



Cloud Shield IDS: Design and Evaluation of a Real-Time AI-Powered Intrusion Detection System Using XG Boost and Cloud Integration

Indhushree S¹, Elanthamil G. P², Fathima Mehin³, Jeevanthraj P⁴, Fathima Jemima P⁵

^{1,2,3,4}Department of Computer Science and Engineering (Cyber Security), United Institute of Technology, Coimbatore, Tamil Nadu, India

⁵Teaching Assistant (TA), Department of Computer Science and Engineering (Cyber Security), United Institute of Technology, Coimbatore, Tamil Nadu, India

To Cite this Article: Indhushree S¹, Elanthamil G. P², Fathima Mehin³, Jeevanthraj P⁴, Fathima Jemima P⁵, “Cloud Shield IDS: Design and Evaluation of a Real-Time AI-Powered Intrusion Detection System Using XGBoost and Cloud Integration”, Indian Journal of Computer Science and Technology, Volume 05, Issue 02 (May-August 2026), PP: 64-71.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by-nc-nd/4.0/); Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: The proliferation of sophisticated cyber-attacks against networked systems demands robust, real-time intrusion detection capabilities that transcend the limitations of purely rule-based approaches. This paper presents CloudShield IDS, a hybrid intrusion detection system that integrates machine learning-based classification with deterministic rule-based heuristics, cloud-based persistent storage via Supabase (PostgreSQL), and an interactive real-time Streamlit dashboard. The system employs an XGBoost classifier trained on network traffic features to distinguish normal activity from malicious patterns including port scans, distributed denial-of-service (DDoS) attacks, brute-force login attempts, and SQL injection exploits. The proposed pipeline captures raw network events through a dedicated listener module, classifies them through a dual-layer processor, and persists structured alerts to a cloud database from which the dashboard retrieves and visualises threat intelligence. Experimental evaluation demonstrates an overall detection rate of 97.8%, a precision of 97.2%, and a false positive rate of approximately 2.8%, outperforming conventional rule-based systems such as Snort and Suricata across multiple attack categories. The architecture is modular, extensible, and deployable both locally and on cloud infrastructure, making it suitable for enterprise and research environments. Claude AI was leveraged throughout the development lifecycle for code generation, architecture design, debugging, and optimisation, illustrating the practical utility of large language models in applied cybersecurity engineering.

Key Words: Intrusion Detection System; XGBoost; Machine Learning; Network Security; Supabase; Streamlit; Real-Time Monitoring; DDoS; Port Scan; Cloud Integration.

I. INTRODUCTION

The modern digital infrastructure is under continuous and intensifying assault from a diverse range of cyber threats. Malicious actors exploit software vulnerabilities, misconfigured services, and weak authentication mechanisms to gain unauthorised access to systems, exfiltrate sensitive data, disrupt operations, or deploy ransomware payloads. The financial and reputational consequences of successful intrusions have escalated dramatically; according to the IBM Cost of a Data Breach Report, the average cost of a data breach reached 4.45 million USD in 2023, representing a multi-year high. Against this backdrop, robust intrusion detection systems (IDS) form a critical layer of defence in any mature cybersecurity architecture.

Traditional intrusion detection systems have historically been partitioned into two broad paradigms: signature-based (rule-based) systems and anomaly-based systems. Signature-based IDS, exemplified by widely deployed open-source tools such as Snort and Suricata, operate by matching observed network traffic or host activity against a curated database of known attack signatures. While highly effective against well-documented, previously characterised threats, they are inherently reactive: zero-day attacks, novel malware variants, and polymorphic exploits that deviate from catalogued signatures readily evade detection. Anomaly-based IDS, by contrast, establish a statistical baseline of normal behaviour and raise alerts when observed activity deviates significantly from that baseline. Their principal advantage lies in the theoretical ability to detect unknown threats; however, naive anomaly detection models suffer from elevated false positive rates that erode analyst confidence and introduce alert fatigue in security operations centres (SOCs).

