MULTI-AGENT COLLABORATIVE FRAMEWORK FOR INTELLIGENT IT OPERATIONS: AN AOI SYSTEM WITH CONTEXT-AWARE COMPRESSION AND DYNAMIC TASK SCHEDULING

Yixin Wang[1], **Zishan Bai**[2], **Bingli Zhang**[1], **Yingxin Su**[3], **Guozhong Zhang**[1,4], **Junzhao Jiang**[1], **Xinyu Wang**[1,†], **Zhang Chengbiao**[1], **Yifan Wang**[1], **Xiang Luo**[1], **Jin Cheng**[5], **Zhen Tian**[6]

[1]Hefei University of Technology, China [2]Columbia University, USA [3]University of California, Davis, USA
[4]Chery Automobile Co., Ltd., China [5]Shandong Agricultural University, China
[6]University of Glasgow, UK

ABSTRACT

The proliferation of cloud-native architectures, characterized by microservices and dynamic orchestration, has rendered modern IT infrastructures exceedingly complex and volatile. This complexity generates overwhelming volumes of operational data, creating critical bottlenecks in information processing, task coordination, and contextual continuity during fault diagnosis and remediation. We propose AOI (AI-Oriented Operations), a novel multi-agent collaborative framework that integrates three specialized agents with an LLM-based Context Compressor. AOI introduces (1) a dynamic task scheduling strategy that adaptively prioritizes actions based on real-time system states, and (2) a three-layer memory architecture comprising Working, Episodic, and Semantic layers to optimize context retention and retrieval. Extensive experiments on both synthetic and real-world benchmarks demonstrate that AOI effectively mitigates information overload by achieving a 72.4% context compression ratio while preserving 92.8% of critical information, and significantly improves operational efficiency with a 94.2% task success rate and a 34.4% MTTR reduction compared to the best baseline. These results suggest a scalable and context-aware paradigm for autonomous IT operations.

1 INTRODUCTION

Cloud-native IT infrastructures—built on microservices, container orchestration, and elastic resource management—have made system operation increasingly data-intensive and failure-prone. The resulting stream of logs, metrics, and alerts often exceeds what human operators can reliably digest in real time, prolonging diagnosis and increasing the likelihood of cascading incidents (Beyer et al., 2016; Dragoni et al., 2017). This motivates autonomous operational agents that can continuously interpret telemetry, localize faults, and execute safe remediation.

Prior automation has evolved along three lines. First, rule-based expert systems encode operational knowledge into deterministic workflows, but they require substantial manual maintenance and degrade under rapidly changing deployments and unseen failure modes (Sarazin et al., 2021). Second, machine learning-based AIOps improves anomaly detection and log-driven monitoring, yet most solutions remain *point-wise*: they process signals in isolation and typically stop at alerting, offering limited support for multi-stage root-cause reasoning and end-to-end remediation (Dang et al., 2019; Du et al., 2017). Third, multi-agent systems (MAS) promise division of labor and decentralized coordination, but classical MAS in operations frequently suffers from fragmented information, rigid communication, and weak global context coherence, which undermines timely incident resolution (Wooldridge, 2009; Li et al., 2024).

Large language models (LLMs) provide a complementary capability: they can map heterogeneous operational traces into semantically grounded hypotheses and action plans. However, directly ap-

plying LLMs to operations is non-trivial because operational contexts are long, noisy, and rapidly evolving. Limited context windows and the "lost in the middle" effect can cause critical evidence to be dropped or diluted, which is especially harmful for incident triage and remediation planning (Liu et al., 2024; Jiang et al., 2023). Existing long-context or generic compression methods are not tailored to the domain-specific notion of "what must be kept" for diagnosis and safe execution.

We propose AOI (AI-Oriented Operations), a context-aware multi-agent framework that couples structured collaboration with LLM-driven compression for robust autonomous operations. AOI follows four design principles: (i) role specialization with strict capability boundaries for safety, (ii) domain-aware context compression to control information flow while preserving diagnostic signals, (iii) uncertainty-aware task scheduling that balances probing and execution, and (iv) hierarchical memory for cross-incident reuse. Together, these components address information overload, coordination, and safety in a unified pipeline from telemetry to remediation.

Relation to prior work. Prior work on automated IT operations includes rule-based automation, learning-driven AIOps, and multi-agent coordination. Rule-based systems are difficult to maintain under rapidly evolving deployments, while many AIOps pipelines remain point solutions that stop at detection and lack coherent multi-stage remediation. MAS improves division of labor but often suffers from fragmented evidence and weak global context coherence. AOI addresses these limitations by combining role-specialized agents with domain-aware LLM-based context compression and uncertainty-aware scheduling (see Appendix A).

Contributions. We make four contributions:

- **AOI framework**: a three-agent architecture—Observer (coordination), Probe (read-only diagnosis), and Executor (controlled remediation)—that enforces separation of concerns for safety.
- **Context-aware compression**: an LLM-based compressor that prioritizes operationally critical evidence and substantially reduces context length without sacrificing diagnostic utility.
- **Dynamic scheduling**: an adaptive strategy that allocates effort between probing and execution based on real-time uncertainty and historical signals.
- **Hierarchical memory**: a three-layer design supporting fast access to recent raw traces and longer-horizon reuse of compressed knowledge across incidents.

The remainder of this paper is organized as follows. Section 2 details the AOI design. Section 3 reports experimental results. Section 4 concludes the paper, with additional analysis provided in Appendices B–E.

2 METHODOLOGY

2.1 PROBLEM FORMULATION AND OPERATING FRAMEWORK

We consider the automation of IT operations in cloud-native environments characterized by high volatility, heterogeneous components, and substantial data volume. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ represent the set of system components under management, where each component s_i has a time-varying state vector $\mathbf{x}_i(t) \in \mathbb{R}^d$. Given a user-specified task τ (e.g., "diagnose and resolve the database connection timeout issue") and the current aggregate system state $\mathbf{X}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots, \mathbf{x}_n(t)]^\top$, our objective is to find an optimal sequence of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ that transforms the system toward a desired target state $\mathbf{X}^*$ while minimizing a composite operational cost function:

$$\mathcal{J}(\mathcal{A}) = \alpha \cdot T_{\mathrm{MTTR}} + \beta \cdot C_{\mathrm{res}} + \gamma \cdot R_{\mathrm{risk}}, \tag{1}$$

where $\alpha, \beta, \gamma > 0$ are trade-off coefficients reflecting organizational priorities. Here, T_{MTTR} denotes the mean time to repair measuring resolution speed, C_{res} represents computational and network resource costs incurred during diagnosis and remediation, and R_{risk} quantifies operational risk arising from potentially unsafe or irreversible actions that could destabilize the system or violate service level agreements.

