# Empirical Evaluation of Single-Agent and Multi-Agent LLM Systems for Nuclear Reactor Control

Joowon Cha [a,b], Soyeon Kim [a,b], Seung Geun Kim [a], Yonggyun Yu [a,b,*]

[a]*Applied Artificial Intelligence Section, Korea Atomic Energy Research Institute, Daejeon, Korea*
[b]*University of Science and Technology, 217 Gajeong-ro, Yuseong-gu, Daejeon 34113, Korea*
[*]*Corresponding author: ygyu@kaeri.re.kr*

***Keywords :** Agentic AI, Large Language Model, Multi-agent system, Single-agent system, Nuclear reactor control*

## 1. Introduction

The adoption of artificial intelligence (AI) in the nuclear industry is being actively promoted to enhance operational efficiency and reduce human error risks [1]. In particular, autonomous agent systems based on Large Language Models (LLMs) are attracting attention as technologies capable of interpreting complex procedures, performing multi-step reasoning, and executing control actions in real-world environments [2].

The Main Control Room (MCR) of a nuclear power plant presents a challenging environment for AI-based assistance. Operators must coordinate multiple monitoring tasks, follow detailed procedural sequences, and verify each action before execution. Previous research demonstrated that LLM agents combined with Retrieval-Augmented Generation (RAG) and Function Calling mechanisms can identify abnormal plant conditions and suggest appropriate responses [3].

However, concentrating all cognitive functions within a single agent introduces risks of verification omission under heavy processing loads. An alternative approach distributes responsibilities across multiple specialized agents communicating through structured protocols, which can potentially mirror the team-based coordination in actual MCR operations [4].

This study compares single-agent and multi-agent LLM architectures through 100 experimental trials per configuration, quantifying differences in task completion rates, computational costs, and knowledge retrieval patterns. The extended trial count provides enhanced statistical reliability compared to smaller-scale preliminary studies.

## 2. Background

This section establishes the theoretical foundation for Agentic AI and the rationale for comparing single-agent versus multi-agent architectures in nuclear reactor control applications.

### *2.1 From LLM to Autonomous Agents*

Large Language Models are trained on extensive text corpora and demonstrate strong performance in context understanding and natural language generation [2]. However, standalone LLMs have inherent limitations: they cannot maintain state between sessions, access real-time external information, or interact with physical systems.

Agentic AI addresses these limitations by augmenting base models with three functional components: a reasoning module for goal decomposition and decision sequencing, a memory module for preserving context across interactions, and a tool interface module for connecting with external systems.

The tool interface operates through Function Calling, where the model generates structured outputs containing function names and typed parameters. This mechanism enables translation from natural language reasoning to concrete operations such as reading sensor values or adjusting equipment settings in a plant simulator.

### *2.2 Retrieval-Augmented Generation*

A critical limitation of LLMs is hallucination, where the model generates plausible but factually incorrect information [5]. In safety-critical nuclear applications, acting on fabricated procedures could lead to improper equipment manipulation with serious consequences.

Retrieval-Augmented Generation (RAG) mitigates this risk by injecting verified reference material into the generation process. Documents are partitioned into segments and transformed into vector embeddings stored in a searchable database. When guidance is required, the system retrieves passages semantically related to the current query and provides them as contextual constraints, anchoring outputs to authoritative documentation.

### *2.3 Architectural Comparison*

A single-agent design consolidates perception, planning, and execution within one model instance. This approach offers advantages including implementation simplicity, lower computational overhead, and uninterrupted context flow. However, it concentrates all failure modes in a single component without external verification checkpoints.

Multi-agent designs partition cognitive tasks across separate specialized agents communicating via message passing [4]. A supervisory agent can approve or reject plans before execution, providing defensive layers against erroneous actions. The trade-offs include additional token overhead for inter-agent communication and increased orchestration complexity.

Nuclear MCRs employ team structures where different personnel handle monitoring, procedure interpretation, and equipment operation with mutual verification. This operational model motivates exploring whether analogous agent role separation yields measurable benefits in automated assistance systems. Determining the optimal architecture requires empirical evaluation under controlled conditions.

## 3. System Implementation

This section details the implementation of single-agent and multi-agent systems, both operating on identical infrastructure to isolate architectural differences as the sole experimental variable.

