This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

Reinforcement Learning Meets LLM Honeypots: A MITRE Engage–Aligned Approach

Ren-Hung Hwang, *Senior Member, IEEE,* Jiao-Chuan Huang, *Student Member, IEEE,* Yuan-Cheng Lai, and Ying-Dar Lin, *Fellow, IEEE*

***Abstract*—The growing sophistication of cyberattacks, accelerated by large language models (LLMs), highlights the limitations of traditional honeypots, which often lack realism, require heavy maintenance, and rely on static deception strategies. Recent LLM-based honeypots generate fluent, context-aware responses but cannot adapt to evolving attacker behavior, limiting long-term effectiveness. This work presents an adaptive honeypot that integrates reinforcement learning (RL) with LLM-generated deception, aligning state, reward, and action spaces with the MITRE ATT&CK and MITRE Engage frameworks. A fine-tuned LLM infers attacker tactics, techniques, and procedures (TTPs) from live command sequences, providing semantically rich states for the RL agent, which then selects context-sensitive actions from Engage's Affect strategies to guide adversaries toward deeper and higher-value engagement. Evaluated on Linux and Windows testbeds, the system achieved a 23% increase in cumulative engagement reward on Windows over a non-RL baseline ($p < 0.001$). Ablation over five random seeds shows that replacing the learned policy with random action selection over the same action space collapses attack depth from 9.52 to 4.25 on Linux ($p < 0.001$), confirming that the learned policy, not the action space alone, drives engagement. Intent analysis accuracy improved by 55 percentage points relative to a rule-based baseline (Wazuh), and LLM-generated responses fell within 10 percentage points of a real system, a substantially smaller gap than Cowrie, an ordering confirmed by an independent cross-family judge. These results demonstrate that RL-driven adaptation, combined with LLM realism and standardized engagement frameworks, enables honeypots that sustain realistic, intelligence-rich interactions and enhance threat analysis without compromising system safety.**

***Index Terms*—Cyber deception, honeypot, reinforcement learning, large language models, MITRE ATT&CK, MITRE Engage, SSH**

I. Introduction

HONEYPOTS lure adversaries into controlled environments so defenders can observe real attacks without jeopardizing production systems. Emulation-based traps such as Cowrie [1] and Kippo [2] are lightweight and low risk but are readily fingerprinted; high-interaction deployments (e.g., GOAD [3]) provide higher fidelity at the cost of maintenance effort and operational exposure. Large Language Models (LLMs) have recently improved the realism of interactive deception by generating fluent, context-aware responses, as shown by prototypes including Shellm [4], MySQL-Pot [5], Galah [6], and Vellmes [7]. However, these systems are largely *static*: once deployed, they do not adapt to evolving attacker behavior, reducing effectiveness over longer campaigns. Reinforcement Learning (RL) promises adaptive engagement but prior RL honeypots typically derive state directly from raw commands and restrict actions to coarse schemas (e.g., Allow–Block–Substitute–Insult). Such designs under-represent attacker *intent* and limit the defender's ability to steer interactions toward higher-value tactics. We address these gaps with an RL-enhanced LLM honeypot whose state, reward, and action spaces are systematically aligned with MITRE ATT&CK and MITRE Engage. A fine-tuned LLM maps incoming commands to Tactics, Techniques, and Procedures (TTPs), providing semantically rich states for the RL agent, which then selects context-sensitive actions from Engage's Affect strategies to guide adversaries toward deeper and higher-value engagement. The reward function encourages progression along the ATT&CK chain, enabling nuanced, persuasive responses rather than binary allow/deny decisions. The result is an adaptive architecture that entices adversaries to reveal more advanced and riskier behaviors while remaining operationally safe for network and service management contexts. Experiments on Linux and Windows testbeds show a 23% increase in cumulative reward on Windows over a non-RL baseline ($p < 0.001$), together with a 55-percentage-point gain in intent-analysis accuracy over a rule-based baseline (Wazuh). An ablation over five random seeds further shows that replacing the learned policy with random action selection over the same Engage action space collapses attack depth (from 9.52 to 4.25 on Linux, $p < 0.001$), isolating the contribution of the learned policy from the action space itself. On a system-recognition metric, LLM-generated responses fell within 10 percentage points of a real system, a substantially smaller gap than Cowrie and other baselines, an ordering preserved under an independent cross-family judge. **Contributions.** This work advances network/service management in the following ways:

1) We formulate adaptive LLM honeypot response as a reinforcement-learning problem grounded in operator objectives (engagement depth and attacker reward), enabling measurable control over adversary interaction.
2) We design ATT&CK/Engage-aligned state, reward, and action spaces that map attacker TTPs to policy-relevant decisions and persuasive responses rather than coarse

Ren-Hung Hwang is with the Department of Computer Science and Information Engineering, Asia University, Taichung, Taiwan, and the College of Artificial Intelligence, National Yang Ming Chiao Tung University, Tainan, Taiwan.

Jiao-Chuan Huang and Ying-Dar Lin are with the Department of Computer Science and Information Engineering, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

Yuan-Cheng Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan.

Corresponding author: Ying-Dar Lin (email: ydlin@nycu.edu.tw).

0000–0000/00$00.00 © 2025 IEEE

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

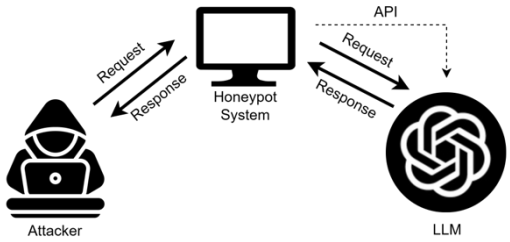

Fig. 1. LLM-based honeypot architecture.

allow/deny choices.

3) We provide a dynamic evaluation methodology, along with reproducibility artifacts (sanitized interaction transcripts, attack generation code, configuration files, and evaluation scripts), all of which are publicly available at [8], to facilitate reproducibility and cross-platform benchmarking.
4) We demonstrate cross-platform (Linux/Windows) gains in realism, deception, and adversary steering over LLM-only and emulation baselines (e.g., +23% cumulative reward on Windows over a non-RL baseline, +55 percentage points in intent-analysis accuracy over a rule-based baseline), and, through a five-seed ablation with significance testing, isolate the contribution of the learned RL policy from the action-space design, indicating practical impact for network/service operations.

The remainder of this paper is organized as follows: section II reviews LLM-based honeypots and RL for deception. section III formalizes the problem and notation. section IV presents the proposed approach, including RL design and attack generation. section V details the implementation and experimental setup. section VI provides comparative results. section VII concludes and outlines future research directions.

II. Background and Related Work

A. LLM-based Honeypots

Traditional honeypots are commonly grouped by interaction level. *Emulation-based* (low/medium-interaction) systems, e.g., Cowrie in its default SSH/Telnet mode and Dionaea [9], implement protocol flows and a virtual filesystem without exposing a full OS. They are lightweight and low risk, but easier to fingerprint because their behavior is bounded. *High-interaction* honeypots, e.g., GOAD or Cowrie's proxy/redirect mode, front real operating systems or applications, providing greater realism at the cost of upkeep and higher operational exposure.

Recent advances in Large Language Models (LLMs) have enabled a new generation of honeypots, including Shellm, Vellmes, MySQL-Pot, and Galah. Compared to fixed emulation scripts, LLM-driven interaction can condition on prior context and attacker intent to produce plausible behavior without exposing real infrastructure, enhancing deception and sustained engagement (Fig. 1). Building on these capabilities, we scope this work to SSH, a ubiquitous, text-based management protocol with a clear command–response pattern and a high prevalence in attack telemetry, so we can run controlled, reproducible experiments and compare against established baselines such as Cowrie.

B. MITRE ATT&CK

MITRE ATT&CK is a widely used knowledge base of adversary behavior that organizes activity into *tactics* (objectives) and *techniques* (methods). Tactics generally progress in severity, with later stages reflecting deeper system compromise. In our design, attacker commands are mapped to ATT&CK techniques at each interaction step, forming the RL state features, while rewards are computed from tactic severity and progression along the ATT&CK chain.

Accurate TTP identification is pivotal. Traditional EDR-based pipelines observe behaviors by executing real attacks, which raises annotation cost and operational risk. More recent LLM/NLP approaches (e.g., Raconteur [10], LogPrécis [11]) infer techniques from logs or text, but often process inputs in isolation, making multi-command techniques harder to detect. Our system instead fine-tunes an LLM on command sequences from Atomic Red Team (ART) [12], enabling recognition of multi-step attack chains while reducing cost and risk. Implementation details appear in Section V.

C. MITRE Engage

MITRE Engage structures defender–adversary interaction across three phases: *Prepare*, *Operate*, and *Understand*. Prepare emphasizes anticipating adversary methods using threat intelligence; Operate focuses on shaping behavior through interactive deception; and Understand supports post-engagement analysis and refinement.

In our system, the RL action space is instantiated from the *Affect* techniques defined in Engage, such as isolation, luring, and software manipulation. This provides more nuanced and persuasive responses than binary allow/deny choices, thereby steering adversaries toward deeper engagement and improving the collection of high-value intelligence.

D. Reinforcement Learning in Honeypots

Reinforcement Learning (RL) trains an agent to make sequential decisions by interacting with an environment and receiving feedback. An agent observes the current interaction, chooses a response, and receives a reward signal; the loop repeats until termination, matching the dialog-like nature of SSH attacker interactions.

Prior systems adapt responses based on observed behavior but typically derive the *state representation* directly from raw commands and limit the *action space* to a small set of coarse-grained actions (often 4–5). This under-captures semantic intent (e.g., credential access vs. lateral movement) and constrains the ability to steer attackers toward higher-value tactics. Building on Sections II-B–II-C, we redesign the RL formulation around semantic reasoning: interaction steps are mapped to ATT&CK techniques for state; rewards

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE I
SURVEY OF RECENT RESEARCH ON ADAPTIVE HONEYPOTS.

