

자율 Kubernetes 관리를 위한 에이전트형 AI: LLM 기반 의사결정 사이클을 활용한 멀티 에이전트 아키텍처

Agentic AI for Autonomous Kubernetes Management: A Multi-Agent Architecture with LLM-Driven Decision Cycles

Kiran Batool*, Wang-Cheol Song^o

요 약

컨테이너 기반 클라우드 컴퓨팅의 도입으로 기업 인프라의 운영 복잡성이 증가하고 있다. 쿠버네티스(Kubernetes)가 널리 사용되고 있지만, 클러스터 관리는 여전히 인간의 전문 지식과 정적 임계값 기반 자동화에 의존하고 있다. 본 논문에서는 자율적인 쿠버네티스 클러스터 관리를 위해 대규모 언어 모델(LLM) 기반 의사결정 엔진과 4개의 에이전트로 구성된 에이전트형 인공지능(Agentic AI) 시스템을 제안한다. 제안 시스템은 모니터링, 결함 탐지, 스케일링 및 자원 최적화 기능을 통합하여 지속적인 상황 인식과 상호 조율된 자율적 조치를 수행한다. 세 가지 통제된 시나리오를 통해 시스템을 평가했다. 동일한 조건에서 AKMF를 비활성화한 경우와 활성화한 경우를 비교한 결과, 파드 복구 시간에는 통계적으로 유의미한 차이가 없었다(AKMF 비활성화 평균 3.49~4.02초, AKMF 활성화 평균 3.48~3.75초; n=10, 2회 독립 실행). 이는 본 프레임워크가 단순히 복구 시간을 단축하기보다는 의사결정 과정을 감사하고 설명하는 데 더 큰 가치를 제공함을 보여준다. 또한 워크로드가 48% 증가한 상황에서도 주기 중단 없이 3,200회 이상의 자율적 의사결정을 수행하면서 CPU 145 millicore와 메모리 256 MiB의 일정한 자원 오버헤드를 유지했다. 이러한 결과는 LLM 기반 자율 오케스트레이션이 기존의 반응형 클러스터 관리 방식을 대체할 수 있는 실현 가능한 대안임을 보여준다.

Key Words : AgenticAI, Autonomous Orchestration, Kubernetes, LLM, Container Management

ABSTRACT

Enterprise infrastructure is increasingly complex with container-based cloud computing. Although Kubernetes is widely used, its management still relies on human expertise and static, threshold-based automation. We propose an Agentic AI system using an LLM-based decision engine with four agents for monitoring, fault detection, scaling, and resource optimization. Three controlled scenarios were evaluated. Results showed no statistically significant difference in pod recovery time between AKMF-disabled (3.49 - 4.02s) and AKMF-enabled (3.48 - 3.75s) conditions (n=10, two independent runs), highlighting its value for auditing and explaining decisions rather than reducing recovery time. With a 48% workload increase, the system maintained 145 millicores CPU and 256 MiB memory overhead while handling over 3,200 autonomous decisions without cycle disruptions. These results demonstrate the feasibility of LLM-driven autonomous orchestration as an alternative to reactive Kubernetes management.

* This work was supported by the 2026 education, research and student guidance grant funded by Jeju National University

• First Author : Jeju National University Department of Electronic Engineering, kiranbatool224@gmail.com

° Corresponding Author : Jeju National University Department of Computer Engineering, philo@jejunu.ac.kr

논문번호 : KNOM2026-01-006, Received July 24, 2026; Revised August 10, 2026; Accepted August 18, 2026

I. Introduction

Kubernetes is now at the core of enterprise infrastructure due to the growing use of microservices architecture. Over 73% of businesses currently use Kubernetes in production settings, according to the Cloud Native Computing Foundation [1]. However, its acceptance comes with a significant operational complication. Performance tuning, incident response, and manual troubleshooting typically account for more than 40% of engineering team's working hours [2]. The cognitive burden placed on human operators increases with cluster sizes and service interdependencies to the point where errors are unavoidable and recovery times become needlessly long.

