Final Project Report

A Collaborative Multi-Role LLM Agent Platform

Tao Tong (tt3010) Columbia University
Kuangyu Chen (kc3827) Columbia University
Jiacheng Gu (jg4874) Columbia University

1. Synopsis

This project transforms the OpenManus LLM agent framework into a distributed, collaborative platform in which specialized agent roles communicate and share context via a message-driven knowledge bus. Beyond throughput gains, the platform enables dynamic role assignment, fine-grained task decomposition, and cross-agent context sharing, mimicking a software engineering team's workflow. We will implement retrieval, planning, execution, validation, and integration agents across multiple nodes, evaluate functionality and performance, and compare against single-node OpenManus and sequential multi-step baselines. Contributions include a novel multi-role coordination architecture, a shared knowledge bus abstraction, and empirical evidence of improved task accuracy, consistency, and scalability.

2. System Design

This project is mainly designed for software developers. By imitating the role allocation of software development in real technology companies, it designs roles such as product manager, R&D, and testing, which can efficiently carry out complete project development. Whether it is a programmer who thinks that single-point agent development is too slow, or a programmer who feels that he is not competent for large-scale system development due to the limitation of LLM memory window, our project can help them.

Our project adopts a distributed framework, and multi-role multi-node parallel processing of subtasks can improve the throughput and response speed of automated tasks. Because the project has task decomposition, cross-agent context sharing, and automatic evaluation capabilities, it can improve the effectiveness of the project and greatly reduce the workload of manual tuning. In addition, developers can also carry out secondary development of our project according to their own needs and expand more agent nodes.

We will design a central Knowledge Bus (Kafka) to route messages and shared context

(JSON objects) between role-specific agents deployed in Docker containers. And a coordinator service will assigns tasks based on capability and workload. We split the overall job into at least five specialized agents:

Retrieval Agent: Gathers all the relevant documentation, code examples, and issue details needed to solve a problem.

Planner Agent: Breaks a big task into smaller, manageable steps so each agent knows exactly what to do.

Executor Agent: Writes code changes or patches and runs tests to ensure they work.

Validator Agent: Checks that the code changes actually fix the issue, flags any failures, and adds comments for improvement.

Integrator Agent: Takes approved fixes and merges them back into the project, creating pull requests for review.

The number of nodes for each agent will depend on workload. All task information and results are saved in shared storage, so agents can stay in sync and nothing gets lost. For each task, we start by listening for a user request. The Planning Agent then breaks that request into smaller tasks and dispatches them to the Retrieval Agent and the Execution Agents. The Retrieval Agent gathers materials as the Execution Agents build the first draft of the code. Once those materials arrive, the Execution Agents polish their work. In the final step, the Validation Agent tests the finished code while the Integration Agent writes documentation and sets up the environment. That completes one full cycle.

