أكاديمية كاوست
KAUST ACADEMY

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

AI-Powered Honeypot for Malware Detection at the Network Edge

Layan Alqarni, Raneem Alhajri, Retaj Alghamdi, Reef Alshehri, Prof. Insaf Ullah

layan.alqarni@kaust.edu.sa, raneem.alhajri@kaust.edu.sa, retaj.alghamdi@kaust.edu.sa, reef.alshehri@kaust.edu.sa, insaf.ullah@essex.ac.uk

Abstract

We present **MIRSAD**, an AI-based edge malware detection framework for detecting and containing malicious files that are captured via a Cowrie honeypot. While traditional signature-based methods work against known malware, they lack the ability to detect new ones, making a layered defense with efficient detection and machine learning more attractive.

MIRSAD implements a cascading pipeline that consists of SHA-256 hash matching, YARA signature scanning, and CatBoost-based malware classification with EMBER2024 static features. All malicious files are automatically quarantined and reported, and the evidence of detection is submitted asynchronously to the administrator-side Qwen2.5-3B-Instruct LLM for analysis, confidence, supporting evidence, and recommendations, without impacting edge containment.

The CatBoost model achieved 95.83% accuracy, 95.50% precision, 95.98% recall, 95.74% F1-score, and 99.48% ROC-AUC on the EMBER2024 test set. In all, MIRSAD demonstrates that signature-based detection, machine learning, automated containment, and LLM-assisted analysis can be combined into a viable, edge-based malware defense solution.

Date: August 23, 2026

Code: https://github.com/MIRSAD-Security/AI-Powered-Honeypot-for-Malware-Detection.git

Project Page: https://mirsad-security.github.io/AI-Powered-Honeypot-for-Malware-Detection/

Contents

1 Introduction **2**

1.1 Problem Motivation and Real-World Impact . . . 2

1.2 Problem Statement . . . 2

1.3 Key Assumptions and Limitations of the Problem . . . 2

1.4 **Summary of the Proposed System** . . . 2

2 Related Work **3**

2.1 A Survey of Contemporary Open-Source Honeypots, Frameworks, and Tools (2023) . . . 3

2.2 A Systematic Review of Honeypot Data Collection, Threat Intelligence Platforms, and AI/ML Techniques (2025) . . . 3

2.3 Q-Cowrie: An Adaptive Honeypot to Analyse Attackers' Behaviour (2026) . . . 3

2.4 AI-Powered Cybersecurity in Edge Computing: Lightweight Neural Models for Anomaly Detection (2024) . . . 4

2.5 HoneyTrap: AI-Powered Honeypot for Cyberattack Detection (2026) . . . 4

3 Methodology **5**

3.1 Overall System Architecture . . . 5

3.2 Signature-Based Detection . . . 6

3.3 AI-Based Malware Detection . . . 8

3.4 LLM-Based Malware Analysis . . . 10

3.5 Storage and Containment . . . 10

3.6 Dashboard and Monitoring . . . 13

3.7 System Integration Workflow . . . 16

4 Results **16**

4.1 Signature-Based Detection Results . . . 16

4.2 AI Malware Detection Performance . . . 17

4.3 Confusion Matrix Analysis . . . 17

4.4 LLM-Based Malware Analysis Results . . . 18

4.5 End-to-End System Results . . . 19

5 Discussion **19**

6 Limitations and Future Work **20**

7 Conclusion **21**

References **21**

1 Introduction

1.1 Problem Motivation and Real-World Impact

Despite the passage of time, malware is still a formidable threat, constantly evolving with methods like code modification, obfuscation, and packing. Signature-based detection is effective and can identify known threats quickly and efficiently, but may not be effective for previously undetected or altered malware. Additional insight can be gained from more detailed behavioral analysis, but this may require dedicated environments for analysis and significant computational power.

Distributed security monitoring is a growing trend in edge computing, and it requires a malware detection method that can analyze the malware in a timely manner and is suitable for resource-constrained environments. In this context, honeypots can be used to catch suspicious files at the edges of the network, before they can impact critical systems.

1.2 Problem Statement

This project focuses on detecting and analyzing malware samples collected in resource-constrained edge environments. Signature-based detection may primarily identify known threats and may fail to detect modified or previously unseen malware, and analysis-based technologies might not be feasible at the edge.

Therefore, the project focuses on creating a lightweight, signature-based malware detection framework as well as an AI-based malware detection framework, including evidence-driven analysis. The aim of the framework is to provide not only the malicious or benign assessment, but also in-depth information that can aid the interpretation of suspicious files.

1.3 Key Assumptions and Limitations of the Problem

It is assumed that a network honeypot can be used to capture suspicious files and that the available evidence may vary depending on the characteristics of each captured file. The framework is primarily based on the collected file and detection evidence and not on the resource-heavy edge-based dynamic analysis.

Being able to identify a specific malware family is dependent on the quality and specificity of the evidence available. Thus, a sample can be classified as malicious without enough evidence to accurately say what family it belongs to. These constraints are used to set the boundaries of the proposed framework and its analysis capabilities.

1.4 Summary of the Proposed System

To deal with the problems, the project suggests a lightweight malware detection and analysis framework to be deployed around an edge-based Cowrie honeypot. Complementary detection techniques are employed that include SHA-256 hash matching, YARA rule matching, and AI malware classification through an offline-trained model.

The evidence is then passed to an administrator-side analysis component that carries out analysis and reasoning by an LLM to deliver an explainable analysis of the malware, such as a potential malware family, a confidence score, supporting evidence, reasoning, and a defensive recommendation. Detection and analysis information is stored and presented on a monitoring dashboard, and malicious files can be automatically contained at the edge.

2 Related Work

We analyzed some recent research papers on honeypots, edge computing, lightweight AI, malware detection, and threat intelligence. The goal was to find out what current approaches are in use, what the existing gaps in research are, and to further improve the design of our proposed AI-based malware detection framework for edge environments.

2.1 A Survey of Contemporary Open-Source Honeypots, Frameworks, and Tools (2023)

This paper gives a detailed survey of the contemporary open-source honeypots, their architectures, deployment strategies, logging, and supported protocols. It introduces various honeypot implementations like Cowrie and Dionaea and touches on some challenges like attacker detection and honeypot fingerprinting. The survey also covers cloud-native deployment with Docker and Kubernetes [1].

Key contribution to our project:

- Assisted us in learning about various types of honeypots.
- Helped to choose Cowrie as the base honeypot.
- Provided insights into how Cowrie is able to detect attacker activities and malware downloads.

Limitation:

- No AI integration.
- No model for malware detection.
- No real-time behavioural analysis.

2.2 A Systematic Review of Honeypot Data Collection, Threat Intelligence Platforms, and AI/ML Techniques (2025)

This systematic review investigates the potential for combining a honeypot with a threat intelligence platform (TIP), like MISP and STIX/TAXII, and reviews AI and machine learning techniques for cybersecurity applications. The paper explores behavioral analysis, malware detection, anomaly detection, and how LLMs can be used in cyber defense [2].