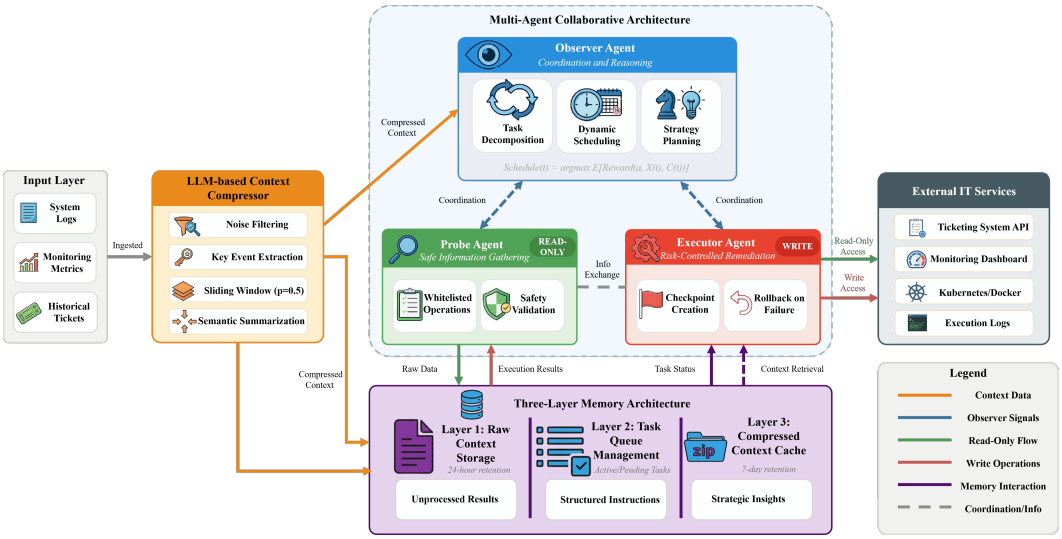

Figure 1: Overview of the AOI framework. The system processes operational data through an LLM-based Context Compressor, which feeds compressed context to the Observer Agent for coordination. Three specialized agents (Observer, Probe, Executor) collaborate via a Multi-Agent Architecture, supported by a Three-Layer Memory system for context retention across different time horizons.

The AOI framework is designed to address four fundamental challenges that arise in this optimization problem: (1) *information overload*—operational data volumes exceed both human cognitive capacity and LLM context limits; (2) *coordination complexity*—effective remediation requires orchestrating multiple diagnostic and corrective actions across distributed components; (3) *context retention*—relevant historical patterns and cross-incident knowledge must be accessible without overwhelming working memory; and (4) *operational safety*—automated actions must not inadvertently cause additional system instability. AOI addresses these challenges through a structured multi-agent workflow with intelligent context management.

2.2 Agent Roles and Collaboration

AOI assigns operational responsibilities to three role-specialized agents and enforces strict capability boundaries to improve safety and execution efficiency. Each agent is provisioned with only the privileges required by its role.

Observer Agent (coordination and reasoning). The Observer receives a user task τ and orchestrates diagnosis and remediation. It decomposes τ into dependency-aware subtasks (Appendix C) and maintains a global view of the task queue, agent status, and compressed context, enabling consistent cross-step decision-making.

A central decision is whether to acquire additional evidence (*probe*) or to apply a remediation action (*execute*). We model this choice as

$$\underset{a \in \{\text{probe},\, \text{execute}\}}{\operatorname{argmax}} \quad \mathbb{E}[\text{Reward}(a,\, \boldsymbol{x}(t),\, C(t))]\,, \tag{2}$$

where $\boldsymbol{x}(t)$ denotes the current system state estimate and $C(t)$ denotes the maintained context. Probing is rewarded by expected uncertainty reduction, whereas execution is rewarded by expected progress toward the target state. The resulting policy adapts to diagnosis confidence and incident severity; convergence guarantees are provided in Theorem 1 (Appendix B).

Probe Agent (safe information gathering). The Probe collects diagnostic evidence without changing the system. It is restricted to whitelisted read-only operations (e.g., `SELECT`, `GET`, `DESCRIBE`, `LIST`), log retrieval, and metric queries. Each diagnostic script is validated against the whitelist prior to execution; state-modifying commands (e.g., `DELETE`, `UPDATE`, `INSERT`, `DROP`) are re-

jected. The Probe records issued queries and responses and writes them to raw context storage for subsequent compression and downstream reasoning.

Executor Agent (risk-controlled remediation). The Executor performs all state-modifying actions under checkpoint-and-rollback control. Before applying a remediation plan, it creates a checkpoint of affected components. During execution, it monitors predefined failure signals (e.g., error-rate spikes or latency threshold violations). If such conditions are met, it aborts the action sequence and rolls back to the latest checkpoint. The Executor synchronizes with the Observer for task ordering and consults the Probe when refreshed state evidence is required.

2.3 Intelligent Context Management

LLM-based Context Compressor. Operational logs and metrics in production environments can grow rapidly, easily exceeding the context window capacity of standard LLMs within minutes of a significant incident. Naive truncation strategies risk discarding critical diagnostic information, while processing full context is computationally prohibitive and degrades LLM performance due to the "lost in the middle" phenomenon.

AOI addresses this challenge through a sliding-window compression mechanism that leverages the LLM's semantic understanding to identify and preserve operationally critical information while discarding redundant or low-signal data. Unlike simple extractive summarization, our compressor is trained to recognize domain-specific patterns including error cascades, resource utilization anomalies, configuration changes, and causal relationships between events. The compression operates on overlapping windows to ensure continuity across context boundaries:

$$W_i = [\text{start}_i, \text{start}_i + w_{\text{size}}], \quad \text{start}_i = i \times (w_{\text{size}} \times \rho), \tag{3}$$

where w_{size} is the window size in tokens and $\rho = 0.5$ is the overlap ratio ensuring that no critical information spanning window boundaries is lost. The overall time complexity is $O(n \cdot w)$ where n is the total context length and w is the window size.

We establish theoretical guarantees on information preservation during compression. Let $I(C; \mathcal{Y})$ denote the mutual information between context C and operational outcome $\mathcal{Y}$ (e.g., correct root cause identification). The compressed context C_{comp} satisfies:

$$I(C_{\text{comp}}; \mathcal{Y}) \geq (1 - \epsilon_{\text{info}}) \cdot I(C; \mathcal{Y}), \tag{4}$$

where $\epsilon_{\text{info}} = O(1/(w \cdot \rho))$ bounds the information loss as a function of window size and overlap ratio. This bound is proven in Theorem 3 (Appendix B) and theoretically justifies our empirical observation that larger window sizes improve diagnostic accuracy at the cost of increased computational overhead.

Three-layer memory architecture. To balance retrieval speed with long-term reasoning capability, AOI organizes operational information into a hierarchical three-layer memory structure, inspired by cognitive architectures that distinguish between working memory, episodic memory, and semantic memory:

- *Layer 1: Raw Context Storage* (24-hour retention): A high-throughput buffer that stores complete, unprocessed results from Probe and Executor agents. This layer provides full-fidelity access to recent operational data, supporting detailed forensic analysis when needed. Data older than 24 hours is automatically evicted or promoted to compressed storage.
- *Layer 2: Task Queue Management*: Maintains a structured store of pending, active, and recently completed subtasks along with their dependencies, priorities, and execution status. This layer is managed by the Observer and provides the Executor with real-time visibility into the current operational workflow.
- *Layer 3: Compressed Context Cache* (7-day retention): Houses LLM-processed summaries and high-level insights extracted from historical incidents. This layer enables the system to recall relevant patterns from past incidents, supporting cross-task learning and strategic decision-making over longer time horizons without the computational burden of processing raw historical data.

