A Conceptual Framework for a Hybrid ML-Agents LLM Architecture for Adaptive Serious Game Scenarios

Jakub Zagora, Peter Sykora, Roberta Hlavata, Miroslav Benco

Department of Multimedia and Information-Communication Technologies, FEIT, University of Zilina, Slovakia
jakub.zagora@feit.uniza.sk

Abstract. Serious games tend to be used for training, scenario-based exercises and simulation learning, but their scripted nature limits their adaptability. Existing research explores reinforcement learning agents in simulation environments, LLM-based game agents, and LLM-powered virtual entities for serious games. However, there is still a lack of modular conceptual architecture that integrates ML-Agents as a behavioral execution layer with an LLM-based planning and communication layer for adaptive serious games and AI validation scenarios. This work aims to suggest the concept of a hybrid ML-Agent and LLM architecture, dividing behavioral execution, LLM planning, validation and action translation into individual layers.

Keywords: Machine Learning, Agents, AI, LLMs, architecture, framework, serious games

1 Introduction

Serious games are increasingly used to support training, decision-making exercises and simulation-based learning. However, many existing systems rely on predefined and scripted scenarios, which limit their adaptability, variability and responsiveness to user behavior.

A case study on the characterization of performance inconsistency pinpoints the important factors on performance inconsistency and variance, using a simple game called Flappy Bird as the experiment's base [1]. In RL domains, the problem of incapability to repeat or reproduce experiment results is common when an environment is generated with random seeds [2]. Aside from the four factors presented by the authors, a supervisory layer to guide the Agent's actions in-between runs could help mitigate performance variance.

Nurse Town is an LLM-Powered simulation game for Nursing Education [3], utilizing GPT-4o for both patient role playing and performance assessment, as it has demonstrated high accuracy and efficiency across various linguistic domains and reasoning capabilities [4]. Various factors are controlled by the LLM, such as player interaction, dialogue, evaluation, which a scripted base keeps the scenario on-topic. A separate LLM component is used for character animations, choosing from a list of pre-defined

© The Author(s) 2026
K. Hollá et al. (eds.), *Proceedings of the SAFAR Scientific Conference 2026 – Building Awareness Through Immersive Learning (SSC-BAIL 2026)*, Atlantis Highlights in Social Sciences, Education and Humanities 57,
https://doi.org/10.2991/978-94-6239-817-7_16

gestures based on the conversation contents. However, the scripted base leads to key limitations, such as a reliance on pre-defined scripts that lack variance. With a limited set of actions the LLM can take within the game environment, it becomes predictable, diminishing the effects of prolonged training. The ability to plan out script variations and execute them would greatly increase scenario variability and experience gain.

Reinforcement learning agents, such as those trained using Unity ML-Agents, can provide dynamic behavior in simulated environments by learning from observations, actions and rewards. However, such agents are primarily optimized for behavior execution rather than high-level interpretation, scenario planning or natural language communication. Complex tasks or direct user communication are typically above the capabilities of standard ML-Agent models [5].

On the opposite end, Large Language Models offer complementary capabilities, including natural language interaction, scenario generation, reasoning over structured context and explanation of decisions. By analyzing context in a variety of data formats, LLMs can gather more information than ML-Agents, and they can directly communicate and collaborate with users through straight verbal communication. However, execution of tasks on the game level tends to be slow and inefficient, with a higher margin for error due to hallucinations or user error [5].

This creates an opportunity to combine ML-Agents and LLMs in a hybrid architecture where ML-Agents control low level agent behavior, while an LLM module supports high level planning, dialogue generation and adaptive scenario management. This work proposes a conceptual framework for such an architecture in the context of adaptive serious games and AI validation scenarios.

The main contributions of this paper are as follows: We propose a conceptual hybrid architecture that combines Unity ML-Agents and LLMs for adaptive serious game scenarios. We define a modular separation between low-level behavioral execution handled by ML-Agents and high-level scenario reasoning handled by an LLM module. We introduce validation and action translation layers to constrain LLM generated scenario adaptations before execution in Unity. We outline an evaluation framework for comparing scripted, ML Agents-based and ML-Agents–LLM-based serious game architectures.