The emergence of machine learning—and in particular gradient-boosted ensemble methods—has created a compelling third paradigm: supervised classification of network traffic into normal and attack categories using features extracted from packet headers and flow-level statistics. Systems built on algorithms such as Random Forest, Support Vector Machines, and XGBoost have consistently demonstrated high accuracy and low false-positive rates on benchmark datasets including NSL-KDD, CICIDS 2017, and UNSW-NB15. Nevertheless, deploying such models in production introduces engineering challenges related to real-time data ingestion, scalable storage, alert management, and interactive visualisation that purely algorithmic research frequently overlooks.

CloudShield IDS addresses these challenges holistically. It presents a complete, end-to-end intrusion detection pipeline that couples an XGBoost-based ML classifier with a deterministic rule engine, a cloud-native database layer (Supabase, built on PostgreSQL), and an interactive Streamlit web dashboard. The system is engineered in Python 3.11, ensuring compatibility with the mature scientific computing ecosystem, and employs environment-variable-based credential management to comply with security best practices. By combining the predictive power of machine learning with the interpretable specificity of rule-based detection, CloudShield IDS achieves a dual-layer detection strategy that mitigates the weaknesses of each individual approach.

A further distinctive aspect of this work is the systematic integration of Claude AI throughout the software development lifecycle. Large language models (LLMs) were leveraged for code generation, architecture design review, unit-test authoring, debugging assistance, and documentation synthesis. This experience provides empirical evidence for the emerging software engineering practice of AI-assisted development and demonstrates that complex, multi-module cybersecurity tools can be designed and iterated upon substantially more rapidly when LLMs serve as collaborative engineering partners.

The remainder of this paper is structured as follows. Section 2 reviews related work in machine-learning-based intrusion detection. Section 3 details the system architecture and design rationale. Section 4 describes the technologies employed. Section 5 elaborates the module-level implementation. Section 6 presents the database schema and data flow. Section 7 documents the security design. Section 8 reports experimental results and performance evaluation. Section 9 compares CloudShield IDS with existing tools. Section 10 discusses challenges encountered and resolutions applied. Section 11 outlines future enhancements. Section 12 concludes the paper.

II. RELATED WORK

Intrusion detection has been an active research domain for over three decades, producing an extensive body of literature spanning statistical methods, expert systems, and, more recently, deep learning architectures. This section surveys the most pertinent prior art and contextualises CloudShield IDS within the existing landscape.

2.1 Rule-Based Systems

Snort, originally released by Roesch in 1999 and subsequently maintained by Cisco Talos, remains one of the most widely deployed network IDS tools globally. Its packet inspection engine evaluates traffic against a continuously updated rule repository, generating alerts when pattern matches occur. Suricata, developed by the Open Information Security Foundation (OISF), extends this model with multi-threaded processing, native IPv6 support, and integrated file extraction capabilities. Both systems have low computational overhead at runtime and exhibit near-zero false positives for known attack signatures. Their critical limitation, however, is the inability to detect novel or obfuscated variants of attacks whose signatures have not been catalogued.

2.2 Anomaly-Based and Hybrid Systems

Bace and Mell (2001) provided an early taxonomy of anomaly detection approaches, distinguishing statistical models, knowledge-based methods, and machine-learning classifiers. Subsequent benchmark studies on the DARPA 1998 and KDD Cup 1999 datasets revealed that naive Bayes and decision-tree classifiers could achieve high detection rates but were sensitive to dataset bias. The NSL-KDD dataset, proposed by Tavallaee et al. (2009), addressed some of these limitations and became a standard benchmark. Studies on NSL-KDD using Random Forest and SVM consistently report detection rates exceeding 95% with false positive rates below 5%. Liao et al. (2013) conducted a comprehensive survey of intrusion detection approaches, identifying feature selection and real-time applicability as persistent open challenges.

2.3 Deep Learning Approaches

The advent of deep learning has motivated exploration of recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and convolutional neural networks (CNNs) for intrusion detection. Javaid et al. (2016) proposed a deep autoencoder-based IDS, demonstrating that unsupervised representation learning could detect anomalous traffic with competitive accuracy. Kim et al. (2016) applied LSTM networks to network log sequences, achieving high detection rates for temporal attack patterns. Despite their expressive power, deep learning models introduce significant training overhead, require large labelled datasets, and produce opaque predictions that hinder analyst interpretability.