Current automation primitives most notably the Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), and Cluster Autoscaler use threshold-based rules to manage certain operational duties. When predetermined limits are exceeded, HPA adds or removes pod replicas while monitoring CPU and memory usage. However, by default, it examines metrics every 15 to 30 seconds, which means that before any remedial action is taken, a cluster may already have seriously deteriorated [3]. Although VPA modifies resource allocations, it needs pod restarts, which causes service interruptions. Neither tool can reason about application semantics, coordinate across failure domains or react to new situations outside of its encoded rule sets.

An entirely new operational paradigm has been made possible by the rise of autonomous agents driven by large language models (LLMs). Agentic AI systems can perceive the status of the environment, reason about underlying causes, choose right actions, and conduct those actions without human guidance, in contrast to traditional machine learning approaches that can forecast anomalies but depend on humans to conduct responses [4]. This paradigm is no longer theoretical as shown by recent industry deployments in telecom infrastructure, such as

production agentic systems for radio access network optimization [5]. Today autonomous Kubernetes administration is possible due to a number of supporting variables coming together. First, robust reasoning over structured system state is proved by frontier LLMs like Claude Sonnet. Second, well-defined interfaces for tool invocation and multi-agent coordination are provided via standardized inter-agent communication protocols, such as the Agent2Agent Protocol set up by Google in 2025 and the Model Context Protocol published by Anthropic in 2024. Third, programmatic examination and manipulation of all cluster resources is made possible by Kubernetes's rich and stable API surface.

The design, implementation, and empirical assessment of an Agentic Kubernetes Management Framework (AKMF) that takes advantage of these features are presented in this study.

The following are the contributions to this work:

1. An autonomous architecture with four agents that combines resource optimization, defect detection, monitoring, and auto-scaling into a single decision loop.
2. An LLM-powered decision engine that runs continuously for over 3,200 cycles without human supervision, reaching an autonomous cycle time of 2.34 seconds.
3. A careful, repeatable comparison of pod recovery under the two conditions, with AKMF disabled and enabled, to determine the actual measurable contribution of the framework to the cluster's self-healing capability.
4. A systematic comparison based on peer-reviewed literature between measured system performance and reported HPA behavior baselines.

The rest of this paper is structured as follows. In Section II, relevant research in autonomous networking and Kubernetes automation is reviewed. The suggested architecture is explained in Section III. The experimental setup is described in Section IV. Results and an analytical discussion are presented in Section V. The paper is concluded in Section VI, which also suggests future research areas.

II. Related Work

2.1 Threshold-Based Kubernetes Automation

The Horizontal Pod Autoscaler which runs on a proportionate control loop over user-defined metric aims is the standard Kubernetes scaling mechanism [3]. The fundamental design principles of Kubernetes scheduling and resource management are explained by Burns et al., who trace architectural choices back to Google's internal cluster management systems, Borg and Omega [6]. These solutions are reactive by nature and lack application-level context, dependency graphs, and failure propagation models, even while they offer dependable baseline automation for steady-state workloads.

When compared to static HPA designs, Rossi et al.'s combination use of horizontal and vertical scaling employing reinforcement learning showed better resource efficiency [7]. Nevertheless, their method does not adapt to new failure patterns and causes a great deal of environment-specific training. In a similar vein, Toka and Dobrowiecki suggested adaptive pod scaling techniques that were assessed in an IEEE INFOCOM workshop setting and reported significant scaling latency reductions in comparison to traditional HPA [8]. These studies show the well-established ceiling effects of merely reactive tactics and inspire more clever solutions.