2 Methodology

The Unity Machine Learning Agents Toolkit (ML Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents within the Unity game engine. It provides implementations (based on PyTorch) of state-of-the-art algorithms to enable game developers and practitioners to train intelligent agents for 2D, 3D and VR/AR games. Researchers also use the provided Python API to train Agents using reinforcement learning, imitation learning, neuroevolution, and other methods [6]. These trained agents can be used for multiple purposes, including controlling NPC behavior (in a variety of settings such as multi-agent and adversarial), automated testing of game builds and evaluating different game design decisions pre-release [7].

By scaling model capacity and training on massive, diverse text corpora, LLMs have demonstrated strong capabilities in language understanding, knowledge generalization, and conversational dialogue. Despite these advances, current LLMs are primarily optimized on fixed, static text corpora. Human intelligence, in contrast, develops through continuous sensorimotor engagement with the environment, for example, by forming perceptual representations from repeated interactions that capture the structure and dynamics of the world, and by adjusting behavior in response to feedback from action outcomes that gradually improves performance [8].

Unlike humans, LLM-based agents lack a physical body, making deep participation in real-world interactions difficult and costly. In contrast, game environments provide a natural testbed for realizing the coupling between agent and environment, and offer a richer, more embodied alternative compared to typical settings of current LLM-based agents, such as dialogue, web navigation, or API tool use [8].

Traditional game agents follow a control-based paradigm, where decision-making is coupled through predefined or learned state–action mappings. Finite state machines, behavior trees, and reinforcement learning agents exemplify this design. In contrast, language serves as a unified medium for LLM-based agents to represent goals, contexts, and interactions, enabling explicit reasoning, reflection, and communication beyond traditional systems. Existing systems touch on the topic from different angles yet largely treat games as a downstream application alongside dialogue, tool use, or web automation, leaving the field of LLM-based game agents underexplored. The complexity and openness of game environments distinguish them from narrowly defined tasks. For instance, while a web-based agent may complete a query or transaction through a handful of API calls, a sandbox game enables researchers to cultivate entire agent societies and allows agents to freely explore, interact, and build within physics-driven worlds. These environments afford a degree of freedom that enables emergent behaviors far beyond constrained, task-oriented interactions. On the other hand, game-focused systems emphasize areas such as game development, educational applications, or content generation [8].

While these research streams provide important foundations, they are often studied separately. Reinforcement learning agents are mainly used for behavior optimization in simulated environments, whereas LLM-based virtual agents focus primarily on dialogue and feedback. There is a lack of a modular architecture that combines ML-Agents-based behavioral execution with LLM-based scenario reasoning, scenario validation and AI evaluation in adaptive serious games.

3 Proposed Hybrid Architecture

There are several existing implementation technologies that can be used for certain parts of the architecture: The proposed LLM module can be implemented using either a general-purpose LLM with specific retrieval augmentation or a domain-adapted LLM. For example, in medical scenarios, using MedGemma [9], Meditron [10], BioMistral [11], Me-LLaMA [12] or related models. Meanwhile, the ML-Agent could be an integrated one, using the Unity ML-Agents plugin and PyTorch.

The flow of data between Unity, the ML-Agent and the LLM is not straightforward and must go through conversions and reformatting to be readable on the other side. The format in which the LLM communicates to the user is plain and understandable to human viewers, like a scene description or a dialogue. But to relay information back and forth between the ML-Agent, the module must structure its data – ideally in a format like a structured JSON file – to ease translation back into Engine code and avoid overloading the Agent with redundant information. Consequently, the LLM must be able to read the engine's output at regular intervals, by interpreting output values or through a computer vision overlook of the simulated environment.

