Research Trends International Journal of Technology and Innovation
An International Peer-Reviewed, Refereed & Open-Access Journal
ISSN : 3144-3273
Available Online at : https://rtijti.com/
doi : https://doi.org/10.5281/zenodo.23561801

AegisSOC: An Adaptive AI-Powered Security Operations Center with Unsupervised Anomaly Detection and Retrieval-Grounded Alert Explanation

Sneha Jadhav, Diksha Dhaigude, Prajakta Khandekar, Sakshi Jadhav, Prof.Swapnil Kale
Department of Computer Engineering, Navsahyadri Education Society's Group of Institutions, Pune, Maharashtra, India

ARTICLE INFO

Article History:
Received : 18 oct 2026
Accepted : 19 oct 2026
Published : 21 oct 2026

Publication Issue
Volume 1, Issue 4
October – December 2026

Page Number
07-14

ABSTRACT

Small and mid-sized organisations, colleges among them, collect security logs from firewalls, Windows desktops, Linux servers and network devices, but rarely have the staff or the budget to watch them. Rule-based SIEM correlation only reports attacks that someone has already written a rule for, and loose rules flood analysts with alerts that are mostly harmless. This paper presents the design and Stage-I implementation of AegisSOC, an AI-assisted Security Operations Center built from open-source components. Logs from four sources are collected through Syslog, Winlogbeat and a Python forwarder, posted to a FastAPI backend, and normalised by a Pandas parser into one JSON schema stored in MongoDB. Features such as failed logins, distinct destination ports, outbound bytes and new process counts are computed over five-minute windows per host, user and source IP. Two unsupervised detectors trained only on normal activity, an Isolation Forest and a dense Autoencoder, score each window, and an alert is raised when either score passes a percentile threshold, with higher severity when both agree. Each alert is then explained by a Large Language Model whose prompt is restricted to MITRE ATT&CK, CVE and playbook passages retrieved from a vector store. Four backend agents classify and group alerts, check process indicators, suggest playbook steps and write PDF incident reports, and no response action runs without analyst approval. Design, literature review and about 40 percent of development are complete. Detection accuracy has not yet been measured; the evaluation protocol on CICIDS-2017, UNSW-NB15 and college lab logs is described in full so that Stage-II results can be checked against it.

Index Terms - Security Operations Center, Anomaly Detection, Isolation Forest, Autoencoder, Retrieval-Augmented Generation, Large Language Models, Log Normalisation, Alert Fatigue

I. INTRODUCTION

1.1 Background and Motivation

Every device on a network keeps a record of what happens on it. A firewall writes a line for each connection it allows or drops, a Windows machine records each logon and each new process in its Security event log, a Linux server writes authentication and service messages to syslog, and routers and switches log interface and configuration changes. Together these logs are the main evidence an organisation has when it needs to know whether it is under attack.

The difficulty is volume and variety. A college lab of sixty machines with one firewall and a few servers produces several hundred thousand log lines a day, and a failed logon looks completely different in a Windows event (Event ID 4625), an sshd line in auth.log and a firewall deny record. Before any analysis, the records have to be collected in one place and brought to one structure. In a typical Security Operations Center (SOC) a Security Information and Event Management (SIEM) tool does this, runs correlation rules written by analysts, and raises alerts that analysts then triage.

Our own department is a fair example of the problem. The lab network has a firewall, Windows desktops and Linux servers, each keeping its own logs, and nobody reads them unless something has already gone wrong. Commercial SIEM products are priced per gigabyte ingested or per endpoint and still need trained staff to write rules [1], [5]. Colleges and small businesses have the logs but not the money or the people to watch them.

1.2 Problem Statement

We set out to build a system that collects security logs from four kinds of source, detects suspicious activity without depending on hand-written rules, and explains each alert to the person who has to act on it, on hardware a college lab already owns. Three properties were fixed at the start. Detection must work without labelled attack

Copyright © 2026 The Author(s) : This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

data, since a small organisation has none. The language model may explain an alert but may never decide whether an event is malicious, and every explanation must point to the reference passage it was built from. And no response action, such as blocking an address, may run without an analyst approving it.