### *3.1 Overall Architecture*

The experimental platform interprets natural language commands, interfaces with a nuclear plant simulator, and logs performance metrics. Core components include a workflow manager for request routing, an agent module for reasoning, a vector database for procedure storage, and a function interface for translating decisions into simulator commands.

Both configurations share this common infrastructure, isolating architectural differences as the sole experimental variable. A monitoring module records real-time plant status, while an evaluation module computes performance metrics after each trial.

### *3.2 Single-Agent Configuration*

In the single-agent configuration, a single GPT-4o instance with an operator persona manages all tasks: parsing user requests, querying procedures via RAG, formulating action sequences, and issuing control commands. Context accumulates within one conversation thread, maintaining continuity but providing no external checkpoint before actions reach the simulator.

The roles of Leader, Planner, and Executor are conceptually performed within this single agent, but without explicit separation or inter-role verification mechanisms.

### *3.3 Multi-Agent Configuration*

The multi-agent system implements a three-tier Leader-Planner-Executor hierarchy using LangGraph's StateGraph framework. The Leader agent receives requests, delegates planning tasks, and grants execution approval based on safety criteria. The Planner agent queries the procedure database and drafts step-by-step action proposals. The Executor agent carries out approved plans and reports outcomes.

Communication flows through a shared state structure containing request history, proposed plans, approval status, and execution logs. When the Leader rejects a proposal, corrective feedback is provided for the Planner to incorporate into revised submissions. This iterative loop enables error recovery before commands reach the physical system.

### *3.4 RAG and Function Calling Modules*

Operational procedures are segmented into semantic chunks, encoded as vector embeddings, and indexed in a searchable database with query caching for optimization. Agents perform semantic searches before plan formulation, grounding outputs in verified documentation to prevent hallucinated procedures from reaching execution.

The Function Calling module maps agent decisions to simulator memory addresses, providing read/write access to key plant parameters including reactor power, control rod positions, coolant temperatures, pressurizer levels, and valve states. Each function call is validated against predefined safety constraints before execution.

## 4. Experiment and Results

This section presents the experimental design and quantitative comparison results from 100 independent trials per configuration.

### *4.1 Experimental Setup*

All trials used GPT-4o (gpt-4o-2024-08-06) with temperature set to 0.3 for consistent responses. Agent orchestration was implemented using Python-based

LangChain and LangGraph frameworks. Application Programming Interface (API) costs were calculated based on GPT-4o pricing (input: $2.50/1M tokens, output: $10.00/1M tokens) and text-embedding-3-small costs ($0.13/1M tokens).

The test scenario replicates a reactor trip followed by controlled restart, spanning 26 procedural steps with 66 discrete actions. The sequence covers trip initiation, parameter verification, control rod repositioning, criticality approach via boron dilution, power ascension, and turbine synchronization. Table I summarizes the procedural breakdown.

Table I: Procedural Breakdown of Test Scenario

| Steps | Phase Description | Actions |
|---|---|---|
| 1–5 | Trip initiation and confirmation | 18 |
| 6–13 | Parameter stabilization and reset | 10 |
| 14–17 | Rod movement and criticality | 11 |
| 18–26 | Power raising and turbine sync | 27 |
| **Total** | **26 Steps** | **66** |

Six metrics were recorded: Task Completion Rate (percentage of successful scenario completions), Token Usage, API Cost, Total Execution Time, RAG Query Count, and Estimated Embedding Tokens. Each configuration underwent 100 independent trials for statistical reliability.

### *4.2 Experimental Results*

Each trial began with simulator initialization to default conditions, proceeded through agent-driven command execution following the procedural sequence, and concluded with automatic metric logging. Figure 1 present results averaged over 100 independent runs per configuration.

The multi-agent configuration achieved a task completion rate of 78.46%, compared to 66.12% for single-agent, representing a 12.34 percentage point improvement (18.7% relative gain). Token consumption was higher for multi-agent (121.54K vs 76.83K), resulting in increased API costs ($0.3284 vs $0.2061). Despite the communication overhead, execution times remained comparable (219.87s vs 228.34s), suggesting efficient task distribution in the multi-agent system.

