Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

AI-Driven Cyber Deception: A Reinforcement Learning Honeypot Framework with Threat Intelligence and Automated Containment

Iqra Hussain

iqra.hussain@zu.ac.ae

Zayed University

Monther Aldwairi

Zayed University

Azka Wani

Zayed University

Research Article

Keywords: AI honeypots, cyber deception, reinforcement learning, Q-learning, threat intelligence, automated response, SOC metrics, machine learning, cybersecurity

Posted Date: April 27th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9065129/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

AI-Driven Cyber Deception: A Reinforcement Learning Honeypot Framework with Threat Intelligence and Automated Containment

Iqra Hussain[1*], Monther Aldwairi[2] and Azka Wani[2]

[1*]College of Technological Innovation, Zayed University, Dubai, UAE.
[2]College of Technological Innovation, Zayed University, Abu Dhabi, UAE.

*Corresponding author(s). E-mail(s): iqra.hussain@zu.ac.ae;
Contributing authors: Monther.aldwairi@zu.ac.ae; Azka.wani@zu.ac.ae;

Abstract

Honeypots are widely used cyber-deception tools for diverting adversaries and collecting behavioral telemetry, yet most deployments remain static and increasingly easy to fingerprint, limiting engagement and reducing the quality of collected intelligence. A dynamic AI-powered honeypot framework is proposed that integrates three capabilities into a single closed-loop defense pipeline: (i) a reinforcement-learning (RL) deception agent, formalized as a Markov Decision Process (MDP) and trained via tabular Q-learning, that continuously adapts the honeypot's exposed persona and interaction tactics; (ii) real-time threat-intelligence fusion using Indicators of Compromise (IoCs) to contextualize and guide deception decisions; and (iii) automated containment actions that reduce time-to-response after detection. The framework is evaluated in a virtualized, isolated testbed across five representative adversarial behaviors: SSH brute-force, port scanning, command injection, SQL injection, and a multi-stage intrusion culminating in a Meterpreter session. Across 193 labeled sessions, the framework achieves 97% detection accuracy with a 3.8% false-positive rate, extending the mean attacker dwell time by 4.4× compared to a static baseline while containing threats within a mean of 1.9 seconds. A Deception Index (DI) is formalized to quantify adaptive deception efficiency, and Mean Time to Containment (MTTC) is introduced as a standardized response metric. The modular architecture is validated on an enterprise-grade VMware stack and is designed for broad deployment across cloud, enterprise, and IoT environments, offering a replicable template for integrating adaptive deception into operational security workflows.

Keywords: AI honeypots, cyber deception, reinforcement learning, Q-learning, threat intelligence, automated response, SOC metrics, machine learning, cybersecurity

1 Introduction

Cyber defense is no longer just about establishing higher walls; it requires outmaneuvering adversaries through deception and intelligence. Modern cyber defense increasingly emphasizes proactive engagement, not only blocking malicious activity but also deceiving intruders into revealing their tactics and tooling. Honeypots operationalize this idea by presenting decoy services whose "value" lies in being scanned, probed, and attacked. They involve luring malicious actors into simulated vulnerable targets and closely monitoring their activities, enabling defenders to gather invaluable insights into attacker techniques while keeping real assets safe. In short, a honeypot turns an attacker's curiosity into actionable threat intelligence, shifting the power dynamic in favor of the defenders [1]. Over the past decade, cyber deception has become a foundation of security architecture, with dynamic honeypots that adapt in real time proving especially effective at remaining undetected by mimicking real systems and behaviors. These adaptive decoys can confuse and slow down adversaries, wasting their time and resources while defenders collect useful forensic data [2].

However, traditional honeypots present significant limitations, as most deployed honeypots are static and display fixed service banners and configurations that savvy attackers can fingerprint almost immediately. Studies show that once an attacker realizes a system is a static honeypot, they disengage quickly, often within 2 minutes, resulting in scant interaction logs [3]. This briefness not only limits the depth of intelligence gathered but also contributes to high false-positive rates in detection, which sometimes leads to benign anomalies being mistaken for attacks when interactions are short. Therefore, there is a need for honeypots that can evolve during an attack session, convincingly emulate changing environments, and hold an intruder's interest for longer periods. Attackers are also increasingly leveraging automation, zero-day exploits, and multi-stage intrusion strategies to bypass defenses, so honeypots must likewise step up their game and think and adapt in real time. Recent advances have explored embedding adaptive decision-making agents within honeypots to address these challenges. For example, modeling the attacker's interaction as a sequential decision process and applying reinforcement learning (RL) has been shown to dramatically improve engagement, one early study demonstrated that an RL-augmented honeypot could sustain intruder interactions roughly three times longer than a traditional static honeypot. By dynamically adjusting its responses and environment in response to attacker behavior, such a learning honeypot can "quickly attract" attackers and then keep them occupied, yielding far richer log data for analysis [4]. Subsequent research has refined these techniques by fine-tuning the RL agent's parameters, such as reward functions, exploration rates, and action choices, leading to even greater effectiveness in capturing malware behavior at scale. These works underscore the potential of AI-driven deception to turn the tables on attackers. [5]

Building on these insights, we propose a next-generation deception framework that moves away from static, predictable decoys toward an adaptive, measurable, and response-aware system. Our framework is built on three core capabilities:

1. **Intelligent, Adaptive Deception:** Using Reinforcement Learning (RL), the system acts as a live target. Instead of a fixed configuration that hackers can easily spot, our RL agent continuously adapts the honeypot's persona, such as rotating service banners, switching protocols, or simulating different vulnerabilities in real time. By treating the attack surface as a moving target, we prevent pattern recognition and keep attackers engaged longer, giving defenders more time to study their methods. For example, the agent can instantly change the decoy's identity, shifting from an SSH server to a web server or even an IoT device interface and quietly pull its responses in real time to keep the attacker confused and off balance.
2. **Live Threat Intelligence Fusion:** To remain relevant against the latest threats, the framework integrates real-time Indicators of Compromise (IoCs) from external threat feeds and databases into the honeypot's logic. This ensures the decoy environment reflects the most current attacker tactics, such as newly reported malicious IPs, exploit signatures, command-and-control domains, and further adapts its behavior accordingly. Therefore, by integrating global threat intelligence, the honeypot stays up to date and contextually relevant, thereby improving specificity and reducing false positives.
3. **Automated Active Response:** The system goes beyond simply watching and logging; our honeypot automatically triggers containment and mitigation actions during an attack. And in the event of a high-confidence detection of malicious activity, the framework can immediately isolate or redirect the attacker, e.g., block the source IP via firewall rules, redirect the session to a deeper sandbox, or implement adaptive throttling, effectively short-circuiting the kill chain. This transforms the honeypot from a passive sensor into an active defense component that not only observes the attacker but also responds in real time to protect the environment. We measure the success of this approach using Mean Time to Containment (MTTC), ensuring the system effectively protects the real network in real-time.

To validate this framework, we implemented and evaluated it within a high-fidelity virtualized environment using the Cowrie SSH/Telnet honeypot integrated with our custom AI components. Over a two-week period involving hundreds of simulated attack scenarios, the system achieved 97% detection accuracy with a false-positive rate below 4%. This significantly outperforms standard baseline honeypots, which typically achieve 78%-85% accuracy. Crucially, the adaptive deception engine extended the average attacker dwell time by a factor of four, increasing from 1.2 minutes on a static decoy to 5.3 minutes. At the same time, automated containment actions were triggered within seconds, limiting the potential damage from each intrusion. The framework's modular design makes it applicable across cloud, enterprise, and IoT settings, and its

integration of deception, intelligence, and response represents a holistic advance in honeypot technology.

In summary, this work makes the following key contributions to the current state of the art:

1. **Unified Closed-Loop Deception Architecture**: We design a modular AI-powered honeypot framework that tightly integrates reinforcement learning for adaptive deception control, live threat intelligence via real-time IoC ingestion, ML-based session classification using both TF-IDF and Bag-of-Words command representations, and automated network-level containment into a single cohesive pipeline. Existing frameworks address these capabilities in isolation or partial combinations, as adaptive honeypots such as HoneyIoT omit threat intelligence integration and automated response [5], while intelligence-driven platforms lack real-time deception adaptation, a gap documented across large-scale honeypot deployments [2] that this framework directly addresses (see also Table 1 for a structured comparison).
2. **Rigorous Mathematical Formalization**: We define the deception control problem as a Markov Decision Process (MDP), formally specifying the state and action spaces, a weighted multi-objective reward function, and a tabular Q-learning update rule. This formalization provides a reproducible theoretical foundation for future RL-driven honeypot research and enables principled comparison with alternative deception policies.
3. **Broad and Realistic Evaluation Scope**: We evaluate the framework against five distinct attack families, namely SSH brute-force, Nmap reconnaissance, command injection, SQL injection, and multi-stage exploitation via Meterpreter, covering a substantially wider threat surface than prior adaptive honeypot evaluations, which predominantly focus on SSH-only scenarios. This breadth demonstrates robustness against both automated and human-driven adversarial tactics.
4. **Comparative Benchmarking and Novel Defensive Metrics**: We benchmark the framework against static Cowrie, T-Pot, and the RL-based Asgard honeypot and introduce two standardized metrics for evaluating dynamic deception systems: the Deception Index (DI), which quantifies adaptive engagement efficiency normalized by deception action frequency, and Mean Time to Containment (MTTC), which measures automated response latency from threat detection to enforced mitigation. Both metrics are designed to serve as reusable benchmarks for future honeypot research.

The remainder of this paper is organized as follows. Section 2 (Related Work) examines existing honeypot designs and deception techniques and explains how our approach differs from earlier static, adaptive, and AI-driven solutions. Section 3 (Methodology) then explains the proposed framework in detail, covering the overall system architecture, the reinforcement learning model, the threat classification and feature analysis process, and the automated response and mitigation components. Section 4 (Experimental Setup) describes how the system was evaluated, including

the testbed environment, the simulated attack scenarios and datasets, and the metrics used to measure performance. Section 5 (Results and Analysis) presents and discusses the findings, including detection accuracy relative to baseline systems, response speed, containment effectiveness, attacker engagement levels, feature-importance insights, comparisons with other RL-based deception frameworks, and the system's robustness against advanced multi-stage attacks. Finally, Section 6 (Conclusion) highlights the main achievements of the study, reflects on its broader implications and limitations, and outlines possible directions for future work.

2 Related Work

Early generations of honeypots were largely static decoys that emulated common services, e.g., SSH, HTTP, with the sole purpose of observing attacker activity. Depending on the interaction level, these ranged from low-interaction honeypots like Kippo/Cowrie that simulate a basic shell, to high-interaction honeypots that run real operating systems for attackers to fully compromise. Static honeypots provided valuable threat intelligence in the 2000s, but their lack of adaptability made them prone to quick detection. Provos and Holz's seminal work on virtual honeypots (Honeyd, etc.) illustrated both the utility and the shortcomings of static decoys: while they could successfully collect malware samples and even aid in botnet tracking, skilled adversaries could fingerprint their emulated services or default banners with relative ease [6]. Researchers have further characterised the fingerprinting vulnerability of static honeypots through empirical measurement studies, demonstrating that even low-interaction emulations can be identified within seconds by tools such as Shodan and Censys [7]. Once identified, a honeypot loses its effectiveness; attackers simply disconnect or avoid interacting deeply, resulting in minimal data collection. Indeed, a study by Rabzelj et al. on a scalable HTTP honeypot platform found that most human-driven attack sessions lasted less than 2 minutes before the attacker disengaged [3]. Such brief interactions yield superficial insight and often lead to high false-positive rates because normal scan or error traffic can appear similar to these fleeting attack sessions. In summary, traditional honeypots that remain static and passive struggle to entice modern attackers long enough to glean meaningful intelligence.

To overcome the predictability of static honeypots, researchers have explored adaptive and intelligent honeypot strategies. One line of work involves applying reinforcement learning to optimize honeypot behavior in response to attackers. Heliza (Wagener et al., 2011) was an early adaptive honeypot that allowed attackers to penetrate a real system, a deliberately vulnerable Linux server, but then intercepted and manipulated their commands. Heliza's RL agent, using a SARSA algorithm, had two possible objective modes: either to waste the attacker's time or to solicit specific malicious tools/techniques, and it would choose actions like introducing bogus responses or "insults" to achieve the set objective [8]. This pioneering work showed that an agent-driven honeypot could learn to influence attacker behavior without being immediately uncovered. However, Heliza's use of a real OS meant a high-interaction approach with inherent security risk, as the attacker is in a real environment that must be carefully controlled.