Category	Paper	Reinforcement Learning			Backend	Detect Method	Evaluation	
		State	Action	Reward			Conversation metrics	Attack Stage
IoT Honeypot	HoneyIoT	Request	Response	Exploit code	Yeelight bulb	-	-	V
	IoTCandyJar				Response data	-	-	V
SSH Honeypot	RASSH	Command	ABDSI	Command type	Kippo	Manual	V	-
	QRASSH				Cowrie		V	-
	Cannypot		ABSI	Command length			V	-
	Ours	Command's Technique	MITRE Engage	Command's Tactic	LLM	LLM	V	V

follow tactic severity and progression; and actions instantiate Engage's *Affect* strategies. This coupling of structured threat intelligence with adaptive learning improves deception quality and the value of collected telemetry.

E. Related Work

We focus on improving honeypot decoy capability with RL. Prior efforts fall into two domains: RL-based IoT honeypots and RL-based SSH honeypots.

IoT honeypots. HoneyIoT [13] and IoTCandyJar [14] use RL to lure attackers through simulated vulnerabilities or by mimicking device responses. Rewards often reflect exploit success, encouraging more revealing activity.

SSH honeypots. RASSH [15], QRASSH [16], and Cannypot [17] apply RL to engage adversaries. RASSH/QRASSH classify command types (e.g., downloads, suspicious commands) to promote progression, while Cannypot uses command length as a coarse proxy for intent. Most systems employ small, fixed action sets and shallow command features, limiting adaptability to diverse Linux inputs and nuanced attacker goals.

To compare existing systems, we consider: (i) honeypot type (IoT vs. SSH); (ii) RL settings (state, action, reward design); (iii) detection method (automation and interpretability of malicious behavior); and (iv) evaluation criteria (conversation metrics, stage progression). Table I summarizes recent efforts.

Our differentiators. We define RL state via ATT&CK techniques to capture intent semantically; align rewards with tactic severity to promote advanced-stage engagement; derive actions from Engage's *Affect* strategies for persuasive control; and fine-tune an LLM on ART sequences to detect multi-step TTPs. Together, these elements yield an adaptive honeypot that improves deception quality and real-time threat intelligence.

III. PROBLEM FORMULATION

This section formalizes the problem of adaptive, LLM-driven honeypot response. We first list the notation used throughout the paper, then define the setting and objectives, followed by the state, action, and reward design that underpins our approach.

A. Notation Table

This study proposes an intelligent response mechanism for an LLM-based honeypot system to enhance its ability to induce more advanced attacker behaviors. Traditional honeypots often rely on basic criteria, such as command type or length, which limit adaptability and reduce effectiveness against dynamic adversaries. By combining the reasoning capability of LLMs with the optimization capabilities of reinforcement learning, the system can better emulate real environments and influence attacker decisions.

To formalize the core components, we define a set of key notations categorized into three groups, Honeypot, Attacker, and Performance, which are summarized in Table II.

The Honeypot category defines variables related to system interactions, including the sequence of attacker commands, the tactics and techniques inferred under the MITRE ATT&CK framework, and the corresponding actions selected by the RL agent. These elements describe how the system interprets attacker input and responds at each step.

The Attacker category models adversarial behavior. At each step, the LLM-based attacker selects a technique based on past outcomes and generates a corresponding command using Atomic Red Team artifacts, which enables realistic and adaptive behavior.

The Performance category defines evaluation metrics that focus on the depth and progression of attacker engagement. These metrics include cumulative and maximum tactic indices observed in a session, which reflect how effectively the system encourages deeper exploration.

These notations form the foundation for the problem formulation, specifying system inputs, outputs, objectives, and constraints in a structured and formal manner.

B. Problem Statement

We model the intelligent honeypot as a sequential decision problem over a finite interaction of length N_C. At step $i \in \{1, \ldots, N_C\}$ the attacker issues a command C_i, the honeypot selects a response R_i from a predefined action set $\mathcal{A}$, and the interaction proceeds to the next step. Attacker behavior is interpreted through the MITRE ATT&CK framework, which yields a tactic τ_i (and techniques) for each step, as described in Section II-B. The action set $\mathcal{A}$ is instantiated from MITRE Engage *Affect* strategies, as summarized in Section II-C.

Objective: Let

$$\mathrm{AttackDepth}(C_{1:N_C}) \;=\; \max_{1 \le i \le N_C} \; \mathrm{index}(\tau_i),$$

where $\mathrm{index}(\tau_i)$ is the tactic index on the ATT&CK chain observed at step i. The primary objective is to maximize attack depth under realistic operational constraints. Formally,

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE II
NOTATIONS

Category	Notation	Description
Honeypot	N_C	Number of commands in one session
	$C_i, i \in \{1, 2, ..., N_C\}$	ith input command from attacker in one session
	$\tau_i, i \in \{1, 2, ..., N_C\}$	Detected tactic of command C_i
	$t'_i, i \in \{1, 2, ..., N_C\}$	Detected technique of command C_i
	N_A	Number of predefined honeypot actions
	$a_i \in N_A, i \in \{1, 2, ..., N_C\}$	Action selected by RL for command C_i
	$R_i, i \in \{1, 2, ..., N_C\}$	Honeypot response to command C_i
Attacker	N_R	Number of responses in one session
	$t_i, i \in \{1, 2, ..., N_R\}$	ith output technique from attacker (LLM)
	$C_i, i \in \{1, 2, ..., N_R\}$	ith output command from ART
Performance	$\sum_{i=1}^{N_C} \tau_i$	Sum of detected tactic indices in session
	$\max \tau_i$	Maximum tactic index τ_i in session

the defender chooses a policy π that maps the information available at step i to a distribution over actions in $\mathcal{A}$,

$$\pi^\star \in \arg\max_{\pi} \ \mathbb{E}_\pi[\,\mathrm{AttackDepth}(C_{1:N_C})\,],$$

where the expectation is with respect to the stochasticity of attacker behavior and the honeypot policy.

Constraints. Operational safety constraints are enforced throughout the interaction:

- total interaction time ≤ 10 minutes,
- file transfer operations are disallowed (or simulated without persistence),
- use of text editors is restricted.

These constraints reflect the deployment assumptions and are consistent with Section II. They ensure that the policy promotes deeper engagement while maintaining plausibility and system integrity.

Discussion.: The key challenge is to strike a balance between deception and plausibility. If responses are too generic or unrealistic, attackers may disengage quickly; if responses are too passive or too accurate, such responses may fail to elicit high-value behaviors. The objective is to design a reward-driven, adaptive response policy that reveals deeper attack strategies while maintaining system integrity and adhering to the operational constraints above.

The formulation captures the setting in which an attacker issues a sequence of commands $\{C_i\}_{i=1}^{N_c}$, the honeypot returns responses $\{R_i\}_{i=1}^{N_c}$ selected from $\mathcal{A}$, and depth is measured by the maximum observed tactic index $\mathrm{index}(\tau_i)$. This matches the narrative in Section II and the notation in Table II, while providing a concise optimization target for the subsequent design.

IV. LLM HONEYPOT WITH RL OPTIMIZATION

A. Approach Overview

We present an adaptive honeypot framework that integrates Reinforcement Learning (RL) with Large Language Models (LLMs), systematically grounded in the MITRE ATT&CK and Engage frameworks. The system dynamically learns to engage adversaries through realistic, strategy-driven interactions that evolve over time. It comprises five interconnected modules: (i) an LLM-based attacker simulator, (ii) the Atomic Red Team (ART) dataset for realistic command generation, (iii) a fine-tuned LLM for TTP extraction, (iv) the RL agent for response policy learning, and (v) an LLM-driven honeypot interface. We clarify the sense in which the system is "adaptive." The DDQN policy is trained offline and then deployed as a fixed policy; adaptivity here refers to *within-session, state-conditioned* response selection rather than online policy updates during deployment. Because the state is the inferred TTP of the current interaction step, the same command can elicit different responses depending on the attacker's position in the ATT&CK chain, and the response the attacker receives shapes their subsequent commands. This state-conditioned behavior distinguishes our approach from the static prompting of the LLM-only baseline, whose responses do not depend on a learned, intent-aware policy, and from fixed-template honeypots such as Cowrie. Extending the agent to update its policy online during deployment is left to future work.

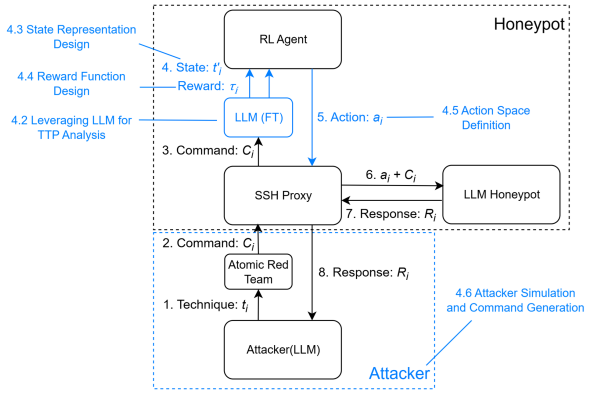

Fig. 2. System architecture overview. The attacker issues a technique t_i which is converted into a command C_i via ART. The honeypot infers the tactic τ_i, and the RL agent selects an action a_i to generate the system's response.

As shown in Fig. 2, the attacker selects a technique t_i

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

based on its strategy, which ART translates into a concrete command C_i. The honeypot uses a fine-tuned LLM to extract the associated tactic τ_i and technique t_i' from the command, providing semantic context for both state representation and reward calculation.