3.1 Architecture overview

The architecture diagram presented in Fig.1 depicts a sequential progression between individual layers with parallel pathing. Layers like the Game Environment and ML-Agents Behavioral Layer mainly involve ML-Agent and Engine processes, but are actively influenced by LLM-focused layers, working in parallel – the Simulation State Collector and Evaluation Module gather data throughout the system's operation, and the ML-Agent Behavioral Layer continues to drive the Agent while the LLM Planning Module prepares script alterations.

The ML-Agent does not consult the LLM side for guidance every loop, as it would be redundant and inefficient. The frequency of communication between the two layers would be lower than the operational frequency of each respective layer, depending on the application. Communication can be facilitated periodically, or through an automated query system to discern whether the environment has changed enough to warrant LLM input. Since the Agent will be able to continue operating in-between sending a query to the LLM layer and receiving instructions, the latency should be low enough so that the environment hasn't changed too much in the meantime.

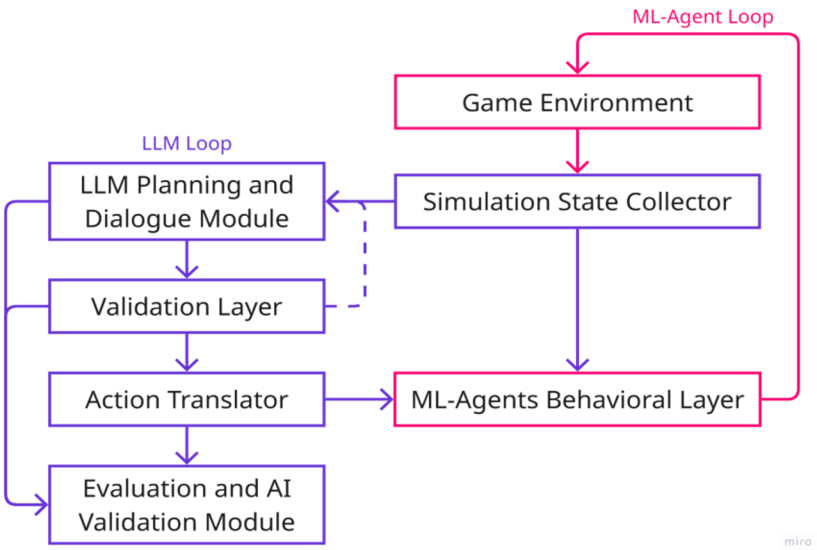


Fig. 1. ML-Agent-LLM proposed architecture diagram

3.2 Game Environment

The Game Environment layer contains a Unity scene, the Agent body, the environment itself, user or AI actions, a UI with vital functions, and event logging. It largely contains anything the agent can directly interact with. Variables like the agent's world position, the position or approximate area of its goal, observations, agent states, and similar.

In a serious game, the Game Environment could be a hospital with doctors and patients, or a construction site with workers and machinery. The accuracy of said environment is limited by the level of detail invested by the developer, thus elevating implementation costs.

3.3 Simulation State Collector

The Simulation State Collector accumulates the current state of the simulation and prepares it for the LLM. The input must be structured and a readable format, and redundant information must be left out to not overload the model with irrelevant data. Depending on the type of simulations, the state collector can collect a different set of observations than the Agent itself requires for its own operation, for example: a bird's eye view of the game environment through, whereas the Agent itself only sees what's right in front of it.

The output must be simple enough to not pollute the LLM with redundant data, but must also be of sufficient quality and not exclude too much information. The data should be structured, ideally in a table format.

3.4 LLM Planning and Dialogue Module

LLM dialogue and planning is not meant to commit physical actions directly in Unity. Its tasks are to interpret simulation states, suggest script changes, generate a dialogue, compile a summary, and to prepare feedback. Subsequently, the validation layer exists to validate the LLM's output, evaluating whether the new script still aligns with the original goal, and whether the actions it suggests aren't too dangerous, difficult, or unrealistic. For example, in a medical simulation, the LLM might make a proposal that could endanger the patient or ignore human physiology, so the validation layer would step in with a rejection. This minimizes the risk of hallucinations and bias, but regular human observation is still advised.

