
AGENT-AS-A-SERVICE BASED ON AGENT NETWORK

Yuhan Zhu
 School of Computer Science
 Wuhan University
 Wuhan, China
 zhuyuhan2333@whu.edu.cn

Haojie Liu
 School of Computer Science
 Wuhan University
 Wuhan, China
 1d1ornd@whu.edu.cn

Jian Wang*
 School of Computer Science
 Wuhan University
 Wuhan, China
 jianwang@whu.edu.cn

Bing Li
 School of Computer Science
 Wuhan University
 Wuhan, China
 bingli@whu.edu.cn

Zikang Yin
 School of Computer Science
 Wuhan University
 Wuhan, China
 2020302111233@whu.edu.cn

Yefei Liao
 School of Computer Science
 Wuhan University
 Wuhan, China
 1206232012@qq.com

June 11, 2025

ABSTRACT

The rise of large model-based AI agents has spurred interest in Multi-Agent Systems (MAS) for their capabilities in decision-making, collaboration, and adaptability. While the Model Context Protocol (MCP) addresses tool invocation and data exchange challenges via a unified protocol, it lacks support for organizing agent-level collaboration. To bridge this gap, we propose Agent-as-a-Service based on Agent Network (AaaS-AN), a service-oriented paradigm grounded in the Role-Goal-Process-Service (RGPS) standard. AaaS-AN unifies the entire agent lifecycle, including construction, integration, interoperability, and networked collaboration, through two core components: (1) a dynamic Agent Network, which models agents and agent groups as vertexes that self-organize within the network based on task and role dependencies; (2) service-oriented agents, incorporating service discovery, registration, and interoperability protocols. These are orchestrated by a Service Scheduler, which leverages an Execution Graph to enable distributed coordination, context tracking, and runtime task management. We validate AaaS-AN on mathematical reasoning and application-level code generation tasks, which outperforms state-of-the-art baselines. Notably, we constructed a MAS based on AaaS-AN containing agent groups, Robotic Process Automation (RPA) workflows, and MCP servers over 100 agent services. We also release a dataset containing 10,000 long-horizon multi-agent workflows to facilitate future research on long-chain collaboration in MAS.

1 Introduction

With the rise of large model-based AI agents, Multi-Agent Systems (MAS) are gaining traction for their potential in automated decision-making, collaborative task execution, and adaptability to complex environments [1]. Meanwhile, the rapid evolution of computing paradigms—such as cloud, edge, and service-oriented computing—is driving software systems toward greater connectivity, intelligence, and service modularity [2]. However, current MAS implementations still largely rely on agent workflows to coordinate collaboration, lacking end-to-end automation across agent construction, integration, interoperability, and collaboration.

To address challenges such as complex tool invocation, data silos, and inconsistent data exchange formats, the Model Context Protocol (MCP) [3] provides a unified communication interface for Large Language Model (LLM) applications. It supports real-time context exchange, significantly improving model scalability and response relevance, and enabling seamless integration of a wide range of tools into agents. However, while MCP facilitates tool integration, it does not

*Corresponding author. Email: jianwang@whu.edu.cn

define collaboration paradigms among agents. As the number of agents rapidly increases, effectively organizing multi-agent systems to fulfill user needs has become a more pressing challenge. Agent2Agent (A2A) [4] further establishes a collaboration paradigm for agents, enabling multi-agent collaboration across heterogeneous agent frameworks through the A2A protocol. It also provides support for task state management, agent capability discovery, and data security. However, challenges remain in automatically discovering appropriate agents based on user needs and organizing them for effective collaboration. Making agents truly plug-and-play services that support scalable multi-agent systems and seamlessly integrate with existing software infrastructures still lacks standardized, service-oriented interoperability protocols.

In this paper, we propose Agent-as-a-Service based on Agent Network (AaaS-AN), a service-oriented agent paradigm built upon the RGPS standard [5]. AaaS-AN aims to unify the entire lifecycle of agents—including construction, integration, interoperability, and networked collaboration—under a service-based framework, enabling plug-and-play multi-agent systems. AaaS-AN consists of two core components: the Agent Network and service-oriented agents, collaborated by the Service Scheduler using interoperability protocols. The Agent Network provides the foundation for agent construction and networked collaboration. It models agents and agent groups as vertexes, and dynamic routing as edges, forming a distributed, self-organizing network. The Service-Oriented Agent component treats agent groups as the basic units and supports distributed agent service execution, service registration and discovery, and standardized agent communication protocols. The Service Scheduler coordinates agent service execution and introduces an Execution Graph as a runtime protocol to handle context management, multi-agent collaboration, task state tracking, and overall system operations in distributed multi-agent systems.

