explicitly define the actions of an LLM agent in response to a specific environmental trigger. Therefore, to reduce the effect of hallucinations, we can only manipulate the agent's environment.

There are various machine learning methods developed to achieve this, as well as alternative approaches outside the ML domain – such as logical reasoning, which we will discuss later in this section. We will consider all relevant methods in the following subsections.

2.1 Machine Learning Methods

Many different machine learning methods have been developed to reduce hallucinations in LLMs. In this section, we will consider some of the most used approaches.

- *In-context Learning.*

  This method encompasses all prompting techniques that involve providing the LLM with examples of similar tasks directly within the prompt. It is widely used due to its cost-effectiveness and as an alternative to fine-tuning.

  There are various in-context learning techniques, such as few-shot learning, where several example input-output pairs are included in the prompt, and Chain-of-Thought (CoT) prompting [4], which encourages the LLM to describe its reasoning process and break the task into smaller, manageable steps.

- *Multi-Agent Debate.*

  This method is specifically designed to mitigate hallucinations or misconceptions through iterative discussions among multiple agents [5-6] (no fewer than two). The core idea is that the same question is posed across several iterations until the agents reach a consensus. Once agreement is achieved, the debate ends, and the final answer is accepted as the truth.

  However, this method is not always reliable. Depending on the model, one agent may be easily influenced by another, leading to biased or incorrect answers. To address this, knowledge bases are sometimes introduced into the debate. These are used by agents to support their responses, forcing the LLM to consider factual information before answering. This introduces a form of learning into the process (see Fig. 1).

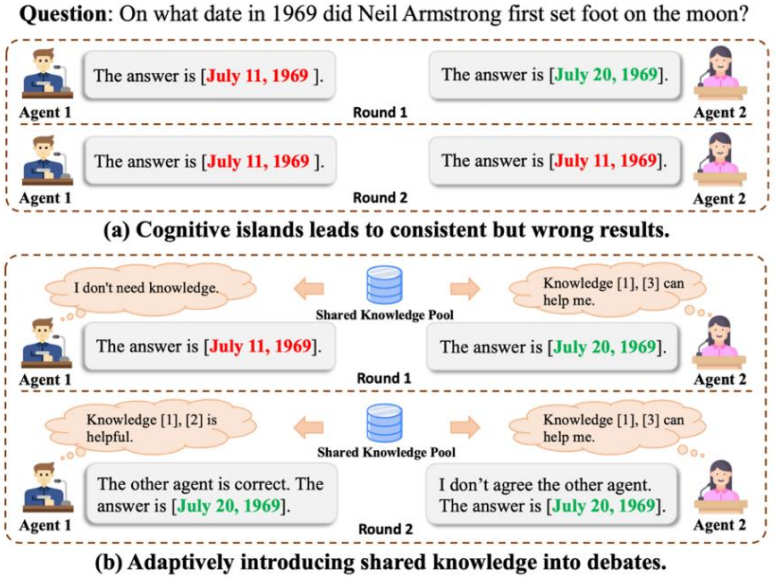

Fig. 1. Example of multi-agent debate scenario with and without the knowledge base [6].

Although the methods mentioned above help reduce hallucinations in LLM agents, the risk is not eliminated. In MAS, this issue is particularly critical, as most tasks require reasoning, and LLMs may struggle to provide coherent answers even for simple tasks [7]. For this reason, we introduce another method – an approach that has long been used prior to recent advancements in AI: logical reasoning.

2.2 Logical Reasoning

In addition to existing machine learning approaches, there is an alternative method in which LLMs are excluded from the reasoning process altogether to avoid hallucinations. Instead, reasoning is performed by matching goals with facts and rules in a structured knowledge base. This is achieved through logic-based agents, which use ontologies and apply the resolution method for logical inference. The knowledge is structured in the form of formal rules and relationships that agents can process to make decisions. This approach demonstrated improved accuracy and performance in question-answering tasks compared to popular LLMs such as GPT-4 [8].

In our current research on developing logic-based agents, we utilize Prolog. Prolog is well-suited for this task because it does not require step-by-step algorithmic instructions for reasoning; instead, it matches facts to rules using first-order logic.

Facts represent known truths. For example, in the case of a MAS for plant care, we can define that a humidity sensor is a sensor, it monitors humidity, and it is associated with a specific plant. This is demonstrated in Listing 1.

```
plant(rose_1).
sensor(humidity_sensor_1).
monitors(humidity_sensor_1, humidity).
connected_to(rose_1, humidity_sensor_1).
```

Listing 1. Examples of facts in Prolog.

Rules are logical statements that describe relationships between facts and allow the inference of new information. For example, we can define a predicate that states that a plant needs watering if the humidity level detected by the sensor connected to it is below the defined threshold (see Listing 2).

```
needs_watering(Plant) :-
   connected_to(Plant, Sensor),
   monitors(Sensor, humidity),
   current_humidity(Sensor, H),
   humidity_threshold(Plant, T),
   H < T.
```

Listing 2. Example of a rule in Prolog.