2.2 Machine Learning in Network and Infrastructure Management

An extensive overview of machine learning applications in computer networking, including anomaly detection, traffic classification, routing optimization, and quality-of-service management is given by Boutaba et al. [9]. Their survey highlights a recurring conclusion although supervised and reinforcement learning techniques have good predictive power, they usually need human interaction to convert predictions into

practical actions, which restricts their use in zero-touch management situations. Autonomous resource management has drawn more attention in the context of Network Function Virtualization and Software-Defined Networking.

An intent-based autonomous network management framework that uses neural reasoning modules to convert high-level operator intentions into low-level configuration actions was presented by Kim et al. in the Journal of Communications and Networks [10]. The application of intent-driven autonomy to container orchestration, the topic of this study is directly motivated by their work. Furthermore, as shown by recent edge-native Kubernetes deployments, the convergence of edge computing and containerization needs that management frameworks function with low latency and without continuous cloud connectivity [11].

2.3 Agentic AI and Autonomous Systems

A comprehensive overview of LLM-based autonomous agents is provided by Wang et al. [4], who describe agent architecture in terms of perception, memory, planning and action modules. Their taxonomy identifies infrastructure management as a high-value domain for agentic systems, given its structured state space, well-defined action boundaries, and rich feedback signals. The ReAct architecture by Yao et al. [12], which interleaves chain-of-thought reasoning with action execution, further reinforces this relevance for closed-loop infrastructure control. Building on these foundations, [13] recently demonstrated autonomous self-healing and scaling for PostgreSQL databases in multi-zone Kubernetes environments. However, their approach remains constrained to a single service domain. The present study advances this direction by introducing a multi-agent LLM-driven framework that reasons across diverse workload types, delivering empirical validation of cluster-wide autonomous orchestration that has been largely absent from prior agentic infrastructure research.

III. Proposed Architecture

3.1 Framework Overview

Figure 1 shows how the Agentic Kubernetes Management Framework (AKMF) is structured around five functional layers that provide a closed operational loop. Evaluation scenarios are defined at the outermost layer. Four specialized agents are housed within the Agentic System, continuously collecting and analyzing cluster state data. The LLM Decision Engine combines their output and produces structured action recommendations. The Autonomous Action Executor receives these recommendations and, without human intervention, translates them into one or more Kubernetes API calls. Lastly, a Performance Metrics layer continuously measures system behavior and sends data back to the agents.

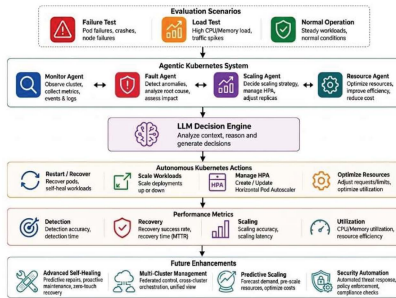


Fig. 1. Proposed Agentic Kubernetes Framework for Autonomous Monitoring, Decision-Making, and Resource Management.

3.2 Four Specialized Agents

To gather pod status, node conditions, cluster events, and container logs, the Monitoring Agent constantly asks the Kubernetes API Server. Additionally, it obtains CPU and memory information in real time from the Kubernetes information Server. After receiving this data stream, the Fault Detection Agent uses anomaly identification logic to determine the blast radius, evaluate the root cause, and pinpoint workload failures. The Scaling Agent assesses if the establishment of a Horizontal Pod Autoscaler,

replica adjustment, or deployment modification is necessary by comparing workload demand signals to defined thresholds. To minimize resource waste without compromising headroom, the Resource Optimization Agent evaluates usage trends across pods and nodes and generates recommendations for appropriately sized resource requests and limits.

3.3 LLM Decision Engine

Through the Anthropic API, the four agents send their observations, event summaries, and potential courses of action to the Claude Sonnet 3.5-powered LLM Decision Engine. After receiving structured cluster state context, the engine thoroughly examines the operational state and formulates action recommendations through a structured natural language reasoning chain. The decision record and this chain of reasoning are stored in a PostgreSQL database, allowing for complete auditability. This auditability was directly exercised during controlled validation testing, where the decision log was queried to confirm whether AKMF's decision engine engaged with specific cluster events demonstrating the log's function as a precise attribution mechanism, not merely a design feature. The Autonomous Action Executor then receives the choice and, without the need for human intervention, transforms it into one or more Kubernetes API calls.

