World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

A COLLABORATIVE MULTI-AGENT FRAMEWORK FOR SELF-EVOLVING SOFTWARE ENGINEERING AND DATA INTELLIGENCE

Kavitha Soppari, Pravallika Dhurbhakula *, Srikara Karthikeya Karanam and Akshay Sriperumbuduru

Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), ACE Engineering College, Hyderabad, Telangana, India.
* Corresponding Author

ORCID Details
Kavitha Soppari: https://orcid.org/0000-0002-5343-2320
Pravallika Dhurbhakula: https://orcid.org/0009-0004-4289-5241
Srikara Karthikeya Karanam: https://orcid.org/0009-0003-4269-4952
Akshay Sriperumbuduru: https://orcid.org/0009-0001-2209-6109

World Journal of Advanced Research and Reviews, 2026, 32(02), 1347–1361

Publication history: Received on 10 October 2026; revised on 16 November 2026; accepted on 18 November 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.32.2.2955

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Large Language Models (LLMs) have demonstrated significant potential in automating software engineering and data intelligence tasks. However, complex multi-step workflows continue to present challenges in task planning, agent coordination, external tool execution, output verification, and learning from previous experiences. This paper proposes a Collaborative Multi-Agent Framework for Self-Evolving Software Engineering and Data Intelligence. The framework integrates LLMs, Retrieval-Augmented Generation (RAG), external tools, and reflective memory to support coordinated task execution. It employs specialized agents for requirement understanding, strategy selection, task planning, tool execution, output verification, and reflection. A Critic Agent evaluates generated outputs, while a Reflection Agent identifies useful lessons from previous executions and stores them for future retrieval. The framework is designed to support software engineering activities, including code generation, debugging, and testing, alongside data intelligence tasks such as data cleaning, analysis, and visualization. The proposed architecture aims to improve task coordination, output reliability, and experience-based adaptation through reusable knowledge and feedback mechanisms. The framework is currently at the proposed design stage; implementation and experimental evaluation are planned as future work.

Keywords: Artificial Intelligence, Data Intelligence, Multi-Agent Systems, Reflective Memory, Retrieval-Augmented Generation, Software Engineering

1 INTRODUCTION

Recent advancements in Artificial Intelligence (AI) and Large Language Models (LLMs) have expanded the automation of tasks such as code generation, data analysis, debugging and problem-solving. However, complex tasks involving multiple steps, external tools and output verification can be challenging for a single AI agent to manage reliably.

Multi-Agent Systems (MAS) address these challenges by distributing tasks among specialized agents that collaborate to achieve a common objective. Each agent performs specific responsibilities, while an orchestration mechanism coordinates their activities. Integrating Retrieval-Augmented Generation (RAG), external tools and memory mechanisms further supports knowledge retrieval, task execution and the reuse of previous experiences.

These capabilities provide a foundation for developing adaptive multi-agent frameworks. This research explores their integration into a collaborative architecture for software engineering and data intelligence, with an emphasis on coordinated execution, output verification and experience-based improvement.

1.1 Motivation

Existing AI systems can generate code, analyse data and assist with problem-solving. However, complex workflows often require coordinated task planning, external tool execution, output verification and the effective use of previous experiences. Managing these activities through an individual agent may introduce difficulties in reliability and task coordination.

The motivation behind this research is to develop a collaborative framework in which specialized agents perform individual tasks, coordinate their activities and evaluate intermediate outputs. By incorporating Retrieval-Augmented Generation and reflective memory, the proposed framework aims to reuse relevant knowledge and execution experiences to improve subsequent task performance across software engineering and data intelligence applications.

2 LITERATURE REVIEW

Recent research has explored multi-agent architectures for domain-specific problem-solving, automated analysis and collaborative decision-making. Five relevant studies were selected to examine their methodologies, contributions and limitations.

2.1 A. Ibrahim et al. (2026) – "TriAgent: An Adaptive Multi-Agent Architecture for Crisis Clinical Decision Support Under Incomplete Information"

This study presents TriAgent, an adaptive multi-agent architecture for clinical decision support under incomplete information. It employs an orchestrator and specialized agents for assessment, information retrieval and safety-related analysis. A Critique Agent evaluates the generated reasoning before the final output.

- **Strengths:** Supports decision-making under incomplete information through adaptive agent coordination and retrieval-supported reasoning. It also incorporates verification mechanisms.
- **Limitations:** The evaluation is retrospective, and further prospective clinical validation is required to establish its effectiveness in real-world clinical settings.

2.2 F. Chen et al. (2026) – "A Conversational Multi-Agent AI System for Automated Plant Phenotyping"

This study introduces PhenoAssistant, a conversational multi-agent system for automated plant phenotyping. It combines specialized agents, computer vision models, retrieval mechanisms and external tools to support plant analysis through natural-language interaction.

- **Strengths:** Automates plant phenotyping workflows, supports tool selection and integrates critic-based verification for evaluating generated outputs.
- **Limitations:** Complex task decomposition and fine-grained visual reasoning remain challenging, and human inspection is required for certain workflows and results.

