An Architecture for Integrating LLM-Based Agentic Systems with Multi-Agent Simulations

Bruno Cascaes Alves[1], **Míriam Blank Born**[1], **Marilton Sanchotene de Aguiar**[1]

[1]Postgraduate Program in Computing – Federal University of Pelotas – Pelotas, Brazil

{bcalves,mbborn,marilton}@inf.ufpel.edu.br

Abstract. *This paper presents an architecture for integrating LLM-based agentic systems with multi-agent simulation platforms, addressing the limitations of traditional, strictly rule-based models. The system couples spatial simulation with context-aware decision-making through a modular, decoupled design. A dedicated API asynchronously manages communication, orchestration, and memory persistence among the simulation environment, a relational database, and the agentic framework. The system's feasibility is demonstrated in a reference implementation that combines the GAMA simulator with the Agno framework. The analysis of the results reveals that the approach significantly enhances the agents' adaptability to dynamic scenarios, although it imposes a considerably higher computational cost due to the inference processes required by the LLMs. Consequently, this study highlights a clear trade-off between computational performance and behavioral adaptability while providing a flexible and interoperable infrastructure.*

1. Introduction

Large Language Models (LLMs) are deep learning architectures trained on extensive text corpora and can comprehend and generate natural language. These models facilitate a broad spectrum of natural language processing tasks, encompassing translation, summarization, and content generation. Furthermore, LLMs can be employed in decision-making processes, functioning as components that interpret input messages, extract contextual information, and generate outputs utilized by computational systems [Ozdemir 2023].

The conventional deployment of LLMs is reactive, in which the model responds to isolated prompts that lack temporal continuity or direct environmental interaction. To address this limitation, agentic systems have emerged that integrate LLMs into a continuous decision-and-action loop. Within this context, LLM-based agentic systems encompass mechanisms such as memory, iterative planning, and the use of external tools, enabling the retention of contextual information across interactions and facilitating action adaptation based on environmental feedback [Sapkota et al. 2025].

Multi-Agent Systems (MAS) represent complex frameworks composed of multiple autonomous entities, designated as agents, which interact with one another and their environment. Within these architectures, each agent adheres to a set of behavioral rules that govern its local actions, permitting system-level patterns to emerge from such interactions. Consequently, MAS frequently model social, urban, and natural systems, wherein aggregated or centralized methodologies fail to adequately capture collective behavior [Dorri et al. 2018].

For the development and execution of multi-agent system models, the literature presents various simulation platforms, including GAMA [Taillandier et al. 2019] and NetLogo [Wilensky 1999], which provide mechanisms for defining agents, environments, and interaction rules. These platforms facilitate the implementation of spatial and temporal simulations across varying levels of complexity. However, a majority of these methodologies rely on explicit rules and predefined behaviors, which can constrain the adaptability and contextualization capabilities of simulated agents in dynamic environments.

Despite the prevalence of agent-based simulation platforms, opportunities remain to extend native mechanisms for integrating language-model-driven cognitive layers. Specifically, structured support for integrating LLM-based agentic systems into agents' decision-making cycles remains largely unavailable across simulation platforms. Consequently, this limitation can restrict the exploration of highly adaptive agent behaviors.

Within this context, the present study proposes an architecture for integrating LLM-based agentic systems into multi-agent simulations. This methodology enables simulation platforms to embed LLMs as decision-making components within agents' operational cycles, independent of any specific model or framework. Consequently, the proposed architecture aims to augment the cognitive capabilities of simulated agents through modular, decoupled integration, thereby facilitating greater flexibility and adaptability within simulation environments.

The remainder of this manuscript is organized as follows: Section 2 reviews related literature; Section 3 describes the proposed architecture and the adopted integration mechanisms; Section 4 details a reference implementation of the framework; Section 5 examines the primary architectural aspects, limitations, and trade-offs of the proposed approach; and finally, Section 6 provides the conclusions and directions for future research.

2. Related Work

This section outlines the primary research directions pertinent to the present study, structured around two main perspectives: the modeling of multi-agent systems within traditional simulation platforms and the utilization of agentic systems based on Large Language Models to support decision-making in intelligent systems.

