WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SURAKSHA AI An Explainable Cyber Threat Detection and Safety-Governed Response Framework A Lightweight System for Telemetry Ingestion, Behavioral Anomaly Detection, Incident Correlation, and Simulated Remediation

Nethra S P[1], Lithika D B[2], Nakshatra S V[3], Lavanya R[4]

SRI SHAKTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

ABSTRACT—

Cybersecurity monitoring systems must transform heterogeneous events into actionable, explainable, and safe incident-handling decisions. This paper presents Suraksha AI, a lightweight, AI-assisted cyber threat detection and response prototype developed using a specification-driven workflow. The system normalizes security telemetry, applies deterministic rules to identify six threat categories, and uses a scikit-learn Isolation Forest model to flag unusual behaviour from rolling behavioural features. It calculates risk and confidence indicators, correlates related events into incidents, records evidence and timelines, and evaluates response actions through policy checks and safety gates. Supported simulated actions include BLOCK_IP, SUSPEND_SESSION, and ISOLATE_ENDPOINT; the prototype is explicitly designed not to alter real operating-system, account, firewall, host, or network state. The interface provides security overview, live-monitoring, threat detection, incident management, evidence, risk-and-response, safety-control, and audit views. Project phase reports describe implementation of the core detection pipeline, stream simulation, behavioural features, incident correlation, and simulated response controls. The available project context reports 15 passing tests for Phase 1, 21 for Phase 2, and 33 for Phase 3; these are phase-level reported counts and should not be treated as a single independently rerun test total. The present evidence supports a functional prototype and a safety-oriented architecture, not a claim of production-grade detection accuracy or live network defence. Quantitative precision, recall, false-positive rate, throughput, and persistence-recovery results remain to be established through a controlled evaluation.

Keywords—Cyber Threat Detection, Anomaly Detection, Isolation Forest, Incident Correlation, Explainable Risk, Simulated Incident Response

I. INTRODUCTION

Organizations generate security events from authentication services, endpoints, network connections, and privileged operations. Individual events often have limited meaning in isolation: repeated login failures may indicate a password-guessing attempt, while a sequence of authentication anomalies, privilege changes, and unusual outbound connections may suggest a broader incident. A useful detection workflow therefore needs event normalization, behavioural context, evidence aggregation, incident-level correlation, and a controlled response process. Incident response guidance emphasizes integrating preparation, detection, response, and recovery into cybersecurity risk management rather than treating alerts as isolated notifications [1].

A. Problem Statement and Objectives

The project addresses five practical problems: (1) security events arrive in inconsistent formats; (2) static rules may miss previously unseen behavioural deviations; (3) separate alerts may describe one developing incident; (4) a risk score without supporting evidence is difficult to investigate; and (5) automated remediation can cause harm if authorization, target scope, or verification is uncertain. Suraksha AI combines deterministic rules and anomaly scoring with incident correlation, explainable risk factors, and safety-constrained simulated actions. Its objective is an inspectable educational prototype, not a replacement for a production SIEM, EDR, or security operations team.

B. Contributions

The paper describes an integrated pipeline comprising telemetry normalization and deduplication; six rule-based threat categories; a six-dimensional behavioural feature representation for Isolation Forest; risk and confidence indicators; correlation by shared user, source IP, or asset within a reported ten-minute window; incident timelines and evidence; a centralized response-policy engine; simulated execution, verification, and rollback protections; role-based access; and an audit-oriented dashboard. The project's core design principle is that anomaly scores provide evidence but do not independently authorize response. The response layer must enforce policy, authorization, scope, auditability, and verification before any simulated action is considered executable.

II. LITERATURE REVIEW

Rule-based detection remains useful for known patterns because its decisions can be inspected and tested, but fixed signatures may not represent every legitimate behavioural baseline or emerging attack. Anomaly detection provides a complementary approach by identifying observations that differ from learned or assumed normal behaviour. Isolation Forest isolates observations using randomly selected features and split values; anomalies tend to be isolated with shorter paths in the resulting trees [2]. Its output is an anomaly indicator, not proof of malicious intent, and it requires evaluation on representative data before operational use.