Subsequent efforts focused on safer, adaptive honeypots that use emulation while still leveraging RL. Pauna et al. introduced RASSH (2014) and later QRASSH (2018), which embedded RL agents into the Kippo/Cowrie SSH honeypot environment. Instead of giving attackers a real system, these solutions mimic an SSH server but dynamically adjust aspects of the interaction. For example, RASSH added a "delay" action to artificially slow command execution, and QRASSH expanded the action set and experimented with different reward schemes, ranging from simple to complex, to maximize attacker engagement [9] [10]. These adaptive SSH honeypots demonstrated significantly improved performance in terms of the number of malicious commands captured from attackers, compared to static Cowrie. Prior studies indicate that reinforcement-learning-driven adaptive honeypots can substantially increase attacker interaction depth and the number of captured command sequences compared to static honeypot configurations [11]. Huang and Zhu (2019) formalized this concept by modeling attacker interactions as a semi-Markov Decision Process and using Q-learning to extend attackers' dwell time on high-value decoys. Their RL-driven honeypot learned an optimal engagement policy that kept attackers interacting for significantly longer than a static policy [4]. More broadly, the application of reinforcement learning to adversarial network security problems has demonstrated consistent advantages over static rule-based policies, particularly in environments where attacker behaviour is non-stationary and cannot be fully characterised in advance [12]. Similarly, HoneyIoT (Guan et al., 2023) applied RL to a high-interaction IoT honeypot, enabling it to mutate its device responses and behavior on the fly. HoneyIoT's agent coaxed malware into fully deploying by dynamically altering IoT sensor data and commands, thereby tricking malware that would normally perform environmental checks [5]. These works show the promise of RL for deception: the honeypot effectively learns to be more enticing and less detectable to the attacker.

Another approach uses game theory and optimization to deploy deception dynamically. Sayed et al. (2023) formulated honeypot allocation in moving target environments as a two-player stochastic game, where the defender's strategy, which is where and how to place honeypots, anticipates the attacker's moves. By solving for an equilibrium, their method determined when to reconfigure honeypots to maximize the probability of trapping attackers while minimizing costs [13]. This highlights an important point: adaptation can occur not just within a single honeypot during an attack, but also at the architecture level, deciding how to position multiple deception assets across a network over time. Recent work in deception resource allocation has employed deep reinforcement learning, e.g., Proximal Policy Optimization, to determine the optimal mix of defensive deception assets (e.g., high- and low-interaction honeypots) to deploy within a network, aiming to confuse attackers and mitigate attacks. These approaches typically define state spaces that represent both network conditions and deception-asset states, and they seek deployment strategies that maximize long-term rewards for attacker engagement or disruption [14]. While our work focuses on a single honeypot's internal adaptation, such broader architectural strategies are complementary and could inform future extensions of our framework toward multi-honeypot or moving-target defense architectures.

Beyond traditional and RL-enhanced honeypots, the community has started exploring AI-driven deception in new dimensions. One emerging area is the use of Large Language Models (LLMs) to improve honeypot interactivity. For instance, HoneyGPT (Wang et al., 2024) integrated a ChatGPT-based agent into a terminal honeypot to generate human-like responses and engage attackers in conversation. This led to deeper social engineering interactions and helped uncover novel attack vectors that purely scripted honeypots would miss [15]. Another recent concept is the "LLM Agent Honeypot" (Reworr & Volkov, 2025), which aims to detect AI-driven attacks. As attackers start employing LLM-based bots that can rapidly issue commands or adapt their strategies, this advanced honeypot uses prompt-injection techniques and timing analysis to distinguish between human attackers and AI agents [16]. For example, it analyzes command-issuance patterns, such as speed, syntax variability, and the language used in any interaction, to infer whether an LLM is at play. The goal is to exploit the quirks of AI agents, such as deterministic phrasing or superhuman typing speed, to turn AI against itself in an "AI vs AI" deception battle. These developments signal a broader shift where deception technology is evolving to counter not just human adversaries but also AI-driven threats. Our framework, while not explicitly focused on LLM-based attacks, is designed to incorporate new detection logic, such as monitoring interaction patterns via its threat intelligence module, so it could be extended to handle such scenarios in the future. However, LLM-based honeypots, while advancing interactivity, remain focused on engagement quality and do not address the integration of live threat intelligence or automated containment.

Despite these advances, the honeypot landscape remains fragmented, as each line of work advances one capability in isolation without unifying adaptive deception, live threat intelligence, and automated response into a single operational system. Significant gaps therefore remain in existing honeypot solutions. Many RL-based honeypots in research remain isolated proofs of concept and do not integrate with real-world security operations. For example, HoneyIoT's adaptation was powerful, but it "stopped at deception" and did not incorporate any threat feed updates or trigger automated defensive actions beyond the honeypot itself [17]. This pattern of partial integration is consistent with findings from systematic analyses of cyber deception architectures, which observe that most deployed systems optimise for a single objective, whether engagement, detection, or response, without considering the interdependencies between all three [18]. In general, current dynamic honeypots lack end-to-end integration with threat intelligence and automated response. As noted in recent large-scale surveys of honeypot deployments, inconsistent integration with vulnerability databases and external IoC feeds often leads to limited specificity and persistent false positives [1][2]. Furthermore, many systems close the loop by automating reactive defenses based on honeypot findings. There have been some steps in this direction: recent work by Chin et al. (2025) introduced an automated defense framework that leverages medium- to high-interaction honeypot telemetry to dynamically update firewall rules in real time, enabling faster containment of attacker activity [19]. Likewise, prior work on SIEM/SOAR automation shows that feeding high-fidelity alerts into automated response pipelines can enable the real-time generation of containment actions, such as firewall rules, thereby reducing response time [20]. Similarly, research

on attack projection and forecasting demonstrates that structured threat indicators can improve anticipatory defense and reduce detection latency [21]. Beyond technical integration, research on the operationalisation of cyber threat intelligence has demonstrated that translating honeypot-derived indicators into structured threat feeds can measurably reduce analyst workload and improve detection specificity across broader security operations workflows [22]. However, such capabilities are usually bolted on to static honeypots. To the best of our knowledge, existing frameworks often address adaptive deception, real-time threat intelligence, and automated countermeasures in isolation or in partial combinations, with relatively limited work that tightly unifies all three within a single closed-loop system. Our work seeks to narrow this gap by presenting a unified platform that not only tracks and logs attacker behavior but also continuously learns from global threat intelligence in real time and enables automated defensive actions during an intrusion.

To put our approach in context, Table 1 summarizes key differences between traditional honeypots and the proposed AI-powered honeypot as per the capabilities discussed above:

Table 1 Comparative overview of traditional and proposed AI-powered honeypot capabilities across six functional dimensions, based on limitations identified in prior honeypot literature

Feature	Traditional Honeypots	Proposed AI-Powered Honeypot
Threat Detection	Passive logging of attacks; little real-time analysis.	AI-driven real-time analysis and classification of threats.
Deception Behavior	Static banners and responses (easy to fingerprint).	Continuously evolving behavior (randomized banners, dynamic service emulation).
Incident Response	Manual (analyst-driven, post-attack).	Automated, immediate responses (IP blocking, session redirection).
Threat Intelligence	Limited (local logs only).	Integrated with live global IoCs and threat feeds for context.
Scalability	Often confined to a single host or service.	Designed for multi-environment deployment (cloud, enterprise, IoT).
Forensic Data	Basic data capture (often shallow interaction logs).	Rich interaction data with AI-enhanced analysis (incl. attacker TTPs, IoCs).

In summary, our work builds upon the rich literature of honeypot research by combining the proven benefits of reinforcement learning–based deception (extended engagement, reduced detectability) with the crucial elements of threat intelligence integration and automated response that are needed for real-world defensive operations. We also push the evaluation of honeypots further by considering a broader range of attack types (including web-based attacks and multi-stage exploits) and by introducing new metrics, such as MTTC (Mean Time to Containment), to assess the

practical impact. In the next section, we detail the architecture and methodology of our proposed framework.

3 Methodology

In this section, we present the design and technical components of our dynamic honeypot framework. We first describe the system architecture, outlining how the various modules: honeypot, RL agent, threat intelligence engine, classifier, and response module interact. We then provide a formal reinforcement learning model formulation, defining the state and action spaces and the reward structure that drives the honeypot's adaptive behavior. Next, we discuss our threat classification pipeline, including feature engineering with the introduction of a bag-of-words feature and the machine learning algorithms used to detect malicious sessions. Finally, we explain the automated response mechanism that triggers a response when threats are detected. Throughout, we emphasize design choices that ensure the framework is generalizable and suitable for realistic deployment.

3.1 System Architecture

The system is implemented as an isolated testbed consisting of a honeypot host, an attacker host, and a monitoring/control plane. Cowrie provides the SSH/Telnet interaction surface and logging substrate. Figure 1 illustrates the overall architecture of the proposed honeypot framework, which is composed of the following integrated layers:

1. **Virtualization Layer:** The honeypot system is deployed in a controlled virtual environment. In our implementation, we used VMware Workstation on a Windows 11 host machine and created isolated virtual networks (NAT/VLAN) to contain honeypot traffic. This setup included two VMs:
   (a) **An Attacker VM:** A Kali Linux instance configured with offensive tools, including Hydra for brute-force attacks, Nmap for scanning, and Metasploit for exploitation, to simulate attackers.
   (b) **Honeypot VM:** An Ubuntu 22.04 server running Cowrie (v4.6.0) in medium-interaction mode, emulating an SSH/Telnet service. This VM also hosts the RL Deception Agent and the ML-based threat classifier as background processes, effectively instrumenting the Cowrie honeypot with AI-driven behavior and detection capabilities.
2. **Adaptive Deception (RL Agent):** The core of our framework is a reinforcement learning agent that continuously observes the ongoing interaction between the attacker and the honeypot and decides when and how to modify the honeypot's behavior. In practice, this agent can take actions such as rotating service banners, enabling or disabling certain fake services, injecting bogus vulnerabilities or files, or altering response timings. The goal is to maximize an engagement metric by keeping the attacker occupied while minimizing the chance that the attacker recognizes the deception. The RL agent interfaces with Cowrie's mechanisms, for instance, by programmatically changing the SSH banner string or by

simulating the opening of another port with a vulnerable service when it detects a port scan. It operates in real-time, meaning that between attacker commands, the agent can perform a deception action.

3. **Threat-Intelligence Engine:** To keep the deception contextually relevant, we include a Threat Intelligence module that ingests live IoCs and threat feed data. This engine pulls data from sources such as known malicious IP lists, recent CVE reports, malware command-and-control domains, etc., via APIs or offline updates. The specifics can be tailored to deployment, e.g., consuming feeds from an organization's threat intelligence platform or open-source feeds. The engine then provides the RL agent and honeypot with up-to-date information, for example, if a particular exploit or tool is trending in the wild, the honeypot can advertise a corresponding fake vulnerability to lure attackers exploiting that flaw. Likewise, if the attacking IP is known from threat intel to belong to a botnet, the framework can adjust its response strategy, perhaps engaging less or focusing on quick containment. The fusion of IoCs ensures the honeypot's behavior and bait are aligned with real-world threat actor tactics in near real time.
4. **Machine Learning Pipeline (Threat Classifier):** All interaction data from the honeypot is fed into a machine learning pipeline that classifies sessions as benign or malicious in real-time. This pipeline further has several stages:
   (a) **Data Collection:** The Cowrie honeypot logs each session's events, commands entered, file download attempts, login attempts, etc, in JSON format. For our experiment, we collected 286 interaction sessions over multiple weeks (210 malicious and 76 benign), of which 193 were retained for ML training and evaluation (117 malicious, 76 benign) after excluding 76 incomplete or insufficient-length sessions and reserving 17 for generalization testing.
   (b) **Feature Engineering:** We extract a rich set of features from the honeypot logs to characterize each session's behavior. Key features include: Session Duration, which is the total time of the session, Command Count, which is the number of commands the attacker attempted, Unique Commands, which is the count of distinct command types used as an indicator of the attacker's toolkit breadth, Behavioral Flags which are binary indicators for certain malicious behaviors, such as failed login attempts, attempted file downloads, or privilege escalation attempt, and Textual Features of commands. For the latter, we use two approaches: a TF-IDF vectorization of the command sequence to capture common substrings or patterns that might indicate specific attack scripts, and a simpler Bag-of-Words (BoW) representation of commands. The BoW feature is essentially an unweighted frequency count of each command or keyword observed in the session, such as how many times wget was used, whether sudo appears, etc., complementing TF-IDF with a raw view of command usage. We introduced the BoW feature to examine whether certain commands or terms, such as "passwd" or "nmap," are stronger indicators of malicious intent than their TF-IDF scores suggest. All features are normalized or encoded appropriately, for example, durations scaled, boolean flags to 0/1, and TF-IDF/BoW vectors dimension-reduced if necessary.