2.4 XGBoost for Network Intrusion Detection

XGBoost (Chen & Guestrin, 2016) has emerged as a consistently top-performing algorithm on structured and tabular data competitions, including network traffic classification. Zhang et al. (2019) demonstrated that XGBoost outperformed Random Forest, k-NN, and SVM on the CICIDS 2017 dataset, achieving 98.4% accuracy with a training time approximately 40% lower than equivalent deep learning models. Gradient boosting's inherent feature importance scoring also supports interpretability, enabling security analysts to understand which traffic features contributed most strongly to a classification decision. CloudShield IDS builds upon this evidence base by deploying XGBoost as the primary classification engine.

2.5 Cloud-Integrated and Dashboard-Enabled IDS

Despite extensive algorithmic research, relatively few published systems address the full pipeline from packet capture to cloud-based persistence and real-time dashboard visualisation. Scarfone and Mell (2007) in the NIST IDS guide highlighted operational deployment and alert management as underserved areas. Sallay et al. (2011) proposed a distributed IDS architecture for cloud environments but did not integrate machine learning. CloudShield IDS directly addresses this gap by coupling ML-based classification with Supabase-backed persistence and a Streamlit dashboard, providing an end-to-end reference implementation suited to both research and operational deployment.

III. SYSTEM ARCHITECTURE

Cloud Shield IDS is designed around a linear, modular pipeline architecture in which each stage performs a well-defined transformation on the data stream. The high-level architecture is expressed as follows:

Network Traffic → Listener Module → Processor (ML + Rules) → Alert Generator → Supabase (PostgreSQL) → Streamlit Dashboard

Each arrow in the pipeline represents an asynchronous handoff in which data is transformed from raw packet representations into progressively higher-level abstractions. The modular decomposition ensures that each component can be independently developed, tested, replaced, or scaled without affecting the remainder of the system.

3.1 Listener Module

The Listener module is responsible for capturing network events from the host operating system's network interface. It abstracts the raw socket or packet capture library (such as `scapy` or `pcap`) into a normalised event representation—a Python dictionary containing source IP address, destination IP address, protocol, port numbers, packet size, and inter-arrival timing. These normalised events are enqueued in an in-memory queue and consumed asynchronously by the Processor.

3.2 Processor Module

The Processor constitutes the analytical core of CloudShield IDS. It applies a two-layer detection strategy. In the first layer, the XGBoost classifier evaluates a feature vector derived from the normalised event and outputs a probability distribution across traffic categories including Normal, Port Scan, DDoS, Brute Force, and SQL Injection. When the highest-probability category exceeds a configurable confidence threshold, the classification is accepted. In the second layer, a deterministic rule engine evaluates traffic against a set of human-authored heuristics, such as SYN packet rate exceeding a threshold (indicative of port scanning), login failure counts within a time window (indicative of brute force), and the presence of SQL meta-characters in HTTP query strings (indicative of injection attempts). The dual-layer approach allows the rule engine to catch high-confidence, well-characterised attacks with zero latency, while the ML model handles novel and ambiguous traffic patterns.

3.3 Alert Generator

When either layer produces a positive detection, the Alert Generator constructs a structured alert record containing all mandatory fields (source IP, destination IP, attack type, severity classification, human-readable reason string, timestamp, and detection method). Severity levels are mapped from attack type according to a pre-defined taxonomy: DDoS and SQL Injection are rated CRITICAL; Port Scan and Brute Force are rated HIGH; ambiguous anomalies are rated MEDIUM; and normal traffic confirmed by both layers is rated LOW or suppressed.

3.4 Storage Layer

Alert records are persisted to a Supabase-hosted PostgreSQL table via the `supabase-py` client library. Supabase provides a fully managed PostgreSQL instance with a RESTful API, real-time subscriptions, and row-level security. The use of a cloud database decouples the detection pipeline from the visualisation layer, enabling both horizontal scaling and multi-node deployments in which multiple Listener-Processor pairs write to a single centralised store.

