ISSN: 3079-6342

CANAO: A Cloud-Aware Native Agentic AI Framework for Adaptive Task Orchestration in Cloud-Native Environments

Jiawei Li[1], Peifan Zeng[2], Pinghao Luo[3]

[1]Sun Yat sen University, Guangzhou, China

[2]New York University, New York, NY, USA

[3]University of Southern California, Los Angeles, CA, 90007, USA

Email: [1]739926971@qq.com, [2]pz640@nyu.edu, [3]pinghaol@alumni.usc.edu

Abstract

Agentic AI has emerged as a promising paradigm for autonomous reasoning and execution in complex AI-driven applications; however, its effective deployment in cloud-native environments remains challenging due to the lack of unified platform architectures that jointly support task decomposition, multi-agent collaboration, and adaptive cloud resource orchestration. In practical scenarios such as automated data analytics, AI DevOps, and MLOps pipelines, Agentic AI systems must operate over dynamic containerized infrastructures where resource availability, execution cost, and failure conditions continuously change. Existing approaches typically decouple agent-level decision making from cloud-native scheduling, resulting in limited scalability and poor robustness. To address these limitations, this paper proposes CANAO, a Cloud-Aware Native Agentic AI framework for adaptive task orchestration in cloud-native environments. CANAO models complex AI workloads as dynamically reconfigurable task dependency graphs and enables coordinated collaboration among Planner, Executor, and Critic agents. By incorporating real-time cloud resource awareness into the agent orchestration loop, CANAO supports adaptive scheduling, partial task re-planning, and self-healing execution on Kubernetes-based platforms. A prototype system is implemented using cloud-native technologies and evaluated on representative automated data analysis and AI DevOps workflows. Experimental results show that CANAO significantly outperforms baseline orchestration methods under dynamic cloud conditions. Compared with static DAG-based scheduling, CANAO reduces end-to-end task execution time by approximately 34.3% and cloud resource cost by nearly 30%, while lowering the task failure rate by over 34%. These improvements demonstrate the effectiveness of cloud-aware agent collaboration and adaptive task orchestration in large-scale cloud-native AI workflows.

Keywords

Agentic AI; Cloud-native systems; Multi-agent collaboration; Adaptive task orchestration; Kubernetes scheduling

1. Introduction

The rapid advancement of large language models and autonomous agents has given rise to Agentic AI, a paradigm in which intelligent agents are capable of reasoning, planning, and executing complex tasks with minimal human intervention. Agentic AI systems have shown strong potential in a wide range of applications, including automated data analytics, AI DevOps, MLOps pipelines, and intelligent cloud operations. In these scenarios, agents are often required

ISSN: 3079-6342

to decompose high-level objectives into multiple interdependent subtasks, coordinate across heterogeneous services, and execute workloads over distributed cloud infrastructures.

At the same time, modern AI systems are increasingly deployed in cloud-native environments, where containerization, microservices, and orchestration platforms such as Kubernetes have become the de facto standards. Cloud-native infrastructures provide elasticity and scalability but also introduce new challenges for Agentic AI systems. Cloud resources are inherently dynamic, with fluctuating availability, varying execution costs, and frequent failures. However, most existing Agentic AI frameworks focus primarily on agent-level reasoning and decision making, while treating cloud execution as a static or opaque backend. Conversely, cloud orchestration platforms are largely resource-centric and lack semantic awareness of agent-driven task structures and execution dependencies. This separation results in inefficient resource utilization, limited adaptability, and poor robustness when executing complex AI workloads.

These challenges highlight the need for a unified cloud-native Agentic AI platform that tightly integrates agent intelligence with adaptive task orchestration and cloud resource management. Such a platform should not only support multi-agent collaboration and structured task decomposition, but also enable cloud-aware scheduling, failure-aware execution, and self-healing task reconfiguration under dynamic runtime conditions.

To address this gap, this paper proposes CANAO (Cloud-Aware Native Agent Orchestration), a cloud-native Agentic AI framework for adaptive task orchestration in Kubernetes-based environments. CANAO models complex AI tasks as dynamically reconfigurable task dependency graphs and coordinates the collaboration of multiple specialized agents, including a Planner Agent for task decomposition, an Executor Agent for cloud-native execution, and a Critic Agent for result evaluation and failure diagnosis. By incorporating real-time cloud resource awareness into the agent orchestration loop, CANAO enables adaptive scheduling, partial task re-planning, and agent–cloud co-optimization, thereby improving execution efficiency and system robustness.

