Hybrid LLM-Agent Architecture for Autonomous Life Simulation: Design and Implementation of Multi-Agent Virtual World

Albert Aksenov
Software Engineering Department
Master's Program, Group SE-01
Course: LLM and AI Agents
Email: al.aksenov@innopolis.university

Project Repository

The project code is available at: https://github.com/CrazyAngelm/llm_life-simulation

***Abstract*—This paper presents the design and implementation of a hybrid Large Language Model (LLM) and AI Agent architecture for autonomous life simulation. The system shows practical integration of local LLM inference through Ollama and cloud-based APIs for multi-agent decision making in a virtual world environment. Our implementation features 10 autonomous Non-Player Characters (NPCs) with dynamic states, social relationships, and decision-making capabilities. These combine rule-based logic with LLM-generated responses. The simulation spans multiple locations and includes time progression with emergent behaviors from agent interactions. Performance evaluation shows successful autonomous operation over extended periods. It maintains narrative coherence and computational efficiency. The system serves as a practical demonstration of modern AI technologies in interactive simulation environments.**

***Index Terms*—Large Language Models, Multi-Agent Systems, Virtual World Simulation, Hybrid AI Architecture, Autonomous Agents, Decision Making Systems.**

I. Introduction

The integration of Large Language Models (LLMs) with autonomous agent systems represents a significant advancement in artificial intelligence applications. This work presents a practical implementation of such integration through a life simulation system that demonstrates both theoretical concepts and real-world applicability of hybrid AI architectures.

Traditional simulation systems rely mainly on deterministic rule-based logic. This is predictable and efficient, but often lacks creative variability and contextual understanding that modern applications require. On the other hand, pure LLM-based systems are highly creative and contextually aware. However, they face challenges in consistency, computational cost, and long-term coherence in complex scenarios.

Our approach addresses these limitations through a hybrid architecture that strategically combines deterministic agent logic with selective LLM integration. The system maintains 10 autonomous NPCs across three distinct locations, each with individual states, relationships, and decision-making capabilities.

The primary contributions of this work include: (1) A modular architecture that efficiently integrates local and cloud-based LLM services, (2) A hybrid decision-making framework that balances creativity with consistency, (3) A practical demonstration of multi-agent systems with emergent social behaviors, and (4) Performance analysis of LLM integration costs and benefits in simulation environments.

II. System Architecture

A. Overall Design Philosophy

The system architecture follows a modular design pattern that separates concerns between agent management, LLM integration, world simulation, and state persistence. This separation enables flexible configuration of LLM usage, easy scaling of agent populations, and maintainable code structure.

The core principle of our design is selective LLM use. Rather than routing all decisions through expensive LLM calls, the system uses a probabilistic approach. 30% of agent decisions involve LLM reasoning while 70% rely on efficient rule-based logic. This balance maintains creative unpredictability while controlling computational costs.

B. Component Architecture

The system comprises five primary components:

World Simulator (simulator.py): The central orchestrator that manages temporal progression, coordinates agent actions, and maintains world state consistency. It implements the main simulation loop and handles day-night cycles, aging, and event processing.

Agent Models (models.py): Defines the NPC and Location classes that encapsulate agent state, relationships, and behavioral parameters. NPCs maintain dynamic statistics including health, energy, hunger, and mood, alongside social relationship matrices with other agents.

LLM Manager (llm_clients.py): Provides abstracted access to multiple LLM services including local Ollama integration and cloud-based DeepSeek API. This component handles connection management, prompt engineering, and response parsing.

Configuration System (config.py): Centralizes simulation parameters, LLM settings, agent definitions, and world initialization data. This enables easy experimentation with different configurations without code modification.

Main Controller (main.py): Provides the entry point and high-level coordination, including initialization, simulation execution, and result output management.

C. Agent State Management

Each NPC maintains a comprehensive state representation that evolves throughout the simulation. The state includes:

- **Static Attributes:** ID, name, role, and age
- **Dynamic Statistics:** Health (0-100), energy (0-100), hunger (0-100), mood (0-100)
- **Social Networks:** Relationship values (-100 to +100) with all other agents
- **Behavioral State:** Current location, daily actions, and survival status

The system implements bounded value updates to maintain realistic constraints on all statistical measures, preventing unrealistic extremes that could destabilize the simulation.