3.5 Streamlit Dashboard

The dashboard is implemented as a Streamlit web application (`app.py`) that queries Supabase at configurable intervals and renders alerts in tabular and graphical formats. Key visualisation components include a real-time alert feed, severity distribution pie charts, attack type frequency histograms, source IP geolocation maps (planned for future enhancement), and time-series line charts depicting alert volume over time. The dashboard employs Streamlit's session state mechanism to maintain filter selections across refresh cycles.

IV. TECHNOLOGIES USED

The technology stack of CloudShield IDS was selected to maximise development velocity, runtime performance, and long-term maintainability. Table 1 summarises the principal technologies and their roles within the system.

Category	Technology	Role	Justification
Backend	Python 3.11	Core logic & data processing	Mature ecosystem for ML/data science
ML Model	XGBoost	Traffic classification	Superior performance on tabular/network data
Frontend	Streamlit	Interactive web dashboard	Rapid prototyping with Python-native UI

Category	Technology	Role	Justification
Database	Supabase (PgSQL)	Persistent alert storage	Open-source Firebase alternative with REST API
Libraries	pandas	Data manipulation & analytics	Industry standard for structured data
Security	python-dotenv	Credential management	Prevents secrets from being hardcoded

Table 1: Technology Stack of CloudShield IDS

Python 3.11 was chosen as the implementation language due to its dominance in the data science and security tooling communities, its extensive library ecosystem, and its ongoing performance improvements (the CPython 3.11 release delivered 10-60% speed improvements over 3.10 in several benchmarks). XGBoost was selected over alternative ensemble methods on the basis of its superior performance on tabular network traffic data documented in the related work review. Streamlit was preferred over heavier frameworks such as Django or Flask for the dashboard component due to its ability to convert plain Python scripts into interactive web applications without requiring HTML/CSS/JavaScript expertise, substantially reducing development time. Supabase was chosen over self-hosted PostgreSQL to eliminate infrastructure management overhead and to leverage its native RESTful API, which integrates cleanly with the supabase-py client.

V. MODULE-LEVEL IMPLEMENTATION

CloudShield IDS is composed of five primary Python modules, each encapsulating a distinct responsibility. Table 2 provides a module-level summary.

Module	File	Primary Function	Key Dependencies
Network Listener	listener.py	Capture & forward network events	socket, pandas
Traffic Processor	processor.py	Apply ML model & rule engine	XGBoost, pandas
Database Client	supabase_client.py	Persist alerts to cloud database	supabase-py
Execution Engine	main.py	Orchestrate modules & event loop	All modules
Dashboard	app.py	Visualize alerts in real-time browser UI	Streamlit, pandas

Table 2: Module Summary

5.1 listener.py

The listener module initialises a raw socket bound to the monitored network interface and enters a blocking receive loop. Each received packet is parsed using a lightweight header parser that extracts the IP layer fields (source and destination addresses, TTL, protocol), the transport layer fields (source and destination ports, TCP flags, UDP length), and application-layer metadata where available. The extracted fields are normalised into a standard Python dictionary and placed onto a thread-safe queue (queue.Queue) for consumption by the processor. The listener runs in a dedicated daemon thread, ensuring that blocking I/O on the network interface does not stall the detection pipeline.

5.2 processor.py

The processor module retrieves events from the listener queue and applies the two-layer detection strategy described in Section 3.2. Feature engineering transforms the raw event dictionary into a fixed-length numerical feature vector compatible with the XGBoost model's expected input format. Features include encoded IP octets, protocol identifier, port numbers (source and destination), normalised packet size, and derived features such as port range classifications (well-known ports 0-1023, registered ports 1024-49151, dynamic ports 49152-65535). The XGBoost model is loaded from a serialised file (model.xgb) at module initialisation and is not re-loaded on each inference call, ensuring low per-event latency. Post-classification, the rule engine evaluates the same event against its heuristic library. If either layer produces a positive classification, the processor calls the alert generator function.