The literature presents various applications of multi-agent systems modeled within geospatial environments. For example, research using the GAMA platform investigates social dynamics in smart cities by modeling daily routines and interactions across urban infrastructure [Olszewski et al. 2019]. Furthermore, in epidemiological and social interaction scenarios, population dynamics have been represented through autonomous agents and their interactions using NetLogo [Alves et al. 2025]. Despite these contributions, a predominant reliance on explicit rules or probabilistic models is evident, which can constrain the representation of adaptive, context-aware behaviors in dynamic scenarios subject to unforeseen events.

Recent advancements in Large Language Models (LLMs) have facilitated the emergence of agentic frameworks for the execution of autonomous reasoning. Architectures such as CrewAI [Venkadesh et al. 2024] and AutoGen [Wu et al. 2023] provide mechanisms for orchestrating LLM-based agents to conduct iterative, context-aware decision-making. These methodologies demonstrate promising results across domains

such as task automation, collaborative problem-solving, and software engineering. However, such frameworks are predominantly designed for environments centered on natural language and remain underexplored within multi-agent simulation platforms characterized by substantial spatial and temporal dependencies.

An analysis of the related literature reveals that the exploration of architectures for the structured integration between multi-agent simulation platforms and LLM-based agentic systems remains limited. Within this context, the present study proposes an architecture that integrates these components via an intermediary layer responsible for communication, context persistence, and the orchestration of agents' decision-making cycles. The proposed methodology aims to preserve independence among the simulation platform, the agentic framework, and the LLM provider.

3. Proposed Architecture

This section outlines the proposed architecture for integrating the reasoning capabilities of LLM-based agentic systems with established multi-agent simulation platforms. The objective of this architecture is to facilitate the combination of detailed spatial environment modeling with context-aware decision-making mechanisms.

3.1. Architectural Overview

Figure 1 illustrates an overview of the proposed architecture. It comprises three primary components: i) the simulation platform, responsible for representing the environment and spatial agents; ii) the integration layer, responsible for communication and data persistence; and iii) the agentic framework, responsible for contextual reasoning and LLM-based decision-making.

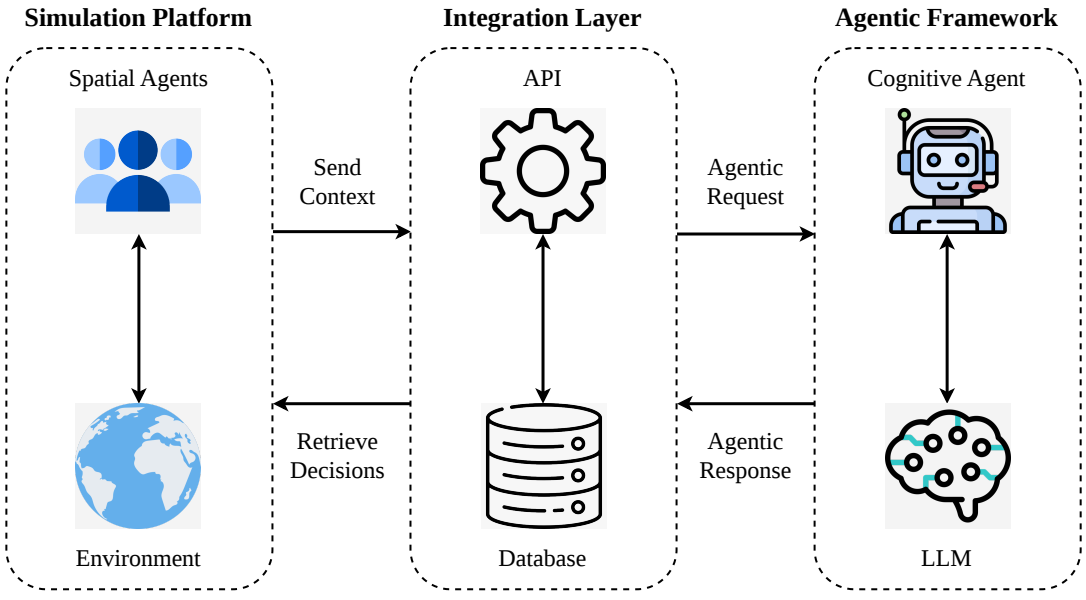


Figure 1. Overview of the architecture integrating a multi-agent simulation platform with an LLM-based agentic framework.