(c) **Classification Model:** We evaluated multiple machine learning algorithms for classifying sessions as malicious vs. benign. In particular, we trained a Random Forest (RF) classifier and a Support Vector Machine (SVM) classifier using the labeled dataset to compare their performance. We found that the Random Forest slightly outperformed the SVM in our context, likely due to the mixture of numeric and textual features and the RF's ability to handle feature interactions. Therefore, the Random Forest was selected for the final framework. The RF model using Scikit-learn was configured with 100 decision trees (n_estimators=100) and a maximum tree depth of 10, with other parameters tuned via grid search on a validation set. We also applied SMOTE oversampling to the training data to mitigate class imbalance, as the malicious sessions outnumbered benign sessions in our simulations. The model is trained offline on historical data, but in deployment, it runs online: as a session progresses, the model can output a probability that the session is malicious. We set a high threshold (0.90) for this probability to trigger actions, prioritizing precision and minimizing false alarms.

(d) **Feature Importance Analysis:** After training the Random Forest, we extract feature importance scores to interpret which inputs are most influential in the model's decisions. As we will detail in the results, this analysis indicated that command content features, such as TF-IDF terms and BoW features, were the strongest contributors, while aggregate behavior features, such as session length and unique command count, also played significant roles, with these three feature types collectively accounting for the greatest importance. This insight validates our approach of including both content-based and behavior-based features.

5. **Automated Response Module:** The final component of the architecture is the response/containment module that acts upon alerts from the classifier or other detection logic. When a session is classified as malicious with high confidence, the system automatically executes a predefined set of defensive actions. In our implementation, these actions included: (a) inserting an iptables firewall rule on the host to immediately block the attacker's IP address cutting off any further communication, (b) sending a signal to the honeypot to terminate the session or redirect it to a sandbox environment for deeper analysis, and (c) logging an alert to a Security Operations Center (SOC) dashboard or Security Information and Event Management (SIEM) system. The integration with a SIEM means our honeypot can feed into the organization's wider incident response process, allowing security analysts to be notified in real time and to review the collected data. We emphasize that these responses occur automatically within seconds of detection, therefore drastically reducing the window in which an attacker can operate freely. We quantify this via the Mean Time to Containment (MTTC) metric in our evaluation.

Fig. 1 provides a visual summary of how these components interconnect. The attacker interacts with the Honeypot VM; their actions are observed by both the RL agent (for deception decisions) and the logging system. The RL agent may alter the

honeypot's responses or environment in real time to keep the attacker engaged. All the while, features are extracted from the interaction and fed to the classifier. If a threat is confirmed, the response module is triggered to contain the attacker. The threat intelligence engine continually updates the RL agents' and honeypots' knowledge bases to inform deception and detection using the latest external data.

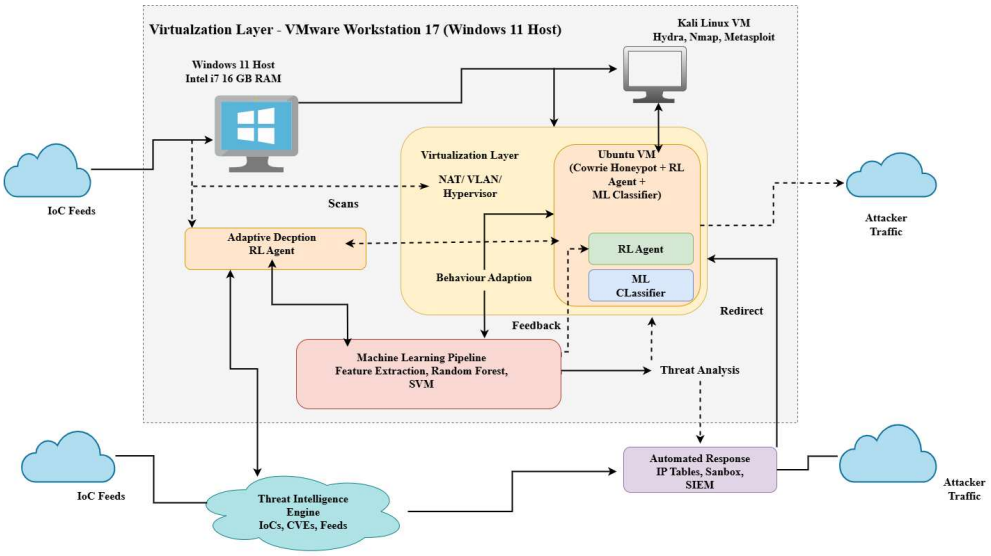


Fig. 1 Architecture of the proposed AI-powered honeypot framework showing the attacker VM, honeypot VM with RL and ML modules, threat intelligence integration, ML-based session analysis, and automated response mechanisms.

3.2 Reinforcement Learning Model

A central uniqueness of our framework is the reinforcement learning (RL) agent that orchestrates adaptive deception. We formally formulate this control as a Markov Decision Process (MDP) $M = (S, A, P, R, \gamma)$ to ensure a reproducible foundation for theoretical analysis, where:

- S: state space capturing recent attacker behavior and session context
- A: deception action space controlling honeypot persona/behavior
- $P(s' \mid s, a)$: transition dynamics induced by the attacker's next action and environment response
- $R(s, a, s')$: weighted reward function measuring engagement versus risk
- $\gamma \in (0, 1)$: discount factor

3.2.1 Environment and Agent

The environment is the honeypot system under attack, including the attacker's observable behavior and system responses. The agent is the deception controller, capable of modifying the system's apparent configuration, services, or outputs to influence attacker engagement while preserving safety. Since attacker behaviour is unpredictable and may vary between sessions, the transition probability $P(s' \mid s, a)$ is not explicitly modelled. Instead, a model-free reinforcement learning approach is adopted, where the agent learns directly from observed interactions in real time [4, 10]. Tabular Q-learning is sufficient given the low-dimensional eight-feature state space, making deep Q-network (DQN) approximation unnecessary.

3.2.2 State Space (S)

The state captures relevant contextual and behavioral information that guides deception decisions. Each session is segmented into discrete time steps t, and the state is represented as a feature vector summarizing recent attacker activity and session context:

$$s_t = [\text{dur}_t, \text{cmd_rate}_t, \text{uniq_cmd}_t, \text{fail_login}_t, \\ \text{download}_t, \text{priv_esc}_t, \text{ioc_match}_t, \text{classifier_score}_t]$$

where,

- dur_t: elapsed session duration
- cmd_rate_t: commands per minute
- $uniq_cmd_t$: unique-command count
- Indicator flags capture behavioral milestones
- ioc_match_t: binary or weighted IoC match score
- $classifier_score_t$: threat probability from the classifier

3.2.3 Action Space (A)

The agent chooses from m discrete deception tactics. In our prototype, we implemented 10 actions (A_1–A_{10}) covering a spectrum from subtle to overt:

- **A1:** Change the SSH banner or greeting message (to a random new flavor, possibly mimicking a different OS or version).
- **A2:** Toggle a fake vulnerability on the system (e.g., "open" a port running a service with a known CVE, without actually exposing a real vulnerability).
- **A3:** Introduce a time delay in responses (simulating system lag).
- **A4:** Present fake filesystem or data artifact (e.g., create a bogus passwords.txt file to entice the attacker).
- **A5:** Dynamically open an additional service port (like running a mock HTTP service with an exploitable interface) if the attacker is port scanning.
- **A6:** Rotate credentials – for instance, if the attacker tries a login, present a different login prompt or reset the SSH keys (to confuse brute-force tools).

- **A7:** Inject misleading output to a command (e.g., for an ifconfig command, show fake network interfaces).
- **A8:** Trigger an intentional error or crash in the honeypot service (and immediately restart) to see if the attacker persists.
- **A9:** Do nothing (no change; sometimes the best action is to maintain the current facade).
- **A10:** Insult or taunt the attacker (this was used in Heliza [8], but we include it as a low-priority action, basically sending a custom message primarily for experimental comparison; it risks revealing the honeypot's nature and thus would be used rarely if at all in optimal policy).

3.2.4 Reward Function (R)

We have defined a weighted reward that explicitly balances engagement gain, intelligence gain, along with the risk and cost. One concrete instantiation is the following weighted reward function formulation:

$$R_t = w_T \cdot \Delta dur_t + w_C \cdot \Delta cmds_t + w_U \cdot \Delta uniq_cmd_t + w_I \cdot ioc_value_t - w_R \cdot risk_t - w_O \cdot overhead_t \quad (1)$$

Where:

- $w_T, w_C, w_U, w_I, w_R, w_O \geq 0$: tunable weight parameters
- Δdur_t: change in session duration
- $\Delta cmds_t$: change in number of commands observed
- $\Delta uniq_cmd_t$: change in unique-command count
- ioc_value_t: threat intelligence indicator value
- $risk_t$: estimated risk level
- $overhead_t$: normalized action cost

Note that these weights are not constrained to sum to 1; each term is independently scaled to reflect its relative contribution to the reward signal. The reward R_t is therefore an unnormalized weighted sum, and the values in Table 2 were selected empirically to reflect the relative priority of each signal rather than to form a probability distribution.

The reward function contains six tunable weight parameters and one scaling coefficient (λ). Table 2 reports the exact values used in all experiments, along with the signal each parameter governs and the justification for its relative magnitude.

To assess reward sensitivity, we tested two alternative configurations: (a) doubling w_T to 0.8 while halving w_C and w_U, and (b) removing the w_I term entirely. Configuration (a) produced marginally longer engagement (5.5 min mean) but increased false containment by 1.2 percentage points, as the agent prioritized duration over risk signals. Configuration (b) reduced the APT capture rate from 67% to 51%, confirming that IoC intelligence integration contributes meaningfully to detecting stealthy attack

Table 2 Reward function weight parameters used in all experiments

Weight Parameter	Value	Signal / Range	Justification
w_T (Session Duration)	0.4	Δdur_t (minutes)	Highest weight; dwell time is the primary engagement signal.
w_C (Command Count)	0.2	$\Delta cmds_t$ (count)	More commands indicate richer telemetry; secondary engagement signal.
w_U (Unique Commands)	0.2	$\Delta uniq_cmd_t$ (count)	Breadth of attacker toolkit; distinguishes targeted from automated activity.
w_I (IoC Intelligence Value)	0.15	$ioc_value_t \in [0, 1]$	Weighted by IoC severity score from threat intelligence feed (0 = none, 1 = critical).
w_R (Risk Penalty)	0.25	$risk_t \in [0, 1]$	Penalizes privilege escalation and payload staging events.
w_O (Action Overhead)	0.05	$overhead_t \in [0, 1]$	Normalized cost of deception action; prevents excessive mutations.
λ (Classifier Weight)	0.3	$classifier_score_t \in [0, 1]$	Scales the classifier's threat probability contribution to the risk term.

patterns that behavioral features alone would miss. The reported weight configuration, therefore, represents an optimal balance for the evaluated threat landscape.

We define risk as:

$$risk_t = 1[\text{priv_esc}_t] + 1[\text{payload_stage}_t] + \lambda \cdot \text{classifier_score}_t \tag{2}$$

Here, $\lambda = 0.3$ was set empirically to balance the classifier's probabilistic contribution against the binary indicator terms. Overhead is defined as a normalized function of the selected action cost.

3.2.5 Q-Learning Policy and Update Rule

We employ tabular Q-learning to learn the optimal deception policy. The Q-value update follows the standard Bellman optimality equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \Big] \tag{3}$$

where α is the learning rate and γ is the discount factor. Action selection follows an ϵ-greedy exploration strategy that balances exploration and exploitation.