Prolog-based agents handle interactions from users or other agents through queries – essentially, questions the agent attempts to answer using its existing facts and rules. The reasoning process follows a depth-first search strategy, exploring possible solutions and backtracking when a chosen path does not lead to a valid answer. Failures occur when variable matching is unsuccessful, as Prolog depends on a unification mechanism to align query terms with the knowledge stored in its database.

3. The Proposed Approach

The core idea of our approach is that the impact of hallucinations from LLM agents on a MAS can be almost entirely eliminated by relying on logic-based agents for reasoning. In this setup, the LLM agent primarily functions as a user interface, acting as a bridge between the user and the system.

Given that the agents within the MAS are based on Prolog logic, we propose the possibility of creating an LLM agent capable of generating Prolog queries from user input and directing them to the appropriate agent in the MAS for processing (see Fig. 2).

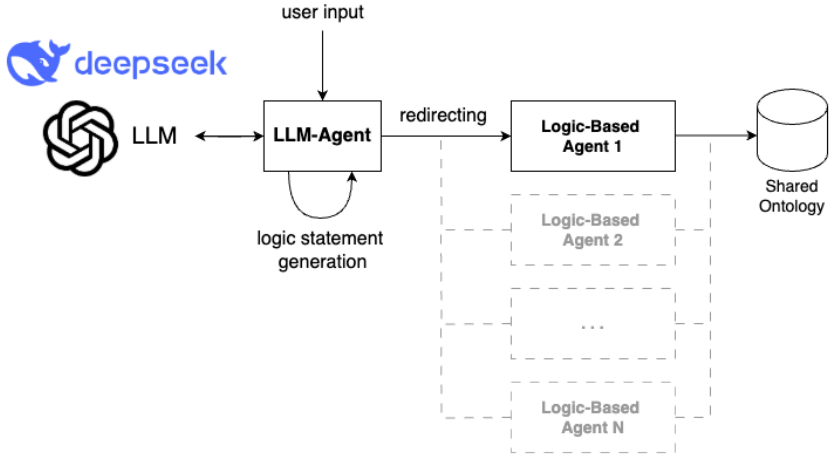

Fig. 2. Generalized representation of the proposed LLM agent workflow.

3.1 Facts and Rules Extraction

Since LLMs have demonstrated strong performance in various text-related tasks, such as semantic parsing and machine translation [9], we decided to employ an LLM-driven parsing approach to transform natural language into Prolog facts and rules. Although it is possible to use an LLM without any additional setup, we considered utilizing in-context learning and CoT to improve accuracy.

The prompt given to the LLM follows a structured format that remains consistent across all queries, to ensure consistency and effectiveness. The prompt consists of the following components:

- *Instruction* – An imperative statement that gives generalized description of the task.
- *Context* – A set of ontology facts and rules that provide the necessary knowledge for accurate translation.
- *Example* – A sample input and output demonstrating a similar task.
- *User Prompt* – An input text string from the user to MAS.

Algorithms that implement LLM-based parsing into logical statements have already been developed [10]. However, these methods share a common limitation: the user must provide the complete context for the LLM to accurately translate the query based on the available knowledge. We propose an alternative that avoids providing the entire ontology to the LLM, as doing so may degrade performance. Instead, we suggest using Retrieval-Augmented Generation (RAG) [11], particularly embeddings, which can effectively represent the ontology. This approach allows us to apply any ontology to our method, as popular ontology formats and Prolog can be converted into vector representations [12]. In future work, we plan to explore this by experimenting with different ontologies as knowledge representations for our agents.

3.2 Correction Algorithm

We need to ensure that the logical statements generated by the LLM agent are correct both in terms of meaning and syntax. To achieve this, we have developed a two-step algorithm that first verifies the meaning and then the syntax. To verify the meaning, we utilize the LLM-as-a-judge method, and implicitly previously mentioned multi-agent debate method.

The first step of the algorithm involves two different LLMs: the first is the LLM agent that generates the Prolog query, and the second is a secondary LLM that acts as a judge to determine whether the generation is correct (see Algorithm 1). We can only proceed to the next step once the secondary LLM is satisfied with the generation. If the generation is deemed unsuccessful, the judge produces a response identifying the issue. The LLM agent will then repeat the generation, using both the user prompt and the error message from the judge as input. This process continues until the result is deemed satisfactory by the judge.

It should be noted that there is a possibility of getting stuck in an infinite loop; however, we can limit the number of verification loops to avoid this.

The second step of the algorithm involves syntax checking. The LLM agent can have a built-in Prolog compiler to perform this check, or it can delegate the task to another agent. In our example, the LLM agent utilizes the compiler (see Algorithm 2) and attempts to execute the code. If the code is not executable, we must repeat step 1 of the algorithm.