The simulation platform models the environment, the spatial agents, and their interactions with the environment. The integration layer serves as an intermediary, facilitating the exchange of information between the simulation platform and the agentic system.

Furthermore, this intermediary layer manages a relational database, ensuring the persistence and traceability of data generated during simulation execution. Finally, the agentic framework processes contextual information derived from the simulation and formulates decisions that are transmitted back to the simulation platform, thereby enabling dynamic updates to agent behavior.

Within this context, the subsequent subsections elaborate on the proposed integration layer between multi-agent simulation platforms and LLM-based agentic frameworks. This exposition encompasses the data persistence mechanisms, the development of the communication API, and the comprehensive architectural flow of the proposed solution.

3.2. Data Persistence

The data persistence layer stores information about the execution of multi-agent simulations, their runs, agents' states, and the decisions generated by the agentic framework. For this purpose, a relational database was modeled utilizing PostgreSQL [Stonebraker and Rowe 1990]. This approach prioritizes JSON-formatted fields to store heterogeneous attributes and dynamic contextual data, thereby ensuring flexibility across diverse simulated scenarios. Figure 2 presents the proposed persistence model, comprising four primary entities: Simulation, Round, Agent, and Decision.

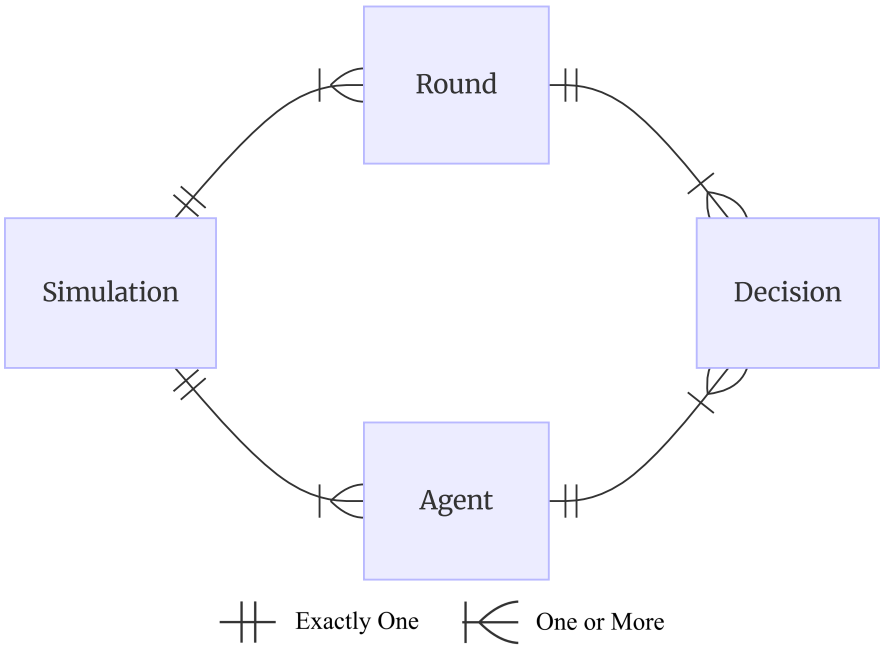


Figure 2. Relational data model for the proposed system.

The Simulation entity records general information about execution, including parameters defined within the multi-agent platform interface (such as the number of agents, seed value, and the maximum number of execution cycles), as well as global metrics such as total processing time. The Round entity delineates a period composed of a set of simulation cycles, which may correspond, for instance, to a day or a specific shift within the environment. This entity is responsible for retaining the round's metrics and contextual attributes. For example, it allows recording parameters such as round execution time, day of the week, and weather conditions during the specified period.