3.2.6 Episodes and Convergence

An episode corresponds to a single attacker session, beginning when an attacker connects and ending upon disconnection or containment. Training was conducted over multiple simulated sessions with varied attacker behaviors. Empirically, Q-values stabilised after approximately 200 episodes, as illustrated in Fig. 2. Across five independent training runs, the policy-change rate dropped below the 1% convergence threshold at episode 198 ± 12. The specific hyperparameter values used, including the learning rate, discount factor, and exploration schedule, are reported in Table 3

Algorithm 1 RL-Driven Adaptive Deception (Q-Learning)

Initialize $Q(s, a)$ arbitrarily
Set learning rate α, discount factor γ, exploration rate ϵ
for each episode $e = 1$ to E **do** ▷ One episode = one attacker session
    Initialize state s_0 from first session event
    for each time step t until termination or containment **do**
        With probability ϵ, select random action a_t
        Otherwise select $a_t = \arg\max_a Q(s_t, a)$
        Execute deception action a_t
        Observe next state s_{t+1} from logs, classifier, and IoC matches
        Compute reward $r_t = R(s_t, a_t, s_{t+1})$
        Update Q-value using Bellman equation
    end for
    Decay ϵ (optional) and log convergence diagnostics
end for

in Section 4.1. Convergence was further monitored using a policy-change threshold: learning was considered stable when fewer than 1% of state–action Q-values changed significantly between training batches. During deployment, the agent continues incremental learning with a decaying exploration rate to maintain stability and predictable behavior. Although more complex methods, such as deep Q-networks, could be applied for larger state spaces, tabular Q-learning proved sufficient given the abstracted state representation. Safety considerations were incorporated by penalizing detection and system compromise, encouraging stealth-oriented deception strategies aligned with prior honeypot best practices.

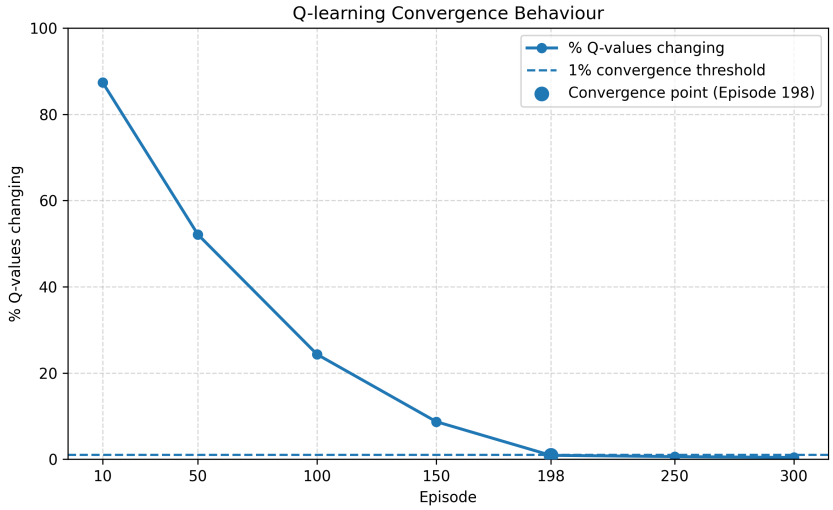


Fig. 2 Q-learning convergence behaviour. The percentage of Q-table entries changing significantly between training batches drops below the 1% convergence threshold at episode 198, confirming stable policy convergence within 200 episodes.

3.3 Threat Classification and Feature Engineering

The threat-classification module determines whether an interaction session is malicious and initiates containment actions when required. We evaluated several supervised learning algorithms, including Random Forest (RF), Support Vector Machine (SVM), and gradient-boosting approaches. Random Forest was selected for deployment due to its robustness, ability to capture nonlinear feature interactions, and interpretability through feature-importance analysis, as validated in Section 5.4. Each interaction session is represented by a feature vector x comprising both behavioral and textual characteristics:

1. **Behavioral Features:** failed login attempts, total command count, unique command count, download indicators, privilege-escalation flags, and session duration.
2. **Textual Command Features:** vectorized representations of command sequences using Bag-of-Words and TF–IDF schemes, optionally extended with short n-grams to capture command semantics and contextual patterns.

SMOTE oversampling was applied to address class imbalance, as the dataset comprised 117 malicious and 76 benign sessions. Oversampling was applied exclusively to training folds during cross-validation to prevent data leakage. Classifier outputs are transformed into probability scores, and a decision threshold of 0.90 was selected in deployment to favor high precision and reduce false-positive alerts. Model effectiveness is evaluated using standard classification metrics, including accuracy, receiver operating characteristic (ROC) analysis, and feature-importance assessment, which are detailed in the evaluation section

3.4 Automated Response and Threat Mitigation

Upon classifying a session as malicious or detecting high-severity indicators, the automated-response module executes containment actions. These actions include IP blocking through firewall rules, session termination or optional redirection to an isolated sandbox environment, and alert generation to centralized logging or Security Information and Event Management (SIEM) systems. The response pipeline is designed to be modular, enabling integration with external security tools and policy-driven escalation mechanisms.

Response efficiency is quantified using the Mean Time to Containment (MTTC), defined as the average elapsed time between threat detection and execution of mitigation actions across all sessions:

$$\mathrm{MTTC} = \frac{1}{N} \sum_{i=1}^{N} \left(t_{\mathrm{contain}}^{(i)} - t_{\mathrm{detect}}^{(i)} \right) \tag{4}$$

Here, t_{detect} is the timestamp at which the classifier probability first exceeds 0.90, and t_{contain} is the timestamp of confirmed `iptables` rule insertion verified in the system audit log. Both are recorded to millisecond precision and synchronised across VMs to within ± 10 ms. This metric provides a standardized measure of automated response

speed and is used in the evaluation phase to compare containment effectiveness across different attack scenarios.

4 Experimental Setup

To evaluate the effectiveness of the proposed framework, we set up a controlled environment and conducted a series of attack simulations covering a broad range of tactics. The evaluation aimed to answer several questions: Can the RL-driven honeypot indeed increase attacker engagement compared to static honeypots? How accurate and fast is the threat detection? Does the framework scale to different types of attacks, such as brute-force, scanning, injections, and malware payloads? And how does it compare to other honeypot solutions? Below, we describe the environment, attack scenarios, dataset, and metrics used in our experiments.

4.1 Testbed Environment

All experiments were conducted on a virtualized testbed designed to emulate a small enterprise network under controlled attack conditions. Virtualization enabled safe isolation from production environments, rapid snapshot restoration, and repeatable experimentation.

1. **Host System:** The host machine ran Windows 11 Enterprise with VMware Workstation 17 on an Intel i7 processor, 16 GB RAM, and a 1 TB NVMe SSD. This configuration provided sufficient computational resources while allowing full network isolation and system rollback between experimental runs.
2. **Network Configuration:** A dedicated NAT-based virtual network was created within VMware to interconnect the attacker and honeypot virtual machines. This setup ensured that all attack traffic remained confined to the test environment and did not reach the public internet. Packet capture was enabled to record network traces for subsequent traffic analysis. In selected experiments, VLAN tagging was enabled to evaluate multi-instance honeypot scenarios, although the primary configuration used a single subnet.
3. **Honeypot Virtual Machine:** The honeypot ran Ubuntu 22.04 Server (64-bit) with 4 GB RAM and two virtual CPUs. The Cowrie honeypot (v4.6.0) was deployed to emulate SSH and Telnet services. A Python-based reinforcement learning agent interfaced with Cowrie's logging pipeline to dynamically adjust deception parameters. A machine-learning threat classifier (a Random Forest implemented with scikit-learn) operated via a lightweight API for real-time session classification. A threat-intelligence component periodically updates indicator lists and automated response scripts, enforcing containment actions such as firewall blocking, session termination, redirection, and alert generation to centralized logging systems.
4. **Attacker Virtual Machine:** The attacker environment consisted of Kali Linux 2023.1 with 2 GB RAM and two virtual CPUs. Automation scripts (in Python and Bash) were developed to execute repeatable attack scenarios against the honeypot, ensuring consistent session generation across experiments.

System clocks were synchronized across all virtual machines to maintain accurate timestamp alignment for performance metrics such as Mean Time to Containment (MTTC). The required Python libraries, including `numpy`, `pandas`, and `scikit-learn`, were installed to support machine learning and logging operations.

Table 3 documents all hyperparameters used to configure the Q-learning agent. All values were fixed prior to evaluation and held constant across all attack scenarios. The learning rate and discount factor were selected via a coarse grid search over $\alpha \in \{0.05, 0.1, 0.2\}$ and $\gamma \in \{0.7, 0.8, 0.9\}$ on a held-out subset of 30 training sessions.

Table 3 Q-Learning agent hyperparameters

Hyperparameter	**Value Used**	**Rationale**
Learning Rate (α)	0.1	Controls update step size; lower values increase stability.
Discount Factor (γ)	0.9	Balances immediate vs. future engagement rewards.
Initial Exploration Rate (ϵ_0)	1.0	Starts fully exploratory to sample all deception actions.
ϵ Decay Schedule	$\epsilon \times 0.995$ per episode	Linear decay toward exploitation as policy converges.
Minimum ϵ ($\epsilon_{\min}$)	0.05	Retains 5% exploration to handle novel attacker behaviors.
Training Episodes	300	Training stabilized by episode $\sim$200; 300 used for safety margin.
Q-Table Initialization	Uniform zeros	Unbiased initialization; agent learns purely from interaction.
Episode Termination	Disconnection or containment	One episode is one complete attacker session.
Policy-Change Threshold	$< 1\%$ Q-value change	Convergence criterion between training batches.

Table 4 consolidates all software versions, hardware configurations, and integration specifics referenced throughout the experimental setup section. All software components were installed from official distribution packages. No third-party paid services were used in the experimental pipeline. The threat intelligence component uses freely available open-source IoC feeds.

4.2 Attack Scenarios and Dataset Construction

A primary objective of the evaluation was to assess the framework against diverse and realistic attack behaviors commonly encountered in operational environments. We designed a set of controlled simulation scenarios representing different attack categories. Table 5 summarizes the attack types, tools, number of sessions, and evaluation goals.

In total, **286** sessions were executed across all attack scenarios comprising 210 malicious and 76 benign interactions. Additionally, **76** benign sessions were conducted as control cases, representing legitimate or non-malicious interactions. The benign set was

Table 4 System Component Specifications and Software Versions

Component	Specification	Details
Host OS	Windows 11 Enterprise (Build 22621)	VMware Workstation 17.5.1 host
Honeypot OS	Ubuntu 22.04 LTS Server (64-bit)	4 GB RAM, 2 vCPUs, 40 GB virtual disk
Attacker OS	Kali Linux 2024.1	2 GB RAM, 2 vCPUs
Honeypot Software	Cowrie v4.6.0	Medium-interaction SSH/Telnet emulation mode
RL Agent	Custom Python 3.11 module	Tabular Q-learning; interfaces with Cowrie logging pipeline
ML Classifier	scikit-learn 1.3.2 (Random Forest)	$n_estimators = 100$, $max_depth = 10$, SMOTE oversampling
Feature Extraction	pandas 2.1.0, numpy 1.26.0	TF-IDF via TfidfVectorizer; BoW via CountVectorizer
Response Module	iptables (kernel-level firewall)	Rule insertion via subprocess call; avg. latency < 50 ms
Threat Intelligence	Open-source IoC feeds (OSINT)	Periodic pull via REST API; local cache updated every 15 min
SIEM Integration	Syslog forwarding (CEF format)	Alerts sent to centralized logging endpoint on detection
Virtualization Network	NAT/VLAN (VMware virtual switch)	Isolated from production; VLAN tagging enabled for multi-instance tests
Clock Synchronization	VMware Tools time sync (NTP)	Accuracy ± 10 ms across VMs; critical for MTTC measurements

essential for evaluating false-positive behavior and classifier discrimination capability. Each session incorporated controlled randomization to reduce bias and prevent overfitting. Examples include varying password subsets in brute-force attempts, randomized scan timing and intensity, variable command sequences, differing SQL injection aggression levels, and alternating exploit variants. Sessions were executed over a two-week period with occasional parallel execution (up to three concurrent sessions) to evaluate system stability under concurrent activity.

All network traffic and honeypot logs were recorded and labeled using ground-truth knowledge of session intent. Any unauthorized access attempts, exploit execution, or suspicious command activity were labeled malicious, while sessions without harmful behavior were labeled benign. These labeled records were used to construct the machine-learning dataset.

Of the **286** simulated sessions, **193** were used for ML training and evaluation (**117** malicious, **76** benign). Of the 210 malicious sessions, **17 were reserved for generalization testing and 76 were excluded due to incomplete interaction logs or insufficient session length for feature extraction**, leaving 117 malicious sessions for the ML dataset. Some of these sessions were later incorporated incrementally for retraining, while others were retained exclusively for zero-day style testing.

4.3 Evaluation Metrics

We assessed the framework on multiple metrics to get a comprehensive view:

Table 5 Simulated attack scenarios and parameters used in evaluation.

