IEEE ComSoc IEEE Communications Society | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS | IEEE Transactions on Machine Learning in Communications and Networking

Received 4 October 2025; revised 11 January 2026 and 6 March 2026; accepted 11 April 2026. Date of publication 17 April 2026; date of current version 28 April 2026.

The associate editor coordinating the review of this article and approving it for publication was A. Sultana.

*Digital Object Identifier 10.1109/TMLCN.2026.3684465*

# Reinforcement Learning-Driven Honeypots: Tactic-Based Defense for Industrial Control Systems

**YING-DAR LIN[1] (Fellow, IEEE), RASUL MANKAEV[2], DIDIK SUDYANA[3] (Member, IEEE), YUAN-CHENG LAI[4], AND REN-HUNG HWANG[5] (Senior Member, IEEE)**

[1]Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300025, Taiwan
[2]EECS International Graduate Program, National Yang Ming Chiao Tung University, Hsinchu 300025, Taiwan
[3]Computer and Network Center, National Cheng Kung University, Tainan 701401, Taiwan
[4]Department of Information Management, National Taiwan University of Science and Technology, Taipei 106335, Taiwan
[5]College of Artificial Intelligence, National Yang Ming Chiao Tung University, Tainan 711010, Taiwan

CORRESPONDING AUTHOR: D. SUDYANA (didik@gs.ncku.edu.tw)

**ABSTRACT** Traditional honeypots in Industrial Control Systems (ICS) often fail to sustain attacker engagement, exposing their deception after a few predictable responses and limiting visibility to early reconnaissance. This weakness reduces their value for defenders seeking insight into later stages of the kill chain. To address this, we propose an adaptive honeypot that embeds reinforcement learning (RL) into an ICS honeypot, enabling it to shape responses in real time based on attacker behavior. Guided by the MITRE ATT&CK for ICS framework, each network interaction is mapped to a tactic, and the RL agent applies tabular Q-learning to select deception strategies such as delaying, modifying, or blocking ICS protocol messages to drive sessions deeper into the attack chain. We evaluated the system using scripted Metasploit-based attacks derived from MITRE Campaigns across three honeypot configurations: Static0 (default ICS honeypot), Static1 (modified Modbus handler to support uncommon but valid function codes), and Dynamic (RL-driven honeypot). Results show that while static honeypots stall at tactic levels 4 and 7, the RL-driven honeypot reaches the latest tactic within 700 attack–response cycles and more than doubles the average engagement time. Even in the first 400 sessions, a small portion of attacks already reach the final tactic, providing early threat intelligence. These findings show that an RL-enhanced honeypot can adapt autonomously, extract deeper threat intelligence, and give defenders broader visibility across ICS tactics.

**INDEX TERMS** Industrial control systems, adaptive honeypot, MITRE ATT&CK, reinforcement learning.

## I. INTRODUCTION

INDUSTRIAL Control Systems (ICS) are cyber-physical platforms that combine digital computing with real-world operations in critical sectors such as energy, water, and manufacturing [1]. While these systems enable automated control and high efficiency, they have also become attractive targets for skilled attackers. The 2015 Ukraine electric power attack, for example, demonstrated how intruders breached both the corporate IT (information technology) network and the operational technology (OT) control layer in a multistage kill chain, ultimately disabling large portions of the grid [2]. This case highlights the evolving risks to ICS, where outdated communication protocols and safety-critical processes remain vulnerable points of exploitation.

Despite ongoing improvements to firewalls, Intrusion Detection Systems (IDS), and other traditional defenses, attackers often find new exploits or pivot through IT networks before compromising OT devices [3]. Honeypots, on the other hand, offer a valuable complement to these existing solutions by attracting attackers to a decoy system, capturing their Tactics, Techniques, and Procedures (TTPs), and providing intelligence that can enhance detection rules and

incident response. However, while IT honeypots typically handle text-based protocols (for example, SSH or HTTP), an ICS honeypot must simulate binary, time-sensitive protocols (for example, Modbus, S7comm), accurately reflect real process states, and handle the complexities of legacy industrial equipment to remain convincing [4]. Existing ICS honeypots such as Conpot [5] and DiPot [6] are considered static because they use a fixed set of predefined responses that are scripted and not adaptive to attacker behavior. Once attackers observe a few interactions, they can easily recognize the repetitive patterns and identify the deception.

