MULTIDISCIPLINARY RESEARCH INSTITUTE INDIA

Multidisciplinary Journal of Research in Engineering and Technology

ISSN: 2348-6953

Volume 13 Issue 2s (2026) | Pages: 153-158

AI-Driven Cloud Honeypot Network

Shilpa Vishwabramha[1], Vaishnavi Chavan[2], Yuvraj Dudhal[3], Aniket Patil[4], Akash Wavhal[5]

[1,2,3,4,5]Department of Computer Engineering, PDEA's College of Engineering Manjri, Pune

[2]*84vschavan@gmail.com*, [3]*dudhalyuvraj@gmail.com*, [4]*ap575819@gmail.com*, [5]*wavhalakash44@gmail.com*

Peer Review Information

***Type**: Article*
***Received**: 27 March 2026*
***Revised**: 12 April 2026*
***Accepted**: 26 May 2026*
***Published**: 16 June 2026*

Abstract

This project introduced the cloud based honeypot system development and uses the data collected by the honeypot to create comprehensive machine learning models and understand patterns and fingerprints to eventually categorize this data into a potential threat or a malicious activity. By utilizing cloud services, the system can be scaled up and down as required, the architecture integrates advanced data capturing techniques and date pipelining for proper isolation of the data collected and also the machine learning models, the architecture is comprised of capturing services, machine learning service, dashboard and a data pipeline to help isolate the individual components and services, data captured through the capturing services is uploaded to the machine learning service for data classification and model making. The research addresses the critical issue of lack of data for machine learning models in security and attack patterns including cloud environments.

Keywords: Cybersecurity; Artificial Intelligence; Cloud Computing; Honeypot; Machine Learning; Random Forest; Threat Intelligence; Network Security.

How to Cite This Article

Vishwabramha, S., Chavan, V., Dudhal, Y., Patil, A., & Wavhal, A. (2026). AI-driven cloud honeypot network. *Multidisciplinary Journal of Research in Engineering and Technology*, 13(2s), 153–158.

Introduction

Traditional security measures are often lacking in nature with respect to dynamic detection and fast processing. Honeypots have been used in the industry for a long amount of time to capture data on malicious and threat actors in order to collect and understand attack patterns from real hackers to understand attacker behavior and capture real IOCs (Indicators of Compromise). Deploying cloud honeypots requires addressing unique challenges including scalability, integration with cloud services and maintaining realistic deceptions. This project develops an intelligent cloud honeypot system that combines traditional deception techniques with Artificial intelligence to create an adaptive mechanism and collect threat patterns, fingerprints and intent.

Literature Survey

From the research about the honeypot we see that the research about the honeypot over the past few years, largely because the attackers have become more sophisticated and advanced attacks are also added. Reviewing exiting work helps us to identify both what has already been tried and where the gaps are particularly the lack of cloud native, ML-integrated honeypot systems that can actually learn from the data they collect.

Mudgal and Bhatia [1] they combine their neural network with honeypot intelligence layer, which gave their system real-time big data processing capability rather than relying on batch jobs. Their reported classification accuracy is 98.09%.What their works demonstrated most clearly to us is that coupling a honeypot with a live ML pipeline but their setup assumed structured IOT traffic, which is different from nosier, mixed-protocol traffic we want to handle it in the cloud environment.

A comparative evaluation across seven honeypot like cowrie and all [2] we get that this existing tools different in detection range, scalability and data quality. From all those we observed that no single honeypot tool is best across all dimensions, and ML and cloud integration were being recommended but not yet implemented in any of the reviewed systems. This gaps directly influenced the direction of our project.

Kareem et al.[3] they introduced Ai-driven adaptive honeypots, this is the system that don't just sit and wait for attacks, but actively modified their behavior based on observed threat patterns. But it introduces some complexity that when and how to trigger configuration changes without alerting an attacker. So from this research we decided to include at least a scheduled retaining mechanism even if full real-time adaptation of our project.