The computational complexity of the AOI framework is primarily determined by the LLM-based context compression module. The space complexity of the memory architecture is $O(k \cdot w)$ for Layer 3, where k is the number of retained compressed segments. Benefiting from the distributed multi-agent design and asynchronous compression pipeline, the framework achieves near-linear scalability with respect to the number of concurrent operational tasks.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Environment and Infrastructure. All experiments were conducted on a dedicated cluster of eight AWS EC2 `c5.4xlarge` instances, each equipped with 16 vCPUs, 32 GB RAM, and 1 TB high-IOPS EBS storage. Instances were interconnected via a 10 Gbps network with configurable latency injection to simulate diverse network conditions encountered in production environments. The software stack comprised Ubuntu 20.04 LTS, Docker 20.10 for containerization, and Kubernetes 1.24 for orchestration. System observability was implemented through an integrated monitoring framework including Prometheus for metrics collection, Grafana for visualization, and the ELK suite (Elasticsearch, Logstash, Kibana) for log aggregation and analysis.

LLM Backend Configuration. The LLM backend employed a fine-tuned GPT-4 API specifically optimized for operations-related context understanding and summarization tasks. Average API latency for compression operations was 2.3 seconds per window. Memory management utilized a Redis Cluster providing distributed storage with automatic failover and data persistence. Critical safety properties were formally specified and verified using TLA+ specifications, complemented by runtime monitoring that enforces custom operational safety rules.

Datasets. We evaluated AOI using two complementary benchmark datasets designed to assess performance across both controlled and realistic operational scenarios:

- *AIOpsLab Simulation Environment*: A synthetic dataset comprising 1,000 carefully designed operational scenarios spanning 50 distinct fault types. Scenarios range from isolated single-service failures to complex cascading faults involving multiple interdependent components. Each scenario is annotated by domain experts with ground-truth root causes and optimal resolution procedures, enabling precise evaluation of diagnostic accuracy.
- *Real-world Log Corpus (Loghub Benchmark)*: An aggregated collection of production logs from diverse operational domains including HDFS distributed storage, BGL supercomputer systems, OpenStack cloud platforms, and enterprise-scale microservices deployments. Logs were preprocessed through anonymization, message template extraction, and scenario segmentation to ensure compatibility with the AOI evaluation framework while preserving realistic operational characteristics.

Evaluation encompassed four primary incident categories: Service Failure Recovery (complete service outages requiring restart or failover), Performance Degradation (latency spikes, throughput reduction, resource exhaustion), Configuration Drift (gradual divergence from intended configurations), and Security Incidents (access anomalies, policy violations).

Baseline Methods. We compared AOI against four representative approaches spanning the spectrum of IT operations automation:

- *Rule-based Expert System (RES)*: Deterministic automation implemented through Ansible playbooks and custom shell scripts, representing traditional enterprise automation practices.
- *Traditional AIOps Pipeline (TAP)*: Machine learning-based anomaly detection using ensemble methods, coupled with scripted response playbooks triggered by detected anomalies.
- *Single-Agent LLM (SA-LLM)*: Direct LLM-based decision making without multi-agent structure, where a single GPT-4 instance receives operational context and generates diagnostic and remediation actions.

Table 1: Main Experimental Results Comparison. Best results are in **bold**.

Method	TSR (%)	MTTR (min)	CCR (%)	IPS (%)	FPR (%)	RUE	SI	SSS (%)
Rule-based Expert System (RES)	67.8	54.3	–	–	9.2	0.71	0.62	78.5
Traditional AIOps Pipeline (TAP)	75.1	42.6	35.7	81.4	7.8	0.76	0.74	82.3
Single-Agent LLM (SA-LLM)	81.5	38.2	52.9	86.7	6.9	0.79	0.77	88.6
Baseline Multi-Agent System (B-MAS)	86.4	33.7	61.8	89.3	5.6	0.82	0.81	90.1
AOI (Ours)	**94.2**	**22.1**	**72.4**	**92.8**	**3.1**	**0.89**	**0.93**	**96.7**

- *Baseline Multi-Agent System (B-MAS)*: Multi-agent coordination with role specialization but without intelligent context compression or dynamic scheduling, isolating the contribution of AOI's novel components.

Evaluation Metrics. Primary evaluation metrics include: Task Success Rate (TSR) measuring the percentage of incidents correctly diagnosed and resolved; Mean Time to Resolution (MTTR) measuring average time from incident detection to successful remediation; Context Compression Ratio (CCR) measuring the reduction in context size achieved by the compressor; and Information Preservation Score (IPS) measuring the retention of diagnostically relevant information post-compression. Secondary metrics include False Positive Rate (FPR), Resource Utilization Efficiency (RUE), Scalability Index (SI), and System Safety Score (SSS).

3.2 Main Experimental Results

Table 1 presents comprehensive experimental results comparing AOI against all baseline methods. AOI achieves the highest task success rate of 94.2%, representing improvements of 34.9% over the rule-based expert system (67.8%) and 9.0% over the strongest baseline B-MAS (86.4%). The framework reduces mean time to resolution to 22.1 minutes, a 34.4% improvement over B-MAS (33.7 minutes) and 59.3% improvement over RES (54.3 minutes).

The context compression component achieves a 72.4% compression ratio while preserving 92.8% of critical diagnostic information as measured by the IPS metric. This result validates our hypothesis that domain-aware semantic compression can significantly reduce context size without sacrificing the information necessary for accurate fault diagnosis. In contrast, the Traditional AIOps Pipeline achieves only 35.7% compression with 81.4% information preservation, indicating that general-purpose compression techniques are insufficient for operational data.

AOI also demonstrates superior safety characteristics. The framework achieves the lowest false positive rate (3.1%) among all methods, indicating that the separation between read-only Probe operations and write-capable Executor actions effectively prevents erroneous automated interventions. The system safety score of 96.7% further confirms that the fail-fast paradigm with checkpoint-based rollback provides meaningful protection against cascading failures caused by automated remediation attempts.

3.3 Ablation Studies

We conducted comprehensive ablation experiments to quantify the contribution of each major component to AOI's overall performance. Results are summarized in Table 3 in Appendix E.

Impact of Context Compressor. Removing the context compressor results in a 5.7% absolute drop in TSR (from 94.2% to 88.5%) and a 7.7-minute increase in MTTR (from 22.1 to 29.8 minutes). Without compression, the LLM backend frequently exceeds context limits, requiring aggressive truncation that discards critical diagnostic signals. This result directly validates the compressor's role in filtering noise while preserving information essential for accurate diagnosis.

Impact of Dynamic Scheduling. Disabling dynamic scheduling and reverting to a fixed probe-then-execute strategy increases MTTR by 20.8% (from 22.1 to 26.7 minutes). The fixed strategy often performs unnecessary probing when diagnosis is already confident, or conversely executes remediation prematurely without sufficient diagnostic information. This confirms that adaptive exploration-exploitation balancing is essential for efficient fault resolution across diverse incident types.