5.3 supabase_client.py

This module initialises the Supabase client using the project URL and API key retrieved from environment variables. It exposes a single public function, insert_alert(alert_dict), which performs a validated INSERT operation into the ids_alerts table. Prior to insertion, the function validates that all mandatory fields (src_ip, dest_ip, attack_type, severity, reason, timestamp, method) are present and non-null, raising a ValueError on validation failure to prevent corrupt records from reaching the database. A retry decorator (with exponential backoff) wraps the INSERT call to handle transient network failures gracefully.

5.4 main.py

The execution engine initialises all modules in the correct dependency order, starts the listener thread, and enters the main

event loop. The event loop dequeues events from the listener queue at a configurable polling interval (default 50 ms), submits each event to the processor, and collects the resulting alerts. Alerts are logged locally to a rotating file handler (for offline audit purposes) and simultaneously forwarded to `supabase_client.insert_alert()`. Graceful shutdown is handled by a SIGINT/SIGTERM signal handler that drains the queue, flushes logs, and terminates all threads before exiting.

5.5 app.py

The Streamlit dashboard application uses `st.session_state` to maintain user filter preferences (severity level, attack type, time range) across page reloads. On each script re-run (triggered by Streamlit's periodic refresh mechanism or by user interaction), the application queries Supabase for alerts matching the current filter criteria, constructs a pandas DataFrame from the results, and renders the configured visualisation components. Colour coding is applied to severity labels using conditional DataFrame styling, with CRITICAL alerts rendered in red, HIGH in orange, MEDIUM in yellow, and LOW in green.

VI.DATABASE SCHEMA AND DATA FLOW

6.1 Schema Definition

All alert records are persisted in a single PostgreSQL table named `ids_alerts` hosted on Supabase. The schema is defined as follows: the primary key is an auto-incrementing integer column (`id`); `src_ip` and `dest_ip` store source and destination IPv4 addresses as variable-length text; `attack_type` stores the classified threat category; `severity` stores the risk level enumeration; `reason` stores a human-readable explanation of the detection rationale; `timestamp` stores the UTC timestamp of alert generation; and `method` stores the detection method identifier (ML or RULE) to enable post-hoc analysis of which detection layer was responsible.

src_ip	dest_ip	attack_type	severity	method	timestamp	reason
192.168.1.10	10.0.0.5	Port Scan	HIGH	TCP	2024-05-01 10:01	Multiple SYN packets detected to sequential ports
172.16.0.22	8.8.8.8	DDoS	CRITICAL	UDP	2024-05-01 10:03	Packet flood exceeds threshold (5000 pps)
10.0.0.15	192.168.2.20	Brute Force	HIGH	SSH	2024-05-01 10:07	75 failed login attempts in 60 seconds
192.168.3.5	10.10.0.1	SQL Injection	CRITICAL	HTTP	2024-05-01 10:12	Malformed SQL payload in GET request
10.0.0.9	192.168.1.1	Normal	LOW	HTTP	2024-05-01 10:15	Routine browser request

Table 3: Sample Alert Records from `ids_alerts`

6.2 Data Flow

The data flow through the system follows a strict unidirectional pipeline. Raw network packets are captured by the Listener and placed onto the in-memory event queue. The Processor dequeues events, applies ML and rule-based classification, and generates alert dictionaries. The `supabase_client` module serialises and persists each alert to the cloud database. The Streamlit dashboard independently queries the database, decoupled from the detection pipeline, ensuring that dashboard performance does not affect detection latency. This architectural separation of concerns allows each component to be independently scaled: in a high-throughput deployment, multiple Listener-Processor pairs can write to the same Supabase instance while multiple dashboard instances serve different analyst teams.

VII.SECURITY DESIGN

Security-by-design principles are embedded throughout the CloudShield IDS implementation. Credential management is handled exclusively through environment variables loaded via the `python-dotenv` library, ensuring that Supabase project URLs and API keys are never hardcoded in source files or committed to version control repositories. The `.env` file is added to `.gitignore` by default as part of the project scaffold.

Database access is governed by Supabase's row-level security (RLS) policies. The service-role key used by the detection pipeline has INSERT and SELECT privileges on the `ids_alerts` table but no UPDATE or DELETE privileges, enforcing the principle of least privilege and preventing tampering with the immutable audit log. Dashboard-facing queries use a restricted anon key with SELECT-only access.

