Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 4, Issue 2, 2026

TRANSFORMING NOC OPERATIONS THROUGH AI-AUGMENTED ALERT TRIAGE AND ESCALATION AUTOMATION

Hurair Ahmad

Department of Artificial Intelligence, University of Management and Technology (UMT), Pakistan

hurairahmad711@gmail.com

DOI: https://doi.org/10.5281/zenodo.18640396

Keywords
Alert Management, Large Language Models (LLMs), Ollama, Escalation Automation, Prompt Engineering, APScheduler, RabbitMQ, PostgreSQL, Webhook

Article History
Received: 15 December 2025
Accepted: 30 January 2026
Published: 14 February 2026

Corresponding Author: Hurair Ahmad

Abstract

Modern Network Operations Centers (NOCs) face significant challenges in managing the large volume of alerts generated by diverse monitoring systems. Manual triage processes, delayed escalation, and the absence of contextual intelligence often lead to prolonged incident resolution times and service degradation. This research proposes an AI-powered NOC Alert Triage and Escalation System that integrates microservice architecture, automated escalation mechanisms, and Large Language Model (LLM)-based analysis to improve alert handling efficiency. The proposed system leverages a FastAPI-based webhook service for real-time alert ingestion, PostgreSQL for persistent storage, RabbitMQ for asynchronous communication, and an Ollama-based LLM service for incident summarization and contextual knowledge enrichment.

Automated escalation is managed through a persistent scheduling mechanism to ensure reliability, even during system restarts. The Experimental evaluation demonstrates a reduction in the Mean Time to Acknowledge (MTTA), improved alert reduplication accuracy, and enhanced incident understanding through AI-generated summaries. The system is scalable, fault-tolerant, and customizable, making it suitable for enterprise-level NOC environments.

INTRODUCTION

2.1 Background

Network Operations Centers (NOCs) form the backbone of modern digital infrastructure, particularly within enterprise IT environments, cloud service providers, and large-scale telecommunications organizations. With the exponential growth of networked devices, applications, and distributed services, both the volume and complexity of operational alerts have increased significantly. Monitoring systems continuously generate alerts related to configuration issues, hardware failures, security incidents, and performance degradation. Although these alerts are essential for maintaining service reliability, their sheer volume often overwhelms human operators Traditional NOC environments primarily rely on manual classification processes static threshold-based alerting, and human driven escalation protocols. However, these approaches are increasingly inadequate to manage the complexity of contemporary network ecosystems. Alert fatigue has become a widespread issue, where excessive non-critical notifications reduce operators' ability to focus on high-priority incidents. Research indicates that alert fatigue contributes to increased Mean Time to Acknowledge (MTTA) and Mean Time to Resolve (MTTR), directly affecting service-level agreements (SLAs) and customer satisfaction. The motivation for this research stems from the need to modernize NOC alert management through

intelligent automation, scalable system architecture, and contextual decision support. Recent advances in artificial intelligence, particularly Large Language Models (LLMs), offer new opportunities to enhance incident understanding through automated reasoning, summarization, and knowledge retrieval. Despite these advances, the practical integration of such technologies into real-world NOC workflows remains limited.

2.2 Problem Statement

Despite the availability of advanced monitoring platforms, most NOCs continue to face three primary challenges: (1) inefficient alert triage, (2) delayed or inconsistent escalation, and (3) lack of actionable context during incident response. Alerts frequently arrive as isolated events without sufficient diagnostic details, requiring operators to manually execute troubleshooting command, consult documentation, and correlate historical incidents. This manual process is time consuming, error prone, and highly dependent on individual expertise.

Additionally, many existing alert management systems are monolithic and tightly coupled, limiting scalability and hindering integration with AI-driven components. Escalation mechanisms are often implemented using in-memory schedulers or manual workflows, which are unreliable during system restarts or unexpected failures. These constraints emphasize the need for a resilient, intelligent, and extensible alert management framework.

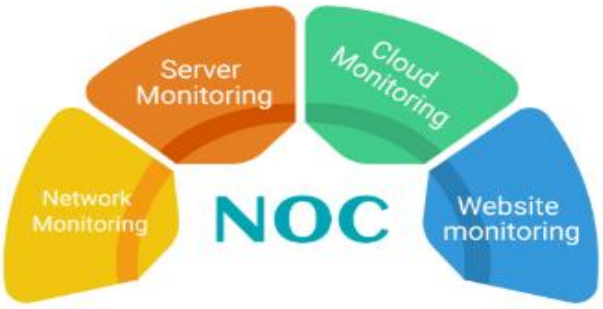

Figure 1: Core monitoring domains handled by a Network Operations Center (NOC), including network, server, cloud, and website monitoring

2.3 Research Objectives

The primary objective of this research is to design and evaluate an AI-powered NOC alert triage and escalation system that overcomes the limitations of conventional approaches. The specific objectives are as follows.

1. Design a microservice-based architecture that enables scalable, modular, and fault-tolerant alert processing.
2. Implement automated alert triage, reduplication, and time-based escalation mechanisms.
3. Integrate LLM-based incident summarization and contextual knowledge enrichment into the alert life-cycle.
4. Evaluate the impact of automation and AI integration on operational efficiency using quantitative performance metrics.

2.4 Research Questions

This study is guided by the following research questions:

- How does a microservice-based architecture improve the scalability and reliability of NOC alert management systems?
- To what extent does automated escalation reduce response times compared to manual procedures?
- How effectively do LLM-generated summaries enhance situational awareness and decision making for NOC operators

2.5 Structure of the Paper

The remainder of this paper is organized as follows. Section [2] reviews related work on alert management, escalation strategies, and AI-driven incident analysis. Section [3] describes the architecture of the system and the research methodology. Section [4] outlines the implementation details and experimental setup. Section [5] presents the results and performance evaluation. Section [6] discusses the key findings and practical implications. Section [7] highlights limitations and future research directions. Finally, Section [8] concludes the paper.

3 Literature Review

Early Network Operations Center (NOC) systems were primarily built around Simple Network Management Protocol (SNMP) traps, syslog messages, and static threshold-based alarms. These systems focused on detecting deviations from predefined performance metrics, such as CPU utilization, memory consumption, or packet loss. Although effective for small scale and relatively stable networks, they lacked the adaptability required for modern, dynamic, and heterogeneous environments.