3.3 Safe Reasoning

Although we suggest that the Prolog statements generated by the LLM agent are syntactically correct and use terms available in the ontology, we cannot guarantee that the LLM agent will not generate statements that are unsafe to execute. To address this, we developed a Safety-Reasoning Agent (SRA). This agent is responsible for maintaining an ontology of safety rules, all of which must be satisfied at any given time within the system. It ensures that the execution of a command or the introduction of a new fact does not violate these safety rules.

Algorithm 1 Generation and correction of Prolog statements

```
Input: P ← User prompt
       E ← Error (initially NULL)
       Ontology ← Path to ontology (vector/graph DB)
       PT ← Prompt template
Output: Status (success/error message), PrologStatement

AttemptCounter = 0
Error = NULL
while AttemptCounter != N do
    AttemptCounter++
                                   ▷ Search for relevant ontology fragments
    OntologyFragments ← RAGSearch(P, Ontology)
                                                        ▷ Prompt forming
    Prompt ← OntologyFragments.InsertInto(PT)
                                                        ▷ LLM request
    Statement ← CallLLM(Prompt, Error)
                                   ▷ Meaning Correction – request 2nd LLM
    Valid, Error ← Validate(Statement, P)
    if Error != NULL then
        Continue to the Next Iteration
    end if
    return (SuccessStatus, PrologStatement)
end while

return ErrorStatus                              ▷ Exceeded the counter limit
```

Algorithm 2 Prolog Statement Compiler Check

```
Input: S ← Statement in Prolog
       P ← Initial prompt
Output: Status (success/error message), PrologStatement

AttemptCounter = 0
Error = NULL
while AttemptCounter != N do
    AttemptCounter++
    Error ← CompilerCheck(S)
    if Error != NULL then
        S ← Algorithm_1(P, Error)
    end if
    return (SuccessStatus, PrologStatement)
end while

return ErrorStatus                              ▷ Exceeded the counter limit
```

We must ensure that commands are executed only if they are deemed safe. Therefore, before interacting with any agent, the LLM agent must consult the Safety-Reasoning Agent (see Fig. 3). The SRA's role is to check the statement generated by the LLM agent against the safe ontology, which is separate from other MAS ontologies. This is a novel approach to creating a logic-based observer agent for MAS that use LLMs, although it is rooted in earlier ideas [13-14].

To demonstrate this approach and the MAS architecture in action, we developed an example of a plant care subsystem for a smart-home system's MAS. This is illustrated in Figure 4, which shows several logical agents, an LLM agent that acts as a supervisor, and a safety-reasoning agent.

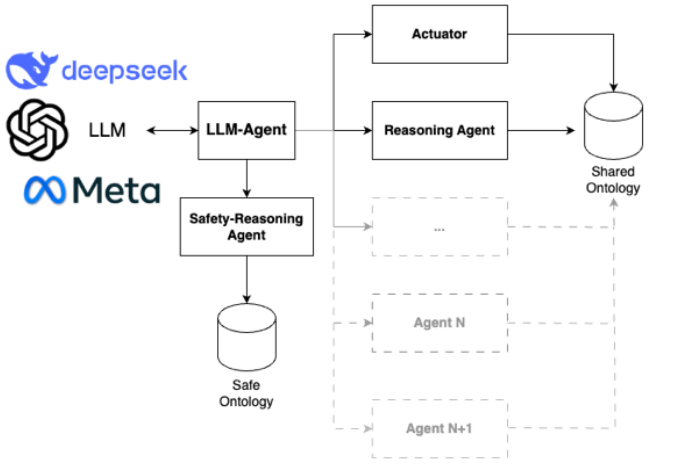

Fig. 3. MAS architecture after the introduction of the safety-reasoning agent

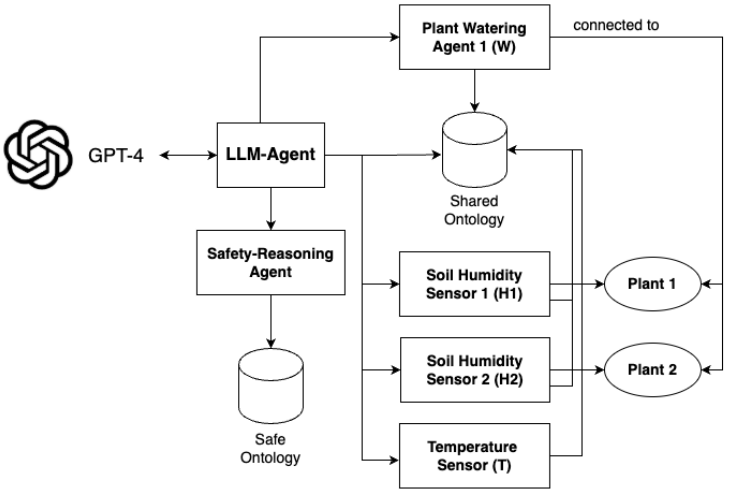

Fig. 4. Example of MAS for plant watering scenario

The process starts with the LLM, which generates commands (or queries) based on user input, and these commands are later executed by MAS agents. The algorithm for generating these commands is outlined in Section 3.2. In our approach, the LLM is supported by an LLM agent, which is tasked with implementing the algorithm. Additionally, the LLM agent functions as a communication facilitator, enabling the exchange of messages (including executable commands) with the appropriate agents.