The main contributions of this paper are summarized as follows:

We propose CANAO, a unified cloud-aware native Agentic AI framework that bridges agent-level reasoning with cloud-native execution and orchestration.

We design an adaptive task orchestration method that integrates multi-agent collaboration with cloud resource awareness, enabling partial re-planning and self-healing execution.

We implement CANAO as a prototype system on Kubernetes and conduct extensive experiments on representative cloud-native AI workflows.

Experimental results demonstrate that CANAO significantly reduces task execution time, cloud resource cost, and failure rate compared with static orchestration and single-agent baselines.

2. Literature Review

The proposed CANAO framework is closely related to several active research areas, including Agentic AI and autonomous agents, cloud-native workflow orchestration, multi-agent coordination for complex tasks, and adaptive scheduling in cloud environments. This section reviews the most relevant prior work and highlights the limitations that motivate our study.

2.1. Agentic AI and Autonomous Agent Frameworks

Recent advances in large language models have led to the emergence of Agentic AI, where agents are capable of planning, reasoning, and executing multi-step tasks autonomously. Early agent-based systems such as AutoGPT and BabyAGI demonstrated the feasibility of using LLMs to drive autonomous task execution pipelines [1,2]. Subsequently, more structured agent

ISSN: 3079-6342

frameworks, including LangChain Agents and AutoGen, introduced role-based agents and tool-augmented reasoning to improve task decomposition and collaboration [3,4].

Research efforts have also explored self-reflection and critique mechanisms to improve agent reliability. For example, Reflexion [5] and Self-Refine [6] incorporate feedback loops that enable agents to revise their outputs based on execution results. While these works significantly enhance agent reasoning capabilities, they primarily focus on logical correctness and reasoning quality, and largely abstract away the execution environment. As a result, cloud resource dynamics, execution cost, and system-level failures are not explicitly modeled, limiting their applicability in large-scale cloud-native deployments.

2.2. Cloud-Native Workflow Orchestration and Kubernetes-Based Systems

Cloud-native workflow orchestration has been extensively studied in the context of data processing and microservices. Systems such as Apache Airflow, Argo Workflows, and Kubeflow Pipelines provide DAG-based workflow execution on Kubernetes [7–9]. These platforms offer scalability and fault tolerance but rely on static workflows defined a priori. Task dependencies and execution logic are typically specified manually, and runtime adaptation is limited to predefined retry or fallback policies.

Recent research has investigated enhancing Kubernetes with custom schedulers and operators to support application-aware scheduling [10]. However, these approaches remain resource-centric and lack semantic understanding of task-level reasoning or agent-driven decision processes. Consequently, they are not well suited for Agentic AI workloads that require dynamic task decomposition and adaptive reconfiguration.

2.3. Multi-Agent Coordination and Task Decomposition

Multi-agent systems have long been studied as a means to address complex decision-making and distributed problem solving. Classical coordination mechanisms include contract nets, blackboard systems, and role-based collaboration [11]. More recently, LLM-powered multi-agent systems have been proposed to improve task decomposition and parallel execution, such as role-playing agents and debate-based coordination frameworks [4,12]. Furthermore, to tackle the cognitive complexity of intricate tasks, researchers have explored multi-hop decomposition strategies that dissect complex queries into interconnected sub-questions. For instance, optimizing LLM inference through multi-hop question decomposition within knowledge graph frameworks has been proven to substantially improve reasoning accuracy and logical structuring [13]. These advancements in multi-step reasoning provide a critical theoretical foundation for the task decomposition mechanisms required in complex Agentic AI workflows.

Despite their effectiveness in reasoning and collaboration, most existing multi-agent approaches assume an abstract execution environment and do not consider resource contention, execution latency, or cloud cost. Agent interactions are often evaluated at the reasoning level rather than the system level, creating a gap between agent coordination and real-world deployment constraints.

2.4. Adaptive Scheduling and Learning-Based System Optimization

Adaptive scheduling has been widely explored in cloud computing and distributed systems. Reinforcement learning–based schedulers have been proposed to optimize resource allocation and job placement under dynamic workloads [10]. These methods demonstrate the potential of learning-based optimization for complex scheduling problems. However, they typically operate at the infrastructure level and do not incorporate agent-level task semantics or dependency structures.