Static honeypots typically yield short sessions, providing minimal information on advanced ICS intrusion strategies. Because they rarely capture multi-stage escalation, defenders often gain only superficial knowledge of how adversaries exploit ICS protocols. A more adaptive approach can prolong attacker sessions by dynamically adjusting responses to each malicious command. Machine learning (ML) offers a path to such adaptiveness, as a model can learn to modify honeypot behavior in response to changing attack patterns. However, traditional supervised ML requires preexisting datasets that quickly become obsolete. In contrast, Reinforcement Learning (RL) continuously refines its policy through real-time adversarial interaction, balancing exploration of new strategies with exploitation of known ones. This adaptive paradigm suits ICS environments, where legacy protocols and multi-stage infiltration demand defensive methods that evolve with the attacker.

To overcome the limitations of static ICS honeypots, this paper proposes an RL-driven tactic-based ICS honeypot. In this approach, each attacker action is mapped to a tactic stage defined by the MITRE ATT&CK for ICS (MITRE ICS in short), allowing the system to track multi-step intrusions in a structured manner. We incorporate MITRE ICS to map each adversarial move to a tactic index, along with details of the ICS protocol (for example, Modbus function codes) [7]. The RL agent selects deceptive actions, such as delaying responses, injecting false data, or blocking commands, to prolong engagement and push adversaries into higher tactics. By explicitly rewarding deeper kill chain progression, this approach yields more granular threat intelligence, allowing defenders to observe attacker techniques across multiple ICS stages.

In order to evaluate the effectiveness of tactic-driven ICS honeypots, this study investigates three key questions. (1) Whether an RL-driven honeypot can draw attackers deeper into the ICS kill chain compared to static decoys, as reflected in tactic depth, session length, reward signals, and the learned Q-table. (2) How the amount of information given to the RL agent influences learning efficiency and the quality of deception. (3) How tuning RL hyperparameters affects the trade-off between fast progression and stable, consistent engagement. Together, these investigations provide practical deployment insights and a clearer view of how adaptive deception strengthens industrial network defense.

Our work contributes in three key ways. First, we address the limitations of current ICS honeypots, emphasizing their static design and lack of adaptability to multistage adversaries exploiting time-sensitive protocols. Second, we propose a novel RL framework that integrates MITRE ICS tactics, protocol context, and a rich set of deception actions, allowing dynamic engagement extension and revealing deeper attack phases. Finally, we demonstrate the efficacy of our RL-driven approach through comprehensive evaluations with real simulations of the ICS protocols and MITRE ICS attack scenarios, showing a marked improvement in session depth, engagement duration, and defense against advanced threats relative to static baselines.

The remainder of this paper is organized as follows: Section II provides background information on ICS, honeypots, RL, and related work. Sections III and IV detail our problem statement and proposed solution, respectively. Section V discusses the implementation. Section VI provides experimental results. Finally, Section VII concludes with key findings and future research.

## II. BACKGROUND AND RELATED WORKS

This section reviews the foundational concepts and prior studies relevant to our work. It first introduces ICS and its security challenges, followed by honeypot designs for both IT and ICS domains. Next, it outlines the MITRE ATT&CK and MITRE Engage frameworks that guide tactic mapping and deception strategies. Finally, it surveys applications of RL in cybersecurity and summarizes related works in adaptive honeypots.

### A. INDUSTRIAL CONTROL SYSTEM (ICS)

ICS orchestrates critical infrastructure processes in energy, water, and manufacturing by integrating physical equipment with computing components [1]. Since many ICS installations remain operational for 15-20 years, they often rely on legacy protocols and devices receiving minimal security updates. This long lifespan creates vulnerabilities that attackers exploit to disrupt or hijack essential services. Unlike standard IT protocols, ICS protocols such as Modbus, S7comm, and DNP3 are often binary, stateful, and proprietary, complicating typical network defenses.

A key component of ICS is the Programmable Logic Controller (PLC), which reads sensor inputs, executes control logic, and issues commands to actuators in real time. If compromised, a PLC can be misused to send unauthorized read/write instructions, launch denial-of-service (DoS) attacks, or alter commands in a man-in-the-middle (MitM) attack [8]. Because PLCs interact directly with physical processes, such intrusions can lead to severe consequences, including power outages, water contamination, or equipment failures.