2.3 A. Joshi et al. (2026) – "A Multi-Agent Large Language Model Framework for Intelligent Vendor Evaluation and Risk-Aware Procurement Decisions"

This study proposes an LLM-based multi-agent framework for vendor evaluation and procurement decision support. Specialized agents perform financial analysis, risk profiling, sentiment monitoring and industry benchmarking to generate structured vendor assessments.

- **Strengths:** Combines multiple evaluation dimensions and supports systematic vendor comparison through specialized analytical agents.
- **Limitations:** The reliability of the generated assessments depends on the quality of available vendor information and the effectiveness of the evaluation process.

2.4 A. Agrawal et al. (2026) – "Artificial Intelligence Driven Multi-Agent Framework for Adaptive Cyber Attack Simulation and Automated Incident Response in Cyber Range Environments"

This study presents an AI-driven multi-agent framework for adaptive cyberattack simulation and automated incident response. It employs coordinated attacker and defender agents within cyber-range environments to support security experimentation and incident-response evaluation.

- **Strengths:** Demonstrates coordinated agent behaviour, adaptive cybersecurity simulation and incident-response capabilities in controlled environments.
- **Limitations:** Experimental findings obtained in cyber-range environments require further consideration when assessing their applicability to real-world cybersecurity operations.

2.5 R. Ferreira et al. (2026) – "Agentic AI Architecture for Multi-Criteria Decision-Making: A Collaborative Human-AI Framework"

This study proposes a collaborative human-AI architecture for multi-criteria decision-making. It combines generative agents, logical agents and human participants to support criteria evaluation, preference analysis and structured decision-making.

- **Strengths:** Integrates AI-based reasoning with deterministic operations and human validation, providing a structured approach to collaborative decision-making.
- **Limitations:** The architecture is conceptual and does not report an implemented system or empirical performance evaluation. Further experimental validation is required.

2.6 Research Gaps

Four gaps emerge from the reviewed work: limited cross-task support, limited reuse of past experience, insufficient output verification, and the absence of a common evaluation framework. These gaps are analysed in detail in Section III-D and motivate the framework proposed in this paper.

2.7 Problem Statement

Existing AI systems face challenges in managing complex tasks that require multiple execution stages, specialized capabilities, external tool utilization and output verification. Although multi-agent architectures enable collaborative task execution, effective coordination, reusable experience management and consistent output validation across different task categories remain important challenges.

Therefore, this research proposes a collaborative multi-agent framework for software engineering and data intelligence that integrates specialized agents, Retrieval-Augmented Generation, external tools, output verification and reflective memory to support coordinated and experience-based task execution**.**

2.8 Research Objectives

The main objective of this research is to design and develop a collaborative multi-agent AI **framework** that supports software engineering and data intelligence tasks.

The specific objectives are:

- **Task Decomposition:** Decompose complex user requests into smaller tasks and assign them to appropriate specialized agents.
- **Agent Collaboration:** Establish coordination and communication among agents to support collaborative task execution.
- **Knowledge Integration:** Incorporate Retrieval-Augmented Generation (RAG) to retrieve relevant information from available knowledge sources.
- **Tool Integration:** Enable task execution through external tools such as Python, Pandas and software testing utilities.
- **Output Verification:** Implement a Critic Agent to evaluate generated outputs, identify errors and support refinement.

- **Reflective Memory:** Develop a memory mechanism to retain relevant execution experiences and reuse them to improve subsequent task strategies.

2.9 Research Contributions

The major contributions of the proposed research are as follows:

- **Unified Cross-Domain Architecture:** Proposes a common multi-agent coordination framework intended to support both software engineering and data intelligence tasks.
- **Integrated Retrieval and Tool Execution:** Incorporates Retrieval-Augmented Generation and external tool utilization within a coordinated task-execution workflow.
- **Independent Output Verification:** Introduces a dedicated Critic Agent to evaluate generated outputs, including code, analytical results and tool-execution outcomes.
- **Reusable Reflective Memory:** Proposes a reflective memory mechanism to retain useful execution experiences and retrieve relevant information for subsequent tasks across different categories.
- **Comprehensive Evaluation Approach:** Defines a common set of evaluation criteria covering task performance, agent coordination, tool utilization, output verification, adaptability and reliability.

These contributions represent the proposed design objectives and architectural features. Their effectiveness will be examined through implementation and experimental evaluation in future work.

3 MATERIALS AND METHODS

3.1 Overview of the Proposed Framework

The proposed system follows a collaborative multi-agent architecture in which multiple specialized AI agents work together to complete complex user tasks. The system uses Large Language Models (LLMs) for reasoning, Retrieval-Augmented Generation (RAG) for knowledge retrieval, and external tools for task execution.

The workflow begins with a user request, which is analysed and converted into smaller tasks. These tasks are assigned to suitable agents for planning and execution. A Critic Agent evaluates the generated results, while a Reflection Agent identifies useful lessons and stores them in memory for future tasks.

The framework is designed to support software engineering activities such as code generation, debugging, and testing, along with data intelligence tasks such as data cleaning, spreadsheet analysis, and visualization.

3.2 System Architecture

The proposed system follows a modular architecture that connects the user interface, Large Language Model (LLM), specialized agents, knowledge retrieval, external tools, and memory.