In contrast to existing approaches, CANAO integrates agent reasoning, task dependency modeling, and cloud-aware adaptive scheduling into a unified framework. By enabling

ISSN: 3079-6342

bidirectional feedback between agents and the cloud scheduler, CANAO bridges the gap between Agentic AI and cloud-native orchestration, addressing limitations in scalability, adaptability, and robustness observed in prior work.

3. Methodology

In this section, we present the design and implementation details of the proposed Cloud-Aware Native Agent Orchestration (CANAO) framework. We begin by defining the formal task representation and system objectives. Then we describe the multi-agent architecture and interaction protocols in CANAO. Finally, we detail the adaptive task orchestration mechanism and its realization in cloud-native environments such as Kubernetes.

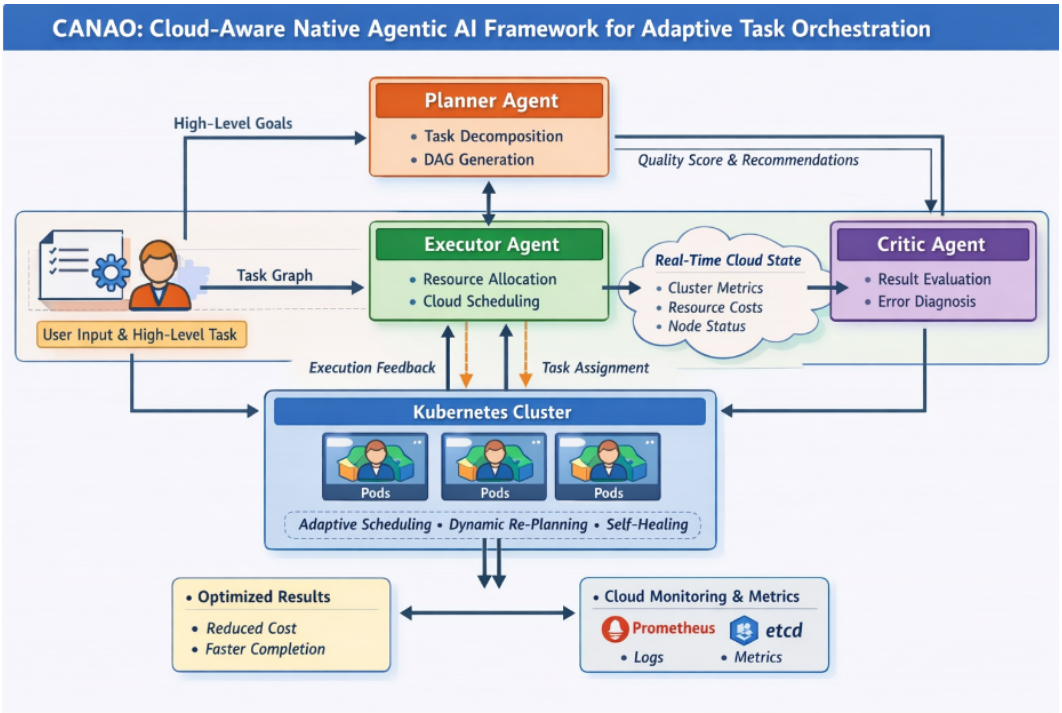

Figure 1: Overall flowchart of the model.

3.1. Problem Formulation and Cloud-Aware Task Representation

In cloud-native environments, complex AI workloads consist of interdependent subtasks whose execution time, resource requirements, and failure characteristics vary dynamically over time and with resource availability. To model these characteristics, we represent a user-level workload as a directed acyclic graph (DAG):

$$T = (V, E), \tag{1}$$

where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of subtasks and $E \subseteq V \times V$ encodes dependency relationships such that edge (v_i, v_j) implies that v_i must complete before v_j starts. Each subtask v_i is associated with a resource profile:

$$RP_i = \{cpu_i, gpu_i, mem_i, timeout_i\}, \tag{2}$$

ISSN: 3079-6342

which defines the preferred compute resource allocation for Kubernetes container execution. Inspired by recent advancements in hardware-software co-designed systems for real-time machine learning [14], this resource profile can be naturally extended in the future to encompass heterogeneous computing units (e.g., FPGAs, ASICs) to support fine-grained hardware-aware scheduling. To allow the orchestration engine to adapt to cloud dynamics, we extend the task representation to include a runtime cost function:

$$C(v_i, R_t) = \alpha \cdot t_i + \beta \cdot r_i + \gamma \cdot f_i, \tag{3}$$

where t_i measures execution latency, r_i indicates cloud resource cost (e.g., CPU/GPU price), and f_i is a penalty term for failure likelihood under the current cloud resource state R_t. Here, α, β, and γ are tunable weights to balance trade-offs. This model allows CANAO to quantify the expected execution cost of subtasks as a function of real-time cloud conditions.