A. Threat Taxonomies and Incident Response

MITRE ATT&CK organizes adversary behaviour into tactics and techniques and can support consistent vocabulary when describing credential access, privilege escalation, lateral movement, command-and-control, and related activity [3]. Suraksha AI's implemented categories—Suspicious Login, Brute Force, Privilege Escalation, Lateral Movement, C2 Beaconing, and Unusual Data Transfer—are project-level detection categories. They should not be represented as a complete ATT&CK mapping unless each rule is mapped to specific techniques and validated. NIST SP 800-61 Rev. 3 places incident response within the broader Cybersecurity Framework 2.0 risk-management context [1]; Suraksha AI follows the compatible high-level principle of connecting detection evidence to investigation and response while retaining human oversight for uncertain cases.

B. Explainability and Safety-Constrained Automation

Operational security decisions need evidence that analysts can review. A scalar risk score is most useful when accompanied by the contributing events, affected assets, confidence indicators, and a traceable incident timeline. Likewise, response automation should not treat a high anomaly score as sufficient authorization. Suraksha AI therefore separates detection evidence from deterministic policy evaluation and simulates remediation rather than performing real host or network changes. This boundary reduces the risk of a student demonstration accidentally disrupting a device or network, but it also means that successful simulation does not demonstrate the effectiveness of a real containment control.

C. Research Gap

The project's contribution is integrative and educational: it brings rules, anomaly scoring, correlation, risk explanation, response policy, simulation, rollback concepts, and audit views into a small application. This paper does not claim a new anomaly-detection algorithm or superiority over commercial products. The key research question is whether this architecture can provide a coherent and testable workflow while making its limitations and safety boundaries explicit.

III. METHODOLOGY

Suraksha AI is organized as a sequence of processing stages: ingest security telemetry; normalize and deduplicate events; update rolling behavioural features; run rule-based and anomaly detection; calculate risk and confidence; correlate related events; evaluate response policy and safety conditions; simulate the permitted action; verify the simulated outcome; and record the decision and its evidence. This sequence separates event interpretation from response authority and provides traceability from raw event to incident and audit record.

TABLE I. PROCESSING STAGES OF THE SURAKSHA AI PIPELINE

Stage	Purpose	Output / control
1. Ingest and normalize	Validate and standardize event fields	Normalized event; malformed input handled
2. Deduplicate and extract features	Handle stream timing and rolling behaviour	Six-dimensional behavioural vector
3. Detect	Apply rules and Isolation Forest	Threat evidence and anomaly indicator
4. Assess and correlate	Calculate risk indicators and group related events	Evidence, incident timeline, risk drivers
5. Evaluate response	Check policy, authorization, scope, kill switch and audit readiness	Permit, deny, alert-only, or human review
6. Simulate, verify, audit	Simulate supported response and record outcome	Verification/rollback record and audit trail

A. System Architecture

The documented stack uses Python with FastAPI [4] and Pydantic for backend APIs, scikit-learn Isolation Forest [5] for anomaly detection, pytest for automated testing, and HTML/CSS/JavaScript for the dashboard. Current completed phases primarily use in-memory runtime state. SQLite persistence and live connectors are identified as later work in the project roadmap and must not be described as already completed. The dashboard is organized into Security Overview, Live Monitoring, Threat Detection Center, Incident Management, Threat Evidence, Risk & Response, Safety Controls, and Audit Logs & Status. Figures 1–4 reproduce user-provided screenshots of these interface views; their visible demo counters are illustrative interface state, not experimental findings.