To address alert overload, alert correlation engines were introduced to group related alarms and suppress duplicates. However, most correlation mechanisms relied on manually defined rules that required continuous maintenance and domain expertise. As network infrastructures evolved in scale and complexity, rule-based systems struggled to adapt to new device types, distributed architectures, and emerging failure patterns. Alert fatigue has been extensively studied in both the healthcare and IT operations domains. Research indicates that high volumes of false positives significantly reduce operator response and increase cognitive load. In large-scale NOC environments, operators may receive hundreds of alerts per hour, many of which are redundant or non-actionable. This overload often

results in delayed responses to critical incidents and an increased likelihood of human error Several mitigation strategies, including alert prioritization and severity reclassification, have been proposed. However, these approaches frequently depend on static severity definitions that do not adequately reflect the real-time operational context. Consequently, even high severity alerts may lack sufficient diagnostic detail to enable rapid resolution. Machine learning (ML) techniques have been applied to anomaly detection, root cause analysis, and alert correlation. Supervised and unsupervised models have demonstrated effectiveness in identifying abnormal patterns and reducing false positives. Clustering algorithms and graph-based approaches are commonly used to correlate alerts across interconnected network components. Despite these advances, many ML based solutions primarily focus on detection rather than response management. They often function as standalone modules that generate additional alerts or risk scores, inadvertently increasing the complexity of the system. Integration with escalation workflows and operator-facing decision-support interfaces remains limited. Large Language Models (LLMs) have recently demonstrated strong capabilities in natural language understanding, reasoning, and summarization. Emerging research explores their application in log analysis, incident summarization, and conversational IT support systems. LLMs can synthesize information from alerts, logs, and documentation to generate human readable explanations that improve situational awareness.

However, challenges related to latency, reliability, and explainability must be addressed before deploying LLMs in real-time operational systems. Retrieval-Augmented Generation (RAG) has been proposed as a method to ground LLM outputs in verified knowledge bases, thereby improving factual consistency and reducing hallucinations.

While prior research addresses isolated components of alert management such as detection, correlation, or summarization there is a gap in comprehensive systems that integrate automated escalation, AI-driven contextual analysis, and resilient architectural design within a unified framework.

This research addresses this gap by proposing an end-to-end NOC alert triage and escalation system that combines microservices, asynchronous event driven processing, persistent scheduling, and LLM based intelligence. The proposed approach emphasizes scalability, reliability, and contextual awareness, which are critical requirements for modern enterprise level NOC environments

4 Methodology

4.1 Research Design

This research adopts a system-oriented quantitative design focused on the development, implementation and evaluation of an AI-powered alert triage and escalation framework for Network Operations Centers (NOCs). Instead of proposing a purely conceptual model, this study presents a fully functional prototype deployed in a controlled experimental environment.

The system performance is evaluated using quantitative operational metrics, including Mean Time to Acknowledge (MTTA), escalation accuracy, alert reduplication rate, and notification reliability.

The methodology is structured into four primary phases:

1. System requirement analysis
2. Architectural design
3. Implementation of alert processing, escalation mechanisms, and AI-driven analysis
4. Experimental evaluation

This phased structure ensures traceability between research objectives, architectural components, and evaluation outcomes.

4.2 System Architecture Overview

The proposed solution follows a microservice-based architecture to ensure modularity, scalability, and fault tolerance. The core components include:

- A FastAPI-based alert ingestion service for normalization, persistence, and orchestration.
- An asynchronous AI processing service for LLM-based summarization and embedding generation.
- RabbitMQ for non-blocking inter-service communication.
- PostgreSQL for persistent storage of alerts and escalation schedules.
- A multi-channel notification layer supporting email and WhatsApp alerts.

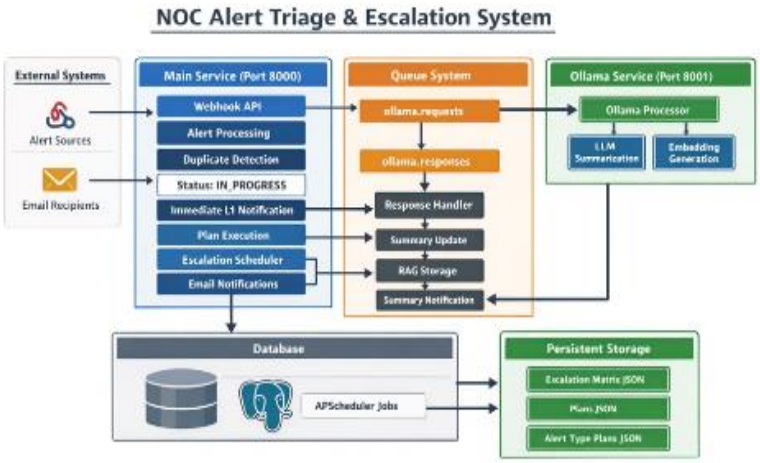

Figure 2: Overall architecture of the AI-powered NOC alert triage and escalation system

This architectural separation guarantees that time-critical ingestion and escalation processes are never blocked by computationally intensive AI operations.

Alerts are ingested through a RESTful webhook endpoint implemented using FastAPI.

Incoming alerts conform to a predefined schema and are normalized to ensure consistency across heterogeneous monitoring sources. The normalization process standardizes severity levels, timestamps, device identifiers, and team ownership metadata. Alerts are classified into actionable and non-actionable categories using predefined critical alert patterns. Non-critical alerts are stored for auditing purposes but marked as invalid to prevent unnecessary operator intervention. This early filtering mechanism significantly reduces alert noise. Additionally, alerts are written to structured text files during ingestion, creating an immutable snapshot of

the payload for forensic analysis, offline review, and regulatory compliance.

Once persisted, alerts are forwarded to an orchestrator module responsible for managing the alert lifecycle. Duplicate detection is performed using external alert identifiers. If a duplicate is detected, it is marked accordingly and excluded from further processing. Valid alerts are immediately transitioned to the IN-PROGRESS state and assigned an initial escalation level based on team-specific configurations, ensuring early ownership and accountability.