1.3 Contributions

The paper makes four contributions. First, a common log schema and field mapping that brings Windows Security events, Linux syslog, firewall records and network device logs into one MongoDB collection while keeping the original line for evidence. Second, a detection engine that runs an Isolation Forest and an Autoencoder side by side on windowed features, with thresholds set on held-out normal data and severity raised when both detectors agree. Third, an explanation step in which a Large Language Model (LLM) is restricted to MITRE ATT&CK, CVE and playbook passages retrieved for the alert, with the passages stored next to the explanation. Fourth, an agent layer that classifies, groups and documents alerts but keeps every response action behind analyst approval. We also set out the evaluation protocol before any results exist, so the Stage-II numbers can be held to it.

1.4 Organisation of the Paper

Section II reviews related work and the research gap. Section III sets out the architecture. Section IV describes the methodology: normalisation, features, the two detectors, retrieval-grounded explanation and the agents. Section V covers the implementation, Section VI the current status and evaluation plan, Section VII the discussion, Section VIII the limitations, and Section IX the conclusion and future work.

II. RELATED WORK

2.1 Rule-Based Monitoring

Signature-based intrusion detection and rule-based SIEM correlation remain at the core of most SOCs. A rule such as "more than ten failed logons for one account in five minutes" is easy to read and cheap to run, so an analyst always knows why it fired. Its weakness is coverage. A rule written for SSH brute force misses a slow password-spraying attack spread across many accounts, and a rule with a low threshold fires on every user who mistypes a password. Sommer and Paxson [16] also warned that machine learning is harder to apply to intrusion detection than to many other fields, because normal traffic varies widely and each false alarm costs analyst time.

2.2 AI, Automation and Agents in the SOC

Khayat et al. [1] mapped machine learning techniques to SOC functions such as detection, triage and response, and showed where adoption is still low, but did not present a working system. Tilbury and Flowerday [5] argued that automation works best on repetitive tasks like enrichment and reporting, while judgement-heavy decisions stay with people; this supports our rule that response actions need approval. Baruwal Chhetri et al. [3] studied human-AI teaming as a way to reduce alert fatigue. Srinivas et al. [2] proposed a taxonomy of eight SOC functions with a capability-maturity framework for LLMs and agents, and Stefanov et al. [4] linked agentic architectures to SOC indicators such as mean time to detect and respond. Tseng et al. [6] used LLMs to automate threat intelligence workflows, and Sood and Zeadally [8] explored generative AI for threat hunting.

2.3 Machine Learning for Intrusion Detection

Anis et al. [7] surveyed deep learning network intrusion detection, Hozouri et al. [9] reviewed ML and DL intrusion detection architectures together with problems such as dataset bias and adversarial input, and Mahendar and Shivakanth [10] surveyed ML-based intrusion detection for cloud environments. Isolation Forest [11] isolates rare records by random partitioning and needs no labels. Autoencoders trained on normal data use reconstruction error as an anomaly score, one of the main families in the survey by Chandola et al. [15]. Retrieval-Augmented Generation (RAG) [12] conditions a language model on documents retrieved from an external store, which for security can be ATT&CK techniques [17], CVE records and response playbooks.

2.4 Gap Addressed

Table 1 summarises the ten SOC-focused papers. Most are surveys or conceptual designs, and the implemented ones handle a single kind of data: network flows [7], threat intelligence reports [6] or cloud logs [10]. None combines multi-source host and network log collection, unsupervised detection, retrieval-grounded explanation and report generation in one system that a small organisation can run. That combination is the gap this paper works on.

Table 1. Summary of reviewed SOC and intrusion detection literature

Work	Focus	Limitation for this problem
Khayat et al. [1]	AI/ML across SOC functions (review)	No working unified system
Srinivas et al. [2]	LLMs and agents for SOC automation	Survey only, no implementation
Baruwal Chhetri et al. [3]	Human-AI teaming, alert fatigue	Collaboration model, not detection
Stefanov et al. [4]	Agentic AI against SOC KPIs	Conceptual, not tested on logs
Tilbury and Flowerday [5]	Automation supporting analysts	No detection engine proposed
Tseng et al. [6]	LLMs for threat intelligence	CTI reports only, no live logs
Anis et al. [7]	DL network intrusion detection	Network traffic only
Sood and Zeadally [8]	Generative AI threat hunting	High compute, no reporting layer
Hozouri et al. [9]	ML/DL IDS architectures	Review, no explanation layer
Mahendar and Shivakanth [10]	ML IDS for cloud	On-premise logs not covered

III. SYSTEM ARCHITECTURE

3.1 Overall Structure

AegisSOC is organised as a layered pipeline, shown in Figure 1. Logs move from the data sources through a collection layer and a processing layer into the detection engine. Alerts produced there pass to four agents and finally to the presentation layer. A knowledge base of ATT&CK techniques, CVE records and playbooks feeds the explanation step, and MongoDB holds raw logs, normalised records, alerts and results.

