2025 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan) | 979-8-3315-8741-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCE-Taiwan66881.2025.11207976

AutoDevFlow: A Multi-Agent System for Automated Development Lifecycle Management

Yu-Hxiang Chen, Wei-Hsiang Sung, and Chia-Yu Lin
Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan
Corresponding Author: Chia-Yu Lin (sallylin0121@ncu.edu.tw)

***Abstract*—Digital transformation forces enterprises to adopt agile Continuous Integration/Continuous Deployment(CI/CD) pipelines for competitiveness. However, traditional tools like Jenkins and GitHub Actions rely on manual configurations, causing errors in code migration, inefficient debugging, and complex multi-cloud deployments. To address these issues, we introduce AutoDevFlow, an innovative LLM-powered multi-agent system that fully automates the software development lifecycle. Through initial task analysis by the Control Layer and division of labor among the four agent teams of the Processing Layer, AutoDevFlow enables accurate, scalable, and adaptive deployments. Experiments show robust error correction, effective cross-language conversion, and automatic deployment, reducing operational burdens and streamlining workflows. Our approach transforms CI/CD practices, enhancing efficiency, delivery speed, and quality.**

***Index Terms*—Large language model, AI agent, multi-agent system, CI/CD**

I. Introduction

As digital transformation accelerates, enterprises face increasing pressure to adopt agile IT architectures and streamlined Continuous Integration/Continuous Deployment (CI/CD) pipelines. However, existing solutions often rely on manual configurations and static processes, leading to inefficiencies such as error-prone code migrations, prolonged debugging cycles, and complex multi-cloud deployments.

Traditional CI/CD tools like Jenkins, GitHub Actions, and SonarQube [1] automated deployments but required extensive manual setup and struggled with emerging technologies. MetaGPT [2] enhanced efficiency through role-based agents but remains rigid when handling unexpected challenges. LLM-driven Infrastructure-as-Code (IaC) generation [3] improved flexibility yet suffers from hallucinations and demands high DevOps expertise. These limitations underscore the need for a fully automated CI/CD solution that ensures scalability, accuracy, and adaptability while minimizing human intervention.

To address these challenges, we introduce AutoDevFlow, an LLM-powered multi-agent system. AutoDevFlow automates the entire software development lifecycle, including code conversion, error correction, and deployment with four system mechanisms: Optimized Task Allocation, Planning First, Iterative Correction Process, and Modular Collaboration Architecture. Experiments demonstrate robust error correction, efficient cross-language handling, and automatic deployment, allowing engineers to focus on innovation.

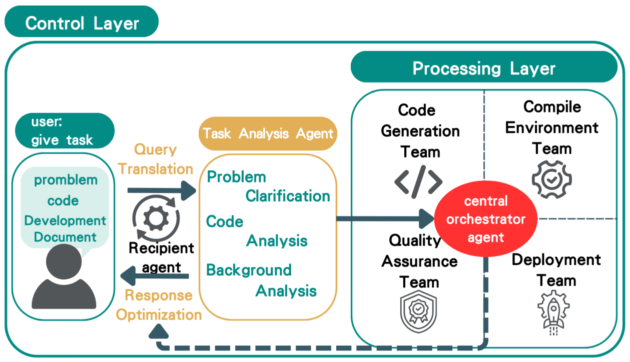

Fig. 1. system architecture of AutoDevFlow.

II. Proposed Framework

The architecture of AutoDevFlow, as shown in Fig. 1, is divided into two distinct layers: the Control Layer and the Processing Layer. The Control Layer initially analyzes the user's task and delegates subtasks to different agent teams in the Processing Layer. The implementation is explained below.

A. Control Layer

As shown in the outermost frame in Fig. 1, after the users give a task, the Recipient Agent performs query translation for better performance. The Task Analysis Agent then clarifies and analyzes the requirements, passing the results to the Central Orchestrator Agent. The Central Orchestrator Agent decides which agent team should be assigned the task for maximum efficiency. Once assigned, the designated team carries out its responsibilities. When the team finishes its work, the flow returns to the Central Orchestrator Agent, which determines the next step, for example, assigning any remaining tasks to another agent team or, if the task is complete, returning to the Recipient Agent. Finally, the Recipient Agent performs response optimization and provides easy-to-understand feedback to the user.

B. Processing Layer

Taking the deployment team as an example, as shown in Fig. 2, once the Central Orchestrator Agent assigns a task to the deployment team, the Planning and Requirement Agents use the Planning First mechanism to design an efficient execution plan. Next, the RAG Agent reviews the user-provided deployment rules and passes the results to the Execution

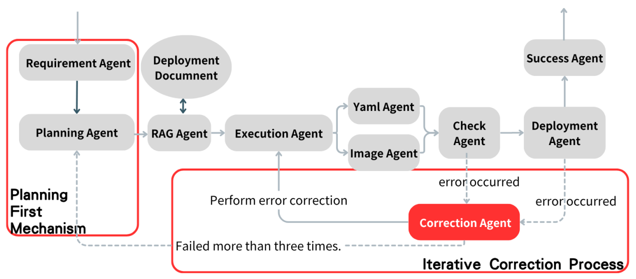

Fig. 2. The workflow of the deployment team of the processing layer.