The remainder of the system can employ actuators or reasoning agents as needed. The MAS benefits from agents built on formal logic models, enabling reasoning without the hallucinations often encountered with LLMs. Agents can either share a common ontology or work with their own individual ones. However, we suggest maintaining a shared ontology for facts while defining rules within each agent separately.

4. Experiments

We have proposed a method for integrating LLM agents with logic-based agents in MAS. The next step is to assess the accuracy of the LLM agent to determine if it introduces hallucinations. We also plan to evaluate the efficiency of the algorithm outlined in Section 3.2. Additionally, we will run the same scenarios with and without the safety-reasoning agent to observe its impact on execution speed.

In this chapter, we present a simple MAS example that incorporates an LLM agent designed according to our approach. The system demonstrates a subsystem for a smart home, specifically focusing on plant watering. A schematic representation of the system is shown in Figure 4.

4.1 Overview

The MAS incorporates several Prolog-based actuator agents that share the same ontology and update it when changes occur in the environment:

- Soil Humidity Sensor (*H*) – Responsible for monitoring and handling changes in soil humidity. There are several agents, each connected with its own plant.
- Temperature sensor (*T*) – Reacts to changes in the air temperature.

Additionally, there is a reasoning plant-watering agent (*W*), which manages turning the turning on/off of the watering. To achieve this, it utilizes decision-making: the plant should be watered only when the soil humidity is low, and the temperature exceeds the minimum required for watering.

```
should_water(Plant) :-
   soil_moisture(Plant, low),
   temperature(Plant, Temp),
   min_temp_for_watering(Plant, MinTemp),
   Temp >= MinTemp.
```

Listing 3. Plant watering rule.

We utilize OpenAI's GPT-4 which is accessed by the LLM agent. When a query is received, the LLM agent relays it to the Safety Reasoning Agent (SRA), which assesses whether the query can be safely executed. In our scenario, activating the watering system is deemed safe as long as the maximum soil humidity level has not been exceeded. If this condition is met, the LLM agent then sends the command directly to the physical watering agent.

4.2 Scenarios Testing

To evaluate the system's performance, we implemented the *SRA*, *W*, *T*, and *H* agents in Prolog, as described earlier. For the LLM agent, we created a separate Python script that generates a text embedding based on the ontology's classes, relations, and properties. The script processes user input, constructs a prompt, and sends it to the OpenAI API. The Prolog query is then extracted using the algorithms outlined in Section 3.2. This query is executed by the Prolog agents, which return either the result or an error if the query cannot be processed.

We conducted several tests, including:

1) Triggering an actuator (*H* agent, *T* agent).
2) Calling the reasoning *W* agent to control watering (on/off).
3) Retrieving knowledge from the ontology.
4) Modifying the ontology.

An example of the test output for the plant watering scenario is provided in Listing 4.

We measured execution times for all scenarios in two variations: (1) when SRA was present in the system, (2) when it was not included in the scenario to see whether it will significantly affect the execution time. The results are presented in Tables 1 and 2.

```
User Input:
        "Water the ficus, please"
Attempt 1: Generated Prolog Query:
        execute_watering(ficus).
        Generation time: 0.7765 seconds
        Validation time: 3.1058 seconds
        Query is valid, sending to Prolog server
Response from Prolog server:
        {
           'status': 'success',
           'result': 'watering_module_1
                    is watering ficus.\n
                    Moisture is now medium
                    for ficus.\n'
        }
Prolog execution time: 0.5929 seconds
Total execution time: 4.4753 seconds
```

Listing 4. Example of an output of the testing program.

Table 1. Test of Query Generation.

Scenario №	Generation Time, s	Validation Time, s
1	1.7311	9.1456
2	0.7765	3.1058
3	1.5481	7.2752
4	1.4584	0.6220

Table 2. Execution Time Test.

Scenario №	Without SRA, s	With SRA, s
1	0.5919	0.5955
2	0.5911	0.5929
3	0.5921	0.5931
4	0.5913	0.5918

We observed that the execution time difference with and without the *SRA* was minimal, the Prolog execution time overall was consistently under 0.6 seconds. Additionally, in this example the system demonstrated 100% accuracy in constructing Prolog queries, effectively mitigating hallucination issues. However, this comes at a significant time cost: while generating the initial query takes about 1 second (thanks to embeddings), the evaluation process can take over 9 seconds, which depends on the complexity and length of the query.

4.3 Simulations Comparison

After implementing the test version of the system and conducting the initial evaluation, the following questions also needed to be answered:

1. Does the proposed approach really reduce the effect of hallucinations on MAS?
2. Is it possible to block the hallucinatory actions of the LLM agent?