We evaluate the capabilities of AaaS-AN-based multi-agent systems in mathematical reasoning and application-level code generation tasks. Furthermore, we integrate a large-scale multi-agent system composed of over 100 agent services, including agent group services, RPA workflows, and MCP Servers. We also release a dataset containing 10,000 long-chain multi-agent collaboration flows, and investigate the challenges and potential solutions revealed within these long-chain flows, contributing to future research on long-horizon multi-agent collaboration.

Our main contributions are summarized as follows:

- We propose AaaS-AN, a service-oriented agent paradigm that enables each agent to operate as a vertex within an Agent Network and collaborate as agent services to more effectively fulfill user requirements.
- We design and implement a plug-and-play framework for service-oriented agents, including service registration and discovery, interoperability protocols, and the service scheduler. This allows agents to be accessed directly as services by users. AaaS-AN outperforms baselines on mathematical reasoning and application-level code generation tasks.
- We integrate a large-scale multi-agent system consisting of over 100 agent services—including agent groups, RPA workflows, and MCP Servers. We release a dataset of 10,000 long-chain multi-agent collaboration flows. This provides a foundation for research on multi-agent collaboration in long-chain flows.

2 Related Work

2.1 Multi-agent System

In the context of web navigation tasks, language model-based agents have been increasingly utilized. Gur et al.[6] introduced WebAgent, which automates web navigation by utilizing a pre-trained model to extract HTML information. This is then combined with a code generation model to produce the necessary operation code, assisting users in automating the process from requirements to web actions. Additionally, Park et al. [7] developed an interactive sandbox environment called "Stanford Town" to study whether 25 instantiated agents, based on large language models, could reliably simulate human behavior. Their research concluded that the key components of the agent architecture—observation, planning, and reflection—are crucial in determining the credibility of the agent's behavior.

In the field of embodied intelligence, to address the issue of data scarcity, Wang et al. [8] further expanded the "Stanford Town" into real-world environments, utilizing robots as carriers. This expansion overcame the challenges posed by the high cost of collecting real-world data, which had previously hindered the exploration of scaling laws in embodied intelligence. This development marked a significant step from simulation to reality. Additionally, Liu et al. [9] proposed a dialogue-based audio-visual embodied intelligence navigation framework, CAVEN, where agents solve navigation tasks through interactions with humans or a pre-set oracle.

In the field of software engineering, Qian et al.[10] introduced the multi-agent chain interaction framework ChatDev, which explores the fully automated process from requirements analysis to application-level code generation. This approach effectively saves significant manpower costs and demonstrates certain capabilities in solving complex tasks in

the full automation of software engineering workflows. Due to the diversity of agent frameworks and the complexity of multi-agent collaboration, Chen et al.[11] proposed the Agent Internet, which introduces agent registration and discovery for organizing complex and heterogeneous multi-agent systems. They employed state machines to control multi-agent collaboration, marking the initial steps toward realizing multi-agent systems through service-oriented approaches. Wang et al. [12] explored the potential of using LLMs for user behavior simulation in recommendation systems. By designing user profiles, memory, and behavior modules, each agent can interact with the recommendation system via one-on-one chats or one-to-many social broadcasts. The AlpacaFarm framework [13] studied a human feedback simulator based on LLMs, which is 45 times cheaper than crowdsourced human feedback. The researchers also found that reward-learning-based methods significantly outperformed supervised fine-tuning (SFT) and showed a high degree of consistency with human feedback in the learning process. The AgentsPLC[14] method achieves automatic generation of PLC code through a multi-agent network. MetaGPT[15] uses a shared information pool communication structure, where agents interact with the shared message pool in a publish-subscribe manner.

However, existing multi-agent systems primarily rely on natural language text for communication, which fails to meet the structured requirements of service deployment. The toolchains that agents depend on (e.g., APIs, service interfaces) and the interactions between agents need to be further standardized and unified through a service-oriented approach. This will enable the practical application of these systems in specific scenarios and ensure seamless integration with existing software and hardware systems. For user interactions with multi-agent systems, agents must rely on services (a clearly defined set of functions) to engage with humans. Therefore, efficiently and reliably transmitting structured information to empower agent networks to accurately fulfill user requirements will be a key research focus in service-oriented agent systems.