3.5 Validation Layer

The Validation Layer accomplishes this through several types of validation: rule-based validation, retrieval-augmented validation, expert-defined constraints, permitted action list, and context consistency check. In addition, certain context sensitive rules must be applied to handle general or edge cases. For example, "Vital functions must remain

within realistic levels", "Deterioration must be gradual", and "Output must be in the desired JSON format."

The reason that planning, validation, and evaluation of LLM output are three separate layers is to more efficiently root out potential problems and errors in judgement. Performing all of these functions within a single layer would be too much for a single prompt. Adapting a step-by-step process allows for course corrections and gradually shaping a desirable output.

3.6 Action Translator

Turning the validated LLM's output into actions for the Agent is the Action Translator's task, translating the script into variables and functions within the ML-Agent's permitted set of actions. These translated actions are then sent to the Agent to perform, like a change of variables or a change of targets. There would be actions that the Agent does not take unless prompted to. The Action Translator is still an LLM layer, so it is implemented with clear prompts and instructions on what kind of code and syntax the Agent and its environment will accept, and the output is then sent to the Engine for processing.

The output must be readable for the Engine and the agent. In the case of Unity, lines of code that apply the new adjusted values and execute predefined functions from a list that is fed into the translator. For example, converting information from a structured JSON file into C# for Unity. It is the main bridge where information travels from the LLM loop to the ML-Agent loop.

The Action Translator must be more rigid than the other LLM layers, to avoid hallucinating functions that do not exist, misplacing values, or duplicating actions. A syntax validator can ensure that the generated code will be readable on the receiving end. If an error does occur, the Agent could send back a request for a revised translation, instead of going through the entire loop again.

3.7 ML-Agents Behavioral Layer

The Behavioral Layer of the ML-Agent handles the Agent's physical actions, encompassing movement, local reactions, animations, interaction with other agents, user response, and adaptive behavior. While the LLM decides what should happen and why, the ML-Agent decides how the Agents behave in the environment.

3.8 Evaluation and AI Validation Module

While training is underway, the Evaluation and AI Validation Module gathers and measures data – reaction time, correctness of actions, critical mistakes, number and frequency of script adjustments, quality of communication, and system failures. The precise evaluation criteria depend on the Agent and scenario being trained and tested, but it serves as a second correction layer for the LLM loop of the system, on top of each layer's internal validation process. It utilizes information from the Simulation State Collector to ensure the LLM loop remains on course with its original objective, as well as considering previous simulation states to evaluate the system's progress. It is the

most crucial layer for feedback and progressive improvement of the system as a whole; be it through supervised and user made adjustment or automated self-adjustment.

The layer may hallucinate non-existent errors or misconstrue their cause, so major code or script revisions should go through user supervision and approval.

4 Illustrative Use Case

For the purposes of this research, we will be proposing a medical scenario as an illustrative use case for the ML-Agent-LLM architecture. Disclaimer – medical LLMs are not automatically safe, nor are they clinically validated. In the case of MedGemma, Google states that models require validation for concrete use and may be further adapted through prompting, fine-tuning, or agentic orchestration [9]. That is why in the proposed architecture, the LLM is meant to be used as: a planning module, a dialogue module, to generate script variations, and as a feedback module – not as an autonomous medical decision-making system without supervision. In safety-critical medical serious games, the LLM should not directly execute clinical decisions. Instead, its output should be constrained by structured prompts, validated by a medical rule-based or retrieval-augmented module and translated into predefined executable actions.

4.1 Scenario

The initial script is thus: patient presents with chest pain.
Patient: 68-year-old male
Initial symptoms: chest pain, sweating, shortness of breath
Initial vital signs: blood pressure (95/65), heart rate (105), oxygen saturation (93%)
Scenario goal: Test whether the user or AI system recognizes clinical deterioration and initiates appropriate response.

