Development of an Autonomous Reactor Operation LLM Agent for Real-Time Detection and Proactive Mitigation

Giheon Han [a,b], Yonggyun Yu [a,c], Seung Geun Kim[a,*]
[a]*Applied Artificial Intelligence Section, Korea Atomic Energy Research Institute, Daejeon 34057, Republic of Korea*
[b]*Korea University of Technology and Education (KOREATECH), Cheonan 31253, Republic of Korea*
[c]*University of Science and Technology, 217, Gajeong-ro, Yuseong-gu, Daejeon, 34113, Republic of Korea*

[*]*Corresponding author: sgkim92@kaeri.re.kr*

***Keywords :** SMR, iPWR simulation, Autonomous Operation, Anomaly Detection, Multi-Agent Orchestration

1. Introduction

Small Modular Reactors are gaining momentum, and fleet-scale deployment will fundamentally change day-to-day operations [1]. As fleets grow, the burden on human operators increases; in centralized control rooms, a single team may be responsible for multiple reactors simultaneously. In such settings, each reactor should be able to assess its own state, operate within normal bounds, and report concise summaries with recommended actions to humans. This motivates a system in which software agents autonomously monitor plant signals and prepare mitigation while keeping humans in the loop for oversight.

Previous work at Korea Atomic Energy Research Institute (KAERI) demonstrated a natural-language control agent that translated operator intents into procedure-level actions on an integral pressurized water reactor (iPWR) simulator, assisting operators by turning natural-language commands into executable steps [2]. However, the agent was demand-driven: it ran only when the operator issued a query and therefore could not initiate detection or raise an alert if signals became abnormal without a preceding command. Planning and execution were coupled, post-execution verification was minimal, and overlapping conditions were not triaged. These limitations affect scalability and workload.

In this paper, we move from operator-initiated assistance to autonomous monitoring and proactive plan initiation. A real-time detection and reporting layer watches key channels (e.g., neutron/thermal power and Reactor Coolant System (RCS) pressure/temperature), converts rule/threshold hits into structured alerts, summarizes the situation, and proposes a control plan. Execution remains operator-supervised: the agent requests explicit approval, applies the planned steps only after approval [3], and then rereads key signals to verify closure. Internally, supervisor, planner, approval router, and worker roles are separated to improve auditability and maintain clear authority boundaries.

This shift differentiates our approach from prior KAERI work and related language-only assistants. The agent no longer waits for a human prompt; it initiates monitoring, incident summarization, and mitigation proposals on its own while preserving operator authority over actuation. The resulting workflow is simple and scalable for multi-reactor oversight: detect anomalies continuously, propose a plan proactively, and execute only after operator approval, with traceable records of what was proposed, approved, and done.

2. Agent System Development for Real-Time Anomaly Detection and Mitigation

2.1 Architecture and Overall Flow

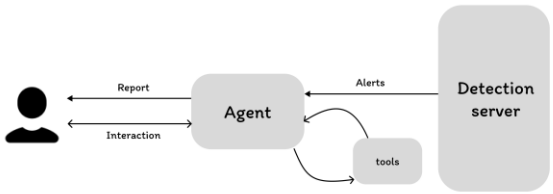

Fig. 1. Overall workflow of the autonomous operation agent

Figure 1 shows the overall flow. In this paper, we refer to the anomaly-monitoring module as Detection Server (hereafter, “Detection server”). The Detection server continuously collects and evaluates key parameters (e.g., reactor power, control-rod position, RCS pressure/temperature). When predefined rules or a condition-diagnosis model determine an abnormal state, the server immediately notifies the agent.

Upon receiving an alert, the agent concisely summarizes the current issue and its evidence (impacts, relevant signals, thresholds) for the operator and simultaneously proposes a remedial procedure. It then asks, via a Human-In-The-Loop approval step (the agent cannot actuate without explicit operator approval), whether to apply the plan; if approved, the agent invokes registered tool modules to issue commands to the simulator/control system (e.g., control-rod movement, setpoint adjustment). After execution, it verifies the effect and reports a brief summary. The operator may also interact with the agent at any time in natural language to issue commands, and the agent acts accordingly. The current prototype prioritizes a single

incident; while an action is in progress, processing of new alerts may be delayed or missed. Accordingly, we describe the workflow under a single-event assumption.