2.2 Agent Paradigm

The architecture of intelligent agents encompasses several key components: context-aware memory [7], planning [16], role-playing [17], and tool utilization [18]. Well-designed architectures significantly enhance agent capabilities. Chain of Thought [19] (CoT) guides LLMs in "stepwise reasoning", decomposing complex tasks into smaller subtasks to improve reasoning performance. An enhanced CoT technique, self-consistency [20], replaces simple greedy decoding by first sampling a diverse set of reasoning paths rather than selecting the most straightforward route, then selecting the most consistent answer through marginalizing these sampled reasoning paths. The ReAct [21] framework integrates reasoning with action, expanding LLMs' action space to enable natural language-based reasoning generation and environmental interaction. Reflexion [22] incorporates reinforcement learning mechanisms to endow agents with dynamic memory and self-reflective capabilities, thereby improving decision-making efficiency and reasoning performance. HuggingGPT [23] framework employs LLMs as task planners, automatically selecting optimal models from HuggingFace platforms based on task requirements and generating final responses through execution results, substantially enhancing LLMs' scalability and adaptability in complex tasks. When conducting self-evaluation, LLMs often exhibit excessive confidence or high randomness, generating stubborn or inconsistent feedback that undermines reflection effectiveness. To address this limitation, the Self-Contrast [24] method provides LLMs with multiple perspectives to mitigate stubborn biases.

3 AaaS-AN

3.1 Overview

LLM-based agent systems often struggle with the limitations of single-agent capabilities, making it difficult to fully address complex task requirements. While multi-agent collaboration offers a potential solution, vague role boundaries frequently hinder effective cooperation and task execution.

To address these challenges, we propose AaaS-AN, a Role-Goal-Process-Service (RGPS)-driven architecture composed of two main components: the Agent Network and service-oriented agents. The Agent Network models agent roles based on domain knowledge, forms agent groups centered around specific goals, and employs agent routes to support complex execution flows. This network-based organization enables structured and scalable multi-agent collaboration. Service-oriented agents encapsulates agent groups as service units. During task execution, a scheduler coordinates these services via the interoperation protocol, while a execution graph is maintained to store and isolate structured contexts across tasks. Built upon AaaS-AN, it supports full-process automation for multi-agent systems, including agent construction, integration, execution, and collaboration, as illustrated in Figure 1.

3.2 Agent Network

The agent network is dynamically structured, where both individual agents and agent groups serve as vertexes, and routes form the edges. Each vertex is accessible via routes, enabling decentralized communication. Agent groups

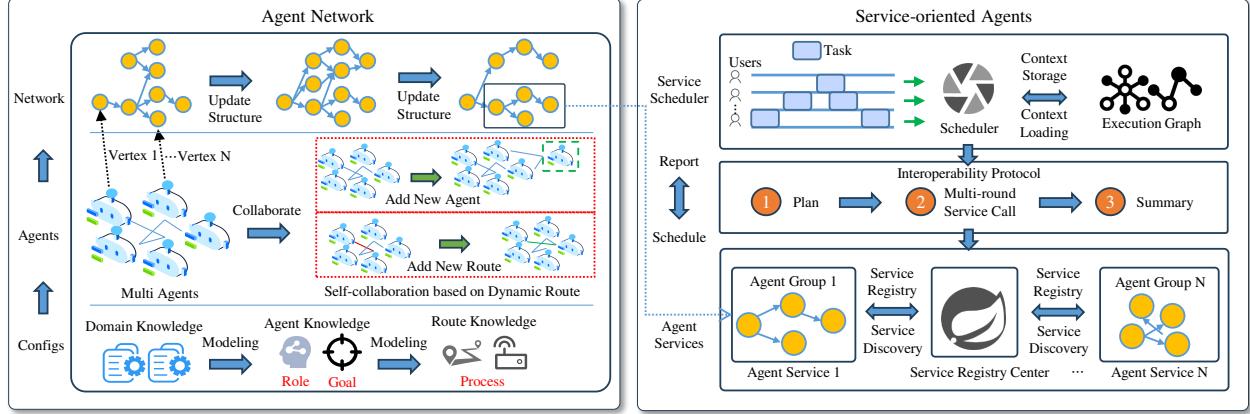


Figure 1: The overview framework of AaaS-AN.

encapsulate multiple agents and can be treated as abstract vertexes, supporting recursive invocation across the network. Any vertex can receive user tasks, triggering distributed execution and yielding aggregated outputs. The network supports concurrent task execution with isolated contexts, while allowing for context sharing at specific vertexes under configurable policies. Both the topology of vertexes and the routing edges are dynamically adjustable, enabling adaptive reconfiguration based on runtime requirements.