Work on Particle Swarm Optimization for malicious node detection in wireless sensor networks [4] is somewhat tangential to our focus, but it highlighted an interesting point: the underlying problem of identifying which node or packet is malicious in a distributed environment is a shared challenge across many security domains. The feature selection insights from PSO-based approaches informed our thinking about which packet-level features are most discriminative.

Kubba et al. [5] provided a thorough systematic review covering honeypot data collection methods, threat intelligence sharing frameworks like MISP and STIX, and AI/ML applications across the cybersecurity stack. Their review confirmed that while LLMs and deep learning are increasingly being explored for anomaly detection, there is still limited real-world deployment of ML models trained directly on honeypot-captured data. This matched our own experience in finding usable, labelled datasets — the lack of good training data is a genuine bottleneck, and it is one of the core motivations behind building a system that generates its own.

Research on honeypot integration with Network Intrusion Detection Systems [6] highlighted the value of honeypots not just as standalone tools but as intelligence sources that feed into broader detection infrastructure. The classification challenges discussed — particularly around false positives in NIDS environments — were relevant to our own labelling strategy, where distinguishing legitimate traffic from scanner noise is non-trivial.

Taken together, the reviewed literature makes clear that the field is moving toward AI-augmented, cloud-deployable deception systems, but that most existing implementations either lack a real ML component or lack the architectural safeguards needed to run safely in an exposed cloud environment. Our system attempts to address both of these gaps simultaneously.

Problem Statement

Old honeypot systems suffer from some critical limitations that reduce their scalability and effectiveness against the cyber-attacks. To develop dynamic detection mechanisms, ample amount of proper data is not usually readily available or requires a lot of preprocessing and cleaning, often times the correct parameters or data features are not available to develop dynamic detection systems. Static rule-matching based detection mechanisms lack the capability and processing to adapt in the AI-driven world of cyberattacks.

Proposed System

Based on the literature survey and research, traditional systems are lacking and falling behind where AI-driven cyberattacks are becoming increasingly common.

The system begins by listening on the ports (22,2222,80,443) which are the standard ports for ssh and http/https servers that capture the session metadata including IP addresses, ports, payloads, protocol and timestamps. All captured data is logged on the honeypot machine for persistence then sent to the machine learning service through the data pipeline, as logs are accumulated with each update the log file is updated and sent to the machine learning service for classification. Logs are preprocessed for feature extraction and stored for training purposes. The system employs the Random Forest Machine Learning that undergoes scheduled training cycles after some accumulation period to train on new data.

