# E3-MAS: A Self-Evolution Multi-Agent System Framework

Ming-Yi Huang, Yao-Zhi Xue, and Chai-Yu Lin
*Department of Computer Science and Information Engineering, National Central University, Taiwan*
Corresponding Author: Chia-Yu Lin (sallylin0121@ncu.edu.tw)

2025 IEEE/IEIE International Conference on Consumer Electronics-Asia (ICCE-Asia) | 979-8-3315-7402-4/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCE-Asia67487.2025.11263582

***Abstract*—Large Language Model (LLM)-based Multi-Agent Systems (MAS) have emerged as a powerful paradigm for solving complex, real-world tasks with tedious workflows and frequent errors. However, current self-evolution MAS approaches require extensive manual tuning and lack dynamic adaptation, precise issue identification, and modular flexibility. This paper presents *E3-MAS*, a general-purpose self-evolution framework that organizes agents into three interacting teams—Execution, Evaluation, and Evolution. We detail role-level designs (Planner/Executor/Replanner, Critic/Evaluator, Analyzer/Prompt Optimizer) and show how task-aware evaluation drives problem attribution and prompt refinement. Using a school administrative assistance scenario (C-Pilot), we demonstrate intelligent search and pipeline automation. In a leave-application case study, E3-MAS improves the task progress rate from $0.83$ to $1.0$ after one evolution cycle. More broadly, the framework consistently achieves progress rates exceeding $0.9$, reduces manual prompt-tuning time by over 50%, and enables modular deployment of MAS applications across heterogeneous environments. These results highlight the potential of E3-MAS as a scalable and adaptive paradigm for reliable multi-agent collaboration.**

***Index Terms*—Multi-Agent system, workflow automation, automated evaluation, self-evolution**

## I. Introduction

LLM-enabled agents are increasingly orchestrated as Multi-Agent Systems (MAS) to address complex, multi-step tasks requiring planning, tool use, and collaboration. While recent works explore reflective and self-improving agents, they often rely on hand-crafted workflows, human-in-the-loop tuning, or narrow domains, limiting generalization and maintainability. We target the following question: *How can a MAS continually improve itself with minimal human intervention while remaining easy to deploy across domains?*

Developing a self-evolution MAS stems from the challenges of real-world applications, such as school administrative task assistance. In these scenarios, MAS are expected to navigate and operate websites or internal systems, where tasks involve fine-grained operational details and multiple sources of potential errors. For example, completing leave applications or retrieving policy documents often requires precise execution steps. Without adaptive mechanisms, manual adjustments by developers become necessary. Such frequent manual tuning not only causes inefficiency but also increases maintenance overhead, ultimately limiting scalability.

However, existing approaches to self-evolving MAS still suffer from critical limitations. First, many frameworks lack dynamic task adaptation in their evaluation process, leading to coarse or generic feedback that fails to capture small operational failures. Second, they provide no mechanism for precise responsibility attribution, making it difficult to determine which agent should be refined or replaced after an error occurs. Third, most designs lack flexible applicability across domains, as their tightly coupled modules prevent seamless deployment in heterogeneous environments.

To address these challenges, we propose **E3-MAS**, a self-evolution MAS framework that closes the loop between execution, evaluation, and evolution. The Execution Team carries out user objectives through planning, execution, and replanning; the Evaluation Team introduces task-aware rubrics and fine-grained performance tracking; and the Evolution Team applies agent-level attribution and problem-driven prompt optimization. Together, these teams enable continuous improvement with minimal human oversight.

Our contributions are: (1) we design a general three-team architecture (Execution–Evaluation–Evolution) for self-evolving MAS; (2) we introduce role-level mechanisms and metrics for dynamic evaluation and targeted evolution; and (3) we demonstrate the framework through a school administrative case study and ablation experiments, showing improvements in task execution, reductions in manual adjustment time, and flexible applicability across MAS deployments.