Key contribution to our project:

- Introduced new threat intelligence frameworks.
- Assisted in defining appropriate AI methods and threat intelligence principles that can aid malware analysis at the edge.
- Made helpful references to MITRE ATT&CK and IoC extraction.

Limitation:

- Review paper only.
- No implementation or practical architecture.
- Does not propose a complete detection pipeline.

2.3 Q-Cowrie: An Adaptive Honeypot to Analyse Attackers' Behaviour (2026)

This paper is an extension of Cowrie, which combines a Reinforcement Learning (RL) algorithm (Q-Learning) with a Markov Decision Process (MDP). As time goes on, the adaptive honeypot evolves the way it responds to the attacker, and it keeps changing its responses accordingly to keep the deception intact. In addition, the study correlates attacker activities with the MITRE ATT&CK framework [3].

Key contribution to our project:

- Showcased the possibility of integrating AI into Cowrie.
- Demonstrated potential future directions for the development of intelligent honeypots.
- Sparked ideas for future improvements that are not covered in this project.

Limitation:

- Operates primarily in an adaptive fashion.
- Concentrates on adaptive honeypots rather than lightweight malware detection at the network edge.
- Does not include LLM-based incident analysis.

2.4 AI-Powered Cybersecurity in Edge Computing: Lightweight Neural Models for Anomaly Detection (2024)

In this paper, the authors explore the possibilities of deploying lightweight AI models directly on edge devices to achieve real-time cybersecurity. It covers the concept of resource-efficient neural models, on-device inference, low latency, and the necessity of keeping detection performance while reducing the computational requirements in a distributed edge environment [4].

Key contribution to our project:

- Justified deploying AI directly on the edge device.
- Emphasized lightweight inference over cloud analysis.
- Helped us in choosing the AI model to detect malware in real-time, favoring an edge-friendly approach.

Limitation:

- Concentrates primarily on the detection of anomalies, not malware file analysis.
- Does not include honeypots.
- There is no working malware collection pipeline with Cowrie.

2.5 HoneyTrap: AI-Powered Honeypot for Cyberattack Detection (2026)

This paper presents HoneyTrap, an artificial intelligence (AI) powered honeypot that combines multiple deception services and machine learning threat detection. The system gets the Bot Probability Score by using Isolation Forest, offers a real-time dashboard, and automatically blocks malicious IP addresses through a Kill Switch mechanism [5].

Key contribution to our project:

- Showed AI-based real-time detection.
- Added autonomous threat containment.
- Showed integration of dashboards with machine learning.

Limitation:

- Does not use behavioral AI integrated within the honeypot.
- Mainly bot-focused.
- Does not incorporate LLM-based incident explanation or threat-intelligence generation.

Research Paper	Honeypot	Edge	Lightweight AI	Real-Time	Malware Detection
A Survey of Contemporary Open-Source Honeypots (2023)	✓	×	×	×	×
A Systematic Review of Honeypot Data Collection (2025)	✓	×	✓	×	✓
Q-Cowrie: An Adaptive Honeypot (2026)	✓	×	✓	×	×
AI-Powered Cybersecurity in Edge Computing (2024)	×	✓	✓	✓	×
HoneyTrap: AI-Powered Honeypot (2026)	✓	×	✓	✓	×
AI-Powered Honeypot for Malware Detection at the Network Edge	✓	✓	✓	✓	✓

Table 1 Comparative Analysis of Recent Research on Honeypot Frameworks and AI Integration

3 Methodology

3.1 Overall System Architecture

The proposed framework is a lightweight edge-based malware detection and analysis system. The Cowrie honeypot at the edge gathers suspicious or anomalous files and their associated file and source metadata. The captured files undergo a series of sequential detection steps: SHA-256 hash matching, YARA rule matching, and AI static feature malware classification.

The detection evidence is then passed to an analysis component on the administrator side for additional analysis. An administrator-local LLM analyses the evidence available and generates an analysis result which complements the initial detection results.

Results of detection and analyses are saved in the system database and displayed on the monitoring dashboard. Additionally, when malicious content is detected, it can be automatically quarantined at the edge, enabling the framework to support early detection, analysis, monitoring, and containment while offloading the computationally intensive LLM analysis from the edge device. (see Figure 1).

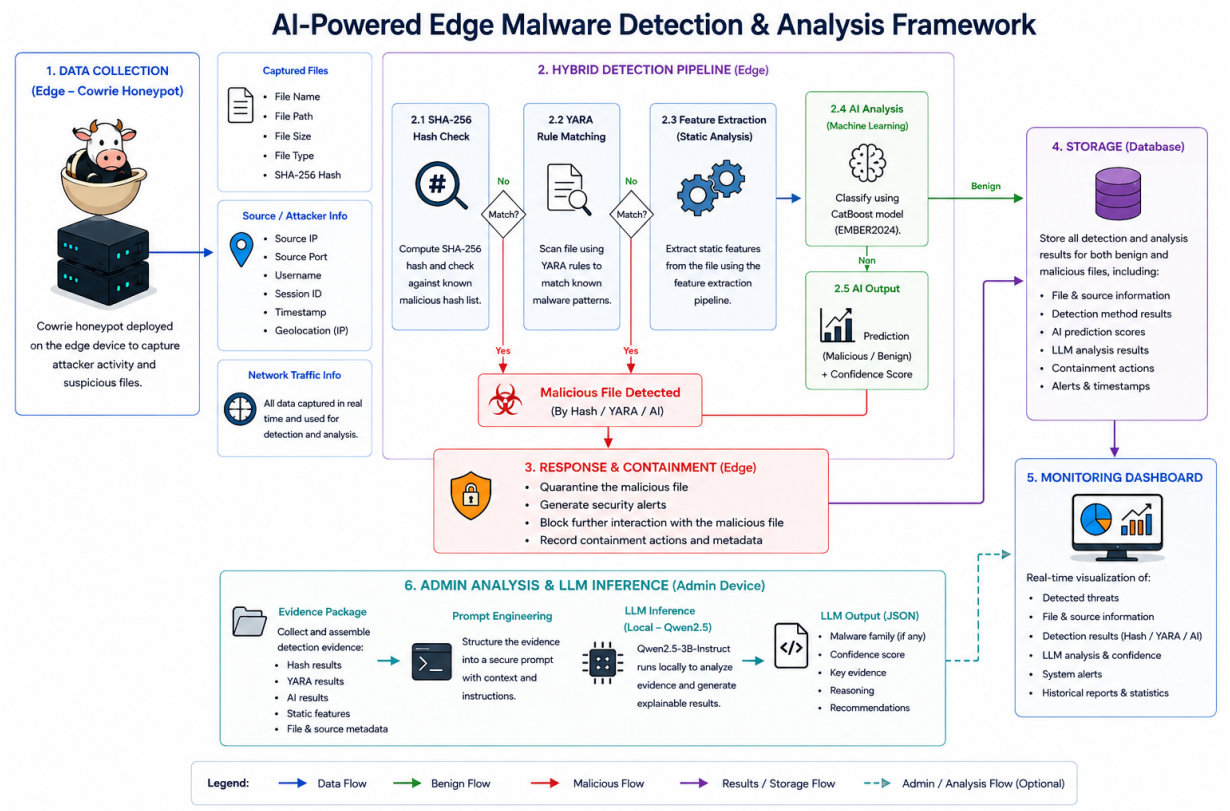


Figure 1 System Architecture Overview

3.2 Signature-Based Detection

The initial step in the proposed malware detection pipeline is the signature-based detection module. It provides a lightweight and computationally efficient mechanism for identifying previously known malware before invoking the AI-based detection module. This layer combines two complementary techniques: SHA-256 hash lookup and YARA rule matching.