Advanced persistent threats (APTs) often exploit protocol weaknesses to achieve these outcomes. For instance, Modbus TCP lacks authentication and allows unauthorized

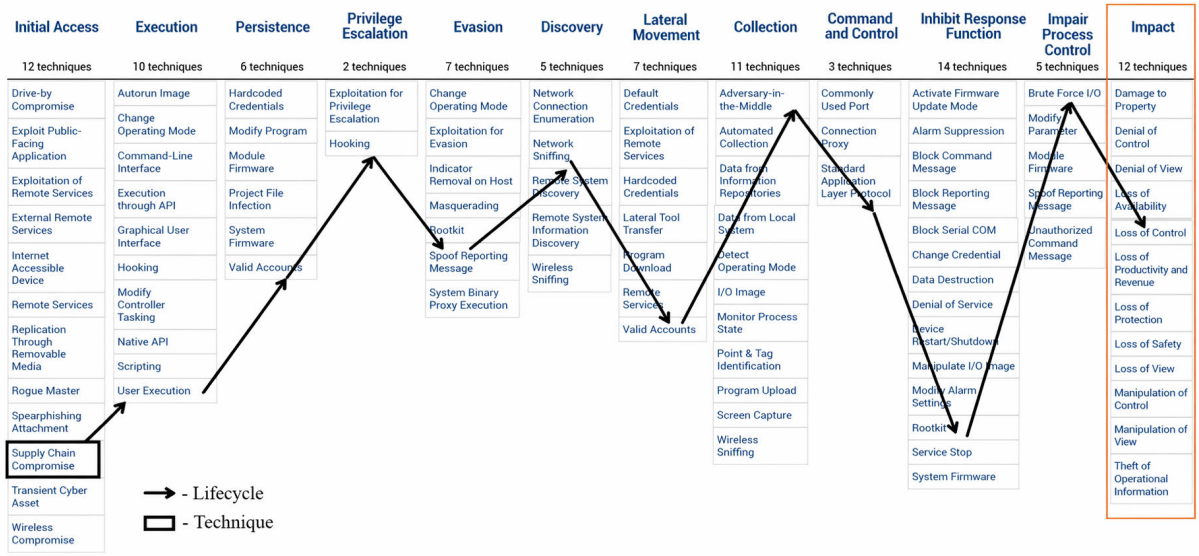


**Fig. 1. MITRE ATT&CK for ICS - tactics and techniques.**

register access [4], while DNP3 can be susceptible to replay attacks if not secured [9]. Real incidents emphasize the risks: the 2015 Ukraine Electric Power Attack demonstrated multi-stage tactics that disabled the grid [2], followed by similar campaigns in 2016 [10] and 2022 [11]. The MITRE ATT&CK Campaigns framework captures such intrusions, mapping adversary progression from reconnaissance to physical disruption, with tactics such as Initial Access, Inhibit Response Function, and Impair Process Control.

### B. HONEYPOTS

A honeypot is a decoy system designed to attract attackers, log their activities, and enrich security intelligence. In IT contexts, honeypots typically handle text-based protocols such as SSH or HTTP, storing commands or payloads in a stateless manner that simplifies their design [15]. In contrast, OT honeypots must simulate timing-sensitive binary ICS protocols and realistic process states, where mismatches in emulation can expose the deception or even create safety risks if linked to live control loops.

Most ICS honeypots remain static, relying on prescripted responses that fail to adapt to attacker input. Such designs can detect scans or simple exploits, but often produce repetitive or unrealistic replies once adversaries probe deeper into commands, yielding limited intelligence. Some provide only partial emulation—for example, handling Modbus read coils but not advanced function codes [4]. By contrast, dynamic honeypots adjust responses in real time, introducing realistic delays or data variations.

### C. MITRE ATT&CK AND MITRE ENGAGE

The MITRE ATT&CK framework for ICS defines twelve tactics that adversaries can use to compromise operational technology, such as Initial Access, Inhibit Response Function (disabling ICS alarms), and Impair Process Control (corrupting PLC logic) [16]. These tactics structure the attacker's progression from reconnaissance to physical disruption. Figure 1 presents these tactics in a structured way, illustrating how an attacker transitions from reconnaissance to damaging physical operations.

Beyond tactics, Figure 1 also illustrates the broader ATT&CK structure, where each tactic is linked to a set of techniques that describe how it can be executed in practice. For instance, the "Initial Access" tactic may involve techniques such as exploiting remote services or spearphishing, while "Impair Process Control" includes modifying control logic or disrupting safety functions. These techniques provide concrete, observable behaviors that map the abstract progression of tactics into specific attacker actions. Taken together, the tactics and their supporting techniques form a lifecycle view of adversary behavior.

Alongside ATT&CK, MITRE Engage focuses on adversary engagement, for example, how defenders might deliberately respond or mislead attackers [17]. While ATT&CK describes the attacker's perspective, Engage outlines defensive techniques, for instance, feeding false information or slowing the attacker's progress via subtle misconfigurations. This concept of adaptive, preplanned deception aligns closely with the usage of a honeypot.