## II. Related Work

Traditional approaches frequently add or remove agents to meet new requirements; however, such reconfiguration necessitates redefining roles, resynchronizing system states, and reallocating resources, all of which increase development and maintenance costs. Moreover, frequent architectural changes can destabilize collaboration and compromise decision consistency. To address these limitations, prompt optimization allows agents to adapt behaviors without altering system structure. Several automated methods have been proposed. Autonomous Prompt Engineering Toolbox (APET) [1] automatically refines prompts based on task demands, reducing reliance on human intervention. Dialogue history-based methods, such as REPROMPT [2], learn from past interactions to iteratively optimize prompts for improved reasoning. Feedback-driven techniques, exemplified by PromptWizard [3], employ batch optimization guided by evaluation signals to dynamically adapt prompts across domains such as customer service, automated decision-making, and knowledge retrieval. Compared with architectural reconfiguration, automated prompt engineering

offers lower cost, real-time adaptability, and improved stability, thereby enhancing MAS resilience in dynamic settings.

While prompt engineering provides mechanisms for adapting agent behaviors, its effectiveness depends on reliable evaluation signals to guide optimization. Evaluating adaptive agents, however, is challenging due to their non-deterministic nature and context-sensitive behavior. Traditional evaluation strategies primarily emphasize task outcomes, which risks overlooking reasoning transparency and decision consistency. Human-as-a-Judge evaluations, while valuable, face scalability and consistency challenges. To overcome these limitations, the Agent-as-a-Judge paradigm has been proposed, where agents automatically evaluate the performance of peers. Wu et al. demonstrated that such evaluations can achieve approximately 90% agreement with human judgment, while also enabling monitoring of intermediate reasoning steps [4]. Building on this idea, Arabzadeh et al. introduced the AgentEval framework [5], consisting of CriticAgent, QuantifierAgent, and VerifierAgent, which collaboratively generate rubrics, quantify performance, and validate evaluation standards. Similarly, Xia et al. proposed Evaluation-Driven Design [6], embedding continuous evaluation throughout the agent lifecycle to ensure sustained adaptation. These advances provide more transparent, explainable, and cost-effective assessment mechanisms, which directly inspire the Evaluation Team design in E3-MAS. By employing structured rubrics and agent-level critics, the framework delivers actionable feedback to the Evolution Team, enabling precise and efficient system refinement.

## III. E3-MAS Framework

E3-MAS is designed as a modular and self-evolving Multi-Agent System (MAS) capable of dynamic adaptation, fine-grained evaluation, and targeted prompt optimization. It consists of three interacting teams—**Execution**, **Evaluation**, and **Evolution**—that operate cyclically to continuously enhance task performance, as shown in Fig. 1.

Besides, we implement C-pilot, an execution-oriented multi-agent system to facilitate administrative tasks on school websites and internal systems. C-pilot demonstrates how modular role separation within the Execution Team can support both intelligent information retrieval and workflow automation.

### A. Execution Team

The Execution Team is responsible for end-to-end tasks by coordinating specialized agents:

- **Planner**: Generates multi-step plans tailored to user requirements and system specifications.
- **Executor**: Interacts with tools, APIs, and web interfaces to execute each step.
- **Replanner**: Intervenes when a failure occurs, revising or repairing incomplete steps.

This architecture is particularly suited for handling complex and repetitive processes as well as dynamic and variable environments. It also retains detailed records of both planning and execution processes, providing a foundation for subsequent evaluation and optimization. Through this mechanism, the system significantly improves robustness in handling long workflows and fragile web interactions. For example, in the school leave application scenario (C-pilot), the Execution Team can automate tasks such as form filling, certificate uploading, and workflow submission, demonstrating reliability and adaptability in real-world applications.

### B. Evaluation Team

The Evaluation Team monitors task progress and generates machine-actionable feedback:

- **Critic**: Establishes evaluation rubrics based on user input and the initial plan to review each step of the task.
- **Evaluator**: Applies task-specific rubrics to provide step-level progress rates and qualitative feedback throughout the execution process.

Unlike traditional manual evaluation, this automated process provides customized evaluation rubrics tailored to each task and refines them down to the step level, making it particularly effective for debugging and tracing multi-step workflows. In addition, each step is fully logged with both the executor and the execution details, supporting responsibility attribution and future optimization. This design ensures transparency in the decision-making process. For instance, in the C-pilot case, when the system failed to upload a certificate file, the Evaluation Team was able to quickly identify the root cause and suggest corrective measures.

