Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

AI-Generated Honeypot Swarm with Adversarial Learning

Rishabh Jeppu
Computer Science and Engineering
Nitte Meenakshi Institute Of Technology (VTU Affiliation)
Bengaluru, India

Shreya
Computer Science and Engineering
Nitte Meenakshi Institute Of Technology (VTU Affiliation)
Bengaluru, India

Vaibhav S H
Computer Science and Engineering
Nitte Meenakshi Institute Of Technology (VTU Affiliation)
Bengaluru, India

Shresha Achari
Computer Science and Engineering
Nitte Meenakshi Institute Of Technology (VTU Affiliation)
Bengaluru, India

Dr. Krishnarao Venkatesh
Computer Science and Engineering
Nitte Meenakshi Institute Of Technology (VTU Affiliation)
Bengaluru, India

Abstract **- The rapid growth of AI-driven cyber threats has reduced the effectiveness of traditional security systems like firewalls and intrusion detection systems. Although traditional honeypots help observe attacker behaviour, they are static and easily detected. This paper proposes an AI-based honeypot system using adversarial learning to create a dynamic and adaptive cyber deception framework. The system uses a swarm of interconnected honeypots that simulate real services such as SSH, web applications, and databases. Each honeypot independently applies reinforcement learning (Q-learning) to improve its responses based on attacker interactions. It evaluates attacker behaviour—such as session duration and command usage—to generate rewards and enhance realism over time. An adversarial module introduces controlled fake attacks to identify weaknesses and strengthen responses. Large language models generate realistic system messages and logs to improve authenticity. Experimental results show that the proposed system keeps attackers engaged longer and adapts better than static honeypots, making it a scalable and proactive defense solution.**

Keywords - Honeypot, Adversarial Learning, Reinforcement Learning, Cybersecurity, Swarm Intelligence, LLMs, AI Defence.

I. INTRODUCTION

Cyber attacks are becoming more advanced and harder to defend against. Attackers now use automated tools, multi-step attacks, and even artificial intelligence to study systems and avoid security defences. Traditional security tools like firewalls and intrusion detection systems mainly react to known attack patterns. Because of this, they are not very effective against new or changing attacks. This creates a need for proactive security systems that can interact with attackers, study their behaviour, and adapt over time.

Honeypots are fake systems designed to attract attackers away from real systems and collect information about their actions. However, most traditional honeypots are static. They use fixed responses, allow limited interaction, and usually work as standalone systems. Experienced attackers can easily identify them by noticing repeated behaviour, unrealistic replies, or mismatches between services. Once attackers recognize a honeypot, it becomes useless and provides little useful information.

To solve these problems, this project proposes an AI-driven honeypot swarm that behaves like a real, distributed system instead of isolated services. Multiple honeypots – such as SSH server, web application (https), and databases – are connected and work together to appear as one realistic environment. This makes attackers stay engaged longer and allows the system to collect deeper and more meaningful attack data.

The core of the system uses reinforcement learning with Q-learning. Each honeypot acts as an independent learning agent that improves it behaviour by interacting with attackers. The system observes things like the type of request, command history, and attack method to understand the current situation. Based on this, the honeypot chooses responses and actions. Rewards are given based on attacker behaviour, such as how long they stay and how deeply they interact. Over time, the honeypots learn better responses that makes the deception more realistic and effective.

To make the system even stronger, an adversarial agent component is added. A controlled fake attacker regularly tests the honeypot swarm to find weak or unrealistic behaviour. This helps the system continuously improve. Large language models are also used to generate realistic logs, messages, and application data, making the environment more believable and harder to detect.

II. RELATED WORK

Background study

Honeypots are specialized security mechanisms designed to attract, detect, and analyze cyberattacks by mimicking real network services. They serve as valuable research tools to understand attacker strategies and collect forensic data without risking production systems. However, conventional

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

honeypots are typically static in configuration and limited in behavioral realism, making them easily detectable once adversaries identify their deceptive patterns or constrained functionalities.