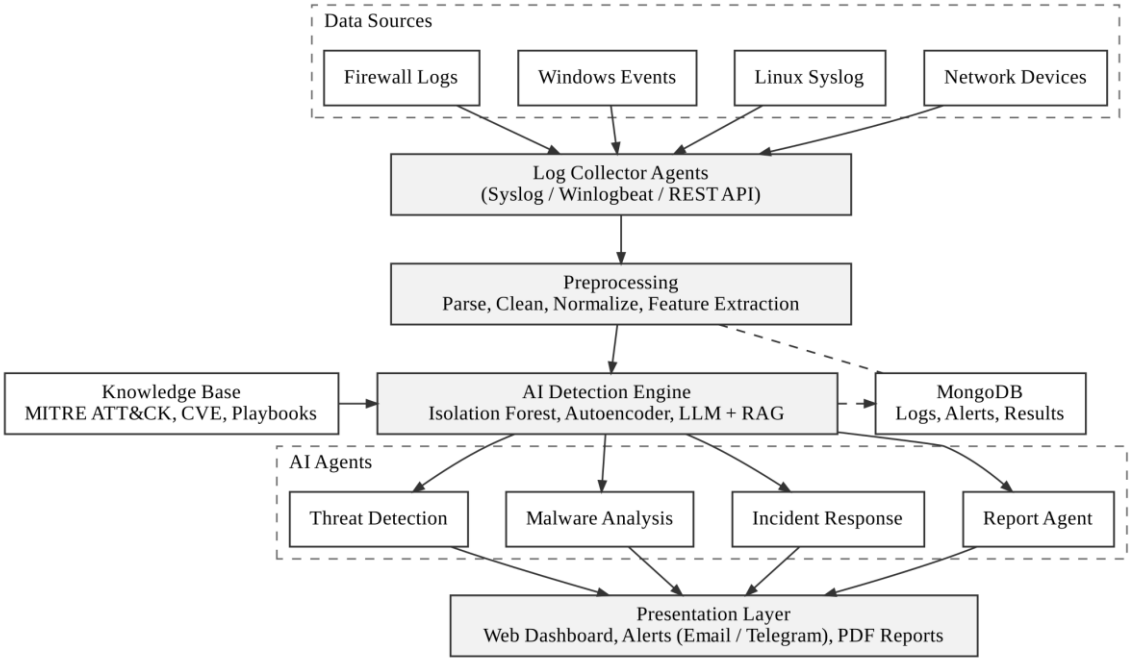


Figure 1. System architecture of AegisSOC. Four log sources feed the collector agents and the preprocessing layer; the detection engine combines Isolation Forest, Autoencoder and LLM with RAG, draws on the knowledge base and writes to MongoDB; four agents act on its alerts and the presentation layer delivers the dashboard, notifications and PDF reports.

3.2 Layers

Data sources. The first version monitors firewall logs, Windows event logs, Linux syslog and router or switch logs. On Windows the Security channel is the main source, with Event IDs 4624 (successful logon), 4625 (failed logon) and 4688 (process creation). On Linux, auth.log and syslog carry SSH, sudo and service messages.

Collection. Linux hosts and network devices send syslog to a central receiver, Windows hosts run Winlogbeat, and a Python forwarder batches records and posts them to the ingest endpoint over REST with a collector token.

Processing. A Pandas parser converts each entry to the common schema, drops noise, normalises timestamp, IP address, user and event type, and computes windowed features.

Intelligence. The detection engine runs both anomaly models and the LLM with RAG module.

Agents and presentation. Four agents run as backend services on a shared alert queue. A React.js dashboard with Chart.js shows live charts and the alert list, alerts are also pushed by e-mail or Telegram, and incident reports download as PDF.

3.3 Processing Workflow

Figure 2 traces one window of log data through the system. Windows that stay below both thresholds are stored as normal and go no further. Windows that cross a threshold become alerts and pass through explanation, agent handling and notification in that order.