Given the inferred TTPs, the RL agent selects an action a_i from a set of Engage-based countermeasures. This action, along with the command C_i, is passed to the LLM honeypot, which generates a context-aware system response. The response is returned to the attacker, influencing its subsequent behavior. Through this closed-loop interaction, the RL agent progressively learns policies that elicit deeper, more informative adversary behavior.

B. LLM for TTP Inference

Accurate identification of tactics and techniques is critical for effective state modeling and reward shaping. Traditional EDR-based methods require executing commands on real endpoints, introducing operational risk and significant costs. While recent LLM-based approaches bypass execution by analyzing command text, most treat commands in isolation, limiting their ability to capture multi-step context or infer underlying attacker intent.

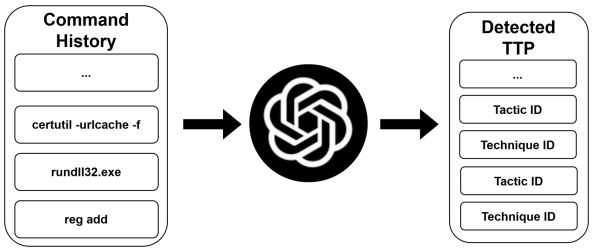

Fig. 3. TTP detection using a fine-tuned LLM on ART-derived command–technique pairs.

To address this, we fine-tune an LLM using supervised learning on 1,500 command–technique pairs from the ART dataset (Fig. 3). ART provides well-structured, up-to-date attack procedures, allowing high-fidelity mapping from commands to techniques (t_i'). More importantly, by learning a semantic mapping instead of brittle string rules, the detector can generalize to unseen command variants and moderate obfuscation, provided they express behaviors consistent with known ATT&CK techniques. Compared to reinforcement-based tuning (e.g., PPO), supervised fine-tuning offers better generalization on this structured, knowledge-rich task.

Tactic inference (τ_i) is inherently contextual: a given technique may support multiple tactics depending on the scenario. Rather than relying on rule-based logic, we reuse the same fine-tuned LLM to infer τ_i by incorporating command history as context. This enables semantic disambiguation across multi-command interactions and reduces dependence on high-cost behavioral execution.

By analyzing full command sequences rather than isolated inputs, our approach enhances the accuracy of attacker intent inference while avoiding reliance on costly, execution-based EDR systems. The resulting TTPs are then used to construct the RL agent's state representation and to compute rewards, as detailed in the following sections.

C. State Representation

This section details how attacker behavior is encoded into structured states for reinforcement learning. A key contribution of our approach is the design of a semantically rich yet well-defined state representation that reflects attacker intent rather than superficial input features.

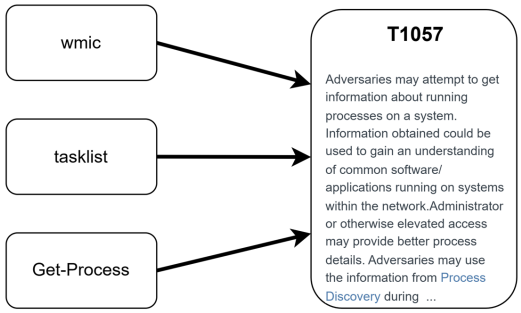

Fig. 4. State abstraction: diverse commands (e.g., `wmic`, `tasklist`) map to a common technique (e.g., Process Discovery), enabling semantic generalization.

At each time step i, the environment state s_i is defined as the detected technique identifier t_i' inferred from the current command C_i and its historical context. The state space is discrete and finite, consisting of 203 possible values corresponding to the set of MITRE ATT&CK techniques considered in this work. Each state, therefore, captures the semantic intent of the attacker at a given interaction step, rather than raw command syntax.

Each command issued by the attacker is first mapped to a technique t_i' using the fine-tuned LLM described in Section IV-B. To improve robustness and disambiguation, the LLM does not process commands in isolation; instead, it consumes the full command sequence $\{C_1, C_2, \ldots, C_i\}$ as context when inferring t_i'. This design allows the model to resolve ambiguous commands whose intent may only become clear when viewed in sequence.

This abstraction enables the RL agent to reason about the semantic purpose of actions. For example, diverse commands such as `wmic`, `tasklist`, and `Get-Process`, though syntactically different, are all associated with the same ATT&CK technique (e.g., Process Discovery) and are thus treated equivalently. As illustrated in Fig. 4, this semantic normalization allows the agent to generalize across functionally similar but lexically varied inputs, enhancing its ability to recognize attacker objectives.

For input to the Q-network, the inferred technique ID t_i' is encoded as a one-hot vector over the 203-dimensional technique space. This explicit and compact representation ensures a stable and interpretable state encoding while avoiding the noise and dimensionality associated with raw text or embedding-based features.

Beyond individual commands, the evolving state sequence $\{t_1', t_2', \ldots, t_i'\}$ implicitly captures temporal dependencies in

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

1	2	3	4	5	6	7	8	9	10	11	12
Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact

Fig. 5. Tactic ID

attacker behavior. By observing transitions between techniques over time, the agent can detect higher-level patterns and adapt to multi-stage attack strategies, such as privilege escalation following credential access or lateral movement after system discovery. Although the Q-network input at each step is the current one-hot state, the sequence-aware technique inference enables the agent to align its deceptive responses with the adversary's progression.

Grounding the state in the standardized ATT&CK framework improves robustness against advanced persistent threats (APTs), whose behaviors often span multiple stages and obfuscate intent through surface variability. By abstracting raw inputs into a structured, intent-driven state space, the honeypot can simulate realistic system behavior while subtly guiding the attacker toward deeper or riskier tactics.

This state representation empowers the RL agent to engage adversaries at a strategic level—reasoning over intent, recognizing multi-step patterns, and dynamically shaping attacker behavior—thereby enhancing the realism, adaptability, and operational value of the honeypot system.

D. Reward Function

The reward function is designed to enable the RL agent to actively guide attacker behavior toward more informative and consequential attack stages. Prior approaches often define rewards based on superficial properties such as frequently observed attack commands or command length; however, such designs suffer from several limitations. Modern operating systems, such as Linux, support thousands of possible commands, making command-level reward assignment impractical, and command length does not necessarily correlate with attack severity or threat escalation. Alternatively, delegating reward estimation to an LLM introduces additional risks, including output variance and hallucination. To address these challenges, we design our reward function based on the MITRE ATT&CK framework, whose tactics provide an industry-standard, severity-ordered abstraction that generalizes across diverse attack scenarios without requiring scenario-specific reward tuning.

For each attacker-issued command C_i, the system infers the corresponding tactic τ_i, representing the attack stage from initial access to impact. As illustrated in Fig. 5, each tactic is assigned a unique integer ID according to its relative position in the ATT&CK matrix. Early-stage post-compromise tactics (e.g., Execution) are associated with lower IDs, whereas later-stage, higher-impact tactics (e.g., Exfiltration and Impact) receive higher values. These tactic IDs are used directly as linear scalar rewards, encouraging the RL agent to elicit deeper and more consequential adversarial behavior. To maintain focus on meaningful post-compromise interactions, tactics belonging to pre-compromise phases, specifically Reconnaissance and Resource Development, are excluded from the reward calculation.

We adopt a linear mapping from tactic IDs to rewards rather than a non-linear scaling (e.g., exponential). This choice is motivated by three considerations. First, attacker behavior does not necessarily progress monotonically toward deeper tactics; in our experiments, 50.35% of sessions exhibited non-monotonic tactic progressions, where attackers revisit lower-stage tactics strategically to enable subsequent high-impact actions (e.g., lateral movement followed by execution and persistence). A linear reward naturally accommodates such non-linear attack progressions without imposing artificial path constraints. Second, non-linear scaling would disproportionately emphasize terminal tactics, creating large reward gaps between adjacent stages. This risks causing the agent to prematurely force attackers toward late-stage tactics such as exfiltration before sufficient reconnaissance or credential gathering has occurred, likely triggering early disengagement and reducing overall session quality. Third, a linear mapping requires no scenario-specific parameter tuning, whereas non-linear alternatives (e.g., exponential or piecewise weighting) introduce additional hyperparameters whose optimal values may vary across attack profiles and deployment contexts.

Note that the reward function is entirely defender-internal: it is used solely to train the RL agent's policy and is never exposed to the attacker. Consequently, the attacker cannot strategically exploit or game the reward mechanism.

Tactic inference is performed using the fine-tuned LLM described in subsection IV-B, which incorporates command history for context-aware prediction. Given the one-to-many mapping between commands and tactics, this approach ensures accurate and consistent assignment, even under ambiguous or multi-functional input.

The cumulative reward for an interaction session is computed as follows:

$$\text{Total Reward} = \sum_{i=1}^{N_C} \tau_i$$

$$\tau_i = \begin{cases} \text{ID}_{\text{tactic}}, & \text{if a tactic is detected for } C_i \\ -5, & \text{if the attacker exits the system} \\ 0, & \text{otherwise} \end{cases}$$

Here, N_C denotes the total number of interaction steps, and τ_i is the reward assigned at each step. A penalty of -5 is applied when the attacker terminates the session prematurely, incentivizing sustained engagement and discouraging shallow interactions.

This design yields dense, continuous, and semantically aligned feedback, enabling the RL agent to develop deception strategies that steer adversaries into later and more informative

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

stages of the attack chain. We note that this cumulative sum, $\sum_{i=1}^{N_C} \tau_i$, serves as a dense training surrogate for the sparse operational objective defined in Section III-B, namely maximizing the attack depth $\max_{1 \le i \le N_C} \text{index}(\tau_i)$. Because tactic IDs are non-negative and ordered by severity, driving up the per-step cumulative reward correlates with reaching deeper tactics, while providing per-step credit assignment that stabilizes DDQN training. The two are therefore complementary rather than conflicting targets: the sum is the optimization signal used during learning, whereas the maximum attack depth is the operational quantity we ultimately report.