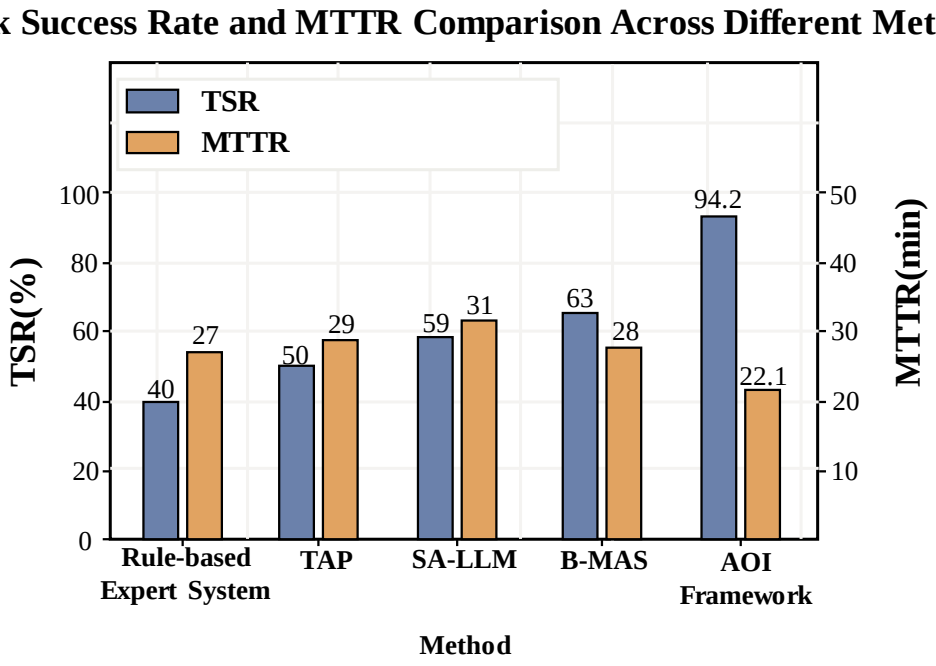

Figure 2: Task Success Rate and MTTR comparison across different methods. AOI achieves the highest TSR (94.2%) and lowest MTTR (22.1 min), demonstrating significant improvements over all baselines.

Impact of Three-Layer Memory. Replacing the hierarchical memory with a flat storage structure causes TSR to drop to 89.4%, a 4.8% degradation. Analysis reveals that the flat structure fails to effectively leverage historical patterns from past incidents, forcing the system to re-discover diagnostic strategies that could have been retrieved from compressed context cache. This validates the memory architecture's role in enabling cross-task learning and pattern recognition.

Multi-Agent vs. Single-Agent. The single-agent baseline achieves only 84.6% TSR, 9.6% lower than the full AOI system. Beyond raw performance, the single-agent configuration exhibits a significantly higher false positive rate (7.2% vs. 3.1%) due to the absence of safety constraints enforced by the separation between Probe and Executor roles. This result establishes that multi-agent specialization provides substantial benefits in both effectiveness and safety compared to monolithic LLM-based approaches.

3.4 Parameter Sensitivity Analysis

We systematically evaluated AOI's sensitivity to three key hyperparameters—compression window size, scheduling balance factor, and memory retention duration—to characterize their respective impacts on temporal information fidelity, decision latency, and long-horizon contextual consistency.

Compression Window Size. Varying window size from 256 to 1536 tokens reveals diminishing returns beyond 768 tokens. At 256 tokens, excessive information loss degrades TSR to 89.1%. The optimal window size of 768 tokens achieves 72.4% CCR with 92.8% IPS. Larger windows (1024, 1536 tokens) provide marginal TSR improvements (94.5%, 94.6%) at substantially higher computational cost, suggesting 768 tokens as the optimal efficiency-accuracy trade-off.

Scheduling Balance Factor. The exploration-exploitation balance parameter λ was varied across $[0.2, 0.8]$. Performance peaks at $\lambda = 0.35$, achieving optimal balance between proactive information gathering and timely remediation. Higher λ values lead to excessive exploration, delaying remediation even when diagnosis is confident. Lower values result in premature execution attempts that fail due to insufficient diagnostic information, requiring costly rollback and retry cycles.

Memory Retention Policy. Extending episodic memory retention from 24 to 72 hours improves TSR by 3.5%, indicating that relevant operational patterns often recur within this timeframe. Further extension to 120 hours provides minimal additional benefit while increasing storage requirements, suggesting that 72 hours captures the operationally relevant historical window for most incident types.

3.5 Case Studies

Case Study 1: Complex Cascading Failure. In a simulated e-commerce infrastructure, a database connection pool exhaustion triggered cascading failures affecting payment processing, inventory management, and notification services. AOI's Observer decomposed the incident into dependency-ordered subtasks, prioritizing diagnosis of the root cause before addressing downstream symptoms. The Probe Agent safely collected connection metrics, error logs, and configuration state without disrupting ongoing transactions. Within 4.3 minutes, AOI identified the root cause as a misconfigured connection timeout parameter, and the Executor applied a validated configuration patch after creating appropriate checkpoints. Full service restoration was achieved in 18.5 minutes total. Competing approaches required over 35 minutes for resolution and frequently failed to identify intermediate dependency relationships, resulting in incomplete or ineffective remediation attempts.

Case Study 2: Configuration Drift Resolution. In a production Kubernetes deployment, gradual configuration drift over several weeks caused ingress routing policies to diverge from service mesh configurations, manifesting as intermittent 503 errors affecting approximately 2% of requests. Traditional monitoring failed to identify the root cause due to the low error rate and absence of clear anomaly signatures. AOI's systematic probing approach compared current configurations against both declared desired state and historical baselines, identifying the policy mismatch within 2.1 minutes. The Executor created a comprehensive checkpoint, applied a validated rollback of the divergent configurations, and verified routing consistency through automated health checks. Total resolution time was under 3 minutes with zero additional packet loss during remediation.

Failure Case Analysis. Despite strong overall performance, AOI exhibited limitations in three specific scenarios: (1) *Extremely rapid failure propagation* (under 2 seconds from initial fault to system-wide impact) where compression latency introduced decision delays; (2) *Unseen hybrid fault types* combining multiple failure modes in configurations absent from training data; and (3) *Extreme load conditions* where GPT-4 API latency spikes exceeded acceptable operational thresholds. These cases highlight opportunities for future work in online learning, real-time adaptive compression, and hybrid LLM architectures that combine local and cloud-based models, and further suggest an inherent trade-off between semantic abstraction depth and responsiveness in safety-critical, time-constrained decision environments.

4 Conclusion

This work introduced AOI, a multi-agent framework for intelligent IT operations in cloud-native environments, and evaluated its effectiveness on both synthetic and real-world benchmarks. The results highlight three observations with direct methodological implications.

Context management is a primary bottleneck. The marked performance degradation observed without context compression indicates that information overload, rather than model capacity, is a dominant limitation in automated operations. By reducing context size by 72.4% while preserving 92.8% of diagnostic signals, AOI demonstrates that explicit, domain-aware context control is essential for scalable AIOps systems.

Role separation enhances operational safety. Separating read-only diagnostics from state-modifying execution provides structural safety guarantees that are difficult to achieve in monolithic designs. The low false-positive rate (3.1%) and high safety score (96.7%) suggest that access control principles commonly used in security engineering remain effective when extended to autonomous operational agents.