The Agent entity manages the static and dynamic attributes of the simulated agents, facilitating the representation of heterogeneous characteristics. In population-based simulations, for instance, attributes such as occupation, age, and gender can be recorded. Additionally, this entity retains the agent's state history across the simulated periods, enabling a comprehensive analysis of its evolution at the end of each round. Finally, the Decision entity logs the actions produced by the agentic framework and adopted by the agents within the simulation platform. This table consolidates contextual attributes, decision types, content generated by the language model, action statuses, and direct relationships with the corresponding agents and rounds.

Ultimately, the relational model ensures structural integrity, traceability, and accurate temporal associations among executions, periods, agents, and the resultant decisions. Concurrently, the use of JSON structures provides the necessary flexibility to map dynamic attributes and contextual variables derived from the simulation.

3.3. API Communication

Communication among the simulation platform, the persistence layer, and the agentic framework is orchestrated through an Application Programming Interface (API) based on the REST (Representational State Transfer) architecture [Richardson and Ruby 2007]. The API was developed in Python utilizing the FastAPI framework [Ramírez 2018], which ensures high performance and supports asynchronous operations.

The API architecture adopts a modular design encompassing routes, schemas, and services that reflect the primary application domains. The system organizes its functionality into four main operational routes:

Simulation: This module manages the simulation lifecycle, encompassing global scenario parameters, execution control, and the recording of simulation initialization and termination data within the database.

Round: This component governs the temporal progression of the simulation. It records contextual information and round-specific parameters and stores the execution duration for each simulation step.

Agent: This module oversees the instantiation of agents modeled within the multi-agent platform and continuously updates their states. At the conclusion of each period, the platform transmits updated agent data, thereby facilitating subsequent analyses of their evolution throughout the simulation.

Decision: This component orchestrates the intelligence architecture. During each execution cycle, the interface retrieves pertinent contextual information for every agent in the current round from the database. Subsequently, the system transmits this data to the integrated agentic framework, which then invokes Large Language Models (LLMs) to infer and formulate the agents' subsequent decisions. The set of possible actions is predefined by the simulation platform.

The architecture was implemented utilizing a concurrent execution model. The persistence layer employs SQLAlchemy [Bayer 2024] with asynchronous drivers to prevent API bottlenecks, while the system parallelizes LLM invocations for agents via asyncio. This methodology facilitates simultaneous decision-making and minimizes the duration required for simulation rounds. Furthermore, the API layer validates all transmitted JSON data, safeguarding attribute integrity and persistence.

3.4. Architectural Flow

Figure 3 presents the proposed sequence diagram for the dynamic use of the architecture, illustrating the chronological interaction flow among the system components. We organized the system orchestration into three fundamental phases, including initialization, cognitive processing, and state consolidation.