Agent, which directs the YAML and Image Agents to generate the necessary artifacts. The Check Agent then verifies these outputs; if they fall short, the Correction Agent corrects the errors and returns to the Execution Agent. If the artifacts meet requirements, the Deployment Agent proceeds. Upon success, the Success Agent integrates deployment details (e.g., port and service information) and communicates the outcome to the Central Orchestrator Agent. In case of failure, errors are routed to the Correction Agent. Notably, if the Correction Agent corrects errors more than three times, the process returns to the Planning Agent for replanning to avoid an infinite cycle. The correction process mentioned above is our system mechanism: Iterative Correction Process.

C. System Mechanism

The system incorporates four key mechanisms:

1) **Optimized Task Allocation**: The Central Orchestrator Agent, highlighted in the red circle in Fig. 1, organizes tasks, coordinates team collaboration, and validates task accuracy to ensure an efficient workflow.
2) **Planning First**: Dedicated agents, highlighted in the left red frame in Fig. 2, conduct detailed task planning, providing clear direction and prioritization.
3) **Iterative Correction Process**: The Correction Agent, highlighted in the correct red frame in Fig. 2, detects errors and applies targeted corrections. If an error persists after three iterations, the system resets and replans from the Planning Agent.
4) **Modular Collaboration Architecture**: This architecture facilitates seamless agent expansion without requiring major structural modifications, enhancing scalability and adaptability. For instance, a User Agent can be added to conduct a final pre-deployment check before the Deployment Agent deploys the system to the cloud within the deployment team, as shown in Fig. 2.

III. EXPERIMENT RESULT

AutoDevFlow is built upon a modified AutoGen framework [4] to enable graphical input of workflows, achieving a fully automated end-to-end development process. Each LLM-integrated agent utilizes GPT-4o mini [5]. Our experiment aims to automatically deploy generated code to Google Kubernetes Engine (GKE) on Google Cloud Platform (GCP). The Retrieval-Augmented Generation (RAG) module employs a faiss [6], a library for efficient similarity search, to enable fast text retrieval and segmentation using GCP documentation as a knowledge base for accurate code generation.

To evaluate the system's feasibility and MAS problem-solving capabilities, we design three challenging tasks:

- Debug and deploy erroneous code
- Convert Python code to Java and deploy
- Determine the required environment and deploy automatically

The first task demonstrates high success due to the strong syntax and formatting accuracy of the LLM. However, the second and third tasks reveal challenges in handling cross-language conversions and managing different versions, where execution is often needed to identify issues.

Our Iterative Correction Process mechanism effectively addresses these challenges. The Correction Agent provides corrections, the Planning Agent re-strategizes on failed retries, and when necessary, the Central Orchestrator Agent reassigns tasks to the coding team for manual intervention, showcasing its strength in task distribution, cross-team coordination, and maintaining smooth deployment workflows.

IV. CONCLUSION

AutoDevFlow introduces a novel CI/CD automation approach using an LLM-powered multi-agent system. It addresses key challenges such as debugging, cross-language code conversion, and dynamic deployment through detailed planning, iterative correction, and modular collaboration. Experiments show it robustly corrects errors and efficiently distributes tasks, underscoring its effectiveness and potential for fully autonomous software development pipelines.

ACKNOWLEDGEMENTS

This work is sponsored by the National Science and Technology Council (NSTC) under the project NSTC 113-2222-E-008-002.

REFERENCES

[1] Sanjay Baitha, Vijayakumar Soorya, Osho Kothari, Shinu M. Rajagopal, and Niharika Panda, "Streamlining software development: A comprehensive study on ci/cd automation," in *International Conference on Sustainable Expert Systems (ICSES)*, 2024, pp. 1299–1305.
[2] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber, "MetaGPT: Meta programming for a multi-agent collaborative framework," in *The Twelfth International Conference on Learning Representations*, 2024.
[3] Junhee Lee, SungJoo Kang, and In-Young Ko, "An llm-driven framework for dynamic infrastructure as code generation," in *Proceedings of the 25th International Middleware Conference: Demos, Posters and Doctoral Symposium*, 2024, p. 9–10.
[4] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation framework," *ArXiv preprint arXiv:2308.08155*, 2023.
[5] OpenAI, "Gpt-4o-mini: A compact llm for multimodal tasks," 2024, Unpublished model. Available at: https://github.com/openai/gpt-4o-mini (accessed on 2024-03-07).
[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazar'e, Maria Lomeli, Lucas Hosseini, and Herv'e J'egou, "The faiss library," *ArXiv preprint arXiv:2401.08281*, 2024.