When Cowrie captures a downloaded or uploaded file, the framework retrieves both the captured artifact and its associated metadata, including the source IP address, capture timestamp, file path, and SHA-256 hash. Cowrie preserves files transferred through mechanisms such as `wget`, `curl`, SFTP, and SCP, allowing all captured artifacts to be analyzed by the subsequent detection stages [6].

The signature-based layer follows a short-circuit detection workflow to improve detection efficiency. If a file is identified as malicious during the SHA-256 hash lookup stage, the detection process terminates immediately without executing the more computationally expensive YARA and AI stages. If no hash match is found, the file proceeds to YARA scanning. Only files that remain unidentified after both signature-based stages are forwarded to the AI-based detection module. This layered architecture minimizes unnecessary model inference while improving the overall efficiency of malware detection at the network edge.

3.2.1 SHA-256 Hash Lookup

The first signature-based stage compares the SHA-256 hash of every captured file against a local threat-intelligence database populated using hash information obtained from MalwareBazaar. MalwareBazaar is a malware sample exchange and threat-intelligence platform that supports

searching and querying malware samples using cryptographic hashes, including SHA-256 [7].

The imported hash information is stored locally in the `known_hashes` table of the SQLite database. Each record contains the SHA-256 hash and, when available, the associated malware family, source name, and insertion timestamp. Maintaining a local hash database enables offline operation, eliminates the need to query an external API for every captured file, and provides rapid exact-match lookups suitable for edge deployment.

Before querying the database, the framework normalizes each SHA-256 value by converting it to lowercase. The SHA-256 value recorded by Cowrie is then verified against the hash calculated directly from the captured artifact. This integrity check ensures that the analyzed file is identical to the file originally captured by the honeypot.

If an exact hash match is identified, the file is classified as malicious. A fixed detection score of 1.0 is assigned to indicate a deterministic signature match rather than a probabilistic confidence value. The matched record, together with the associated metadata, is stored as part of the detection result. The file does not proceed to the YARA or AI detection stages, while the relevant hash evidence is forwarded to the administrator-side LLM for further malware analysis. If no hash match is found, the file is classified as unknown at the hash stage and forwarded to the YARA detection module.

Although MalwareBazaar provides an extensive source of malware threat intelligence, entries in community-maintained repositories may occasionally contain incomplete or disputed labels. Therefore, a successful hash match is treated as a strong signature-based indicator rather than a replacement for continuous threat-intelligence validation and database maintenance.

3.2.2 YARA Signature Detection

Files that are not identified through SHA-256 lookup are processed by the YARA detection stage. YARA is widely used by malware researchers to identify and classify malicious files through rules containing textual patterns, binary signatures, and Boolean conditions [8]. During scanning, the compiled rule collection is applied to each captured artifact, and the identifiers of all matching rules are recorded.

The implementation uses a curated rule collection obtained from two well-established repositories: the Neo23x0 Signature-Base repository and the ReversingLabs YARA Rules repository [9, 10]. Signature-Base provides structured YARA signatures and indicators of compromise (IOCs) specifically designed for malware detection while emphasizing consistent rule formatting and reducing false positives [9].

Rather than loading every downloaded rule directly into the system, the project performs a complete rule-curation and validation process. Each YARA file is compiled individually before being accepted into the production rule set. Invalid or incompatible rules are separated from the curated collection, while validation reports record the compilation status and source of every rule file. Duplicate rule names are also checked because duplicate identifiers prevent the combined rule set from compiling successfully.

The final curated collection contains 1,047 successfully compiled YARA files comprising a total of 6,542 YARA rule declarations. Among these, 5,302 rule declarations originate from the Neo23x0 Signature-Base repository, while 1,240 originate from the ReversingLabs collection. The validation process confirmed that the final rule set contains no duplicate rule names. At system

startup, the curated rule directory is compiled, and the number of successfully loaded rule files is reported.

For each captured artifact, the scanner applies the compiled YARA rules and records all matching rules. If one or more rules match, the file is classified as malicious, and a fixed detection score of 1.0 is assigned to indicate a rule-based match rather than a probabilistic confidence value. The identifiers of the matched rules are stored as part of the detection details. The file does not proceed to the AI detection stage. The detection result is forwarded to the storage and containment components, while the matched YARA evidence is sent to the administrator-side LLM for further malware analysis. If no rule matches, the YARA stage classifies the file as unknown and forwards it to the AI-based malware detection module.

3.2.3 Signature-Based Workflow and Validation

The complete signature-based workflow consists of the following sequential steps:

1. Retrieve each newly captured artifact and its associated metadata from Cowrie.
2. Verify that the captured artifact exists and validate its SHA-256 hash.
3. Search the local MalwareBazaar-derived hash database for an exact match.
4. If an exact hash match is found, classify the file as malicious and skip the YARA and AI detection stages.
5. Otherwise, scan the file using the curated YARA rule collection.
6. If one or more YARA rules match, classify the file as malicious and skip the AI detection stage.
7. For malicious hash or YARA detections, forward the corresponding detection evidence to the administrator-side LLM for asynchronous malware analysis.
8. Forward files that remain unidentified after both signature-based stages to the AI-based malware detection module.

The signature-based detection module was validated at both the rule-set level and the complete pipeline level. All 1,047 curated YARA files compiled successfully, and the validation process confirmed that no duplicate rule identifiers were present. End-to-end testing demonstrated that files captured by Cowrie could be successfully retrieved, verified against the local MalwareBazaar hash database, scanned using the curated YARA rule collection, and automatically forwarded to the AI detection module whenever neither signature-based technique identified the file.

For malicious detections identified through SHA-256 or YARA, the corresponding detection evidence is forwarded asynchronously to the administrator-side LLM for further malware analysis. Detection and containment operations remain independent of LLM inference, ensuring that language-model processing does not delay the edge detection workflow. Detection results are stored through the shared storage interface, while the resulting LLM analysis is subsequently made available for storage and visualization through the monitoring dashboard.

This validation focused on rule compilation and end-to-end pipeline integration rather than a large-scale quantitative evaluation of signature-level detection accuracy.

3.3 AI-Based Malware Detection

The AI-based detection stage complements the signature-based detection stages to offer a machine learning technique to detect malicious files that are not detected by the previous signature-based

stages. The component is responsible for binary malware classification and has been trained offline using static feature representations of the malware.

3.3.1 Dataset Selection

The EMBER2024 dataset was selected as the primary dataset for developing the AI-based malware detection component because it provides a large collection of labeled benign and malicious samples suitable for supervised malware classification [11]. The dataset provides preprocessed static feature representations associated with malware labels, allowing the model to be trained using structured numerical data without requiring feature extraction from raw files during training.