III. LLM Integration Strategy

A. Hybrid Decision Architecture

The decision-making process uses a two-tier approach that balances computational efficiency with creative output. For each agent decision point, the system first evaluates whether LLM consultation is needed. This is based on configurable probability thresholds and agent state complexity.

Rule-based decisions handle routine activities such as basic survival actions, location-specific behaviors, and predictable social interactions. These decisions execute rapidly and maintain consistent behavioral patterns aligned with agent roles and current states.

LLM-enhanced decisions address complex social situations, creative problem-solving, and narrative-rich interactions. These calls provide contextual awareness that considers full world state, agent relationships, and historical actions. They generate nuanced responses that would be difficult to encode in deterministic rules.

B. Multi-LLM Service Integration

The system integrates two distinct LLM services to optimize for different use cases:

Ollama (Local LLM): Provides fast, cost-effective inference for routine decision-making. The local deployment ensures privacy and eliminates API rate limiting while supporting the llama3.2 model for moderate complexity reasoning tasks.

DeepSeek API (Cloud LLM): Handles complex narrative generation, particularly for end-of-simulation chronicle creation. The cloud service provides access to more sophisticated models for tasks requiring extensive context understanding and creative writing capabilities.

This dual-service approach optimizes both cost and capability, utilizing local resources for frequent operations while leveraging cloud capabilities for specialized tasks.

C. Prompt Engineering and Context Management

Effective LLM integration requires careful prompt engineering that provides sufficient context while maintaining token efficiency. The system constructs prompts that include:

- Current agent state and recent actions
- Relevant relationship information with nearby agents
- Location context and available activities
- Temporal information and world state summary

Context windows are carefully managed to include essential information while avoiding token limits that could truncate critical context or increase inference costs.

IV. Implementation Details

A. Simulation Loop and Temporal Management

The core simulation implements a discrete time progression model where each day represents a complete cycle of agent activities. The daily loop processes:

1) State initialization and cleanup from previous day
2) Age progression and vital statistic updates
3) Rule-based decision processing for all agents
4) LLM-enhanced decision processing for selected agents
5) Random event generation and resolution
6) Relationship updates based on interactions
7) State persistence and logging

This structured approach ensures consistent temporal progression while maintaining agent autonomy and interaction opportunities.

B. Social Relationship Modeling

Agent relationships form a dynamic network that evolves based on interaction outcomes. The relationship system tracks bidirectional relationship values between all agent pairs, updated through:

Interaction-based updates: Direct agent interactions modify relationship values based on action outcomes and compatibility factors.

Proximity influence: Agents sharing locations experience gradual relationship changes based on extended contact.

Event-driven modifications: Significant events can cause rapid relationship shifts that reflect dramatic social changes.

The relationship matrix serves as input for LLM decision-making, enabling contextually appropriate social behaviors that reflect established agent connections.

C. Asynchronous Processing and Performance

The system employs Python's asyncio framework to handle concurrent LLM requests efficiently. This approach prevents simulation blocking during API calls and enables parallel processing of multiple agent decisions when computational resources permit.

Asynchronous processing is particularly valuable during LLM-heavy simulation periods, where multiple agents may require complex reasoning simultaneously. The implementation includes proper error handling and fallback mechanisms to maintain simulation stability when external services are unavailable.

V. Experimental Results and Analysis

A. Simulation Performance Metrics

Performance evaluation was conducted over multiple 25-day simulation runs with varying LLM utilization rates. Key metrics include:

Temporal Performance: Average daily processing time remained under 2 seconds for 10 agents with 30% LLM utilization, demonstrating real-time capability.

Agent Autonomy: All agents maintained active status throughout simulation periods, with dynamic state changes and relationship evolution observed consistently.

Narrative Coherence: Generated events and agent actions maintained logical consistency with established world rules and agent characteristics.

System Stability: No critical failures occurred during extended runs, with graceful degradation when external LLM services were unavailable.