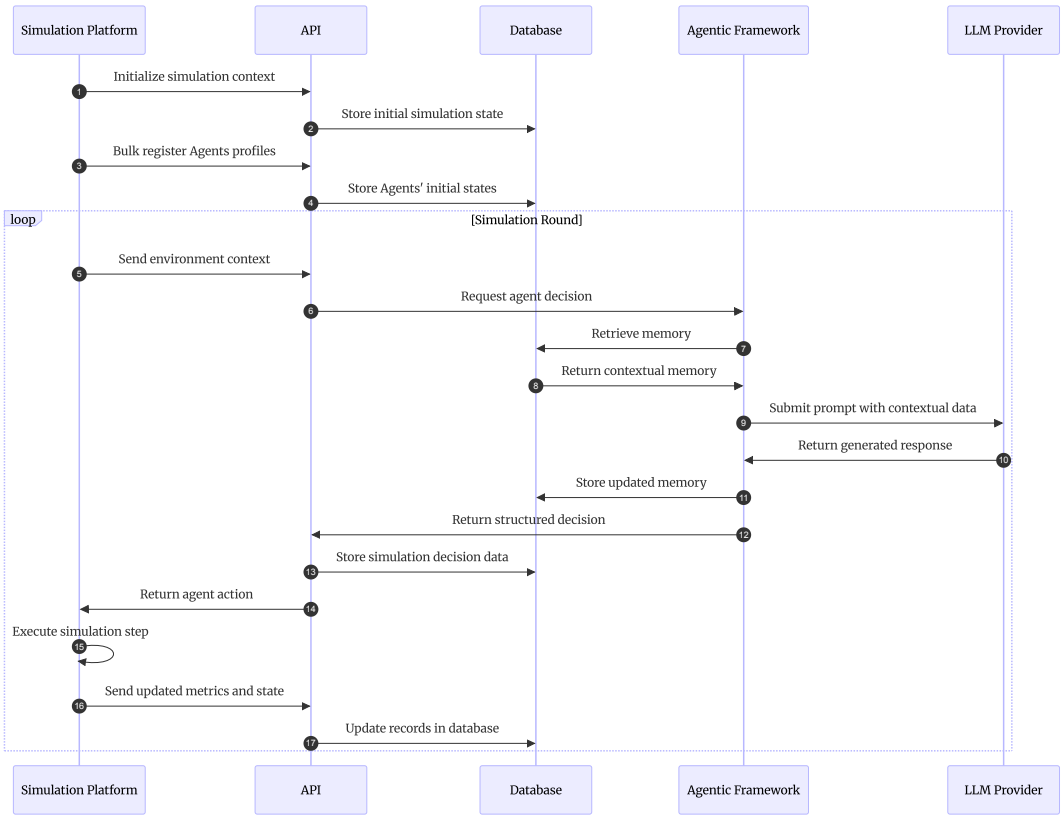


Figure 3. Sequence diagram detailing the interaction flow among the simulation platform, the integration layer, and the agentic framework with the LLM provider.

The process begins with the simulation platform, which serves as a primary component of the architecture. The platform transmits the global scenario context and submits the profiles of the spatial agents in batches. The API receives these requests and stores the system's initial data in the database, thereby preparing the infrastructure for continuous processing.

The primary execution proceeds iteratively during each simulation round. Throughout this cycle, the simulation platform transmits updated spatial perceptions and environmental context to the API, which subsequently requests the agents' actions or decisions. The API delegates processing to the agentic framework module, which operates autonomously as the system's cognitive engine. At this juncture, the agentic framework integrates spatial information derived from the simulation environment with the LLM's reasoning capabilities. The framework retrieves the agent's contextual memory and past interactions from the database, then combines them with the agent's physical perceptions obtained from the simulation.

Utilizing both agent-specific and environmental data, the system constructs an enriched prompt and submits it to the LLM provider. Consequently, the language model serves as the agent's cognitive core, evaluating environmental variables and formulating decisions guided by its profile and historical context. Following the inference process, the framework updates its internal memory with the new interaction and returns the structured decision to the API. As part of the traceability process, the API persists the comprehensive decision data in the database before returning it to the simulation platform.

Finally, upon resolving the cognitive actions, the simulation platform physically executes the simulation step, encompassing spatial movement, route modification, or status updates. To conclude the cycle and maintain architectural synchronization, the platform transmits the agent's performance metrics and the final state for the current round. The API consolidates this information within the database, ensuring that the subsequent cycle commences with an updated temporal and spatial context.

4. Reference Implementation

This section presents a reference implementation of the proposed architecture to illustrate the integration process among the system components. To facilitate the utilization and future extensions of the architecture, the developed API and reference integration code are provided in a GitHub repository[1]. As a reference implementation, GAMA [Taillandier et al. 2019] was adopted as the spatial multi-agent simulation platform, alongside Agno [Agno Team 2024] as the LLM-based agentic framework. The developed integration layer follows a modular design, enabling its adaptation to diverse simulation platforms, agentic frameworks, and language model providers.

4.1. Multi-agent Simulation Platform

The GAMA platform provides a comprehensive environment for agent-based simulation with native support for spatial and geographic modeling. It offers robust mechanisms for representing dynamic environments across diverse domains, such as urban mobility, epidemiology, and social dynamics. Despite its flexibility, GAMA lacks native, LLM-based cognitive mechanisms to support agent decision-making. Consequently, behavioral logic is typically implemented through predefined, explicit rules that can constrain the representation of highly adaptive, context-aware, and memory-oriented behaviors.