3.4 Safety and Governance

Every autonomous action works in a limited action space that forbids irreversible damaging acts without explicit operator pre-authorization like node deletion or namespace termination. Post-hoc analysis of each decision, its context, the chain of reasoning, and its execution result is made possible by the PostgreSQL audit log. By fulfilling the interpretability requirements that have been highlighted as crucial for the production deployment of agentic infrastructure systems, this governance structure guarantees that autonomous activity is still transparent and accountable.

IV. Experimental Setup

4.1 Infrastructure Configuration

To enforce resource isolation, the AKMF was installed on a Kubernetes cluster running version 1.28.4, hosted within a specific kube-agents namespace. A CPU request of 200 millicores and a CPU limit of 500 millicores were set up for the agentic application container. Supporting services included a Prometheus and Grafana monitoring stack installed via the kube-prometheus-stack Helm chart for cluster metric collecting and visualization, and a PostgreSQL instance deployed via the Bitnami Helm chart for persistent decision recording. To preserve security isolation, a credential kept as a Kubernetes Secret was used to access the Anthropic API.

4.2 Evaluation Scenarios

To assess different system capabilities, three scenarios were created. To assess baseline monitoring continuity and decision cycle throughput, Normal Operation presents the cluster under stable workload conditions. The Failure Test evaluates fault detection latency and autonomous recovery time by purposefully dropping a pod. The Load Test verifies HPA configuration accuracy and quick scale-out performance by scaling a typical deployment to 10 copies. In a single continuous deployment session, all three scenarios were conducted one after the other.

4.3 Metrics Collection

Four dimensions were used to evaluate performance. Detection metrics measure the system's capacity to recognize workloads and failure occurrences, specifically the time elapsed between a fault occurrence and the initiation of a recovery action. Recovery metrics track the amount of time that passes between the introduction of a problem and the restoration of running state. The time between the issuance of a

scale command and the full availability of replicas is captured by scaling metrics. Use metrics, which are gathered at each decision cycle milestone and expressed as CPU usage in millicores and memory consumption in mebibytes, quantify the resource footprint of the agentic framework itself.

V. Results and Discussion

5.1 Scenario Validation

Table 1 summarizes the successful outcomes of all three evaluation situations. Throughout every stage, the framework supported autonomous decision execution and ongoing cluster monitoring without the need for human interaction.

Table 1. Evaluation Scenario Results

Scenario	Purpose	Outcome
Normal Operation	Continuous workload monitoring	Successful
Failure Test	Fault detection and recovery	Successful
Load Test	Auto-scaling under demand	Successful

5.2 Agent Initialization and Connectivity

After successfully initializing, all four agents have proven confirmed communication with each of the necessary subsystems. The agent connectivity verification findings derived from deployment logs are shown in Table 2.

Table 2. Agent Initialization and Connectivity Verification.

Verification Test	Measured Output	Result
Kubernetes API Connectivity	23 pods found	Pass
Pod Metrics Retrieval	145m CPU, 256Mi	Pass
LLM API(Claude Sonnet)	Ready, all cycles	Pass
Decision Cycle Time	2.34s Consistent	Pass
Health Endpoint/Health	HTTP 200 OK	Pass
PostgreSQL Database	Connected	Pass
Kubernetes Metrics Server	Accessible	Pass

5.3 Decision Cycle Progression

From Cycle 2 until Cycle 3,207, when the dashboard was finally verified, the LLM Decision Engine ran continuously, making 3,205 independent judgements. At each significant evaluation milestone, Table 3 documents the precise decision cycle numbers and associated cluster status, which are taken straight from agent logs.