2.2 Agent Implementation

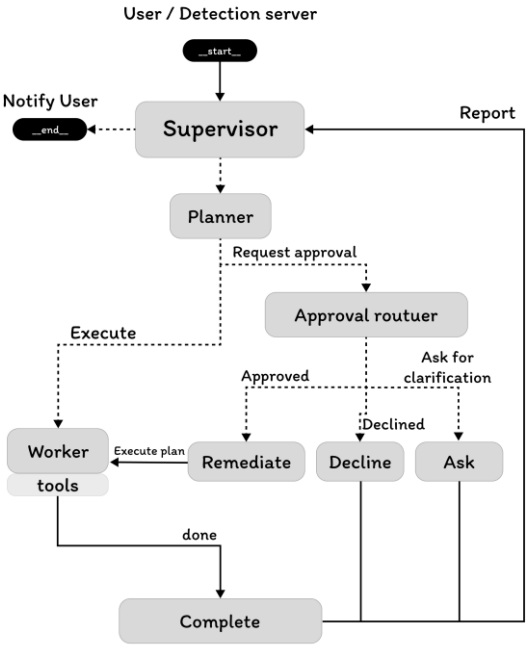

Fig. 2. Agent workflow and roles (Supervisor, Planner, Approval Router, Worker).

The proposed agent distinguishes operator queries from Detection server alarms and handles alarms through a simple report, approval, and action workflow. First, it identifies the input source. If the input is a Detection server alarm, the agent presents a concise report to the operator before any execution. The report summarizes the issue and proposes steps to fix it. Second, only after the operator approves the summary and proposed actions does the agent invoke the registered tools to perform the required control actions (for example, control-rod movement and setpoint adjustments). No execution occurs before approval.

The agent runs as a multi-agent system (Fig. 2). A Supervisor oversees each turn and decides whether the agent should produce a plan or end the turn. When planning is needed, the Planner writes the plan for the operator; it does not call tools. It produces a clear problem summary and step-by-step actions. After the operator reviews the plan, the Approval Router interprets the reply and routes the next step: execute if approved, hold if declined, or ask for clarification if the reply is unclear. When approval is given, the Worker carries out the plan by calling the registered tools and applying the actions to the simulator. These roles work together so that only approved plans are executed and the decision flow is easy to follow.

We apply explicit prompts at three points: Supervisor, Planner, and Approval Router, while execution is performed by the Worker. The Supervisor selects the next actor from {Planner, FINISH} via a structured route response. The Planner does not call tools and outputs a typed JSON plan with fields: summary, actions (list of objects with keys “action”, “params”, and “rationale”), verification, risks, and rollback. The Approval Router classifies operator replies into {REMEDIATE, DECLINE, ASK} and gates execution accordingly. The Worker calls only registered tools and runs only after explicit approval. If an alert originates from the Detection server (sender “DetectionServer”) or the incident flag is set, the Planner routes the plan to the Approval Router. Otherwise, the plan proceeds directly to the Worker.

2.3 Detection Server Implementation

The Detection server runs outside the agent and creates alerts based on a small set of predefined rules. At present it uses simple triggers: if a control-rod position falls below a preset limit, or if reactor power exceeds a preset limit, an alert is generated. These limits can be adjusted for each scenario or operating condition, and the rule set can be extended as needed. Each alert includes basic context such as which rule fired, the signal involved, and the current value versus the threshold. The alert is delivered to the agent as a message whose sender name is “DetectionServer”; the state flag “is_incident_active” may also be set to mark the turn as an incident. On receipt, the agent switches to the report, approval, and action path, and it does not execute any control action until the operator approves.

3. Experimental Setup

3.1 Execution Environment

This evaluation is conducted in an iPWR simulator under a single-incident assumption. The Detection server generates alerts using simple predefined rules, and the agent distinguishes alerts from operator queries and follows a report, approval, and action sequence.

The agent runs on LangGraph with a fixed thread_id (ipwr_thread_id) and a recursion_limit of 50. The LLM is set to temperature 0 for deterministic behavior. The Planner does not call tools and produces only a plan in JSON format. The Worker calls tools registered in the TOOL_REGISTRY to execute approved plans. The Approval Router classifies operator responses as approve, decline, or question. After execution, the CompleteIncident stage resets the incident state.

Although the architecture is designed to run on-prem, experiments used GPT-4o (OpenAI) connected via a cloud API.

3.2 Detection Server Rules and Alert Format

The Detection server uses two basic rules. It generates an alert when a control-rod position drops

below a preset lower limit, and it generates an alert when reactor power exceeds a preset upper limit. These thresholds can be adjusted according to the scenario. Alerts are delivered as messages with the sender name set to "DetectionServer". The alert also activates the incident-response mode immediately by setting the "is_incident_active" flag. When the agent receives an alert, it presents a brief report and an approval request before any execution.