To address this limitation, the proposed architecture introduces an external cognitive layer that integrates LLM-based reasoning into the simulation workflow. Under this paradigm, the platform retains responsibility for spatial execution, environmental management, and the temporal evolution of agents, while the external LLM-based framework formulates the cognitive decisions. Thus, the architecture facilitates the incorporation of sophisticated, context-aware decision-making mechanisms without requiring intrusive modifications to the simulation platform's core structure.

Listing 1 presents a simplified code snippet illustrating the communication between the GAMA platform and the agentic framework via the proposed API. Using the GAML language, the platform establishes an API connection via the GAMA "network skill" by creating an agent (species) for external communication. During initialization, GAMA defines connection parameters, such as server address, protocol, and API.

[1] `https://github.com/brunocascaes/agentic-mas-api`

At each simulation cycle, the agents transmit HTTP requests to the API, including the simulation identifier, round identifier, agent identifier, and environmental perceptions. Upon receiving the request, the API forwards the contextual information to the agentic framework, which then invokes the LLM to infer the agent's subsequent action. The generated decision is then returned to the simulation platform, which updates the agent's behavior based on the inferred cognitive response. This workflow facilitates the incorporation of contextual and adaptive capabilities into spatial multi-agent simulations while preserving the platform's original execution structure.

```
global {
    init {
        create networking number: 1 {
            do connect to: api_host
            protocol: api_protocol port: api_port raw: false;
        }
    }
}

species networking skills: [network] {
    reflex request_agent_decision {
        do send to: "/decisions/" contents: [
            "POST",
            to_json(
                map(
                    "simulation_id" :: simulation_id,
                    "round_id" :: round_id,
                    "agent_id" :: agent_id,
                    "context" :: context,
                    "config" :: config,
                    "framework" :: framework
                )
            ),
            ["Content-Type" :: "application/json"]
        ];
        map<string, unknown> decision <- from_json(
            fetch_message()
        );
    }
}
```

Listing 1. Simplified GAMA code snippet responsible for establishing communication with the integration API, sending contextual agent data, and receiving the decision generated by the LLM-based framework.

4.2. LLM-based Agentic Framework

The Agno framework serves as the cognitive module of the proposed architecture, orchestrating LLM-based applications while supporting asynchronous execution and persistent memory. These features facilitate the integration of contextual reasoning into the simulation workflow, bridging the multi-agent platform and the selected language model.

A central advantage of this framework is its autonomous management of memory persistence when the `enable_agentic_memory` option is activated. Under this configuration, the framework stores interactions in a dedicated structure within the relational database, typically using an `agno_memories` table generated and managed by

the framework itself. This mechanism preserves historical interactions, enabling agents to incorporate behavioral history into their decision-making.

Within the proposed architecture, the agentic framework receives the simulation context, automatically retrieves the agent's historical record, and uses this information to construct the prompt forwarded to the LLM. Consequently, the decision-making process depends not only on the current state of the environment but also on previously recorded interactions, thereby enriching the contextual information leveraged during inference.

Listing 2 illustrates the instantiation of an agent and its corresponding execution function. During configuration, the system initializes the agent with a language model, such as Google's "gemini-2.5-flash-lite", a PostgreSQL-based persistence backend, and operational instructions. The framework natively supports persistent memory. Subsequently, the system executes asynchronous inference and validates the LLM's response, ensuring the final output is a structured JSON object compatible with the integration layer.

```
def create_agent(llm_model: str, db_url: str, enable_memory: bool) ->
    Agent:
     return Agent(
         model=Gemini(llm_model),
         db=PostgresDb(db_url=db_url),
         description="Decision-making agent for multi-agent simulation",
         instructions="Strict JSON output, context-only reasoning",
         enable_agentic_memory=enable_memory
     )

async def request_decision(agent: Agent, input_prompt: str, user_id:
    str) -> dict:
     response = await agent.arun(input_prompt, user_id=user_id)
     decision = json.loads(clean_json(response.content))
     return decision
```

Listing 2. Simplified Python implementation of asynchronous LLM-based agent inference with persistent memory support in Agno.