Over time, attackers have adopted automated reconnaissance and AI-driven detection methods capable of recognizing anomalies in honeypot responses, timing patterns, and communication structures. These advances have rendered traditional honeypots less effective in modern threat environments, where adaptive and intelligent attack models continuously evolve. To remain relevant, honeypot systems must shift from static deception to dynamic and learning-based frameworks capable of real-time behavioral adaptation.

With the emergence of Artificial Intelligence (AI) and Machine Learning (ML), the potential for designing autonomous and intelligent honeypots has expanded significantly. Reinforcement Learning (RL) enables these systems to learn optimal deception policies from continuous interactions, while Adversarial Learning (AL) strengthens resistance to detection by generating deceptive yet realistic responses. Meanwhile, Large Language Models (LLMs) contribute by simulating authentic data and communication patterns, increasing the believability of the deception environment.

Motivation

The increasing use of AI-powered attack tools has exposed the limitations of conventional honeypots, which rely on static configurations and predefined deception rules. Modern attackers can easily detect and evade such systems using machine learning–based fingerprinting and behavioral analysis. As a result, there is a growing need for honeypots that can adapt in real time, evolve with attacker behavior, and maintain long-term deception effectiveness.

This project is motivated by the goal of creating a self-learning, intelligent honeypot swarm capable of continuous adaptation and realistic interaction. By integrating Reinforcement Learning (RL) and Adversarial Learning (AL), the system can dynamically refine its responses and resist detection. The use of Large Language Models (LLMs) further enhances realism by generating authentic content, enabling honeypots to appear indistinguishable from genuine systems. Together, these technologies aim to establish a proactive and evolving cyber defence mechanism against modern AI-driven attacks.

III. SYSTEM MODEL

The ShopHub Honeypot Swarm is a distributed honeypot framework that emulates a realistic e-commerce environment to study attacker behavior. It integrates multiple interconnected honeypots—SSH, HTTPS, and Database—which communicate via Apache Kafka as a centralized event bus. The system employs Reinforcement Learning (RL) and Adversarial Learning (AL) to adapt deception strategies and continuously improve its realism and resilience against evolving attack patterns.

A. Proposed Architecture

Figure 1 illustrates the architecture of the ShopHub Honeypot Swarm, highlighting its major functional components and data flow. The system begins with the Honeypot Swarm Layer, which includes SSH, HTTPS, and Database honeypots that simulate different interaction surfaces of a realistic e-commerce environment. These honeypots communicate through the Kafka Event Bus, which serves as a centralised stream for inter-honeypot coordination and real-time event synchronisation.

The collected data is then processed by the Data Pipeline, where raw logs are transformed into structured information. This output is subsequently utilised by two conceptual modules: the Reinforcement Learning Module, which uses engagement metrics to refine response strategies, and the Adversarial Learning Module, which periodically generates simulated attacks to test and enhance the system's adaptability and realism.

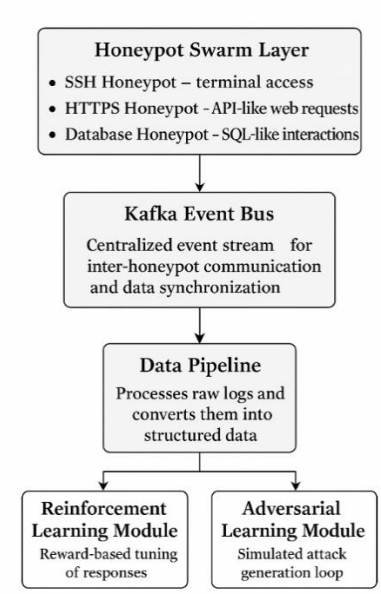

Fig 1. System architecture

B. System Workflow

Figure 2 below illustrates the data flow and operational process of the ShopHub Honeypot Swarm. The interaction begins when an attacker engages with one of the honeypots—SSH, HTTPS, or Database—each designed to emulate realistic system interfaces. The honeypots record incoming connections, commands, and payloads, which are then transmitted to the Data Pipeline for processing and formatting. The processed information is passed to the Cleaned Data Module, where redundant logs are filtered, and meaningful insights and reinforcement feedback are generated. These feedback signals are looped back to the honeypots to dynamically adjust response strategies and improve deception quality. Simultaneously, the Dashboard visualises activity trends, connection patterns, and engagement metrics for real-time monitoring. Additionally, an Adversarial Agent periodically launches simulated attacks to evaluate and enhance the swarm's adaptability and resilience.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