3.3 Scenarios

We used a simple rule-based detector: an alert fires when rod C ≤ 70 steps or neutron power ≥ 100%. In each scenario the operator manually imposed the condition and we observed the agent's response.

S1 Control-rod deviation (manual): Lower rod C to 69 steps. Confirm the alert, see a brief report and approval request, no action before approval, then return rod C to 80 steps and close the incident.

S2 Power overshoot (manual): Raise neutron power to 102%. Confirm the alert and pre-execution approval step, then reduce power stepwise toward about 90% and close the incident.

4. Case Study Results

4.1 S1 Control-rod deviation

By manually lowering rod C to 69 steps, an alarm occurred as shown in Fig. 3, and the agent requested operator approval. After approval, the agent verified that the rod control mode was manual and then drove rod C back up to 80 steps. As seen in Fig. 4, the agent reported successful completion and incident closure.

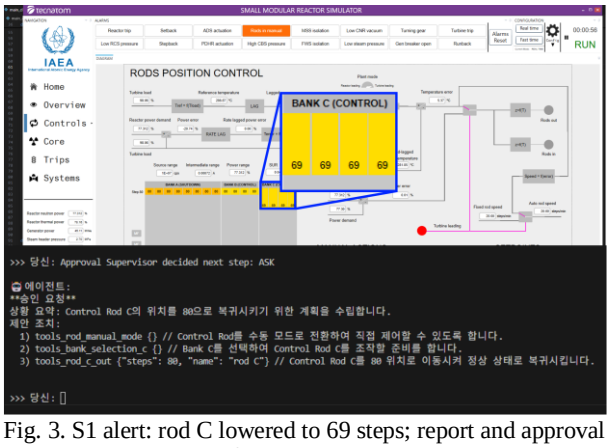

Fig. 3. S1 alert: rod C lowered to 69 steps; report and approval shown; no execution before approval.

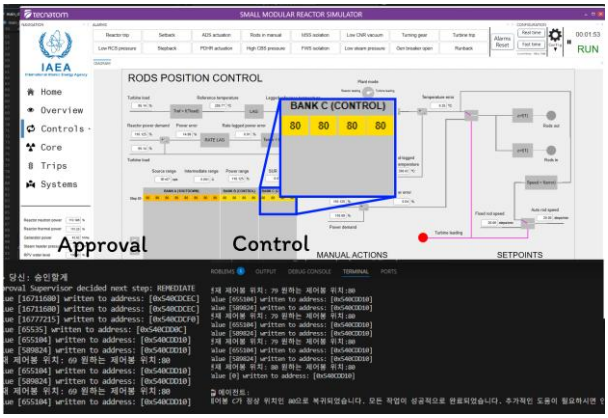

Fig. 4. S1 recovery: after approval, rod C restored to 80 steps; incident closed.

4.2 S2 Power overshoot

By raising neutron power to 102%, an alarm was triggered and the agent presented an approval request, as in Fig. 5. After approval, the agent set the required modes and reduced power stepwise. Specifically, the plan switched the reactor to leading mode, kept the rods in automatic, and set the Reactor Power Demand to approximately 90%. The agent then executed these steps. Fig. 6 shows the plan and the outcome, with the incident closed.

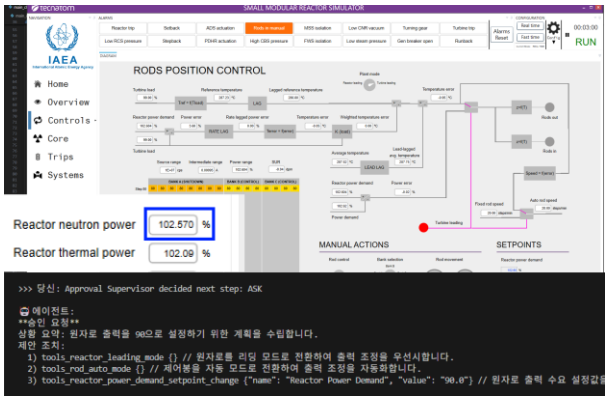

Fig. 5. S2 alert: neutron power at 102% (≥100%); report and approval request shown.