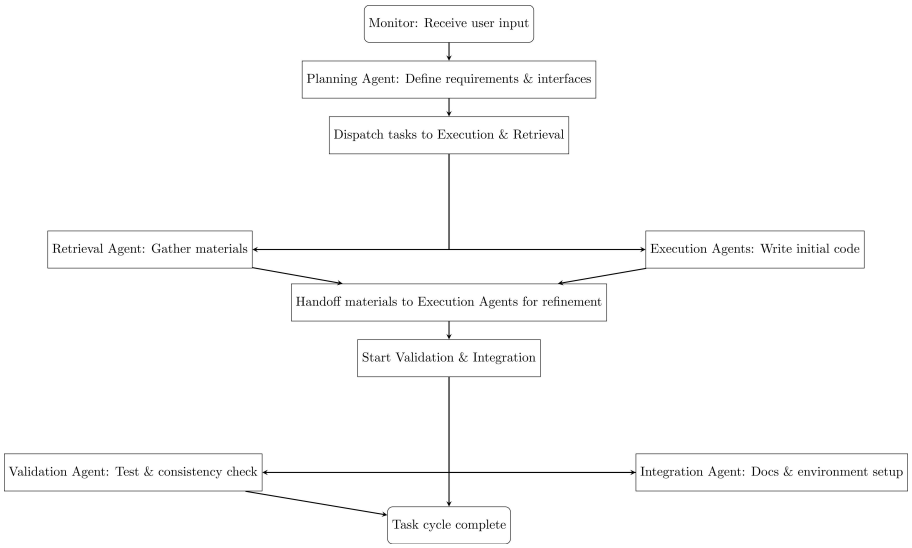

Figure 1 Workflow of Multi-agent System

3. Experiment

We mainly test the project code generation capability of our platform. My experiment uses 10 demand use cases generated by ChatGPT-4o[1] to test the single-point OpenManus[2] agent, the multi-agent platform we developed, and the mature commercial code generation framework Cursor[3] tool chain. The test steps for each system are the same. We use DeepSeek-V3[4] as our base LLM of all three projects. The same test case is used as the input query, and then the original code files are extracted from the target generation directory of each system for evaluation after the system completes the task autonomously. Before and after code generation, we did not perform any manual intervention except restarting the test when the project generation was interrupted due to network problems. We mainly evaluate the time used for code generation, the amount of generated code, and the quality of code.

The generation time calculation starts from the input query and ends when the code output is completed, covering all the steps required for the agent to process input, retrieve information, write code, generate files, etc. The amount of generated code is measured by the size of the target folder. When calculating, the environment files (such as package-lock.json) and image files that are not generated autonomously by the agent are excluded.

We use Claude 3.5 Sonnet[5] to assist in judging the quality of the generated code. We defined five indicators: front-end completeness, front-end robustness, back-end completeness, back-end robustness, and front-end and back-end consistency. Each indicator was assigned 5 points, and each test of each system was scored 50 points in total. In addition, the scoring was relative, and all scores were given by LLM based on the output code of the three systems at the same time, which truly reflected the difference in code quality between different systems. I have stored the test output and scoring details of the three systems in the GitHub repository, and readers who are interested can take a closer look.

Table 1 Test Result

Metric	Single-node Agent	Multi-node Agent	Cursor Tool
Average Time	10′ 09″ (609 s)	9′ 28″ (568 s)	4′ 32″ (272 s)
Average Score	22.4 / 50	37.9 / 50	34.5 / 50
Average Code Size	30.4 KB	68 KB	33.2 KB
Score per Second (Efficiency)	0.037 score / s	0.067 score / s	0.127 score / s
Score per KB (Density)	0.74 score / KB	0.56 score / KB	1.04 score / KB