TABLE II. ARCHITECTURE LAYERS, RESPONSIBILITIES AND STATUS CAVEATS

Layer / component	Documented responsibility	Status caveat
Telemetry and feature layer	Normalize events, deduplicate, calculate rolling metrics	In-memory runtime described
Detection layer	Six rules plus Isolation Forest	Accuracy not quantified
Risk and incident layer	Risk indicators, evidence, correlation and timelines	Calibration not established
Policy and response layer	Safety gates and simulated actions	Tier semantics and core-asset re-strictions need verification
API and dashboard	FastAPI and browser UI	Endpoint inventory to be confirmed from repository
Persistence / live connectors	Durable history and additional ingestion	Roadmap items, not completed capabili-ties

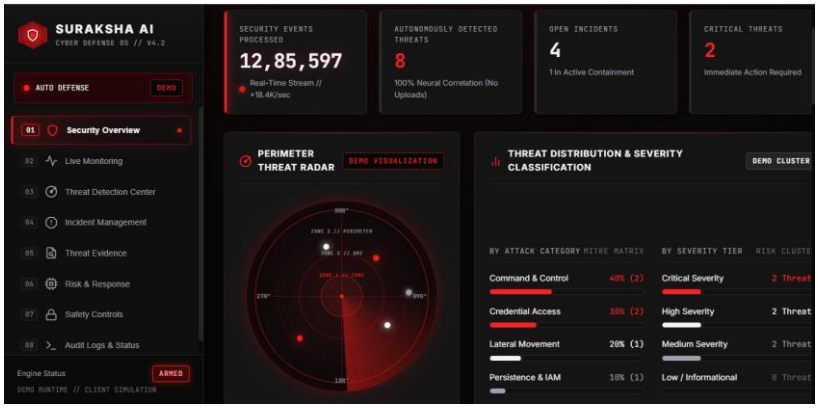


Fig. 1. Suraksha AI Security Overview. Demo counters are illustrative UI state.

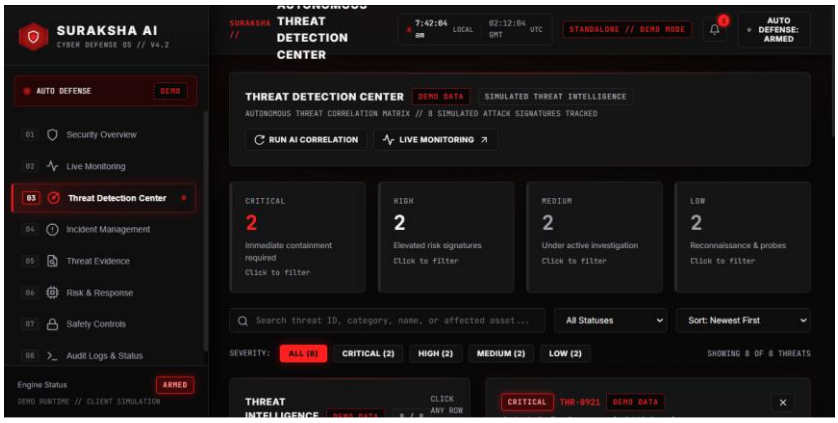


Fig. 2. Threat Detection Center with severity filters and threat records.

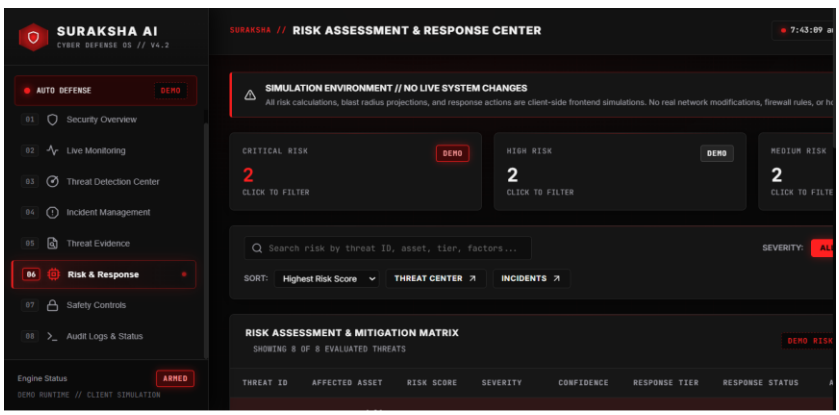


Fig. 3. Risk Assessment and Response Center.