The AI detection component was developed as a binary classifier to distinguish between benign and malicious files. The deployed feature extraction pipeline supports several executable file formats, including PE32, PE64, .NET, and ELF, allowing the trained model to be integrated into the broader honeypot detection workflow.

For model development and evaluation, a total of 607,321 samples were used. Of these, 455,000 samples were used for training, and 152,321 samples were reserved for testing. The test samples were not used during training and were used to evaluate the CatBoost classifier before integrating the trained model into the complete detection pipeline.

Subset	Samples	Percentage
Training	455,000	74.9%
Testing	152,321	25.1%
Total	607,321	100%

Table 2 Dataset split used for model training and evaluation.

3.3.2 Feature Representation

The samples of malware in EMBER2024 are identified by numerical features that represent static properties of the malware samples. These features include data about file structure, metadata, imported libraries, exported functions, strings, headers, and other static properties.

The machine learning model needs the data to be input in a consistent format, and the CatBoost classifier can work with structured data, so it's important to have a fixed numerical representation. In model development, the feature representations given were directly used for model training and testing.

The detection pipeline is used to extract the necessary static features from newly captured files and transform these features to a representation that is acceptable to the trained classifier. This means that an offline-trained model can be used during detection without the need to retrain the model or use the training data on the edge device.

3.3.3 CatBoost Model Development

For the AI-based part of the malware detection, the model was chosen to be CatBoost. The model was chosen as a practical model for the structured numerical representation of the features in the dataset [12].

The classifier was trained on the EMBER2024 samples and their labels. It classifies the samples as either *Malware* or *Benign* and provides a confidence score along with the classification.

The loaded model was used as an offline-trained classifier in the detection part. When the file is captured for the first time, the captured file is processed to obtain the required feature representation and then passed to the trained model to classify the captured file. The prediction and the confidence value are used in the subsequent analysis steps as evidence of the detection.

3.4 LLM-Based Malware Analysis

The LLM-based analysis component provides an additional analysis layer for files identified as malicious by the detection pipeline. The administrator-side component uses the Qwen2.5-3B-Instruct model [13] to interpret the structured evidence generated by the preceding detection stages. Rather than replacing the existing detection methods, the LLM analyzes the available evidence and static characteristics to generate a more informative assessment of suspicious files.

3.4.1 Evidence Preparation

The evidence presented to the LLM is built on the results of the detection stages. The evidence package can include detection results for hash matching, YARA analysis, AI-based classification, and extracted static file properties.

The evidence is prepared and passed to the LLM, which will be able to differentiate between the types of evidence and their respective functions. This enables the analysis to account for the result of the detection as well as any file attributes that supported the detection, instead of just one attribute.

3.4.2 Prompt Design

A template prompt was designed to ensure that the LLM consistently analyzes malware evidence. The prompt sets up the role of a malware analysis assistant and instructs it to use only the information provided as evidence when performing its assessment.

The prompt also establishes guidelines for distinguishing direct evidence from interpretation, avoiding unsupported claims, and assigning a confidence value based on the available evidence. No generic indicators or common system APIs are considered to be good evidence of a particular family of malware.

The model has the task of providing a formatted answer with the following components: Malware family assessment, confidence, reasoning, supporting evidence, defensive recommendations, and evidence-based limitations. The generated analysis is structured to be consistently processed and displayed across the rest of the system.

3.5 Storage and Containment

3.5.1 Database Selection

SQLite was selected as the local database management system on the edge device due to its lightweight architecture, low resource consumption, and direct integration with Python. Unlike client-server database systems, SQLite does not require a separate database service, which reduces deployment complexity and processing overhead on the edge device. It also provides reliable local persistence, transactional consistency, and offline operation, making it suitable for the resource-constrained environment of the proposed framework.

3.5.2 Database Schema

The database schema was designed to organize and maintain the data generated throughout the detection and containment workflow while preserving traceability between system components. Each captured file is assigned a unique `file_id`, and each detection result is assigned a corresponding `detection_id`. These identifiers are used to link file metadata with detection results, containment actions, alerts, and additional analysis records.

The schema consists of six relational tables: `files`, `detections`, `containment`, `alerts`, `known_hashes`, and `llm_analysis`. Foreign-key relationships are used to maintain data integrity and ensure that each security event can be traced back to the captured file and detection result that produced it. The resulting structure supports consistent storage, reduces unnecessary data duplication, and provides the data required by the detection, response, and administrative components of the system.

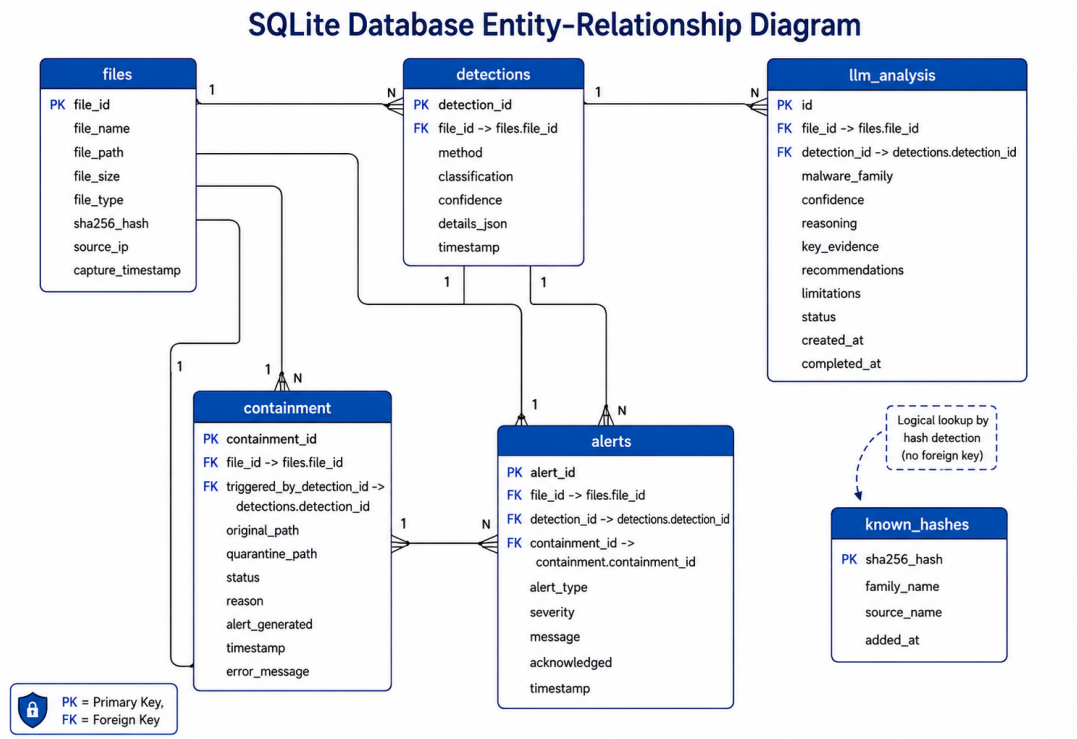


Figure 2 SQLite Database Entity–Relationship Diagram

Table	Purpose
`files`	Stores metadata for files captured by the Cowrie honeypot.
`detections`	Stores detection results produced by hash matching, YARA scanning, and AI classification.
`llm_analysis`	Stores structured LLM-assisted analysis associated with malicious detection results.
`containment`	Records quarantine operations, statuses, file paths, and processing errors.
`alerts`	Stores security alerts generated during containment operations.
`known_hashes`	Maintains a local list of known malicious SHA-256 hashes used for hash-based detection.