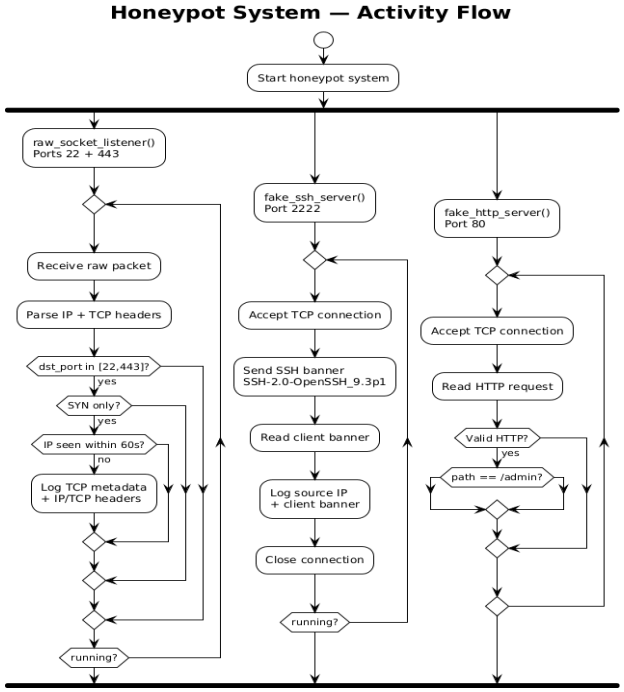

Fig. 1. *Honeypot System Activity Flow Diagram*

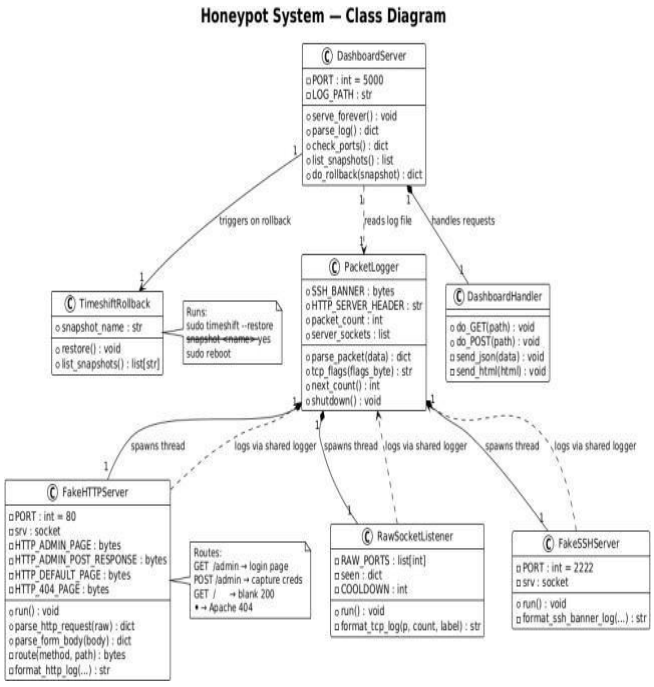

Fig. 2. *Class Diagram of the AI-Driven Cloud Honeypot Network*

System Architecture

The system architecture is comprised of various components working together, each component either deployed on a different system or combined on a single system. (Even though the whole system is majorly developed in python, some components like the dashboard may make use of HTML, CSS and JavaScript)

- RawSocketListener: The socket listener monitors all the packets going through the ports 22 and 443, even though binding is a good option binding the honeypot program to the port disables ssh access into the server
- FakeSSHServer: SSH is mainly the primary utility used to get access into server environments or systems, thus a fake ssh server is bound to the port 2222 which accepts the tcp connection, reads and logs the client banner and other required data without actually giving access to anyone.
- PacketLogger: The logger service is a secondary component of the honeypot allowing the system to log all the packet which come through the ports being actively monitored.
- DashboardServer: The dashboard server makes use of Python Flask to display the metrics of how many unique IP addresses have interacted with the system, which ports are open and also a rollback function to provide a rollback for the honeypot to its previous state in case of compromise.
- Data Pipeline: The data pipeline makes use of SSH and SCP to send the packet logs to the machine hosting the machine learning model, each time the logs are updated.