The process of using the projected system would be as follows:

1. User asks the patient a question.
2. LLM generates the patient's answer.
3. LM-Agent controls the patient's behavior.
4. In the absence of user intervention, the LLM suggests deterioration of the patient's state.
5. Validator checks whether the deterioration is realistic.
6. Action Translator initiates Unity actions.
7. Validation Module evaluates reactions.

Game Environment loads the patient state:

- age
- symptoms
- blood pressure
- heart rate
- oxygen saturation

— consciousness level
— pain level

The state collector then translates the current patient state for the LLM in a JSON format:

```
{
"scenario": "chest_pain",
"patient":{
"age": 68,
"symptoms": ["chest pain", "sweating", "shortness of
breath"],
"blood_pressure": "85/55",
"heart_rate": 118,
"oxygen_saturation": 89,
"consciousness": "confused"
},
"user_actions":[
"asked_about_symptoms",
"checked_pulse"
],
"time_elapsed_seconds": 120,
"scenario_goal": "test recognition of critical deteriora-
tion"
}
```

Through the LLM planning and dialogue module, a course of action is proposed:

```
{
"scenario_intent": "increase_urgency",
"patient_dialogue": "I feel worse... it is harder to
breathe.",
"proposed_event":{
"type": "oxygen_saturation_drop",
"target": "patient_01",
"change": -4,
"duration_seconds": 60
},
"reason": "The user has not initiated oxygen support or
emergency response."
}
```

To evaluate the following output, the Medical Validation Layer accepts or rejects the previous layer's proposal.
LLM proposal: SpO2 drops from 98% to 40% within 5 seconds.

Validation result: Rejected or modified because the change is too abrupt for the selected scenario.

With a validated course of action, the Action Translator will rewrite the proposal from JSON into Unity C# in accordance with the chosen method and permitted action list.

Proposal in JSON:

```
{
"type": "oxygen_saturation_drop",
"change": -4,
"duration_seconds": 60
}
Translated action:
SetVitalSign(patient_01, "SpO2", current_value - 4, 60)
PlayAnimation(patient_01, "breathing_difficulty")
TriggerDialogue(patient_01, "dyspnea_line_02")
NotifyAgent(nurse_01, "patient_deteriorating")
```

Finally, the ML-Agents behavioral layer carries out the proposed changes and course of action within the game scene and the agents within it, like Agent movement, local reactions, animations, patient or personnel behavior, response to user intervention, and adaptive behavior.

The process would then repeat itself while the Evaluation layer gathers critical data in-between cycles, such as medical consistency, number of script alterations, performance speed, correctness of actions, and the system's ability to identify errors.

Concerning safety, expert supervision could be integrated into the evaluation chain, where the final proposed action plan would be presented to a user to accept or reject it. This could be done with all queries and proposals, or only those that have been deemed potentially dangerous by the Validation Layer. Users would also be allowed to pause the scenario and check each layer's input and output individually and directly monitor the flow of information.

5 Evaluation Framework

To assess the potential of the new hybrid architecture, we compare its projected advantages and disadvantages against existing architectures:

A: Scripted serious game

B: ML-Agents-based serious game

C: ML-Agents + LLM + Validation Layer

5.1 Scripted serious game

A scripted framework relies on hard-coded logic, state machines, and behaviour trees. Developers explicitly write every rule, response and consequence of the simulation en-

vironment. This framework is thus predictable and reliable but limited with deterministic outputs and lacking adaptability. Due to its rigid structure, it cannot handle unforeseen user actions or edge cases. While its simplicity leads to minimal hardware needs, its low upfront development cost rises high when scaling with complexity.

5.2 ML-Agents-based serious game

Working off a scripted base, it primarily relies on reinforcement learning and neural networks. Agents learn optimal behaviors by interacting with the game environment and receiving rewards and penalties. If sufficiently trained, Agents can be highly adaptive and respond dynamically to unique situations or user interactions. But development costs are high and hardware needs are intensive. The Agent still struggles in scenarios it wasn't trained for and falls behind when placed in increasingly complex situations.