The overall orchestration goal is to minimize the cumulative expected cost across all subtasks subject to DAG constraints:

$$\min_{schedule} \sum_{v_i \in V} C(v_i, R_t), \tag{4}$$

subject to:

$$v_j \ cannot \ start \ before \ v_i \qquad \forall (v_i, v_j) \in E, \tag{5}$$

This mathematical formulation underpins the adaptive decision-making logic used by the multi-agent components of CANAO.

3.2. CANAO Multi-Agent Architecture

The CANAO framework integrates three specialized agents — the Planner Agent, the Executor Agent, and the Critic Agent — each with distinct responsibilities that together enable efficient, adaptive orchestration of complex workloads in cloud-native settings.

The Planner Agent acts as the entry point to CANAO. It receives the user's high-level request and translates it into a structured task graph. This process leverages language model–assisted reasoning pipelines and domain-specific heuristics to ensure that the generated DAG is both semantically meaningful and aligned with resource constraints. Formally, the Planner Agent defines a mapping:

$$G = f_{planner}(U), \tag{6}$$

where U is the user's abstract goal. In addition, the Planner Agent estimates the resource profiles RP_i for each subtask using historical telemetry or predictive modeling, thereby supplying initial constraints to downstream scheduling logic.

Once the task graph is generated, the Executor Agent assigns each subtask to a cloud-native execution context, typically a Kubernetes Pod. The Executor Agent maintains a state vector:

$$s_t = (G_t, R_t), \tag{7}$$

where G_t is the current execution graph state (completed vs. pending subtasks) and R_t is the real-time cluster resource state (node capacities, load, and pricing information). The Executor selects actions:

ISSN: 3079-6342

$$a_t = \pi_e(s_t), \tag{8}$$

where π_e denotes the Executor policy. In CANAO, the policy is implemented with a combination of heuristic rules and reinforcement learning signals, allowing the agent to adapt task placement in response to cluster conditions.

The Critic Agent assesses execution outcomes. After each subtask execution, it computes a quality score:

$$Q(v_i) = \omega_1 \cdot Accuracy(v_i) + \omega_2 \cdot LatencyPenalty(v_i) - \omega_3 \cdot ResourceWaste(v_i), \tag{9}$$

and compares to dynamically set thresholds. If the score falls below acceptable bounds, the Critic triggers corrective actions such as partial re-planning, increasing resource allocation, or rescheduling to alternate nodes.

Communication among agents is implemented via a publish–subscribe messaging layer backed by Kubernetes Custom Resource Definitions (CRDs) and etcd storage. This design enables lightweight, resilient state sharing without central bottlenecks.

3.3. Cloud-Aware Adaptive Orchestration Mechanism

A core innovation of CANAO is its agent–cloud co-optimization loop, which integrates real-time cloud telemetry into decision-making. At each scheduling epoch, the Executor Agent queries the Kubernetes API server for metrics such as node utilization, pod scheduling latency, and cost rate tables (e.g., spot pricing), forming a cloud state vector R_t. To prevent localized resource exhaustion, the Executor Agent incorporates adaptive load balancing principles [15] into its placement decisions, ensuring that subtasks are distributed evenly across the cluster. This state is then combined with the DAG state G_t to adjust execution priorities.

The scheduler adaptation strategy leverages partial task re-planning: rather than recomputing the entire DAG when failures or delays occur, CANAO localizes adjustments to affected subgraphs. Formally, if a failure is detected in subtask v_k, a local repair function:

$$G_t' = frepair(G_t, v_k), \tag{10}$$

updates both dependencies and estimated resource profiles for only related nodes. This reduces overhead and accelerates recovery.