3.2.1 Agent Role

In the RGPS requirements meta-model, a role serves as the knowledge foundation for modeling individual agents in the agent network. The resulting agent knowledge (denoted as A) comprises six key components:

$$A = \{A^n, A^d, A^p, A^i, A^o, A^c\} \quad (1)$$

where A^n denotes the name of the agent role, A^d denotes the description, A^p denotes the system prompt, A^i and A^o denotes the structured input and output parameters, and A^c denotes the logic code. The role name summarizes the core functionality of an agent, for example, in a code generation task, there may be a "programmer" agent specifically responsible for writing code. The role description provides a textual explanation of the agent's responsibilities and functional scope.

The system prompt is input to the large language model, enhancing the alignment of the agent and maintaining consistency across the task lifecycle. The structured input and output parameters serve as both detailed descriptions and operational constraints for the agent, which define the context the agent requires and produces, and also function as runtime checks that reinforce causal relationships between upstream and downstream agents during multi-agent collaboration, as well as providing valuable context to chat with LLMs.

Finally, the agent logic code transforms input parameters into output parameters, completing the functional definition of the agent.

3.2.2 Agent Group

Agent Group is the set of goal-oriented agents for better collaboration. Based on goal modeling in the RGPS meta-model, the construction of goal-oriented agent groups relies on explicit goal decomposition and collaboration mechanisms. Within a group, each agent assumes a specific role, and the system-level objective is achieved through a structured goal decomposition strategy.

Goal Decomposition: High-level global objectives are decomposed into multiple sub-goals, and agent roles are assigned or organized around each sub-goal.

Role Assignment and Group Organization: Based on the nature of each sub-goal and the knowledge representation of agent roles, agents are assembled into a group such that each agent can effectively perform its designated task. The agent group is centered around a system-level goal, and its structured knowledge (denoted as G) consists of the following components:

$$G = \{G^n, G^d, G^p, G^i, G^o, G^A\} \quad (2)$$

where G^n denotes the name of the group, G^d denotes the goal description, G^p denotes the system prompt of the group, G^i and G^o denotes the structured input and output parameters to align task context with the goal of the group, and G^A denotes the agents within the group. The agent group name summarizes the core functionality associated with a system-level goal. The goal description is a textual summary of the overall responsibilities and intended objectives of the group. The group prompt is provided to the language model as input to enhance consistency in task execution and collaboration among all agents within the group. The input and output parameters define the data dependencies and serve as operational constraints at the group level, allowing the agent group, as a knowledge unit, to collaborate further with other agent groups or individual agents. In code generation tasks, there may be a dedicated agent group responsible for analyzing requirements based on user tasks, including a requirement refinement agent and a programming language selection agent, each responsible for translating high-level requirements into executable development tasks.

3.2.3 Agent Route

The agent route is designed to enable knowledge-based modeling of multi-agent collaborative processes, based on "Process" in RGPS. By facilitating context exchange and coordinated decision-making, it allows vertexes within the agent network to collaborate effectively. The network supports multiple types of routes, including "HARD", "SOFT", and "EXT", enabling both intra-group and inter-group self-collaboration among agents.

HARD Route: Hard route is designed to define and preserve the structural collaboration patterns within the agent network. On the one hand, certain agents require predefined interaction sequences to accomplish specific tasks. These can be encoded as hard routes using prior domain knowledge, injecting fixed workflows as structured knowledge into the network. On the other hand, during the dynamic evolution of the agent network, a large number of execution trajectories are generated. By analyzing the correlation between execution traces and task success, recurring structural patterns can be identified and retained through hard routes, enabling the network to accumulate and reuse effective collaboration structures.

SOFT Route: Soft route is designed to support self-collaboration structures within an agent group. When an agent group vertex receives a group-level task, it must dynamically organize the constituent agents based on task-specific requirements. Soft route enables this collaboration by organizing intra-group agents in a flexible and adaptive manner, facilitating effective collaboration aligned with the task.