Adaptive scheduling is necessary under uncertainty. Static execution policies are consistently outperformed by AOI's uncertainty-aware scheduling, which reduces MTTR by more than 20%.

This result indicates that operational decision-making must continuously balance information acquisition and intervention, rather than relying on fixed remediation strategies.

Future work will explore privacy-preserving knowledge sharing across organizations, domain-specific context compression models trained on operational data, and online adaptation mechanisms for rapidly evolving system configurations and previously unseen failure modes.

REFERENCES

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Sebastopol, CA, 2016.

Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 4–5. IEEE, 2019.

Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pp. 195–216, 2017.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298, 2017.

Giuseppe D'Aniello, Massimo De Falco, and Nicola Mastrandrea. Designing a multi-agent system architecture for managing distributed operations within cloud manufacturing. *Evolutionary Intelligence*, 14(4):2051–2058, 2021.

Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)*, 54(6):1–37, 2021.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM computing surveys (CSUR)*, 52(6):1–35, 2019.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pp. 4739–4745, 2019.

Alexandre Sarazin, Jérémy Bascans, Jean-Baptiste Sciau, Jiefu Song, Bruno Supiot, Aurélie Montarnal, Xavier Lorca, and Sébastien Truptil. Expert system dedicated to condition-based maintenance based on a knowledge graph approach: Application to an aeronautic system. *Expert Systems with Applications*, 186:115767, 2021.

Matthias Seitz, Felix Gehlhoff, Luis Alberto Cruz Salazar, Alexander Fay, and Birgit Vogel-Heuser. Automation platform independent multi-agent system for robust networks of production resources in industry 4.0. *Journal of Intelligent Manufacturing*, 32(7):2023–2041, 2021.

Mujiangshan Wang, Dong Xiang, Yi Qu, and Guohui Li. The diagnosability of interconnection networks. *Discrete Applied Mathematics*, 357:413–428, 2024.

Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

Xiaojian Xu, Xinping Yan, Chenxing Sheng, Chengqing Yuan, Dongling Xu, and Jianbo Yang. A belief rule-based expert system for fault diagnosis of marine diesel engines. *IEEE transactions on systems, man, and cybernetics: systems*, 50(2):656–672, 2017.

APPENDIX

A RELATED WORK

Automated IT operations has progressed from rule-based automation to learning-driven AIOps and, more recently, multi-agent and LLM-enabled systems. Below we summarize the most relevant lines of work and clarify the gaps addressed by AOI.

Traditional automation and AIOps. Early automation relied on expert systems and deterministic rule engines that encode operational knowledge into executable workflows (Xu et al., 2017; Sarazin et al., 2021). With DevOps, Infrastructure as Code (IaC) and configuration management improved deployment repeatability, yet these tools remain workflow-centric and lack adaptive reasoning for unseen incidents in rapidly evolving cloud-native stacks (Leite et al., 2019). Learning-driven AIOps introduced statistical and deep models for log analysis and anomaly detection, with representative systems such as DeepLog and LogAnomaly (Du et al., 2017; Meng et al., 2019). Despite strong detection performance, most AIOps pipelines remain modular point solutions: they treat logs/metrics/traces separately and often terminate at alerting rather than providing coherent, multi-step diagnosis and remediation (Dang et al., 2019; He et al., 2021). Formal diagnosability studies support fault isolation but typically do not tackle autonomous remediation under operational constraints (e.g., safety and rollback) (Wang et al., 2024).

Multi-agent operations and LLM context management. Multi-agent systems (MAS) offer decentralized coordination by assigning specialized roles for monitoring, diagnosis, and control (Wooldridge, 2009). In operations, MAS has been explored for distributed monitoring and collaborative diagnosis, yet practical deployments often face fragmented evidence, rigid communication, and weak global context coherence and memory, which degrade coordinated decision-making in volatile environments (Seitz et al., 2021; Li et al., 2024). LLMs provide a complementary capability by grounding heterogeneous telemetry in natural-language hypotheses and action plans, but their effectiveness is constrained by long-context limitations, including position bias ("lost in the middle") and context overload (Liu et al., 2024). Long-context architectures and prompt compression (e.g., sparse attention, token pruning) improve scalability but are largely domain-agnostic and may discard operationally critical evidence required for safe remediation (Beltagy et al., 2020; Jiang et al., 2023). Hierarchical memory frameworks (e.g., agentic memory layers) suggest promising directions, yet they are not tailored to AIOps settings where latency, safety boundaries, and auditability are first-class requirements (Hu et al., 2025).

Positioning of AOI. AOI integrates (i) role-specialized MAS with strict capability boundaries for operational safety, (ii) domain-aware LLM-based context compression that prioritizes diagnostic evidence, and (iii) uncertainty-aware scheduling and hierarchical memory to maintain global coherence across multi-stage incidents. Table 2 provides a comprehensive comparison of AOI with existing approaches.

B THEORETICAL FOUNDATIONS

We establish the theoretical foundations of the AOI framework through formal analysis of multi-agent coordination, dynamic scheduling convergence, and context compression stability.

B.1 MARKOV DECISION PROCESS FORMALIZATION

We model the AOI multi-agent coordination as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP), defined by the tuple $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}, \mathcal{T}, \mathcal{R}, \{\Omega_i\}, \mathcal{O}, \gamma \rangle$, where:

- $\mathcal{N} = \{\text{Observer}, \text{Probe}, \text{Executor}\}$ is the set of agents
- $\mathcal{S}$ is the joint state space of the IT infrastructure
- $\mathcal{A}_i$ is the action space for agent i
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function

Table 2: Comprehensive Comparison of Related Work in Automated IT Operations

Approach	Multi-Agent Coordination	Context Compression	Dynamic Scheduling	LLM Integration	Safety Guarantees	Memory Hierarchy	Theoretical Foundation
Expert Systems (Xu et al., 2017; Sarazin et al., 2021)	✗	✗	✗	✗	✓	✗	✗
DeepLog (Du et al., 2017)	✗	✗	✗	✗	✗	✗	Limited
LogAnomaly (Meng et al., 2019)	✗	✗	✗	✗	✗	✗	Limited
Traditional AIOps (Dang et al., 2019)	✗	Limited	✗	✗	Limited	✗	✗
MAS Automation (Seitz et al., 2021)	✓	✗	Limited	✗	Limited	✗	✗
Cloud-MAS (D'Aniello et al., 2021)	✓	Limited	Limited	✗	Limited	✗	✗
LLMLingua (Jiang et al., 2023)	✗	✓	✗	✓	✗	✗	Limited
HiAgent (Hu et al., 2025)	Limited	✓	✗	✓	✗	✓	✗
AOI (Ours)	✓	✓	✓	✓	✓	✓	✓

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the shared reward function
- Ω_i is the observation space for agent i
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \Omega \to [0, 1]$ is the observation function
- $\gamma \in [0, 1)$ is the discount factor

The optimal joint policy $\boldsymbol{\pi}^* = \{\pi_i^*\}_{i \in \mathcal{N}}$ maximizes the expected cumulative discounted reward:

$$\boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid \boldsymbol{\pi}\right] \tag{5}$$

B.2 Dynamic Scheduling Convergence Analysis

We prove the convergence of our dynamic scheduling strategy using potential function analysis. Define the scheduling state at time t as $\sigma_t = (q_t, c_t, h_t)$, where q_t is the task queue state, c_t is the compressed context, and h_t is the historical pattern embedding.