### D. REINFORCEMENT LEARNING

RL is a framework in which an agent interacts with an environment by observing states, taking actions, and receiving rewards [23]. Through iterative feedback, the agent learns to maximize long-term returns, making RL well-suited for adversarial settings such as honeypot deception, where the environment (the attacker and ICS protocol) evolves unpredictably.

**TABLE 1. Survey on honeypot for ICS.**

| Study | Backend | Simulated services | Application Field | Attack Types | Data Analyzed | RL Utilization | Dynamic Response |
|---|---|---|---|---|---|---|---|
| Petre et al. [12] | VM | Modbus | Water System | Unauthorized access | Database entries | - | - |
| ICSpot [13] | Hybrid | S7comm, PLC TCP/IP stack, etc. | | - | Number of interactions, unique IPs | - | - |
| Zamiri et al. [14] | | Veeder-Root ATG | Gas System | - | - | - | - |
| DiPot [6] | VM | HTTP, Modbus, Kamstrup, etc. | ICS | Modbus and Kamstrup scan, etc. | Access sequences, IPs | - | - |
| Conpot [5] | | Modbus, S7comm, HTTP, SNMP | | NMAP scan, Shodan | Access to Honeypot | - | - |
| Ours | VM | PLC, Modbus, etc. | ICS | ICS Kill Chain MITRE Campaigns | Network Traffic | ✓ | ✓ |

**TABLE 2. Survey on RL-based adaptive honeypots.**

| Study | Network | Category | Reinforcement Learning | | | | Evaluation | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | State | Action | Reward | Method | Conventional metrics | Attack Stage | MITRE ATT&CK |
| RASSH [18] | IT | SSH Honeypot | Command | ABDSI | Command Type | SARSA | ✓ | - | - |
| QRASSH [19] | | | | | | DQN | ✓ | - | - |
| Cannypot [20] | | | | ABSI | Command length | TD | ✓ | - | - |
| HoneyIoT [21] | | IoT Honeypot | Request | Response | Exploit Code | PPO | - | ✓ | - |
| IoTCandyJar [22] | | | | | | Q-Learning | - | ✓ | - |
| Ours | OT | ICS Honeypot | Tactic of Interaction | MITRE Engage | Tactic of Interaction | Q-Learning | ✓ | ✓ | ✓ |

For an ICS honeypot, the state may include the MITRE ATT&CK tactic index [7], protocol type (e.g., Modbus, S7comm), function code (read coil, write coil), and internal sensor or actuator states. Actions extend beyond simple 'allow' or 'block' to include register manipulations, false sensor data, or timed delays that mimic real ICS latencies. Rewards encourage deeper engagement by assigning positive values when attackers escalate tactics and penalties when they disconnect or detect the honeypot.

Conventional defenses such as signatures or supervised ML struggle with novel ICS exploits, especially when protocols are proprietary or underdocumented. More importantly, standard Deep Learning (DL) approaches are typically designed for offline inference from fixed datasets and require labeled targets (supervised learning) or proxy objectives (unsupervised learning). They do not directly optimize a long-horizon interaction objective such as "sustain engagement while steering the attacker through tactics," nor do they naturally model the exploration–exploitation trade-off that is fundamental when the system learns from live interactions. RL, by design, is the appropriate learning paradigm for this closed-loop setting because it learns directly from environment feedback and explicitly optimizes cumulative reward over interaction sequences. In other words, the limitation is not that DL cannot be used at all, but that DL alone does not address the policy-learning requirement of adaptive deception; RL is selected to provide this adaptivity.

### *E. RELATED WORKS ON ICS HONEYPOTS AND RL*

Research on ICS honeypots has focused mainly on static or only partially emulated architectures, with relatively few attempts to leverage RL for adaptive deception. In this section, we present two different survey tables: Table 1 summarizes representative ICS honeypots, and Table 2 discusses RL-based honeypots from IT or IoT domains. Many of these efforts highlight the importance of luring attackers away from real ICS assets, but do not fully address advanced multistage intrusions that require real-time dynamic responses.

In Table 1, the studies can be organized according to the main emulation strategy and the scope of the protocol (for example, Modbus, S7comm, HTTP). [12], for example, focused on Modbus data collection but lacked dynamic response, meaning real-time adaptation of replies based on attacker input, once attackers advanced beyond basic read/write queries. Conpot [5] became a well-known multiprotocol honeypot operating in a virtual machine (VM) environment, capable of detecting broad scans (for example, from Shodan or Nmap) but still returning static responses for deeper interactions. DiPot [6] recorded interactions similarly across multiple protocols (HTTP, Modbus, Kamstrup), while [14] concentrated on a Veeder-Root Automatic Tank Gauge in the gas sector. ICSpot [13] introduced a partial S7comm stack and PLC-like behavior for a water system, but mainly recorded connection information or minimal ICS command sequences.