The architecture consists of the following major components:

- **User Interface**: Accepts user requests and displays the generated results.
- **LLM Engine:** Handles natural-language understanding, reasoning, and solution generation.
- **Agent Layer:** Contains specialized agents responsible for requirement understanding, strategy selection, task planning, and tool execution. The Critic Agent and Reflection Agent perform output verification and experience analysis, respectively.
- **RAG Component:** Retrieves relevant information from available documents and knowledge sources.
- **External Tools:** Uses Python, Pandas, and testing utilities to perform coding and data-processing tasks.
- **Critic Agent:** Evaluates generated outputs and identifies errors or missing requirements.
- **Reflective Memory:** Stores useful lessons from previous executions for potential use in future tasks.

These components work together to support task execution, output verification, and experience-based adaptation. The architecture is proposed and will be integrated and tested during the development phase.

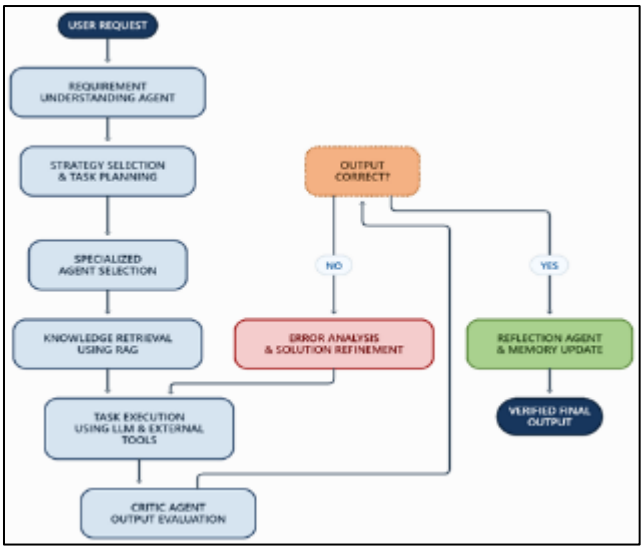


Figure 1: Proposed System Architecture of the Self-Evolving Collaborative Multi-Agent AI System.

3.3 Specialized Agent Architecture

The proposed framework consists of specialized agents, each responsible for a particular stage of task execution. These agents use the LLM engine for reasoning and operate through a coordinated workflow to complete user requests.

The major agent roles are:

- **Requirement Understanding Agent:** Analyses the user's request, identifies the objectives, and extracts the necessary requirements.
- **Strategy Selection and Planning Agent:** Determines an appropriate approach, divides complex requests into smaller tasks, and prepares an execution plan.
- **Tool Selection and Execution Agent:** Selects suitable external tools, such as Python, Pandas, and testing utilities, and performs the assigned tasks.
- **Critic Agent**: Evaluates the generated outputs, checks whether the requirements are satisfied, and identifies errors or missing information.
- **Reflection Agent:** Analyses execution outcomes, identifies useful lessons, and stores relevant experiences in reflective memory for potential use in future tasks.

These agents work collaboratively through a structured workflow. If errors are identified during evaluation, the results are sent back for refinement and re-execution. Once the output satisfies the required criteria, the Reflection Agent updates the memory with useful observations.

The specialized agent architecture is proposed as part of the system design and will be implemented and evaluated during the development phase.

3.4 Agent Coordination and Task Planning

The proposed framework uses a structured coordination mechanism to manage interactions between specialized agents. When a user submits a request, the Requirement Understanding Agent analyses it and identifies the tasks involved The Strategy Selection and Planning Agent determines the execution sequence and assigns each sub-task to the appropriate specialized agent based on its responsibilities and required capabilities.

The coordination process involves the following steps:

- **Task Identification:** Understand the user request and identify the required operations.
- **Task Decomposition:** Divide complex requests into smaller, manageable sub-tasks.
- **Agent Assignment:** Assign each sub-task to the appropriate specialized agent.
- **Task Execution:** Enable agents to perform their assigned operations using LLM reasoning and external tools.
- **Result Integration:** Combine the outputs generated by different agents into a unified response.
- **Feedback and Refinement**: Send outputs for evaluation and repeat execution when errors are identified.

This coordination mechanism is designed to support systematic task execution, reduce unnecessary processing, and improve collaboration among agents. The proposed approach will be evaluated during the implementation phase.

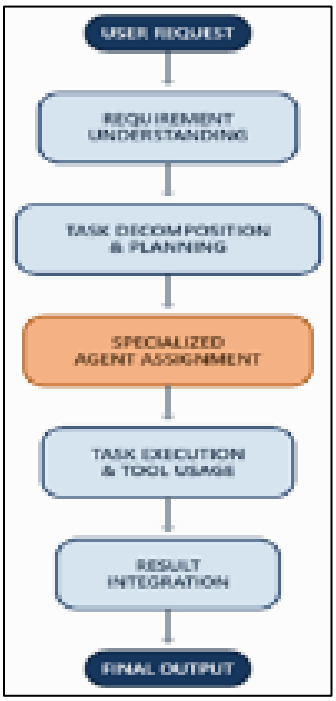


Figure 2: Agent Coordination and Task Execution Workflow.

3.5 Retrieval-Augmented Knowledge Integration

The proposed framework integrates Retrieval-Augmented Generation (RAG) to provide relevant information during task execution. RAG enables the system to retrieve knowledge from available documents and external knowledge sources before generating a response.

