LLM-based multi-agent system for autonomous maintenance process of machine tools

Jongsu Park[1], Seongwoo Cho[1], Sena Nur Durgunlu[2], Yoonji Chae[3], and Jumyung Um[1,2,3]

[1] *Department of Artificial Intelligence, Kyung Hee University,*
1732, Deogyeong-daero, Yongin-si, 17104, Gyeonggi-do, Republic of Korea
whdtn95@khu.ac.kr
whtjddn1564@khu.ac.kr
jayum@khu.ac.kr

[2] *Department of Industrial and Systems Management Engineering, Kyung Hee University,*
1732, Deogyeong-daero, Yongin-si, 17104, Gyeonggi-do, Republic of Korea
senanurdurgunlu@khu.ac.kr

[3] *Department of Metaverse, Kyung Hee University,*
1732, Deogyeong-daero, Yongin-si, 17104, Gyeonggi-do, Republic of Korea
cynj20@khu.ac.kr

Abstract

The rapid improvements of modern machine tools have became new challenge of traditional maintenance operators. While machine manuals are growing with new modules like robot or pallet system, understanding and acting on these instructions of all facilities remains a challenge for operators. This paper proposes a large language model based multi-agent system to ask human operator's need, to search manuals about the given error code, and to suggest maintenance procedures. These sequences include action, user interface target, preconditions, and expected outcomes, and are executed by agent capable of interacting with Human–Machine Interfaces. The multi-agent system is comprising four agents: a chatbot, solution_finder, actor, and supervisor. Each agent operates based on role-specific prompts that define their responsibilities and decision rules. Instead of relying on predefined rule sets, the system interprets unfamiliar or previously unseen alarms by reasoning over machine manuals and context, enabling flexible and scalable maintenance. The system was implemented on a commercial system of CNC machine tools and successfully performed automatic responses to selected alarms.

1. Introduction

Appropriate maintenance is critical in manufacturing companies, where unexpected machine failures lead to significant losses. While some maintenance tasks require technical expertise, a significant number of minor issues can be addressed directly via Human–Machine Interfaces (HMIs) by adjusting parameters or executing simple commands. In these cases, the issue is resolved by conducting troubleshooting according to error codes in the manual.

But understanding and following these instructions can be difficult for unskilled workers especially. For instance, when alarm message shows "G28 FOUND IN SEQUENCE", the troubleshooting instruction is "Perform the reference position return". Operators must interpret what each terminology means (ex. G28 code or SEQUENCE), and also need to know how return the reference position of each axis. Because advanced machine tools have been fully customized, in particular, a single human operator is not possible to understand all terminologies and all operation methods of many machine manuals in a short time. This knowledge gap requires for someone who assist and guide step-by-step instruction suitable to unskilled workers.

Recent advancements in Large Language Models (LLMs) provide new opportunities for automating the interpretation and execution of industrial maintenance tasks. LLMs can understand unstructured text, including technical manuals and error logs, and generate contextually appropriate responses. With the integration of LLM agents and tools, their applications now extend beyond question answering to include task execution such as code generation and robot control.

This paper presents a multi-agent framework powered by an LLM to support autonomous maintenance of industrial equipment. The system processes user inputs, including error codes, identifies corresponding procedures from manuals, and converts them into executable action sequences. By recognizing the positions of User Interface (UI) components on the HMI and executing each step, the system achieves autonomous maintenance.

The framework consists of four agents: a chatbot agent that communicates with the user to understand the task, a solution finder agent that searches the manuals for appropriate procedures and converts them into action steps, an actor agent that performs HMI-level operations, and a supervisor agent that coordinates the entire workflow. The system can handle unfamiliar or previously unseen alarms by reasoning over textual documentation, enabling general-purpose maintenance without relying on strictly predefined rules.

The proposed approach was implemented on the HMI of a Computerized Numerical Control (CNC) machine tool. The system successfully responded to operational alarms by retrieving procedures and executing the necessary actions. These results demonstrate the potential of LLM-based prompt-driven maintenance systems for use in real industrial environments. Unlike ChatGPT with external servers, the proposed framework employs an on-premise LLM to ensure full data confidentiality within industrial facilities. This local deployment prevents sensitive maintenance data from being transmitted outside the factory network, and enables real-time reasoning and control in latency-critical environments. The main contributions of this paper are as follows:

1. Manual-driven troubleshooting interpretation: We propose a method that leverages LLMs to interpret error codes and textual troubleshooting instructions from machine tool manuals, thereby bridging the knowledge gap for unskilled operators.

2. On-premises deployment for industrial security: We ensure data confidentiality and low-latency reasoning by deploying a fully local LLM within the factory network, addressing the security and privacy concerns inherent to cloud-based solutions.

3. Multi-agent framework for autonomous maintenance: We design a multi-agent architecture that coordinates user interaction, retrieval of manual-based procedures, task planning, and execution, enabling scalable and flexible automation.

4. Executable HMI-level actions: We demonstrate automated maintenance through direct GUI manipulations on real HMIs, showing how high-level reasoning can be grounded into executable actions.