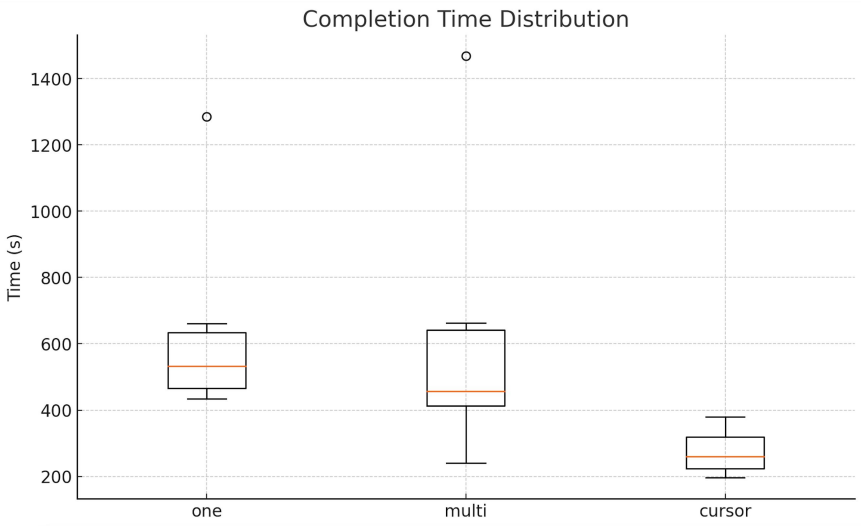

Figure 2 Time Distribution of Test

In terms of time, the single-node Agent takes an average of 10 minutes and 19 seconds, and the multi-node Agent takes an average of 9 minutes and 28 seconds. The multi-node is faster than the single-node because we divide some steps of code generation into different agents to make them parallel. However, the cursor tool is significantly faster than the other two, taking only 4 minutes and 32 seconds on average. This is because the cursor accesses the LLM API less times than the previous two methods, and does not encapsulate the web search tool, but generates code purely based on LLM feedback. In addition, since our agents need to communicate with each other, this also slows down our code generation.

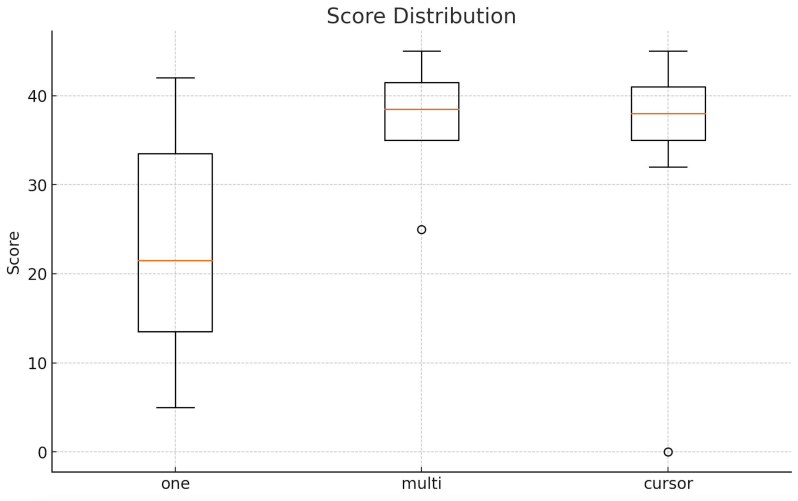

Figure 3 Score Distribution of Test

In terms of code quality, we can see that the average score of the single-node agent is only 22.4 points, the cursor tool is second, with an average score of 34.5, and the highest is the multi-node agent we developed, with an average score of 37.9. It is worth mentioning that the cursor tool had a file generation error during the test, resulting in empty output. We also made multiple attempts to ensure that it was not a network or personal host system problem. Therefore, it can be seen that our platform tools and

agents are more robust, and there are no output abnormalities.

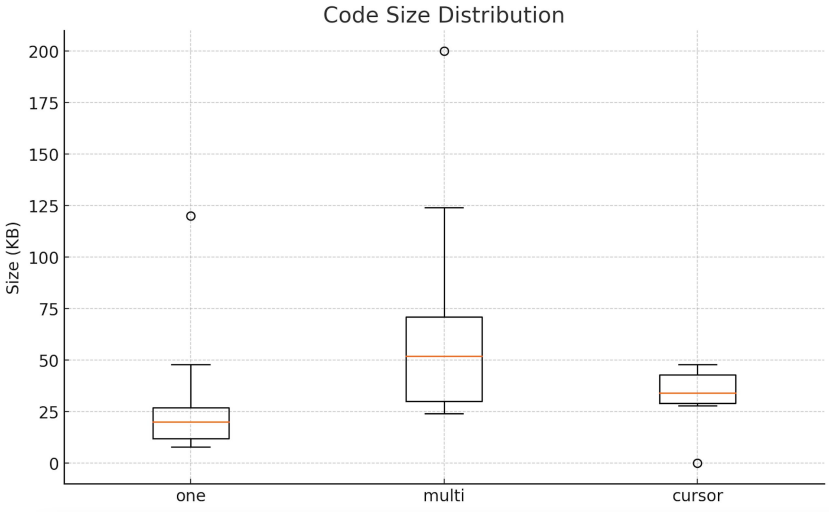

Figure 4 Code Size Distribution of Test

Finally, let's focus on the code size. The average code size generated by a single-node agent for a project is 30.4KB, and the cursor tool is slightly higher, reaching 33.2KB. The highest is our multi-node agent platform, reaching 68KB.

