

Smart Threat Response System Leveraging Standard Operating Procedures and Large Language Models

Jyotibha R. Chinchankar, Ananya Devadiga, Anujna, Nidhi V. Poojari, Vamshi P. Das

Department of Computer Science (IoT & Cybersecurity incl. Blockchain), Alva's Institute of Engineering and Technology, Moodbidri, Karnataka, India

Published October 2026

Abstract

When an alert reaches a SOC analyst, someone has to work out what kind of incident it is, find the Standard Operating Procedure (SOP) that covers it, judge how serious it is and decide what to do, usually while other alerts are waiting. Searching long SOP documents by hand is slow, and under pressure it is easy to pick the wrong procedure. We introduce the Smart Threat Response Assistant, a decision-support system that takes a normalized security alert, retrieves the matching passages from the organization's SOPs using a vector database, and gives them together with the alert to a large language model (LLM). The model returns an incident summary, a severity estimate and step-by-step actions, each citing the SOP section it came from. Every recommendation is rated High, Medium or Low confidence, and Low ones go to an analyst before anything else happens. The system never acts on its own: an analyst approves, edits or rejects each recommendation, and the alert, the retrieved SOPs, the model output, the analyst's decision and the time are written to an audit log protected with SHA-256 hashes. We describe the architecture and the eight-stage pipeline, compare the system with prior work on LLMs in the SOC, and set out a plan for evaluating it on simulated alerts. This paper presents a design demonstrated on simulated alerts; we do not report measured field results.

Keywords: audit logging; human-in-the-loop; large language models; retrieval-augmented generation; standard operating procedures

1 Introduction

Incident response in most organizations still depends on people reading documents. SOPs describe how an analyst should recognize, prioritize and handle phishing, ransomware, insider misuse and unauthorized access, but they are long, and the analyst who gets the alert has to locate the right section before doing anything useful. In a busy Security Operations Center (SOC) this lookup competes with every other alert in the queue. The cost shows up as slower containment, as different analysts handling the same kind of incident differently, and as mistakes made under time pressure.

LLMs are good at reading and summarizing text, so they are an obvious candidate for this problem. The difficulty is that an LLM answering from its training data can produce a confident answer that is not based on the organization's policy, or not true at all. In security work, a wrong containment step is worse than no suggestion. We therefore use Retrieval-Augmented Generation (RAG): the system first retrieves the relevant SOP passages and only then asks the model to write, so the output can be traced to a policy document.

We built the Smart Threat Response Assistant around that idea. It is a decision-support layer, not an autonomous responder. For each alert it retrieves SOP passages from a vector database, asks an LLM for a summary, a severity estimate and recommended steps with SOP citations, attaches a confidence

rating, and then waits for the analyst. Nothing takes effect until the analyst approves, edits or rejects it, and the whole exchange is logged.

The rest of this paper is organized as follows. The next section covers related work, followed by the problem statement and objectives. The architecture, the pipeline with the triage algorithm and the technology stack are then described. The evaluation plan is set out next, followed by a discussion of limitations and future work and the conclusion.

2 Related work

Work on LLMs in security operations falls into three groups that matter for our design: studies of how analysts use LLMs, surveys of what automation exists, and architectures that push toward autonomy.

Singh et al. [5] observed GPT-4 in the SOC of eSentire for about ten months, covering 3,090 queries from 45 analysts, and coded the interactions at the level of the query, the conversation and the analyst. Analysts used the model as a support tool, mostly to interpret commands, scripts and logs and to improve how they documented findings. Of the queries, 93% related to analytic competencies in the NICE framework and about 4% were simple questions about whether a threat existed. This tells us what analysts want from an LLM, but the study does not tie the model to an organization's own procedures.

Srinivas et al. [6] reviewed 105 papers on LLMs and agents across eight SOC tasks: log summarization, alert triage, threat intelligence, ticket management, incident handling, report writing, asset discovery and vulnerability tracking. They propose a five-level maturity scale from mostly manual work to full automation, matching human-in-the-loop, human-on-the-loop and human-out-of-the-loop control, and find that most tools sit at level one or two. They list limited explainability, hallucination, integration with legacy systems, adversarial robustness and privacy as open problems. These are the problems our design tries to address directly.

Roy and Singh [7] go the other way with AgentSOC, an agentic architecture aimed at risk-aware autonomous decisions. An LLM generates attack scenarios based on MITRE ATT&CK, a simulation engine checks them against the network topology and identity-privilege relationships to remove infeasible paths, and a scoring module weighs the disruption of a response against its benefit. On the LANL authentication dataset the proof of concept responded in 506 ms. AgentSOC needs a detailed model of the network; our system needs only the SOPs the organization already has.