5.3 ML-Agents + LLM + Validation Layer

ML-Agents-based serious games provide efficient low-level intelligence and adaptation, while the addition of an integrated LLM can substitute higher level functions that previously lacked automation or required user input. The Agent should be able to better analyze its environment, plan out a course of action, and directly communicate with the user. The validation layer evaluates the LLM's output.

To efficiently compare the given architectures against each other and evaluate the effectiveness of the proposed hybrid architecture, we present an evaluation framework based on the following criteria detailed in Table I. This is a suggestion for future evaluation methods, not a complete experiment.

Table 1. Evaluation criteria and metrics

Evaluation	
Criteria	**Metrics**
Scenario variability	Number of unique generated scripts and their diversity, to avoid stagnant and duplicated scenarios and encourage progression and advancement
Scenario consistency	Percentage of scripts that are in accordance with basic scenario rules, instructions, environment changes, and past actions
Realistic behavior	Realism of Agent behavior, depends on scenario requirements; requires expert user observation and evaluation
Natural dialogue	How accurate the LLM output is to natural and fluent conversation; by external LLM and expert user observation and evaluation, spell checking
Adaptivity	Number of suitable adaptations based on user actions and external factors, the system's ability to respond to changes within the environment
Explainability	Quality and comprehension of system feedback and comments, explanation of errors; requires expert user evaluation
System reliability	Number of technical errors and system faults, faulty or missed corrections, mistranslations, performance fluctuations, latency spikes
Safety	Number of invalid/rejected dangerous proposals, expert user evaluation
Latency	LLM module response time, communication speed between layers, communication between LLM and Agent loops, frequency of communication

6 Discussion

Thanks to the use of LLM for analysis and planning, we expect higher scenario variability, natural and realistic agent communication, and the ability to generate edge-case scenarios. With the validation module, we would also reach higher process comprehension.

However, there are still projected risks that may occur [13], detailed as follows:
Inaccuracy of outputs:
- Data scarcity (esp. rare conditions)
- LLM hallucinations
- LLM reasoning errors
- Real-time processing constraints

Scenario validation errors:
- Inconsistent data formats
- Lack of standardized workflows
- Rare case representation

Arduous technical implementation:
- Integration into existing systems
- Workflow disruptions
- Maintenance & updates
- Provider trust & acceptance
- Requires expert evaluation

Ethics and safety concerns:
- Data privacy (HIPAA, GDPR)
- Bias & underrepresented groups
- Black-box reasoning
- Continuous-learning certification
- Accountability & transparency
- Cybersecurity threats

6.1 Potential solutions

The communication between the LLM and Unity/ML layers can be improved by using a structured JSON format, while output validation should be performed by a separate rule-based model rather than the model that created the output itself. Through a limited list of permitted actions, the translated output would be more streamlined and easier to read for the Engine and ML-Agent on the receiving end.

Decision logging would help the system keep track of previous actions to avoid repetitive behavior and provide users with additional data to analyze and to pinpoint potential strengths and weaknesses in the architecture and its individual layers, mainly the LLM. Additionally, an expert control option would help with the testing and exploration of edge cases and error replication. Finally, dialogue generated for user interaction could be divided from critical decision making, to avoid polluting the input with irrelevant information.

6.2 Limitations

It is vital to note that the architecture suggested in this paper has not yet been implemented and only exists on a conceptual level. Likewise, the medical use case is an illustrative example of how the architecture could work in practice. The LLM output lacks clinical validation, and the framework requires expert evaluation.

Aspects of the system, such as latency, safety, reliability and scalability have not yet been tested.