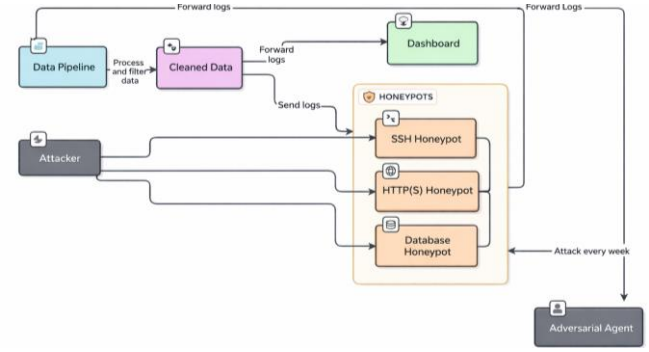

Fig 2. System workflow

C. Reinforcement and Adversarial Learning Concept

The reinforcement mechanism operates conceptually as a feedback loop:

Step	Description
State (s)	Current attacker behaviour — type of request, command sequence, etc.
Action (a)	Honeypot's generated response (e.g., valid output, access denied, fake data).
Reward (r)	Computed from interaction duration and command diversity.
Policy Update (π)	Adjusts response selection heuristics for the next interaction.

Table 1. Reinforcement mechanism

Although reinforcement learning is not yet automated, the system's design accommodates future integration of RL agents to learn optimal response policies from historical interaction data.

Adversarial Learning Logic

To enhance robustness against deception-aware attackers, the proposed honeypot swarm incorporates an adversarial learning component that exposes the system to detection-oriented attack behaviours. While reinforcement learning optimises responses based on observed interactions, adversarial learning introduces worst-case probing patterns to prevent policy overfitting and improve resilience.

D. Engineering Standards and Design Constraints

The system employs lightweight large language models (LLMs) to ensure low-latency and context-aware response generation, enhancing the realism of attacker interactions without compromising performance. To maintain security and safety, all honeypots are isolated from live internet traffic, effectively preventing any unintended exposure or misuse of system resources. Furthermore, the architecture is designed with a strong emphasis on modularity, enabling seamless integration of additional honeypots, data processing modules, or learning components in future expansions.

Constraint Type	Description
Technical	Limited compute resources for concurrent honeypot sessions.
Economic	Fully open-source stack (Kafka, Python, AsyncIO).
Safety	Honeypots deployed in isolated local network to prevent real breaches.
Legal	All interactions use synthetic traffic; compliant with cybersecurity ethics.
Sustainability	Modular, extensible design for long-term scalability.
Schedule	Reinforcement and adversarial loops are planned for future integration.

Table 2.Types of Constraints

IV. METHODOLGY

Overview

The proposed methodology aims to build an adaptive and intelligent honeypot system that learns from attacker interactions and gradually improves the realism of its responses. Unlike traditional honeypots that rely on fixed, predefined behaviour, this approach uses a swarm-based design where multiple connected honeypots work together to imitate a real application environment. By combining data collection, preprocessing, and reinforcement learning, the system continuously adapts its behaviour based on how attackers interact with it.

To ensure safe and ethical operation, the system runs in a closed and controlled environment. This setup provides strong isolation from real production systems and complies with standard cybersecurity research practices. Attacker actions are treated as feedback signals, allowing the system to learn, refine its behaviour, and improve deception quality over time.

Honeypot Swarm Architecture

The system is built as a honeypot swarm made up of multiple heterogeneous honeypots, each designed to imitate a commonly targeted service. The swarm includes:

1. **SSH Honeypot** – Simulates remote shell access and command-line interactions
2. **Web (HTTPS) Honeypot** – Emulates webapplication components such as login pages and APIs
3. **Database Honeypot** – Mimics backend database services and query behaviour.