Other work uses LLMs and RAG for narrower security tasks: building actionable threat intelligence from knowledge graphs and LLMs [1], TTP analysis with retrieval-augmented generation [2], incident timeline analysis with RAG [3], and automatic construction of threat-intelligence knowledge graphs [4]. These support analysis of threat data. They do not retrieve the organization's response procedures for a live alert.

None of these combine, in one workflow, SOP retrieval for an incoming alert, an LLM-generated response with SOP citations, a confidence gate and an analyst decision recorded in a tamper-evident log. That combination is what this work adds. Table 1 compares the approaches.

Table 1. Comparison of the Smart Threat Response Assistant with related work.

System	Type	Control pattern	Grounding
Singh et al. [5]	Field study, GPT-4 in a SOC	Analyst-driven	Not reported

System	Type	Control pattern	Grounding
Srinivas et al. [6]	Survey of 105 papers	Mostly levels 1–2	N/A
AgentSOC [7]	Agentic architecture	Risk-aware autonomous	Network topology, ATT&CK
This work	RAG + LLM assistant	Human-in-the-loop	Organization's own SOPs

3 Problem statement and objectives

3.1 What goes wrong today

Three things tend to go wrong when an alert arrives. First, time is lost: during a live incident someone has to find the right SOP, judge severity and settle on a response, and as alert volumes grow that lookup takes longer and mistakes become more likely. Second, handling is inconsistent. The outcome depends heavily on which analyst is on shift and how well they know the SOPs. Third, existing automation does not fit well. Tools built on fixed playbooks do not adapt when something unexpected comes up, and fully autonomous AI is risky because a hallucination or a gap in context can produce advice that is wrong or unsafe.

Looking at existing solutions, several things are usually missing:

- awareness of the SOP that applies to the alert in front of the analyst;
- transparency about why an AI recommendation was made;
- traceable logs tied to each step of the process;
- a human validation step before anything is acted on;
- fast search through the organization's security procedures.

3.2 Objectives

The system has five objectives:

1. Build an assistant for cybersecurity incident response using RAG and LLMs.
2. Retrieve the SOPs that match an incoming security alert.
3. Generate explainable recommendations that cite the SOP passages behind them.
4. Give step-by-step guidance for triage, containment and remediation aligned with the SOPs.
5. Score every recommendation and flag the uncertain ones for analyst review.

4 System architecture

The system is built in modular layers: data ingestion, RAG, LLM reasoning and human validation. These break down into 11 components (Figure 1), with a React front end and a Python service layer behind it.

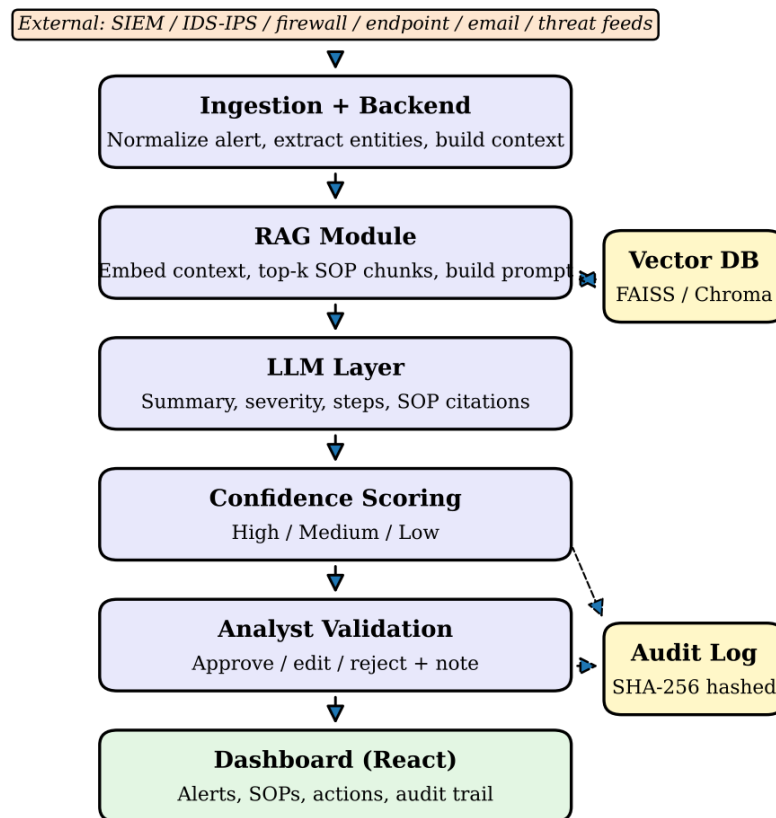


Figure 1. System architecture. The vector database serves the RAG module, and every stage from the LLM onward writes to the hash-protected audit log.