3. Will it be possible to compensate for the small number of LLM model parameters by using the proposed approach?

To answer each of these questions, two systems (with and without using the proposed approach) were developed to model a MAS that simulates a greenhouse, its environment and the system controlling its agents.

System #1 (see Fig. 5) does not utilize the approach developed in the study. It includes sensor agents:

- Soil Humidity Sensor – an agent responsible for monitoring changes in soil moisture. Several such agents may be present in the system, each associated with a different plant.
- Soil Temperature Sensor – an agent responsible for monitoring changes in soil temperature. Several such agents may be provided in the system, each associated with a different plant.
- Air Temperature Sensor – an agent responsible for responding to changes in air temperature.
- Air Humidity Sensor – an agent responsible for monitoring changes in air humidity.
- Carbon Dioxide (CO2) Sensor – an agent responsible for monitoring changes in the level of carbon dioxide in the air (percentage).

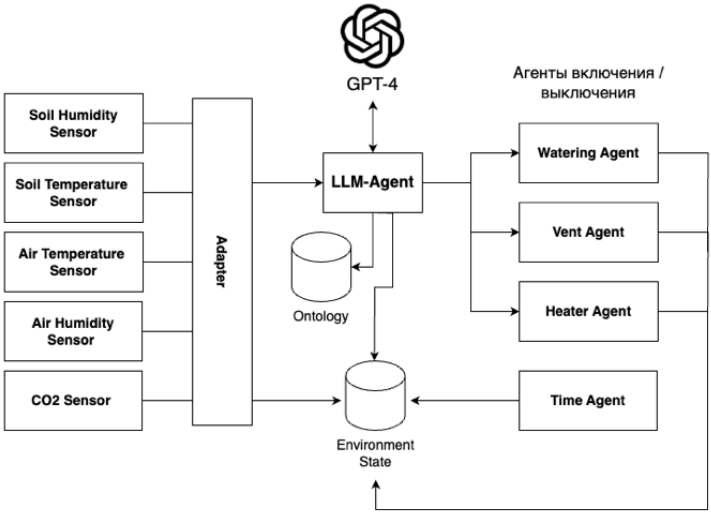

Fig. 5. Representation of a greenhouse simulation. Suggested approach is not implemented.

The state of the greenhouse environment is simulated by sensor agents, as well as a time agent that updates the time within the system, the time of day in the simulation (morning, day, evening, night). The indicators of the external environment (humidity, temperature) can be influenced by action agents:

- Heater – an agent responsible for turning the heating on and off. It simulates power consumption and updates the air temperature value.
- Watering – an agent responsible for turning watering on and off. It simulates water consumption and updates the soil humidity value.
- Vent – an agent responsible for turning ventilation on and off. Simulates electricity consumption and updates the value of carbon dioxide level in the air.

These agents switch between two states – activation and deactivation. In active mode, the agents change the value of a certain environmental parameter at regular intervals (set within the agent) until they are stopped. To control the action agents, an LLM agent is included in the system. Its task is to

decide on the activation/deactivation of the action agent depending on the state of the environment broadcast by the sensor agents. Thus, in this case, the agents of the system do not reason, but only perform actions at the request of the LLM agent.

Logging to .csv files was implemented to track changes within the system. Each record contains information necessary for analysis (see Table 3), as well as for working with process mining tools (such as ProM or pm4py). One log file record contains information about the day number within the simulation (case_id), time, environment parameters (air temperature and humidity, soil temperature, etc.), amount of consumed water and electricity, the event that occurred, and which agent caused it.

System #2 (see Fig. 6) utilizes the approach developed in the study. The MAS includes sensor agents similar to System #1, as well as an LLM agent that redirects requests from the sensor agents to the action agents. However, in this case, the LLM agent is not responsible for making the final decision as to which agent will be activated. Action agents in this case are Prolog-agents that check the state of the system regarding facts that will result in an agent not being switched without explicitly stating so in the ontology. For example, a plant watering agent that controls the on/off of watering uses the rule from Listing 3: a plant should only be watered when the soil moisture is low, and the temperature is above the minimum required for watering.

The GPT-4 model from OpenAI was used in the MAS implementation, access to which was granted to the LLM agent. It should be noted that the LLM agent was implemented similarly to the one presented in Section 4.1.

The difference between the LLM agent of System #2 and System #1 is also that when a response with a command is received from the LLM, the LLM agent passes it to the Safety Reasoning Agent (SRA), which performs a check whether the request can be safely executed. In our scenario, activation of the irrigation system is considered safe as long as the maximum soil moisture level is not exceeded. If this condition is met, the LLM agent sends the command directly to the logical irrigation agent. It is worth noting that both System #1 and System #2 use the same ontology, i.e., the agent behavior constraints are defined in the same way.