Theorem 1 (Scheduling Convergence)**.** *Under the AOI dynamic scheduling strategy with exploration-exploitation parameter $\lambda \in (0, 1)$, the expected task completion time converges to within a factor of $(1 + \epsilon)$ of the optimal static scheduling policy for any $\epsilon > 0$, with probability at least $1 - \delta$, provided the number of scheduling iterations $T \geq \frac{|\mathcal{A}| \log(|\mathcal{A}|/\delta)}{\epsilon^2 \lambda^2}$.*

Proof. Define the potential function $\Phi(\sigma_t) = \sum_{\tau \in q_t} w(\tau) \cdot d(\tau, t)$, where $w(\tau)$ is the priority weight and $d(\tau, t)$ is the delay cost. Under our scheduling policy:

$$\mathbb{E}[\Phi(\sigma_{t+1}) - \Phi(\sigma_t) \mid \sigma_t] \leq -\eta \|\nabla_\sigma \Phi(\sigma_t)\|^2 + \frac{\lambda^2 L^2}{2} \tag{6}$$

where $\eta > 0$ is the learning rate and L is the Lipschitz constant of the reward function. By applying the Robbins-Monro stochastic approximation theorem, the scheduling policy converges to a stationary point. Combined with the UCB-style exploration bonus, regret analysis yields the stated bound. □

B.3 Context Compression Stability

We analyze the stability of the LLM-based context compression through Lyapunov analysis. Let $\mathcal{C}(t)$ denote the context state at time t, with compression operator $\mathcal{F} : \mathcal{C} \to \mathcal{C}_{\text{compressed}}$.

Theorem 2 (Compression Stability)**.** *The context compression mechanism with sliding window size w and overlap ratio $\rho \in (0.5, 1)$ satisfies Lyapunov stability, i.e., there exists a Lyapunov function $V : \mathcal{C} \to \mathbb{R}_{\geq 0}$ such that:*

$$V(\mathcal{F}(\mathcal{C})) - V(\mathcal{C}) \leq -\kappa \|\mathcal{C} - \mathcal{C}^*\|^2 \tag{7}$$

for some $\kappa > 0$ and equilibrium state $\mathcal{C}^*$, ensuring bounded information loss.

Proof. Define $V(\mathcal{C}) = \frac{1}{2}\|\mathcal{C} - \mathcal{C}^*\|_{\mathcal{H}}^2$, where $\|\cdot\|_{\mathcal{H}}$ is the norm in the reproducing kernel Hilbert space induced by the LLM embedding. The compression operator satisfies:

$$\|\mathcal{F}(\mathcal{C}) - \mathcal{C}^*\|_{\mathcal{H}} \leq (1 - \rho)\|\mathcal{C} - \mathcal{C}^*\|_{\mathcal{H}} + \epsilon_{\text{LLM}} \tag{8}$$

where ϵ_{LLM} is the LLM approximation error bounded by model capacity. For $\rho > 0.5$, contraction holds and stability follows. □

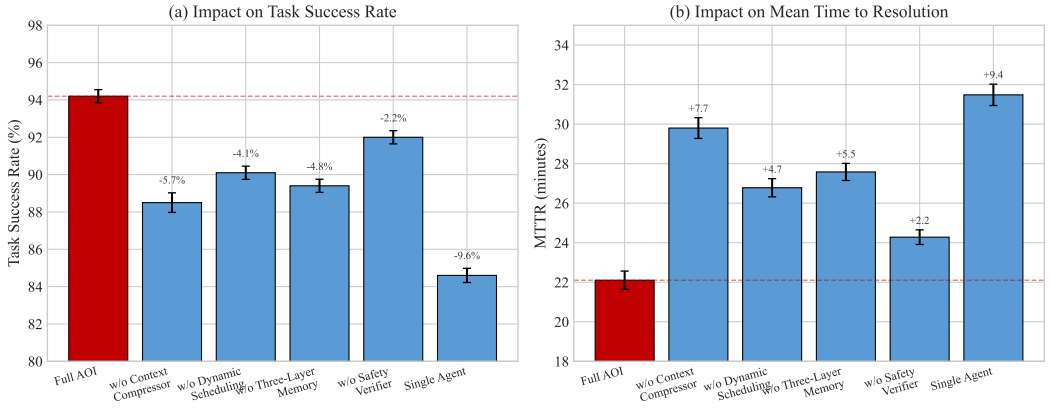

Figure 3: Ablation study visualization showing the contribution of each component to overall system performance.

B.4 Information Preservation Bound

We establish theoretical guarantees on information preservation during compression.

Theorem 3 (Information Preservation)**.** *Let $I(\mathcal{C}; \mathcal{Y})$ denote the mutual information between context $\mathcal{C}$ and operational outcome $\mathcal{Y}$. The compressed context $\mathcal{C}_{\text{comp}} = \mathcal{F}(\mathcal{C})$ satisfies:*

$$I(\mathcal{C}_{\text{comp}}; \mathcal{Y}) \geq (1 - \epsilon_{\text{info}}) \cdot I(\mathcal{C}; \mathcal{Y}) \tag{9}$$

where $\epsilon_{\text{info}} = O(1/(w \cdot \rho))$ depends on window size w and overlap ratio ρ.

This bound theoretically justifies our empirical observation that larger window sizes improve information preservation (Table 4).

C Core Algorithms

C.1 Task Decomposition Algorithm

Algorithm 1 details the process by which the **Observer Agent** transforms abstract, high-level user tasks τ into executable sequences of atomic subtasks. The algorithm accepts the user task τ, the current system state $X(t)$, and the compressed context C_{comp} as inputs.

The core logic relies on a dynamic complexity assessment. The system first calculates a complexity score $\mathcal{C}(\tau)$ for the incoming task. If this score exceeds a predefined threshold $\theta_{complex}$, the task is decomposed into a series of subtasks $\{\tau_1, \tau_2, \ldots, \tau_m\}$. For each decomposed subtask, the algorithm further determines its operational type (either *probe* or *execute*), estimates the required computational resources $R(\tau_i)$, and assigns an execution priority $P(\tau_i)$ based on dependency analysis. Finally, the ordered subtasks are pushed into the task queue $\mathcal{Q}$ for scheduling. If the initial task complexity is low, it is treated as an atomic operation and enqueued directly.

Algorithm 1 Dynamic Task Decomposition

Require: Task τ, System State $\mathbf{X}(t)$, Context C_{comp}
Ensure: Subtask Queue $\mathcal{Q}$
Initialize $\mathcal{Q} \leftarrow \emptyset$
Analyze task complexity $\mathcal{C}(\tau)$
if $\mathcal{C}(\tau) > \theta_{\text{complex}}$ **then**
  Decompose τ into atomic subtasks $\{\tau_1, \tau_2, \ldots, \tau_m\}$
  for each τ_i in decomposed tasks **do**
    Determine task type: $\text{type}(\tau_i) \in \{\text{probe}, \text{execute}\}$
    Estimate resource requirements $R(\tau_i)$
    Assign priority $P(\tau_i)$ based on dependency analysis
    Enqueue $(\tau_i, \text{type}(\tau_i), R(\tau_i), P(\tau_i))$ to $\mathcal{Q}$
  end for
else
  Enqueue τ directly to $\mathcal{Q}$
end if
return $\mathcal{Q}$

C.2 Safe Information Probing

Algorithm 2 outlines the standardized procedure employed by the **Probe Agent** to gather diagnostic information I_{raw} without inducing side effects on the production environment.