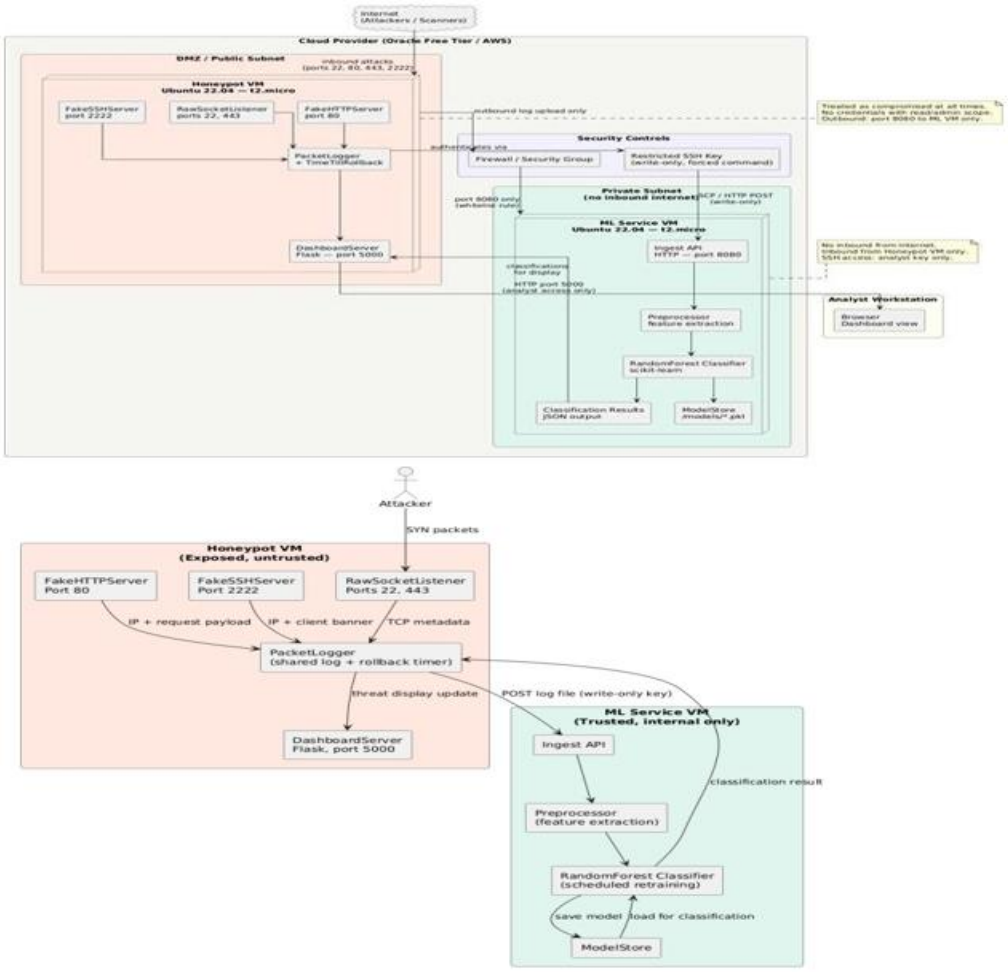

Fig. 3. *Deployment Diagram*

Implementation Details

Tech stack: Python, Python Libraries (Scapy, Paramiko, Flask, scikit-learn, pandas, numpy etc.)

Port binding: 22, 443 are the ports used for SSH and HTTPS respectively by default, no programs are bound to these port for ease of access however each scan and handshake is logged and captured, ports 2222 and 80 are ports where the program binds to send fake banners and data to the malicious actor since 2222 is a general port and 80 is the port for HTTP which is an unencrypted protocol.

Rollback Mechanism: The roll back mechanism is a safety measure used to ensure the safe rollback of the entire system in case of compromise or malicious activity outside the scope of the system, since the honeypot exists as a virtual machine on the VMware platform after the complete build and test a snapshot of the image of the virtual machine is taken and saved thus allowing easy and safe rollback of the system through the dashboard by entering the snapshot number provided.

Isolation: Lateral movements are attack patterns when the malicious actor will move latterly and gain access to other systems associated or connected with the honeypot to avoid this problem components are isolated and kept separate from the attack surface, root access is protected by a strong password. The ML system exists on a completely different system all the data is parsed and cleaned on the ML system separate from the honeypot and attack surface.

SCP/Upload mechanism: Upload mechanism is authenticated via SSH key and restricted permissions, which restricts SSH to only run the SCP commands required for the uploading of log files, with limited access a malicious actor can't SSH directly into the ML system and since all the data is cleaned after being received malicious data cannot be sent through to compromise the system.