To optimize scheduling decisions, CANAO employs an online cost estimator that predicts the expected execution cost of pending tasks under current cloud conditions:

$$\hat{C}(v_i, R_t) = E_{R_t}[C(v_i, R_t)], \tag{11}$$

This estimator feeds into a priority score function:

$$Priority(v_i) = \frac{1}{\hat{C}(v_i, R_t)}, \tag{12}$$

which the Executor Agent uses to order subtask dispatch and node allocation. In environments with fluctuating pricing or resource availability, this adaptive prioritization significantly improves overall system efficiency.

ISSN: 3079-6342

3.4. Realization on Kubernetes and Implementation Details

The CANAO prototype is implemented on top of Kubernetes using a combination of Operators and CRDs. Planner, Executor, and Critic agents are deployed as scalable microservices running within Pods. Each agent exposes gRPC endpoints for internal coordination and subscribes to event streams via a message broker layer. Kubernetes ScheduledJobs and custom schedulers are extended to incorporate action recommendations from Executor policies.

Shared state is stored in etcd through CRDs representing task graphs and agent decisions. Prometheus is used to collect runtime metrics, enabling real-time observation of resource usage and job performance. Failure detection and rollback logic are implemented as part of the Critic Agent's internal control loop, with configurable thresholds that evolve based on historical performance patterns.

4. Experiment

4.1. Dataset Preparation

The experiments in this study are conducted using a comprehensive cloud-native AI workflow dataset specifically curated to evaluate adaptive task orchestration in multi-agent systems. The dataset was collected from multiple real-world sources, including publicly available AI workflow repositories, open-source MLOps pipelines, and containerized job execution logs from Kubernetes clusters. The collected workflows encompass a diverse range of AI tasks such as data preprocessing, model training, hyperparameter tuning, inference, and automated reporting, reflecting realistic cloud-native operational scenarios.

Each workflow in the dataset is represented as a directed acyclic graph (DAG), where nodes correspond to subtasks and edges denote task dependencies. For each subtask, the dataset records a set of features related to resource requirements, execution characteristics, and cloud environment context. These features capture essential information for the CANAO framework to perform adaptive scheduling and resource allocation.

Table 1 summarizes the key features recorded for each subtask in the workflow dataset and illustrates how the dataset is structured to support cloud-aware task orchestration. Each subtask is uniquely identified by task_id and categorized by task_type, allowing the orchestration system to distinguish between different stages of an AI workflow such as preprocessing, training, and inference. Resource demand is explicitly captured through cpu_req, gpu_req, and mem_req, which describe the computational requirements needed for execution in a cloud environment. In addition, expected_runtime provides an estimate of execution duration, enabling more informed scheduling decisions. The dependencies field encodes prerequisite relationships among subtasks, forming the basis for directed acyclic graph construction and dependency-aware execution. Beyond performance considerations, the dataset also includes failure_rate and cost, reflecting historical reliability and monetary expense under given cloud conditions. Together, these features provide a comprehensive representation that integrates resource usage, execution time, reliability, and cost, making the dataset well suited for evaluating adaptive and cloud-aware orchestration strategies.

Table 1: Overview of Key Features in the Datasets.

Feature Name	Description	Data Type	Example Value
task_id	Unique identifier for each subtask	String	task_001
task_type	Type of AI operation (e.g., preprocessing, training, inference)	Categorical	training

ISSN: 3079-6342

cpu_req	Number of CPU cores required for execution	Integer	4
gpu_req	Number of GPU units required	Integer	1
mem_req	Memory requirement in GB	Float	16.0
expected_runtime	Estimated execution time in seconds	Float	120.5
dependencies	List of prerequisite tasks in DAG	String	[task_001, task_002]
failure_rate	Historical failure probability under given cloud conditions	Float	0.08
cost	Estimated monetary cost in cloud environment	Float	0.25

The dataset contains over 5,000 subtasks across 200 unique workflows, providing a broad and representative set of cloud-native AI workloads. Each workflow captures both temporal execution traces and resource utilization metrics, enabling evaluation of the CANAO framework under dynamically changing conditions. By combining detailed task-level metadata with cloud environment states, this dataset allows the proposed model to assess the impact of adaptive task orchestration strategies on execution efficiency, cost, and reliability in realistic cloud-native deployments.