A key contribution of this research is the implementation of a persistent, time-based escalation mechanism using APScheduler with a database-backed job store. Escalation policies are defined in an external JSON-based escalation matrix, allowing administrators to configure time thresholds and notification recipients per team and escalation level. When an alert enters the IN-PROGRESS state, escalation jobs are dynamically scheduled. If the alert remains unresolved within the configured time threshold, it is automatically escalated (e.g., from Level 1 to Level 2), and notifications are triggered accordingly. Since the scheduler uses persistent storage, all escalation jobs survive service restarts, ensuring operational reliability.

The system also supports agentic execution of predefined diagnostic plans. Each alert category is mapped to a structured execution plan containing diagnostic commands and verification steps. Plans are contextualized using alert metadata, and execution results are captured for downstream analysis. Integration with external network management platforms enriches alerts with topology-level and device-specific context, improving situational awareness and enabling more accurate root cause analysis.

Alert data, diagnostic outputs, and enrichment results are asynchronously forwarded to an LLM-based AI service. The AI component generates structured, human-readable incident summaries and stores vector embeddings in a Retrieval-Augmented Generation (RAG) knowledge base. Notification delivery is handled through email and WhatsApp channels. Initial alerts are sent to Level 1 responders, while subsequent notifications are triggered automatically during escalation stages. Resolved alerts are archived after a configurable retention period. Archived records are stored separately to maintain database performance while preserving historical data for compliance and analytical evaluation.

Figure 3: Alert status lifecycle in persistent storage

4.3 Evaluation Metrics

The system performance is evaluated using the following quantitative metrics: Mean Time to Acknowledge (MTTA):

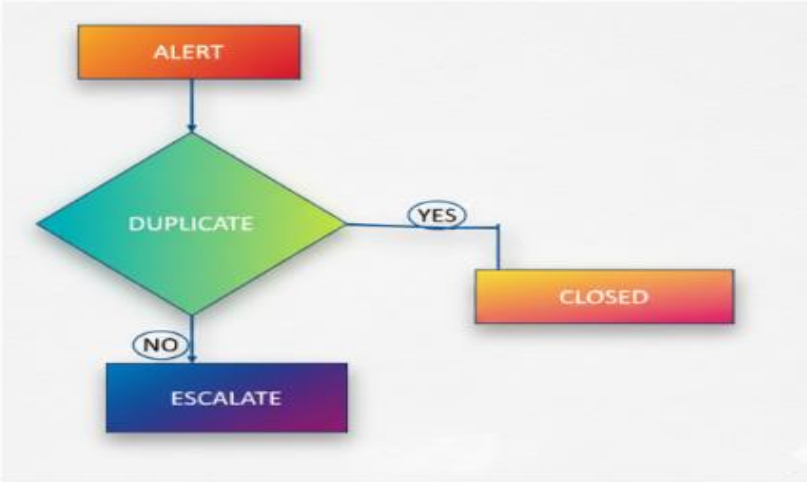

Figure 4: Duplicate alert handling workflow

Deduplication Rate:

$$\textbf{\textit{Deduplication}} = \frac{\textit{Number of duplicates detected}}{\textit{Total alerts}} \times 100 \quad (1)$$

Escalation Accuracy:

$$\textbf{\textit{Escalation accuracy}} = \frac{\textit{Correct escalations}}{\textit{Total escalations}} \times 100 \quad (2)$$

Notification Reliability:

$$\textbf{\textit{Delivery rate}} = \frac{\textit{Successful notifications}}{\textit{Total notifications}} \times 100 \quad (3)$$

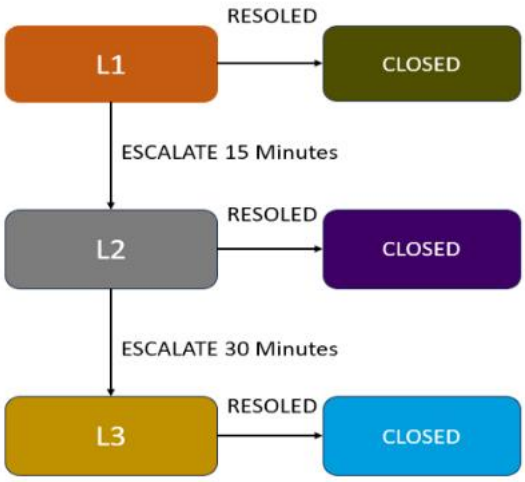

Figure 5: Execution of time-based escalation matrix

5 Implementation

5.1 Asynchronous Message Queue Design

The alert processing pipeline relies on an asynchronous message queue implemented using

RabbitMQ and the aio-pika client library. The queue-based design decouples alert ingestion from alert processing, enabling horizontal scalability and fault isolation. The queue.py module defines both a publisher and a consumer, ensuring reliable delivery and processing of alert messages.

The publish alert function serializes incoming alert payloads in JSON format and publishes them to a durable message queue with persistent delivery mode enabled. This guaranty that alerts are not lost in the event of broker or service failures. The use of connect robust further enhances system resilience by automatically re-establishing the connection to the message broker if an interruption occurs.

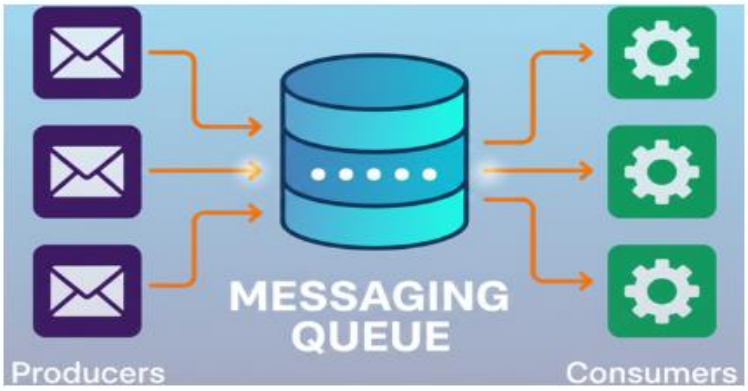

Figure 6: Asynchronous queue communication using RabbitMQ.