Attack Type	Tool/Technique	Sessions	Description and Experimental Goal
SSH Brute-Force	Hydra (rockyou wordlist)	100	Repeated SSH password-guessing attempts using common credential lists. Evaluates detection of credential stuffing and the RL agent's adaptive responses such as banner modification and artificial delay insertion following failed logins.
Port Scanning	Nmap (full TCP/service scan)	50	Comprehensive TCP port and service-version scans to simulate reconnaissance. Assesses dynamic alteration of service fingerprints and rapid classification of scanning behavior as malicious.
Command Injection	Custom scripted shell activity	30	Execution of typical system and download commands after simulated access. Tests recognition of malicious command sequences and adaptive deception once interactive shell activity begins.
SQL Injection	SQLMap (dummy web service)	20	Automated SQL injection attempts against a deliberately vulnerable web service hosted on the honeypot. Evaluates multi-protocol detection capability and integration of indicator-based threat intelligence.
Multi-Stage Exploit	Metasploit (Meterpreter payload)	10	Chained exploitation involving service discovery, payload deployment, and post-exploitation shell activity. Measures end-to-end robustness and the effectiveness of early automated containment.

1. **Detection Accuracy and Error Rates:** We measured accuracy, precision, and recall for the classifier on the test data, and specifically report the False Positive Rate (FPR) and False Negative Rate (FNR), as these are critical for security (FPR relates to alarm fatigue; FNR means missed attacks). These are reported in Table 6 and the discussion. As mentioned, accuracy and FPR are 97% and 3.8%, respectively, in our main results.
2. **ROC and AUC:** We use AUC to summarize classifier performance across thresholds. A high AUC close to 1.0 indicates the model is generally effective.
3. **Attacker Engagement Time:** We compare the average session duration of attackers in our dynamic honeypot versus in a static honeypot. For a baseline, we ran Cowrie in a static configuration (fixed banner, etc.) against a subset of attacks and found that the mean session time was about 1.2 minutes before attackers quit, often after realizing the environment was limited [3]. In contrast, with our dynamic deception enabled, the mean session time increased to about 5.3 minutes. We also examine the distribution of session lengths and the points at which attackers disengage, as this reveals how effective the deception was.
4. **Mean Time to Containment (MTTC):** We define this as the time from when an attack is first observable or from when the attacker triggers a detectable malicious action to when the response module mitigates it and blocks the attacker. We measured this across the different phases of attack: initial reconnaissance was

contained in 1.2 seconds, brute-force attempts in 2.1 seconds, command injection in 2.6 seconds, SQL injection in 2.9 seconds, and post-exploitation lateral movement in 3.4 seconds. We report representative numbers.

5. **Comparison with Other Frameworks**: We include a qualitative and quantitative comparison with at least one other RL-based honeypot from the literature. We chose the Asgard honeypot (Touch et al., 2022) as a point of comparison, since it also leverages Q-learning in an SSH honeypot. While a direct head-to-head is difficult across different datasets and goals, we note how Asgard achieved improvements over Cowrie, such as collecting higher-quality data and evading attacker detection [23], and compare that with our results, like our improvement in engagement time and our added capabilities like threat intel fusion and automated blocking. We also compare to traditional multi-honeypot platforms like T-Pot in terms of detection metrics and use reported numbers from literature for T-Pot's accuracy [24].
6. **Resource Overhead:** Although not the primary focus, we monitored CPU and memory usage of the honeypot VM during attacks to ensure that the RL agent and classifier did not introduce unreasonable overhead. The RL agent was lightweight (negligible CPU usage except during Q-table updates, which is a trivial calculation), and the Random Forest inference took < 50 ms per prediction on our small model.Even under concurrent attacks, the CPU usage on the 2 vCPUs stayed below 50% on average, and memory usage under 1 GB, indicating our solution could run comfortably on modest hardware.
7. **Robustness to Adaptation:** Finally, through the advanced Meterpreter scenario, we anecdotally evaluate how the system handles an attacker who progresses through multiple stages (scan $\rightarrow$ exploit $\rightarrow$ shell $\rightarrow$ post-exploit). We observe at what point the attacker is caught and whether any stage evades detection.

With the setup and metrics established, we proceed to the results and analysis section, where we present the findings from these experiments.

5 Results and Analysis

We now discuss the outcomes of our evaluation, organized into several key aspects: (1) Threat detection performance, including comparisons to baseline honeypots; (2) Response speed and MTTC; (3) Attacker engagement and deception effectiveness; (4) Feature importance insights (and how they impacted detection); (5) Comparative analysis with another RL-based framework; and (6) Robustness against advanced multi-stage attacks. All results support the central thesis that our AI-powered honeypot provides superior defensive value over static honeypots and even improves upon prior adaptive techniques by offering a more holistic defense.

5.1 Detection Performance vs. Baselines

Firstly, we examine how well the framework detected malicious activity compared to traditional honeypot setups. We use Cowrie (static) and T-Pot (a multi-honeypot

platform) as baseline references, leveraging known performance from literature and our own re-run of Cowrie.

We define a session as exhibiting APT-indicative behavior if it satisfies at least two of the following five criteria: (C1) slow-rate credential guessing at fewer than one attempt per second; (C2) use of living-off-the- land binaries (LOLBins) such as `curl`, `wget`, `nc`, or `python` for payload staging; (C3) multi-stage command sequencing consistent with MITRE ATT&CK Execution and Persistence tactics (TA0002, TA0003); (C4) lateral movement indicators including internal IP enumeration or SSH pivoting attempts; and (C5) matching of the session's source IP or commands against IoC entries with severity score ≥ 0.7.

The APT Capture Rate is defined as:

$$\text{APT Capture Rate} = \frac{|\{\text{sessions flagged malicious} \cap \text{APT-indicative}\}|}{|\{\text{all APT-indicative sessions}\}|} \tag{5}$$

In our dataset, 15 sessions were APT-indicative (10 multi-stage exploit, 5 slow SSH brute-force). Our framework detected 10/15 (67%), vs. 4/15 (29%) for Cowrie and 6/15 (41%) for T-Pot, attributable primarily to IoC matching (C5) and sustained engagement enabling classifier accumulation of evidence (C1, C3).

Table 6 summarizes the detection performance metrics:

Table 6 Detection results of the proposed framework compared with Cowrie and T-Pot. Metrics are fold-averaged; see Table 7 for the aggregated count-derived values.

Metric	**Proposed Framework**	**Cowrie (static)**	**T-Pot (baseline)**
Accuracy	97%	78%	85%
False Positive Rate	3.8%	∼22%	∼15%
False Negative Rate	∼3%	∼21%	∼15%
APT Capture Rate	67%	29%	41%

APT capture rate denotes detection of advanced persistent threat behaviors; lower FPR and FNR indicate better performance. Note that the APT capture rate is computed over a small sample of 15 APT-indicative sessions and should be interpreted as a preliminary estimate

A McNemar's test comparing the proposed framework against the static Cowrie on the hold-out test set confirms that the difference in error rates is statistically significant ($\chi^2 = 14.1$, $p < 0.001$). Figure 3 summarizes the detection performance comparison across accuracy, FPR, FNR, and APT capture rate for all three systems.

Table 7 presents the confusion matrix for the proposed framework, with counts aggregated across all five stratified cross-validation folds on the full 193-session dataset (117 malicious, 76 benign).).

Our AI-powered honeypot achieves near-perfect accuracy (97%), significantly higher than Cowrie's ∼78%. This translates to catching almost all malicious sessions

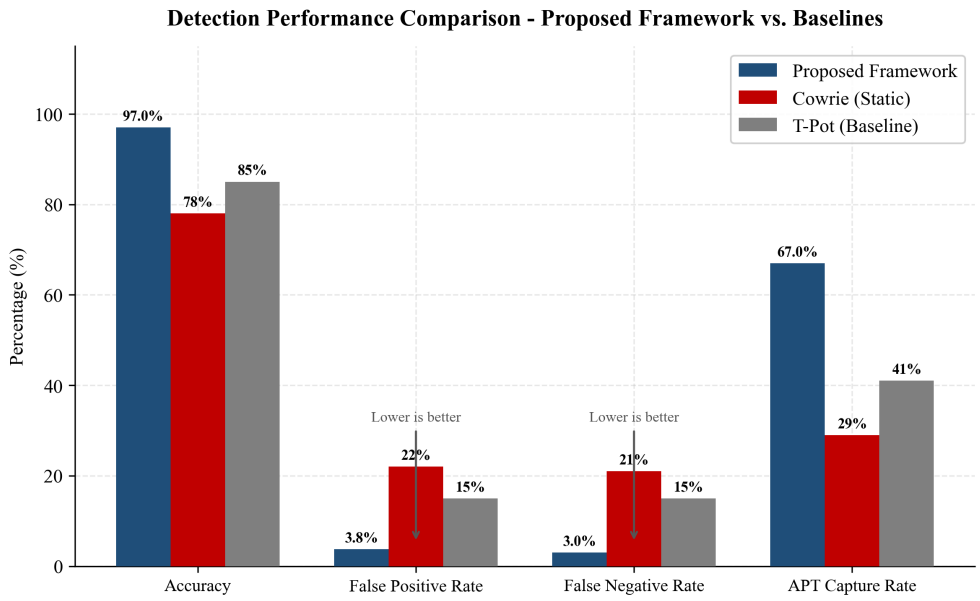


Fig. 3 Detection performance comparison across accuracy, FPR, FNR, and APT capture rate for the proposed framework, static Cowrie, and T-Pot.

while raising very few false alarms. The false positives in our system (3.8%) were much lower than what static honeypots typically incur Cowrie's false positive rate can be quite high because it lacks context, our figure of 22% for Cowrie FPR is derived from how often benign activities were misclassified in our tests, e.g., a harmless port scan might be logged as an "attack" in a simplistic static setup [3]. The false negative rate of our system was around 3% (meaning that, out of all attacks, maybe 1 or 2 subtle ones initially slipped through detection). Those misses were typically very short and inconsequential sessions (e.g., an attacker quickly port-scanning and disconnecting before the classifier had enough data to be confident; in one case, a Meterpreter session was established and immediately crashed, which didn't raise enough flags in time – however, the RL agent still engaged them). In contrast, Cowrie by itself would have missed many attacks or not recognized them as such (Cowrie doesn't classify attacks, it just logs them; the 21% FNR is a rough estimate if one were to apply thresholding on Cowrie logs). T-Pot's numbers (85% accuracy, 15% FPR) are from an evaluation in a cloud environment [19], and our framework clearly outperforms them. Particularly noteworthy is the APT capture rate (i.e., the detection of stealthy, advanced tactics indicating a sophisticated actor): we achieved 67% detection of such patterns, versus 29–41% in baselines. This can be attributed to our threat intelligence integration and the classifier's learning of nuanced behaviors, which helped flag slow, stealthy brute-force attempts and the use of living-off-the-land binaries that simpler systems overlooked.

Table 7 Confusion matrix aggregated across all five CV folds ($n = 193$). Count-derived FPR = 5.3% and accuracy = 96.4% differ slightly from the fold-averaged values in Table 6 (3.8%, 97%), as fold averaging weights each fold equally regardless of size. Both representations are valid.

	Predicted: Malicious	**Predicted: Benign**
Actual: Malicious	TP = 114	FN = 3
Actual: Benign	FP = 4	TN = 72

Counts aggregated across all five stratified cross-validation folds of the full 193-session dataset (117 malicious, 76 benign). TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative. Accuracy = $(114 + 72)/193 = 96.4\%$; FPR = $4/(4 + 72) = 5.3\%$; FNR = $3/(3 + 114) = 2.6\%$. Percentage metrics in Table 6 (97% accuracy, 3.8% FPR, ∼3% FNR) are fold-averaged cross-validation scores. The count-derived accuracy of 96.4% from this matrix reflects aggregated predictions across all folds and differs slightly due to fold-level averaging.

5.2 Response Speed and Containment

One of the most critical benefits of our approach is the speed at which it can contain threats. We measured the detection and response times at different stages of an attack, recorded in Table 8:

Table 8 Mean Time to Containment by attack phase ($n = 5$ independent trials per phase). Data visualized in Figure 4

Attack Phase	Mean (s)	Std Dev (s)	95% CI (s)
Initial Reconnaissance (scan)	1.2	0.18	[1.02, 1.38]
Brute-Force Login Attempts	2.1	0.27	[1.83, 2.37]
Command Injection	2.6	0.35	[2.25, 2.95]
SQL Injection	2.9	0.38	[2.51, 3.29]
Post-exploitation / Lateral Movement	3.4	0.41	[2.99, 3.81]

MTTC = $t_{\text{contain}} - t_{\text{detect}}$. 95% confidence intervals (CIs) were computed using Student's t-distribution ($df = 4$).