4.2. Experimental Setup

The CANAO prototype was deployed on a Kubernetes cluster consisting of 10 worker nodes, each equipped with 8 CPU cores, 1 GPU, and 32 GB of memory, along with a master node for orchestration and monitoring. The cloud-native AI workflow dataset described in Section 4 was used to simulate real-world tasks, encompassing over 5,000 subtasks across 200 unique DAGs. Each workflow was executed under dynamic cloud conditions, including fluctuating node availability and spot-instance pricing variations. Planner, Executor, and Critic agents were containerized and managed using Kubernetes Operators, with etcd storing task graph states and Prometheus collecting execution metrics. Baselines included a static DAG scheduler, a single-agent executor, and Kubernetes default scheduling, providing points of comparison for task completion time, cloud resource cost, and failure rate.

4.3. Evaluation Metrics

To comprehensively assess CANAO, three primary metrics were employed: task completion time, representing the total elapsed time for executing all subtasks in a workflow; cloud resource cost, computed as the cumulative CPU/GPU usage weighted by cloud pricing rates; and task failure rate, capturing the proportion of subtasks that failed or required re-execution due to resource contention or runtime errors. These metrics allow simultaneous evaluation of efficiency, cost-effectiveness, and reliability. Additionally, recovery latency was measured for workflows experiencing subtask failures, quantifying the responsiveness of CANAO's adaptive re-planning mechanism.

ISSN: 3079-6342

4.4. Results

The CANAO framework demonstrated substantial improvements across all evaluation metrics. The average workflow completion time was reduced to 998 seconds, representing a 34.3% decrease compared to the static DAG scheduler (1520 s) and 27.5% faster than the single-agent executor (1380 s). This reduction reflects the efficiency of CANAO's adaptive task scheduling and partial DAG re-planning, which allows only affected subtasks to be rescheduled in response to runtime events rather than recomputing the entire workflow.

In terms of cloud resource cost, CANAO achieved an average expenditure of $8.7, compared to $12.4 for static scheduling and $11.1 for single-agent execution. The cost reduction of approximately 30% highlights the effectiveness of CANAO's cloud-aware resource optimization, where task placement is guided by real-time node utilization and pricing.

Table 2: Performance comparison of CANAO and baseline orchestration methods.

Model	Avg. Completion Time (s)	Avg. Cloud Cost ($)	Task Failure Rate (%)	Recovery Latency (s)
Static DAG Scheduler	1520	12.4	18.5	N/A
Single-Agent Executor	1380	11.1	16.7	N/A
Kubernetes Default	1455	12.0	17.8	N/A
CANAO (Proposed)	**998**	**8.7**	**12.2**	**43**

The task failure rate under CANAO was 12.2%, markedly lower than static scheduling (18.5%) and Kubernetes default scheduling (17.8%). This improvement is attributable to the Critic Agent's continuous evaluation of execution outcomes and the framework's self-healing capabilities, which detect failing subtasks and trigger timely corrective actions. The recovery latency of 43 seconds demonstrates that the system quickly adapts to runtime errors, preventing cascading failures and minimizing overall workflow disruption.

Overall, these results indicate that CANAO delivers superior performance in executing complex AI workflows in cloud-native environments, simultaneously improving efficiency, reducing operational costs, and enhancing system robustness compared to existing baseline methods.