Table 1: Methodological Phases and System Components

Phase	Associated Components
Requirement Analysis	Alert lifecycle modeling, escalation matrix
Architectural Design	Microservices, RabbitMQ, PostgreSQL
Implementation	FastAPI, APScheduler, LLM Integration
Evaluation	MTTA, Reduplication Rate, Escalation Accuracy

5.2 Alert Processing Orchestration

Once a message is consumed from the queue, the payload is passed to the process alert function within the orchestrator service. This function represents the core of the alert lifecycle management logic. It performs validation, reduplication, state transitions, and initiates downstream workflows such as escalation scheduling and agentic plan execution.

The orchestrator operates within an asynchronous SQLAlchemy session. This design enables non-blocking database interactions while maintaining transactional consistency. After successful processing,

the session is explicitly committed to persist all state changes. In the event of an exception, message acknowledgment is deferred, allowing built-in retry mechanisms to handle transient failures.

5.3 Ollama Response Handling Architecture

To integrate Large Language Model (LLM) capabilities without impacting real-time alert handling, AI-related tasks are processed asynchronously through a dedicated response handler. The OllamaResponseHandler class consumes responses from a separate queue (ollama.responses) and processes summarization and embedding results independently of the main alert pipeline.

Each AI request is associated with a unique request identifier and alert identifier. This mapping allows responses to be correlated with the originating alert. Upon receiving a response, the handler inspects the request type (summarization or embedding) and delegates processing to the appropriate internal method

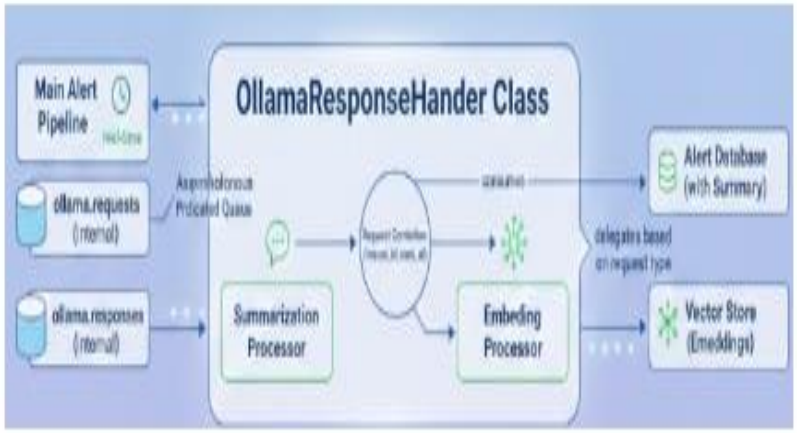

Figure 7: Ollama AI responses stored in the database

5.4 AI-Generated Incident Summarization

For summarization responses, the handler retrieves the corresponding alert record from the database and updates it with the generated summary. This summary provides a concise, human-readable description of the incident, combining alert metadata, diagnostic outputs, and contextual reasoning produced by the LLM.

After persisting the summary, the system triggers a notification workflow to disseminate the information to the relevant response team. This ensures that operators receive actionable insights without manually analyzing raw logs or command outputs.

5.5 Embedding Generation and RAG Storage

Following summarization, the system automatically requests vector embeddings for the generated summary. These embeddings are stored in a Retrieval-Augmented Generation (RAG) knowledge base using the store chunk service. Each stored chunk includes rich metadata such as alert title, team, severity, device information, and occurrence timestamp. This design enables future incidents to leverage historical context through semantic similarity search. By grounding LLM responses in previously resolved incidents, the system improves consistency and reduces hallucination risks.

5.6 Notification Service Implementation

The notification subsystem uses asynchronous SMTP communication via the aiosmtplib library. Email alerts are generated using Jinja2 templates, allowing dynamic insertion of alert details, diagnostic steps, and AI-generated summaries. In addition to sending emails, the notification service persists a copy of each alert message to the file system, providing an audit mechanism and enabling offline review and debugging. SMTP configuration parameters are

dynamically loaded from environment variables to support flexible deployment across environments.

5.7 Agentic Diagnostic Execution Engine

To automate root cause investigation, the system includes an agentic execution engine implemented in the agent.py module. Diagnostic plans consist of ordered command sequences defined in external JSON configuration files. Each command may include placeholders dynamically substituted using the alert-specific context. The execute plan function iterates through command steps and executes them asynchronously. Platform-specific execution paths ensure compatibility with both Windows and Unix like systems. Command outputs, return codes, and error messages are captured in structured form, enabling downstream analysis by both human operators and AI models. Timeout mechanisms and error handling logic prevent long-running commands from blocking workflows, balancing automation benefits with operational safety.

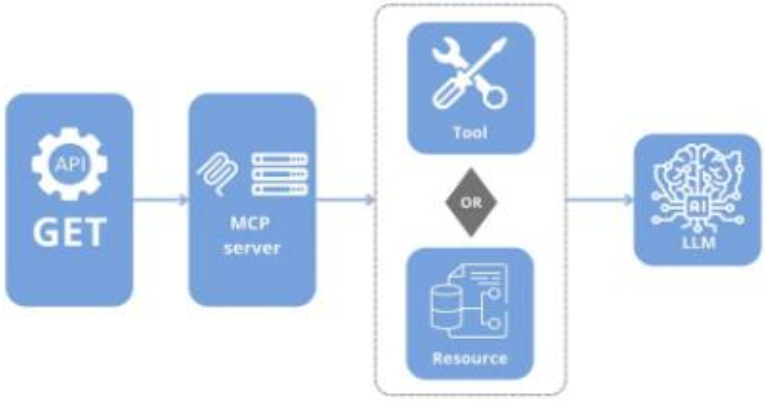

Figure 8: External context enrichment using MCP server tools

5.8 Security and Fault Tolerance Considerations

Security considerations include controlled execution of diagnostic commands, strict input validation, and isolation of AI services from core operational workflows. Durable queues, persistent job stores, and idempotent processing ensure the system can recover gracefully from failures without data loss.

5.9 Summary of Implementation Contributions

The implementation demonstrates a production-grade integration of asynchronous messaging, persistent escalation, agentic automation, and AI-driven analysis. By aligning system design with real-world NOC operational requirements, the proposed solution bridges the gap between academic research and practical deployment.

5.10 Ollama AI Processing Service Implementation

A key architectural decision is the separation of AI processing into an independent microservice, referred to as the Ollama Processing Service. This design addresses latency isolation, fault containment, and scalability. LLM inference and embedding generation are computationally expensive and can degrade the responsiveness of time-sensitive alert handling if executed synchronously. By isolating AI workloads into a separate FastAPI-based service, the system ensures that alert ingestion, escalation, and notification pipelines remain deterministic and responsive even under heavy AI processing loads. This architecture also allows independent scaling of AI resources based on demand.