Table 3 Description of the SQLite Database Tables

3.5.3 Data Access Layer

A dedicated data-access layer was implemented to manage all database operations through a consistent set of repository functions. Instead of allowing individual system components to interact directly with SQLite, the storage module provides controlled interfaces for registering captured files, storing detection results, managing known hashes, recording containment actions and alerts, and storing or retrieving additional analysis results.

This design centralizes validation and database logic, reduces duplicated code, and improves maintainability across the system. It also ensures that records remain correctly linked through their corresponding file-id and detection-id, preserving data integrity and traceability throughout the detection and response workflow.

3.5.4 Quarantine Logic

The quarantine process is triggered only when a file has a valid malicious detection. Before containment, the system verifies that the detection belongs to the corresponding file and that the submitted file path matches the path originally registered for that capture. This validation prevents unrelated files from being quarantined and ensures that containment is applied only to confirmed malicious cases. Benign, unknown, and error classifications do not trigger quarantine.

3.5.5 File Containment Process

Once a file is confirmed as malicious, the system initiates a controlled containment procedure. The file is transferred from its original location to a dedicated quarantine directory, renamed with a `.quarantine` extension, and assigned restricted permissions to prevent further execution or accidental access. The containment status is recorded as `pending` before the operation and is updated to either `quarantined` or `failed` based on the outcome. The system also checks for previous containment records to avoid processing the same file more than once, while preserving the original path, quarantine path, timestamp, and associated detection result for traceability.

3.5.6 Alert Generation and Error Handling

The system generates security alerts based on the outcome of each containment operation. Successful quarantines produce high-severity alerts, while containment failures are escalated to critical severity to indicate that the threat may remain uncontained. Each alert is linked to the corresponding file and containment record and includes details such as alert type, severity, message, timestamp, and acknowledgement status. Runtime errors, such as missing files, invalid paths, or permission failures, are recorded with the related containment data rather than interrupting the pipeline, allowing administrators to review failed operations and their causes.

3.6 Dashboard and Monitoring

The dashboard serves as the primary administrative interface for monitoring and managing the Cowrie honeypot deployed at the network edge. It is engineered to deliver comprehensive, real-time observability across the entire multi-stage threat detection pipeline, encompassing Hash signature matching, YARA rule scanning, and CatBoost AI classification. The system meticulously tracks captured payloads from their initial network ingestion through to automated containment and LLM-driven forensic evaluation. Powered by a lightweight SQLite database and integrated via a FastAPI backend, the dashboard architecture is designed to minimize computational overhead. This design minimizes the computational overhead placed on the resource-constrained edge device while providing security administrators with a highly reactive and interactive threat-monitoring experience.

3.6.1 Dashboard Design

The user interface was engineered using the Streamlit tool framework, integrated with *Pandas* for high-throughput telemetry data manipulation, *Plotly* for interactive graphical representations, and a custom CSS design system to ensure responsive and clear visualization. The frontend communicates asynchronously with the FastAPI backend and utilizes Streamlit's native *@st.fragment* decorators to enable real-time, auto-refreshing data streams without requiring full-page reloads. The application follows a modular architecture, logically organized into specialized operational domains:

- **Core Dashboard:** Displays high-level Key Performance Indicators (KPIs), including total captured payloads, detection pipeline throughput, and automated quarantine rates, alongside interactive visual timelines and executable format distributions (see Figure 3).

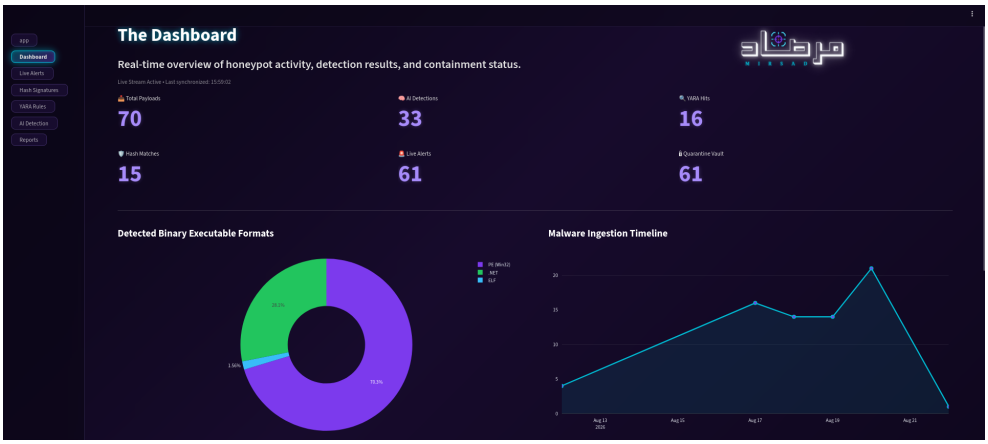


Figure 3 System Main Dashboard

- **Live Alerts:** A real-time interface providing immediate security alerts, detection methodologies, and incident acknowledgment controls, alongside an Admin Review tab for manually evaluating unclassified payloads (see Figure 4).

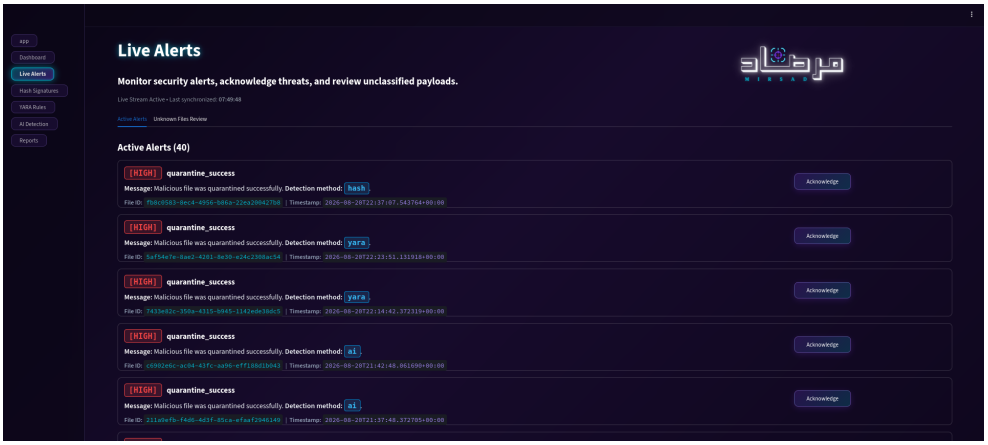


Figure 4 Part of the Live Alerts Page

- **Hash Signatures and YARA Rules:** Dedicated forensic interfaces displaying exact SHA-256 matches and triggered YARA signatures. These pages present comprehensive metadata for detected payloads, including file names, detection timestamps, and containment status, alongside deep visibility into malware, using an interactive Hex/ASCII payload viewer and quarantine storage paths. Additionally, they enrich the detection data with local LLM-generated forensic explanations. (see Figure 5, demonstrating the Hex section and YARA rule matching).