Rather than operating in isolation, all honeypots are logically connected and share contextual information such as session identifiers and interaction history. This coordination allows the swarm to present a consistent and unified system state, making it appear as a real, functioning application. Each honeypot independently processes attacker requests while contributing to a shared view of the overall interaction.

Data Collection and Logging

All interactions between attackers and the honeypot swarm are recorded in detail. The system captures:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

1. Source information such as IP address, timestamp, and protocol
2. Commands or payloads sent by the attacker
3. Responses generated by the honeypots and internal system states
4. Session length and depth of interaction

Logs from all honeypots are sent to a centralized logging pipeline, where they are timestamped, normalized, and stored in a structured format. This unified logging approach supports both real-time monitoring and later analysis for learning and evaluation purposes.

Data Preprocessing and Feature Extraction

The raw interaction logs are processed to extract useful information for learning. During preprocessing, redundant entries are removed, malformed data is filtered out, and interactions are grouped at the session level.

From this cleaned data, several key features are extracted, including:

1. Number of commands executed during a session
2. Types and frequency of requests
3. Time gaps between consecutive actions
4. Session duration and how the session ends

These features are used to represent the system state and to compute reward values for the reinforcement learning process.

Reinforcement Learning Framework

To enable adaptive behaviour, the honeypot swarm uses reinforcement learning based on Q-learning, a model-free algorithm well suited for environments where attacker behaviour is unpredictable.
In this framework:

1. Each honeypot functions as an independent learning agent
2. The environment consists of attacker actions and honeypot responses
3. Learning happens through repeated interaction and feedback

State Definition: The state captures the current interaction context of a honeypot. It includes information such as:

- The type of service being accessed (SSH, web, or database)
- The category of the current command or request
- Indicators of session progress, such as the authentication stage or query depth

Action Space: The action space defines the possible responses a honeypot can generate. These actions include:

- Accepting or rejecting login attempts
- Returning simulated command outputs or database query results
- Introducing realistic delays or controlled error messages

Reward Function: The reward function is designed to encourage realistic and engaging attacker interactions. Rewards are assigned based on factors such as:

- Longer session duration
- Continued and consistent attacker activity
- Increased depth and diversity of commands

Higher rewards indicate successful deception and sustained engagement, while short or prematurely terminated sessions receive lower rewards.

Q-learning Update: Each honeypot updates its action-value function using the standard Q-learning update rule:

$$Q(s,a) = Q(s,a) + \alpha\,[\,r + \gamma\, max\, Q(s',a') - Q(s,a)\,]$$

where α represents the learning rate, and γ is the discount factor. Over time, this update process allows each honeypot to learn response strategies that maximise long-term engagement and improve the overall realism of the deception environment.

Adversarial Learning Framework

In the proposed system, adversarial learning is implemented through a LangGraph-orchestrated attacker agent powered by Google's Gemini large language model. Rather than injecting fixed detection-oriented sequences, the agent autonomously plans, executes, and evaluates multi-protocol attack campaigns against the honeypot swarm. The agent operates on a structured state graph in which each node corresponds to a discrete phase of the attack lifecycle, executed sequentially across SSH, HTTP, and database protocols.

Each attack cycle begins with a planning node in which Gemini receives a structured prompt describing the target host and active honeypot ports, and produces a tactical narrative specifying its intended attack strategy, expected findings, and risk assessment. This plan governs subsequent nodes. For each protocol, the agent first executes a reconnaissance node that runs a sampled set of commands or queries drawn from a predefined pool, then passes the recon output to a Gemini-powered command-generation node. Gemini analyses the reconnaissance results and emits a JSON array of targeted exploit commands, HTTP request paths, or SQL queries, respectively, which are then executed in the exploit node. A random variation seed is injected into each generation prompt to ensure behavioural diversity across episodes.