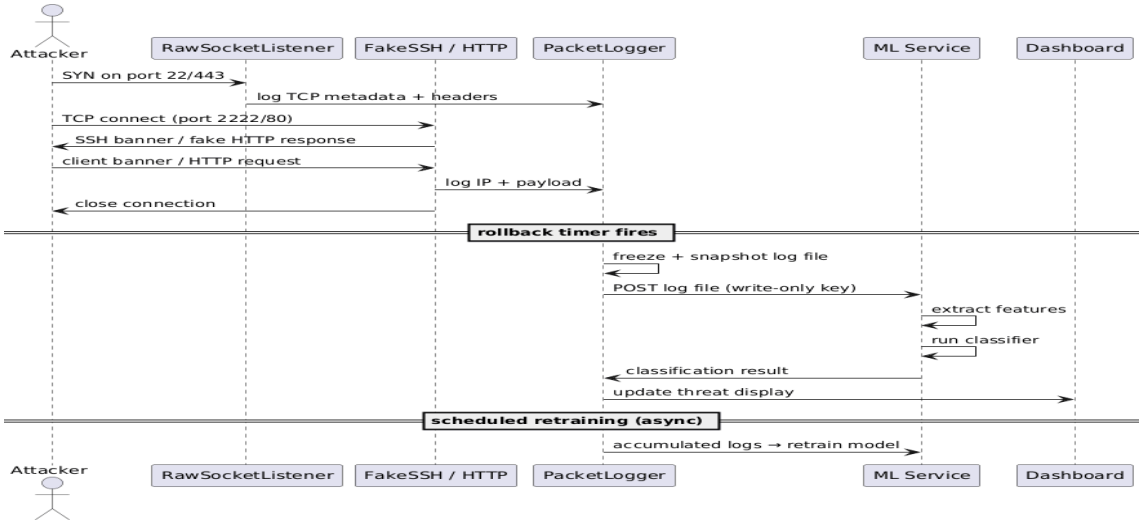

Fig. 4. *Sequence Diagram*

Machine Learning

- Feature Engineering: Features are extracted from the log files on the Machine Learning system, features such as destination port, TCP flags, payload-size, time, client banner, HTTP method, User-Agent are important for the ML model to accurately classify different packets as malicious or friendly.
- Label Definition: Each data point is primarily defined by a label for training as malicious or verified at the early stages while training, for the purpose of development another machine is being used as an attack vector to simulate attacks on the honeypot and ML system for the purpose of testing and identification.
- Training Schedule: Training happens in timed intervals as data is accumulated on the system with different simulations and tests, frequent training or training with each log data update can waste compute and overfit the model, however missing training or training on less data can cause the model to lose accuracy or underfit.
- Feature Importance: Random Forest classifiers give out the importance of features in the data or which feature contributed the most in drawing out a classification.

MITRE ATT&CK Mapping

Table 1. *Mapping of Observed Honeypot Behaviors to MITRE ATT&CK Techniques*

Observed Behaviour	**Technique ID**	**Technique Name**	**Tactic**
Automated SYN probes across multiple ports	T1046	Network Service Discovery	Reconnaissance
SSH banner grabbing and version fingerprinting	T1592.002	Gather Victim Host Information: Software	Reconnaissance
Repeated SSH credential attempts	T1110.001	Brute Force: Password Guessing	Credential Access
Credential stuffing via common username/password pairs	T1110.004	Brute Force: Credential Stuffing	Credential Access
HTTP User-Agent spoofing to masquerade as browsers	T1036	Masquerading	Defense Evasion

Challenges and Limitations

- Honeypot Realism: Sophisticated malicious attackers can easily detect a low interaction honeypot which allows them to exploit the honeypot itself or alert other attack vectors preventing the honeypot from collecting important data.
- Data Volume: Continuous collection of data, collecting and storing every packet can not only overwhelm the compute but also cause the model to lose accuracy and predictability due to imbalance of meaningful data.
- Feature Quality: Understanding the features is the most important step to help improve the machine learning model and retain accuracy and precision.

- Evasion: Threat actors can understand the honeypot environment as mentioned previously and gain lateral access to the entire system causing the compromise of the entire honeypot and data pipeline even though a rollback system is provided the sophistication and advancement of a malicious actor can never be known.