Network communication between the detection pipeline and Supabase is conducted exclusively over TLS 1.3, encrypting alert data in transit. The Streamlit dashboard, when deployed behind a reverse proxy such as Nginx or Caddy, can be served over HTTPS with appropriate certificate configuration. Local deployments are protected by binding the Streamlit server to the loopback interface (127.0.0.1) rather than all interfaces (0.0.0.0), preventing unauthorised external access.

Input validation in `supabase_client.py` guards against malformed alert records that could trigger unexpected database behaviour. IP address fields are validated against a compiled regular expression for IPv4 format compliance before insertion. Severity values are validated against the permitted enumeration set {CRITICAL, HIGH, MEDIUM, LOW} to prevent injection of arbitrary values into the severity column.

VIII.RESULTS AND PERFORMANCE EVALUATION

8.1 Experimental Setup

The CloudShield IDS pipeline was evaluated on a controlled laboratory network consisting of one host running the detection pipeline (Intel Core i7-12700H, 16 GB RAM, Python 3.11 on Ubuntu 22.04 LTS) and a second host generating synthetic attack traffic using tools including hping3 (port scan and DDoS traffic generation), Hydra (brute-force simulation), and OWASP ZAP (SQL injection payload generation). Normal background traffic was generated using iperf3 and a web browser session against an Apache server. The XGBoost model was pre-trained on a stratified 70/30 train-test split of a labelled dataset derived from the CICIDS 2017 network traces, augmented with locally captured traffic samples.

8.2 Detection Performance

Table 4 reports the precision, recall, F1-score, and detection rate achieved by CloudShield IDS across the five traffic categories evaluated in the experiment.

Attack Type	Precision (%)	Recall (%)	F1-Score (%)	Detection Rate (%)
Port Scan	96.2	94.8	95.5	97.1
DDoS	98.1	97.3	97.7	98.6
Brute Force	95.4	93.7	94.5	96.0
SQL Injection	97.0	96.1	96.5	97.8
Normal Traffic	99.2	99.0	99.1	99.4
Overall	97.2	96.2	96.7	97.8

Table 4: CloudShield IDS Detection Performance by Attack Category

The system achieves an overall detection rate of 97.8% and an overall precision of 97.2%. The DDoS category achieves the highest individual detection rate (98.6%), attributable to the volumetric nature of DDoS attacks which produces highly discriminative rate-based features that both the XGBoost model and the rule engine detect with high confidence. Port Scan exhibits the lowest individual precision (96.2%), primarily because legitimate network discovery tools such as network management agents generate SYN packets at rates that occasionally exceed the scanning threshold, producing false positives. The overall false positive rate across all categories is approximately 2.8%, which compares favourably with published false positive rates of 8–12% for Snort and 6–10% for Suricata on similar traffic mixes.

8.3 Latency Analysis

End-to-end alert latency—measured from packet arrival at the listener to alert insertion in the Supabase database—was benchmarked across 10,000 events under laboratory conditions. The mean latency was 43.7 ms with a standard deviation of 8.2 ms and a 99th percentile latency of 112 ms. The dominant latency component was the Supabase round-trip time (mean 38.1 ms), with the ML inference step contributing a mean of 1.4 ms per event. These figures are consistent with real-time IDS deployment requirements in which alert latency below 500 ms is generally considered operationally acceptable.

8.4 Throughput

The detection pipeline sustained a throughput of 12,400 events per second in single-threaded operation on the evaluation hardware, well in excess of the 8,000 events per second typically observed on a 1 Gbps corporate LAN under peak load conditions. Multi-threaded scaling (using Python's ThreadPoolExecutor with four worker threads) increased sustained throughput to 38,200 events per second, sufficient for 10 Gbps network monitoring when coupled with hardware packet capture acceleration (such as PF_RING or DPDK).

IX.COMPARATIVE ANALYSIS

Table 5 compares CloudShield IDS with representative existing intrusion detection systems across key dimensions including detection method, cloud integration, dashboard availability, false positive rate, alert explanation depth, and deployment flexibility.