7 Conclusion

This paper proposed a conceptual hybrid architecture that combines Unity ML-Agents and large language models for adaptive serious games. The architecture separates low-level behavioral execution from high-level scenario planning and introduces medical validation and action translation layers to ensure safety and executability. It aims to combine the benefits of both LLM and ML-Agent integration, so that the two systems rely on and benefit from each other. Currently, the system still exists only in a conceptual form, and the shortcomings of the combined systems present unique problems of safety, ethics, and implementation. Future work will focus on creating a pilot prototype, evaluating the architecture in a selected scenario and comparing it with scripted and ML-Agents-only approaches. Selecting the ideal simulation environment, LLM and ML-Agent plugin is a vital step, comparing various compositions and discerning preferable use cases. Early tests would measure speed and accuracy, and once a more sophisticated prototype is developed, more complex scenarios would be utilized to test longevity and real-world application.

Acknowledgment

This article was supported by the project 101100700-TEF-Health - Digital Europe Work Programme.

References

1. A. Shakerimov, D. Li, and J.-G. Park, “A Case Study: Characterization of Performance Inconsistency for Reinforcement Learning on Flappy Bird Game,” 2021 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1–6, Oct. 2021.
2. Lynnerup, Nicolai A., Laura Nolling, Rasmus Hasle, and John Hal-lam. “A Survey on Reproducibility by Evaluating Deep Reinforcement Learning Algorithms on Real-World Robots.” In Conference on Robot Learning, pp. 466–489. PMLR, 2020.
3. Y. Hu, Q. Xiong, L. Yi, and I. Yoon, “Nurse Town: An LLM-Powered Simulation Game for Nursing Education,” 2025 IEEE Conference on Artificial Intelligence (CAI), pp. 215–222, May 2025.

4. S. Shahriar, B. D. Lund, N. R. Mannuru, M. A. Arshad, K. Hayawi, R. V. K. Bevara, A. Mannuru, and L. Batool, “Putting GPT-4o to the sword: A comprehensive evaluation of language, vision, speech, and multimodal proficiency,” Applied Sciences, vol. 14, no. 17, pp. 7782, 2024.
5. S. Jusoh and H. Al Fawareh, “AI Games and Algorithms: An Overview of Categories,” 2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), pp. 1–6, June 2023.
6. A. Juliani et al., “Unity: A General Platform for Intelligent Agents,” arXiv.org, 2018.
7. R. Meshram, S. Krishna, O. Kulkarni, R. R. Patil, G. Kaur, and S. Maheshwari, “NPC Behavior in Games Using Unity ML-Agents: A Reinforcement Learning Approach,” 2025 International Conference on Automation and Computation (AUTOCOM), pp. 1519–1523, Mar. 2025.
8. S. Hu et al., “A Survey on Large Language Model-Based Game Agents,” arXiv.org, 2024.
9. S. Liu and J. Kim, “MedGemma-Critic: Fine-Tuning Medical Language Models for Domain-Specialised Text Evaluation,” 2025 International Conference on Digital Image Computing: Techniques and Applications (DICTA), pp. 1–5, Dec. 2025.
10. A. Meirkhan and P. Kumar, “AI-Powered Medical Chatbot with Neural Network Symptom Analysis and Automated Appointment Scheduling,” 2026 18th International Conference on Electronics, Computer, and Computation (ICECCO), pp. 1–6, Apr. 2026.
11. K. Sinha, V. Singh, A. Vishnoi, P. Madan, and Y. Shukla, “Healthcare Diagnostic RAG-Based Chatbot Triage Enabled by BioMistral-7B,” 2024 International Conference on Emerging Technologies and Innovation for Sustainability (EmergIN), pp. 333–338, Dec. 2024.
12. Xie, Qianqian, et al. "Me-LLaMA: Foundation Large Language Models for Medical Applications" (version 1.0.0). PhysioNet, 20
13. S. Vatsal, H. Dubey, and A. Singh, “Agentic AI in Healthcare and Medicine: A Seven-Dimensional Taxonomy for Empirical Evaluation of LLM-Based Agents,” *IEEE Access*, vol. 14, pp. 4840–4863, 2026, doi: 10.1109/access.2026.3651218.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