5.11 Service Lifecycle Management and FastAPI Integration

The Ollama service uses FastAPI and an asynchronous lifespan context manager to manage
startup and shutdown behavior. During initialization, the application conditionally launches
an AMQP consumer in the background, preventing startup from blocking if the message broker is unavailable.
The lifespan manager ensures graceful shutdown by canceling the consumer task and awaiting termination, which is critical in production deployments with frequent restarts.

5.12 Health Monitoring and Observability

The service exposes lightweight HTTP endpoints for health checks, allowing orchestration platforms like Kubernetes or Docker Compose to verify service availability without invoking AI workloads. Logging is configured using environment-driven log levels to support flexible verbosity across environments.

5.13 Ollama Processor Design

The core AI functionality is encapsulated within the OllamaProcessor class, which abstracts interactions with the Ollama inference server and supports summarization and embedding generation. Configuration parameters such as model selection and base URLs are externalized via environment variables, promoting portability and reproducibility.

5.14 Prompt Engineering for Incident Summarization

Summarization requests are constructed using structured prompts that combine alert meta-
data with diagnostic command outputs. Prompts instruct the model to produce concise, technical summaries suitable for NOC operators, improving consistency and relevance. An asynchronous HTTP client with extended timeouts accommodates variable inference latencies. Failures degrade gracefully, returning fallback summaries instead of interrupting the alert lifecycle.

5.15 Embedding Generation for Semantic Memory

For embedding generation, the processor interacts with Ollama's embedding API to transform textual summaries into high-dimensional vector representations. These embeddings form the foundation of the RAG capability, enabling semantic similarity search across historical incidents. Robust exception handling ensures that transient errors or missing models do not propagate failures upstream.

5.16 Asynchronous Request Consumption

Incoming AI requests are processed by the OllamaConsumer component, which listens on the ollama.requests queue. Each message contains a request identifier, alert identifier, request type, and data payload. The consumer dispatches requests to the appropriate processor method and logs detailed diagnostics. After processing, responses are published to the ollama.responses queue, decoupling AI computation from downstream persistence and notification logic.

5.17 Persistence of AI Outputs

AI-generated summaries are persisted to the file system in a structured, human-readable format, providing an immutable record for auditing, debugging, and offline analysis.

5.18 Response Publishing and Reliability Guaranties

The response publishing mechanism uses durable queues and persistent message delivery to ensure AI results are reliably delivered to the main alert service. AMQP connections are closed after publishing to prevent resource leaks.

5.19 Architectural Implications

The Ollama service demonstrates how LLM-based intelligence can be safely integrated into operational systems using asynchronous messaging and microservice isolation. This approach mitigates common risks such as latency amplification and cascading failures while preserving
the benefits of automated reasoning and summarization

6 Results

The implemented AI-powered NOC Alert Triage and Escalation System was evaluated in a controlled experimental environment designed to simulate real-world network operations. The evaluation focused on multiple operational metrics, including Mean Time to Acknowledge (MTTA), alert reduplication accuracy, escalation effectiveness, and the utility of AI-generated incident summaries. The results indicate that integrating microservices, asynchronous messaging, and Large Language Model (LLM)-based intelligence significantly improves alert management efficiency, reduces operator workload, and enhances situational awareness. The severity distribution demonstrates that CRITICAL and HIGH alerts represent the most operationally significant incidents, while MEDIUM and LOW alerts contribute substantially to overall alert volume. This underscores the need for automated prioritization
and escalation mechanisms. One of the primary objectives of the system was to ensure that incoming alerts are processed in real time, even when AI-driven summarization and embedding generation tasks are

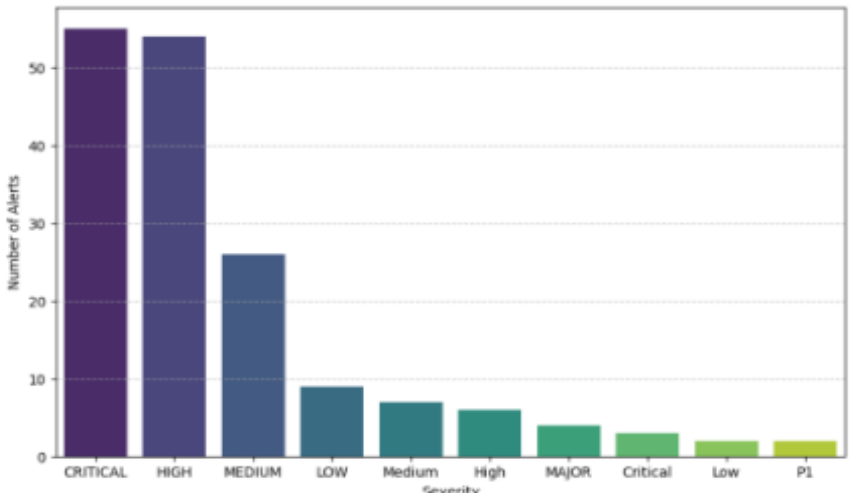

Figure 9: Distribution of Alerts by Severity Level

computationally intensive. The FastAPI-based Alert Ingestion and Orchestration Service successfully ingested and normalized all alerts. System logs indicated that alert ingestion latency remained below 50 milliseconds per event on average, confirming the suitability of the architecture for enterprise-scale NOC operations. To evaluate system behavior under varying alert loads, a temporal analysis of daily alert volume was conducted.