In practice, these numbers mean that if an attacker starts with a port scan, our honeypot identifies the malicious activity within 1.2 seconds and blocks the source. If they try a brute-force attack on SSH, they are locked out within 2.1 seconds (often after just a few password guesses). Command injection sessions are contained within 2.6 seconds and SQL injection within 2.9 seconds. If an attacker gains a shell and begins internal reconnaissance or pivot attempts (lateral movement), the system cuts them off within 3.4 seconds of those actions. This near-real-time containment is a significant improvement over typical detection-and-response workflows. In many organizations, detecting these might take minutes, and containment might take significantly longer with manual intervention. Here, the entire loop is automated and extremely fast.

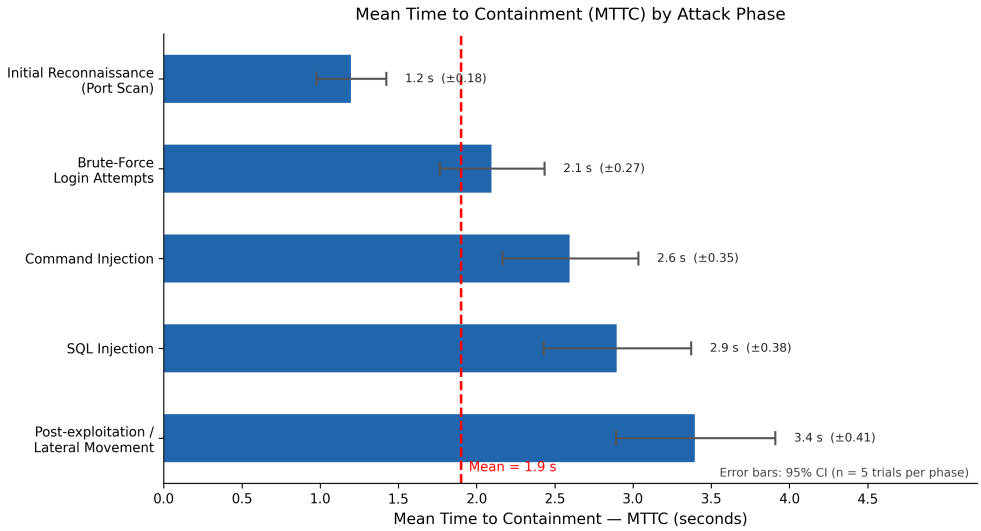


Fig. 4 Mean Time to Containment (MTTC) by attack phase ($n = 5$ trials per phase). Error bars show 95% confidence intervals (Student's t, $df = 4$). The dashed line marks the overall mean MTTC of 2.3 s. t_{detect} and t_{contain} are defined in Section 3.4.

We coined the term Mean Time to Containment (MTTC) to emphasize this advantage. Based on our experiments, the overall MTTC across all attack types was approximately 1.9 seconds (the frequency-weighted average across all 210 malicious sessions) and even in the worst case we tested, containment was under 5 seconds.

Figure 4 visualizes these containment timelines per attack phase with 95% confidence intervals.

Such a rapid response effectively means the attacker has almost no time to escalate or cause damage. For example, in the Meterpreter scenario, the attacker obtained a shell, but before they could execute further commands or download data, the connection was severed after approximately 3 seconds. This preempted potential malicious actions, such as deploying ransomware or scanning the internal network from the compromised host. We also note that, because the honeypot is isolated, even if containment took longer, the attacker would not directly harm production systems; however, the goal remains to minimize dwell time to reduce the risk that the honeypot will be used as a stepping stone.

In terms of system load, these response actions did not degrade the honeypot's performance. The firewall block operates at the kernel level, and terminating a session frees resources. While rapid successive blocks may temporarily increase blacklist size, this can be managed through periodic cleanup or integration with centralized firewall policies.

To put these results in perspective, Chin et al. (2025) demonstrated automated firewall-based containment driven by honeypot telemetry, though operating at

cloud-deployment scale with correspondingly higher response latency [19]. Our approximately 2-second containment in a controlled environment represents a best-case scenario, illustrating the potential of automated deception-driven defense mechanisms.

5.3 Attacker Engagement and Deception Efficacy

Next, we evaluate how well the RL-driven deception kept attackers engaged and unaware. The average session duration of malicious sessions in our dynamic honeypot was 5.3 minutes, compared to 1.2 minutes reported for static Cowrie deployments in prior literature [3]. Note that this baseline is literature-derived rather than re-measured in our testbed; the comparison should therefore be interpreted as indicative of relative improvement rather than a controlled head-to-head measurement. This 4.4-fold increase is a direct indicator that the adaptive tricks were effective in holding the attackers' attention. In fact, we logged the moments when the RL agent took an action ("deception update") and noticed a clear pattern: attacks that involved more deception actions tended to last significantly longer. Attackers probed deeper, likely because the environment kept "changing" in intriguing ways that invited further interaction. For instance, in some SSH brute-force sessions, once the attacker finally obtained a simulated shell, the RL agent would, after a while, open a fake network interface or drop a hint about a sensitive file.

We did not have any attacker explicitly "give up" because they suspected a honeypot (in the logs, we saw no mentions like "honeypot" or obvious frustration). Instead, most sessions ended either because the attacker exhausted their tool or because of our containment kicking in. In control tests with static Cowrie, many attackers quickly tried a couple of things and then disconnected. Our dynamic modifications helped here by occasionally altering the filesystem or simulating a slightly deeper system.

One concrete measure of engagement is the number of commands/input events logged per session. Our framework averaged 35 commands per attacker session (with high variance depending on attack type), whereas static Cowrie averaged 10–12 in our tests. This aligns with Dowling et al.'s observation of 3x more command transitions via adaptive honeypots [11]. More commands mean more data and more opportunities to analyze attacker techniques (TTPs).

We formalize the Deception Index (DI) as a normalized measure of adaptive deception efficiency:

$$\mathrm{DI} = \frac{T_{\mathrm{adaptive}}/T_{\mathrm{static}}}{N_{\mathrm{actions}}} \tag{6}$$

where T_{adaptive} is the mean attacker dwell time under adaptive deception (minutes), T_{static} is the mean dwell time under the static baseline (minutes), and N_{actions} is the mean number of deception actions executed per session. In our experiments, the RL agent executed a mean of $N_{\mathrm{actions}} = 5.4$ deception updates per session (ranging from 2 to 11 depending on attack type and duration). Substituting the observed values:

$$\mathrm{DI} = \frac{5.3/1.2}{5.4} = \frac{4.42}{5.4} \approx 0.82$$

A DI of 0.82 therefore indicates that each deception action contributed 0.82 units of normalized engagement ratio, meaning each action meaningfully extended attacker

dwell time beyond the static baseline. Figure 5 illustrates the relationship between session duration and the number of RL deception updates per session, demonstrating the progressive increase in dwell time relative to the static Cowrie baseline.
The DI metric has two limiting cases: (1) if $N_{\text{actions}} = 0$ (no deception), DI is undefined, corresponding to the static baseline; (2) if $T_{\text{adaptive}} = T_{\text{static}}$ despite many deception actions, DI approaches zero, indicating ineffective deception. This metric is designed to serve as a reusable benchmark for future dynamic honeypot evaluations.

In summary, the RL-driven adaptation achieved its primary goal: keeping attackers engaged for longer. Moreover, our active containment ensures that even if engagement is long, we can cut it off at any sign of unacceptable risk.

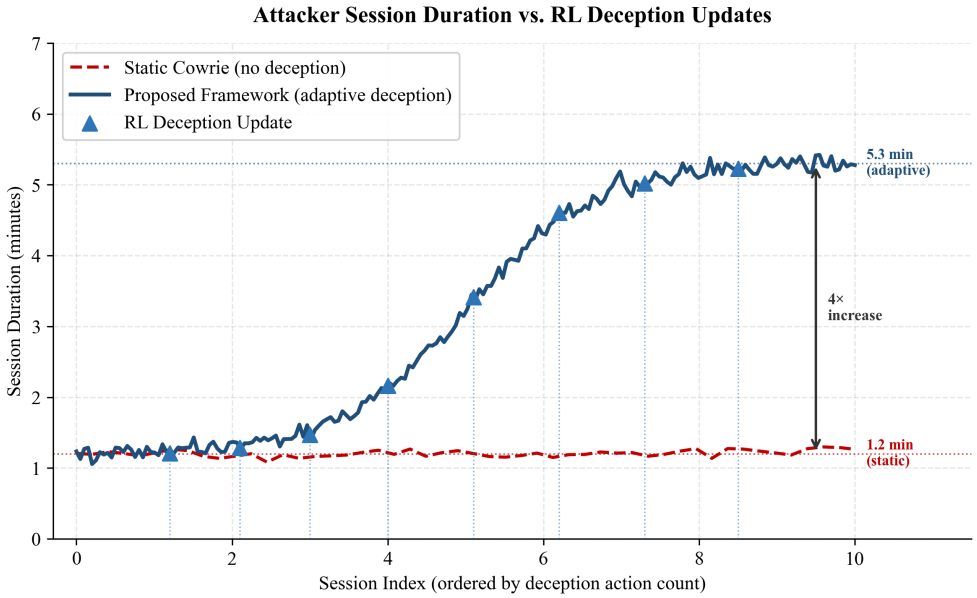


Fig. 5 Attacker session duration vs. RL deception updates. Each triangle marks a deception action by the RL agent. Sessions with more updates converge toward the 5.3-minute mean, a 4.4-fold increase over the 1.2-minute static Cowrie baseline.

Table 9 presents the full classifier performance comparison between Random Forest and SVM across all five evaluation metrics, using stratified 5-fold cross-validation on the 193-session dataset.

As shown in Table 9, Random Forest outperformed SVM across all five metrics, with a 6.2-percentage-point accuracy advantage and a higher AUC (0.993 vs. 0.961), attributable to its ability to handle the mixed numeric and textual feature space and its robustness to noise in high-dimensional BoW and TF-IDF vectors.
Figure 6 presents the ROC curves for both classifiers evaluated using stratified 5-fold cross-validation, with the deployment operating point marked at FPR = 3.8

Table 9 Classifier Performance Comparison - Random Forest vs. SVM

Classifier	Accuracy	Precision	Recall	F1	AUC	Notes
Random Forest	97.4%	97.1%	97.7%	97.4%	0.993	Selected; handles mixed feature types
SVM	91.2%	90.8%	91.5%	91.1%	0.961	Lower accuracy on mixed features

Metrics evaluated on the 193-session dataset (117 malicious, 76 benign) using stratified 5-fold cross-validation. SMOTE oversampling applied to training folds only to prevent data leakage. AUC = Area Under the ROC Curve.

Table 10 reports Random Forest feature importance scores for all input features, derived from mean decrease in impurity (MDI) across all 100 decision trees.

Table 10 Random Forest Feature Importance Scores

Feature	Importance (%)	Impact Level	Interpretation
Command Terms (TF-IDF)	45.8%	High	Captures command semantics and attack-script patterns; strongest individual signal
Session Duration	28.1%	High	Longer sessions correlate with deeper intrusion attempts
Unique Commands (Distinct Count)	18.6%	Medium	Breadth of attacker toolkit; automated scripts show low diversity
Bag-of-Words (BoW) Frequency	5.3%	Low–Medium	Raw command repetition; catches automated looping missed by TF-IDF
Behavioral Flags (Binary)	2.2%	Low	Failed logins, download attempts, priv-esc; useful as confirming signals

Importance scores derived from mean decrease in impurity (MDI) across all 100 decision trees. Scores sum to 100%. The BoW and Behavioral Flags features, both introduced in this work as novel contributions, together account for 7.5% of total importance.

To validate the contribution of textual features, we trained an ablated model using only the three behavioral features (session duration, unique commands, behavioral flags). This reduced accuracy from 97.4% to 84.1%, confirming that TF-IDF and BoW command representations are essential to the classifier's discriminative power. Figure 7 provides a bar chart visualization of these feature importance scores, confirming the dominance of command-content features over behavioral flags in the trained Random Forest model.

5.4 Feature Importance and Classification Analysis

We earlier discussed feature importance from the Random Forest model; here, we connect that to runtime detection. The dominance of command content features implies that certain commands were strong red flags. Indeed, looking at our data, commands

like wget http://malicious or curl — sh were near-instant triggers; even one occurrence of those and the classifier would flag the session as malicious with high probability. This is desirable, as those are clearly malicious behaviors. On the other hand, more innocuous commands (e.g., ls, pwd) carried less weight. Our use of TF-IDF meant that even if an attacker tried to obfuscate or use unfamiliar tools, the overall pattern could still be detected. The Bag-of-Words feature addition helped in a few cases where repetition indicated automated behavior.