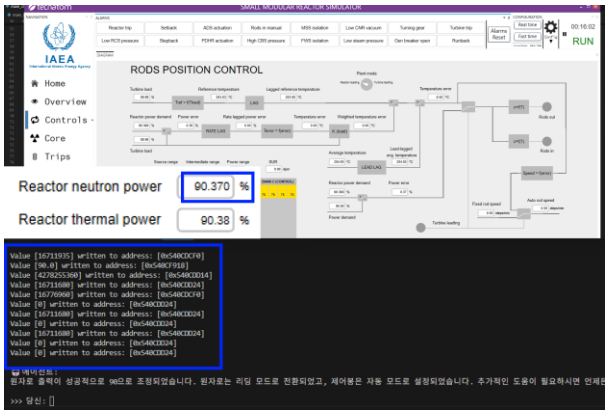

Fig. 6. S2 execution: after approval, agent reduced reactor power to 90%; incident closed.

4.3 Repetitive experiments and success rates

We repeated S1 and S2 ten times each, and both scenarios achieved a success rate of 1.0 (10/10).

Table I: Scenario definitions and outcome metrics

Scenario	S1	S2
Runs	10	10
Alerts raised	10	10
Report shown	10	10
Pre-approval violations	0	0
Post-approval action	10	10
Success	10	10

Success was counted when all of the following were satisfied: an alert was raised, a pre-execution report was shown, there were zero violations of the no-execution-before-approval rule, the planned actions were executed after approval, and the incident was closed.

5. Discussion and Limitations

In this case study, the agent consistently followed a report, approval, and action flow under simple rule-based detection. In S1 and S2, when an alert occurred, the agent showed a situation summary and an approval request before any execution, and no action was taken before approval. Only after approval were the planned steps applied, and the incident state was cleared. Messages and logs recorded the plan, the approval prompt, the routing result, and the execution report, enabling traceability. The plans reflected the rules to return rod C to 80 steps and to reduce power stepwise toward about 90%, and repeated runs showed a success rate of 1.0.

The current implementation has the following limitations. First, it assumes single-incident priority. If a new alert arrives during execution, handling may be delayed or deferred to the next turn. Second, the Detection server relies on simple threshold rules such as rod C $\leq$ 70 steps and neutron power $\geq$ 100%, which may miss combined patterns or noisy edge cases. Third, although the plan object includes a verification field, there is no dedicated post-execution verification stage that re-reads values and automatically determines pass or fail; the flow currently moves from Worker directly to Complete. Fourth, all tests were performed in a simulator, so there may be differences from a real plant in hold logic, interlocks, and delays.

To address these limitations, we will handle concurrent alerts by assigning priorities and using preemption and scheduling so new alarms are processed without blocking. We will improve detection by adding a deep-learning–based state estimator that is more accurate than simple rules. After each action, the agent will recheck key signals to confirm that the target has been achieved and will mark pass or fail, with rollback or escalation if a check fails. We will also enforce tool-level access controls, separating tools that require reporting and explicit operator approval from low-risk utilities the agent may use autonomously. For predefined low-risk situations, fully automatic actions may be allowed under safeguards and auditing.

6. Conclusions

We implemented an agent for an iPWR simulator and confirmed that a report, approval, and action workflow operate reliably under simple rule-based alerts. The two scenarios (S1: rod C $\leq$ 70 steps, S2: neutron power $\geq$ 100%) were manually induced by the operator. When a condition occurred, the agent detected the problem and, before the operator took any action, presented a report and waited for approval. It applied the planned actions only after approval. In repeated runs, both scenarios achieved a success rate of 1.0 (10/10). These results suggest a practical starting point that standardizes alarm response and reduces time while maintaining an operator-centered approval principle.

In the future, we will extend from a single-reactor assistant to fleet-level operations so that one control-room team can safely supervise multiple reactors. Each reactor will host a local agent with its Detection server to monitor signals and propose a plan, while a Fleet Supervisor will coordinate across units, maintain a shared situational picture, prioritize overlapping alerts, de-conflict operator attention, and stage approvals so that no unit acts before its plan is explicitly approved. We will validate this concept by running multiple iPWR simulators on separate machines and exercising cross-unit scenarios such as simultaneous power excursions and rod-position deviations; evaluation will report operator workload, time-to-first-action per unit, scheduling effectiveness, and fleet-level incident completion.

ACKNOWLEDGEMENTS

This work was supported in part by Korea Atomic Energy Research Institute R&D Program under Grant KAERI-524540-25.

REFERENCES

[1] International Atomic Energy Agency, Small Modular Reactors: Advances in SMR Developments, 2024 Edition, Vienna, 2024.

[2] Lee, Y. P., Cha, J., Yu, Y., & Kim, S. G. (2025). Large Language Model Agent for Nuclear Reactor Operation Assistance. Nuclear Engineering and Technology, 103842.
[3] U.S. Nuclear Regulatory Commission, NUREG-0711: Human Factors Engineering Program Review Model, Rev. 3, 2012.