The knowledge integration process involves the following steps:

- **Knowledge Collection:** Gather relevant documents and information from available knowledge sources.
- **Document Processing:** Process the collected information and prepare it for retrieval.
- **Knowledge Retrieval:** Retrieve relevant information based on the user's request or assigned sub-task.
- **Context Integration**: Provide the retrieved information as context to the LLM for reasoning and response generation.
- **Response Generation:** Generate outputs using the retrieved knowledge and the assigned agent's capabilities.

By integrating RAG, the proposed framework aims to improve context availability and support more informed task execution. It is intended to reduce reliance on the LLM's parametric knowledge by incorporating relevant external information when required.

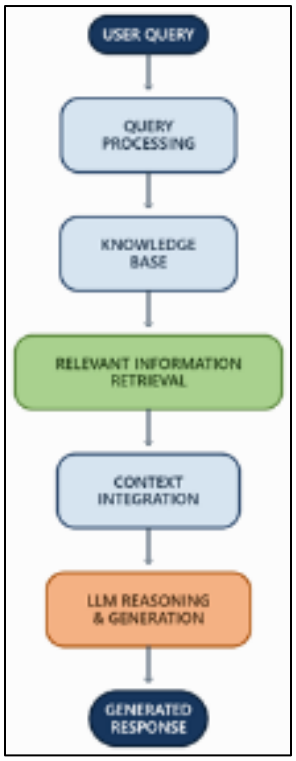


Figure 3: Retrieval-Augmented Generation Workflow.

3.6 External Tool Integration

The proposed framework integrates external tools to perform tasks that require computation, data processing, and software testing. While LLMs handle reasoning and task generation, external tools help execute operations that require specialized functionality.

The framework is designed to support the following tools:

- **Python:** Executes code, performs computations, and supports general-purpose task execution.
- **Pandas:** Handles spreadsheet operations, data cleaning, data manipulation, and analysis.
- **Testing Utilities:** Support code execution checks, error detection, and software testing.
- **Data Visualization Tools:** Generate visual representations of processed data to support analysis and interpretation.

The Tool Selection and Execution Agent identifies suitable tools based on the assigned task. It invokes the selected tools, captures their outputs, and passes the results to the Critic Agent for evaluation.

This integration aims to extend the framework beyond text generation by enabling practical task execution. The external tool integration will be implemented and tested during the development phase.

G. Output Verification Mechanism

The proposed framework incorporates a Critic Agent to verify the outputs generated during task execution. This mechanism aims to identify errors, incomplete results, and deviations from the user's requirements before delivering the final output.

The verification process involves the following steps:

- **Output Collection:** Collect the results generated by the assigned agents and external tools.
- **Requirement Validation:** Compare the generated outputs with the original user requirements.
- **Error Identification:** Identify incorrect results, missing information, and execution errors.
- **Feedback Generation:** Provide feedback to the responsible agent for correcting the identified issues.
- **Output Refinement:** Re-execute or refine the task based on the feedback.

- **Final Approval:** Accept the output when it satisfies the defined evaluation criteria and meets the original user requirements.

The Critic Agent acts as an independent evaluation component within the framework. It aims to improve output reliability by introducing a verification and feedback loop before task completion. The effectiveness of this mechanism will be assessed during system implementation and testing.

3.7 Reflective Memory and Self-Evolution

The proposed framework incorporates reflective memory to retain useful information from previous task executions and support experience-based improvement. The Reflection Agent analyses execution outcomes, identifies lessons, and stores relevant observations for potential use in future tasks.

The reflective memory process involves the following steps:

- **Execution History Collection:** Collect information about completed tasks, agent actions, and execution outcomes.
- **Performance Analysis:** Analyse successful and unsuccessful executions to identify strengths and errors.
- **Reflection Generation:** Generate reflections based on identified issues, feedback, and task outcomes.
- **Memory Storage:** Store useful reflections and lessons in long-term memory.
- **Memory Retrieval:** Retrieve relevant experiences when similar tasks are encountered in the future.
- **Strategy Improvement:** Use previous experiences to support better task planning and execution.

This mechanism aims to enable the framework to utilize previous execution outcomes rather than treating every task independently. Retrieved experiences are intended to guide future task planning, agent selection and execution strategies. This form of self-evolution focuses on experience-based adaptation through reflective memory rather than automatic modification or retraining of the underlying LLM. Its effectiveness will be evaluated during the implementation and testing phases.

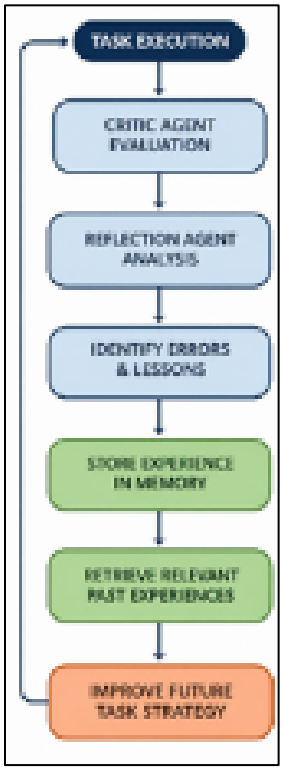


Figure 4: Reflective Memory and Self-Evolution Cycle.