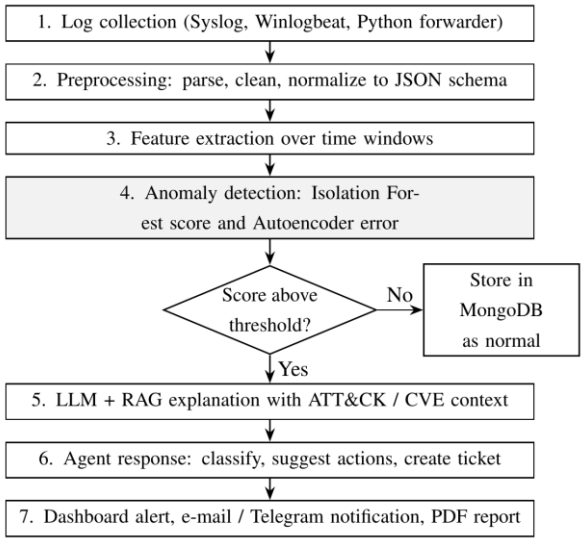


Figure 2. Processing workflow for one window of log data, including the path for windows that stay below threshold.

IV. METHODOLOGY

4.1 Log Normalisation

Every source is mapped to one record with the fields timestamp, source, host, src_ip, dst_ip, dst_port, user, event_type, event_id, process, severity and raw. Fields that a source does not provide are stored as null rather than guessed, and the original line is kept in raw so the analyst can always see the evidence. Table 2 shows how the key fields are obtained.

Table 2. Mapping of source fields to the common schema

Schema field	Windows event log	Linux syslog	Firewall / device
timestamp	TimeCreated	syslog header time	record time
user	TargetUserName	user in sshd or sudo line	usually empty
src_ip	IpAddress	"from ip" in sshd line	source address
event_type	from EventID (4624, 4625, 4688)	program and message pattern	action (allow, drop, deny)
dst_port	not present	port in sshd line	destination port

4.2 Feature Extraction

Single log lines say little; attacks show up as patterns over time. Features are therefore computed over sliding windows grouped by host, user or source IP, with five minutes as the starting window length. Table 3 lists the planned features. Numeric features are standardised with statistics fitted on training data only, and long-tailed counts such as bytes_out are transformed with $\log(1 + x)$ first so that a few very large transfers do not dominate the scale.

Table 3. Planned window features

Feature	Grouped by	Threat it helps reveal
failed_logins	user, src_ip	brute force, password spraying
distinct_users_per_ip	src_ip	spraying across accounts
distinct_dst_ports	src_ip	port scanning
bytes_out	host	exfiltration, abnormal volume
connections_denied	src_ip	scanning, blocked attempts
new_process_count	host	malware execution, unusual tools
off_hours_flag	user	logins outside working hours
privilege_events	user	sudo or admin logons by unusual users

4.3 Isolation Forest

Isolation Forest [11] builds t random trees on subsamples of size ψ. For a window x, let $h(x)$ be the number of splits needed to isolate it in one tree and $E[h(x)]$ its mean over all trees. The anomaly score is

$$s(x, \psi) = 2^{-E[h(x)] / c(\psi)} \quad (1)$$

where the normalising term is the average path length of an unsuccessful search in a binary search tree,

$$c(\psi) = 2H(\psi - 1) - 2(\psi - 1) / \psi \quad (2)$$

and $H(i) \approx \ln(i) + 0.5772$ is the harmonic number. Scores near 1 indicate anomalies; scores well below 0.5 indicate normal windows. We use the Scikit-learn implementation [18] with $t = 100$ and $\psi = 256$ as starting values, as suggested in the original paper. In code the score is taken as the negative of score_samples, so that larger means more anomalous.

4.4 Autoencoder

The Autoencoder, built in TensorFlow, is trained only on normal windows. An input of d features passes through dense ReLU layers of 32 and 16 units to an 8-unit bottleneck and is rebuilt symmetrically through 16 and 32 units to a linear output of size d. It is trained with the Adam optimiser on mean squared error for 30 epochs with batch size 256. For an input x and its reconstruction $\hat{x}$, the reconstruction error is

$$e(x) = (1/d) \sum_{j=1..d} (x_j - \hat{x}_j)^2 \quad (3)$$

A window the network has learned to rebuild well looks like normal activity; a window it rebuilds poorly does not. The Autoencoder can notice combinations of features that are each normal on their own but unusual together, which a tree ensemble splitting on one feature at a time may miss.