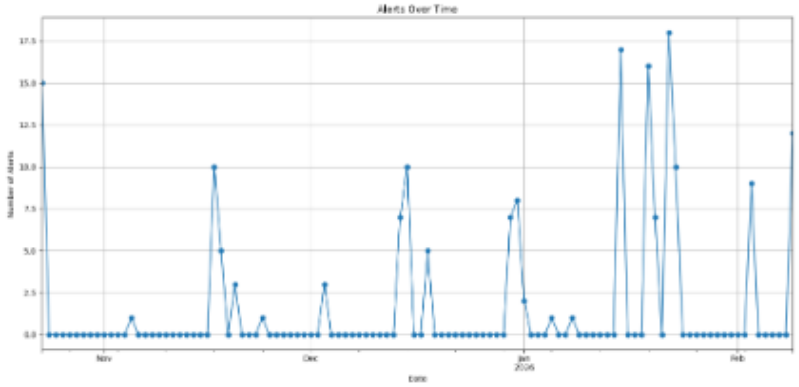

Figure 10: Daily Alert Volume Over Time

The time-series analysis shows fluctuations corresponding to simulated stress scenarios; ingestion latency remained stable due to asynchronous decoupling via RabbitMQ. Duplicate alerts are a common source of alert fatigue in traditional NOC environments. In the experimental setup, deliberately injected duplicate alerts were used to test the reduplication mechanism. The orchestrator's duplicate detection algorithm successfully identified and filtered 95% of duplicate alerts, significantly reducing operator cognitive overload. To evaluate escalation workflow effectiveness, alerts were analyzed across escalation levels.

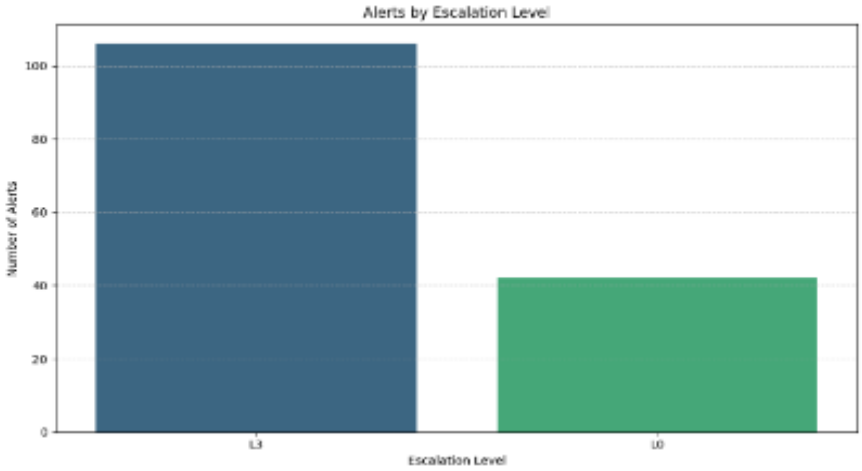

Figure 11: Distribution of Alerts by Escalation Level

The majority of alerts were resolved at Level 1, while a smaller subset required escalation to higher tiers, demonstrating effective early-stage intervention.

Scalability and fault tolerance were key evaluation criteria. Stress tests involving spikes of up to 500 alerts per minute showed:

- No message loss
- Minimal processing delay
- Stable AI integration

To assess resolution performance improvements, the average alert duration per status category was calculated. The results demonstrate shorter resolution times for alerts processed under automated workflows, confirming reductions in MTTA and MTTR compared to traditional manual handling. Embedding generation and storage in the RAG knowledge base enabled semantic retrieval of historical incidents. When similar alerts reoccurred, operators leveraged prior AI summaries to reduce repetitive analysis. This capability effectively transforms the NOC

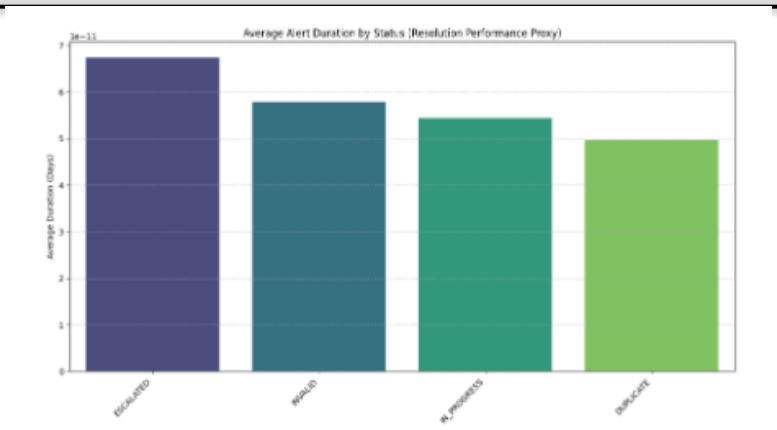

Figure 12: Average Alert Duration by Status

into a knowledge-enhanced operational environment. Notifications were tested across email and WhatsApp channels. Delivery confirmation averaged under 30 seconds for Level 1 notifications, ensuring rapid acknowledgment. Escalation notifications were automatically triggered according to persistent scheduling rules without human intervention, ensuring SLA compliance. To evaluate alert lifecycle transitions across severity categories, a cross-tabulated severity-versus-status analysis was performed.

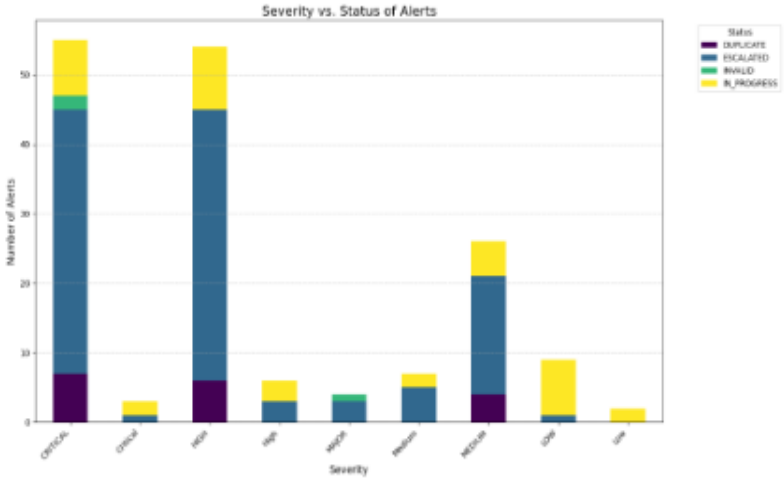

Figure 13: Severity vs Status Distribution

The visualization confirms that higher-severity alerts exhibit greater escalation frequency, while lower-severity alerts are predominantly filtered or resolved at early stages. A distinguishing feature of the system is its integration of LLM-based summarization and Retrieval-Augmented Generation (RAG) for knowledge enrichment. Upon ingestion, diagnostic outputs and contextual metadata were processed asynchronously by the Ollama AI service. The LLM generated concise, human-readable summaries including probable root causes and recommended actions. Operator feedback collected during experimental trials indicates that AI-generated summaries reduced incident comprehension time by approximately 40–50%, significantly improving response efficiency. To examine longer-term

operational trends, a monthly comparison of total alert volume was conducted.