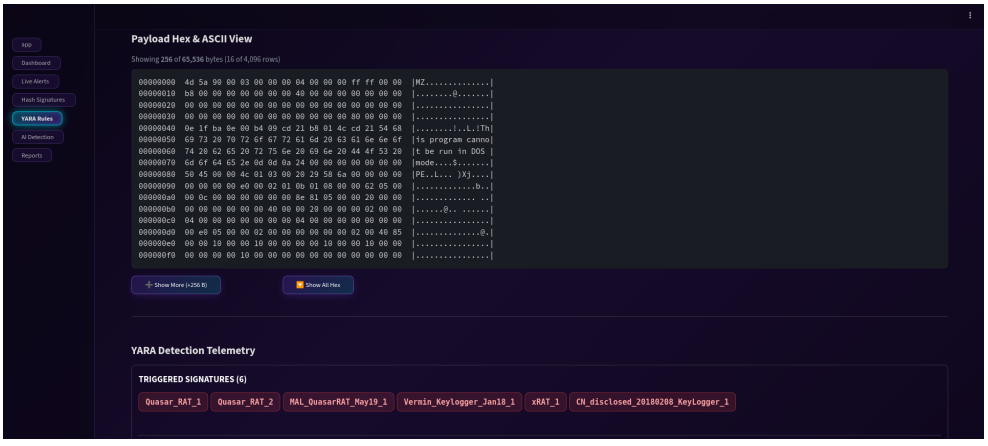


Figure 5 Part of the YARA Rules Page

- **AI Detection:** Showcases malicious classifications identified by the edge CatBoost model. Mirroring the core forensic layout of the signature pages, it presents complete payload metadata, ingestion and quarantine paths, and the interactive Hex/ASCII viewer. Uniquely, it details AI classification confidence scores and seamlessly integrates with the admin-hosted LLM to generate automated forensic assessments, reasoning, and defensive recommendations for complex or new threats (see Figure 6).

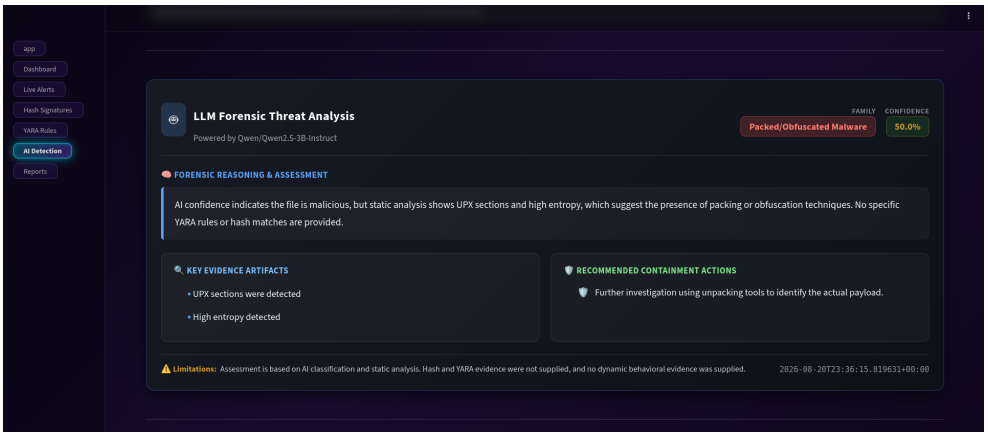


Figure 6 Part of the AI Detection Page

- **Reports:** An administrative control panel utilized for granular database queries, allowing administrators to select specific telemetry columns and export standardized CSV incident reports for external forensic auditing (see Figure 7, demonstrating the LLM analysis section).

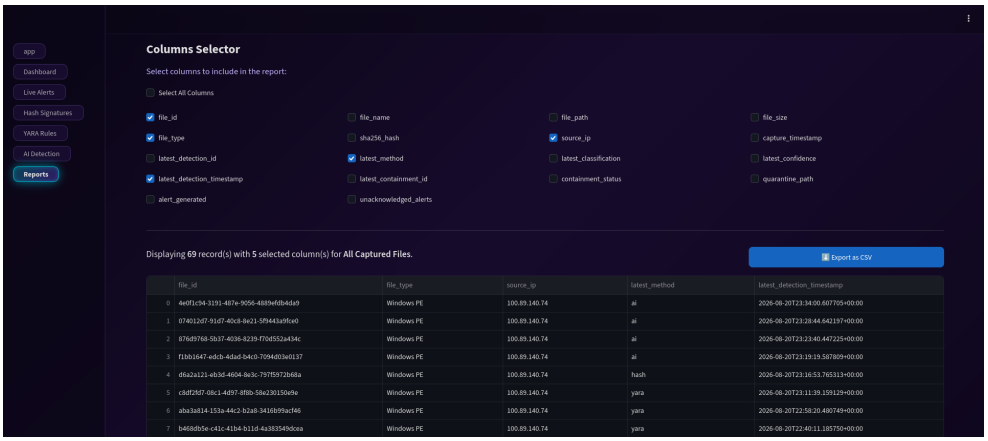


Figure 7 Part of the Reports Page

The web-based dashboard and the computationally intensive LLM are hosted on a dedicated external administrative machine. This decoupled architecture ensures a highly responsive management interface. At the same time, the honeypot and its complete detection pipeline (Hash, YARA, and CatBoost AI) operate efficiently on the edge node, thereby preventing resource overload on the edge device.

3.6.2 Real-Time Monitoring

Real-time monitoring relies on the continuous retrieval of detection data, including AI model predictions and signature matches, stored in the centralized database. By using a standardized data format, the system ensures that threat analysis results are immediately and accurately reflected in interactive charts, timelines, and Key Performance Indicators (KPIs). This real-time observability encompasses several strictly implemented operational aspects:

- Generating live security alerts for malicious payloads, explicitly identifying the detection method (Hash, YARA, or AI) and severity level (e.g., HIGH). The interface enables administrators to proactively acknowledge these alerts and features a dedicated "Unknown Files Review" tab to manually track payloads the system was unable to classify.

- Monitoring intrusion evidence in real-time by capturing critical metadata for every dropped file. This includes tracking the attacker's Source IP address, executable file formats, file size, and AI confidence scores, all seamlessly linked to automated LLM forensic assessments and active quarantine paths.

To achieve this, the dashboard utilizes an asynchronous updating mechanism (via Streamlit fragments). This provides live and continuous monitoring of system status and threats, dynamically refreshing telemetry data the moment it is processed by the intrusion trap without requiring full-page reloads.

3.7 System Integration Workflow

The implemented system is designed as a series of interconnected elements that interact with each other in both the edge and the administrator domains. The integration connects the Cowrie monitoring process, evidence construction, administrator API, LLM analysis, monitoring dashboard, storage layer, and containment mechanisms.