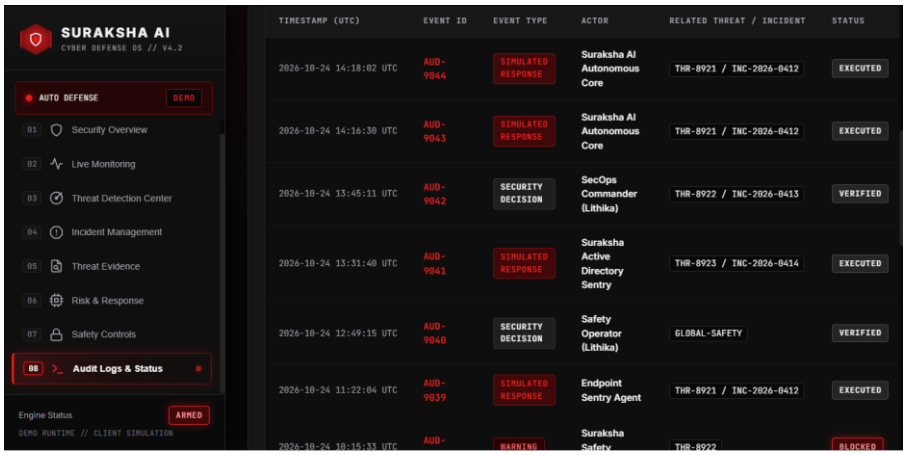


Fig. 4. Audit Logs and Status view; records shown are simulation/demo data.

B. Telemetry Normalization and Stream Handling

The Phase 1 and Phase 2 project context [6] reports telemetry normalization, batch ingestion, a stream simulator, sliding-window deduplication using a reported 60-second window, out-of-order event handling, and malformed-input fallback. The feature engine maintains rolling metrics for failed-login count, login frequency, off-hours activity, unique destinations, privilege-change events, data-transfer volume, and connection frequency. These features form a six-dimensional vector for the Isolation Forest stage. The paper does not assign a specific external dataset, event volume, or ground-truth label set because none is documented in the supplied project context. Consequently, the feature design can be described, but dataset representativeness and generalization cannot yet be quantified.

C. Rule-Based and Behavioural Detection

Six rule categories are documented: Suspicious Login, Brute Force, Privilege Escalation, Lateral Movement, C2 Beaconing, and Unusual Data Transfer. Rules offer explicit evidence for known event patterns. Isolation Forest provides a complementary anomaly score based on the behavioural feature vector. The intended architecture combines the signals for triage and risk assessment; an anomaly score alone must not authorize a response. Before publication of an empirical accuracy claim, the implementation should be evaluated against labelled benign and malicious scenarios, with thresholds fixed on a validation set and a separate test set reserved for final reporting.

D. Risk Scoring and Incident Correlation

The prototype reports a risk score on a 0–100 scale and a confidence score on a 0–1 scale, with explainable risk drivers represented through evidence fields such as why_evidence. The supplied project context does not specify a validated calibration procedure for these values; they should therefore be presented as system indicators rather than calibrated probabilities of compromise. Related events are correlated using a shared user, source IP, or asset within a reported ten-minute window. Correlated events are assembled into incident timelines and exposed through investigation APIs and dashboard views. The correlation window is a project parameter, not evidence that all events within ten minutes belong to the same attack.

E. Response Policy and Safety Gates

The response layer evaluates proposed actions such as BLOCK_IP, SUSPEND_SESSION, and ISOLATE_ENDPOINT in a simulation-only environment. The project context lists ten pre-execution checks: authorization; policy compatibility; kill switch; target scope and core-asset checks; blast-radius limit; audit-persistence pre-check; pre-state capture; idempotency; timeout handling; and independent verification. The intended response-tier semantics are Tier A: automatic only when all required safety checks pass; Tier B: automatic with immediate notification, subject to policy and checks; Tier C: human approval required, with no automatic execution. Missing or uncertain safety information should fail closed to alert-only or human review. The project roadmap explicitly flags tier semantics and action-specific core-asset restrictions for verification; they must not be claimed as fully validated until code and regression tests confirm them.