EXT Route: Extended route is designed to support self-collaboration across agent groups. During task execution, an individual agent group may not always possess sufficient capabilities or context to fulfill the task independently. In such cases, extended route enables the agent group vertex to proactively discover and collaborate with other vertexes in the agent network. It allows the system to dynamically expand its collaboration scope beyond the original group, facilitating task completion.

Based on the above three types of route mechanisms, it is possible to effectively integrate fixed workflows enriched with prior knowledge and dynamically coordinated agent interactions. This integration facilitates efficient collaboration among vertexes in the agent network, ultimately enabling more effective achievement of user goals.

4 Experiments

In this section, we conducted a systematic evaluation of AaaS-AN's performance across three distinct domains: mathematical reasoning, application-level code generation, and real-world long-chain workflow tasks. The framework's capabilities were rigorously examined through quantitative benchmarking, qualitative case studies, and comparative analysis against state-of-the-art baselines in each respective task category.

4.1 Mathematical Reasoning

For comprehensive evaluation of our multi-agent framework's generalization capacity on mathematical reasoning tasks, we constructed a balanced test suite comprising 72 problems per category (504 total) from the MATH benchmark. This stratified sampling approach enables systematic analysis of performance variations across mathematical domains.

Experimental Setups: We benchmark against three state-of-the-art multi-agent frameworks:

- MetaGPT: The first framework to introduce workflows through meta-programming, enabling agents with human-like domain expertise to verify intermediate results and reduce errors.

- AutoGen: A popular multi-agent framework proposed by Microsoft, capable of decomposing and resolving complex tasks via multi-round dialogues. The framework demonstrates strong performance and generalization capabilities across multiple domains.
- MACM: An advanced multi-agent framework for solving complex mathematical problems, which uses condition mining to solve mathematical problems, expected to improve the reasoning capabilities of large language models on advanced mathematical problems.

To enhance assessment reliability, we use state-of-art LLM for semantic consistency scoring between agent outputs and reference solutions instead of exact matching, using the prompt template below:

```

system:
You are an experienced mathematics teacher with a strong grasp of logical reasoning and precise calculations,
capable of quickly identifying the core of mathematical problems and evaluating the consistency between
answers and solution processes.
User:
Here is the math problem: {problem} with standard solution: {solution} The student's answer is: {answer}
Please check whether the answer is correct or not. Please answer in True or False directly without any additional
explanations.

```

Model	Method	Accuracy	Token Cost	Time(s)
qwen2.5-32b-instruct	MACM	35.13%	4429.05	44.26
	MetaGPT	57.52%	2264.17	37.91
	AutoGen	57.85%	5688.95	40.38
	AaaS-AN	63.62%	<u>2297.42</u>	41.77

Table 1: The performance on the mathematical reasoning tasks of AaaS-AN and other baselines. The top scores are in bold, with the second-highest underlined in Quality. Given that the token consumption and time consumption are comparable to those of other state-of-the-art models, AaaS-AN has significantly improved the accuracy(5.77%).

Results and Analysis: The experimental results are shown in Table 1. In the mathematical reasoning task, AaaS-AN achieved a significantly higher accuracy rate than other baseline models, with an average improvement of 5.77%. Meanwhile, the Token consumption and time consumption of AaaS-AN when solving problems were very close to those of the current state-of-the-art methods. In an analysis of 504 randomly sampled problems, the average Token consumption of AaaS-AN was only 1.47% higher than that of the best method, and the average time was 10.18% longer. Considering the network instability that exists when calling the large language model API, we believe that a 10% deviation in time is within a reasonable range.

Compared with the mainstream multi-agent frameworks currently in use, the information transmission among agents in AaaS-AN is more efficient and precise, thanks to the advantage of structured context. As a result, AaaS-AN is able to maintain excellent problem-solving efficiency and enhance the success rate of problem resolution. Meanwhile, the low accuracy rate of MACM may be attributed to the specialization of its multi-agent collaboration approach for a certain type of problem. In our dataset, which emphasizes problem diversity, a multi-agent framework aiming to solve these mathematical problems needs to possess better generalizability. This is where AaaS-AN excels.