### C. Evolution Team

The Evolution Team achieves self-improvement through targeted optimization, with its core mechanism being a troubleshooting process grounded in responsibility attribution:

- **Analyzer**: Detects issues and attributes responsibility to the appropriate agent by leveraging evaluation reports and task trajectories.
- **Prompt Optimizer**: After responsibility attribution, conducts targeted prompt engineering for the corresponding agent, introducing refinements such as verification steps, retry logic, and contextual enrichment.

Completed improvements undergo regression testing and canary deployment before being integrated into the production environment. The Evolution Team adopts a "one agent at a time" adjustment strategy to reduce optimization complexity and leverages multiple rounds of iterative testing to gradually improve overall task performance. Continuing the C-pilot example, the team refined the Pipeline Executor's prompts by adding file path verification and retry mechanisms.

### D. Key Advantages

The E3-MAS framework provides the following advantages:

- **Dynamic Adaptation**: Agents possess automated optimization capabilities, adapting to task contexts with minimal manual intervention. They can respond immediately to unexpected errors or environmental changes, ensuring uninterrupted task execution.

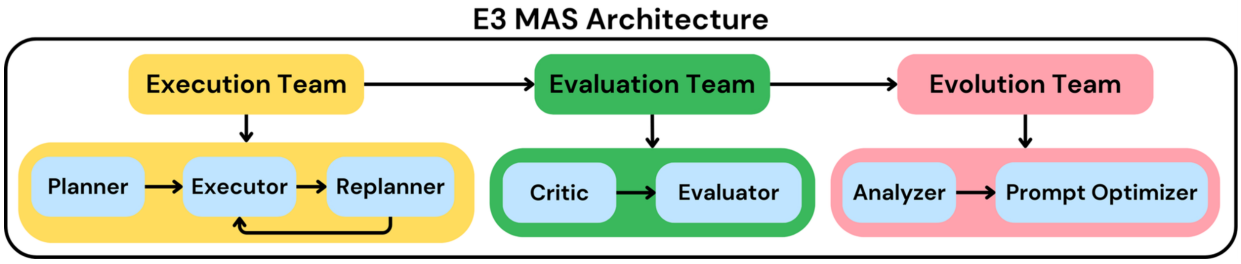


Fig. 1. The architecture of E3-MAS.

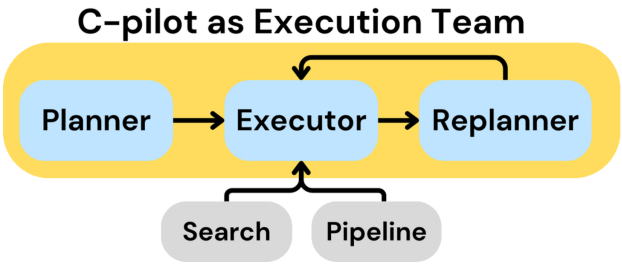


Fig. 2. The architecture of C-pilot as the Execution Team, comprising a Planner, Executor, and Replanner.

- **Transparent Evaluation**: Step-level rubric monitoring delivers quantitative results, enhancing decision interpretability and execution stability. This prevents black-box decisions and strengthens system trustworthiness.
- **Low-Cost Evolution**: Failures can be accurately attributed to specific agents and refined through targeted prompt engineering. This "single-point correction" reduces maintenance costs and risks, shortens iteration cycles, and accelerates adaptation.
- **Cross-Domain Applicability**: The modular design allows Execution Teams to be replaced or reorganized as needed, making the framework applicable across domains such as administrative automation and industrial processes. Cross-domain migration requires only module-level adjustments without redesigning the entire system.

## IV. Experiments

We use C-pilot to validate the E3-MAS framework. The architecture of C-pilot consists of three core roles: **Planner, Executor, and Replanner,** as shown in Fig. 2. The Planner decomposes high-level user requests into structured task sequences. The Executor interacts with online platforms and system interfaces to carry out each step, including form submission, document retrieval, and data entry. When execution errors occur, the Replanner adjusts the workflow dynamically by revising task steps or introducing fallback strategies. This design ensures resilience against task failures and reduces the need for human intervention.