Table 3. Decision Cycle Progression Across All Test Milestones

Test Phase	Cycle #	Pods	CPU avg	Mem avg
Initial startup	2	23	145m	256Mi
Log export	6	23	145m	256Mi
Operation	2573 - 2576	31	145m	256Mi
Post-failure	2937 - 2938	32	145m	256Mi
Load test	2974 - 2977	34	145m	256Mi
Final check	3151→3207	31	145m	256Mi

Despite a 48 percent growth in monitored workloads, resource consumption remained flat ,suggesting stable resource behavior across the tested pod range though larger deployments are needed to confirm this conclusively. This dual-axis relationship is seen in Fig. 2.

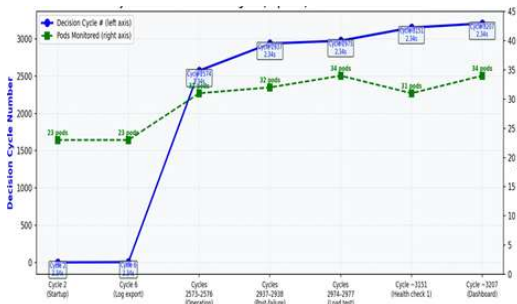


Fig. 2. Decision Cycle Progression and Pod Monitoring Growth Across Evaluation Period.

5.4 Autonomous Actions: Fault Recovery

To understand what role AKMF actually plays in pod recovery, we ran a direct comparison with the same deployment and the same failure method. In the first round of testing (10 trials), the average recovery time was 3.49 seconds (without AKMF) and 3.48 seconds (with AKMF); in the second round of testing (10 trials), it was 4.02 seconds (without AKMF) and 3.75 seconds (with AKMF). Both rounds showed no statistically meaningful difference between the two conditions. A review of AKMF's own decision log during these time windows revealed no recorded decisions whatsoever, despite the system being fully connected and operating normally throughout. The most likely explanation is that native Kubernetes recovery completes faster than AKMF's monitoring cycle can detect a problem and take action.

This finding is notable in light of the ~12-second recovery figure reported in the original submission, which was based on a single trial (n=1). The repeated-trial results reported here (n=10 per condition, two independent runs) are considered more reliable and representative of AKMF's actual pod-recovery behavior, as shown in Fig. 3.

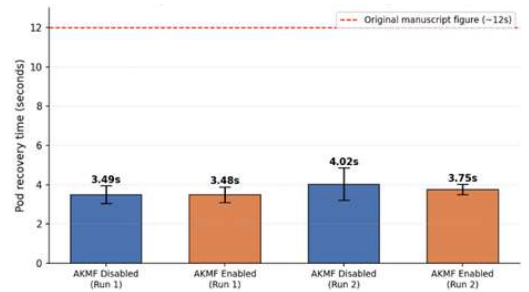


Fig. 3. Pod-crash recovery time: AKMF-disabled vs. AKMF-enabled (n=10/condition, two runs).

5.5 Auto-Scaling Behavior: HPA-AKMF Attribution

In order to test if it could scale on its own (Fig 4), without having to type a scaling command at any time, we created real traffic against the target deployment and observed the results CPU

consumption rose from a 0% baseline to approximately 190% of the configured request well above the 50 percent target and the deployment increased on its own, from 2 replicas to 4 replicas and then 4 to 8 replicas. After the load ceased it reduced itself, without our help, back down to 2 replicas again. This entire process was done by Kubernetes’ own Horizontal Pod Autoscaler and not by AKMF, as can be seen in the audit log that had no corresponding event from AKMF during this time, and as we saw in the recovery test.