4.1 Ingestion and backend processing

The ingestion layer takes security data from SIEM/SOC alerts, IDS/IPS, firewall logs, endpoint security tools, email phishing reports and threat intelligence feeds. The backend processing component turns each alert into a structured incident by normalizing it, parsing it, extracting entities and building context. The output has the same shape whichever source the alert came from, so the rest of the pipeline does not need to know where it originated.

4.2 SOP knowledge base

SOPs in PDF, DOCX or TXT format are cleaned, normalized and split into chunks. A Sentence Transformers model converts each chunk to a dense vector, and the vectors are stored in FAISS or ChromaDB for fast similarity search. We keep the source document and section with every chunk, because the later citation step depends on being able to point back to the exact SOP section. This store is what ties the LLM to the organization's procedures.

4.3 RAG module and LLM layer

The retrieval section searches the vector database for the chunks closest to the alert context. The context generator then combines these chunks with the incident details into one prompt. The prompt goes to the LLM (OpenAI API or LLaMA), which returns an incident summary, a severity analysis, response recommendations, step-by-step instructions and the SOP references behind them. We ask

for this as a structured response so that each recommendation can be shown next to its citation on the dashboard.

4.4 Confidence scoring

Each recommendation is rated High, Medium or Low, based on how relevant the retrieved SOP passages were and how certain the generated output appears. A Low rating is routed to an analyst automatically and is not processed further. We chose a three-level rating over a numeric score because an analyst needs to decide quickly whether to look harder, not read a probability.

4.5 Human validation and audit logging

The analyst reviews the recommendation and can approve, reject or modify it, and can attach a note. The audit log records the incident details, the SOPs referenced, the AI response, the analyst's decision and the processing time. Each entry is protected with a SHA-256 hash so that later changes can be detected. Analyst decisions are also fed back to improve retrieval and recommendations over time.

4.6 Dashboard

A web dashboard shows the alert summary, the SOPs used, the AI suggestion, the severity, the audit log and the action taken. Analysts and managers see it through role-based access control.

5 Pipeline and data flow

The methodology has eight stages. Stage 1 gathers and cleans the SOPs. Stage 2 encodes them and stores the vectors. Stage 3 generates simulated alerts for phishing, malware, brute-force attempts and unauthorized access, each with the fields real tools produce: source, timestamp, indicators of compromise and affected assets. Stage 4 retrieves SOP chunks for an incoming alert and builds the prompt. Stage 5 generates the recommendation with the LLM. Stage 6 assigns the confidence rating. Stage 7 is analyst validation and audit logging. Stage 8 presents everything on the dashboard, and feedback from the dashboard returns to the retrieval and recommendation steps.

5.1 Alert triage algorithm

Algorithm 1 (Table 2) runs for every alert. Its key property is that the model never sees anything except the alert and the retrieved SOP chunks, and no recommendation reaches an action without passing through the analyst.

Table 2. Algorithm 1: Alert triage with SOP retrieval.

Require: alert a , vector index I , number of chunks k

Ensure: logged recommendation and analyst decision

1: $inc \leftarrow \text{normalize}(a)$

2: $q \leftarrow \text{embed}(\text{context}(inc))$

3: $C \leftarrow \text{top-}k \text{ chunks of } I \text{ nearest to } q$

4: $\text{prompt} \leftarrow \text{build}(inc, C)$

5: $out \leftarrow \text{LLM}(\text{prompt})$

6: $conf \leftarrow \text{score}(C, out)$

Require: alert a, vector index l, number of chunks k

7: if conf = Low then flag out for review

8: present out and conf to the analyst

9: d ← analyst decision + note

10: append_audit_entry(inc, C, out, conf, d)

11: return d

5.2 Security and threat model

The system reads text it does not control: alert fields can contain attacker-supplied strings, and SOP documents come from many authors. We treat both as untrusted input to the LLM. The model's output is requested in a fixed structure, it cannot trigger any action itself, and every output is shown to an analyst before use. Access is controlled by roles, with JWT-based authentication, so that analysts and managers see and change only what their role permits. The audit log is separate from the recommendation path, and hash verification can reveal after-the-fact edits.

6 Technology stack and implementation

Table 3. Technology stack.

Layer	Technologies	Role
Front end	React, HTML, CSS	Analyst dashboard
Back end	Python (Flask or FastAPI)	Alert intake, SOP processing, API layer
AI / NLP	LLaMA or OpenAI API, LangChain	Recommendation generation, RAG orchestration
Retrieval	Sentence Transformers, FAISS / ChromaDB	Embeddings, similarity search
Persistence	MongoDB, PostgreSQL	Alert records, SOP metadata
Security	JWT, SHA-256	Authentication, log integrity

The technologies used in each layer are listed in Table 3. Functionally, the system must upload and parse SOPs, chunk and embed them, ingest simulated alerts, retrieve SOP sections for each alert, generate summaries and recommendations, assign confidence scores, let analysts accept or reject output, and log timestamped audit entries. Non-functionally, it must handle breach data securely with role-based access between analysts and managers, return explainable recommendations in real time, and keep the SOP and vector database layer scalable with reliable audit logs.