In addition to the structural configuration, the architecture employs system prompts to govern the LLM's behavior by defining output constraints, inference rules, and the anticipated decision format. Listing 3 presents an example of the prompt utilized to direct model inference within the proposed architecture. These prompts function as a behavioral control mechanism, enhancing consistency in response generation and ensuring adherence to the structured format mandated by the integration layer.

5. Discussion

This section examines the principal aspects observed during the architecture's development and the initial experiments. It is noteworthy that a component of the architecture adopted in this study was previously explored in a separate investigation [Alves et al. 2026], which focused on urban mobility modeling using a Multi-Agent System and LLMs as cognitive mechanisms to support agents' decision-making processes. In contrast, this article focuses specifically on the architectural mechanisms required to integrate multi-agent simulation environments with agentic systems.

The experimental scenario, detailed in [Alves et al. 2026], modeled urban mobility mode-choice decisions for a synthetic agent population under varying weather and

time-of-day conditions, using GAMA as the simulation platform, Agno as the agentic framework, and Gemini as the underlying LLM. Reported metrics include total execution time and variation in transportation mode choices across scenarios.

```
You are an autonomous decision-making agent operating within a multi-
   agent simulation. Your task is to select the most appropriate
   action based only on the provided context.

Input:                          Output:
- Simulation state              {
- Agent context                     "action": "<selected_action>",
- Task type                         "reason": "<brief justification>"
- Available actions             }

Rules:
- Do not assume external information
- Use only the provided context
```

Listing 3. Example of prompt used to guide the contextual decision-making process of LLM-based agents in the proposed multi-agent architecture.

Generally, observations from the experimental scenario presented in [Alves et al. 2026] indicate that incorporating LLM-based cognitive agents increases the simulation's computational cost, primarily due to the additional inference and communication stages required by the agentic framework. In the evaluated scenario, an approximate 230% increase in the simulation's total execution time was observed. Conversely, the agents exhibited an enhanced capacity to adapt their mobility decisions to the simulation context, particularly in response to fluctuations in environmental and temporal variables, such as weather conditions and time of day. Consequently, an approximate 155% increase in the variation in transportation mode choices between favorable and unfavorable scenarios was observed compared with the exclusively rule-based model.

Within this context, integrating an LLM-based agentic system with a traditional multi-agent simulation platform broadens the potential for developing highly adaptive, context-aware agents. Nevertheless, this integration increases the simulation's architectural complexity, particularly due to the need for continuous communication with external inference services and supplementary mechanisms for persistence, validation, and context management. Another critical aspect pertains to the trade-off between cognitive capabilities and computational performance. The incorporation of LLM-based inference can substantially increase both operational costs and simulation execution time, especially in scenarios involving a large number of agents or frequent decision-making cycles. Consequently, strategies such as parallelization, asynchronous request management, and caching are essential to mitigate architectural bottlenecks.

The necessity for structural validation of language model responses underscores the importance of intermediate control mechanisms between the simulation platform and the agentic framework. In this context, implementing a decoupled integration layer affords greater control over persistence, traceability, and error management, enhancing the robustness and extensibility of the proposed solution. Furthermore, the architecture promotes interoperability among diverse simulation platforms and agentic frameworks. Because components are decoupled via API-based communication and standardized JSON

structures, substituting language models, agentic frameworks, or multi-agent platforms can be done without substantial modifications to the system's core structure.

6. Conclusion

This study presents an architecture for integrating LLM-based agentic systems with multi-agent simulation platforms. The proposed methodology prioritizes modularity and architectural decoupling, enabling diverse simulation platforms, agentic frameworks, and language models to function as interchangeable components within agents' decision-making cycles. Within this context, the architecture incorporates mechanisms for REST-based API communication, relational persistence, contextual memory management, and asynchronous orchestration between simulation environments and external inference services.