5. End-to-end system implementation and validation: We implement the proposed framework on a CNC machine tool HMI and validate its effectiveness by successfully responding to selected alarms, thereby showcasing the feasibility of LLM-driven prompt-based maintenance in real industrial environments.

The remainder of this paper is organized as follows: Section 2 reviews related works of LLM based maintenance system. Section 3 describes the system architecture composed of four agents. Section 4 presents the experimental results of the prototype. Section 5 discusses the advantages and limitations. Section 6 concludes this paper with future directions.

2. RELATED WORK

In industrial environments, maintenance is essential for maximizing productivity and reducing lead time. For this reason, recent studies have explored a variety of approaches to predictive and preventive maintenance. CNN–LSTM hybrid models integrating vibration and temperature sensor data have been used to predict component failures days in advance (Garcia, Rios-Colque, Peña, & Rojas, 2025). To address real-time adaptability, IoT data streams have been leveraged to dynamically prioritize maintenance tasks (Pinciroli, Baraldi, & Zio, 2023). A preventive maintenance framework based on a modified TPM methodology was implemented in an automotive assembly line and reduced equipment failure rates by 15% (Hardt, Kotyrba, Volna, & Jarusek, 2021). Efforts have also been made to develop containerized solutions. For example, four primary stages of software maintenance, namely image detection, scheduling, security measures, and evaluation, have been compared (Malhotra, Bansal, & Kessentini, 2024).

However, most of these approaches focus on prediction or detection. This means that even if a fault is anticipated or detected, human intervention is still required to carry out the actual corrective actions, leading to delays and potential downtime. Moreover, they require large amounts of historical sensor data and domain-specific model training, and depend on customized IoT infrastructure, which limits scalability and delays deployment. To address these limitations, there is a need for a system capable of interpreting maintenance instructions and executing them autonomously.

Unlike traditional rule-based or statistical models, LLMs have been noted for their flexible reasoning and natural language understanding, enabling adaptive operation in heterogeneous and dynamic environments (He, Treude, & Lo, 2025). The integration of LLMs into industry has been recognized as a promising approach for enhancing decision-making and supporting human–machine collaboration (Shao, Basit, Karri, & Shafique, 2024). Research has also been conducted on tuning LLM models to effectively understand machine tool equipment manuals and provide users with targeted assistance based on that understanding (Cho, Park, & Um, 2024).

In industrial contexts, most LLM applications have remained at the chatbot level, focusing primarily on natural language understanding. Only recently has research advanced toward using LLMs as active agents to support various industrial functions.

Intelligent agents are autonomous entities that perceive their environment, make decisions, and perform actions to achieve objectives. While single-agent systems have been valued for their simplicity, they face scalability limitations. Multi-agent systems overcome these constraints by distributing responsibilities among specialized agents, enabling parallelism, specialization, and fault tolerance. Several notable studies illustrate these advancements.

An LLM-based industrial automation multi-agent system with validator and reprompting agents was developed to improve control reliability and safety (Vyas & Mercangöz, 2024). A multi-agent framework combining mechanical, electronics, control, and software engineering agents was designed to produce functional prototypes with minimal human intervention (Wang et al., 2025). The LLM-Agent-Controller was introduced as a domain-specific framework in which a supervisor coordinated specialized agents for controller design, modeling, analysis, and simulation (Zahedifar, Mirghasemi, Baghshah, & Taheri, 2025).

MASDebugFW presented an LLM-assisted debugging environment for multi-agent industrial control systems, enabling real-time inspection of agent states, condition-based breakpoints, and stepwise execution in model-driven simulations (Tezel & Kardas, 2025). DeCoAgent (Jin, Ye, Lee, & Qiao, 2024) employed LLMs within a decentralized framework based on blockchain smart contracts, allowing secure and trustless cooperation between industrial agents.

Collectively, these works demonstrate the flexibility, scalability, and adaptability of LLM-based multi agents system, while also emphasizing the need to address safety, large-scale integration, and robustness for industrial deployment. Building upon this foundation, this paper proposes a practical framework that uses a local LLM-based multi agents system to secure the private data, while automatically executing maintenance cases on the HMI.

While recent studies have demonstrated the potential of LLM-driven multi-agent frameworks for industrial control and automation, most of these systems depend on cloud-hosted models or simulated environments. For instance, (Vyas & Mercangöz, 2024) relied on external APIs to coordinate agentic control tasks, and (Zahedifar et al., 2025) focused on conceptual controller design rather than execution on physical HMIs. Similarly, DeCoAgent (Jin et al., 2024) emphasized decentralized cooperation but did not address security constraints in factory networks. In contrast, the present work deploys an entirely on-premise LLM that performs retrieval, reasoning, and control within a closed industrial network. This local deployment ensures data confidentiality, low latency, and direct interaction with real HMI interfaces, representing a practical step toward secure and fully autonomous maintenance. In other words, unlike cloud-dependent or simulation-only approaches, this paper highlights a factory-ready, secure, and fully local multi-agent system where LLM-driven reasoning directly translates into executable maintenance actions.