C-pilot supports two major functions: **intelligent search and pipeline automation.** Intelligent search lets the system navigate across heterogeneous sources (e.g., school portals, databases, and document repositories) to extract relevant information. Pipeline automation, in turn, allows repetitive and rule-based administrative tasks to run seamlessly, such as leave applications, course registration workflows, or batch status updates. By combining planning, execution, and adaptive replanning, C-pilot exemplifies how the Execution Team within E3-MAS can improve efficiency, reliability, and scalability in real-world educational environments.

### A. Intelligent Search

To evaluate the intelligent search capability of C-pilot, we design an experiment where the system is tasked with retrieving information related to scholarship applications. Specifically, the Executor follows the user query: *"Please help me gather information related to scholarship applications."*

During the initial execution, the system fails to extract all relevant in-page hyperlinks that provide further information, which results in an evaluation score of 0 for the corresponding step. The Evaluation Team issues structured feedback highlighting the missing hyperlinks and the need for improved thoroughness in information extraction, as shown in Fig. 3.

Subsequently, the Evolution Team performs attribution analysis and identifies the root cause as insufficient extraction logic in the Search Executor. Targeted prompt refinement is then applied, focusing on hyperlink extraction, completeness checks, and clearer output formatting. After evolution, the task completes successfully with all hyperlinks listed, and the progress rate improved from 0.83 to 1.0, as shown in Fig. 4.

### B. Pipeline Automation

C-pilot is further evaluated on pipeline automation through a school leave application workflow. The user requests a personal leave between April 29 and April 30, 2025, and provides a leave certificate file located in a local directory.

The Planner generates a correct multi-step workflow, including the document upload step. However, during execution, the Pipeline Executor fails to upload the leave certificate file, resulting in a rubric score of 0 and an overall progress rate of 0.83, as illustrated in Fig. 5. The Evaluation Team analyzes the execution log and suggests verifying file paths, validating element identifiers, and retrying uploads with proper targeting.

The Evolution Team attributes the failure to the Pipeline Executor and refines its prompts to include explicit verification steps, retry logic, and improves error handling. After refinement, the leave-application workflow is completed end-to-end.

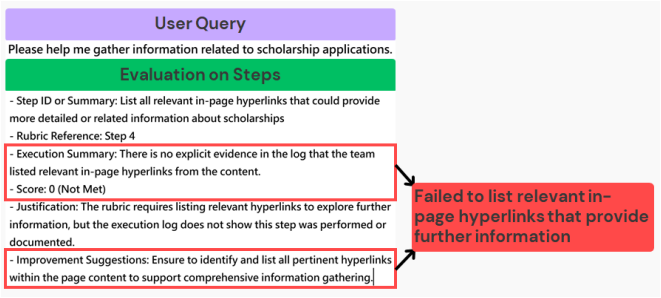


Fig. 3. The evaluation result of Intelligent Search before evolution (failure to list relevant hyperlinks).

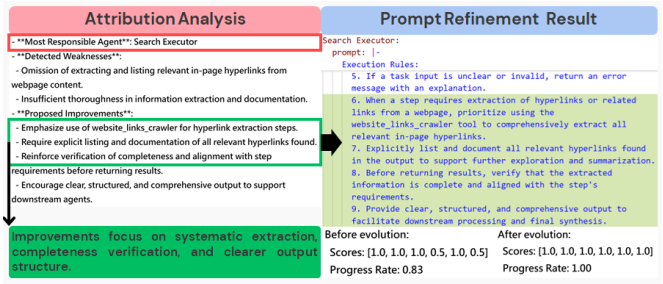


Fig. 4. The prompt refinement result and performance improvement after evolution in Intelligent Search.

The task progress rate increases from 0.83 to 1.0, and upload-related failures are fully eliminated, as shown in Fig. 6.