Table 3. Example of a fragment of the simulation system log file.

case_id	time	day	air_t	air_hum	co2	soil_t	soil_h	electr	water	triggered_by	event_type	timestamp
1	00:04	LATE_ NIGHT	12	42	30	18	40	0.0	0.0	air_temperature_1	SENSOR_AIR_ TEMPERATURE _stable	23.05.2025 01:19:29
1	04:46	EARLY_ MORNING	20	52	26	12	93	7000	4248.3	ACTOR_VENT	ACTOR_VENT _active	23.05.2025 01:24:11
1	04:53	EARLY_ MORNING	14	53	26	8	94	7000	4414.9	ACTOR_ WATERING	ACTOR_ WATERING _active	23.05.2025 01:24:18
2	10:41	MORNING	19	46	10	17	97	49250	32820.19	soil_humidity_1	SENSOR_SOIL_ HUMIDITY _decrease	23.05.2025 01:54:14
2	10:42	MORNING	25	46	10	21	92	49500	32820.19	ACTOR_ HEATER	ACTOR_HEATER _active	23.05.2025 01:54:15
2	10:44	MORNING	25	46	10	19	100	49500	32903.49	ACTOR_ WATERING	ACTOR_ WATERING _active	23.05.2025 01:54:17
3	00:38	LATE_ NIGHT	17	57	42	16	97	70625	43565.9	carbon_1	SENSOR_CO2 _increase	23.05.2025 02:08:16

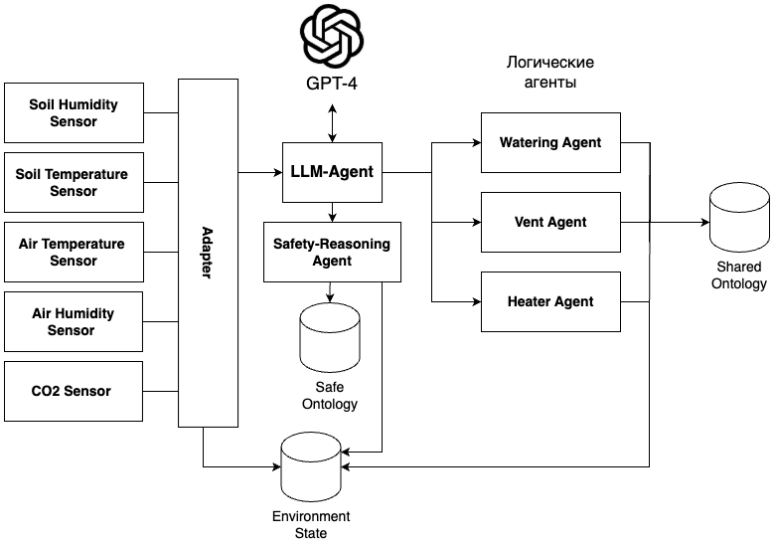

Fig. 6. Representation of a greenhouse simulation. Suggested approach is implemented.

As part of testing, the implemented MASs were run, simulating changes in the external environment and greenhouse agents. The log files were collected and analyzed. As a result, the question of reducing the effect of hallucinations on MAS can be looked at from different perspectives:

- From the point of view of occurrence of undesirable actions – in this case it is necessary to check that agents in both systems do not allow dangerous actions (such actions are marked in the ontology).
- From the point of view of energy efficiency of the whole system – in this case it is necessary to consider what impact the system has on water and electricity consumption.

As a result of analyzing the log files, it was revealed that System #1 allowed actions that can be characterized as “dangerous”. Also, a number of actions that were inefficient in terms of resource consumption were identified (for example, switching on the ventilation and irrigation agent occurred when the values were within the acceptable level). System #2 allowed a different number of “dangerous” actions in each test run, but this number was lower than for System #1. Also, during the testing of System #2 no actions of the agents that were not justified by the environmental factors were observed.

Both systems were also tested with different variations of LLMs connected to the agent LLM (variations of gpt-3 and gpt-4). Regardless of the LLM configuration, System #2 proved to be more energy efficient and safer compared to the system without logical agents.

5. Developing the Hardware

Currently, we are exploring the possibility of creating a real-life prototype of a smart home system based on our approach. To facilitate this, we have developed a watering device. The hardware subsystem is built on the Raspberry Pi Pico W board, which features the RP2040 microcontroller and provides network interaction with the server part via the built-in Wi-Fi module. Figure 7 shows a schematic diagram of the device, which includes the following components:

- Raspberry Pi Pico W microcontroller – the central component of the system, responsible for data processing, controlling connected devices, and networking.
- Servo actuator – controls the water supply process.
- Soil moisture sensor – provides data for analyzing the condition of the crop environment.

- LED – serves as a visual indicator of the device's status.
- Power supply – ensures the stable operation of the device.

The idea is for soil moisture sensor to continuously check the moisture content of the soil, sending the data to the server. When the moisture level falls below the preset threshold, the LLM agent should trigger the watering process, activating the servo actuator to release water to the plants. Afterward, the device checks the moisture level again to determine if the required watering amount has been reached.