3. SYSTEM FRAMEWORK

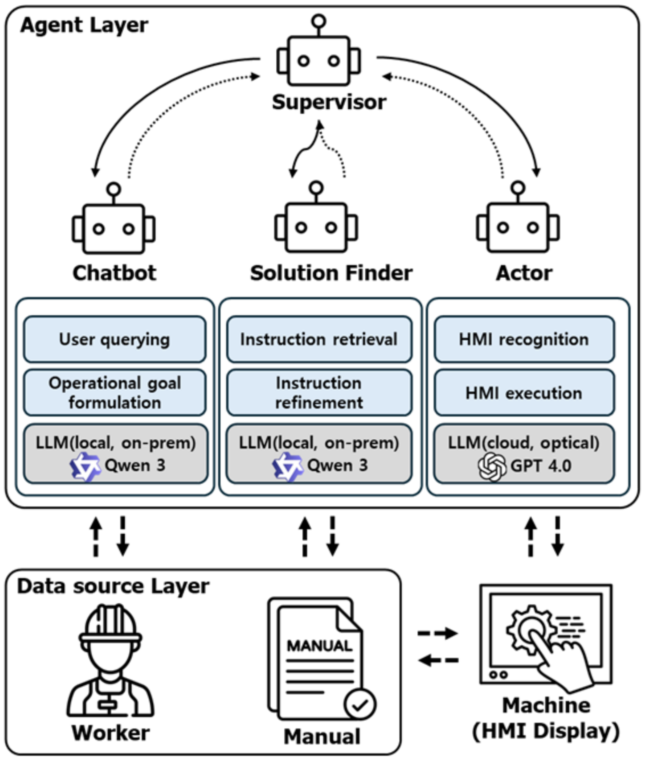


Figure 1. An architecture of the proposed system.

This paper proposes an autonomous maintenance system that is based on a multi-agent architecture with a LLM at its core. The system supports industrial equipment maintenance, reduces operator workload, and improves productivity.

The intended application involves maintenance tasks that can be performed through basic HMI operations, for example parameter adjustments or mode changes. To resolve an alarm or error without operator intervention, the problem context is interpreted, the correct procedure is identified from machine documentation, and the required actions are executed through

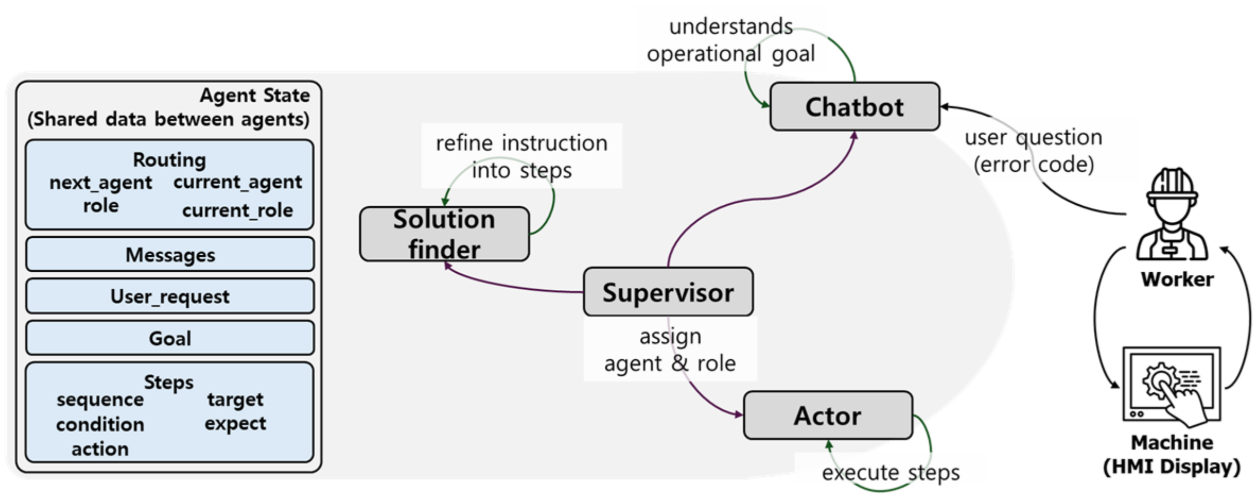


Figure 2. Agent State and the data flow of the proposed system.

the HMI. By following official machine manuals, consistent execution is ensured and maintenance information remains accessible.

The proposed architecture is independent of specific equipment types, which allows deployment across various machines. To meet security requirements in industrial environments, a local LLM and Retrieval-Augmented Generation (RAG) techniques are employed so that sensitive data is retained within the facility(Lewis et al., 2020).

Figure1 presents the overall control flow, showing how the Supervisor agent coordinates the workflow by activating the Chatbot, Solution Finder, and Actor agents as required. The multi-agent framework provides modularity, flexibility, and robustness, and the distribution of responsibilities across agents for user interaction, documentation retrieval, action execution, and workflow supervision enables efficient operation, adaptability to different configurations and error conditions, and effective use of LLM capabilities through role-specific configurations.