Future Work

The current system gives us a working foundation, but there are some several direction which we get during the development and testing time. Our low-interaction honeypot does a reasonable job of capturing scanners and automated credential-stuffing bots, but a more careful attacker who tries to do an actual SSH handshake will quickly notice that something is off — the server doesn't behave quite like a real OpenSSH installation. Moving toward medium-interaction services that simulate fuller protocol stacks more convincingly would help capture a wider range of attacker behavior, particularly from human operators rather than just bots.

In the machine learning pipeline setup our project used the random forest for classifying individual packets and sessions, but it doesn't handle multi-stage attack sequences naturally. Also we want to experiment with LSTM-based sequence models that can take session-level data as input and detect these slower, attack patterns. During the testing with IoT protocol we currently target the SSH and HTTP, which are the most commonly attacked services in general cloud environments, but industrial and embedded systems communicate over protocols like MQTT and Modbus that have very different attack patterns.

Conclusion

Building this system taught us quite a bit about the gap between how honeypots are described in research and how they actually behave when exposed to real internet traffic. The theoretical appeal of combining deception technology with machine learning is clear, but the practical challenges — getting reliable labels, managing data volume, keeping the ML system isolated from the attack surface — are non-trivial and rarely discussed in existing literature at the implementation level.

What we ended up with is a system that captures SSH and HTTP connection attempts across standard ports, logs session metadata including IP addresses, client banners, TCP flags, payload sizes and timestamps, and then routes that data through a secure pipeline to a separate ML service for classification. The Random Forest classifier is retrained on a schedule as new data accumulates, which avoids the instability of per-update retraining while still allowing the model to adapt over time.

The architectural decision we are most satisfied with is the zero-trust boundary between the honeypot VM and the ML service. Treating the honeypot as compromised by default — restricting its outbound access to write-only log delivery via forced SSH commands and a whitelist firewall rule — means that even a full honeypot compromise cannot be used to pivot into the ML system or the analyst workstation. This kind of network-level trust boundary is conceptually obvious but rarely implemented correctly in prototype systems, and we think it represents a meaningful contribution to practical honeypot engineering.

However, there are a few points where our system falls short or has some shortcomings. A determined attacker who knows they are dealing with a honeypot can evade it relatively easily — the absence of a real SSH key exchange is a giveaway to anyone who looks carefully. The training labels carry noise because our ground truth comes from heuristics and threat feeds rather than confirmed human analyst review. And the data poisoning risk is real: an attacker who suspects they are being studied could deliberately generate benign-looking traffic to degrade the model over successive retraining cycles. These are not problems we have solved — they are open challenges that informed the future work directions described above.

Overall, we believe this project demonstrates that a lightweight, cloud-deployed honeypot with an integrated ML classification pipeline is practical to build and operate at small scale. More importantly, it generates its own training data — addressing one of the most commonly cited barriers to applying machine learning in this domain. As attack tooling continues to incorporate AI-assisted techniques, defenders need systems that can learn and adapt rather than match rules against known signatures. This project is one small step in that direction.

References

1. Mudgal and S. Bhatia, "Big data with machine learning enabled intrusion detection with honeypot intelligence system on Apache Flink (BDMLIDHIS)," Springer, 2024.
2. J. Buzzio-Garcia, "Creation of a high-interaction honeypot system based on Docker containers," in Proc. 5th World Conf. on Smart Trends in Systems, Security and Sustainability (WorldS4), 2021.
3. S. A. Kareem, R. C. Sachan, and R. K. Malviya, "AI-driven adaptive honeypots for dynamic cyber threats," IEEE, 2024.
4. A. Kubba, O. M. Al Mutasim, M. Abu Talib, and Q. Nasir, "A Systematic Review of Honeypot Data Collection, Threat Intelligence Platforms, and AI/ML Techniques," SSRN Electronic Journal, 2024.
5. Z. Moric, V. Dakic, and D. Regvart, "Advancing Cybersecurity with Honeypots and Deception Strategies," 2024.
6. P. Cisar, "The Place and Role of Honeypot Solutions in Network Intrusion Detection Systems," 2025.