4.2 Application-level Code Generation

To evaluate the capability of AaaS-AN in zero-shot complex generation tasks, we conduct experiments on two benchmarks focused on application-level code generation: SRDD and ProgramDev. SRDD, proposed by ChatDev, comprises 1,200 software task prompts that are meticulously categorized into five main domains: Education, Work, Life, Game, and Creation. Each domain is further divided into 40 subcategories, with each subcategory containing 30 unique task prompts. ProgramDev, on the other hand, targets the discovery of deep, often-overlooked failures in multi-agent collaborative coding that may degrade application-level code generation performance. This benchmark includes 30 lightweight programs inspired by classic games, covering a diverse range of interaction logic and functional implementations.

Experimental Setups: We benchmark AaaS-AN against several state-of-the-art multi-agent systems as well as leading large language models used as single-agent systems.

- ChatDev[25]: A chat-powered software development framework that integrates LLM-based agents with diverse social roles, enabling them to autonomously collaborate and generate comprehensive solutions through multi-agent interaction.
- GPTSwarm[26]: A LLM-based agents as computational graphs, it implements operations as nodes and information flow between operations as edges to solve the problems by recursively combined graphs.

We adopted Quality, Token, Cost, and Time to measure the performances. Quality is defined as the product of task success rate, code completion rate, and executability, providing a comprehensive measure of the accuracy, completeness, and usability of multi-agent systems in application-level code generation tasks. AaaS-AN and all baseline methods are instantiated following the software agent team structure used in ChatDev, including agents for requirement analysis, coding, code review, testing, and documentation editing, in order to evaluate their performance.

Results and Analysis: AaaS-AN exhibits consistently superior performance on both the SRDD and ProgramDev benchmarks. In comparison with state-of-the-art multi-agent frameworks including ChatDev and GPTSwarm, AaaS-AN achieves notable performance gains across various foundation models. In comparison to ChatDev, one of its key advantages lies in its ability to significantly reduce token consumption. This efficiency comes from the strategy of AaaS-AN to eliminate redundant historical messages and superfluous "chat" rounds. The benefit is especially pronounced in code generation tasks, where large language models often struggle to pinpoint issues due to minor variations in lengthy and repetitive outputs. To address this, AaaS-AN leverages a self-coordination mechanism that proactively initiates reflective reasoning when no substantive changes are detected in the generated code. This process effectively filters out unproductive code generation attempts, thereby enhancing overall performance while minimizing unnecessary token usage caused by ineffective agent actions.

In comparison with state-of-the-art large language models (as a single agent), multi-agent collaboration offers an effective solution to the common issues of incomplete code snippets and ill-structured logic in the code generation process. By incorporating iterative interactions that integrate code review and testing within the execution environment, such collaboration significantly enhances the quality and reliability of the generated code. Furthermore, AaaS-AN is particularly effective at orchestrating multi-agent collaboration even when individual models demonstrate suboptimal performance. It achieves this while substantially reducing computational and token-related costs, thus improving overall efficiency and scalability in resource-constrained settings.

AaaS-AN enables autonomous coordination among multiple agents within the agent network to achieve improved performance. Its structured context allows large language models to more accurately infer agent intentions from shorter prompts, thereby reducing token consumption without sacrificing task effectiveness.

4.3 AaaS-AN Based Agent Services and Long-chain Flows

To further evaluate the capability of AaaS-AN in addressing general tasks, we collect a data set that includes a large number of intelligent agent services and processes and conduct an analysis of these data from the perspective of task, protocol, and service.

4.3.1 Task Perspective

Our tasks and subtasks are divided into several predefined types, including **New**, **Running**, **Success**, and **Fail**. The **New** type indicates that the task has only completed its initial creation. The **Running** type means that the task is in a pending state and has not been completed or timed out. The **Success** and **Fail** types, respectively, indicate whether the task was successfully completed or not.

From the perspective of task status distribution, the number of successful tasks reaches the highest level, indicating that AaaS-AN is capable of reliably accomplishing tasks in most cases, which reflects its stability. To ensure reliable task completion, more resources are typically required. Therefore, tasks in the successful status tend to consume more time and tokens. In contrast, **Fail** tasks often follow incorrect or even looping execution paths, resulting in a longer chain-flow.

4.3.2 Protocol Perspective

AaaS-AN demonstrates good scalability, supporting various types of tasks such as those involving Agents and RPAs, while maintaining a high success rate in handling them.