3.1. Role of Agents

The following section describes the role of each agent in the proposed system in detail. As shown in Figure1, the architecture consists of four primary agents: Supervisor, Chatbot, Solution Finder, and Actor. Dividing the task among different agents allows the system to handle complex tasks in an organized and dependable way.

The Supervisor manages the overall workflow, acting as the central decision-maker of the system. It reads the current state and recent messages, determines the next agent to act, and assigns a concise role description. The Supervisor ensures that the process follows the intended sequence: initiating the conversation through the Chatbot, retrieving and structuring the solution via the Solution Finder, executing the procedure with the Actor, and returning to the Chatbot for user feedback. It focuses solely on process coordination, error handling, and ensuring task completion.

The Chatbot serves as the direct interface between the user and the system, beginning by asking the user for their maintenance request and collecting any necessary details such as tool names or parameter values. It delivers responses from the other agents and requests feedback to confirm whether the task has been completed to the user's satisfaction. Beyond simply relaying messages, the Chatbot interprets the content of the conversation to identify the maintenance problem that must be addressed. For example, if a user states, "Error code 1 occurred, please help." the Chatbot understands this as a requirement to resolve the specific fault described by "Error code 1" and frames it as a clear operational goal. This goal is then forwarded to the Solution Finder, ensuring that subsequent steps are directly aligned with the user's actual need.

The Solution Finder uses the problem definition provided by the Chatbot to locate the exact maintenance instructions within the machine's documentation. This search is carried out through a Retrieval-Augmented Generation (RAG) process, which enables direct access to the most relevant sections of large manuals.

By using RAG, the system can have several advantages. It allows the underlying documentation to be changed and managed easily for each specific machine, which makes adaptation simpler. It reduces the need for fine-tuning or building large datasets, since the language model can be used without additional training. It keeps all base documentation stored locally, which ensures data security. It also performs a retrieval step before answer generation, helping to minimize hallucinations and improve the reliability of the output.

Once the instructions are found, the Solution Finder refines them into a set of atomic actions that can be directly executed on the HMI. Each action is described in a structured format with one discrete operation per step, including clear preconditions, targets, and expected outcomes. By converting technical instructions into an execution plan, the Solution Finder eliminates interpretation errors and ensures the Actor can carry out the procedure reliably.

The Actor executes the maintenance procedure on the HMI according to the executable action list provided by the Solution Finder. It interacts with the machine interface by performing actions consists of clicking UI in the HMI, and entering text values into fields. The Actor ensures that each action is carried out in the correct order and verifies that the expected outcome of each step is achieved before proceeding. This execution layer connects textual instructions with direct system operation, enabling automatic maintenance without human intervention.

3.2. Agent Workflow

Figure2 presents the data flow among the agents, showing how information moves through the system. The exchange of data is managed through a predefined agent state.

In a multi-agent framework, the agent state is an essential design element that serves as shared memory. It allows agents to work independently while keeping a unified view of the task. By storing the current status, goal, interaction history, and execution details, it supports integration and tracking of work. It functions like an observation in reinforcement learning.

In this system, the agent state contains messages, user request, goal, steps, routing, and information about the last agent and its role. Messages are updated as the process moves through each agent, allowing the system to track progress. The user request is the initial question provided by the user, and the goal is the operational objective derived from that request. These elements guide the flow so that agents maintain focus on the defined task. The agent state also stores routing details for the next agent, including its role and whether user input is needed. Steps are an executable list of actions. Each step contains a sequence, which is a natural language description of the instruction. A condition describes the requirement before executing the step. An action is a single atomic operation such as clicking a UI element or entering text. A target is the UI element or textbox to be acted upon, with possible details like name, type, optional user input, and coordinates. An expected outcome explains the result if the step is performed correctly.

The Supervisor reads the agent state both at the initial run and after each agent has completed its role, then sets routing according to the current state. For example, at the start, to identify the operational goal, the Supervisor routes to the Chatbot. Once the operational goal is clearly defined, the Supervisor routes to the Solution Finder to obtain the relevant instruction information. The Chatbot engages with the user, records the request in the state, and determines the operational goal from the dialogue. The Solution Finder retrieves the exact instruction from the machine documentation that matches the goal. The retrieved instruction and its file path are then provided to the LLM model within the Solution Finder. Then this content is transformed into a sequence of atomic actions that can be executed on the HMI, with each step containing a single operation, condition, target, and expected result. When the step list is completed, routing is updated to the Actor for execution. The Actor reads the step list from the state and performs the specified actions on the HMI. If text input is required, the Actor selects the textbox, requests the value from the user, and enters it. If a click is needed, the LLM model in the Actor uses the target name to find the x and y coordinates in the HMI and then clicks the coordinates. After all steps are completed, the Supervisor sets routing back to the Chatbot to report the results. The Chatbot informs the user of the outcome. If the task is complete, the Supervisor routes to finish. If further assistance is needed, the goal is updated and a new cycle starts.