As a reference implementation, the architecture was integrated with the GAMA multi-agent simulation platform and the Agno agentic framework, demonstrating the feasibility of incorporating contextual reasoning capabilities into spatial simulations without requiring modifications to the MAS's internal structure. Furthermore, the implementation demonstrates how the architecture dynamically combines contextual information derived from the simulation environment with persistent memory and LLM-based inference to facilitate adaptive decision-making.

Overall, the proposed architecture enhances the integration between traditional multi-agent simulation environments and emerging LLM-based cognitive systems. Discussions throughout the manuscript indicate that this integration broadens the potential for representing highly adaptive, context-aware agent behaviors while preserving flexibility regarding the selected simulation platform, agentic framework, and language model provider. Nevertheless, the study also highlights significant challenges associated with this integration paradigm, particularly in terms of computational cost, architectural complexity, and communication overhead introduced by external inference services. In this context, asynchronous execution and scalable orchestration mechanisms are essential for enabling large-scale simulations involving cognitive agents.

Future work will investigate advanced persistent memory mechanisms, coordination strategies among multiple cognitive agents, and hybrid decision-making approaches that combine explicit rules with LLM-based reasoning. Additionally, subsequent research will explore the integration of local-language models, distributed inference architectures, and performance evaluation within large-scale simulation scenarios involving numerous concurrent cognitive agents.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This work was supported by the Kunumi Institute. The authors thank the institution for its financial support and commitment to advancing scientific research.

Generative Artificial Intelligence tools, including Gemini and Grammarly, were employed to refine the linguistic quality during the preparation of this manuscript. The authors did not fabricate data, manipulate experimental results, generate fictitious references, or misrepresent the scientific interpretation. All content was reviewed and verified by the authors, who assume full responsibility for the final version of the manuscript.

References

Agno Team (2024). Agno. Available at: https://docs.agno.com/. Accessed: 2026-05-26.

Alves, B., Caldas, L., Junior, U. F., Santos, A., Grossmann, R., Born, M., and Aguiar, M. (2025). Sistema multiagente para simulação de propagação viral no contexto da covid-19 utilizando a ferramenta netlogo. In *Anais do XIX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 324–331, Porto Alegre, RS, Brasil. SBC.

Alves, B. C., Born, M. B., and Aguiar, M. S. d. (2026). Integrating large language models as cognitive agents into the gama platform for urban mobility simulation. *The International FLAIRS Conference Proceedings*, 39(1).

Bayer, M. (2024). Sqlalchemy. `https://www.sqlalchemy.org/`. Accessed: 2026-05-26.

Dorri, A., Kanhere, S. S., and Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*.

Olszewski, R., Pałka, P., Turek, A., Kietlińska, B., Płatkowski, T., and Borkowski, M. (2019). Spatiotemporal modeling of the smart city residents' activity with multi-agent systems. *Applied Sciences*, 9(10).

Ozdemir, S. (2023). *Quick Start Guide to Large Language Models: Strategies and Best Practices for Using ChatGPT and Other LLMs*. Addison-Wesley, Boston, MA, USA.

Ramírez, S. (2018). FastAPI. Available at: https://fastapi.tiangolo.com/. Accessed: 2026-05-26.

Richardson, L. and Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media, Sebastopol, CA, 1 edition.

Sapkota, R., Roumeliotis, K. I., and Karkee, M. (2025). Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *Information Fusion*, 126.

Stonebraker, M. and Rowe, L. A. (1990). The postgres next-generation database management system. *Communications of the ACM*, 33(10):78–92.

Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.-N., Marilleau, N., Caillou, P., Philippon, D., and Drogoul, A. (2019). Building, composing and experimenting complex spatial models with the GAMA platform. *Geoinformatica*, 23(2):299–322.

Venkadesh, P., Divya, S. V., and Kumar, K. S. (2024). Unlocking AI Creativity: A multi-agent approach with CrewAI. *Journal of Trends in Computer Science and Smart Technology*, 6(4):338–356.

Wilensky, U. (1999). *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Li, C., Zhang, L., Rowe, M., Jiang, Z., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.