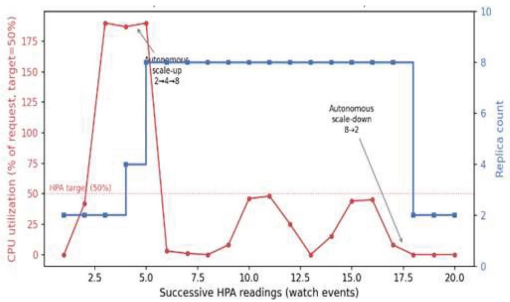


Fig. 4. Autonomous HPA scaling under real HTTP load: CPU utilization and replica count (2→4→8→2) with no manual intervention.

5.6 Component Health

Health status assessment (Fig 6) of the Prometheus monitoring stack revealed that 13 of 17 watched components were operational (76.47 percent health rate), while 4 components remained unavailable. The Monitoring Agent’s core metrics pipeline was unaffected by the missing components, which were limited to supplementary Prometheus exporters.

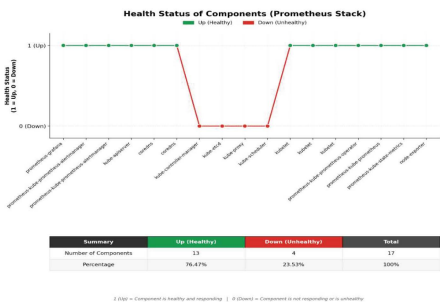


Fig. 5. Component health monitoring results.

5.7 Resource Utilization

At every cycle milestone, the Monitor Agent recorded the AKMF container’s CPU and memory usage. Despite the 48 percent increase in checked workloads , both indicators remained entirely flat throughout all five measurement points, as seen in Fig. 5. Memory remained at 256 MiB throughout, and CPU use remained at 145 milli cores, or 29% of the specified 500 milli core maximum. This constant resource usage is a promising sign, though testing on larger clusters would be needed to confirm the pattern holds at scale.

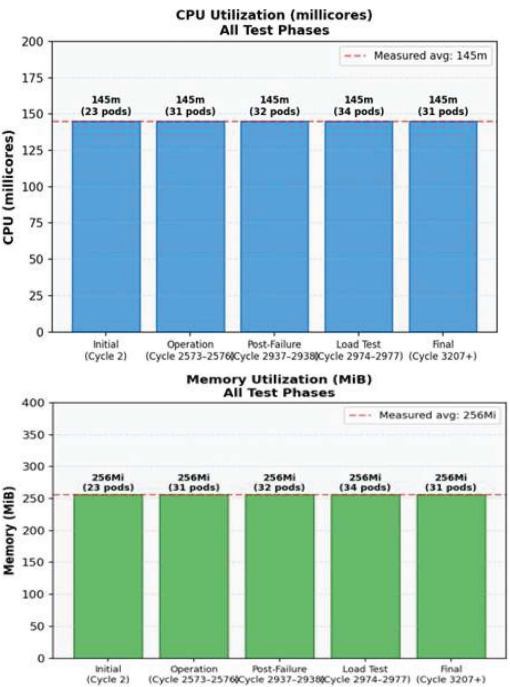


Fig. 6. CPU and Memory Utilization of the AKMF Framework Across All Test Phases.

5.8 Monitoring Dashboard

The final verification attests to the fact that every framework component continued to function during the assessment period. Stable communication with all supporting services allowed for the execution of over 3,200 autonomous choices. These findings confirm that the suggested Agentic Kubernetes framework for autonomous cluster management is dependable, scalable, and useful.

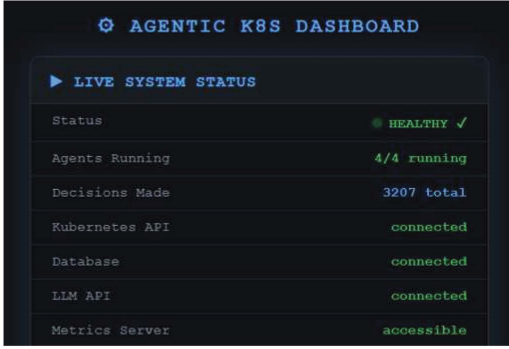


Fig. 7. Agentic Kubernetes Monitoring Dashboard: Live System Status and Connectivity Overview.