Each agent functions as an independent module, using the provided agent state information and updating it as needed. This modular design allows any single agent to be replaced or modified without changing the rest of the system. For example, the Actor agent in the current implementation can be replaced with an Actor agent with tailored input methods for the specific environment, improving usability and accuracy. Alternatively, instead of an Actor agent, another agent who delivers the step list through an Augmented Reality interface could be introduced, enabling use as a co-pilot system. This independence of each agent ensures that the architecture remains versatile and applicable across diverse industrial environments.•

4. IMPLEMENTATION & EXPERIMENT

This section details the implementation of the proposed multi-agent LLM framework and the experiments conducted to assess its effectiveness in HMI-driven maintenance tasks. We first summarize the technology stack and runtime environment used to build the system. We then describe the end-to-end agent workflow as it operates in practice from task intake to HMI execution. Finally, we present three evaluations that measure (i) the stability and adaptability of the multi-agent architecture, (ii) quantitative performance across repeated runs, and (iii) the effectiveness of transforming manual instructions into an executable action list.

Technology stack and runtime environment. The system is implemented with the following components:

- **Large Language Models:**

  - Qwen3(Yang et al., 2025): local on-premise model inference to meet security requirements.
  - GPT4.0(Achiam et al., 2023): multi-modal model working on cloud for UI component detection only.
- **HMI control:** PyAutoGUI and PyWinAuto for programmatic UI interaction on CNC HMI screens.
- **LLM usage and RAG pipeline:** A LangChain framework was used to construct retrieval workflow (document loading, chunking, vector retrieval) and LLM usage.
- **Visualization and logging:** UI screenshots and step-by-step execution traces for debugging; per-run logs for timing and success-rate analysis.

Building on this foundation, the system operates through a coordinated sequence of agents. Each agent performs a dedicated role in the maintenance workflow, enabling the process to progress smoothly from initial problem intake to final execution on the HMI.

For experimental validation, a custom error code and corresponding manual was developed. The selected test case involves an error code triggered when attempting to run a program without a registered program on the HMI, requiring the registration of a new program. The HMI platform used in this implementation was SketchTurn (DNsolution, Korea).

Chatbot Agent: The Chatbot agent acts as the primary interface with the user, explicitly prompting for CNC or machine tool–related issues (e.g., *"How can I assist you with your CNC or machine tool?"*). It collects relevant details such as alarm codes, machine parameters, or contextual descriptions, and reformulates them into a concise operational goal stored in the agent state. Upon system initialization, the user is prompted accordingly, and once the request is received (e.g., "Alarm A250625 occurred. Help me."), the Chatbot records it in the user_request field and, using its LLM model, transforms it into a clear operational goal (e.g., "Resolve the alarm A250625"). This goal serves as the foundation for subsequent problem-solving by other agents.

Example (from experiment)

- **User request:**
  "Alarm No. A250625 occurred, help me"
- **Reformulated operational goal:**
  "Resolve the alarm No. A250625".
- **Output:**
  Updated agentstate with user_request and operational goal.

Solution Finder Agent: Once the operational goal is stored in the agent state, the Solution Finder agent's LLM model searches the machine manual via a RAG pipeline to locate the most relevant instruction. To minimize hallucinations in an industrial context, the retrieved content is returned verbatim, without modification. The instruction is then refined into an executable action list, where each step specifies a precondition, a single atomic operation, the UI element to interact with, and the expected outcome. This structured format ensures that the Actor can execute the procedure deterministically. One of the critical evaluation points in this paper was the transformation of textual instructions into fully atomic, execution-ready commands. This step ensures that the Actor agent can directly perform each operation without additional interpretation, enabling seamless automation.

Example (from experiment): as shown in Figure 3

- **Retrieved procedure:**
  Navigate to Program Manager $\rightarrow$ Create new program $\rightarrow$ Confirm.
- **Generated steps:**
  Atomic actions (e.g., click `Program Manager` button, enter program name, enter description, confirm with `OK`).
- **Output:**
  Updated agentstate with action list including conditions, actions, targets, and expected outcomes.

Actor Agent: The Actor agent executes the action list step-by-step on the HMI, interacting with UI components. The agent identifies each UI component and manipulates them by entering parameters or pushing buttons. UI component recognition is conducted through prompt-based multimodal LLM processing. This enables identification of UI components from the HMI screenshot. And using LLM based recognition is easily adapted to diverse HMI configurations to ensure compatibility and generalizability. While most modules run on a secure on-premise local LLM to protect sensitive maintenance data, cloud LLM 'GPT-4.0' is selectively employed for UI recognition to maximize detection accuracy. To minimize shared data in the public cloud, the proposed system sends only screenshot image of current HMI to GPT-4.0 cloud server, ensuring strict information security.

Example (from experiment)

- Clicked `Program Manager` and `Program New`.
- Entered program name: "Test Program".
- Entered description: "for test".
- Confirmed with `OK` and pressed `ENTER`.
- Returned to main screen and checked program settings.
- **Outcome:**
  Alarm A250625 cleared successfully.

Supervisor Agent: The Supervisor agent oversees the entire workflow by interpreting the current agentstate and determining the next agent to execute. When a maintenance request is initiated, it assigns the Chatbot as the first agent to gather detailed input from the user. After each agent completes its task, the Supervisor updates the agentstate and routes control accordingly. The workflow follows a clear sequence: the Chatbot receives the user's request, the Solution Finder generates a structured solution, and the Actor executes the plan, with the Supervisor coordinating each step based on the evolving agent state.