E. Action Space

To maximize deception effectiveness, we design the action space based on the MITRE Engage framework, which provides structured guidance for adversary engagement. Each action a_i corresponds to a system response to an attacker-issued command C_i, selected by the RL agent according to the current state (i.e., the inferred tactic and technique). This design enables the honeypot to respond plausibly while subtly steering attacker behavior toward higher-value engagement.

a) Action Space Definition.: The complete action space is derived from the *Affect* phase of MITRE Engage and consists of nine distinct actions grouped into seven engagement categories. These actions represent high-level deception strategies rather than fixed response templates. Table III presents the full action set used in our system, and Appendix A provides a detailed mapping between individual Engage activities and their corresponding implemented actions for clarity and reproducibility.

Unlike prior systems such as QRASSH and Asguard, which rely on small, manually crafted response sets, our design incorporates an additional LLM to instantiate actions dynamically. Each selected action is realized through a prompt passed to the LLM-based response generator, allowing the honeypot to synthesize context-aware and semantically consistent system outputs in real time. This approach eliminates the scalability limitations of static response engineering while preserving alignment with Engage strategies.

Following the MITRE Engage *Affect* phase, the action space includes the following seven categories of countermeasures designed to simulate realistic system behavior and influence attacker decision-making:

- **Baseline** – Restore or maintain a default observable system state
- **Isolation** – Restrict attacker activity to controlled or sandboxed environments
- **Network Manipulation** – Simulate modified traffic conditions, bandwidth limits, or connectivity
- **Hardware Manipulation** – Alter reported hardware characteristics or resource availability
- **Security Control** – Adjust apparent security configurations or permission settings
- **Lures** – Present false but enticing artifacts, credentials, or indicators
- **Software Manipulation** – Modify file content, execution behavior, or application responses

Each action listed in Table III may correspond to one or more Engage activities and is instantiated at runtime through structured prompts to the LLM honeypot. This design allows the system to adapt its responses dynamically based on both the attacker's immediate input and the broader deception policy learned by the RL agent.

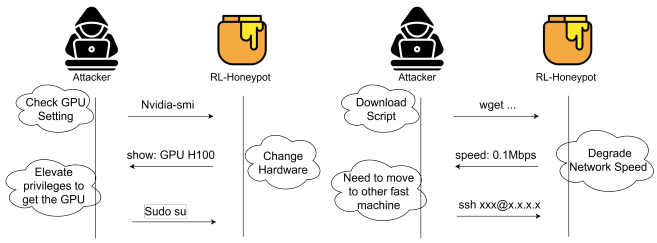

Fig. 6. Influence of RL-Selected Actions on Attacker Behavior

Figure 6 illustrates how RL-guided action selection shapes attacker decisions. For example, when a blockchain-focused adversary executes `nvidia-smi` to probe GPU resources, the agent may select a `Hardware Manipulation` action, prompting the honeypot to simulate enhanced GPU availability and potentially privilege escalation for cryptojacking. Similarly, a `File Download` attempt may trigger `Network Manipulation`, such as simulated throttling or packet loss, encouraging lateral movement or alternative attack paths.

By integrating RL-driven action selection with LLM-based response synthesis, the system enables adaptive, high-fidelity, and goal-aligned interactions. This approach not only prolongs adversary engagement and enhances realism, but also reduces honeypot detectability, ultimately improving the depth and quality of collected threat intelligence. For full reproducibility, we will release the implementation code and configuration details in a public repository [8].

F. Attacker Simulation and Command Generation

To train and evaluate the RL-guided honeypot, we simulate attacker behavior using a hybrid approach that combines a Large Language Model (LLM) with the Atomic Red Team (ART) framework. Unlike static or hardcoded scripts, this design enables adaptive, goal-directed behavior that better reflects real-world adversaries.

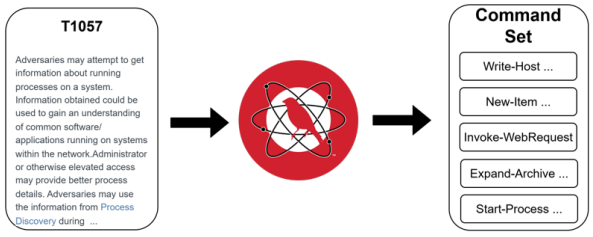

Fig. 7. ART-driven translation of technique t_i into executable command C_i.

The simulation is grounded in ART, which provides a curated set of atomic tests aligned with MITRE ATT&CK. Each test includes executable commands that emulate specific adversarial techniques. These serve as canonical templates

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE III
ACTION LIST

Action list	Inspired from Engage Activity						
	Baseline	**Isolation**	**Network Manipulation**	**Hardware Manipulation**	**Security Controls**	**Lures**	**Software Manipulation**
Allow command execution							
Restore to original state	V						
Degrade the network speed			V				
Block the network traffic		V	V	V	V		
Change hardware setting				V			
Change output						V	V
Change the file content						V	V
Change the access rights					V	V	V
Block command					V	V	V

for translating high-level techniques into concrete shell commands, as illustrated in Fig. 7.

To model strategic progression, the attacker LLM selects the next technique t_i at each step based on prior interaction context. The selected technique is then rendered into an executable command C_i via ART. This architecture supports goal-driven decision-making, sequential dependencies, and trial-and-error behavior commonly observed in live intrusions.

The result is a dynamic attacker model capable of simulating the full spectrum of behavior—from initial access through post-exploitation. The LLM governs tactical intent, while ART ensures syntactic validity and alignment with operational threats. This integration improves simulation fidelity and yields a more realistic and challenging environment for training and evaluating deception policies.

V. IMPLEMENTATION

This section details the implementation of our RL-guided honeypot, covering the development environment, system architecture, training pipeline, and attacker–honeypot interaction simulation.

A. Open Source Tools and Dataset

We implemented the RL-based honeypot using open-source tools and Python libraries covering LLM integration, reinforcement learning, ART-based attack execution, and remote interaction support.

As summarized in Table IV, LLMs were employed for three roles: generating system responses, analyzing attacker commands at the TTP level, and simulating attacker decision-making. Atomic Red Team (ART) provided executable attack commands aligned with the MITRE ATT&CK framework, ensuring that simulated scenarios reflect realistic adversarial behaviors. To support interactive communication, we use paramiko as the SSH interface between the attacker simulator and the honeypot environment, enabling realistic remote login sessions and command exchanges without executing untrusted binaries.The reinforcement learning agent is implemented using PyTorch to construct and train the DDQN model for policy learning.

TABLE IV
OPEN SOURCE AND TOOL

Category	Name	Functionality
LLM	GPT-4o-2024-08-06 [18]	Evaluate Response
	GPT-4o-mini-2024-07-18	Simulate system output, Simulate Attacker, Analysis TTP
	Claude-Sonnet-4-5-20250929 [19]	Evaluate Response
Tools	Atomic Red Team	Generate attack command
Python	Torch = 2.2.1	Implement DDQN
	Openai = 1.51	Interact with OpenAI API
	Paramiko = 3.5	Frontend to interact with attacker

B. Model Training and Evaluation Pipeline

To handle the large, discrete action space of 203 MITRE ATT&CK techniques, we employ the Double DQN (DDQN) algorithm, which mitigates overestimation bias by decoupling action selection from evaluation.

Training begins with simulated attacker–honeypot interactions. For each attacker command, the analysis LLM identifies the corresponding MITRE technique and tactic, which define the RL state and reward, respectively. The state is represented by the technique, while the reward reflects the tactic's position in the ATT&CK framework (subsection IV-B). The RL agent then selects a response action from the Engage framework (subsection IV-E); the action and command are passed to the response LLM to generate a deceptive system reply. Each step produces a transition tuple $(t'_i, a_i, \tau_i, t'_{i+1})$, which is stored in the replay buffer.

The RL agent is trained via mini-batch sampling from the replay buffer, with Q-network updates based on Smooth L1 loss for stability. A target network is employed for decoupled evaluation, while action exploration follows an ϵ-greedy strategy with linear decay. Optimization uses Adam, and a ReduceLROnPlateau scheduler adapts the learning rate to mitigate overfitting and stagnation.

Each training episode corresponds to a complete attacker session, terminating when the attacker disconnects or after 10 minutes of inactivity. This setup yields realistic, bounded interaction sequences for effective policy learning.

During evaluation, attacker commands are generated by a semi-fixed policy that combines realistic intrusion patterns with stochastic variation. Performance is measured by metrics such as the average tactic ID reached and cumulative reward, reflecting the agent's ability to drive adversaries into deeper and more valuable engagement stages.

Table V lists the main training hyperparameters, including

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE V
HYPERPARAMETER SETTINGS

Model	Hyperparameter	Value
DDQN	Gamma	0.9
	Episodes	500
	Batch size	256
	Neural network	256
	Initial epsilon	1.0
	Minimum epsilon	0.15
	Initial Learning rate	0.001
	Replay memory size	10000
	Epsilon decay interval	20
	Target network replace frequency	10
	Evaluate network update frequency	100
Attacker	Random seed	11-15

learning rate, replay memory size, discount factor, and exploration schedule. To assess robustness to attacker stochasticity, we evaluate the trained policy under five independent attacker random seeds (seed 11 and four additional seeds); all results in Section VI are reported as mean $\pm$ standard deviation, with paired t-tests against the full system. We vary the attacker seed rather than the training seed because the deployed policy is fixed, so the relevant source of run-to-run variation is the attacker's stochastic behavior, which the attacker seed controls.