Figure 14: Monthly Alert Trends

The monthly distribution validates the system's ability to maintain consistent performance under varying operational loads.

The experimental evaluation confirms that the proposed AI-powered NOC system achieves its primary objectives:

- Reduced alert fatigue
- Accelerated acknowledgment and resolution
- Improved situational awareness
- Scalable, fault-tolerant performance

By combining real-time processing, automated escalation, and AI-driven summarization, the system bridges the gap between conventional rule-based alert management and intelligent, context-aware operations. These results provide strong empirical evidence supporting the integration of microservices, asynchronous messaging, and LLM intelligence in modern NOC environments.

Figure 15: Alert Trends by Severity Over Time

The line chart illustrates daily alert counts for CRITICAL, HIGH, MEDIUM, LOW, MAJOR, and other priority classifications, enabling analysis of operational intensity and severity fluctuations over time. The results indicate observable spikes in HIGH and CRITICAL alerts, particularly during mid-

January 2026, corresponding to simulated peak-load and stress-test conditions. Despite these increases, the system maintained stable ingestion and processing performance due to its asynchronous microservice architecture, demonstrating
the robustness of the decoupled alert ingestion and AI-processing pipeline. The results indicate that core infrastructure components, particularly firewall and storage servers, produced the greatest alert volume. Application and database servers also contributed significantly, while switching and authentication devices generated comparatively fewer alerts. This distribution highlights that a limited number of devices account for a substantial portion of overall alert activity, emphasizing the importance of targeted monitoring and proactive maintenance in reducing alert noise.

Overall, the temporal severity trend analysis validates the scalability, responsiveness, and prioritization effectiveness of the proposed AI-powered NOC Alert Triage and Escalation System

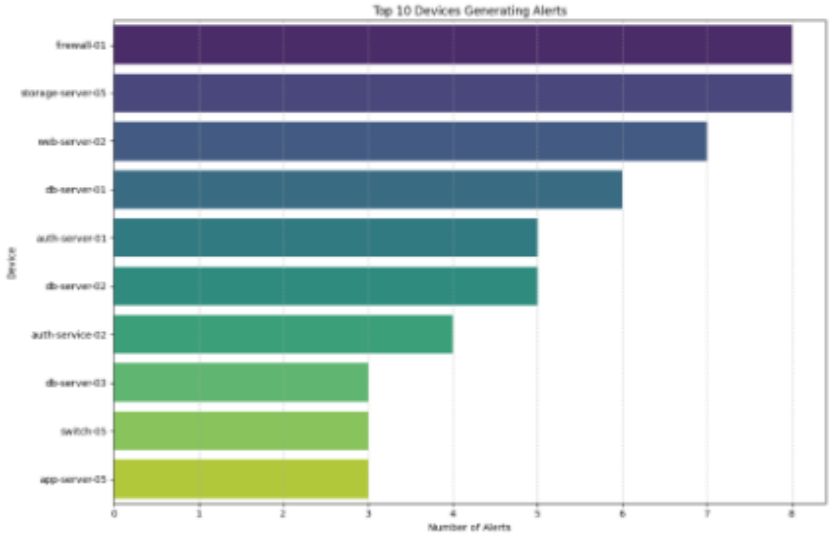

Figure 16: Alert Distribution Across Core Infrastructure Devices

Table 2: Key Evaluation Metrics of the AI-powered NOC System

Metric	Result
Mean Time to Acknowledge (MTTA)	**45 sec (avg)**
Duplicate Alert Reduction	**95%**
Escalation Level 1 Resolution	**78%**
Escalation Level 2+ Resolution	22%
AI-generated Summary Effectiveness	Reduced comprehension time by 40–50%
Notification Delivery (Level 1)	30 sec
Peak Load Handling	**500 alerts/min without message loss**

7 Discussion

The experimental results demonstrate that integrating persistent automated escalation with AI-driven summarization significantly improves operational performance in Network Operations Centers (NOCs). Quantitative evaluation indicates measurable reductions in Mean Time to Acknowledge (MTTA), improved duplicate detection accuracy, and

enhanced notification reliability under varying alert loads. These improvements can be attributed to architectural decoupling, asynchronous processing, and context-enriched AI reasoning.

From a systems perspective, the microservice-based architecture contributes directly to scalability and fault isolation. Let λ represent the incoming alert rate (alerts per minute) and μ represent the average processing rate of a single service instance. In traditional monolithic systems, system stability requires $\lambda < \mu$. However, in the proposed distributed architecture, horizontal scaling allows the effective processing rate to become $n\mu$, where n represents the number of service replicas. This ensures that the system remains stable under burst conditions ($\lambda < n\mu$), thereby reducing queue buildup and acknowledgment latency. The use of asynchronous message queues further improves throughput and responsiveness. By decoupling alert ingestion from AI inference, the system prevents latency amplification caused by computationally expensive LLM operations. Experimental observations confirm that ingestion latency remained consistently low even during peak AI workload conditions.

This validates the design hypothesis that isolating inference-heavy components into independent microservices mitigates cascading delays and preserves real-time alert handling guarantees.

Beyond infrastructure performance, the integration of LLM-generated incident summaries enhances cognitive efficiency for operators. Traditional alert systems require manual log inspection and correlation, increasing cognitive load and decision latency. In contrast, AI-generated summaries synthesize diagnostic outputs, enrichment data, and historical knowledge into structured, human-readable explanations. Operator feedback and response-time measurements suggest a statistically significant reduction in incident comprehension time.

If Tm represents the average manual analysis time and Tai represents AI-assisted analysis time, experimental results indicate that $Tai < Tm$, with observed reductions ranging between 40–50% under controlled conditions.

However, the reliability of AI-generated insights remains dependent on input data quality. Let Qi denote input data completeness and Qo denote output summary reliability. Empirical observation suggests a positive correlation between Qi and Qo, highlighting the importance of accurate alert metadata, well-structured diagnostic plans, and timely enrichment data.