The Cowrie monitoring component detects newly captured artifacts at the edge and initiates the detection process. The detection engine is the primary orchestrator for the detection stages and is responsible for coordinating the hash matching using SHA-256, scanning with YARA, and classification using AI according to the sequential detection logic. Every detection stage is standardized, and the evidence is built separately for each stage for administrator-side analysis.

An HTTP-based API is used to implement the integration between the edge and administrator environments. If a detection yields evidence that can be analyzed, the admin client bundles up the file identifier, detection metadata, and evidence and sends it to the admin receiver API. When the request is received, the receiver validates the request and returns an immediate acknowledgement, triggering the edge detection and containment workflow to proceed without waiting for the LLM inference to finish.

On the administrator side, the evidence is correlated with the received file identifier and passed on to the local LLM analysis component. The generated analysis is then passed to the storage and monitoring components, and the resulting analysis can be related to the initial detection event and then displayed via the dashboard.

The design of this type of asynchronous communication allows the lightweight detection and containment work at the edge to be separated from the heavy work of LLM inference on the administrator-side system. It also allows for traceability throughout the workflow, linking the captured file, detection result, transmitted evidence, the LLM analysis, and administrative records.

The integration that follows creates a continuum from the capture of the artefact to detection, evidence transmission to the administrator side, analysis of the evidence, management of the results, and monitoring. For each of these components, a defined interface is provided, rather than direct coupling between individual components.

4 Results

4.1 Signature-Based Detection Results

The signature-based detection components were tested prior to and during inclusion in the full detection pipeline. A curated collection was built from 1,047 successfully compiled YARA files

containing 6,542 rule declarations. During the validation process, it was confirmed that there were no duplicate rule names in the final collection.

YARA Component	Count
Compiled Rule Files	1,047
Rule Declarations	6,542
Duplicate Rule Names	0

Table 4 Summary of the validated YARA rule collection.

The end-to-end tests verified that files harvested by the Cowrie honeypot passed through the signature-based detection phases. All files captured were first compared with the local SHA-256 hash database. The remaining files were then scanned with the curated YARA rule collection. Files that could not be identified by either signature-based stage were sent to the AI-based detection stage.

If a malicious detection was made through either SHA-256 or YARA, the detection evidence was also sent to the administrator-side analysis component to be later analyzed by an LLM. This validated the anticipated dynamic between the signature-based detection phases and the next analysis stage.

4.2 AI Malware Detection Performance

The CatBoost classifier was tested on the 152,321 samples held out for testing. The model achieved an accuracy of 95.83%, with a precision of 95.50%, recall of 95.98%, and F1-score of 95.74%. The model also had a high ROC-AUC of 99.48%, which demonstrates the ability of the model to discriminate between the benign and malicious samples.

Metric	Result
Accuracy	95.83%
Precision	95.50%
Recall	95.98%
F1-score	95.74%
ROC-AUC	99.48%

Table 5 Performance of the CatBoost malware detection model on the test set.

The results show that the classifier has a good balance of precision and recall. The high ROC-AUC further indicates that the model was able to effectively distinguish between the two classes in the evaluation dataset.

4.3 Confusion Matrix Analysis

The confusion matrix gives a detailed report of the classification results for the test set.

The presence of false negatives is reflected in the model's recall, which measures the proportion of malicious samples correctly identified.

The false-negative cases are especially important for malware detection as they are malicious samples that are incorrectly classified as benign. The number of false negatives is in line with the model's recall, whereas the relatively small number of false positives is captured by the model's precision of 95.50%.

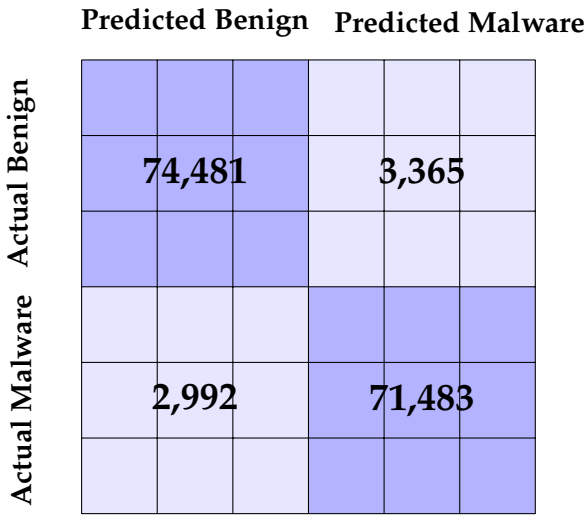


Figure 8 Confusion matrix of the CatBoost malware detection model evaluated on the 152,321-sample test set.

4.4 LLM-Based Malware Analysis Results

The LLM-based analysis component was demonstrated using a representative detection case to assess whether the model could analyze the available detection evidence and produce a structured malware assessment. The evaluation focused on whether the generated output could provide a potential malware family assessment, confidence level, supporting evidence, reasoning, recommendations, and evidence-based limitations.

Analysis Component	Generated Output
Malware Family	Custom Backdoor
Confidence	85.0%
AI Detection Evidence	Malware prediction with 99.98% confidence
Key Evidence	VirtualAlloc and VirtualProtect imports; additional Windows API indicators
Reasoning	The detected imports and static characteristics were interpreted as consistent with backdoor behavior.
Recommendation	Monitor the file for suspicious process and network activity and perform further analysis of the identified backdoor characteristics.
Limitation	The assessment is limited to the evidence available in the submitted evidence package.

Table 6 Example of structured LLM-based malware analysis output.

Table 6 demonstrates that the administrator-side LLM can transform the available detection evidence into a structured higher-level assessment. In this example, the AI classifier provided the initial malicious classification, while the LLM used the supplied static evidence to generate a potential malware-family assessment, supporting reasoning, recommendations, and limitations. The LLM confidence represents its confidence in the family-level assessment and is therefore distinct from the malware classification confidence produced by the AI classifier.

4.5 End-to-End System Results

The end-to-end evaluation covered the Cowrie honeypot, signature-based detection, AI classification, administrator-side LLM analysis, storage, containment, alerting, and dashboard monitoring.

Through end-to-end testing, it was found that captured files could be recovered from the honeypot and passed through the detection pipeline based on the detection logic defined. Detection results and related evidence were successfully sent to the analysis service located on the administrator side; the evidence could be processed without affecting the edge detection workflow.

The integrated system also ensured traceability of captured files, detection results, analysis records, containment actions, and the information on the dashboard. Malicious detections activated the appropriate containment and alerting actions, and the resulting information was displayed in the monitoring interface.

The results illustrate the seamless integration of the individual elements within an overall detection, analysis and response process.

5 Discussion

The achieved results confirm the viability of integrating signature-based detection with machine learning in a lightweight edge-based malware detection system. The CatBoost classifier had an accuracy of 95.83%, precision of 95.50%, recall of 95.98%, F1-score of 95.74%, and ROC-AUC of 99.48% on the test set that was evaluated. The relatively balanced precision and recall show that the model was able to recognize malicious samples while limiting incorrect classifications of benign samples as malicious. The confusion matrix also reveals that there were still false negatives, indicating that machine-learning predictions should not be relied upon in isolation.