3.8 Proposed Execution Algorithm

The proposed framework follows a sequential execution process in which specialized agents collaborate to complete user requests. The algorithm combines requirement analysis, task planning, knowledge retrieval, tool execution, output verification, and reflective memory updates.

Algorithm: Self-Evolving Collaborative Multi-Agent Execution

- **System Initialization:** Initialize the LLM engine, specialized agents, RAG component, external tools, and reflective memory.
- **User Request:** Accept the task or query submitted through the user interface.
- **Requirement Analysis and Planning:** Analyse the request, identify objectives, and decompose it into smaller sub-tasks.
- **Knowledge and Memory Retrieval:** Retrieve relevant information from the knowledge base and previous execution experiences.
- **Agent Selection and Task Assignment**: Select suitable agents and assign the identified sub-tasks.
- **Task Execution**: Execute assigned tasks using LLM reasoning and external tools.
- **Output Verification:** Evaluate the generated results using the Critic Agent.
- **Error Correction and Reverification:** If errors are detected, refine the solution, re-execute the task, and verify the updated output.
- **Reflection and Memory Update:** Analyse the execution outcome and store useful lessons in reflective memory.
- **Final Result:** Present the verified output to the user.

The proposed algorithm establishes a feedback-driven execution process in which evaluation and reflection support experience-based adaptation. Its practical performance will be examined during the implementation and testing stages.

3.9 Technology Stack

The proposed framework uses a combination of Artificial Intelligence technologies, programming tools, and supporting components to enable collaborative task execution, knowledge retrieval, and experience-based adaptation.

The major technologies and their intended roles are:

- **Large Language Models (LLMs):** Support natural-language understanding, reasoning, planning, and response generation.
- **Multi-Agent Architecture**: Enables specialized agents to collaborate and perform different stages of task execution.
- **Retrieval-Augmented Generation (RAG):** Retrieves relevant information from external knowledge sources to support context-aware responses.
- **Python:** Serves as the primary programming language for implementing agent workflows and task execution.
- **Pandas**: Supports spreadsheet processing, data cleaning, manipulation, and analysis.
- **External Testing Utilities:** Support code validation, error identification, and software testing.
- **Reflective Memory:** Stores useful execution experiences and reflections for future retrieval and improvement.
- **Data Visualization Tools:** Support the generation of charts and visual representations for data analysis.

The integration of these technologies is intended to provide a modular and extensible foundation for the proposed system. The final technology configuration will be determined during implementation based on compatibility, performance, and project requirements.

4 RESULTS AND DISCUSSION

4.1 Comparative Performance Analysis of Reviewed Studies

The five reviewed studies demonstrate the effectiveness of multi-agent architectures across different application domains. Their reported performance metrics are summarized below.

TriAgent [1] achieved 65.7% overall accuracy and 85% critical-case recall, outperforming its standalone LLM and traditional RAG baselines. PhenoAssistant [2] achieved an average tool-selection evaluation score of 4.66/5 with the Manager-plus-Critic configuration and 85% accuracy in data-analysis tasks. The vendor evaluation framework [3] reported a highest BERTScore of 0.758 for Piramal Pharma, while Dr. Lal Path Labs achieved the highest BLEU (0.19) and METEOR (0.14) scores. The cybersecurity framework [4] reported a 91% detection F1-score and 5.3-second response latency. The human-AI MCDM framework [5] presented a conceptual architecture without empirical performance results.

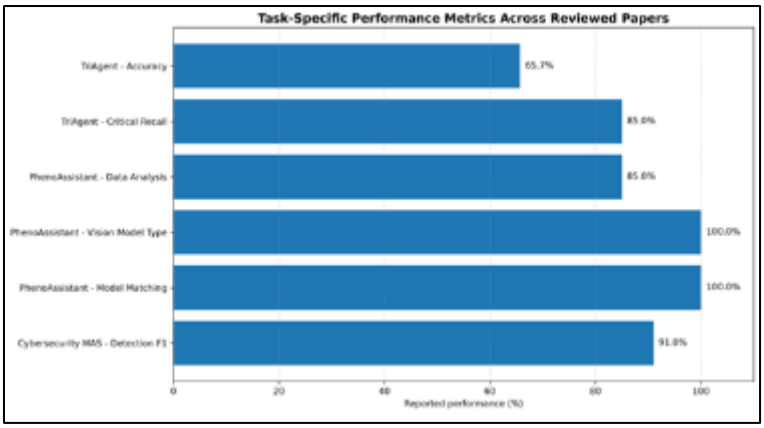


Figure 5: Comparative analysis of task-specific performance metrics reported across the reviewed multi-agent frameworks.