Benchmark	Method	Model	Quality	Token	Cost	Time(s)
SRDD	Qwen3-32b	-	0.463	13707.569	0.056	128.606
	GPT-4.1-Mini	-	0.747	14294.313	0.086	125.707
	Claude-3.7-Sonnet	-	0.584	<u>45442.749</u>	0.487	314.538
	GPTSwarm	GPT-3.5-Turbo	-	-	-	-
		Qwen2.5-32b-Instruct	-	-	-	-
		Deepseek-V3	-	-	-	-
	ChatDev	GPT-3.5-Turbo	0.839	21044.765	0.103	450.903
		Qwen2.5-32b-Instruct	0.891	29021.105	0.089	537.933
		Deepseek-V3	0.854	<u>59152.232</u>	<u>0.202</u>	1203.494
	AaaS-AN	GPT-3.5-Turbo	0.872	9037.320	0.044	279.534
		Qwen2.5-32b-Instruct	<u>0.899</u>	11946.271	0.037	357.782
		Deepseek-V3	0.900	23766.519	0.081	<u>887.330</u>
ProgramDev	Qwen3-32b	-	0.452	15199.401	0.091	269.085
	GPT-4.1-Mini	-	0.750	14411.400	0.121	108.444
	Claude-3.7-Sonnet	-	0.838	21400.035	0.366	326.691
	GPTSwarm	GPT-3.5-Turbo	-	-	-	-
		Qwen2.5-32b-Instruct	-	-	-	-
		Deepseek-V3	-	-	-	-
	ChatDev	GPT-3.5-Turbo	0.767	24740.100	0.123	534.935
		Qwen2.5-32b-Instruct	0.716	<u>32189.967</u>	0.095	555.670
		Deepseek-V3	0.803	44078.333	<u>0.148</u>	2145.833
	AaaS-AN	GPT-3.5-Turbo	0.774	10362.100	0.052	266.242
		Qwen2.5-32b-Instruct	0.900	14562.933	0.043	312.528
		Deepseek-V3	<u>0.870</u>	16990.633	0.057	<u>1163.419</u>

Table 2: The performance on the application-level code generation of SRDD and ProgramDev benchmark. The top scores are in bold, with the second-highest underlined in Quality.

Task Status	Number	Average Length of Chain Flows	Average Time	Average Token Cost
New	919 (10.9%)	0.0	0.0	0.0
Running	363 (4.3%)	3.6	57.1	1918.0
Success	4518 (53.7%)	4.5	230.2	2254.6
Fail	2620 (31.1%)	5.6	102.8	759.3

Table 3: Overview of Tasks

Subtask	Status				
	Total	New	Running	Success	Fail
Agent	44200	6870 (15.5%)	0 (0.0%)	28323 (64.1%)	9007 (20.4%)
RPA	3055	4 (0.1%)	191 (6.3%)	2433 (79.6%)	427 (14.0%)

Table 4: Scale of Subtasks

Vertex	Part			
	Number	Average Time	Average Token Cost	Success Rate
Agent	34947	19.8	710.8	97.1%
RPA	2650	144.4	-	96.5%