Upon completing all six attack nodes, an evaluation node aggregates the results and invokes Gemini a final time to produce a campaign summary identifying which protocols yielded the most valuable honeytoken intelligence, assessing evasion effectiveness, and recommending whether the honeypot's file structure should evolve. A scalar reward is also computed from the protocol outcomes using the function defined in the reward section below. This reward signal closes the feedback loop: it informs the evolution decision engine and is logged alongside the full campaign JSON for offline reinforcement learning policy updates.

Adversarial campaigns are scheduled at a configurable interval (default 300 seconds) and repeat for a configurable

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

number of episodes per cycle. Between episodes the agent waits a brief delay, then reinitialises its state and launches a fresh campaign. Because Gemini generates new exploit commands each episode based on live recon output, the attack surface explored varies naturally across runs without requiring manual scenario authoring. The suspicion accumulator tracks cumulative risk across protocols within a single episode; if it exceeds 0.75, remaining protocols are skipped, mirroring realistic attacker abort behaviour under elevated detection risk.

V. IMPLEMENTATION

This chapter details the actual implementation of the ShopHub Honeypot Swarm based on the design presented earlier. The system was implemented in Python and deployed to Google Kubernetes Engine (GKE) Autopilot on Google Cloud Platform. Each honeypot runs as an independent asynchronous server containerised in Docker and communicates through Kafka topics over a private Kubernetes cluster network.

System Overview

The proposed honeypot swarm was implemented as a modular and distributed system designed to emulate a realistic multi-service application environment. The implementation integrates coordinated honeypots, centralised logging, data preprocessing, and a reinforcement learning module based on Q-learning to enable adaptive response behaviour. All components were deployed in an isolated virtual environment to ensure safety and reproducibility.

Honeypot Swarm Implementation

The honeypot swarm consists of multiple heterogeneous honeypots, each emulating a commonly targeted service. These include an SSH honeypot, a web (HTTPS) honeypot, and a database honeypot. Each honeypot runs as an independent service but shares contextual information such as session identifiers and interaction history to maintain consistency across services.

The SSH honeypot simulates authentication attempts and command-line interactions, responding with predefined but configurable outputs. The web honeypot emulates application endpoints such as login and data retrieval interfaces, while the database honeypot processes structured query inputs and returns synthetic records. This coordinated deployment presents attackers with the illusion of a single, integrated application rather than isolated decoy systems.

Logging and Data Pipeline

All honeypots generate detailed interaction logs capturing connection metadata, attacker inputs, system responses, and session lifecycle events. These logs are forwarded asynchronously to a centralized data pipeline. The pipeline performs normalization and assigns unique session identifiers to correlate activity across multiple honeypots.

Structured logs are stored in persistent storage and made available for analysis and reinforcement learning. This centralized logging mechanism ensures reliable data collection while minimizing performance overhead on individual honeypot instances.

Data Processing and Feature Extraction

Raw interaction logs are processed to remove noise and extract features relevant to learning. Duplicate entries, malformed records, and incomplete sessions are filtered during preprocessing. The remaining data is aggregated at the session level to compute features such as session duration, number of commands executed, request diversity, and interaction frequency.

These extracted features are used to define the state representation and reward signals required for reinforcement learning.

Reinforcement Learning Implementation

Adaptive behaviour in the honeypot swarm is achieved using Q-learning, a model-free reinforcement learning algorithm. Each honeypot maintains an independent Q-table, enabling service-specific learning while preserving modularity.

1. **State and Action Representation:** The state space is discretised to represent the current interaction context, including the service type, request or command category, and session stage. The action space consists of possible honeypot responses, such as authentication outcomes, simulated command outputs, query results, and controlled delays.
2. **Policy and Action Selection:** An ε-greedy policy is employed to balance exploration and exploitation. During early interactions, random actions are selected to explore the response space. As learning progresses, actions with higher Q-values are increasingly favoured.
3. **Reward Design:** Rewards are computed based on attacker engagement indicators, including session length, number of interactions, and continuation of activity following honeypot responses. Higher rewards correspond to increased engagement and successful deception.
4. **Q-learning Update:** After each interaction, Q-values are updated using the standard Q-learning rule:

$$Q(s,a) <- Q(s,a) + \alpha\,[\,r + \gamma \max Q(s',a') - Q(s,a)]$$

where α denotes the learning rate, and γ denotes the discount factor. Over successive sessions, the policy converges toward response strategies that maximise long-term engagement.