4.5 Thresholds and Combination Rule

Normal windows are split 80/20. The scaler and both models are fitted on the 80 percent part, and the thresholds τ_{IF} and τ_{AE} are set at the 99th percentile of each score on the held-out 20 percent, never on test data. An alert is raised when

$$s(x) > \tau_{IF} \text{ or } e(x) > \tau_{AE} \quad (4)$$

with severity set to high when both conditions hold and medium when only one does. Both raw scores and the three most unusual features are stored with the alert, so the effect of the OR rule against an AND rule can be measured in Stage-II instead of argued in advance.

4.6 Retrieval-Grounded Explanation

When an alert is created, the RAG module builds a short query from the event type, the unusual features, the host role and any CVE identifiers in the log. The query is embedded and matched against a vector store of ATT&CK technique descriptions, CVE records and response playbooks, and the top four passages go into the prompt together with the alert (Figure 3). The prompt asks for a plain-language explanation, the likely ATT&CK technique and next steps, and instructs the model to answer only from the supplied passages and to say so when they do not cover the alert. The retrieved passages are stored with the explanation so the analyst can check them. Only alert summaries are sent to the LLM API, never raw logs, and user names can be masked before the prompt is built.

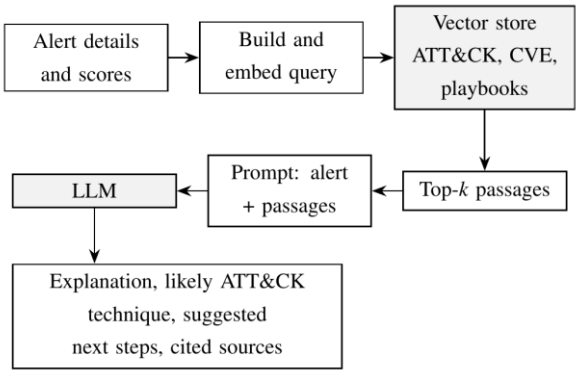


Figure 3. Alert explanation using an LLM with Retrieval-Augmented Generation.

4.7 Agents

The Threat Detection Agent assigns each alert a category such as brute force, scanning, privilege misuse or abnormal traffic, maps it to an ATT&CK technique where possible (T1110 Brute Force for repeated failed logons, for example), and groups alerts from the same source IP or user into one incident to cut the alert count. The Malware Analysis Agent checks process names, command lines and file hashes from Event ID 4688 against a local indicator list and flags unknown executables started from temporary or user folders. The Incident Response Agent picks steps from the playbooks, such as a password reset or a firewall block, and opens a ticket; every step is a suggestion until an analyst approves it on the dashboard. The Report Agent collects the alert, explanation, timeline, decision and actions taken into a PDF report when the incident is closed.

V. IMPLEMENTATION

5.1 Environment and Stack

Development targets a single machine with an Intel i5 or Ryzen 5 processor, 16 GB RAM (8 GB minimum) and a 256 GB SSD; an NVIDIA GPU is optional and only shortens Autoencoder training. The software runs on Ubuntu 22.04 and Windows 10/11 test hosts. The backend is Python 3.12 with FastAPI, the detectors use Scikit-learn and TensorFlow, storage is MongoDB, and the dashboard is React.js with Chart.js. Every service is packaged with Docker Compose so that the whole system can be brought up on one lab machine.

5.2 Backend API

The FastAPI backend exposes the endpoints in Table 4. They are the only interfaces between the collectors, the dashboard and the backend.

Table 4. Planned REST API endpoints

Method	Endpoint	Purpose
POST	/api/logs/ingest	Receive a batch of records from a collector
GET	/api/alerts	List alerts filtered by severity, source and time
GET	/api/alerts/{id}	Alert details, scores, explanation and sources
POST	/api/alerts/{id}/decision	Analyst approves or rejects a suggested action
GET	/api/reports/{id}	Download the PDF incident report
GET	/api/stats	Counts and time series for dashboard charts

5.3 Security and Privacy Controls

AegisSOC holds sensitive data itself, so it is protected like any other server. The dashboard and API require login, only analyst accounts can approve actions, and collectors authenticate to the ingest endpoint with a token. The agents have no permission to change firewall rules in this version. Normalised records keep only the fields needed for detection, raw logs have a fixed retention period, saved models are loaded only from the project's own storage, and log collection is limited to lab machines with department permission.