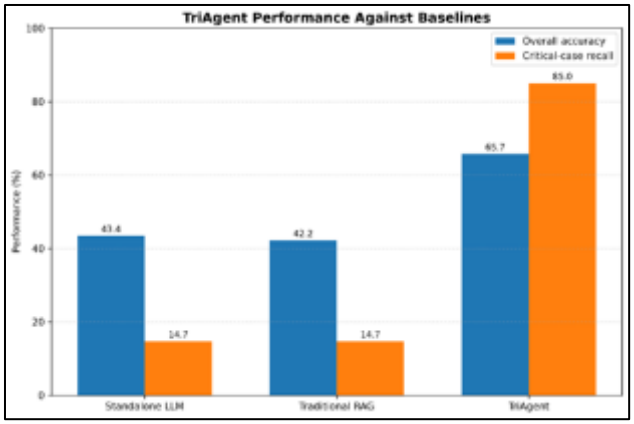


Figure 6: Performance comparison of the TriAgent framework with standalone LLM and traditional RAG baselines.

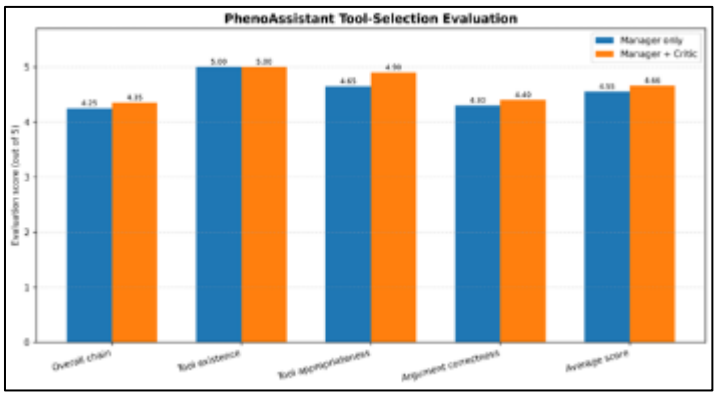


Figure 7: Tool-selection evaluation of PhenoAssistant under Manager-only and Manager-plus-Critic configurations.

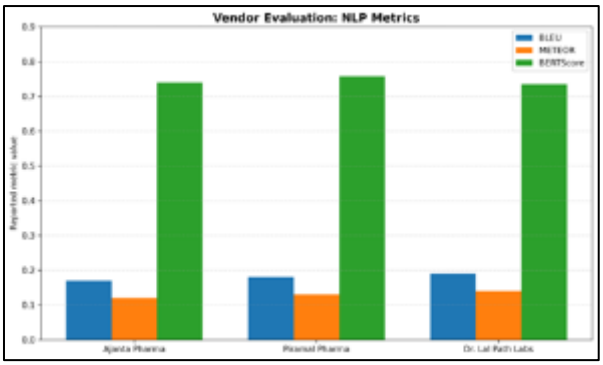


Figure 8: Comparative evaluation of vendor-generated outputs using BLEU, METEOR and BERT Score.

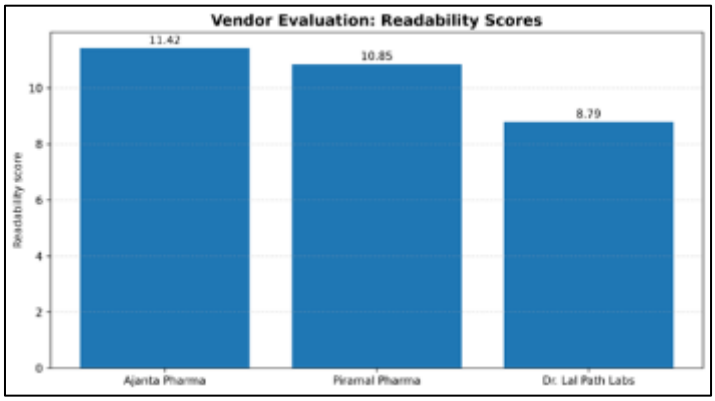


Figure 9: Readability score comparison of generated vendor evaluations.

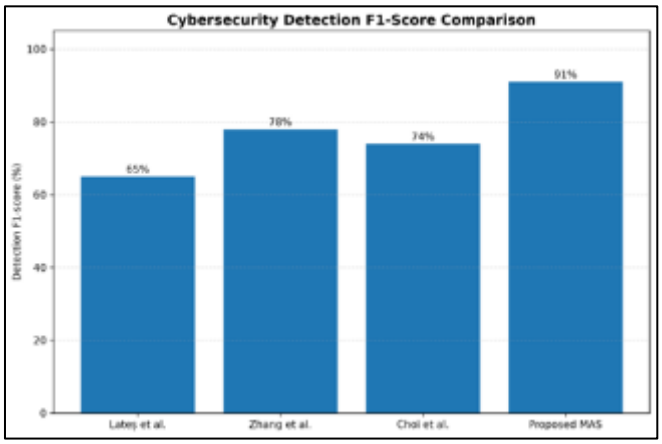


Figure 10: Comparative detection F1-scores of the evaluated cybersecurity frameworks.

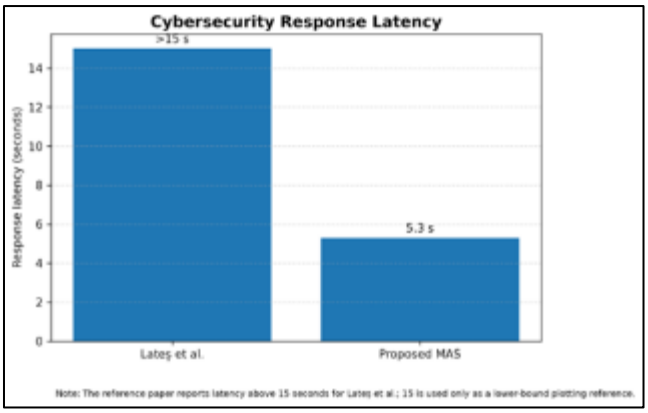


Figure 11: Comparison of reported incident-response latency in cybersecurity frameworks.