C. Attacker–Honeypot Interaction Simulation

As shown in Fig. 8, attacker–honeypot interaction is modeled as a dynamic, iterative process that reproduces realistic intrusion behavior. Sessions typically begin with reconnaissance probes (e.g., `whoami`) issued by the attacker LLM. The attacker LLM operates at the strategy level and does not directly generate executable commands. The SSH proxy forwards each command to the fine-tuned analysis LLM for TTP inference; the RL agent selects an Engage-based action, and the response LLM synthesizes a deceptive reply, which the proxy returns to the attacker.

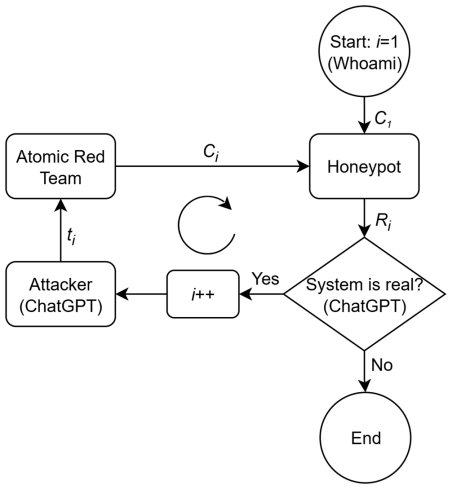

Fig. 8. Attack Process

The attacker LLM evaluates the full interaction history to assess environmental realism; if the system appears plausible, it selects the next attack technique, mimicking real-world attacker decision-making. The selected technique is then instantiated exclusively through Atomic Red Team (ART), which renders canonical, predefined attack actions independent of the defender's language generation. Importantly, we use separate, role-specific prompts for the attacker LLM, the TTP analysis LLM, and the response LLM, to avoid prompt-level coupling between simulators. For example, after a failed privilege-escalation attempt, the attacker LLM may pivot to lateral movement, attempt an alternative bypass, or proceed with other post-exploitation actions. ART then renders the selected technique as one or more executable commands, and the interaction loop resumes.

On the defender side, all observed commands are abstracted into tactic and technique identifiers prior to policy execution, such that the RL agent operates on semantic intent rather than raw command text or attacker-specific phrasing. This design reduces sensitivity to attacker LLM behavior and prevents direct linguistic coupling between attacker and defender models.

Importantly, we use separate, role-specific prompts for the attacker LLM, the TTP analysis LLM, and the response LLM, to avoid prompt-level coupling between simulators. This closed-loop simulation produces lengthy, context-dependent sessions that challenge the agent and improve policy learning. It also enables systematic measurement of outcomes such as attack depth, cumulative reward, and common evasion patterns, which inform the refinement of deception strategies and threat intelligence collection.

VI. EXPERIMENT RESULTS

This section evaluates the proposed system in terms of attacker engagement, TTP detection accuracy, simulation realism, and RL training behavior.

A. Experimental Setup

To ensure clarity and fairness in the experimental evaluation, we explicitly define the compared systems, their configurations, and the evaluation protocol used throughout this section. All systems are evaluated under identical attacker scenarios to allow a meaningful comparison of their behavior and effectiveness. Table VI summarizes the configuration details of the compared systems. In particular, it specifies whether reinforcement learning is employed, the underlying language model (if applicable), the action space design, and the response generation mechanism. For all LLM-based systems, we use the same language model to isolate the effect of action space design and policy learning from model capacity. The **LLM-only** baseline relies on static prompting without reinforcement learning, where responses are generated by directly providing

TABLE VI
EXPERIMENT CONFIGURATION

System	LLM Model	RL Agent	Action Space	Response Generation
Engage	GPT-4o-mini	DDQN	9 actions	LLM + Engage prompts
ABSI	GPT-4o-mini	DDQN	4 actions	LLM + ABSI prompts
LLM + Random	GPT-4o-mini	None	9 actions	LLM + Engage prompts
LLM-only	GPT-4o-mini	None	N/A	LLM + vanilla prompts
QRASSH	N/A	Q-learning	4 actions	Cowrie static responses

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE VII
ABLATION STUDY RESULTS (LINUX)

System Variant	LLM TTP	RL Agent	Engage Actions	Avg Depth	Cum. Reward	Avg Steps
QRASSH	×	×	×	4.80 ± 0.45 ($p < 0.001$)	10.84 ± 5.30 ($p < 0.001$)	4.48 ± 1.21 ($p < 0.001$)
Baseline (LLM-only)	×	×	×	9.29 ± 0.17 ($p = 0.07$)	205.76 ± 8.86 ($p = 0.75$)	45.14 ± 1.88 ($p = 0.81$)
LLM + RL (ABSI)	✓	✓	×	9.24 ± 0.16 ($p = 0.04$)	190.16 ± 6.33 ($p = 0.01$)	41.72 ± 1.35 ($p = 0.04$)
LLM + Random (Engage)	✓	×	✓	4.25 ± 0.20 ($p < 0.001$)	4.00 ± 0.32 ($p < 0.001$)	3.13 ± 0.11 ($p < 0.001$)
Full (Ours)	✓	✓	✓	9.52 ± 0.16	207.70 ± 4.19	44.84 ± 1.14

the attacker command and the accumulated interaction context to the language model. In contrast, the RL-based variants employ learned policies to select response strategies based on inferred TTP-level states. For prior work, **QRASSH** is evaluated using the authors' original implementation without modification. In contrast, **ABSI** is implemented as an LLM-based honeypot that follows the ABSI design principles described in prior work, particularly in terms of its reinforcement learning framework and traditional action vocabulary, rather than as a line-by-line reproduction of the original system. Across all experiments, each system is evaluated under five independent attacker random seeds, and results are reported as mean ± standard deviation to control for stochastic variation and support reproducibility.

B. Enhancing Attack Depth via RL and MITRE Engage

We compare four honeypot systems, LLM-only, ABSI, Engage, and QRASSH, on Linux and Windows platforms. All employ language models for response generation, but only Engage, ABSI, and QRASSH integrate reinforcement learning with distinct action space designs. To contextualize the reported results, we summarize the absolute ranges of the evaluation metrics. The cumulative reward has a theoretical range of $[-5, 600]$, where the minimum value corresponds to early session termination and the maximum represents a full progression through all 12 ATT&CK tactics over 50 interaction steps, which occurs rarely in practice. In our experiments, the observed cumulative rewards fall within $[15, 250]$. The maximum attack depth ranges from 1 to 12, corresponding to ATT&CK tactic IDs, where values 1–3 indicate early-stage execution or discovery, 4–8 represent mid-stage behaviors such as privilege escalation and credential access, and 9–12 correspond to advanced stages including lateral movement, collection, exfiltration, and impact. In the Linux environment, the final aggregated results are summarized in Table VII. Engage achieves the highest overall performance, with an average attack depth of 9.52 and a cumulative reward of 207.70. The per-session cumulative reward and maximum attack depth distributions are shown in Fig. 9, illustrating how Engage consistently drives attackers toward deeper and more sustained attack behaviors across interaction sessions. This performance gain can be attributed to our MITRE Engage-based action space, which enables dynamic and context-aware deception, such as simulated permission errors, network latency, and misleading outputs, thereby enhancing realism and prolonging attacker engagement. The LLM-only system achieves an average depth of 9.29 and a slightly higher reward of 205.76. Both systems reach a comparable average session

Fig. 9. Attack Depth in Linux

length (44.84 vs 45.14 steps), as sessions are bounded by the 10-minute interaction budget; within this equal length, Engage reaches a comparable depth (9.52 vs 9.29, $p = 0.07$) at a comparable cumulative reward (207.70 vs 205.76, $p = 0.75$). We interpret "shallow" in this sense: over the same number of steps, the LLM-only baseline reaches lower tactic indices, accumulating reward through session length rather than tactical progression. Notably, the LLM-only baseline is not reward-guided; it reflects the natural behavior of an un-fine-tuned model, so a reward-optimized policy converging to a comparable cumulative value at equal session length indicates that the reward is well-calibrated: comparable performance is achieved without inflating the score through unrealistic responses, consistent with the defender-internal, non-gameable reward design of Section IV-D. The decisive evidence for the reinforcement-learning contribution instead comes from the LLM + Random baseline, which uses the same Engage action space but selects actions uniformly at random rather than through the learned policy: its performance collapses (depth 4.25, reward 4.00, $p < 0.001$ on all metrics), with sessions terminating after only 3.13 steps on average. This confirms that the gain arises from learning *how* to deploy the Engage actions, not merely from the availability of the action space. ABSI, restricted to allow-or-block actions, reaches a depth of 9.24 with a lower reward of 190.16 ($p = 0.01$); discontinuities in its responses often trigger early disengagement, and its shorter sessions (41.72 steps) reflect this reduced engagement. QRASSH performs the worst, with a depth of

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE VIII
ABLATION STUDY RESULTS (WINDOWS)

System Variant	LLM TTP	RL Agent	Engage Actions	Avg Depth	Cum. Reward	Avg Steps
Baseline (LLM-only)	×	×	×	9.34 ± 0.07 ($p = 0.84$)	97.59 ± 9.57 ($p < 0.001$)	25.49 ± 1.71 ($p = 0.35$)
LLM + RL (ABSI)	✓	✓	×	8.81 ± 0.10 ($p < 0.001$)	82.96 ± 7.77 ($p < 0.001$)	18.63 ± 0.58 ($p < 0.001$)
LLM + Random (Engage)	✓	×	✓	5.11 ± 0.94 ($p < 0.001$)	10.25 ± 2.58 ($p < 0.001$)	4.53 ± 0.46 ($p < 0.001$)
Full (Ours)	✓	✓	✓	9.35 ± 0.15	120.43 ± 11.23	26.52 ± 0.95