B. LLM Integration Analysis

The hybrid approach proved effective in balancing creativity with efficiency. LLM-generated decisions demonstrated significantly higher narrative quality and contextual appropriateness compared to pure rule-based alternatives, while the selective application strategy maintained reasonable computational costs.

Response quality analysis showed that context-aware prompts produced coherent agent actions that reflected both individual agent characteristics and broader social dynamics. The dual-LLM architecture successfully optimized for both routine decision-making and complex narrative generation.

C. Emergent Behaviors and Social Dynamics

Extended simulation runs revealed emergent social patterns arising from agent interactions. Relationship networks evolved into complex hierarchies that reflected agent roles and interaction histories. These emergent structures were not explicitly programmed but arose naturally from the interaction of simple behavioral rules and contextual LLM decisions.

Particularly interesting patterns included alliance formation among compatible agents, gradual relationship deterioration between incompatible personalities, and adaptive behavioral changes in response to social pressures.

VI. Challenges and Limitations

Several challenges emerged during development and testing:

LLM Consistency: Maintaining behavioral consistency across multiple LLM calls required careful prompt engineering and validation mechanisms.

Cost Management: API costs for cloud-based LLM services necessitated strategic usage patterns and careful token management.

Scalability Constraints: Current implementation scales effectively to dozens of agents but would require architectural modifications for hundreds or thousands of entities.

State Complexity: As agent relationships and world state become more complex, context management for LLM calls becomes increasingly challenging.

VII. Future Work and Improvements

Several avenues for enhancement have been identified:

Hierarchical Agent Management: Implementing multiple agent activity levels (active, background, dormant) could improve scalability while maintaining simulation richness.

Advanced Relationship Modeling: More sophisticated relationship types and dynamics could enhance social simulation realism.

Improved LLM Integration: Fine-tuned models specifically trained for simulation contexts could improve response quality while reducing costs.

Extended World Complexity: Additional locations, economic systems, and environmental factors could create richer simulation experiences.

VIII. Conclusion

This work shows successful integration of LLM capabilities with autonomous agent systems in a practical simulation environment. The hybrid architecture balances computational efficiency with creative capability. It produces emergent behaviors and narratives that are better than purely rule-based approaches.

The modular design allows flexible experimentation with different LLM integration strategies. It maintains system stability and performance. The implementation serves as both a practical demonstration of modern AI technologies and a foundation for more complex multi-agent systems.

Key findings include the effectiveness of selective LLM use for cost management. Also important is careful prompt engineering for consistent agent behavior. The system shows potential for emergent social dynamics in well-designed multi-agent environments.

The system successfully demonstrates practical applications of LLM and AI Agent technologies. This integrated architecture could serve as a foundation for more complex interactive systems, gaming applications, and social simulation research.

IX. AI Assistance Disclosure

The author acknowledges the use of artificial intelligence tools in the preparation of this document. AI assistance was used for the following purposes:

- Translation from Russian to English language
- Grammar checking and language correction
- Document formatting according to IEEE standards
- Text structure organization and improvement
- Technical terminology verification

All technical concepts, system design, implementation details, and experimental results represent the author's original work and analysis. The core ideas, architecture decisions, and research findings are the product of the author's independent study and development efforts.

The AI tools were used as writing assistants to improve clarity and presentation, but did not generate the fundamental content or conclusions of this research.

REFERENCES

[1] Brown, T., et al. "Language Models are Few-Shot Learners." Advances in Neural Information Processing Systems 33 (2020): 1877-1901.

[2] Stone, P., and M. Veloso. "Multiagent Systems: A Survey from a Machine Learning Perspective." Autonomous Robots 8.3 (2000): 345-383.

[3] OpenAI. "GPT-4 Technical Report." arXiv preprint arXiv:2303.08774 (2023).

[4] Wooldridge, M. "An Introduction to MultiAgent Systems." John Wiley & Sons, 2009.

[5] Russell, S., and P. Norvig. "Artificial Intelligence: A Modern Approach." Pearson, 4th edition, 2020.

[6] Vaswani, A., et al. "Attention is All You Need." Advances in Neural Information Processing Systems 30 (2017): 5998-6008.