These metrics represent different task-specific evaluation criteria and should not be interpreted as directly comparable measures of overall system performance.

4.2 Comparative Analysis of Multi-Agent Capabilities

The reviewed frameworks employ specialized agents for different tasks, as summarized in Table I.

Table 1: Comparative Analysis of Agent Capabilities Across Reviewed Studies

Agent capability	TriAgent [1]	PhenoAssistant [2]	Vendor Evaluation [3]	Cybersecurity MAS [4]	MCDM [5]
Agent collaboration	Yes	Yes	Yes	Yes	Yes
Task coordination	Adaptive orchestration	Manager-based planning	Specialized agents	Coordinated agents	Structured workflow
RAG integration	Yes	Yes	Not established	Not established	Not established
Memory mechanism	Not clearly specified	Pipeline saving and reproduction	Not clearly specified	Not clearly specified	Not clearly specified
Output verification	Critique and safety checks	Critic Agent	Multi-metric evaluation	Detection and validation	Human validation
Experience-based adaptation	Adaptive orchestration	Pipeline reuse	Not established	Reinforcement-learning-based adaptation	Not empirically demonstrated

Overall, the reviewed studies demonstrate specialized agent collaboration, task coordination and output verification. However, differences in retrieval integration, memory mechanisms and experience-based adaptation indicate opportunities for developing a unified self-evolving multi-agent framework.

4.3 Methodology-Oriented Simulation Results

Five Python simulations were executed to demonstrate the principal workflows of the reviewed architectures using synthetic inputs.

The simulations successfully demonstrated task planning, specialized agent execution, output verification and human oversight. However, they use synthetic data and simplified logic and do not reproduce the original papers' experiments or reported metrics.

Table 2: Summary of Simulation Results

Simulation	Key outcome
TriAgent	Retrieval and safety checks performed; human review required.
PhenoAssistant	Synthetic plant traits generated and output accepted by Critic Agent.
Vendor Evaluation	Synthetic Vendor C ranked first with a composite score of 81.38.
Cybersecurity MAS	Two suspicious events detected and correlated; no real system actions executed.
Human-AI MCDM	Synthetic Option B ranked first with a score of 82.9; human approval pending.

4.4 Comparative Evaluation and Research Gaps

The reviewed studies use different evaluation approaches, including accuracy, recall, F1-score, response latency, linguistic metrics and conceptual analysis. Figure 12 summarizes the evaluation evidence reported across the five studies.

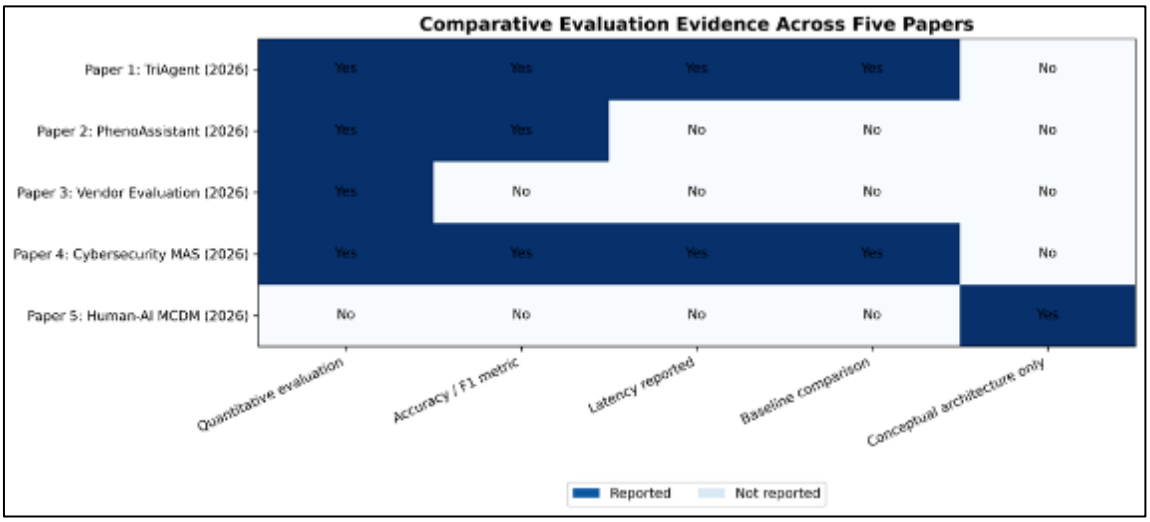


Figure 12: Comparative evaluation evidence matrix highlighting the reported assessment approaches across the five reviewed studies.

The comparative analysis identifies several research gaps in the existing multi-agent AI architectures. Although the reviewed systems demonstrate domain-specific capabilities, opportunities remain for developing a unified framework that supports broader task execution and continuous improvement.