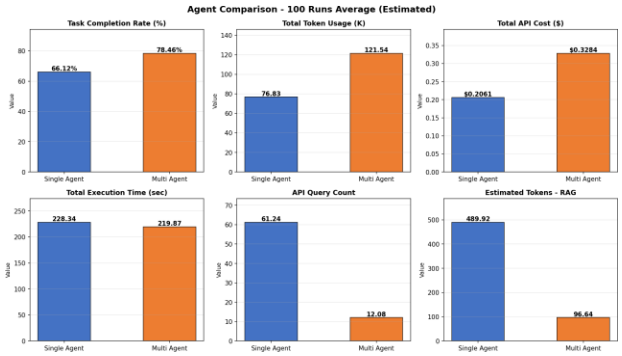


Fig. 1. Performance comparison averaged over 100 trials.

A notable difference emerged in RAG query behavior. The single-agent issued an average of 61.24 queries with 489.92 embedding tokens, while the multi-agent used only 12.08 queries and 96.64 tokens, representing approximately 80% reduction. This efficiency gain is attributed to information sharing through the shared state structure, where retrieved procedures remain accessible to all agents without redundant fetching.

Failure analysis of unsuccessful trials revealed that errors were concentrated in the criticality approach and power raising phases (Steps 14–26), where procedural interdependencies are highest. In the single-agent configuration, failures were primarily attributed to context overwriting within the single conversation thread, causing earlier retrieved procedure constraints to be effectively lost as the sequence progressed. In the multi-agent configuration, the Leader agent's rejection mechanism intercepted erroneous proposals before simulator execution; however, repeated rejection cycles occasionally resulted in stalled execution when the Planner failed to produce an approvable plan within the iteration limit, accounting for a substantial portion of the remaining failures in that configuration.

## 5. Discussion

The higher task completion rate of the multi-agent configuration suggests that the explicit approval step before execution may help prevent erroneous plans from propagating to the simulator. When the Leader agent reviews and authorizes each action sequence, flawed proposals can potentially be identified and revised before affecting the physical system. This mechanism is analogous to the mutual verification protocols employed in actual MCR operations.

The substantial reduction in RAG queries for the multi-agent system appears to result from context propagation through the shared state structure. Procedure excerpts retrieved by the Planner remain accessible to other agents, which may eliminate redundant lookups that occur when a single agent must re-query at each decision point. While inter-

agent messaging increases token consumption, the retrieval savings partially offset this overhead, resulting in comparable execution times between configurations.

The approximately 59% increase in API cost represents a trade-off between reliability and computational expense. For safety-critical applications where task completion is paramount, this cost premium may be acceptable.

Several limitations should be noted. First, this study compared only one multi-agent topology (Leader-Planner-Executor) without examining performance variations with different agent counts or alternative role distributions. Second, experiments were conducted on a single scenario (Reactor Trip and Restart); validation across diverse operational scenarios including emergency procedures would strengthen the generalizability of findings. Third, while 100 trials provide reasonable statistical basis, formal confidence interval analysis would further clarify result stability.

## 6. Conclusions

This study compared single-agent and multi-agent LLM architectures for nuclear reactor control through 100 experimental trials per configuration. The multi-agent system with Leader-Planner-Executor hierarchy achieved a task completion rate of 78.46%, while the single-agent baseline achieved 66.12%.

The multi-agent architecture also demonstrated approximately 80% reduction in RAG queries through shared state information management, partially compensating for increased inter-agent communication overhead. These results suggest that role-separated architectures may provide reliability benefits for safety-critical applications, though with higher computational costs.

Future work will extend evaluation to additional operational scenarios, explore alternative agent configurations with varying role distributions, and conduct formal statistical analysis to further validate the observed performance differences.

## ACKNOWLEDGMENTS

This work was supported by Korea Atomic Energy Research Institute (KAERI) grant No. KAERI-526140-26.

## REFERENCES

[1] International Atomic Energy Agency (IAEA), "Artificial Intelligence for Accelerating Nuclear Applications," IAEA Nuclear Energy Series, Vienna, 2022.

[2] L. Wang et al., "A Survey on Large Language Model based Autonomous Agents," Frontiers of Computer Science, vol. 18, no. 6, 2024.

[3] Y. P. Lee, J. Cha, Y. Yu, and S. G. Kim, "Large Language Model Agent for Nuclear Reactor Operation Assistance," Nuclear Engineering and Technology, vol. 57, p. 103842, 2025.

[4] E. Salas, N. J. Cooke, and M. A. Rosen, "On Teams, Teamwork, and Team Performance," Human Factors, vol. 50, no. 3, pp. 540-547, 2008.

[5] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459-9474, 2020.