5.9 Comparison with Traditional HPA

Table 4 is a comparison of AKMF to the traditional Kubernetes HPA behavior. The reference values for HPA (polling interval, scale stabilization) were taken from official Kubernetes documentation [3] and the decision-cycle timing and monitoring characteristics of AKMF are based on its measurements. Pod recovery is based on a controlled experiment using AKMF with and without AKMF (all other conditions the same: $n=10$ trials/condition, 2 runs), so that it can be directly attributed without relying on external literature. The observed recovery behavior was primarily driven by native Kubernetes reconciliation (not AKMF's decision engine) and no statistically significant difference in recovery time was found between conditions ($p>0.3$).

Note: The values for polling interval and scale-stabilization are taken from the Kubernetes official documentation [3] for the Traditional HPA. In this study the direct measurement of pod-recovery values for AKMF was performed under the controlled, same cluster, same conditions testing (AKMF-disabled vs. AKMF-enabled). Polling interval and AKMF decision-cycle time are of different type of metrics and cannot be compared directly as speed.

Table 4. Comparison of Traditional HPA and AKMF

Metrics	Traditional HPA	AKMF	Ref.
Polling interval	15s	2.34s per cycle	[3]
Scale stabilization	5 min window	Real-time event	[3]
Monitoring scope	CPU/Memory only	API + DB + LLM + Metrics	This study
LLM reasoning	Not supported	Claude Sonnet	This study
Audit logging	Not supported	PostgreSQL full log	This study
Autonomous decisions	Rule-Based only	3,207+ monitoring cycles executed	This study
Pod recovery — AKMF-disabled	—	3.49s - 4.02s ($n=10$ /run, 2 runs)	This study
Pod recovery — AKMF-enabled	—	3.48s - 3.75s; no significant difference disabled ($p>0.3$)	This study

5.10 Limitations and Scope

This evaluation is a proof-of-concept deployment. The pod scale ranged from 23 to 34 pods during testing which will have to be verified at the production cluster scale of several thousand pods. More trials need to be performed in future research to obtain a statistical picture of variation, as there were repeated trials ($n=10$ /condition, two runs) for the failure scenario, but a single trial for the load/scaling scenario, with means and ranges presented here, and standard deviations and failure rates from a larger number of trials for a fuller statistical picture. This evaluation also does not measure the LLM's decision-cycle latency, cost of tokens, or API cost, nor does it describe the structure of prompts and the output-validation

protections the LLM Decision Engine uses.

Documenting all of this would be a necessary future task, especially to determine the cost effectiveness at scale. During testing reported metrics were not affected by the four Prometheus exporters being temporarily unavailable, but subsequent installations should ensure that the whole monitoring-stack is healthy. Finally, this study only examined one type of failure (pod deletion) and one type of load (HTTP traffic), it would be useful to see how many different types of failure and load could be tested to determine the generality of these findings.

VI. Conclusion

The aim of this research is to create and test an Agentic Kubernetes Management Framework with an LLM-powered decision engine comprised of four separate agents. After an initial evaluation, a controlled comparison was conducted which showed no statistically significant difference in pod-recovery time between AKMF-enabled and AKMF-disabled conditions (3.48s - 3.75s vs. 3.49s - 4.02s, n=10 per condition, two independent runs), and demonstrated the true scaling. The framework consistent operations across three evaluation scenarios for over 3,207 decision cycles at a steady cycle time of 2.34 seconds, with a constant overhead of 145 millicores CPU and 256 MiB memory despite a 48% workload increase though this will need to be validated at production scale before drawing strong conclusions. And also, future work will quantify the LLM's contribution to decision latency and operating cost, add repeated-trial statistical validation at larger cluster scale, and evaluate a broader range of failure and load conditions, as detailed in Section 5.10.