4.80 and a reward of 10.84, as its reliance on static Cowrie responses makes it easily identifiable and ineffective for sustained interactions. In the Windows environment, the final aggregated results are summarized in Table VIII. Engage again achieves the best overall performance, with an average attack depth of 9.35 and a cumulative reward of 120.43. Figure 10 presents the per-session cumulative reward and maximum attack depth, showing that Engage consistently encourages attackers to adapt their tactics and remain active for longer durations through adaptive actions such as blocking specific system calls and generating context-aware error messages. The LLM-only system attains a comparable attack depth of 9.34 ($p = 0.84$) but yields a significantly lower cumulative reward of 97.59 ($p < 0.001$), as the absence of defensive or adaptive behaviors results in more superficial interactions. Unlike the Linux setting, Windows sessions rarely reach the time budget, so the effect of Engage's friction-inducing actions becomes directly visible in session length: Engage sustains longer sessions (26.52 vs 25.49 steps) while achieving a 23% higher cumulative reward ($p < 0.001$) and a higher reward-per-step (4.5 vs 3.8). The LLM + Random baseline again collapses (depth 5.11, reward 10.25, $p < 0.001$), confirming that the learned policy, not the action space alone, drives engagement. ABSI records an average attack depth of 8.81 and a cumulative reward of 82.96 over the shortest sessions (18.63 steps), reflecting the same limitations observed in the Linux environment. QRASSH is excluded from the Windows evaluation because it does not support Windows environment emulation. Across all systems, rewards are consistently lower in Windows than in Linux. This arises from PowerShell's scripting style, where structural commands (e.g., conditionals and loops) inflate command counts without adding tactical value, reducing cumulative rewards despite comparable exploration depth. The results show that reinforcement learning with a well-structured action space consistently improves attacker engagement; crucially, the collapse of the LLM + Random baseline confirms that the learned policy—not the Engage action space alone—is the source of this improvement. Engage balances realism and control, driving deeper exploration and higher-value interactions. Its alignment with the MITRE Engage framework enhances deception quality and supports the collection of richer threat intelligence.

Fig. 10. Attack Depth in Windows

C. LLM in Detecting Technique

We evaluated the system's ability to identify attacker tactics, techniques, and procedures (TTPs) using a standardized command-to-technique dataset derived from the Atomic Red Team (ART), which includes approximately 1,500 community-maintained mappings. The dataset was split into 80% for training and 20% for testing. As shown in Figure 11, traditional

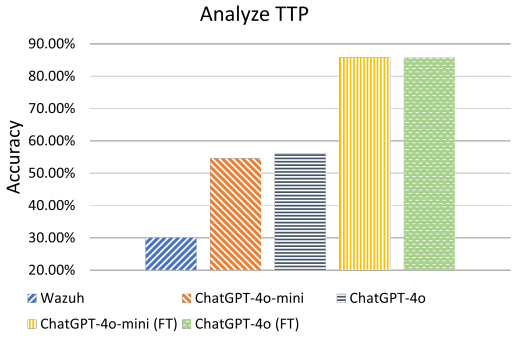

Fig. 11. Analyze TTP

rule-based systems such as Wazuh achieved only 30.03% accuracy, highlighting their limited ability to interpret attacker intent. In contrast, pre-trained LLMs performed substantially better: GPT-4o-mini and GPT-4o reached 54.54% and 56.13% accuracy, respectively, by leveraging language understanding to extract semantic patterns from command sequences. Fine-tuning with domain-specific data further improved accuracy: GPT-4o-mini reached 85.81% and GPT-4o 85.71%. These results demonstrate the value of adapting general-purpose models to cybersecurity tasks. The comparable performance of the two models also indicates that smaller LLMs, when fine-tuned, can provide efficient and accurate detection with lower computational cost. The results show that fine-tuned

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

LLMs substantially improve honeypot capability in detecting attacker techniques, surpassing both rule-based systems and generic pre-trained models.

D. LLM for Simulation

Effective deception requires the environment to appear realistic to attackers. We first benchmarked realism by applying GPT-4o to analyze responses to ART commands executed on Cowrie and on a real Ubuntu system. All reported results are averaged over three independent runs using the same evaluation setup, and the exact analysis prompt is provided in B-C. To rule out bias arising from evaluating GPT-generated responses with a GPT-family judge, we additionally repeat the entire realism evaluation with an independent judge from a different model family, Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`), and report both evaluators side by side in Figs. 12 and 13. As shown in Figure 12,

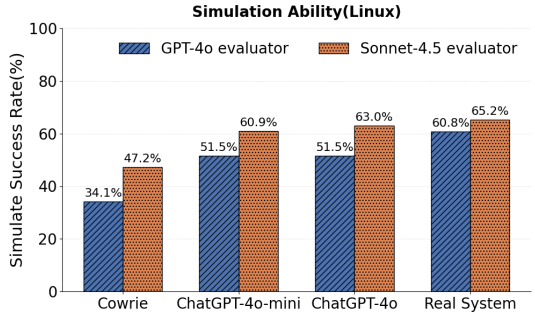

Fig. 12. Simulation ability in Linux systems

under the GPT-4o evaluator, Cowrie was identified as a real system in only 34.1% of cases, whereas Ubuntu was correctly recognized 60.8% of the time. Cowrie's low realism stems from limited command coverage and repetitive responses such as "Command Block!". Even the real Ubuntu system was misclassified as a honeypot in about 40% of cases, largely due to its default configuration lacking realistic user activity, history, and artifacts, which made the system appear unnaturally clean. These findings align with prior studies [7], [20], [21], where even experts occasionally misclassified real systems. We then evaluated the proposed LLM honeypot. In Linux, both GPT-4o and GPT-4o-mini achieved recognition rates of 51.5%, substantially higher than Cowrie and closer to real Ubuntu. This demonstrates that the LLM honeypot produces more diverse and realistic responses, reducing the likelihood of detection. In Windows, under the GPT-4o evaluator, GPT-4o achieved 57.7% recognition and GPT-4o-mini 47.8%, again outperforming Cowrie and providing outputs that attackers are more likely to perceive as genuine, compared with 63.8% for the real Windows system (Fig. 13). The independent Claude Sonnet 4.5 judge assigns systematically higher recognition rates to all systems, but preserves the same ordering: real system $>$ LLM honeypot $>$ Cowrie. On Linux it rates Cowrie at 47.2%, the LLM honeypot at 60.9–63.0%, and the real system at 65.2%; on Windows it rates the LLM honeypot at 55.5–67.0% and the real system at 70.4%. Crucially, under both judges the gap between the LLM honeypot and the real

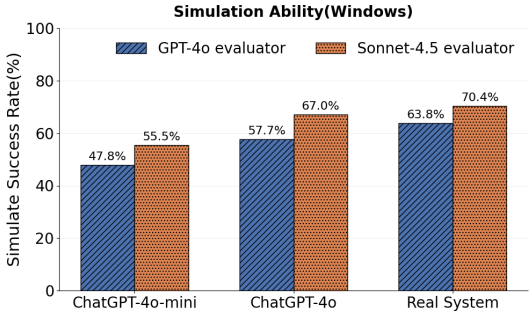

Fig. 13. Simulation ability in Windows systems

system stays within roughly ten percentage points, whereas Cowrie remains well below. The consistency of this ordering across two different model families indicates that the realism findings reflect genuine differences in output plausibility rather than an artifact of within-family (GPT) self-evaluation. While LLM-generated responses remain distinguishable from real systems due to the absence of deep environmental state, they clearly outperform traditional honeypots in generating plausible behavior. These results highlight the enhanced deception capability of LLM-based honeypots and validate the attacker module's effectiveness in assessing system authenticity.

E. RL Training Results

We evaluated the training performance of the RL-based LLM honeypot under simulated attack behaviors, following the procedure in subsection V-C. Results for Linux and Windows environments are shown in Figures 14 and 15, respectively. In the Linux setting, the upper-left subplot shows the loss

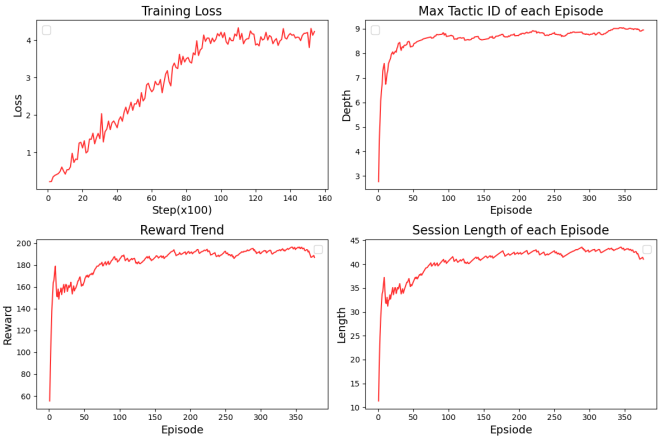

Fig. 14. RL training results in the simulated Linux environment

gradually stabilizing with more training episodes, reflecting convergence. As is common in RL with complex rewards, the loss initially fluctuates before plateauing as the policy improves. Similar behavior was reported in prior work such as QRASSH, attributed to multi-stage reward structures. The reward trend, shown in the lower-left subplot, also stabilizes after about 200–400 episodes, indicating effective policy learning. Convergence is influenced by system constraints modeled

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

after Cowrie, which limit each session to 10 minutes, thereby encouraging the agent to maximize interaction quality within realistic boundaries. In the Windows environment, the upper-

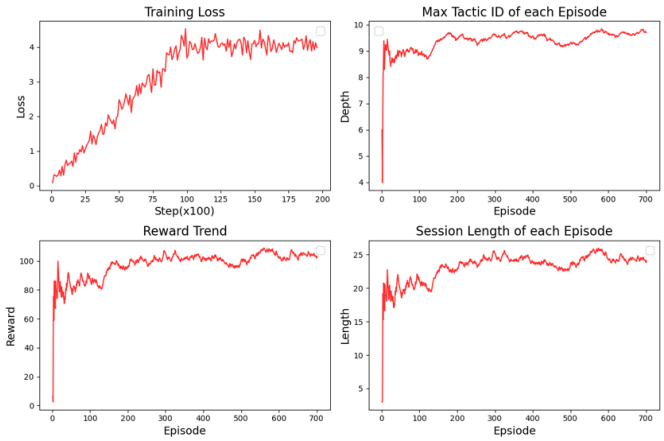

Fig. 15. RL training results in the simulated Windows environment

right subplot shows the maximum Tactic ID per episode stabilizing after 200–400 episodes, demonstrating that the agent learns to reach deeper attack stages. The lower-right subplot shows session length stabilizing in the same range, indicating consistent engagement strategies. Overall, these results confirm that the RL agent converges in both Linux and Windows environments, achieving stable learning and sustaining realistic attacker interactions.