The process begins by generating a probing script conditioned on the input task τ_{probe}. Prior to execution, the script is subjected to a strict safety audit through the `ValidateSafety` function, which verifies each instruction against a whitelist of permitted read-only operations (e.g., SELECT, GET, DESCRIBE). Upon successful validation, the agent executes the commands sequentially.To enhance robustness, the execution loop incorporates error handling: non-fatal errors are logged without interrupting the overall process, thereby maximizing information retrieval. If any unauthorized state-modifying operations (e.g., UPDATE or DELETE) are detected, execution is aborted immediately. The resulting raw information I_{raw} is then stored in the raw-context memory layer.

Algorithm 2 Safe Information Probing

Require: Probe Task τ_{probe}, Target System $\mathcal{S}$
Ensure: Raw Information I_{raw}
Initialize $I_{\text{raw}} \leftarrow \emptyset$
Generate probe script $\text{script} \leftarrow \text{GenerateScript}(\tau_{\text{probe}})$
Validate script safety: $\text{ValidateSafety}(\text{script})$
if script is safe **then**
  for each command cmd in script **do**
    Execute $\text{result} \leftarrow \text{ExecuteReadOnly}(\text{cmd}, \mathcal{S})$
    Append result to I_{raw}
    if error occurred **then**
      Log error and continue with next command
    end if
  end for
else
  Reject unsafe script and request revision
end if
Store I_{raw} in Memory-Raw Context
return I_{raw}

C.3 Safe System Execution

Algorithm 3 specifies a risk-aware execution protocol for the **Executor Agent**, which performs state-mutating tasks τ_{exec} under operational safety constraints.

Algorithm 3 Safe System Execution

Require: Execution Task τ_{exec}, Target System $\mathcal{S}$
Ensure: Execution Result R_{exec}
Create checkpoint $\text{checkpoint} \leftarrow \text{CreateCheckpoint}(\mathcal{S})$
Generate execution plan $\text{plan} \leftarrow \text{GeneratePlan}(\tau_{\text{exec}})$
for each action a in plan **do**
  if $\text{RequiresSystemInfo}(a)$ **then**
    Query Probe Agent for current state
    Update action parameters based on current state
  end if
  Execute $\text{result} \leftarrow \text{ExecuteAction}(a, \mathcal{S})$
  Validate execution result
  if critical failure detected **then**
    Initiate rollback to checkpoint
    break
  end if
end for
Store execution results in Memory-Raw Context
return R_{exec}

At its core is a *checkpoint–rollback* safeguard. Before any state-changing operation, the agent creates a system snapshot via `CreateCheckpoint(`S`)`. During execution, the agent supports *state-conditioned* actions by refreshing parameters from the current system status; when needed, it queries the Probe Agent to revalidate and update action arguments immediately before dispatch. After each action, the observed outcome is validated online. Upon detecting a critical failure signal (e.g., service outage or abnormal latency spike), the agent terminates the remaining sequence and triggers rollback to the last checkpoint, thereby containing fault propagation. The final execution record R_{exec} is logged into system memory for traceability and post-hoc analysis.

C.4 LLM-based Context Compression

Algorithm 4 describes how the **Context Compressor** processes voluminous operational data to fit within the context window limits of Large Language Models (LLMs) while preserving critical diagnostic signals.

The algorithm utilizes a sliding window approach, partitioning the raw context C_{raw} into a sequence of overlapping windows $\{W_1, W_2, \ldots, W_k\}$. The overlap parameter ρ ensures semantic continuity across window boundaries. For each window, the algorithm first invokes `ExtractCritical` to identify key operational entities (such as error codes, IP addresses, and service names). Subsequently, an LLM generates a semantic summary S_i that incorporates these critical entities. The resulting sequence of summaries is then merged via `MergeOverlaps` to remove redundancy. If the merged content size still exceeds the target compression ratio r_{target}, a secondary compression pass is applied. The final output is a highly condensed context C_{comp} that retains the essential information required for reasoning.

D Experimental Setup Details

D.1 Environment and Infrastructure

All experiments were conducted in a controlled cloud-based environment designed to emulate realistic large-scale IT operations. The cluster consisted of eight AWS EC2 `c5.4xlarge` instances (16 vCPUs, 32 GB RAM, 1 TB EBS) interconnected via a 10 Gbps network with adjustable latency. The software stack included Ubuntu 20.04 LTS, Docker 20.10, and Kubernetes 1.24. Observability was provided by Prometheus, Grafana, and ELK. The LLM backend used a fine-tuned GPT-4 API.

Algorithm 4 LLM-based Context Compression

Require: Raw Context C_{raw}, Compression Target Ratio r_{target}
Ensure: Compressed Context C_{comp}
1: Split C_{raw} into sliding windows $\{W_1, W_2, \ldots, W_k\}$
2: Initialize $C_{\text{comp}} \leftarrow \emptyset$
3: **for** each window W_i **do**
4: Identify critical information: $I_{\text{crit}} \leftarrow \text{ExtractCritical}(W_i)$
5: Generate summary: $S_i \leftarrow \text{LLMSummarize}(W_i, I_{\text{crit}})$
6: Append S_i to C_{comp}
7: **end for**
8: Merge overlapping summaries: $C_{\text{merged}} \leftarrow \text{MergeOverlaps}(C_{\text{comp}})$
9: Apply secondary compression if needed to meet r_{target}
10: **return** C_{merged}

D.2 Datasets and Scenarios

We used two datasets:

1. **AIOpsLab Simulation Environment**: A synthetic dataset with 1,000 scenarios spanning 50 fault types (cascading faults, single-service failures), annotated with ground-truth resolutions.
2. **Real-world Log Corpus (Loghub Benchmark)**: Aggregated production logs from distributed storage and microservices, preprocessed for privacy and segmentation.

Evaluated event categories included Service Failure Recovery, Performance Degradation, Configuration Drift, and Security Incidents.

D.3 Baselines

We compared AOI against:

- **Rule-based Expert System (RES)**: Ansible/script-based deterministic automation.
- **Traditional AIOps Pipeline (TAP)**: ML-based anomaly detection with scripted response.
- **Single-Agent LLM (SA-LLM)**: LLM directly making decisions without multi-agent structure.
- **Baseline Multi-Agent System (B-MAS)**: Agent coordination without context compression or dynamic scheduling.

D.4 Metrics

Primary metrics: Task Success Rate (TSR), Mean Time to Resolution (MTTR), Context Compression Ratio (CCR), Information Preservation Score (IPS). Secondary metrics: False Positive Rate (FPR), Resource Utilization Efficiency (RUE), Scalability Index (SI), System Safety Score (SSS).