5.4 Prototype of the Alert Flow

A working prototype of the alert flow has been built. When the threat assessment engine flags a login as unusual, for instance from an unrecognised location or after repeated authentication errors, the user is asked for multi-factor verification and the diagnostic details of the alert are shown. Figure 4 shows the verification screen with source IP, operating system, browser signature, risk classification and classifier confidence; Figure 5 shows the explanation of why the gate fired and the code entry field. The 74 percent risk value on these screens is produced by the prototype and is not a measured detection result.

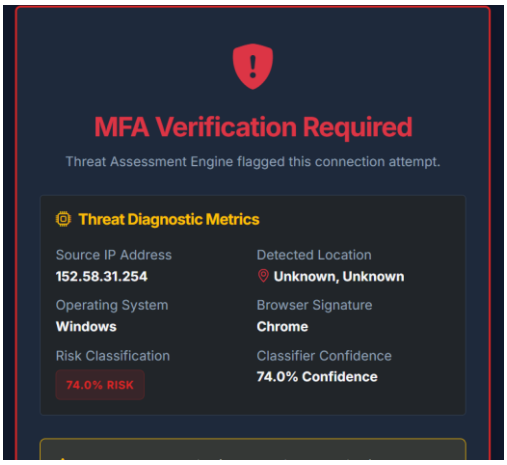


Figure 4. MFA verification screen with threat diagnostic metrics from the prototype.

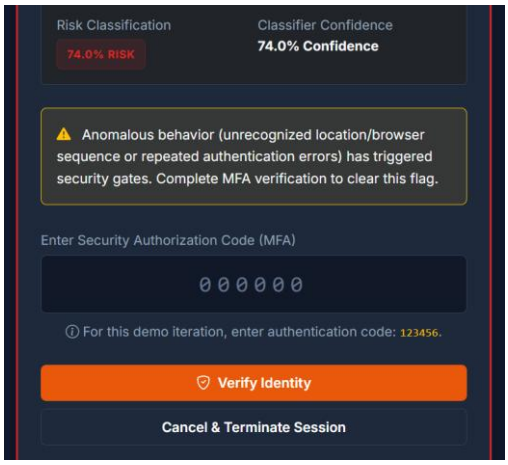


Figure 5. Anomaly explanation and MFA code entry in the prototype.

VI. CURRENT STATUS AND EVALUATION PLAN

6.1 Stage-I Progress

Table 5 gives the status of each phase at the end of Stage-I. Requirement analysis, the literature review and system design are finished, development is about 40 percent done, and overall Stage-I progress is about 60 percent. No detection accuracy is reported here because the models have not yet been trained or tested.

Table 5. Project Stage-I progress

Phase	Status	Details
Requirement analysis	100%	Existing SOC tools studied, scope fixed
Literature survey	100%	Ten recent papers reviewed
System design	100%	Architecture, data flow, modules final
Development	40%	Collector and preprocessing in progress, dashboard UI started, alert-flow prototype built
Testing and deployment	0%	Planned for Stage-II

6.2 Datasets

CICIDS-2017 [13] was recorded over five working days and contains benign traffic with brute-force FTP and SSH, DoS, DDoS, Heartbleed, web attacks, infiltration, botnet and port scans, described by about 80 CICFlowMeter features. UNSW-NB15 [14] contains nine attack families (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms) with 49 features. Both are network-flow datasets and contain no Windows or Linux host logs, so AegisSOC will also be tested on one to two weeks of normal logs from the college lab, followed by controlled attacks (SSH brute force, port scan, unusual process launch) run on an isolated lab segment with permission.

6.3 Metrics

Attack windows are the positive class. With TP, TN, FP and FN counted against the labels,

$$\text{Precision} = TP / (TP + FP) \quad (5)$$

$$\text{Recall} = TP / (TP + FN) \quad (6)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall}) \quad (7)$$

$$FPR = FP / (FP + TN) \quad (8)$$

For a SOC the false-positive rate matters most. At 100,000 normal windows a day, even an FPR of 1 percent produces 1,000 false alerts, which is exactly the alert fatigue the system is meant to remove. Accuracy alone is misleading because normal windows far outnumber attacks in both datasets, so precision, recall, F1 and FPR will be reported together with the confusion matrix and ROC curve for Isolation Forest, the Autoencoder and the combined rule on each dataset separately. Beyond detection, we will measure delay from log arrival to dashboard alert, alerts per day before and after grouping, and analyst ratings of a sample of LLM explanations for correctness and usefulness.