In addition, evaluation will be expanded to production-scale clusters of 500 - 5,000 pods, network latency and traffic-engineering agents will be integrated into the multi-agent architecture, formal governance protocols will be developed for safe production rollout, and an open-source

implementation will be made available to enable community validation and expansion of this work.

References

- [1] Cloud Native Computing Foundation (CNCF), "CNCF Annual Survey 2024," CNCF Technical Report, 2024. [Online]. Available: CNCF Annual Survey 2024
- [2] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR, USA: IT Revolution Press, 2016.
- [3] Kubernetes Authors, "Horizontal Pod Autoscaling," Kubernetes Documentation, 2024. [Online]. Available: Horizontal Pod Autoscaling
- [4] A. Wang, B. Chen, C. Liang, F. Wu, X. Wang, and Z. Chen, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024, doi: 10.1007/s11704-024-40231-1.
- [5] Rakuten Mobile and Rakuten Symphony, "Rakuten Mobile and Rakuten Symphony Deploy Intelligent AI-powered RIC Platform in Rakuten's 4G and 5G Open RAN Network in Japan," Press Release, May 2025. [Online]. Available: Rakuten Mobile and Rakuten Symphony press release
- [6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade," *ACM Queue*, vol. 14, pp. 70–93, 2016, doi: 10.1145/2898442.2898444.
- [7] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, Milan, Italy, 2019, pp. 35–45, doi: 10.1109/CLOUD.2019.00020.
- [8] L. Toka and O. Dobrowiecki, "Adaptive scaling of Kubernetes pods," in *Proc. IEEE INFOCOM 2021 Workshops Vancouver*, BC, Canada, 2021, pp. 1–6, doi: 10.1109/INFOCOMWKSHPS51825.2021.9484516.
- [9] R. Boutaba, M. A. Salahuddin, N. Limam, S.

Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun. 2018, doi: 10.1186/s13174-018-0087-2.

- [10] J. Kim, S. Lee, and H. Park, "Intent-based autonomous network management using neural reasoning for software-defined infrastructure," *Journal of Communications and Networks*, vol. 25, no. 3, pp. 312–325, Jun. 2023, doi: 10.23919/JCN.2023.000021
- [11] M. Rosendo, P. Costan, G. Antoniu, and K. Bhattacharya, "Distributed intelligence on the edge-to-cloud continuum: A systematic literature review," *Journal of Parallel and Distributed Computing*, vol. 166, pp. 71–94, Aug. 2022, doi: 10.1016/j.jpdc.2022.04.004.
- [12] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 2023. [Online]. Available: arXiv:2210.03629
- [13] T. Amin, M. Asif, U. Mahmood, and W. Song, "Autonomous PostgreSQL operator for self-healing and scaling in multi-zone Kubernetes environments," *KNOM Review*, vol. 28, no. 2, pp. 67–77, Dec. 2025.

<Field of Interest>: Agentic AI and LLMs, Autonomous and Intent-Based Systems, Cloud-Native Computing and Kubernetes, Software-Defined Networking (SDN), and Internet of Things (IoT).

Wang-Cheol Song



1986 February: B.S in Food Engineering, Yonsei University, Republic of Korea

1989 February: B.S in Electronic Engineering, Yonsei University, Republic of Korea.

1991 February: M.S in Electronic Engineering, Yonsei University, Republic of Korea.

1995 August: Ph.D. in Electronic Engineering, Yonsei University, Republic of Korea.

<Field of Interest>: SDN/NFV, Intent-based networking, and Network management.

Kiran Batool



2021 September: B.S. in Information Technology, Government College University Faisalabad, Pakistan.

2024 September: M.S. in Information Technology,

National University of Sciences and Technology (NUST), Pakistan.

2026 March - Present: Ph.D. in Electronic Engineering, Jeju National University, Republic of Korea.