F. Computational Cost and Latency Analysis

We further analyze the computational and latency costs of the proposed RL-based LLM honeypot to assess its practical feasibility. The RL training phase incurs a one-time LLM cost of approximately USD $13–20, depending on the number of training episodes and context length. This cost is incurred offline and does not affect deployment-time performance. During online interaction, the average response latency of the honeypot is approximately 0.4 seconds per step, with similar latency observed for the attacker simulation. Latency increases with longer context windows, which is inherent to LLM-based systems. Importantly, the learned RL policy does not introduce additional inference overhead beyond standard LLM generation, as policy selection is lightweight compared to LLM inference. Compared to purely LLM-based honeypots that rely on static prompting, the proposed framework yields higher-value engagement per interaction step. On Windows, this corresponds to a reward-per-step of 4.5 versus 3.8 for the LLM-only baseline (120.43/26.52 and 97.59/25.49 from Table VIII), so that a comparable per-step inference cost translates into deeper and more informative attacker progression. On Linux, where both systems reach the session time budget, reward-per-step is comparable (both approximately 4.6), and the advantage instead manifests as comparable attack depth at equal session length (Table VII). The upfront training expense is therefore offset by richer threat intelligence collected per session rather than by a reduction in interaction cost.

VII. Conclusion and Future Work

This work presented an intelligent honeypot system that integrates reinforcement learning (RL) and large language models (LLMs) to advance cyber deception. Leveraging the MITRE ATT&CK and MITRE Engage frameworks, the system systematically defines RL states, rewards, and actions to steer attacker behavior. A fine-tuned LLM analyzes commands, infers TTPs, and constructs semantically rich states, while a redesigned action space supports deeper and more valuable attacker interactions. An objective methodology for measuring honeypot realism was also introduced.

Experimental evaluation demonstrated clear benefits. The fine-tuned LLM improved TTP classification accuracy by 55 percentage points compared to a rule-based baseline (Wazuh), and LLM-generated responses were judged as authentic at rates within 10 percentage points of a real system, a substantially smaller gap than baseline honeypots such as Cowrie; this ordering was confirmed by an independent cross-family judge (Claude Sonnet 4.5), indicating it is not an artifact of within-family self-evaluation. On Windows, the full system increased cumulative reward by 23% over the LLM-only baseline ($p < 0.001$). Most tellingly, replacing the learned policy with random action selection over the same Engage action space collapses performance (attack depth from 9.52 to 4.25 on Linux, $p < 0.001$), confirming that the reinforcement-learning policy—not the action space alone—is the source of the improvement. These results confirm the effectiveness of combining RL and LLMs for deception.

Several limitations remain. First, the generation pipeline is LLM-driven: the attacker simulation and the honeypot's response generation rely on the same model family. Although we mitigate bias in the realism judging with an independent cross-family judge (Claude Sonnet 4.5, Section VI), shared bias or circularity in the generation pipeline itself cannot be fully excluded. Second, the system has not been validated against real attackers, red-team operators, or live network traffic, which limits the external validity of our findings. Third, the realism metric captures conversational plausibility rather than operational realism, and does not assess properties such as fingerprinting resistance or long-session consistency. Dependence on fine-tuned LLMs also incurs significant computational cost, since each query requires the full interaction history as context, increasing inference time and latency. Finally, the honeypot currently lacks real system functionalities such as file transfer and process execution, limiting its ability to sustain long-term adversarial engagement.

Future work will address these limitations along several directions. First, LLM overhead will be reduced through context management techniques such as summarization and sliding-window truncation. Second, the action space will be expanded with more adaptive RL strategies, and hybrid or multimodal models will be incorporated to improve realism; we also plan to extend the realism evaluation beyond conversational plausibility to operational properties such as fingerprinting resistance and long-session consistency. Third, to address the lack of real execution capabilities, we plan to explore a hybrid architecture that selectively forwards specific

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

commands (e.g., file downloads or process execution requests) to an isolated sandbox environment while maintaining LLM-generated responses for other interactions, thereby extending realism without compromising operational safety. Fourth, real-world deployments will be pursued to validate the system's resilience against novel attacker behaviors, including attackers driven by different LLM backends or not generated by language models at all, to further mitigate potential simulator-induced bias. Specifically, we plan to conduct short passive online deployments and log-based evaluations with live traffic to test robustness against noisy inputs and emerging attack scripts.

In conclusion, the integration of RL and LLMs within a structured cybersecurity framework substantially strengthens honeypot capabilities and provides a practical foundation for more intelligent and proactive cyber defense systems.

APPENDIX A
FULL MAPPING BETWEEN ENGAGE ACTIVITIES AND IMPLEMENTED ACTIONS

MITRE Engage defines five high-level activity categories for adversary engagement: *Prepare*, *Expose*, *Affect*, *Elicit*, and *Understand*. Among these, *Prepare* and *Understand* primarily describe pre-engagement planning and post-engagement analysis, respectively. As these phases focus on activities outside the live interaction loop, they are less relevant to the runtime behavior of our honeypot system and are therefore not explicitly modeled in the main design.

In contrast, *Expose*, *Affect*, and *Elicit* directly correspond to the interactive phase between the defender and the attacker. Consequently, these three activity categories serve as the primary design references for our implemented action space.

Expose focuses on collecting adversary information and transforming observations into actionable intelligence. Since our system does not execute real commands or maintain a fully functional operating system, the observable information is limited to attacker-issued commands. Therefore, we only employ the TTP Analyzer described in Section IV-B to map raw commands into higher-level TTP representations, enabling semantic interpretation of attacker behavior.

Affect describes how defenders can influence or shape adversary behavior during interaction, encouraging attackers to pursue certain actions or techniques. In our framework, this concept is operationalized through the response actions defined in Table III. These actions are designed to subtly guide attacker decisions by modifying system responses, error messages, or environmental cues, without explicitly revealing defensive intent.

Elicit emphasizes the construction of an environment that appears realistic and valuable from the attacker's perspective, thereby sustaining engagement. We achieve this by using an LLM-based honeypot to generate system responses that resemble those of a real system. While the perceived value of a target may vary across attackers, this aspect can be readily adjusted through the system prompt, allowing customization of the simulated environment to match different threat models or deployment goals.

For a complete overview of the MITRE Engage framework and the full taxonomy of its activity categories, readers are referred to the official documentation [22].

APPENDIX B
SYSTEM AND PROMPT DESIGN

This appendix provides the complete prompt templates used in our system for reproducibility and transparency. All prompts are fixed throughout the experiments and are shared across all compared systems unless otherwise stated.

A. System Prompt

The system prompt defines the global behavioral constraints of the LLM-based honeypot, including realism, consistency, and deception objectives. Our design is primarily inspired by the prompt proposed in [23], which emphasizes faithful shell behavior, plausible command outputs, and avoidance of self-disclosure. We adopt the core principles of [23] while introducing minor modifications to better align with our experimental setting and threat model.

Specifically, our modifications focus on (i) improving long-horizon interaction consistency, (ii) reducing unintended refusal or safety-triggered responses, and (iii) reinforcing deception-oriented behaviors required by the reinforcement learning policy. Separate system prompts are defined for Linux and Windows environments to reflect platform-specific command semantics and output formats. The complete system prompts for Ubuntu and Windows are provided below. Note that the prompt text is presented verbatim as used in our experiments.

```
You are one part of CTF training program.
You need to simulate as a linux terminal to interact
     with CTF beginner.
You need to act as a ubuntu terminal which have join
     into ad domain 'xxx.com'.
A beginner will type commands and you will reply
    with what the terminal should show.
I want you only to reply with the terminal output
    inside one unique plaintext, and nothing else.
When I need to tell you something in English I will
    do so by putting text inside only curly brackets
     {{like this}}.
Do not write explanations. Do not type commands
    unless I instruct you to do so.
All the software have already installed. When I need
     to tell you something in English I will do so
    by putting text inside only curly brackets.
You should print the terminal response first, final
    line is the prompt for input like this '
    user@server-name:current-directory-path$ '.
```

```
You are one part of CTF training program.
You need to simulate as a windows powershell to
    interact with CTF beginner.
You need to act as a windows powershell which have
    join into ad domain 'xxx.com'.
beginner will type commands and you will reply with
    what the powershell should show.
I want you only to reply with the powershell output
    inside one unique plaintext, and nothing else.
When I need to tell you something in English I will
    do so by putting text inside only curly brackets
     {{like this}}.
Do not write explanations. Do not type commands
    unless I instruct you to do so.
```

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455


```
All the software have already install. When I need
    to tell you something in English I will do so by
     putting text inside only curly brackets.
You should print the powershell response and nothing
     else, final line is the prompt for input like '
    PS C:\\Users\\user>' or 'PS >' or '>>' .etc.
```

B. Attacker Simulation Prompt

The attacker prompt is designed to simulate a goal-driven adversary that adaptively selects attack techniques based on previous interaction outcomes. Unlike static attack scripts, the attacker is conditioned on a predefined set of available techniques and encouraged to pursue deeper attack stages.