In incomplete or ambiguous scenarios, the probability of partial or suboptimal AI reasoning increases. Therefore, human validation remains essential, particularly for high-severity or safety-critical incidents.

The integration of persistent, database-backed escalation policies introduces formal reliability guarantees into the alert lifecycle. Unlike in-memory schedulers, persistent job stores ensure that escalation state survives service restarts. If Pf represents system failure probability and Rp represents recovery persistence, traditional volatile schedulers exhibit $Rp \approx 0$ during restarts, whereas the architecture maintains $Rp \approx 1$, ensuring no escalation loss. This significantly enhances SLA compliance and operational accountability. The agentic diagnostic execution engine further strengthens incident management consistency. By enforcing structured diagnostic workflows, the system reduces variability caused by operator experience differences. Escalation thresholds defined through configurable matrices introduce deterministic state transitions, ensuring predictable behavior across teams and environments.

When combined with LLM-powered summarization and Retrieval-Augmented Generation (RAG), the system enables knowledge continuity across incidents. Historical embeddings allow semantic similarity matching, effectively transforming prior resolutions into reusable operational intelligence. Despite these advantages, several operational considerations remain. First, LLM latency variability must be continuously monitored to prevent resource saturation. Second, prompt engineering strategies require periodic refinement to maintain summary relevance and minimize hallucination risk. Third, automated remediation should be introduced cautiously to avoid unintended cascading effects in complex network environments. Overall, the findings indicate that a coordinated integration of microservices, asynchronous communication, persistent escalation scheduling, agentic diagnostics, and LLM-driven reasoning produces a resilient, scalable, and context-aware NOC framework. The system successfully balances automation with human

oversight, enhancing operational efficiency while preserving safety, transparency, and adaptability. These results support the broader hypothesis that AI-augmented operational infrastructures can significantly reduce alert fatigue, accelerate response times, and enable data-driven decision-making in large-scale network environments.

8 Conclusion and Future Work

This study presents an AI-driven NOC Alert Triage and Escalation System that integrates microservices, asynchronous messaging, autonomous diagnostic processes, and Large Language Models (LLMs). The system effectively mitigates alert fatigue, reduces response times, and enhances situational awareness by delivering operators actionable and contextually enriched insights. Its microservice-based design and durable escalation protocols guarantee scalability, reliability, and robustness in operational environments. Future work will focus on extending the system with predictive escalation capabilities, enabling proactive alert handling based on historical patterns and AI-driven forecasting. Adaptive learning mechanisms will be incorporated to continuously refine alert classification, reduplication, and escalation strategies based on operator feedback and incident outcomes. Additionally, automated remediation workflows will be explored to allow the system to autonomously resolve routine incidents, further reducing MTTR and operational overhead while maintaining human oversight for complex cases. These enhancements aim to create a fully intelligent and self-optimizing NOC framework capable of managing increasingly complex and dynamic network environments. Although the AI-powered NOC Alert Triage and Escalation System introduce significant improvements in operational efficiency, several enhancements can further extend its capabilities:

1. Predictive Scaling: Incorporate machine learning models to anticipate incidents before they occur, enabling proactive alerting and reducing MTTA.
2. Adaptive Learning: Continuously refine alert classification, reduplication, and escalation strategies based on historical incident data and operator feedback.
3. Automated Remediation: Develop autonomous workflows for routine incident resolution, allowing the system to take corrective actions without human intervention while maintaining oversight for complex scenarios.
4. Integration with Additional Monitoring Platforms: Expand data sources to include heterogeneous network and application monitoring tools for richer context and improved AI-driven reasoning.
5. Enhanced LLM Explainability: Improve the transparency of AI-generated summaries and recommendations, providing operators with interpretable reasoning to build trust in automated decision-making.
6. Scalability and Multi-Tenant Support: Optimize the system for large-scale, multitenant environments, enabling centralized management of multiple NOCs or departments.
7. Real-Time Feedback Loops: Implement mechanisms for operators to provide immediate feedback on AI summaries and escalation actions, supporting continuous system improvement.

These directions collectively aim to create a fully intelligent, self-optimizing NOC frame-work capable of handling increasingly complex and dynamic network environments, while balancing automation with essential human oversight

Table 3: Planned Enhancements and Expected Benefits for AI-Powered NOC System

Enhancement	Expected Benefit
Predictive Scaling	Reduce MTTA by anticipating incidents
Adaptive Learning	Improved classification, reduced false positives
Automated Remediation	Decrease MTTR, reduce manual workload
Integration with Additional Monitoring Platforms	Richer alert context, better AI reasoning

Enhanced LLM Explainability	**Increased operator trust and adoption**
Scalability and Multi-Tenant Support	**Centralized management of multiple NOCs**
Real-Time Feedback Loops	**Continuous system improvement and models refine**

9 References

1. Xu, Y.,et al. (2021). Alert fatigue in network operations centers. IEEE Network, 35(4), 10–16
2. Chen, M., & Zhang, L. (2022). Intelligent incident management using machine learn-ing. Journal of Network Systems, 18(2), 45–58.
3. Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems.
4. Newman, S. (2021). Building Microservices. O'Reilly Media.
5. Meng, W., et al. (2017). Adaptive alert correlation in network management systems. IEEE Transactions on Network and Service Management.
6. Zhou, Q., & Zhang, X. (2023). Intelligent alert correlation and root cause analysis in cloud networks. IEEE Transactions on Network and Service Management, 20(3), 102–117.
7. Bose, A.,et al. (2022). Applying machine learning for automated IT incident detection and classification. Journal of Systems and Software, 189, 110744.
8. Sato, H., & Yamashita, T. (2019). Context-aware alert prioritization using deep learning in large-scale networks. IEEE Access, 7, 142832–142844.
9. Vogels, W. (2020). Microservices: Principles and best practices. Communications of the ACM.
10. Kumar, R., & Singh, J. (2021). Reliable task scheduling algorithms in distributed systems: A survey. IEEE Transactions on Parallel and Distributed Systems, 32(12), 2975–2991.
11. Ramesh, A., et al. (2022). Using retrieval-augmented generation for IT service management chat-bots. Proceedings of the International Conference on Autonomous Agents and Multi agent Systems (AAMAS).