Example (from experiment)

- **Output:**
  Next agent → `Chatbot`
  Role → `ask user what they need help with`
  Require user input → `True`

To evaluate the computational efficiency of the proposed multi-agent system, the execution time of each node—*Supervisor*, *Chatbot*, *Solution Finder*, and *Actor*—was measured individually 30 independent runs, and the average execution time was calculated. In addition, the total execution time for the complete workflow, in which all nodes are executed sequentially, was measured under the same conditions. Table 1 summarizes the results. The results indicate that while the *Solution Finder* node exhibits the highest average execution time due to its complex reasoning process, the total runtime for the full workflow remains within acceptable limits for real-time or near-real-time applications. It is worth noting that for the *Chatbot* and *Actor* nodes, the presence of user input stages can cause variation in total execution time, depending on how quickly the user responds.

To assess the functional accuracy of the proposed system, two evaluation metrics were considered: (1) the success rate of generating the correct action sequence for a given task, and (2) the success rate of detecting the correct UI coordinates corresponding to the identified targets. Both metrics were evaluated over a series of test cases representative of realistic industrial HMI operations. As shown in Table 2, the system achieved a 76.6% success rate in sequence generation and a 23.3% success rate in coordinate detection. These results indicate that the system can reliably interpret tasks and interact with HMI elements, although occasional failures in coordinate detection suggest potential improvements in visual element recognition under varying interface conditions.

Table 1. Average execution time (seconds) 30 runs for each node and the entire workflow

Node / Workflow	Average Execution Time (s)
Supervisor Node	0.45
Chatbot Node	1.34
Solution Finder Node	35.15
Actor Node	139.89
Entire Workflow	183.35

Table 2. Success rate of the full system in generating correct sequences and detecting correct UI coordinates

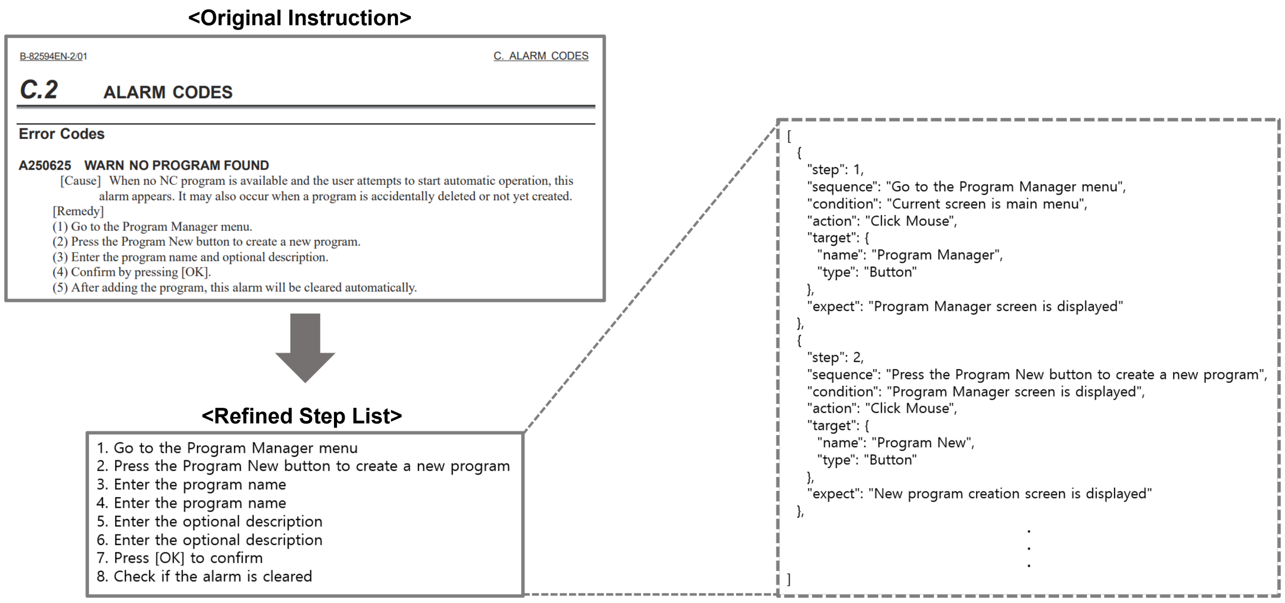
Evaluation Metric	Success Rate (%)
Correct sequence generation	76.6
Correct UI coordinate detection	23.3

5. DISCUSSION

This paper presents an autonomous maintenance framework powered by a multi-agent LLM system and implements it with machine tool HMIs. Unlike traditional approaches where operators must manually perform all actions, the proposed system can identify and execute solutions for simple maintenance scenarios on its own. This reduces downtime, increases productivity, and eases the operator's workload. It is particularly beneficial for inexperienced operators or situations where a single operator must manage multiple machines. In the implementation phase, the proposed multi-node architecture was realized through the integration of supervisor, chatbot, solution finder, and actor nodes. Performance evaluation was conducted in two aspects: computational efficiency and functional accuracy. The average execution time of each node was measured over 30 runs, along with the average time for executing the entire workflow. Results show that while the supervisor and solution finder nodes maintained relatively stable processing times, the chatbot and actor nodes exhibited variations due to the dependency on user input duration. In terms of functional accuracy, the system achieved a 76.6% success rate in generating correct sequences and a n% success rate in detecting correct UI coordinates. These results indicate that the system is both computationally efficient and capable of producing precise, context-aware action plans.