6.4 Protocol

Duplicates and infinite values are removed from the benchmark files, and the file list and label mapping are recorded. Both models are trained on benign records only, thresholds are set on a separate benign set, and testing uses a held-out mix of benign and attack records. Random seeds, library versions and model parameters are recorded for every run. Results on the public datasets and on lab logs are reported separately and never averaged together.

VII. DISCUSSION

7.1 Comparison with Existing Approaches

Table 6 sets AegisSOC against a rule-based SIEM and a commercial SOC platform on the points that matter to a small organisation. The table compares design properties, not measured performance.

Table 6. Comparison of AegisSOC with existing approaches

Aspect	Rule-based SIEM	Commercial SOC	AegisSOC
New attacks	Only if a rule exists	Rules plus vendor analytics	Unsupervised anomaly detection
Explanation	Rule name only	Varies by product	Grounded in ATT&CK and CVE passages
Triage and reports	Manual	Partly automated (SOAR)	Agents for grouping and PDF reports
Response	Manual	Automated playbooks	Suggested, analyst approves
Cost	Low licence, high staff time	High licence and staff cost	Open-source stack plus LLM API calls
Target user	Teams that write rules	Large enterprises	Colleges, small organisations

7.2 Anomaly Is Not Attack

Both detectors flag unusual behaviour, not malicious behaviour. A software update pushed to every lab machine at once, a night-time backup or a student working late will all look unusual. This is the reason for

the explanation step and for keeping the analyst in the loop, and it is also why the alert count per day will be tracked from the first week of deployment. Grouping by the Threat Detection Agent, threshold tuning on lab data and the analyst decision recorded with each alert are the planned controls.

7.3 Keeping the Language Model in Its Place

An LLM can write a confident explanation that is wrong. Retrieval reduces this risk without removing it. We therefore separate the two roles sharply: the detectors and the analyst decide whether something is an attack, and the model only describes an alert that already exists, from passages that are stored and shown alongside its text. An analyst who disagrees with an explanation can see what it was built from and reject it on that basis.

7.4 Concept Drift

Normal behaviour changes over a semester. Exam weeks, new software and new users all move the baseline. The models will need periodic retraining on recent normal data, and a sustained rise in the alert rate will be treated as a signal to review the baseline before retraining, so that an ongoing attack is not quietly absorbed into normal.

VIII. LIMITATIONS

8.1 No Measured Results Yet

The central limitation is that no detection metric exists yet. The design choices in Section IV, including the percentile thresholds, the five-minute window and the OR combination rule, are starting points that Stage-II must test.

8.2 Benchmark Mismatch

CICIDS-2017 and UNSW-NB15 are network-flow datasets recorded in controlled settings. Good scores on them will not show that the system works on Windows and Linux host logs, which is why lab logs form a separate test. The lab collection is itself small and short, and simulated attacks will be simpler than real ones.

8.3 External Dependency and Scale

Explanation quality depends on the knowledge base and on an external LLM API, which adds a cost per call and sends alert summaries outside the organisation. The first version handles four sources on one machine, and throughput beyond lab scale has not been designed for.

IX. CONCLUSION AND FUTURE WORK

9.1 Conclusion

This paper described the design and Stage-I implementation of AegisSOC, an AI-assisted SOC for organisations that cannot run a commercial one. Logs from firewalls, Windows hosts, Linux servers and network devices are normalised into one schema in MongoDB and summarised as windowed features. An Isolation Forest and an Autoencoder trained on normal activity score each window, with thresholds set on held-out normal data and severity raised when both agree. A language model explains each alert only from retrieved ATT&CK, CVE and playbook passages, four agents group, check, suggest and report, and every response action waits for an analyst. Design and literature work are complete, development is about 40 percent done, and an alert and MFA verification prototype is running. We make no accuracy claim at this stage.