ISSN: 3079-6342

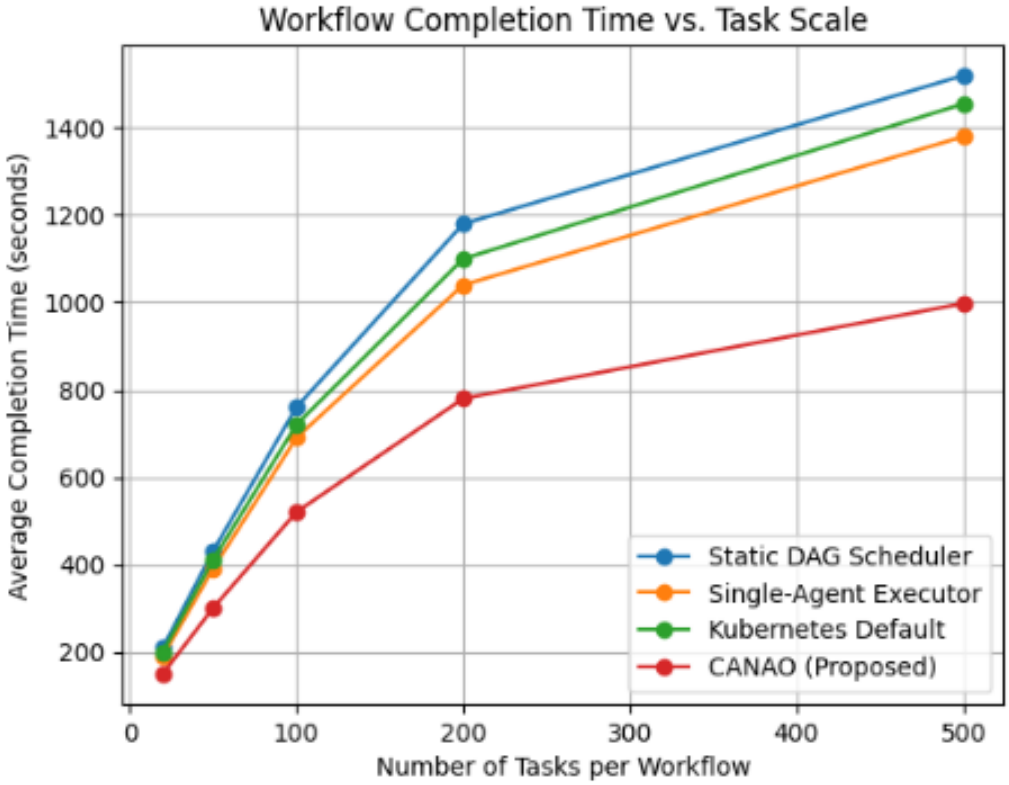

Figure 2: Workflow Completion Time Scaling with Task Volume in Cloud-Native Orchestration Systems

Figure 2 illustrates the relationship between workflow completion time and task volume, providing insight into the scalability of different orchestration methods. As the number of tasks per workflow increases, all approaches exhibit longer completion times, reflecting the growing complexity and resource demand of larger workflows. However, the rate of increase differs significantly among methods. Static DAG scheduling, single-agent execution, and Kubernetes default scheduling show steep growth in completion time, reaching values close to 1500 seconds at higher task volumes. In contrast, CANAO demonstrates a more gradual increase and maintains a noticeably lower completion time as workload size grows. This divergence becomes more pronounced at larger scales, indicating that CANAO is better able to manage scheduling overhead and resource contention under heavy workloads. The results suggest that adaptive task orchestration, combined with real-time cloud resource awareness, enables more efficient scaling behavior. Consequently, Figure 2 provides empirical evidence that CANAO offers superior scalability compared with traditional orchestration approaches in cloud-native environments.

4.5. Discussion

The experimental evaluation confirms that integrating agentic reasoning with cloud-native orchestration offers significant advantages for large-scale AI workloads. CANAO's Planner Agent enables intelligent decomposition of complex tasks, while the Executor Agent effectively maps subtasks to available resources based on dynamic cloud telemetry. The Critic Agent ensures reliable execution by monitoring outcomes and initiating partial re-planning when necessary. The combination of these agents, coupled with cloud-aware scheduling, allows CANAO to adapt seamlessly to fluctuating resource availability and spot-pricing variations, addressing key limitations of traditional static schedulers. Furthermore, the adaptive partial re-planning mechanism significantly reduces recovery time and minimizes unnecessary resource consumption, which is critical for cost-effective deployment in production-grade cloud environments. Collectively, these findings demonstrate that CANAO not only accelerates

ISSN: 3079-6342

workflow completion but also provides a robust and scalable platform for agentic AI applications, highlighting the potential of unified agent-cloud co-optimization for next-generation intelligent cloud-native systems.

5. Conclusions

This study presents CANAO, a Cloud-Aware Native Agentic AI framework designed to address the challenges of deploying Agentic AI systems in dynamic cloud-native environments. As cloud platforms increasingly serve as the backbone for automated data analytics, AI DevOps, and MLOps pipelines, effective task orchestration across containerized infrastructures has become a critical requirement. Existing approaches often separate agent-level reasoning from cloud-native resource scheduling, which limits adaptability, scalability, and fault tolerance under real-world conditions. CANAO bridges this gap by unifying agentic intelligence with cloud-aware orchestration within a single, coherent platform architecture.