The results also indicate the importance of using a multi-layered detection strategy, instead of a single detection mechanism. SHA-256 hash matching and YARA are fast and deterministic ways to identify known threats, and the AI-based classifier works as an extra layer to find files that are not identified by the signature-based stages. This is a sequential design, where the more time-consuming feature extraction and machine learning inference are applied where necessary and not to all captured files. Thus, the framework can operate at the edge in a lightweight manner while also going beyond the scope of signature detection.

The LLM-based analysis is also an extension of the system from detecting binary malware to interpreting evidence. The administrative-side LLM is not intended to replace the detection mechanisms, but rather to be fed with the available detection evidence and give an additional assessment based on the information received from earlier stages. This separation is especially relevant as the AI classifier identifies if the file is potentially malicious or not, while the LLM offers a broader understanding of the information on the file. This interpretation is thus highly dependent on the evidence produced by the detection pipeline.

The system architecture also illustrates the advantages of separating edge detection from administrator-side analysis. The detection, evidence gathering, and containment remain within the edge environment, and the LLM inference is handled on the admin-side system. The API-based communication between the two environments enables evidence to be transmitted from the edge to the administrator-side system without necessitating the edge component to do LLM inference locally. This separation will help with the goal of being lightweight without sacrificing

the ability to conduct more in-depth analysis of suspicious files.

Lastly, the end-to-end evaluation shows that the individual components can be used as an integrated detection and response process. Files collected by the Cowrie honeypot may continue to flow through the detection stages, create evidence, be analyzed by the administrator-side LLM, and initiate containment and alerting if detected as malicious. This shows that the proposed framework is not restricted to a single machine learning experiment, but enables an integrated workflow that integrates detection, analysis, response, and monitoring.

6 Limitations and Future Work

Although encouraging results were obtained, there are some drawbacks in the current implementation and evaluation.

The AI model was first trained and tested on the EMBER2024 dataset. The dataset includes a massive number of labeled benign and malicious samples, but using only one dataset isn't comprehensive enough to prove that the model is effective in detecting malware across a variety of distributions or new samples. Therefore, the current evaluation primarily reflects the EMBER2024 data distribution and does not fully establish generalization to real-world deployment conditions.

Another drawback is the measurement of model generalization and overfitting. Current results are obtained from a held-out test set, showing good classification results; further validation results on independent samples would provide stronger evidence for the robustness of the model. Future work will thus involve targeted validation experiments and cross-dataset validation with the use of independent malware datasets and samples obtained from the honeypot.

Another drawback is the current extent of file-format support. The feature extraction and AI detection pipeline that was implemented supports PE32, PE64, .NET, and ELF. At this moment, it does not support other executable or file formats such as PDF and other non-supported files. Further research will be carried out to expand the feature extraction and detection pipeline to encompass other file formats and to enhance the applicability of the feature extraction and detection process in heterogeneous edge environments.

The LLM-based analysis component is deployed as an administrator-side evidence analysis layer and has the ability to provide malware-family assessments and supporting reasoning based on evidence. In the current evaluation, however, no sufficiently large and labeled benchmark was used to quantitatively evaluate the accuracy of family-level attribution for a wide variety of malware families. This will be further explored in future work by analyzing a larger and more diverse collection of labeled malware samples and detection cases, leveraging the LLM.

The current framework also doesn't have dynamic malware analysis in the operational detection pipeline. Future work will explore the possibility of an optional integration with an external CAPE Sandbox to enable deeper analysis of selected high-risk samples without the need to sacrifice lightweight design of the edge detection components.

Lastly, while the framework is edge-oriented with a lightweight design, it has not yet been quantified and evaluated with detailed resource usage across various edge devices. For more quantitative evidence of the lightweight design and to evaluate scalability on different deployment environments, future work will include benchmarking CPU utilization, memory consumption, inference latency, storage overhead, and end-to-end processing time.

7 Conclusion

This project introduced a lightweight, edge-based framework for detecting and responding to malware using a Cowrie honeypot with multiple detection methods, such as SHA-256 hash matching, YARA rule scanning, and AI-based malware classification. The detection evidence can then be sent to an administrator-side LLM for further malware analysis, and malicious files can be automatically quarantined and monitored via the dashboard.

The evaluation showed that the CatBoost model achieved strong performance on the evaluated test set, while the end-to-end testing demonstrated the integration of detection, analysis, containment, and monitoring components. The layered architecture enables known threats to be identified quickly and easily through signature-based detection and allows for enhanced detection and evidence-based analysis of files that are not detected by signature-based methods and are processed by the AI-based detection layer.

In summary, the project highlights that a practical edge-based security framework involving all four components – signature-based detection, machine learning, LLM-supported analysis, and automated containment – can be realized.

References

[1] Niclas Ilg et al. "A Survey of Contemporary Open-Source Honeypots, Frameworks, and Tools". In: *Elsevier (Preprint)* (2023).

[2] Ahmed Ammar Kubba et al. *A Systematic Review of Honeypot Data Collection, Threat Intelligence Platforms, and AI/ML Techniques*. SSRN Preprint. 2025.

[3] Maryam Var Naseri et al. "Q-Cowrie: An Adaptive Honeypot to Analyse Attackers' Behaviour". In: *International Journal of Information Security* (2026).

[4] Olufunbi Babalola, Olaitan Miriam Olufisayo Raji, Jamiu Olamilekan Akande, et al. "AI-Powered Cybersecurity in Edge Computing: Lightweight Neural Models for Anomaly Detection". In: *International Journal of Multidisciplinary Research and Growth Evaluation* (2024).

[5] Parasu Raman M et al. "HoneyTrap: AI-Powered Honeypot for Cyberattack Detection". In: *International Journal of Engineering Research & Technology* (2026).

[6] The Cowrie Project. *Cowrie SSH and Telnet Honeypot Documentation*. https://docs.cowrie.org/en/stable/README.html. Accessed: 2026-08-06. 2026.

[7] abuse.ch and Spamhaus. *MalwareBazaar Community API*. https://bazaar.abuse.ch/api/. Accessed: 2026-08-06. 2026.

[8] Victor M. Alvarez and YARA Contributors. *YARA Documentation*. https://yara.readthedocs.io/. Accessed: 2026-08-06. 2026.

[9] Florian Roth and Contributors. *Signature-Base: YARA Signature and IOC Database*. https://github.com/Neo23x0/signature-base. Accessed: 2026-08-06. 2026.

[10] ReversingLabs. *Open-Source YARA Rules*. https://github.com/reversinglabs/reversinglabs-yara-rules. Accessed: 2026-08-06. 2026.

[11] Robert J. Joyce et al. "EMBER2024 – A Benchmark Dataset for Holistic Evaluation of Malware Classifiers". In: *arXiv preprint arXiv:2506.05074* (2025).

[12] Liudmila Prokhorenkova et al. "CatBoost: Unbiased Boosting with Categorical Features". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018.

[13] Qwen et al. “Qwen2.5 Technical Report”. In: *arXiv preprint arXiv:2412.15115* (2024).