No matter from which perspective, our project performance is better than the original single-agent node system. Although the performance of the commercial code toolchain cursor is better than our agent platform from the perspective of throughput, scoring efficiency, and scoring density, our code quality is higher. This is due to the collaboration between our multiple agents, including macro-demand splitting and micro-data retrieval and code detail optimization.

4. Research Questions

In this section we will answer several questions we raised in my proposal.

1) How much does the distributed, multi-agent platform increase task throughput compared to single-node OpenManus and a sequential multi-step baseline?

In our test, the single-point task throughput is 5.91 tasks/hr, and the code throughput is 179.6KB/hr. The multi-agent platform task throughput is 6.34 tasks/hr, and the code throughput is 431.1KB/hr. This is a significant improvement.

2) Does enabling dynamic assignment of agent roles reduce end-to-end latency for completing an Issue automation task?

Yes. When the number of nodes increases, although the time cost required for communication increases, our tasks are parallelized, and the total time is reduced after testing.

3) How does fine-grained task decomposition affect overall system performance versus coarser decomposition?

From the execution time point of view, if the project is relatively simple, the coarse-grained effect will be better. This is mainly reflected in the performance of the cursor tool. Since it can output all the code with very few or even one LLM API call, it can achieve high-quality projects in a very short time. But for more complex projects, fine-grained task splitting will obviously make the system perform better. Because we cannot output a set of fully functional and robust code with one API call due to window limitations, subdividing tasks, calling the API multiple times, and using different tools to optimize the code can improve our code quality.

4) To what extent does sharing intermediate context over the knowledge bus improve the accuracy and consistency of final outputs compared to agents working in isolation?

In my development process, it brought about a 50% improvement. When I tried to let each agent perform its own tasks, even for the same query, the codes they generated were generally inconsistent, such as the inconsistency between the front-end code and the back-end code interface. So I needed them to share their own code information and interface information to ensure consistency.

5) How does increasing the number of compute nodes and corresponding agents impact throughput and resource utilization? Is there a point of diminishing returns?

During the development process, I found that for ordinary web projects, two agents generated by actual code are the best. When the number of nodes increases to three, file editing conflicts will occur. Of course, this can be solved by optimizing the prompt and using a more advanced llm api, but I plan to try these in the future.

6) What is the communication overhead introduced by the message-driven knowledge bus, and how does it trade off against the gains from distributed coordination?

Sending and listening to messages will incur overhead, but I will try to increase the degree of system parallelism, use computational cost to trade time cost, and reduce the number of synchronizations between agents. For example, in the early stage, we planned to wait for the retrieval agent to complete before starting to execute the execution agents, but the time overhead was too large, so we turned to the solution of executing the execution agents and the retirement agent in parallel.

Ablation Study

To prove our conclusion, we conduct a series of ablation experiments to isolate the impact of each major component in our platform. Specifically, we compare the following variants against the full system:

No Dynamic Assignment: roles are statically pre-assigned.

No Context Sharing: agents operate in isolation, without broadcasting their intermediate outputs.

Coarse-Grained Decomposition Only: the Planner issues a single large task per project.

Serial Execution: Execution Agents start only after the Retrieval Agent completes.