The main advantages of the proposed system are as follows:

- Autonomous problem-solving: The system can indepen-



B-82594EN-2/01 C. ALARM CODES

C.2 ALARM CODES

Error Codes

A250625 WARN NO PROGRAM FOUND

[Cause] When no NC program is available and the user attempts to start automatic operation, this alarm appears. It may also occur when a program is accidentally deleted or not yet created.

[Remedy]
(1) Go to the Program Manager menu.
(2) Press the Program New button to create a new program.
(3) Enter the program name and optional description.
(4) Confirm by pressing [OK].
(5) After adding the program, this alarm will be cleared automatically.

1. Go to the Program Manager menu
2. Press the Program New button to create a new program
3. Enter the program name
4. Enter the program name
5. Enter the optional description
6. Enter the optional description
7. Press [OK] to confirm
8. Check if the alarm is cleared

```
[
  {
    "step": 1,
    "sequence": "Go to the Program Manager menu",
    "condition": "Current screen is main menu",
    "action": "Click Mouse",
    "target": {
      "name": "Program Manager",
      "type": "Button"
    },
    "expect": "Program Manager screen is displayed"
  },
  {
    "step": 2,
    "sequence": "Press the Program New button to create a new program",
    "condition": "Program Manager screen is displayed",
    "action": "Click Mouse",
    "target": {
      "name": "Program New",
      "type": "Button"
    },
    "expect": "New program creation screen is displayed"
  },
                              .
                              .
                              .
]
```

Figure 3. Comparison of the original instruction and the refined step list

dently identify and execute solutions for alarm codes, reducing lead time and easing maintenance workload.

- From manuals to action: It transforms conventional manual instructions into precise, execution-ready steps, allowing the LLM agent to carry them out seamlessly.
- Highly scalable and adaptable: Through RAG and prompt engineering, data can be updated quickly without retraining, enabling smooth adjustments to manuals or operational requirements.
- Security-first design: By running on a local LLM, the system prevents sensitive maintenance information from leaving the facility.
- Modular and customizable: Its multi-agent architecture allows tailored actor agents to boost performance in specific industrial contexts.

Traditional rule-based systems follow predefined rules for specific situations. They work reliably for known scenarios but cannot easily adapt when new alarm codes appear or procedures change. Updating such systems usually requires manual rewriting of rules, which slows response in rapidly changing industrial settings.

In previous work, LLM applications in industrial contexts have generally been limited to interpreting manuals or providing conversational assistance. The system described here extends the role of LLM from an information provider to both a task planner and an executor, enabling a smooth progression from language understanding to HMI operation. The framework performs the full workflow, starting with alarm code identification, then procedure retrieval, UI action mapping, sequence execution, and result reporting. Within this process, the LLM interprets procedures from official manuals, breaks them down into atomic action steps, and translates them into executable HMI operations such as button presses or parameter entries.

The proposed system offers strengths in flexibility, scalability, and modularity. It does not rely on fixed rules, allowing quick adjustment to new errors or changes in maintenance instructions. Its design supports deployment across diverse industrial systems, provided that manuals and HMI configurations are available. The modular architecture, consisting of the Supervisor, Chatbot, Solution Finder, and Actor, enables targeted enhancements and environment-specific customization through the replacement of individual components.

However, there are limitations and areas for future work. Since key functions such as decision-making and UI identification rely heavily on the LLM, system stability can be affected by the model's performance. Adapting to dynamic HMIs requires robust recognition and interaction in environments where UI layouts or element positions change. Preventing hazardous actions necessitates safeguards to intercept and block unsafe operations. Mitigating hallucinations requires verification and filtering mechanisms to ensure only valid, manual-based procedures are executed. Cross-platform validation is also needed to confirm performance across devices with different HMIs, protocols, and documentation structures. Current research includes developing LLM-in-the-loop and hybrid systems combining partial rule-based logic to support LLM performance.

In summary, this work presents a system that can understand maintenance steps and execute them directly through the HMI. This approach addresses the rigidity of older automation methods and moves toward more versatile and intelligent maintenance solutions.

6. CONCLUSION

The proposed LLM-based multi-agent framework demonstrates that autonomous maintenance of machine tools can be achieved by combining natural language understanding with direct HMI operation. Through integration of the Supervisor, Chatbot, Solution Finder, and Actor, the system connects procedural interpretation from manuals to fully executable action sequences. Its successful implementation on a CNC machine tool shows that manual-based procedures can be automated accurately and securely with a local LLM. While further refinement is needed for handling dynamic interfaces and ensuring operational safety, the results highlight the potential of this approach to extend beyond CNC machines to a variety of industrial systems.