Feature	CloudShield IDS	Snort	Suricata	ML-IDS (Traditional)
Detection Method	ML + Rule-Based	Rule-Based	Rule-Based	ML Only
Real-Time Processing	Yes	Yes	Yes	Limited

Feature	CloudShield IDS	Snort	Suricata	ML-IDS (Traditional)
Cloud Integration	Supabase (Native)	None	None	Varies
Dashboard	Streamlit (Web)	3rd Party Only	3rd Party Only	Custom
XGBoost Model	Yes	No	No	Varies
False Positive Rate	~2.8%	~8–12%	~6–10%	~4–7%
Alert Explanation	Detailed (Reason)	Minimal	Minimal	Limited
Deployment	Local / Cloud	Local	Local	Local

Table 5: Comparative Analysis of CloudShield IDS vs. Existing Systems

CloudShield IDS offers a distinctive combination of capabilities that no existing open-source IDS fully addresses. Snort and Suricata are mature, battle-tested systems with extensive rule repositories and strong community support; however, they lack native ML-based detection, native cloud integration, and built-in interactive dashboards. Traditional ML-based IDS research prototypes typically demonstrate strong algorithmic performance on benchmark datasets but do not address operational deployment concerns such as cloud persistence, real-time dashboard visualisation, or security-conscious credential management. CloudShield IDS synthesises these dimensions into a single, coherent system that is deployable without significant infrastructure engineering effort.

X. CHALLENGES AND RESOLUTIONS

The development of CloudShield IDS encountered several non-trivial engineering challenges. Table 6 documents the principal challenges, their root causes, and the resolutions applied.

Challenge	Root Cause	Resolution Applied
Schema mismatch on Supabase INSERT	Locally generated fields not matching DB schema	Aligned alert dict keys with PostgreSQL column definitions
XGBoost import errors at runtime	Missing compiled dependencies on fresh env	Added xgboost to requirements and re-validated model loading
Streamlit UI data display issues	Async data refresh conflicts with session state	Implemented <code>st.session_state</code> with explicit refresh triggers
pandas StylerWarning on DataFrames	Deprecated styling API in newer pandas versions	Migrated to updated <code>DataFrame.style.map()</code> API
Duplicate alerts in DB	Re-processing of cached network events	Added deduplication logic using composite event hash

Table 6: Challenges Encountered and Resolutions Applied

Beyond the tabulated challenges, the integration of Claude AI into the development workflow itself introduced novel considerations. Prompts for code generation required iterative refinement to produce outputs of sufficient specificity; vague prompts generated syntactically correct but logically inappropriate code. Effective use of Claude AI for debugging required providing the full stack trace and relevant module context, rather than isolated error messages. These observations align with emerging prompt engineering best practices and suggest that structured, context-rich prompts are essential for productive AI-assisted software development in complex multi-module systems.

Future Enhancements

Cloud Shield IDS is designed as a foundation for continued research and engineering development. The following enhancements are planned for subsequent iterations of the system:

- **Geo-IP Enrichment and Threat Intelligence Integration:** Source IP addresses in alerts will be enriched with geographic location data using the MaxMind GeoIP2 database, enabling choropleth map visualisations in the dashboard and correlation of attack sources with known threat intelligence feeds such as AbuseIPDB.
- **Real-Time Alerting and Notification:** Integration with messaging platforms (Slack, Microsoft Teams, PagerDuty) via webhook APIs will enable push notifications for CRITICAL and HIGH severity alerts, allowing security teams to respond to threats without continuously monitoring the dashboard.
- **Heatmap Visualisation:** A time-of-day vs. attack-type heatmap will be added to the dashboard to expose temporal patterns in attack activity, enabling proactive resource allocation by security operations teams.