9.2 Future Work

Stage-II will complete and integrate the detection engine, agents and dashboard, then train and test on CICIDS-2017, UNSW-NB15 and lab logs under the protocol in Section VI. After that, the Incident Response Agent can be allowed to block an IP or isolate a host on one-click approval, with an audit log and an undo action. Further work covers cloud (AWS, Azure), web server and IoT log sources; showing per-feature Isolation Forest contributions and Autoencoder reconstruction errors next to each explanation; using analyst decisions to tune thresholds per host and, once enough labelled alerts exist, to train a supervised classifier; a mobile app for alerts and approvals; and a locally hosted open-weight model in place of the external API, which would remove per-call cost and keep alert data inside the organisation.

REFERENCES

[1] M. Khayat, E. Barka, M. A. Serhani, F. Sallabi, K. Shuaib and H. M. Khater, "Empowering Security Operation Center With Artificial Intelligence and Machine Learning: A Systematic Literature Review," IEEE Access, vol. 13, pp. 19162-19197, 2025, doi: 10.1109/ACCESS.2025.3532951.

[2] S. Srinivas, B. Kirk, J. Zendejas, M. Espino, M. Boskovich, A. Bari, K. Dajani and N. Alzahrani, "AI-Augmented SOC: A Survey of LLMs and Agents for Security Automation," Journal of Cybersecurity and Privacy, vol. 5, no. 4, Art. no. 95, 2025, doi: 10.3390/jcp5040095.

[3] M. Baruwal Chhetri, S. Tariq, R. Singh, F. Jalalvand, C. Paris and S. Nepal, "Towards Human-AI Teaming to Mitigate Alert Fatigue in Security Operations Centres," ACM Transactions on Internet Technology, vol. 24, no. 3, pp. 1-22, 2024, doi: 10.1145/3670009.

[4] M. Stefanov, K. Stefanov, L. Niure Kandel, S. Crouse and B. Jekov, "Autonomous Agentic AI Architectures for Optimizing Security Operations Centers (SOC) KPIs: Methodology, Impact on Detection, Response, and Recovery," Land Forces Academy Review, no. 3, pp. 479-493, 2025, doi: 10.2478/raft-2025-0046.

[5] J. Tilbury and S. Flowerday, "Humans and Automation: Augmenting Security Operation Centers," Journal of Cybersecurity and Privacy, vol. 4, no. 3, pp. 388-409, 2024, doi: 10.3390/jcp4030020.

[6] P. Tseng, Z. Yeh, X. Dai and P. Liu, "Using LLMs to Automate Threat Intelligence Analysis Workflows in Security Operation Centers," arXiv preprint arXiv:2407.13093, 2024.

[7] F. M. Anis, M. Alabdullatif, S. Aljbli and M. Hammoudeh, "A Survey on the Applications of Deep Learning in Network Intrusion Detection Systems to Enhance Network Security," IEEE Access, vol. 13, pp. 185357-185373, 2025, doi: 10.1109/ACCESS.2025.3624952.

[8] A. K. Sood and S. Zeadally, "Revolutionizing Cyber Defense: Leveraging Generative AI for Adaptive Threat

Hunting," Internet Technology Letters, Wiley, 2025, doi: 10.1002/itl2.70039.

[9] Hozouri et al., "A Comprehensive Survey on Intrusion Detection Systems with Advances in Machine Learning, Deep Learning and Emerging Cybersecurity Challenges," Discover Artificial Intelligence, Springer, 2025, doi: 10.1007/s44163-025-00578-1.

[10] K. Mahendar and G. Shivakanth, "A Survey of Intrusion Detection Systems Based on Machine Learning for Cloud Security," SSRG International Journal of Electrical and Electronics Engineering, vol. 12, no. 5, pp. 226-242, 2025, doi: 10.14445/23488379/IJEEE-V12I5P119.

[11] F. T. Liu, K. M. Ting and Z.-H. Zhou, "Isolation Forest," in Proc. 8th IEEE Int. Conf. Data Mining (ICDM), 2008, pp. 413-422.

[12] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459-9474.

[13] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in Proc. 4th Int. Conf. Information Systems Security and Privacy (ICISSP), 2018, pp. 108-116.

[14] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set)," in Proc. Military Communications and Information Systems Conf. (MilCIS), Canberra, 2015, pp. 1-6.

[15] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, Art. no. 15, 2009.

[16] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in Proc. IEEE Symp. Security and Privacy, 2010, pp. 305-316.

[17] The MITRE Corporation, "MITRE ATT&CK: Adversarial Tactics, Techniques and Common Knowledge." [Online]. Available: https://attack.mitre.org. [Accessed: 19-Oct-2026].

[18] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.