Execution Flow

The implemented system operates in a closed feedback loop. Attacker interactions are processed by the honeypots, logged by the data pipeline, and analysed to compute rewards. Updated Q-values are applied to subsequent interactions, enabling continuous adaptation without manual intervention.

Adversarial learning

The adversarial learning component is implemented as a LangGraph state machine that orchestrates a Gemini-powered attack agent against the live honeypot swarm. The agent is defined in sequential_attack_runner.py and executes fully automated, multi-protocol attack campaigns consisting of eleven sequential graph nodes: plan_attack, attack_ssh_recon, gen_ssh_exploit_cmds,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

attack_ssh_exploit, attack_http_recon, gen_http_exploit_cmds, attack_http_exploit, attack_db_recon, gen_db_exploit_cmds, attack_db_exploit, and evaluate_campaign. Each node updates a shared AgentState TypedDict that accumulates attack results, suspicion scores, LLM-generated commands, and reward values across the full campaign lifecycle.

The agent's graph is constructed using LangGraph's StateGraph API. Conditional edges route execution based on the cumulative suspicion score: if overall_suspicion exceeds 0.75 after any recon node, remaining protocol branches are skipped, emulating realistic abort behaviour under elevated detection risk. Each recon node invokes the corresponding live attacker class (SSHAttacker, HTTPAttacker, or DatabaseAttacker), which uses asyncssh or raw asyncio socket connections to interact with the honeypot. If the live connection fails, the node transparently falls back to a deterministic simulator that returns pre-defined realistic responses, ensuring the campaign graph always completes and produces loggable results.

The reward function implemented in _calculate_reward aggregates per-protocol scores: a successful recon contributes +2.0 points, a successful exploit contributes +8.0 points, and each kilobyte of data extracted contributes +1.0 point. A suspicion penalty of −5.0 per 0.1 unit of suspicion above 0.5 discourages overly aggressive behaviour. Formally, the reward R for a campaign episode is computed as:

$$R = \sum_{p} (2 \cdot 0_{recon} + 8 \cdot 0_{exploit} + \frac{bytes}{1024}) - 50 \cdot max(0, \sigma - 0.5)$$

where **0** denotes a binary success indicator per phase and σ is the cumulative suspicion score. This reward is logged with each campaign JSON and can be used as a training signal for offline policy optimisation. An evolution decision node examines the reward and evaluation summary: if the score falls below a configurable threshold or Gemini identifies specific protocol weaknesses, the FileStructureEvolver class is invoked to mutate the honeypot's file_structure.py in place, planting new honeytoken files, rotating credentials, poisoning bash history, or adding new decoy database rows. This closes the adaptive loop: the adversarial agent probes the honeypot, the reward signal quantifies deception quality, and the evolver updates the honeypot's environment to be more convincing in the next campaign cycle.

VI. IMPLEMENTATION ENVIRONMENT

Component	Details
Language	Python 3.12
Libraries	asyncio, asyncssh, confluent-kafka, langchain, langgraph, langchain-google-genai, aiohttp, structlog
Message Broker	Apache Kafka
Operating System	Ubuntu 22.04 (container), GKE Autopilot (asia-south1)
Development Tools	VS Code, Docker, GitHub Actions CI/CD, Google Cloud CLI, kubectl
Testing Tools	mysql-client, curl, ssh, sequential_attack_runner.py (Gemini agent), kubectl logs
Cloud Platform	Google Cloud Platform (GCP), GKE Autopilot
Container Registry	Google Artifact Registry (asia-south1)
CI/CD Pipeline	GitHub Actions with Workload Identity Federation
Component	**Details**
Language	Python 3.12
Libraries	asyncio, aiokafka, re, json, typing
Message Broker	Apache Kafka
Operating System	Windows 11 / Ubuntu 22.04
Development Tools	VS Code, Kafka CLI
Testing Tools	netcat, curl, local shell connections

Table 3. Environment

VII. RESULTS

A. Experimental Results

The proposed honeypot swarm was evaluated in a controlled and isolated environment using synthetic attacker interactions targeting SSH, HTTPS, and database services. Attack scenarios included credential brute force attempts, command execution, malformed requests, and multi-service probing to emulate realistic attacker behaviour. All experiments were conducted locally to ensure safety and reproducibility.