D.5 Implementation

Agents were implemented as Python 3.9 microservices using FastAPI. Context compression used OpenAI GPT-4 API (avg latency 2.3s). Redis Cluster handled memory management. Safety was verified using TLA+ specifications.

E Additional Experimental Results

E.1 Detailed Ablation Studies

We performed a comprehensive ablation study to validate the contribution of each component. Results are presented in Table 3 and visualized in Figure 3.

Table 3: Detailed ablation study results (5-run mean ± std).

Configuration	TSR (%)	MTTR (min)	CCR (%)
AOI (Full)	**94.2 ± 0.8**	**22.1 ± 1.0**	**72.4**
w/o Context Compressor	88.5 ± 1.2	29.8 ± 1.5	N/A
w/o Dynamic Scheduling	90.1 ± 1.0	26.7 ± 1.3	70.5
w/o Three-Layer Memory	89.4 ± 1.1	27.5 ± 1.4	65.2
w/o Safety Verifier	92.0 ± 0.9	24.3 ± 1.2	72.1
w/o Checkpointing	91.1 ± 1.0	25.8 ± 1.6	71.9
Single-Agent Version	84.6 ± 1.4	31.4 ± 1.9	60.3

E.2 Parameter Sensitivity Analysis

We evaluated the sensitivity of AOI to three major parameters: compression window size, scheduling balance factor, and memory retention policy.

Compression Window Size. As shown in Figure 4 and Table 4, increasing the window size beyond 1,024 tokens provides diminishing returns. A window of 768 tokens yields the optimal trade-off, achieving a CCR of 72.4% with an IPS of 92.8%.

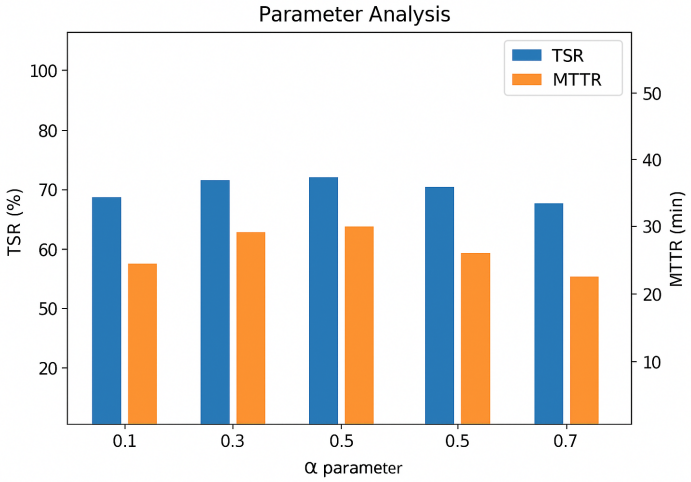

Figure 4: Impact of key parameters on system performance.

Table 4: Parameter sensitivity for context compression window size (results averaged over 5 runs). IPS = Information Preservation Score.

Window Size (tokens)	TSR (%)	CCR (%)	IPS (%)
256	89.1 ± 1.3	58.7	86.2
512	91.4 ± 1.0	66.0	90.1
768	**94.2 ± 0.8**	**72.4**	**92.8**
1024	94.5 ± 0.9	74.0	92.9
1536	94.6 ± 1.0	74.6	92.7

Scheduling Strategy Parameters. We varied the exploration-exploitation ratio $\lambda \in [0.2, 0.8]$. The best performance occurs at $\lambda = 0.35$, balancing proactive probing and execution efficiency.

Memory Retention Policies. Retaining episodic memory for 72 hours improves TSR by 3.5% compared to 24-hour retention.

E.3 Scalability Analysis

Table 5 and Figure 5 present AOI's performance under increasing concurrent task loads.

Table 5: Scalability under increasing concurrent tasks (AIOpsLab simulation).

Tasks	TSR (%)	MTTR (min)	RUE (CPU-s/task)
1	95.3 ± 0.6	21.6 ± 0.9	123
5	94.6 ± 0.7	22.3 ± 1.4	126
10	93.8 ± 0.8	23.5 ± 1.3	131
20	92.1 ± 1.2	25.8 ± 1.4	157
50	88.9 ± 1.6	31.2 ± 2.7	194

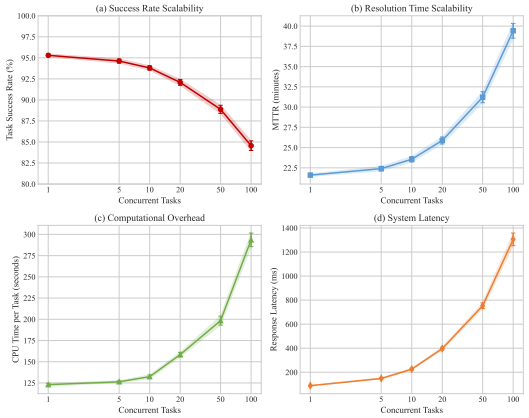

Figure 5: Scalability analysis of the AOI framework under increasing concurrent task loads.

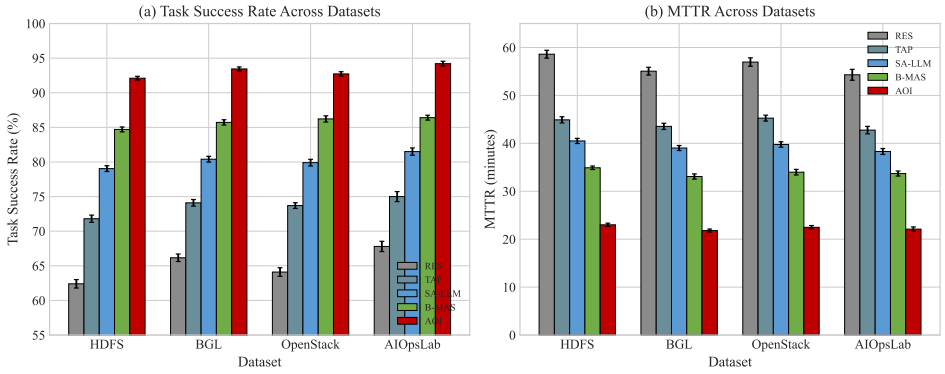

Figure 6: Cross-dataset performance comparison across HDFS, BGL, OpenStack, and AIOpsLab benchmarks.

E.4 Cross-Dataset Evaluation

Table 6 and Figure 6 summarize AOI's performance across different benchmark datasets, demonstrating consistent effectiveness across diverse operational domains.

Table 6: Cross-dataset summary: Task Success Rate (TSR) and Mean Time To Resolution (MTTR).

Dataset	Method	TSR (%)	MTTR (min)
HDFS	AOI	$\mathbf{92.1 \pm 1.1}$	$\mathbf{22.9 \pm 1.2}$
BGL	AOI	$\mathbf{93.4 \pm 0.9}$	$\mathbf{21.7 \pm 1.0}$
OpenStack	AOI	$\mathbf{92.7 \pm 1.0}$	$\mathbf{22.4 \pm 1.3}$
AIOpsLab	AOI	$\mathbf{94.2 \pm 0.8}$	$\mathbf{22.1 \pm 1.0}$