## V. Conclusion

In this work, we proposed E3-MAS, a self-evolving multi-agent system framework that integrated task-aware evaluation, precise attribution, and modular system design. The Evaluation Team provided structured rubrics and fine-grained performance tracking to capture execution quality in detail. The Evolution Team complemented this process by attributing responsibility to specific agents and applying problem-driven prompt optimization, enabling targeted and efficient refinement. Furthermore, the modular system design allowed seamless replacement of MAS components and supported framework-agnostic integration, ensuring flexibility across diverse applications. Through these combined mechanisms, E3-MAS consistently achieved a task progress rate exceeding 0.9, reduced manual prompt tuning time by more than 50%, and supported scalable deployment of MAS across heterogeneous environments. The framework demonstrated how execution, evaluation, and evolution can be tightly integrated to provide reliable, adaptive, and efficient multi-agent collaboration.

## Acknowledgments

This work is sponsored by the National Science and Technology Council (NSTC) under the project NSTC 114-2218-E-A49-017-.

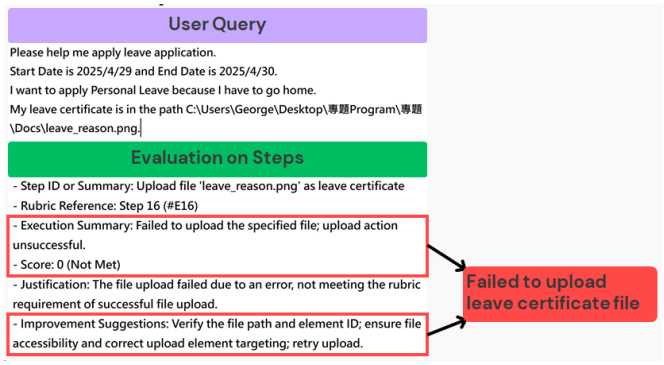


Fig. 5. The evaluation result of Pipeline Automation before evolution (failure to upload certificate file).

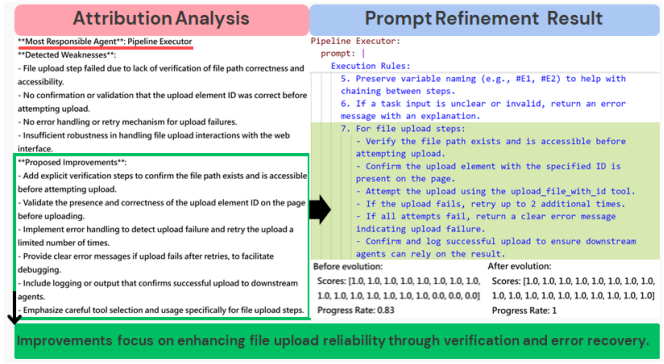


Fig. 6. The prompt refinement result and performance improvement after evolution in Pipeline Automation.

## References

[1] D. Kepel and K. Valogianni, "Autonomous prompt engineering in large language models," *arXiv preprint arXiv:2407.11000*, 2024.

[2] W. Chen, S. Koenig, and B. Dilkina, "Reprompt: Planning by automatic prompt engineering for large language models agents," *arXiv preprint arXiv:2406.11132*, 2024.

[3] E. Agarwal, J. Singh, V. Dani, R. Magazine, T. Ganu, and A. Nambi, "Promptwizard: Task-aware prompt optimization framework," *arXiv preprint arXiv:2405.18369*, 2024.

[4] M. Zhuge, C. Zhao, D. Ashley, W. Wang, D. Khizbullin, Y. Xiong, Z. Liu, E. Chang, R. Krishnamoorthi, Y. Tian *et al.*, "Agent-as-a-judge: Evaluate agents with agents," *arXiv preprint arXiv:2410.10934*, 2024.

[5] N. Arabzadeh, S. Huo, N. Mehta, Q. Wu, C. Wang, A. Awadallah, C. L. Clarke, and J. Kiseleva, "Assessing and verifying task utility in llm-powered applications," *arXiv preprint arXiv:2405.02178*, 2024.

[6] B. Xia, Q. Lu, L. Zhu, Z. Xing, D. Zhao, and H. Zhang, "An evaluation-driven approach to designing llm agents: Process and architecture," *arXiv e-prints*, pp. arXiv–2411, 2024.