Two configurations were compared:

1. **Static Honeypot Configuration** – predefined responses without learning
2. **Adaptive Honeypot Configuration** – reinforcement learning with adversarial learning enabled

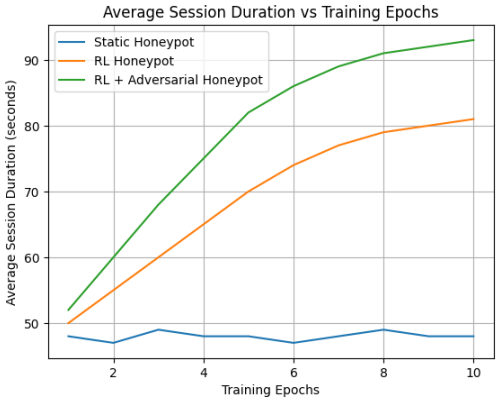

Fig.3 illustrates the evolution of average attacker session duration over successive training epochs. While the static honeypot shows no improvement, reinforcement learning enables gradual adaptation. The adversarially trained honeypot converges faster and achieves higher sustained

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

engagement, indicating improved robustness against detection-oriented probing.

Each configuration was evaluated over multiple interaction epochs. Performance was measured using attacker engagement–oriented metrics, which are commonly used to assess honeypot realism and deception effectiveness.

B. Reinforcement Learning Performance

Attacker Engagement Analysis: The primary objective of the honeypot swarm is to sustain attacker engagement and encourage deeper interaction. Table 4 summarises the comparative results between static and adaptive honeypots.

Metric	Static Honeypot	Adaptive Honeypot
Average Session Duration (s)	48.3	**114.6**
Average Interaction Depth	6.1	**18.4**
Session Continuation Rate (%)	41%	**84%**

Table 4. Engagement Performance Comparison

The adaptive honeypot swarm consistently achieved longer session durations and deeper interaction sequences. These results indicate that reinforcement learning enabled the honeypots to generate more realistic and convincing responses, thereby sustaining attacker interest.

Reinforcement Learning Convergence: The learning behaviour of the honeypot agents was evaluated by tracking the average reward over successive training epochs. A steady increase in reward values was observed, followed by convergence after several iterations. This trend suggests that the Q-learning agents successfully adapted their response strategies in response to attacker feedback.

Additionally, policy stability was observed after convergence, indicating that the learned strategies generalised across similar attack patterns without oscillatory behaviour.

C.Adversarial Evaluation

To evaluate robustness against deception-aware attackers, adversarial testing was conducted using detection-oriented attack sequences, including rapid command bursts, randomised request ordering, malformed inputs, and cross-service probing.

Performance under adversarial conditions was quantified using the **Robust Engagement Ratio (RER)**:

$$\text{RER} = \frac{E_{adv}}{E_{normal}}$$

where E_{adv}represents average engagement under adversarial probing and E_{normal}represents engagement under standard attacker interactions.

The adaptive honeypot swarm achieved an RER of **0.81**, compared to **0.52** for the static honeypot configuration. While adversarial probing reduced engagement in both cases, the adversarially trained honeypot maintained significantly higher interaction depth and session continuity.

Furthermore, the system demonstrated **policy recovery behaviour**, where engagement metrics temporarily declined during aggressive probing but recovered within subsequent epochs. This indicates that adversarial learning improved resilience rather than optimizing solely for benign attacker behaviour.