ACKNOWLEDGMENT

This study is supported by Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korea Government(MOTIE) (RS-2025-13672968, Development of international standards for digital product passports for vision data-based manufacturing and recycling processes for a circular economy). This work is also supported by Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Ministry of Trade, Industry and Energy (No. 20026431)

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... others (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Cho, S., Park, J., & Um, J. (2024). Development of fine-tuned retrieval augmented language model specialized to manual books on machine tools. *IFAC-PapersOnLine*, *58*(19), 187–192.

Garcia, J., Rios-Colque, L., Peña, A., & Rojas, L. (2025). Condition monitoring and predictive maintenance in industrial equipment: An nlp-assisted review of signal processing, hybrid models, and implementation challenges. *Applied Sciences (2076-3417)*, *15*(10).

Hardt, F., Kotyrba, M., Volna, E., & Jarusek, R. (2021). Innovative approach to preventive maintenance of production equipment based on a modified tpm methodology for industry 4.0. *Applied Sciences*, *11*(15), 6953.

He, J., Treude, C., & Lo, D. (2025). Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, *34*(5), 1–30.

Jin, A., Ye, Y., Lee, B., & Qiao, Y. (2024). Decoagent: Large language model empowered decentralized autonomous collaboration agents based on smart contracts. *IEEE Access*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... others (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, *33*, 9459–9474.

Malhotra, R., Bansal, A., & Kessentini, M. (2024). A systematic literature review on maintenance of software containers. *ACM Computing Surveys*, *56*(8), 1–38.

Pinciroli, L., Baraldi, P., & Zio, E. (2023). Maintenance optimization in industry 4.0. *Reliability Engineering & System Safety*, *234*, 109204.

Shao, M., Basit, A., Karri, R., & Shafique, M. (2024). Survey of different large language model architectures: Trends, benchmarks, and challenges. *IEEE Access*.

Tezel, B. T., & Kardas, G. (2025). Debugging in the domain-specific modeling languages for multi-agent systems. *Journal of Computer Languages*, *83*, 101325.

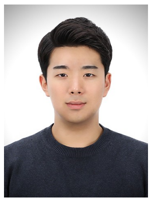
Vyas, J., & Mercangöz, M. (2024). Autonomous industrial control using an agentic framework with large language models. *arXiv preprint arXiv:2411.05904*.

Wang, Z., Lo, F. P. W., Chen, Q., Zhang, Y., Lin, C., Chen, X., ... Lo, B. P. (2025). An llm-enabled multi-agent autonomous mechatronics design framework. In *Proceedings of the computer vision and pattern recognition conference* (pp. 4205–4215).

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., ... others (2025). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zahedifar, R., Mirghasemi, S. A., Baghshah, M. S., & Taheri, A. (2025). Llm-agent-controller: A universal multi-agent large language model system as a control engineer. *arXiv preprint arXiv:2505.19567*.

BIOGRAPHIES



JONGSU PARK received the Master of Engineering degree in the Department of Industrial and Management Systems Engineering from Kyung Hee University, Yongin, Republic of Korea, in 2023. He is currently a Ph.D. candidate in the Department of Artificial Intelligence at Kyung Hee University, Yongin, Republic of Korea. He is interested in human-supporting systems in smart factories. He is conducting research about designing integrated systems and introducing various new technologies, including AI application, into industrial sites.

SEONGWOO CHO is currently a researcher at the Smart Factory Laboratory in the Department of Industrial Management Engineering and a master student at the Graduate School of Artificial Intelligence at Kyung Hee University. His research interests include natural language processing, human-machine interfaces, and generative

AI, such as large language models and image generation models.

SENA NUR DURGUNLU is currently a researcher at the Smart Factory Laboratory in the Department of Industrial Management Engineering and a master's student at the Graduate School of Industrial and Management Systems Engineering at Kyung Hee University, Yongin, Republic of Korea. Her research interests include agentic artificial intelligence, digital twin, and industrial automation.

YOONJI CHAE is currently a researcher at the Smart Factory Laboratory in the Department of Industrial Management Engineering and a master's student at the Metaverse Convergence Graduate School, Kyung Hee University. Her research interests encompass intelligent manufacturing systems, industrial automation, and AI-driven design and control technologies.

JUMYUNG UM received the B.S. degree in mechanical engineering and information communication engineering from Sung Kyun Kwan University, Suwon, Republic of Korea, in 2003 and the Ph.D. degree in industrial and management engineering from Pohang university of science and technology in Republic of Korea, in 2012. Since 2018, he is currently serving as an associate professor in the Department of Industrial and Management System Engineering at Kyung Hee University, Yongin, Republic of Korea. From 2012 to 2014, he was a Post-doc researcher with Ecole Polytechnique Fédérale de Lausanne(EPFL), Switzerland. After leaving EPFL, He worked as a research associate at the University of Cambridge until 2015 and served as a senior researcher of Technology Initiative for SmartFactoryKL, Germany by 2018. His research interest includes the usage of Augmented reality for smart operators and Plug-n-Produce of modular factory systems.