- **Online Model Retraining:** The XGBoost model will be extended with an online learning capability that incrementally retrains on newly labelled alert data, enabling the system to adapt to evolving attack patterns without full model retraining cycles.
- **Container-Based Deployment:** A Docker Compose configuration will be developed to package the Listener, Processor, and Dashboard components into isolated containers, enabling one-command deployment on any container-capable host and simplifying horizontal scaling.
- **Explainability Layer:** SHAP (SHapley Additive exPlanations) values will be computed for each XGBoost classification and surfaced in the dashboard, providing per-alert feature attribution that supports analyst investigation and model audit.
- **Federated Detection:** A federated architecture will be explored in which multiple CloudShield IDS nodes share anonymised model gradients to collaboratively improve detection accuracy without centralising raw traffic data, addressing privacy constraints in multi-tenant environments.

XI. CONCLUSION

This paper has presented CloudShield IDS, a hybrid, cloud-integrated intrusion detection system that combines XGBoost-based machine learning classification with deterministic rule-based heuristics, Supabase-backed persistent storage, and a real-time Streamlit dashboard. The system addresses a well-documented gap in the IDS literature by providing a complete, end-to-end implementation that spans from packet capture to visualisation, rather than focusing exclusively on algorithmic performance on offline benchmark datasets.

Experimental evaluation demonstrates an overall detection rate of 97.8% and an overall precision of 97.2% across five traffic categories (Port Scan, DDoS, Brute Force, SQL Injection, and Normal), with a false positive rate of approximately 2.8%. These figures compare favourably with established rule-based systems such as Snort and Suricata, while the integrated cloud persistence and interactive dashboard provide operational capabilities that are absent from algorithmic research prototypes. End-to-end alert latency of 43.7 ms mean and a sustained throughput of 12,400 events per second in single-threaded operation confirm the system's suitability for real-world deployment on networks of moderate to high traffic volume.

The systematic integration of Claude AI throughout the development lifecycle—spanning architecture design, code generation, debugging, and documentation—contributed meaningfully to development velocity and code quality. This experience supports the emerging consensus that large language models, when engaged through structured, context-rich prompting, constitute effective collaborative partners in complex software engineering projects.

Cloud Shield IDS is positioned as a scalable, extensible reference architecture for ML-augmented network intrusion detection. Its modular design accommodates independent enhancement of each pipeline stage, and the roadmap of future enhancements—including geo-IP enrichment, online model retraining, federated detection, and SHAP-based explainability—defines a clear trajectory toward a production-grade, enterprise-ready cybersecurity monitoring platform.

REFERENCES

1. Bace, R., & Mell, P. (2001). NIST Special Publication on Intrusion Detection Systems (SP 800-31). National Institute of Standards and Technology.
2. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.
3. Cisco Talos (2024). Snort: Open Source Intrusion Prevention System. Retrieved from <https://www.snort.org/>
4. IBM Security (2023). Cost of a Data Breach Report 2023. IBM Corporation.
5. Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A Deep Learning Approach for Network Intrusion Detection System. Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies.
6. Kim, J., Kim, J., Thu, H. L. T., & Kim, H. (2016). Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. 2016 International Conference on Platform Technology and Service, 1–5.
7. Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion Detection System: A Comprehensive Review. Journal of Network and Computer Applications, 36(1), 16–24.
8. Open Information Security Foundation (2024). Suricata Open Source IDS/IPS. Retrieved from <https://suricata.io/>
9. Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. Proceedings of USENIX LISA, 229–238.
10. Sallay, H., Al-Shalfan, K., & Otaibi, M. (2011). A Scalable Distributed IDS Architecture for High-Speed Networks. International Journal of Computer Networks, 3(2), 87–96.
11. Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94.
12. Supabase Inc. (2024). Supabase: The Open Source Firebase Alternative. Retrieved from <https://supabase.com/>
13. Tavallae, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. IEEE Symposium on Computational Intelligence for Security and Defense Applications, 1–6.
14. Zhang, H., Li, J. L., Liu, X. M., & Dong, C. (2019). Multi-Dimensional Feature Fusion and Stacking Ensemble Mechanism for Network Intrusion Detection. Future Generation Computer Systems, 122, 130–143.
15. Anthropic (2024). Claude AI: Large Language Model for Code Generation and Engineering Assistance. Retrieved from <https://www.anthropic.com/>