D.Experimental Rigor and Limitations

Experimental rigor was ensured by executing identical attack scripts across all configurations and averaging metrics over multiple sessions to minimize variance. Evaluation focused on engagement-based metrics, consistent with the system's deception-oriented objectives.

The experiments were conducted in a controlled environment using synthetic attackers, which may not fully represent real-world adversarial diversity. Additionally, discretized state and action spaces limit modeling of complex multi-stage attacks. Despite these constraints, the results demonstrate that integrating reinforcement learning with adversarial learning significantly improves honeypot realism and robustness compared to static deception approaches.

VIII.CONCLUSION

This work presents a distributed honeypot swarm designed to emulate a realistic e-commerce environment through coordinated SSH, HTTPS, and database honeypots communicating via Kafka. By leveraging reinforcement learning, the system adapts dynamically to attacker behaviour, using engagement duration as a feedback signal to enhance realism and interaction quality. The integration of lightweight LLMs further enhances believability through context-aware responses and the generation of synthetic content. The proposed architecture demonstrates how modular, data-driven honeypots can evolve beyond static deception to form adaptive, intelligent defence systems. Future work will focus on refining the reinforcement feedback loop, integrating automated model retraining, and deploying the swarm in larger-scale, real-world network environments to validate its robustness and scalability.

REFERENCES

[1] J. A. Christli, C. Lim and Y. Andrew, "AI-Enhanced Honeypots: Leveraging LLM for Adaptive Cybersecurity Responses," 2024 16th International Conference on Information Technology and Electrical Engineering (ICITEE), Bali, Indonesia, 2024, pp. 451-456, doi: 10.1109/ICITEE62483.2024.10808265.

[2] C. Sun et al., "Application of Artificial Intelligence Technology in Honeypot Technology," 2021 International Conference on Advanced Computing and Endogenous Security, Nanjing, China, 2022, pp. 01-09, doi: 10.1109/IEEECONF52377.2022.10013349.

[3] P. Patel, A. Dalvi and I. Siddavatam, "Exploiting Honeypot for Cryptojacking: The other side of the story of honeypot deployment," 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA, Pune, India, 2022, pp. 1-5, doi: 10.1109/ICCUBEA54992.2022.10010904.

[4] Hetzler, Connor & Chen, Zachary & Khan, Tahir. (2023). Analysis of SSH Honeypot Effectiveness. 10.1007/978-3-031-28073-3_51.

[5] V. A. Memos and K. E. Psannis, "AI-Powered Honeypots for Enhanced IoT Botnet Detection," 2020 3rd World Symposium on Communication Engineering (WSCE), Thessaloniki, Greece, 2020, pp. 64-68, doi: 10.1109/WSCE51339.2020.9275581.

[6] N. Ilg, D. Germek, P. Duplys and M. Menth, "Beekeeper: Accelerating Honeypot Analysis With LLM-Driven Feedback," in IEEE Access,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

vol. 13, pp. 168034-168054, 2025, doi: 10.1109/ACCESS.2025.3613118.

[7] H. Kim, F. Z. Zuluaga, C. Evans, S. R. Dalal, V. Misra and D. Rubenstein, "Real-Time Risk Scoring of Ongoing Cyber Attacks," 2025 IEEE Conference on Communications and Network Security (CNS), Avignon, France, 2025, pp. 1-6, doi: 10.1109/CNS66487.2025.11195057.

[8] N. Ilg, P. Duplys, D. Sisejkovic, and M. Menth, "A survey of contemporary open-source honeypots, frameworks, and tools," *J. Netw. Comput. Appl.*, vol. 220, Nov. 2023, Art. no. 103737.

[9] U. Raut, A. Nagarkar, C. Talnikar, M. Mokashi and R. Sharma, "Engaging Attackers with a Highly Interactive Honeypot System Using ChatGPT," 2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India, 2023, pp. 1-5, doi: 10.1109/ICCUBEA58933.2023.10392228.

[10] W. Fan, Z. Yang, Y. Liu, L. Qin, and J. Liu, "HoneyLLM: A Large Language Model-powered medium-interaction honeypot," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2024, pp. 253–272.s