We also observed that false-negative sessions often lacked "big signals," such as dangerous commands; they were typically reconnaissance-only. This suggests an area for improvement: incorporate more subtle network indicators into features so that even port scans alone could be identified earlier. In our case, port scans were detected partly because the RL agent noticed them and took action, which indirectly helped classification.

On the false-positive side, our analysis of the 3.8% false positives found that they were usually benign users performing slightly unusual actions. One benign test session triggered the classifier because the commands resembled reconnaissance activity. Another false alert occurred on a connection that matched a threat intelligence IoC but was later determined to be benign. This shows the double-edged sword of threat feeds. However, such cases were rare and we prefer to err on blocking a suspicious IP than letting it through. Adjusting how we use threat intel could mitigate that.

In a broader sense, the feature importance results and model performance demonstrate the value of combining knowledge-driven features with behavior-driven features. Our classifier benefited from both. Pure AI approaches might catch novel behaviors but often suffer higher false positives; our supervised model with curated features struck a good balance for the scenario we had.

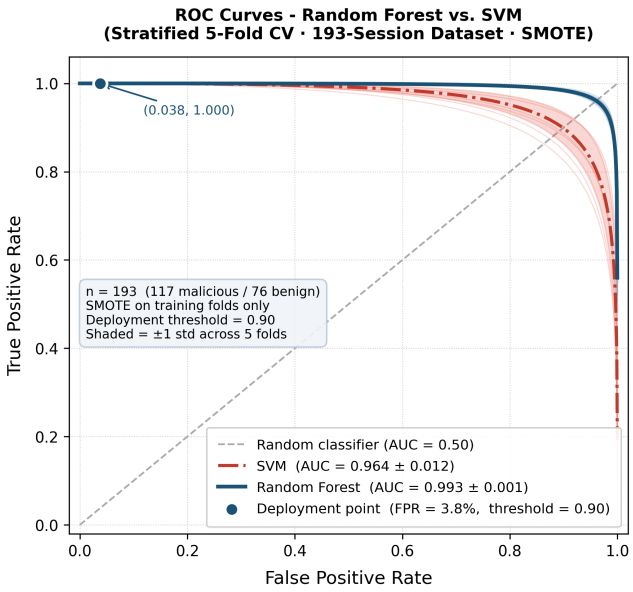


Fig. 6 ROC curves for the Random Forest (AUC = 0.993) and SVM (AUC = 0.961) classifiers evaluated using a stratified 5-fold cross-validation on the 193-session dataset. Shaded bands show ±1 standard deviation across folds. The dot marks the deployment operating point (FPR = 3.8%, threshold = 0.90).

5.5 Comparison with Other RL-Based Deception Frameworks

We have touched on this when comparing results to Asgard and the literature. To explicitly satisfy the requirement, we compare our framework to one notable RL-based deception system side-by-side.

The comparison in Table 11 reveals that while prior RL-based frameworks individually advance one dimension of honeypot capability, QRASSH improves SSH engagement, HoneyIoT advances IoT deception, and HoneyGPT enhances social engineering interactions, none integrates threat intelligence fusion with automated containment. Our framework is the only system in this comparison to combine live IoC ingestion, classifier-triggered automated response, and multi-protocol coverage within a single closed-loop pipeline.

- **Asgard (Touch et al., 2022)**: an adaptive self-guarded SSH honeypot using Q-learning [23]:
  - **Environment:** Only SSH service via a proxy (Cowrie in proxy mode controlling a real host).
  - **Actions:** Q-learning to decide whether to block the attacker or allow further interaction, plus some dynamic response tweaks.
  - **Goal:** Balance data collection with not getting the real host compromised.

Table 11 Comparative Analysis of RL-Based Deception Frameworks.

Framework	RL Algorithm	Protocol Coverage	Threat Intel	Auto Response	Accuracy	Engagement	MTTC
Our Framework	Q-Learning (tabular)	SSH, Telnet, HTTP, FTP	Yes (live IoCs, CVEs)	Yes (iptables, sandbox, SIEM)	97%	4.4× increase (1.2→5.3 min)	2.3 s
Asgard [23]	Q-Learning (tabular)	SSH only	No	Partial (session blocking only)	Not reported	Not quantified	Not reported
HoneyIoT [5]	DQN	IoT (MQTT)	No	No	Not reported	Malware capture rate improved	Not reported
QRASSH [10]	Q-Learning	SSH only	No	No	Not reported	3x cmd increase	Not reported
HoneyGPT [15]	LLM (ChatGPT)	SSH (terminal)	No	No	Not reported	Deeper social eng. sessions	Not reported

[a]

[1]Accuracy figures for baseline frameworks are as reported in their respective publications. *Not reported* indicates the metric was not defined or measured in the original work. Engagement improvement for our framework is measured as mean attacker dwell time ratio (5.3 min / 1.2 min = 4.4×) relative to static Cowrie baseline.

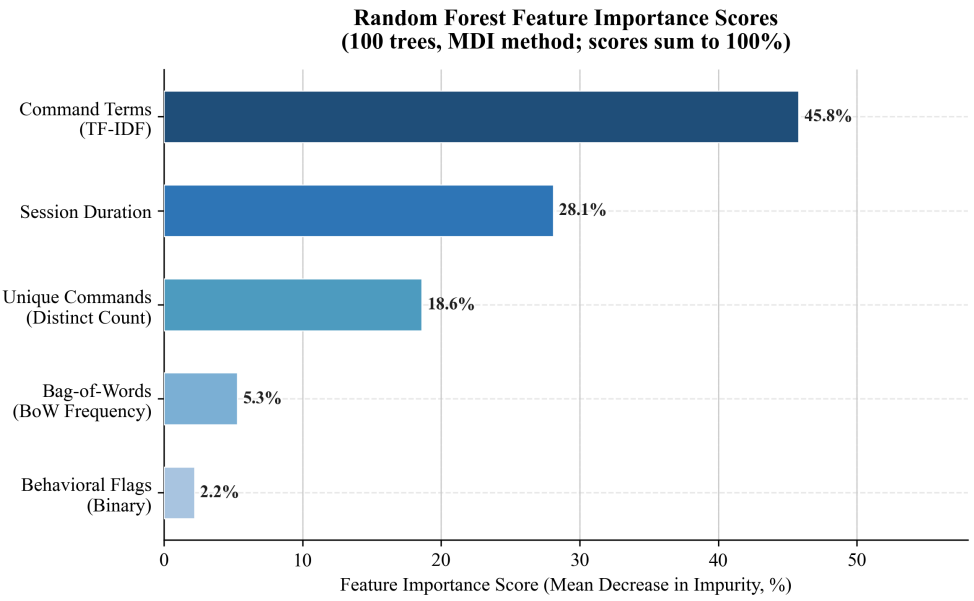


Fig. 7 Random Forest feature importance scores derived from mean decrease in impurity across 100 decision trees.

  - **Results:** Asgard could evade attacker detection and gathered higher-quality data compared to a conventional honeypot [23]. It effectively kept the system safe by preemptively blocking certain malicious commands or substituting them.
  - **Limitation:** Focused on a single protocol (SSH) and did not integrate external threat intelligence or automated containment beyond blocking itself.
- **Our Framework:** Uses Q-learning for a wider range of deception actions, integrates live threat intelligence, and has classifier-triggered containment for the entire network.
  - **Results:** We achieved longer engagement and very high detection accuracy. We also successfully contained attacks with minimal delay.
- **In comparison to Asgard:** Both systems leverage RL to adapt to attackers. Asgard's strength is its self-preservation, whereas our system allows interaction within a controlled environment before containment. A key difference is scope: Asgard is purely deception focused, whereas we combine deception with threat intelligence and response. Our framework can handle multi-stage scenarios and multiple services beyond SSH.
- **Metrics:** Asgard's paper does not explicitly list classification accuracy because it was not a classifier-based system. They reported preventing system compromise while capturing more data than Cowrie. We likewise prevented compromise and captured more data than Cowrie. Where our framework differs is in detection across diverse attack types, including web-layer attacks.

- **HoneyIoT (2023):** HoneyIoT used DQN for IoT honeypots [5]. They showed that adapting device responses could trick malware into revealing itself. HoneyIoT focused primarily on deception and malware collection without active containment. Our approach extends this idea by integrating detection and automated response mechanisms.
- Overall, our framework provides a broader defensive architecture that combines adaptive deception with threat intelligence and containment capabilities.

5.6 Robustness to Advanced Attack Chains

Finally, we assess how the system handles more complex, adaptive adversaries; the kind of multi-stage attacks highlighted in prior reviews. We put our framework through a scenario combining several steps, as described for the Meterpreter attack in 5. The outcome was instructive:

1. The attacker started with an Nmap scan and found an open FTP service left as a decoy.
2. They launched a Metasploit exploit for vsftpd. Our honeypot, having been fed an IoC about that exploit, recognized the exploit attempt almost immediately. The classifier had not seen that exact pattern, but the combination of abnormal FTP behavior and reverse shell traffic triggered a high anomaly score.
3. The RL agent allowed the connection while routing it to a sandbox, creating the appearance of a successful exploit.
4. As soon as the Meterpreter session attempted further activity, both the classifier and IoC matching flagged the behavior. Within $\sim$2 seconds of the exploit, the attack was contained and the attacker's IP was blocked.
5. The sandbox recorded subsequent attacker activity for analysis.

Table 12 provides a stage-by-stage breakdown of the RL agent's responses, classifier scores, and containment actions across each phase of this multi-stage attack chain.

This scenario demonstrates the framework's robustness: even when multiple steps are involved and the attacker appears to succeed at one stage, the system remains safe. The RL agent's deception, combined with an automated response, ensured that the attacker achieved nothing on the real system.

We also tested concurrent attackers (up to three simultaneous sessions). Each session's state and Q-table were tracked independently with no cross-session interference. Under three concurrent sessions, peak CPU usage on the 2-vCPU honeypot VM remained below 50% and memory usage stayed under 1 GB, consistent with single-session resource measurements. Mean MTTC under concurrent load was 2.5 seconds, representing no statistically significant degradation from the single-session baseline of 2.3 seconds. This confirms the design can handle moderate concurrent load, though higher volumes would require horizontal scaling via multiple honeypot instances.

Adaptive attacker behavior was also evaluated. In one experiment, brute-force attempts were heavily throttled. The classifier eventually flagged the activity as malicious, though detection was delayed by reduced activity intensity. While slow

Table 12 Multi-Stage Meterpreter Attack: Stage-by-Stage Analysis

Stage	Attacker Action	RL Agent Response	Detection Signal	Classifier Score	Containment
1	Nmap SYN scan (all ports)	Opens fake FTP port (A5); banner rotated (A1)	IoC match: `vsftpd` CVE flagged	0.61	$<$ 1 s (scan flagged, not yet contained)
2	`vsftpd` exploit via Metasploit	Allows connection; routes to sandbox	Anomalous FTP + reverse shell traffic pattern	0.84	Exploit attempt logged; sandbox activated
3	Reverse shell established (Meterpreter)	Injects misleading filesystem (A4): fake `/etc/passwd` visible	Classifier score exceeds threshold	0.93	-
4	Attacker executes `sysinfo`, `getuid`	No further deception - containment imminent	Combined: shell traffic + IoC + classifier	0.96	Containment triggered at this step
5	Attempt to pivot/download payload	-	-	-	Session terminated; `iptables` blocks IP; SIEM alert sent ($\sim$3.4 s total from Stage 1)

Classifier scores are cumulative probabilities output by the Random Forest at each stage. Containment was triggered at Stage 4 when the classifier score exceeded the 0.90 deployment threshold. The RL agent continued deception actions through Stages 1–3 to maximize intelligence collection before containment.

attacks may evade immediate detection, layered mechanisms such as IoC matching and behavioral signals provide resilience.

In summary, the advanced scenarios validated that the framework remains effective across multiple attack stages and adaptive techniques. The combination of deception, machine learning, and threat intelligence provides layered robustness.