7 Evaluation methodology and expected outcomes

This paper presents a design demonstrated on simulated alerts, not a deployed system. Rather than report field results we do not have, we set out how we intend to evaluate it.

7.1 Retrieval and citation accuracy

We will build a set of simulated alerts for the four incident types and label, by hand, the SOP sections that should be retrieved for each. We will report how often the correct section appears in the top-k

results, and how often the SOP section cited by the LLM actually supports the recommendation that cites it.

7.2 Confidence calibration

We will check whether recommendations rated Low are in fact the ones that are wrong or unsupported more often than those rated High, and how many correct recommendations the Low rating sends to analysts unnecessarily.

7.3 Analyst effort

We will compare the time an analyst needs to find and apply the correct procedure manually with the time needed to review the system's recommendation, on the same set of alerts.

7.4 Audit integrity

We will test the audit log by deliberately modifying stored entries and confirming that hash verification detects the change.

7.5 Expected impact

We expect the system to reduce the time analysts spend cross-referencing SOPs, since each recommendation arrives with its SOP section and confidence rating, and to make handling more consistent between analysts. These are expectations and not measured results; the evaluation above is needed to confirm them.

8 Discussion, limitations, and future work

The largest risk is that retrieval returns the wrong SOP chunk, or the LLM misreads a correct one. Citations make such errors easier for an analyst to spot, but they do not prevent them. The confidence rating only helps if it is well calibrated, and we have not yet tuned it. The quality of the output also depends on the quality of the SOPs: outdated or ambiguous procedures will produce outdated or ambiguous guidance.

A second limitation is the data. The system has been exercised only on simulated alerts, which are cleaner than the output of real SIEM and endpoint tools. Real alerts are noisier and often lack the fields we assume. A third is the choice of model. Using the OpenAI API means incident data leaves the organization, while a locally hosted LLaMA model keeps data in-house but may be less capable. Organizations will need to make that trade-off for themselves.

Future work includes evaluating on real SOC data, supporting more SOP formats and versioning so the system knows which SOP revision it retrieved, using analyst feedback to improve retrieval, and measuring how different LLMs behave on the same alerts.

9 Conclusion

We presented the Smart Threat Response Assistant, a decision-support system that combines RAG with an LLM to turn SOP content into guidance an analyst can act on for incidents such as phishing, ransomware, insider threats and unauthorized access. Each recommendation carries its SOP references and a confidence rating, the analyst makes every final decision, and a hash-protected audit log records what happened. Prior work has shown LLMs helping analysts, mapped the state of SOC

automation and explored autonomous architectures, but we know of none that combines these pieces around the organization's own SOPs with the analyst kept in control. The evaluation plan in Section VII describes how we intend to test whether the design delivers what it is expected to.

References

- [1] Fieblinger R, Alam T, Rastogi N, et al. Actionable cyber threat intelligence using knowledge graphs and large language models. *arXiv:2407.02528* [Preprint]. 2024.
- [2] Fayyazi R, Taghdimi R, Yang SJ. Advancing TTP analysis: Harnessing the power of large language models with retrieval-augmented generation. *arXiv:2401.00280* [Preprint]. 2024.
- [3] Loumachi FY, Ghanem MC, Ferrag MA. GenDFIR: Advancing cyber incident timeline analysis through retrieval augmented generation and large language models. *arXiv:2409.02572* [Preprint]. 2024.
- [4] Cheng Y, Bajaber O, Tsegai SA, Song D, Gao P. CTINexus: Automatic cyber threat intelligence knowledge graph construction using LLMs. *arXiv:2410.21060* [Preprint]. 2025.
- [5] Singh R, Tariq S, Jalalvand F, Baruwat Chhetri M, Nepal S, Paris C, Lochner M. LLMs in the SOC: An empirical study of human-AI collaboration in security operations centres. *arXiv:2508.18947* [Preprint]. 2025.
- [6] Srinivas S, Kirk B, Zendejas J, Espino M, Boskovich M, Bari A, Dajani K, Alzahrani N. AI-augmented SOC: A survey of LLMs and agents for security automation. *Journal of Cybersecurity and Privacy*. 2025;5(4):95. <https://doi.org/10.3390/jcp5040095>
- [7] Roy J, Singh SK. AgentSOC: A multi-layer agentic AI framework for security operations automation. *arXiv:2604.20134* [Preprint]. 2026.