F. Verification, Rollback, and Auditability

The design includes simulated execution, outcome verification, rollback protections, a kill switch, role-based access, and audit logging. The audit trail is intended to connect the alert or incident, the selected response, the policy decision, the simulated result, and the verification or rollback outcome. Phase 3 context reports these as implemented features, but a publication-ready evaluation should separately test successful execution, failed verification, timeout, repeated requests, idempotency, audit-store failure, and rollback behaviour. Because the present system is simulation-only, verification means that the application's simulated state matches its expected result; it does not demonstrate that an external firewall, account, process, or endpoint was changed.

G. API and User Interface

FastAPI exposes the backend through HTTP endpoints, and the project includes telemetry ingestion, detection, incident investigation, and response-related API functionality. Exact endpoint paths and schemas should be copied from the actual repository and OpenAPI specification before submission; they are not invented here. The screenshots show a dark, red-accented interface with overview counters and a perimeter-style threat visualization (Fig. 1), a threat detection center with severity filters (Fig. 2), a risk assessment and response matrix (Fig. 3), and an audit event table (Fig. 4). Counts dis-

played in the screenshots—such as eight demo threats, four open incidents, and two critical threats—are not treated as independently measured system performance.

H. Formal Workflow and Complexity

Let E be an input event, $F(E)$ its normalized representation, B_t the behavioural feature vector at time t, D_r the rule-derived evidence, D_a the anomaly output, and C the set of related events assigned to an incident. The detection stage produces evidence $A = \text{combine}(D_r, D_a, F(E), B_t)$, where combine denotes the implemented integration logic and should be replaced by the exact repository function or equation in a final reproducibility revision. A risk function $R(A)$ yields the system's risk indicator and associated evidence. A response proposal P is passed to a policy function $G(P, \text{user, asset, state, audit_status})$; execution is permitted only when the required gates pass. Otherwise the result is alert-only, denied, or routed for human review. This formulation is intentionally conceptual because the supplied project context does not define a single mathematical risk formula or the exact Isolation Forest threshold. Complexity and latency should be measured from code under a stated workload rather than inferred from the number of pipeline stages.

I. Implementation and Configuration

The documented implementation stack is Python, FastAPI, Pydantic, scikit-learn, pytest, and a lightweight browser interface. The current completed phases primarily rely on in-memory runtime state. SQLite persistence, single-event real-time ingestion, generic JSON webhooks, optional local UDP/syslog-style input, and restart recovery appear in the Phase 4 roadmap and are therefore future work, not current capabilities. Likewise, live connectors and production enforcement are outside the verified scope. For reproducibility, the final submission should record the commit hash, dependency lockfile, Python version, operating system, model parameters, random seeds, event schema, and exact test command.

J. Evaluation Design

Evaluation should cover functional correctness, detection quality, operational behaviour, and response safety. Functional tests should verify schema validation, normalization, deduplication, out-of-order events, feature calculation, rule triggers, anomaly outputs, incident correlation, lifecycle transitions, RBAC, and audit records. Detection evaluation should use a documented labelled dataset or a transparently generated replay corpus and report precision, recall, F1-score, false-positive rate, and a confusion matrix per category. Operational tests should report throughput and p50/p95/p99 latency at stated event rates and concurrency. Safety tests should cover each gate, core-asset restrictions, kill-switch behaviour, audit persistence failure, verification failure, rollback, and idempotency. Results must identify the environment, dataset, sample count, and whether values are measured, simulated, or analytically derived.

IV. RESULTS

The evidence available for this revision is implementation-context documentation and four screenshots of the interface. It does not include a fresh run of the source repository, a labelled benchmark dataset, or controlled load-test outputs. The results below are therefore limited to project-reported implementation and testing status; they do not claim measured detection accuracy or production performance.