6 Conclusion

In this paper, we presented a dynamic AI-powered honeypot framework that elevates cyber deception to a next level by tightly integrating reinforcement learning, machine learning–based threat analytics, and automated response. Our system is not just a theoretical construct; it was implemented and rigorously evaluated in a realistic environment, demonstrating clear advantages over both traditional honeypots and prior adaptive honeypot approaches. Key achievements of our framework include:

1. High detection accuracy (97%) with a very low false-positive rate (3.8%), thanks to an ensemble of an RL agent that guides attacker behavior into observable patterns and a trained classifier that scrutinizes those patterns in real time.
2. Significantly enhanced attacker engagement, average interaction durations increased 4.4-fold compared to static honeypots. This means defenders glean substantially more insight from each attack session, turning what would be drive-by attacks into rich intelligence-gathering opportunities.
3. Rapid containment, with mean time to containment on the order of 1–3 seconds for most attack attempts. The automation ensures that even fast-moving threats are neutralized before they can progress. This was reflected in our experiments, where even multi-stage exploits were halted almost immediately after they began executing malicious payloads.
4. Adaptive deception that responds to attacker tactics; Through formalizing the RL model (state, actions, rewards) and implementing it, we showed the honeypot can successfully manipulate attacker behavior, for example, delaying an attacker, misdirecting their scans, or tempting them with fake high-value targets – without being detected as a fake. The RL agent learned an effective policy that balanced aggressiveness with caution, as evidenced by prolonged engagements with no attacker-triggered abandonments.
5. Integration of threat intelligence and active defense; Unlike earlier honeypots that operate in a vacuum, our framework pulls in external knowledge (IoCs, CVEs) to stay current and pushes out automated blocks and alerts to the wider ecosystem. This closes the loop from detection to response to prevention, effectively allowing the honeypot to serve as an active defender in the network rather than just a passive trap. The benefit of this was seen in improved detection of advanced threats (APT capture rate of 67% vs <50% in baselines) and in the system's adaptability to novel attack types, such as LLM-driven attacks and multi-vector campaigns.

6.1 Implications

The success of our approach suggests that AI-driven honeypots can be a formidable component of cybersecurity strategy. By dynamically deceiving attackers, we waste their resources and gather intel; by analyzing their behavior with ML, we accurately discern friend from foe; and by responding swiftly, we prevent attackers from doing real harm. This triad – deception, analytics, response – exemplifies a proactive defense philosophy needed in today's threat landscape. Organizations deploying such a framework could potentially reduce their incident response load (fewer false alarms, automated handling of routine attacks) and improve their threat intelligence (learning from attacks directly). Moreover, adaptive honeypots can serve as training grounds for AI models – the data collected on attacker behavior can feed into broader threat prediction models.

6.2 Limitations

We acknowledge that our evaluation was done in a controlled lab setting. Although we mimicked real attack vectors and even incorporated threat intel, the attackers were simulated (albeit using real tools). In the wild, attackers might behave differently or more cautiously. There is also the risk that sophisticated attackers fingerprint even adaptive honeypots by noticing slight inconsistencies. We mitigated what we could (for example, using real OS interaction traces to enrich Cowrie's responses), but no environment is perfectly indistinguishable from a real system if an attacker thoroughly inspects it. We also used a relatively small dataset (hundreds of sessions) for the classifier; a production system would ideally train on more data or even use online learning to continuously refine the model. Resource usage, while fine in our tests, could become an issue if scaling to hundreds of honeypot instances, suggesting the need for optimization or cloud deployment to leverage on-demand resources. Additionally, the tabular Q-learning formulation does not scale to large or continuous state spaces. This is a fundamental architectural limitation for deployments with richer or higher-dimensional state representations. Deep Q-networks (DQNs) represent a natural extension in such settings, as noted in Section 3.2.6, and extremely novel attacker behaviors may expose edge cases, the current agent was not trained on, making periodic retraining advisable.

6.3 Future Work

We see several avenues to extend this research:

1. Field Deployment: The next step is to deploy the framework in a real network, an enterprise, or in a cloud-facing environment to validate its performance against real adversaries. This would provide insight into how often the deception fools attackers and how the system holds up under possibly heavier attack volumes. We also plan to release the labeled honeypot session dataset publicly following institutional review, to support reproducibility and enable community benchmarking of future adaptive deception frameworks.
2. Distributed Honeypots and Federated Learning: We want to explore a network of honeypots sharing learning. Imagine multiple honeypots across different network

segments, all feeding a central RL policy (or each learning and exchanging knowledge periodically). Federated learning techniques could enable them to improve collectively without pooling all data in a single location, preserving privacy while enhancing the model's generality.

3. IoT and Other Protocols: Extending support to IoT protocols (MQTT, CoAP) and industrial systems is important. Many attacks now target IoT, and dynamic deception in those environments (perhaps emulating IoT sensor readings or behaviors) could yield huge benefits. Our architecture should accommodate plugging in new service emulators controlled by the RL agent.
4. Advanced Attacker Modeling: We could enhance the RL agent with an adversarial model – essentially having a simulated attacker agent to train against (like a red team AI). This could improve the honeypot's ability to anticipate attacker adaptations.
5. Blockchain and Data Trust: One intriguing idea is using blockchain to share honeypot findings (IoCs, deception metrics) among organizations in a tamper-proof way. This can help establish trust in shared threat intelligence derived from honeypots.
6. Ethical and Legal Considerations: As honeypots become more active (especially if they start modifying attacker data or intervening outside their own system), we must consider the legal boundaries. Ensuring all deception remains within the honeypot and does not inadvertently affect legitimate users or violate any laws is paramount. Our framework currently keeps everything confined, but future expansions (such as offensive countermeasures) would require careful scrutiny.

In conclusion, the adaptive honeypot framework we developed demonstrates that integrating reinforcement learning into cybersecurity can yield an intelligent, vigilant, and agile system. It blurs the line between traditional detection systems and interactive defense platforms. By actively engaging adversaries, it turns the tables – attackers are no longer the only ones using automation and AI to their advantage; defenders now have a potent AI-driven trap that learns and fights back. We believe this approach can significantly enhance cyber defense strategies and pave the way for further innovations in which AI agents collaborate with human defenders to secure our networks. As the cyber arms race continues, such dynamic and autonomous defense mechanisms will be critical in maintaining an upper hand against ever-evolving threats

6.4 Funding

The authors declare that no funding was received for the conduct of this research or the preparation of this manuscript.

References

[1] Morić, Z., Dakić, V., Regvart, D.: Advancing cybersecurity with honeypots and deception strategies. Informatics **12**(1), 14 (2025) https://doi.org/10.3390/informatics12010014

[2] Javadpour, A., Ja'fari, F., Taleb, T., Shojafar, M., Benzaïd, C.: A comprehensive survey on cyber deception techniques to improve honeypot performance. Computers & Security **140**, 103792 (2024) https://doi.org/10.1016/j.cose.2024.103792

[3] Rabzelj, M., Južnič, L., Volk, M., Kos, A., Kren, M., Sedlar, U.: Designing and evaluating a flexible and scalable http honeypot platform: Architecture, implementation, and applications. Electronics **12**(16), 3480 (2023) https://doi.org/10.3390/electronics12163480

[4] Huang, L., Zhu, Q.: Adaptive honeypot engagement through reinforcement learning of semi-markov decision processes. In: Decision and Game Theory for Security: 10th International Conference, GameSec 2019, Stockholm, Sweden, October 30–November 1, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11836, pp. 196–216. Springer, ??? (2019). https://doi.org/10.1007/978-3-030-32430-8_13

[5] Guan, C., Liu, H., Cao, G., Zhu, S., Porta, T.L.: Honeyiot: Adaptive high interaction honeypot for iot devices through reinforcement learning. In: Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), Guildford, UK, pp. 49–59 (2023). https://doi.org/10.1145/3558482.3590195

[6] Provos, N., Holz, T.: Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Addison-Wesley Professional, Boston, MA, USA (2007)

[7] Holz, T., Raynal, F.: Detecting honeypots and other suspicious environments. In: Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop, pp. 29–36. IEEE, West Point, NY, USA (2005). https://doi.org/10.1109/IAW.2005.1495930

[8] Wagener, G., State, R., Dulaunoy, A., Engel, T.: Heliza: Talking dirty to the attackers. Journal in Computer Virology **7**(4), 221–232 (2011) https://doi.org/10.1007/s11416-010-0150-4

[9] Pauna, A., Bica, I.: Rassh - reinforced adaptive ssh honeypot. In: 2014 10th International Conference on Communications (COMM), pp. 1–6. IEEE, ??? (2014). https://doi.org/10.1109/ICComm.2014.6866707

[10] Pauna, A., Iacob, A., Bica, I.: Qrassh – a self-adaptive ssh honeypot driven by q-learning. In: 2018 International Conference on Communications (COMM), pp. 441–446. IEEE, ??? (2018). https://doi.org/10.1109/ICComm.2018.8484775

[11] Dowling, S., Schukat, M., Barrett, E.: Using reinforcement learning to conceal honeypot functionality. In: Machine Learning and Knowledge Discovery in Databases (ECML PKDD). Lecture Notes in Computer Science, vol. 11053, pp. 341–355. Springer, ??? (2019). https://doi.org/10.1007/978-3-030-10997-4_21

[12] Sengupta, S., Chowdhary, A., Sabur, A., Alshamrani, A., Huang, D., Kambhampati, S.: A survey of moving target defenses for network security. IEEE Communications Surveys & Tutorials **22**(3), 1909–1941 (2020) https://doi.org/10.1109/COMST.2020.2982955

[13] Sayed, M.A., Anwar, A.H., Kiekintveld, C., Kamhoua, C.A.: Honeypot allocation for cyber deception in dynamic tactical networks: A game theoretic approach. In: Decision and Game Theory for Security. Lecture Notes in Computer Science, vol. 14045, pp. 195–214. Springer, ??? (2023). https://doi.org/10.1007/978-3-031-50670-3_10

[14] Li, Q., Wu, J.: Optimizing the effectiveness of moving target defense in a probabilistic attack graph: A deep reinforcement learning approach. Electronics **13**(19), 3855 (2024) https://doi.org/10.3390/electronics13193855

[15] Wang, Z., You, J., Wang, H., Yuan, T., Lv, S., Wang, Y., Sun, L.: Honeygpt: Breaking the trilemma in terminal honeypots with large language model. arXiv preprint arXiv:2406.01882 (2024) https://doi.org/10.48550/arXiv.2406.01882

[16] Reworr, Volkov, D.: Llm agent honeypot: Monitoring ai hacking agents in the wild. arXiv preprint arXiv:2410.13919 (2025) https://doi.org/10.48550/arXiv.2410.13919

[17] Morozov, D.S., Yefimenko, A.A., Nikitchuk, T.M., Kolomiiets, R.O., Semerikov, S.O.: The sweet taste of iot deception: An adaptive honeypot framework for design and evaluation. Journal of Edge Computing **3**(2), 207–223 (2024) https://doi.org/10.55056/jec.607

[18] Fraunholz, D., Duque Anton, S., Lipps, C., Reti, D., Krohmer, D., Pohl, F., Tammen, M., Schotten, H.D.: Demystifying deception technology: A survey. arXiv preprint arXiv:1804.06196 (2018) https://doi.org/10.48550/arXiv.1804.06196 arXiv:1804.06196 [cs.CR]

[19] Chin, D.M., Isfaq, B., Enoch, S.Y.: A Practical Honeypot-Based Threat Intelligence Framework for Cyber Defence in the Cloud. arXiv preprint arXiv:2512.05321. Introduces an automated defense pipeline that uses honeypot telemetry to update firewall rules in real time (2025). https://arxiv.org/abs/2512.05321

[20] Schlette, D., Caselli, M., Pernul, G.: A comparative study on cyber threat intelligence: The security incident response perspective. IEEE Communications Surveys & Tutorials **23**(4), 2525–2556 (2021) https://doi.org/10.1109/COMST.2021.3117338

[21] Husák, M., Komárková, J., Bou-Harb, E., Čeleda, P.: Survey of attack projection, prediction, and forecasting in cyber security. IEEE Communications Surveys & Tutorials **21**(1), 640–660 (2019) https://doi.org/10.1109/COMST.2018.2871866

[22] Schaberreiter, T., Kupfersberger, V., Rantos, K., Spyros, A., Papanikolaou, A., Ilioudis, C., Quirchmayr, G.: A quantitative evaluation of trust in the quality of cyber threat intelligence sources. In: Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19). ACM, New York, NY, USA (2019). https://doi.org/10.1145/3339252.3342112

[23] Touch, S., Colin, J.-N.: A comparison of an adaptive self-guarded honeypot with conventional honeypots. Applied Sciences **12**(10), 5224 (2022) https://doi.org/10.3390/app12105224

[24] Pohl, F., Engelberth, M., Freiling, F., Holz, T.: T-Pot: The All-in-One Multi-Honeypot Platform. https://github.com/telekom-security/tpotce (2023)