Table 5: Protocol of Vertices

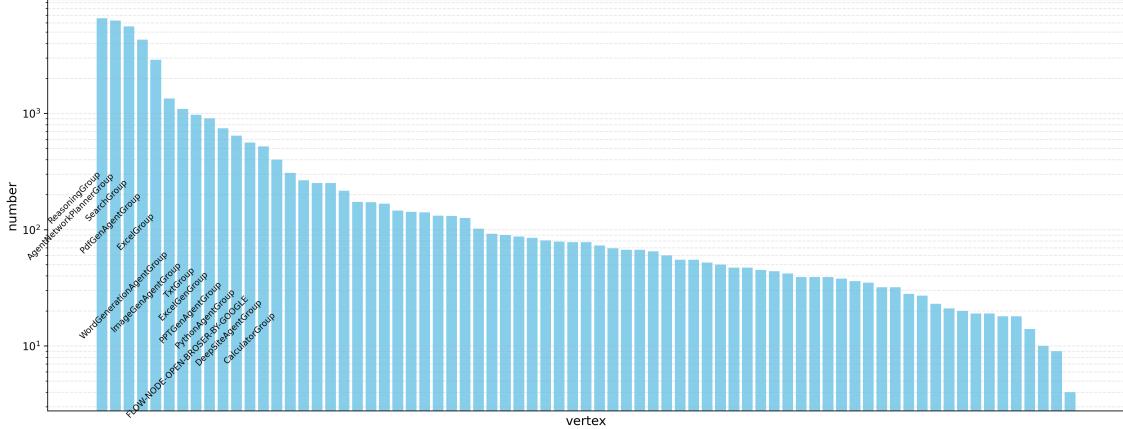


Figure 2: Distribution of Vertices

4.3.3 Service Perspective

In AaaS-AN, vertexes are the fundamental units of service. During task data processing, it has been identified that the vertexes exhibit a long-tail distribution pattern. Although the uncommon vertexes in limited statistical samples generally have minimal impact on task execution performance, their wide variety and accumulated volume grant them significant value within AaaS-AN. The distribution of vertexes is illustrated in Figure 2.

The agent services(vertexes) scheduled by AaaS-AN play different roles in the task execution process. We define the contribution of a service as the average similarity between its input and the final output of the task. We also collect statistics on the success rate and contribution of each service to identify those with strong performance and those requiring further improvement, thereby providing insights for future enhancements of AaaS-AN.

5 Conclusion

We proposed AaaS-AN, a service-oriented framework for organizing large-scale multi-agent systems. Built upon the RGPS standard, AaaS-AN models agents and agent groups as dynamic network vertexes and integrates service discovery, registration, and execution through a unified scheduling mechanism. Our experiments on mathematical reasoning and application-level code generation demonstrate that AaaS-AN outperforms competitive baselines. We further validate its scalability via a deployment of over 100 agent services, including agent groups, RPA workflows and MCP servers. To facilitate future research, we release a dataset of over 10,000 multi-agent flows for evaluating long-chain tasks.

References

- [1] Weiqiang Jin, Hongyang Du, Biao Zhao, Xingwu Tian, Bohang Shi, and Guang Yang. A comprehensive survey on multi-agent cooperative decision-making: Scenarios, approaches, challenges and perspectives. *CoRR*, abs/2503.13415, 2025.
- [2] Hung Du, Srikanth Thudumu, Rajesh Vasa, and Kon Mouzakis. A survey on context-aware multi-agent systems: Techniques, challenges and future directions. *CoRR*, abs/2402.01968, 2024.
- [3] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (MCP): landscape, security threats, and future research directions. *CoRR*, abs/2503.23278, 2025.
- [4] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions, 2025.
- [5] Jian Wang, Zaiwen Feng, Jia Zhang, Patrick C. K. Hung, Keqing He, and Liang-Jie Zhang. A unified rgps-based approach supporting service-oriented process customization. In *Web Services Foundations*, pages 657–682. Springer, 2014.

- [6] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023.
- [7] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*, pages 1–22, 2023.
- [8] Hanqing Wang, Jiahe Chen, Wensi Huang, Qingwei Ben, Tai Wang, Boyu Mi, Tao Huang, Siheng Zhao, Yilun Chen, Sizhe Yang, et al. Grutopia: Dream general robots in a city at scale. *arXiv preprint arXiv:2407.10943*, 2024.
- [9] Xiulong Liu, Sudipta Paul, Moitreya Chatterjee, and Anoop Cherian. Caven: an embodied conversational agent for efficient audio-visual navigation in noisy environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 3765–3773, 2024.
- [10] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6(3), 2023.
- [11] Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv preprint arXiv:2407.07061*, 2024.
- [12] Lei Wang, Jingsen Zhang, Hao Yang, Zhi-Yuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Hao Sun, Ruihua Song, et al. User behavior simulation with large language model-based agents. *ACM Transactions on Information Systems*, 43(2):1–37, 2025.
- [13] Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069, 2023.
- [14] Zihan Liu, Ruinan Zeng, Dongxia Wang, Gengyun Peng, Jingyi Wang, Qiang Liu, Peiyu Liu, and Wenhui Wang. Agents4plc: Automating closed-loop plc code generation and verification in industrial control systems using llm-based agents. *arXiv preprint arXiv:2410.14209*, 2024.
- [15] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.
- [16] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.
- [17] Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 2023.
- [18] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.
- [19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [20] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [22] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.
- [23] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

- [24] Standard Self-Reflection. Self-contrast: Better reflection through inconsistent solving perspectives.
- [25] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15174–15186, 2024.
- [26] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbulin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.