Currently, the device allows users to remotely monitor and control the watering system, providing an interface for manual intervention if needed. In future iterations, we plan to enhance the system's capabilities by incorporating more advanced features, such as adaptive watering schedules based on weather forecasts or plant types, integrating additional sensors.

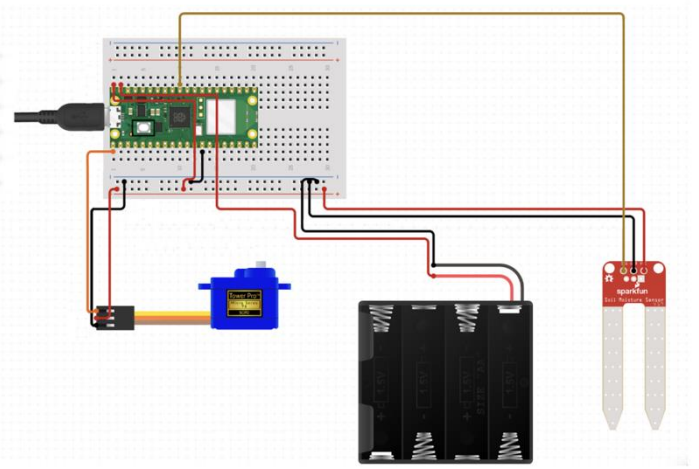

Fig. 7. The schema of plant watering device.

6. Related Works

This paper investigates the potential to fully eliminate the risk of hallucinations. To achieve this, reasoning in the agents is carried out by aligning their goals with facts and rules from a structured knowledge base. This method has proven effective and has been successfully used in the development of chatbots to enhance creativity and maintain focus on specific topics [8]. The combination of logic and ontologies has also been shown to improve accuracy and efficiency in tasks like question answering, outperforming popular LLMs [15]. While this paper focuses on Prolog ontologies and Prolog-based agents, it's important to note that the same approach can be implemented using other declarative programming languages, such as ASP (Answer Set Programming) [16].

Though the combination of LLMs and logic is a well-explored area with various applications [17-18] limited research has been conducted in the context of multi-agent systems (MAS). Our work places particular emphasis on ensuring system safety by introducing a safety-reasoning agent. A similar concept has been explored in previous studies [13-14], showing that although these works are not recent, their foundational ideas can still be adapted and applied today.

7. Future Work

In future research, we plan to explore additional MAS architectures and configurations to broaden the applicability of our approach. We will also investigate the algorithms for the dynamic enrichment of ontologies, allowing for more adaptive knowledge representation. As part of our ongoing work,

we intend to further develop the smart-home system based on the principles outlined in this paper. This will involve refining and optimizing the algorithms we've designed to improve their performance and scalability, ensuring that our solution can handle more complex scenarios and larger-scale implementations. Additionally, there is a big potential to explore new techniques such as real-time learning and cross-agent collaboration, to enhance the system's capabilities and efficiency.

8. Conclusion

In this paper, we introduced a novel approach for integrating Large Language Models (LLMs) into Multi-Agent Systems (MAS) with a focus on minimizing the risks of hallucinations and enhancing system safety. Our method leverages LLMs primarily for the transformation of natural language into logical statements, while the decision-making process is carried out by predicate logic agents. To ensure the correctness of the generated logical statements, we developed a two-step algorithm that incorporates meaning verification and syntax checking. Additionally, we introduced a Safety-Reasoning Agent (SRA), which ensures that only safe actions are executed within the system.

Our results, demonstrated in a smart home automation scenario focused on plant care. The developed two-step algorithm for generating Prolog logic statements from natural language proved successful in reducing hallucinations and ensuring the correctness of the generated code. The incorporation of the Safety-Reasoning Agent strengthened the system by preventing unsafe actions. Despite the promising results, the system's validation process remains computationally intensive, suggesting a need for further optimization in future work in terms of scalability.

We believe that this approach holds significant potential for application in other Multi-Agent Systems, particularly in the context of smart homes, where the integration of LLMs can improve user experience. The inclusion of a Safety-Reasoning Agent introduces an additional layer of reliability, which is crucial for environments where human safety and security are of the primary importance.

References

Janbi N., Katib I., Mehmood R. Distributed artificial intelligence: Taxonomy, review, framework, and reference architecture. Intelligent Systems with Applications, vol. 18, 2023, p. 200231. DOI: 10.1016/j.iswa.2023.200231.

Guo T., Chen X., Wang Y., Chang R., Pei S., Chawla N.V., Wiest O., Zhang X.N. Large language model based multi-agents: A survey of progress and challenges. arXiv preprint, 2024. DOI: 10.24963/ijcai.2024/890.

Christman J., Mele A.R. Autonomous agents: From self-control to autonomy. The Journal of Philosophy, vol. 96, no. 2, 1999, pp. 95–100. DOI: 10.2307/2564674.

Wei J., Wang X., Schuurmans D., Bosma M., Ichter B., Xia F., Chi E., Le Q.V., Zhou D. Chain-of-thought prompting elicits reasoning in large language models. arXiv preprint, vol. arXiv:2201.11903, 2023. Available at: https://arxiv.org/abs/2201.11903.