TABLE III. PROJECT-REPORTED TEST STATUS BY PHASE

Phase	Reported scope	Reported tests	Evidence status
Phase 1	Foundation, normalization, six rules, initial Isolation Forest, risk/confidence, incidents, simulated responses	15 passed	Reported; not rerun here
Phase 2	Stream simulation, deduplication, behavioural features and anomaly evidence	21 passed	Reported; not rerun here
Phase 3	Correlation, response policy, safety gates, simulated verification/rollback, RBAC and audit	33 passed	Reported; semantics need verification

A. Reported Phase-Level Tests

The project context reports 15 tests passed for Phase 1, 21 for Phase 2, and 33 for Phase 3. These figures are retained as reported phase-level results. They should not be summed and presented as a single verified total because the supplied context does not establish whether tests overlap, whether all phases were rerun together, or whether the current repository passes in a clean environment. A final journal version should include a reproducible test log with command, date, environment, collected/passed/failed/skipped counts, and any known failures.

B. Demonstrated Functional Areas

The project context records the following as completed across Phases 1–3: normalization; six rule-based categories; an initial Isolation Forest detector; risk and confidence indicators; stream simulation and behavioural features; event deduplication and malformed-input handling; incident correlation and timelines; response policy evaluation; simulated response, verification, and rollback protections; RBAC and audit logging; FastAPI endpoints; and a lightweight dashboard. Figures 1–4 document the appearance of key dashboard areas. These screenshots demonstrate interface layout only; they do not independently prove that each displayed control is connected to a working backend or that displayed counters were generated by a live data stream.

C. Detection and Performance Metrics Not Yet Established

No precision, recall, F1-score, false-positive rate, labelled test-set size, sustained throughput, or controlled p50/p95/p99 latency result is available in the

supplied project context. Accordingly, no such metric is reported. The Isolation Forest stage is described as implemented, but the available material does not establish its detection accuracy or comparative advantage over rules alone. A controlled experiment should compare (i) rules only, (ii) anomaly detection only, and (iii) the combined pipeline on the same labelled scenarios. Thresholds must be fixed before testing, and results should be reported per threat category as well as overall.

D. Response Safety Evidence

The architecture and project roadmap specify safety checks and simulated actions. However, the context itself marks response-tier semantics and action-specific core-asset restrictions as items to verify. The publication should therefore characterize the safety design as implemented in the prototype but subject to further semantic verification, unless repository inspection and regression-test results demonstrate otherwise. No claim is made that Suraksha AI has blocked a real IP address, suspended a real session, or isolated a real endpoint.

V. DISCUSSION

The main contribution is the integration of detection, evidence, incident context, policy evaluation, and simulated response in one inspectable workflow. The use of rules alongside anomaly detection can help represent both known patterns and deviations from observed behaviour, while event correlation can give analysts a more coherent incident view than a list of independent alerts. The explicit separation between detection evidence and response authority is a useful safety property for an educational system.

A. Interpretation and Implications

The prototype supports a demonstration of the end-to-end concept: telemetry enters the pipeline, detection produces evidence, related events are correlated, risk is displayed, and a policy-governed simulated action is audited. The safety boundary is central: the application is not intended to change the real machine or network. The dashboard screenshots are appropriate illustrations of system design, but visual polish should not be confused with evidence of backend correctness. In particular, a button that appears in the UI should be tested for the expected API request, response handling, and visible state change.

B. Threats to Validity

Construct validity is limited because risk and confidence indicators are not documented as calibrated probabilities, and anomaly outputs do not themselves establish malicious intent. Internal validity is limited by the absence of a supplied, independently rerun test log and controlled benchmark. External validity is limited because no representative production telemetry, multi-organization study, or broad environment evaluation is documented. Reproducibility is also constrained by the lack of a recorded repository commit, exact dependency versions, and a fully specified dataset. The reported phase test counts should be treated as project records pending reproducible confirmation.

C. What Has Not Been Measured

The available materials do not establish detection precision or recall, false-positive burden, robustness to adversarial evasion, performance under sustained concurrent streams, memory usage, persistence and restart recovery, or the operational effectiveness of real containment. They also do not establish that response-tier semantics and core-asset rules have passed a complete adversarial test suite. These are evaluation gaps, not negative findings; they should be addressed before making stronger claims.