4.4.1 Limited Cross-Domain Task Support

Most reviewed systems are designed for specific application domains, such as clinical decision support, plant phenotyping, vendor evaluation, cybersecurity, or multi-criteria decision-making. Their architectures are generally optimized for their respective workflows. This creates an opportunity for a framework that can support multiple categories of software engineering and data intelligence tasks through specialized agents.

4.4.2 Limited Reflective Experience Reuse

Some studies incorporate memory, pipeline reproduction, feedback, or reinforcement learning. However, these mechanisms address different domain-specific requirements. A unified mechanism that evaluates previous execution outcomes, identifies recurring errors, and stores reusable lessons for future tasks remains an important architectural opportunity.

4.4.3 Need for Stronger Output Verification

The reviewed studies demonstrate different verification approaches, including Critic Agents, safety checks, quantitative evaluation, and human-in-the-loop validation. However, complex multi-step tasks may still require additional verification of generated code, analytical results, tool execution, and final outputs. This highlights the need for a systematic verification mechanism across different task categories.

4.4.4 Need for Broader and Systematic Evaluation

The reviewed studies use different datasets, evaluation metrics, and experimental conditions. Some focus on accuracy, while others evaluate response time, semantic similarity, or task-specific reliability. A broader evaluation approach is required to assess agent coordination, execution efficiency, retrieval relevance, output correctness, and adaptation within a common framework.

These gaps motivate the development of a collaborative multi-agent architecture that integrates specialized task execution, knowledge retrieval, external tool interaction, output verification, and reflective memory.

4.5 Relevance to the Proposed Framework

The literature analysis identifies the need for a unified architecture integrating specialized agents, task planning, retrieval-augmented knowledge, external tool utilization and output verification.

The proposed framework aims to address these requirements through collaborative agent coordination and reflective memory. Its self-evolution mechanism is intended to utilize previous execution outcomes and feedback to improve subsequent task strategies, particularly in software engineering and data intelligence applications.

4.6 Expected Evaluation Criteria

The proposed framework will be evaluated after implementation using the following criteria.

Table 3: Expected Evaluation Metrics

Evaluation dimension	Metrics
Task performance	Accuracy, task completion rate
Agent coordination	Handoff success rate, coordination overhead
Tool utilization	Tool-selection accuracy, execution success
Verification	Error detection rate, verification accuracy
Knowledge retrieval	Precision@K, Recall@K
Self-evolution	Improvement in task success and output correctness across repeated comparable tasks
Efficiency	Execution time, token consumption
Reliability	Failure rate, recovery rate

These metrics will help evaluate the framework's effectiveness, adaptability, reliability and operational efficiency.

5 CONCLUSION

The proposed research presents a Self-Evolving Collaborative Multi-Agent AI System designed to support software engineering and data intelligence tasks. The framework integrates Large Language Models, specialized agents, Retrieval-Augmented Generation, external tools, output verification, and reflective memory to enable coordinated task execution.

The proposed architecture aims to address limitations in existing AI systems related to cross-task support, task coordination, output reliability, and experience-based improvement. The integration of a Critic Agent and Reflection Agent is intended to support output verification and the reuse of lessons from previous execution experiences.

The framework is currently at the proposed design stage. Its practical effectiveness has not yet been experimentally validated. Future work will focus on implementing the architecture, integrating the required tools and memory components, conducting experiments across different task categories, and evaluating its performance against baseline approaches.

Overall, the proposed framework provides a foundation for developing more collaborative, reliable, and adaptive AI systems for software engineering and data intelligence.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The authors sincerely express their gratitude to Dr. Kavitha Soppari, Professor and Head of the Department, CSE (Artificial Intelligence & Machine Learning), ACE Engineering College, for her valuable guidance, encouragement, and continuous support throughout this research work.

Disclosure of conflict of interest

The authors declare no conflicts of interest regarding this manuscript.

REFERENCES

[1] Ibrahim, Ahmed, Ali AlSanousi, and Ahmed Serag. 2026. "TriAgent: An Adaptive Multi-Agent Architecture for Crisis Clinical Decision Support Under Incomplete Information" AI 7, no. 6: 230. https: //doi.org/10.3390/ai7060230

[2] Chen, F., Stogiannidis, I., Wood, A. *et al.* A conversational multi-agent AI system for automated plant phenotyping. *Nat Commun* **17**, 6391 (2026). https: //doi.org/10.1038/s41467-026-71090-y

[3] Joshi, A., Singh, M., Parashar, D. *et al.* A multi-agent large language model framework for intelligent vendor evaluation and risk-aware procurement decisions. *Sci Rep* **16**, 19826 (2026). https: //doi.org/10.1038/s41598-026-50952-x

[4] Agrawal, A., Nadeem, M., Al Nuaim, A. *et al.* Artificial intelligence driven multi agent framework for adaptive cyber attack simulation and automated incident response in cyber range environments. *Sci Rep* **16**, 11673 (2026). https: //doi.org/10.1038/s41598-026-45937-9

[5] Ferreira, R., Araújo, M., Tereso, A. *et al.* Agentic AI Architecture for Multi-Criteria Decision-Making: A Collaborative Human-AI Framework. *Group Decis Negot* **35**, 45 (2026). https: //doi.org/10.1007/s10726-026-10004-1