Wang H., Du X., Yu W., Chen Q., Zhu K., Chu Z., Yan L., Guan Y. Learning to break: Knowledge-enhanced reasoning in multi-agent debate system. Neurocomputing, vol. 618, 2025, p. 129063. Available at: https://www.sciencedirect.com/science/article/pii/S0925231224018344.

Jiang Y.-H., Li R., Zhou Y., Qi C., Hu H., Wei et al. AI agent for education: von Neumann multi-agent system framework. In Proc. of the 28th Global Chinese Conference on Computers in Education, 2024.

Nezhurina M., Cipolina-Kun L., Cherti M., Jitsev J. Alice in Wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models. arXiv preprint, vol. arXiv:2406.02061, 2024. Available at: https://arxiv.org/abs/2406.02061.

Zeng Y., Rajasekharan A., Basu K., Wang H., Arias J., Gupta G. A reliable common-sense reasoning socialbot built using LLMs and goal-directed ASP. Theory and Practice of Logic Programming, vol. 24, no. 4, 2024. DOI: 10.1017/S147106842400022X.

OpenAI. GPT-4 Technical Report. arXiv preprint, vol. arXiv:2303.08774, 2024. Available at: https://arxiv.org/abs/2303.08774.

Wang Z., Liu J., Bao Q., Rong H., Zhang J. ChatLogic: Integrating logic programming with large language models for multi-step reasoning. arXiv preprint, vol. arXiv:2407.10162, 2024. Available at: https://arxiv.org/abs/2407.10162.

Lewis P.S.H., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Kuttler H., Lewis M., Yih W.-T., Rocktäschel T., Riedel S., Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv preprint, vol. arXiv:2005.11401, 2020. Available at: https://arxiv.org/abs/2005.11401.

Chen J., Hu P., Jiménez-Ruiz E., Holter O.M., Antonyrajah D., Horrocks I. OWL2Vec*: Embedding of OWL ontologies. arXiv preprint, vol. arXiv:2009.14654, 2020. Available at: https://arxiv.org/abs/2009.14654.

Modgil S., Fox J. A guardian agent approach to safety in medical multi-agent systems. In Safety and Security in Multiagent Systems, 2009.

Nimis J., Lockemann P.C. Robust multi-agent systems: The transactional conversation approach? 2004.

Cabalar P., Fabiano F., Gebser M., Gupta G., Swift T. Proceedings of the 40th International Conference on Logic Programming (ICLP 2024). In Electronic Proceedings in Theoretical Computer Science (EPTCS), University of Texas at Dallas, Dallas, TX, USA, Oct. 2024, pp. 69–77.

Lifschitz V. What is Answer Set Programming? In Proceedings of the National Conference on Artificial Intelligence, vol. 3, 2008, pp. 1594–1597.

Lin X., Wu Y.-C., Yang H., Zhang Y., Zhang Y., Ji J. CLMASP: Coupling large language models with answer set programming for robotic task planning. arXiv preprint, vol. arXiv:2406.03367, 2024. Available at: https://arxiv.org/abs/2406.03367.

Yang Z., Ishay A., Lee J. Coupling large language models with logic programming for robust and general reasoning from text. arXiv preprint, vol. arXiv:2307.07696, 2023. Available at: https://arxiv.org/abs/2307.07696.

Информация об авторах / Information about authors

Людмила Александровна РЕЗУНИК – магистр программной инженерии, НИУ ВШЭ. Сфера научных интересов: разработка мобильных приложений, мультиагентные системы, большие языковые модели LLM, архитектура программного обеспечения.

Lyudmila Aleksandrovna REZUNIK – Master of Software Engineering, HSE. Research interests: mobile application development, multi-agent systems, LLM, software architecture.

Михаил Алексеевич ПРОЗОРСКИЙ – студент НИУ ВШЭ, стажер-исследователь научно-учебной лаборатории облачных и мобильных технологий НИУ ВШЭ. Сфера научных интересов: разработка мобильных приложений, системы умного дома, архитектура программного обеспечения.

Mikhail Aleekseevich PROZORSKIY – student at HSE, researcher at the Educational and Research Laboratory of Cloud and Mobile Technologies, HSE. Research interests: mobile application development, smart home systems, software architecture.

Дмитрий Владимирович АЛЕКСАНДРОВ – доктор технических наук, профессор, заведующий научно-учебной лаборатории облачных и мобильных технологий НИУ ВШЭ. Сфера научных интересов: методы и технологии искусственного интеллекта, разработка мобильных приложений, разработка программного обеспечения, инженерия знаний.

Dmitry Vladimirovich ALEXANDROV – Dr. Sci. (Tech.), Professor, Head at the Educational and Research Laboratory of Cloud and Mobile Technologies, HSE. Research interests: methods and techniques of artificial intelligence, mobile application development, software development, knowledge engineering.