Table 2 Ablation Study Result

Variant	Task Throughput (tasks/hr)	Code Throughput (KB/hr)	Latency (s/task)
Full System (Baseline)	6.34	431.1	568
– No Dynamic Assignment	5.82 (– 8.2%)	402.5 (– 6.6%)	612 (+7.8%)
– No Context Sharing	4.21 (– 33.6%)	289.2 (– 32.9%)	820 (+44.4%)
– Coarse-Grained Only	5.10 (– 19.6%)	310.4 (– 28.0%)	710 (+25.1%)
– Serial Execution	5.45 (– 14.1%)	388.7 (– 9.8%)	650 (+14.5%)

5. Deliverables

GitHub repository (https://github.com/Tongs2000/Multi-Role-LLM-Agent-Platform.git) contains all source code and configurations (agents, Docker/Kafka), test cases and performance results, milestone documents (proposal, progress report, slides, final report), a README with reproduction instructions and demo video link.

6. Self-Evaluation

1) Tao Tong

In this project, I am mainly responsible for the overall framework design of the project and the writing and deployment of execution agents. Manus is an agent software that has been very popular in the past few months, and I am very interested in it. At the same time, the simplified version of the code OpenManus was open source, so I decided to learn the source code and use open source tools to build my own application. There are two points that I find challenging. One is that I don't know how to design the entire project. For example, which agents should I design and what kind of relationship should exist between each agent. The second is that I don' t know how to make our project have

advantages. There are many mature and even commercial agent projects on the market, such as Manus, the source of our project, and Cursor, which is widely used and well-received. It is actually very difficult to make a project stronger than them. For the first challenge, my solution is to try more. I tried different combinations of agents and different forms of communication, and finally formed the current project architecture. For the second challenge, my method is to do it first, find the comparative advantages, and then optimize it in a targeted manner. During the testing process, I initially found that there was still a gap between the performance of our project and the cursor. The execution speed of our project was relatively slow due to the many steps, which made me feel very frustrated. But I thought our features would be more than his, and the quality of output would be better. So I focused on modifying the logic of our execution agents and the architecture of the entire project, and finally beat Cursor in output quality. I think I did a good job in this project. I learned the current mainstream direction of LLM and the principles of agent application, and gained practical experience. In addition, our result is also a product that is easy to use in programmers' daily development, and I am very proud of it.

2) Kuangyu Chen

In this project, I wrote a planning agent and a retrieval agent. I encountered many problems and challenges during the development process. When I was writing the planning, I was struggling with how to design the topic for communication in Kafka. Should all agents listen to my topic or should I only send messages to certain agents? Finally, I thought about it and decided to send messages to only two of the agents, the retrieval agent and the execution agent. Because sending messages to the remaining two seems to have no effect, and the extra topics will cause them to execute very slowly. Then there were a lot of bugs when writing the retrieval agent. For example, when my agent wakes up the browser, it often visits some invalid websites, such as Youtube, because I don’ t have a tool to read videos. But in the process of solving one bug after another, I learned a lot. I know that it is not difficult for an agent to be fully functional, but it is very difficult to be robust. This requires developers to consider emergencies in all environments and provide fault tolerance and bottom-line measures.

3) Jiacheng Gu

I wrote the project's integration agent and validation agent. I learned the meaning of teamwork and many new technologies during the project. Our project architecture is like a real project team, with product managers, programmers, and test engineers. I also used my previous internship experience in the company to handle my tasks. I recalled the previous project process and wrote some good steps into our project, so that these last two agents can ensure that our project output can operate normally. It is well known that LLM often has hallucinations, which is specifically reflected in the fact that the output of our project sometimes does not come as expected, and it is very challenging to solve these problems. All I have to do is to reorganize the output and ensure the integrity and robustness of the output, and I have indeed done it through my own efforts. I think I will use this project to assist my development in my future studies and career.

7. Reference

[1] https://openai.com/index/hello-gpt-4o/

[2] https://github.com/FoundationAgents/OpenManus.git

[3] https://github.com/civai-technologies/cursor-agent.git

[4] https://api-docs.deepseek.com/

[5] https://www.anthropic.com/news/claude-3-5-sonnet