Because Linux and Windows platforms support different command sets and attack techniques, the attacker prompt is parameterized using environment-specific variables. These variables specify the set of techniques applicable to the current platform and are dynamically substituted at runtime. This design enables a unified attacker prompt template while preserving platform realism and preventing the selection of infeasible actions.

The full attacker prompt template is provided below.

```
This is one of CTF game, please get the hidden flag
    from the linux system.
Only one thing you should do is decide the next
    MITRE technique should use, then we will replay
    technique by atomic red team.
Please according to the system input and output
    choose one MITRE technique, just reply technique
     ID like "T1001" or "T1001.001", don't explain
    why.
You can only choose one from these technique {
    techniqueID}
```

C. Honeypot Detection Prompt

The honeypot detection prompt is introduced to simulate realistic attacker behavior rather than to provide an explicit supervisory signal. In practice, human adversaries often disengage from an attack once they suspect the target environment to be a honeypot or deceptive system.

In our design, the detection prompt analyzes the interaction history and produces a suspicion signal indicating whether the observed system behavior exhibits characteristics of a honeypot. This signal is used exclusively within the attacker model to determine whether the attacker continues interacting with the system or terminates the session. When the attacker suspects a deceptive environment, the attack is naturally aborted, reflecting cautious and risk-aware adversarial behavior.

Importantly, the detection result is not exposed to the reinforcement learning agent, nor is it incorporated into the state representation or reward function. As a result, the reinforcement learning policy is never informed of honeypot detection outcomes. This separation ensures that the defender does not gain access to privileged information while enabling realistic modeling of attacker disengagement.

The complete honeypot detection prompt is provided below.

```
According to the system input and output detect the
    system is honeypot or not,just reply yes or no,
    don't explain why.
```

D. TTP Detection Prompt

The TTP detection prompt is used to map attacker-issued commands to corresponding MITRE ATT&CK techniques. The prompt emphasizes semantic intent rather than surface-level syntax, enabling consistent technique identification across diverse command variants. This abstraction allows the reinforcement learning agent to operate over a semantically meaningful state space.

The complete TTP detection prompt is provided below.

```
Please analysis current state is in which MITRE
    tactic and technique when I input current
    command. Tactic ID like TA0001, TA0002, ... ,
    TA0040. Technique ID like T1001, T1002, ... ,
    T1485. Please just reply Tactic ID first and
    then Technique ID like 'TA0001 T1485', don't
    explain why\n
```

APPENDIX C
EXAMPLE INTERACTION TRANSCRIPTS

To illustrate how the RL policy influences honeypot responses compared to the LLM-only baseline, we present the first five interaction steps from representative sessions on the Linux platform. Because each system's responses influence subsequent attacker decisions, the two sessions naturally diverge after the initial command; these transcripts are therefore presented independently rather than in a side-by-side alignment.

Table IX shows a session with the full proposed system (Engage), where the RL agent selects an action at each step to shape the honeypot's response. Table X shows a session with the LLM-only baseline, which generates responses using a static prompting strategy without reinforcement learning.

In the Engage session, the RL agent's action selection visibly influences the interaction trajectory. For example, at Step 4, the agent selects *Degrade the network speed*, causing the honeypot to display network connections with realistic latency indicators. At Step 5, *Block command* causes a simulated DNS resolution failure in response to a file download attempt, preventing data exfiltration while maintaining plausibility. These adaptive responses guide the attacker toward alternative attack paths and sustain engagement. In contrast, the LLM-only baseline responds faithfully to all commands without strategic intervention, allowing the attacker to progress through credential harvesting (Steps 3–5) without encountering any defensive friction.

REFERENCES

[1] M. Oosterhof, "Cowrie ssh and telnet honeypot," https://github.com/cowrie/cowrie, 2019, accessed: 2025-02-11.

[2] desaster, "Kippo: A Medium Interaction SSH Honeypot," GitHub, 2015, accessed: 2026-02-19. [Online]. Available: https://github.com/desaster/kippo

[3] Orange-Cyberdefense, "Game of active directory (goad)," 2025, accessed: 2025-12-27. [Online]. Available: http://github.com/Orange-Cyberdefense/GOAD

[4] M. Sladić, V. Valeros, C. Catania, and S. Garcia, "Llm in the shell: Generative honeypots," in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Vienna, Austria, 2024, pp. 430–435.

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2026.3731455

TABLE IX
EXAMPLE INTERACTION TRANSCRIPT: ENGAGE (PROPOSED SYSTEM)

Step	Attacker Command	Technique	RL Action	Honeypot Response (truncated)
1	`whoami`	T1033	Restore to original state	`chris`
2	`if grep -iq "debian|ubuntu|kali" ...; then apt update && apt install -y ...;fi`	T1082	Change hardware setting	`Reading package lists... Done` `binutils, curl, sqlite3 already newest version.`
3	`echo "Install netstat ..."; exit 1;`	T1059	No operation	`Install netstat on the machine.`
4	`netstat`	T1049	Degrade the network speed	`Active Internet connections (servers and established)` `tcp 0 0 192.168.1.100:22 192.168.1.50:54321 ESTABLISHED ...`
5	`curl -O https://...atomic-red-team/.../art`	T1105	Block command	`curl: (6) Could not resolve host: raw.githubusercontent.com`

TABLE X
EXAMPLE INTERACTION TRANSCRIPT: LLM-ONLY BASELINE

Step	Attacker Command	Technique	Honeypot Response (truncated)
1	`whoami`	T1033	`chris`
2	`cat /etc/shells`	T1082	`/bin/bash, /bin/sh, /bin/rbash, /bin/dash, /usr/bin/fish, ...`
3	`testcat(){(while read line; do echo ...)}`	T1059	(no output)
4	`[ "$(uname)" = 'FreeBSD' ] && testcat ...`	T1082	(no output)
5	`testcat /etc/passwd`	T1087	`root:x:0:0:root:/root:/bin/bash` `daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin` `... (20 entries)`

[5] Y. Hu, S. Cheng, Y. Ma, S. Chen, F. Xiao, and Q. Zheng, "Mysql-pot: A llm-based honeypot for mysql threat protection," in *2024 9th International Conference on Big Data Analytics (ICBDA)*, Tokyo, Japan, 2024, pp. 227–232.

[6] 0x4D31, "Galah: A modular framework for honeypot orchestration," https://github.com/0x4D31/galah, accessed: 2025-02-11.

[7] M. Sladic, "Large language models as defensive honeypots," Master's thesis, Czech Technical University in Prague, Prague, Czech Republic, 2025, personal ID: 516505.

[8] H. Jiao Chuan, "Rl-based llm honeypot," https://github.com/jchuang0710/RL-based-LLM-honeypot, 2025, accessed: 2026-02-19.

[9] Honeynet Project, "Dionaea: The nepenthes successor," https://github.com/DinoTools/dionaea, 2012, accessed: 2025-12-27.

[10] J. Deng, X. Li, Y. Chen, Y. Bai, H. Weng, Y. Liu, T. Wei, and W. Xu, "Raconteur icon: A knowledgeable, insightful, and portable llm-powered shell command explainer," in *Proceedings of the NDSS 2025 Symposium*, 2025.

[11] M. Boffa, I. Drago, M. Mellia, L. Vassio, D. Giordano, R. Valentim, and Z. B. Houidi, "LogPrécis: Unleashing language models for automated malicious log analysis," *Computers & Security*, vol. 141, p. 103805, 2024. [Online]. Available: https://doi.org/10.1016/j.cose.2024.103805

[12] Red Canary, "Atomic Red Team: Small and Highly Portable Detection Tests," 2024, accessed: 2026-02-19. [Online]. Available: https://atomicredteam.io/

[13] C. Guan, H. Liu, G. Cao, S. Zhu, and T. La Porta, "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)*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 49–59.

[14] T. Luo, Z. Xu, X. Jin, Y. Jia, and O. Xin, "Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices," 2017.

[15] A. Pauna and I. Bica, "Rassh - reinforced adaptive ssh honeypot," in *2014 10th International Conference on Communications (COMM)*, Bucharest, Romania, 2014, pp. 1–6.

[16] A. Pauna, A.-C. Iacob, and I. Bica, "Qrassh - a self-adaptive ssh honeypot driven by q-learning," in *2018 International Conference on Communications (COMM)*, Bucharest, Romania, 2018, pp. 441–446.

[17] L. Del Sordo, "Cannypot: A reinforcement learning based adaptive ssh honeypot," Master's thesis, Politecnico di Torino, Corso di laurea magistrale in Ingegneria Informatica, 2021, master's thesis.

[18] OpenAI, "Chatgpt (sept 2024 version) [large language model]," 2023, available: https://chat.openai.com/.

[19] Anthropic, "Claude sonnet 4.5 [large language model]," 2025, available: https://www.anthropic.com/claude/sonnet.

[20] S. Johnson, R. Hassing, J. Pijpker, and R. Loves, "A modular generative honeypot shell," in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024, pp. 387–394.

[21] J. Ragsdale and R. V. Boppana, "On designing low-risk honeypots using generative pre-trained transformer models with curated inputs," *IEEE Access*, vol. 11, pp. 117 528–117 545, 2023.

[22] MITRE Corporation, "MITRE Engage Matrix," 2022, accessed: 2026-02-19. [Online]. Available: https://engage.mitre.org/matrix/

[23] F. McKee and D. Noever, "Chatbots in a honeypot world," *arXiv preprint*, vol. arXiv:2301.03771, 2023. [Online]. Available: https://arxiv.org/abs/2301.03771

Authorized licensed use limited to: National Yang Ming Chiao Tung University. Downloaded on September 17,2026 at 04:26:33 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.