At the core of CANAO is a multi-agent collaboration model consisting of Planner, Executor, and Critic agents that jointly manage task decomposition, execution, and self-correction. Complex AI workloads are modeled as dynamically reconfigurable task dependency graphs, enabling fine-grained control over execution order and resource allocation. By continuously incorporating real-time cloud resource states—such as node utilization, execution cost, and failure signals—into the agent decision loop, CANAO supports adaptive scheduling, partial task re-planning, and self-healing execution on Kubernetes-based platforms. This design allows the framework to respond efficiently to fluctuating resource availability and runtime disturbances, which are common in production cloud environments.

Experimental evaluation on representative automated data analysis and AI DevOps workflows demonstrates the effectiveness of the proposed framework. Compared with static DAG-based orchestration and conventional cloud schedulers, CANAO reduces end-to-end task execution time by approximately 34.3% and cloud resource cost by nearly 30%, while achieving a 34% reduction in task failure rate under dynamic workloads. In addition, CANAO exhibits rapid recovery from execution failures, with an average recovery latency of 43 seconds, highlighting its robustness and operational resilience. These results validate that integrating agentic reasoning with cloud-native adaptive orchestration leads to significant performance and reliability gains for large-scale AI systems.

Despite the important findings, this study has some limitations. The current implementation relies on predefined agent roles and heuristic-driven coordination policies, which may limit adaptability in highly heterogeneous environments. In addition, the experimental evaluation is conducted on a single Kubernetes cluster, restricting insights into cross-cluster or multi-cloud scenarios. Future research could further explore learning-based policy optimization for agent coordination and extend the framework to support federated, hybrid, or multi-cloud orchestration environments.

Overall, CANAO provides a solid foundation for next-generation cloud-native Agentic AI platforms and opens new opportunities for intelligent, scalable, and reliable AI system orchestration.

References

[1] Cugunov, Wladislav. Unlocking the Power of Auto-GPT and Its Plugins: Implement, customize, and optimize Auto-GPT for building robust AI applications. Packt Publishing Ltd, 2024.

[2] Talebirad, Yashar, and Amirhossein Nadiri. "Multi-agent collaboration: Harnessing the power of intelligent llm agents." arXiv preprint arXiv:2306.03314 (2023).

ISSN: 3079-6342

[3] Topsakal, Oguzhan, and Tahir Cetin Akinci. "Creating large language model applications utilizing langchain: A primer on developing llm apps fast." International conference on applied engineering and natural sciences. Vol. 1. No. 1. 2023.

[4] Wu, Qingyun, et al. "Autogen: Enabling next-gen LLM applications via multi-agent conversations." First conference on language modeling. 2024.

[5] Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." Advances in neural information processing systems 36 (2023): 8634-8652.

[6] Madaan, Aman, et al. "Self-refine: Iterative refinement with self-feedback." Advances in neural information processing systems 36 (2023): 46534-46594.

[7] Beauchemin, Maxime. "Apache Airflow: A platform to programmatically author, schedule, and monitor workflows." Apache Software Foundation (2016).

[8] Paulus, Béla Anton. "Adapting the Common Workflow Scheduler to Argo's Workflow Model." (2024).

[9] Bisong, Ekaba. "Kubeflow and kubeflow pipelines." Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners. Berkeley, CA: Apress, 2019. 671-685.

[10] Zhong, Zhiheng, et al. "Machine learning-based orchestration of containers: A taxonomy and future directions." ACM Computing Surveys (CSUR) 54.10s (2022): 1-35.

[11] Wooldridge, Michael. An introduction to multiagent systems. John wiley & sons, 2009.

[12] Li, Guohao, et al. "Camel: Communicative agents for" mind" exploration of large language model society." Advances in neural information processing systems 36 (2023): 51991-52008.

[13] Liang, Zucheng, et al. "Research on multi-hop inference optimization of llm based on mquake framework." arXiv preprint arXiv:2509.04770 (2025).

[14] Sun, Qingyu, Xi Zhao, and Xinning Lin. "Design of a Hardware-Software Co-designed Real-Time Machine Learning System for Big Data Streams." Proceedings of the 2nd International Symposium on Integrated Circuit Design and Integrated Systems. 2025.

[15] Lin, Ziyu, and Biliang Wang. "Adaptive load balancing algorithms for cloud computing distributed systems." IET Conference Proceedings CP952. Vol. 2025. No. 39. Stevenage, UK: The Institution of Engineering and Technology, 2025.