D. Limitations, Risks, and Ethical Considerations

A false positive can consume analyst time, while a false negative can leave malicious activity undetected. Behavioural baselines may drift, and an anomaly detector can be sensitive to feature scaling, contamination assumptions, and training data. Correlation based on shared identifiers and a fixed time window can merge unrelated events or split one campaign into multiple incidents. Audit logs may contain sensitive identifiers and should be access-controlled, minimized, and retained according to a defined policy [7]. The system's simulated-only response boundary must be clearly disclosed in demonstrations. Any future real-response integration would require additional authorization, change control, independent verification, rollback planning, and testing in a controlled environment.

VI. CONCLUSION

This paper presents Suraksha AI, a lightweight AI-assisted cyber threat detection and safety-governed response prototype. Its documented workflow combines normalized telemetry, deterministic detection rules, Isolation Forest anomaly scoring, behavioural features, risk evidence, incident correlation, response policy checks, simulated remediation, verification concepts, rollback protections, role-based access, and audit-oriented dashboard views. The project context reports completed Phase 1–3 implementation areas and phase-level test counts, but the supplied evidence does not establish production detection accuracy, sustained performance, persistent recovery, or real-world containment effectiveness. The system's most defensible contribution is its integrated, explainable workflow and explicit rule that AI evidence must not independently authorize remediation.

A. Takeaways

Suraksha AI provides a practical educational architecture for linking event-level detection to incident-level investigation and controlled simulated response. The approach emphasizes explainability, fail-closed safety behaviour, and auditability. These properties are valuable only when they are confirmed by code-level and regression tests; the next development stage should prioritize verification of response tiers and core-asset restrictions before adding further integrations.

B. Limitations

The present paper is based on the supplied project roadmap and screenshots rather than an independent code execution. It lacks labelled detection benchmarks, controlled performance results, and confirmed SQLite-based persistence. The project context explicitly lists persistence and live-ingestion reliability as planned Phase 4 work. The reported response-tier semantics also require verification before the safety model can be described as fully validated.

C. Future Work

Near-term work should first verify the Phase 3 response-tier semantics and action-specific core-asset restrictions, then implement and test SQLite persistence for events, incidents, response actions, verification, rollback, and audit history. A single-event ingestion API and generic JSON webhook can follow, with optional local syslog-style ingestion if practical. The system should fail closed when mandatory audit persistence cannot be guaranteed. Later work should build replayable normal and attack scenarios, evaluate detection with precision/recall/F1 and false-positive rate, measure throughput and tail latency, test restart recovery, document installation and demonstration steps, and perform end-to-end regression testing. Real firewall, account, process, or endpoint actions should remain out of scope unless separately designed and authorized.

REFERENCES :

[1] A. Nelson, S. Rekhi, M. Souppaya, and K. Scarfone, “Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile,” NIST Special Publication 800-61 Rev. 3, National Institute of Standards and Technology, Apr. 2025. doi: 10.6028/NIST.SP.800-61r3.

[2] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation Forest,” in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.

[3] MITRE, “MITRE ATT&CK Enterprise Matrix.” Available: https://attack.mitre.org/matrices/enterprise/ (accessed 30 Oct. 2026).

[4] FastAPI, “FastAPI Documentation.” Available: https://fastapi.tiangolo.com/ (accessed 30 Oct. 2026).

[5] scikit-learn developers, “IsolationForest,” scikit-learn User Guide and API Reference. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html (accessed 30 Oct. 2026).

[6] Suraksha AI project documentation supplied by the authors: Project Requirements, Architecture, Threat Model, Phase 1–3 completion context, and Phase 4 roadmap. Unpublished internal project materials, 2026.

[7] OWASP Foundation, “Logging Cheat Sheet,” OWASP Cheat Sheet Series. Available: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html (accessed 30 Oct. 2026).