Although these solutions mark an important step in attracting adversaries away from live ICS assets, their lack of real-time adaptation limits the depth of engagement they can sustain. Static honeypots, in particular, rely on a fixed set of scripted responses that do not adjust to attacker behavior. As soon as adversaries issue advanced ICS commands or attempt lateral movement across protocols, the repetitive and predictable replies expose the deception, making it clear

they are not interacting with a fully operational ICS. This limitation highlights the need for dynamic honeypots that can adapt responses in real time, sustaining longer sessions and capturing deeper adversarial behaviors.

Table 2 discusses RL-based honeypots developed for the IT and OT domains, comparing the protocols addressed, how states and actions are represented, and the nature of the reward mechanism. Although IT/IoT RL solutions demonstrate the potential for adaptive deception, especially in prolonging attacker sessions and gathering richer data, they largely revolve around simpler text-based protocols such as SSH or single-packet IoT exploits. RASSH [18] and QRASSH [19] used RL in an SSH environment, effectively determining whether to allow or block typed commands based on session behavior. HoneyIoT [21] and IoTCandyJar [22] also integrated RL logic to handle code exploits or request patterns in consumer IoT devices. Although these approaches validate the importance of continuously refining honeypot responses, they do not incorporate ICS-specific constraints, such as time-sensitive operations, function code complexity, or multistage sabotage attempts across ICS networks. The reward functions and action sets in these IT/IoT systems also remain relatively narrow, often limited to letting commands pass or terminating the interaction.

While these RL-based honeypots show significant progress in the IT and IoT domains, they are inherently designed for environments with general-purpose protocols such as SSH or HTTP. Tools such as Cowrie [24] and Kippo [25], for example, focus on terminal command emulation and basic command filtering [25]. However, ICS are based on time-critical operations, binary protocol structures, and multistage interactions involving components such as PLCs and HMIs. These domain differences pose significant limitations when applying IT honeypot designs to ICS contexts, such as the lack of support for the protocol, real-time response logic, and process awareness, which often makes them unconvincing to sophisticated attackers. This fundamental gap in protocol and interaction modeling is a key motivation for our work, where the honeypot is designed specifically around the characteristics and tactic progression of the ICS network using the MITRE ICS framework.

Bridging the gap between these two fields of research: static ICS honeypots lacking real-time emulation and RL-based honeypots confined to IT/IoT protocols motivates our work on an RL-driven ICS honeypot. Building on Table 1's insights, we extend ICS honeypot designs beyond static or partially emulated responses, incorporating robust ICS protocol simulation and the MITRE ICS Tactics to map each adversarial request to a known stage of an ICS kill chain. Meanwhile, inspired by the continuous adaptation shown in Table 2, our system employs RL methods to dynamically alter coil registers, introduce timing delays, or modify sensor outputs in real time, an approach rarely seen in ICS contexts. By combining ICS protocol fidelity with adaptive RL-based deception, we seek to preserve attacker engagement for multiple kill chain phases without prematurely betraying the honeypot's artificiality, thus gathering deeper intelligence on ICS-targeted threats.

## III. PROBLEM FORMULATION

This section describes the ICS honeypot problem in three parts: notation table, where key symbols and variables are defined, model description, outlining how attacks and honeypot interactions are represented, and problem statement, which formalizes the objective and constraints for selecting honeypot actions that maximize depth of attacker tactics and session length.

### A. NOTATION TABLE

To formulate the interactions of the ICS honeypot, we employ a consistent set of symbols, summarized in Table 3. These notations are grouped into three categories: (1) ICS Honeypot, where we define attacker interactions and honeypot responses, (2) MITRE ATT&CK, encompassing the ICS tactics that label each attacker action, and (3) Performance, which contains symbols that measure how deeply an attacker progresses.

Table 3 shows each symbol, its term, and any specialized meaning. In particular, $N_P$ represents the total number of attacker interactions in a session, and $P = (p_1, p_2, \ldots, p_{N_P})$ denotes the ordered interaction sequence (trace) of these interactions. The honeypot selects actions $a_i$, each drawn from the predefined action set $A = \{\alpha_j \mid j = 1, \ldots, |A|\}$. We also adopt $t_i$ to indicate the MITRE ICS tactic detected at each step, where $T = \{\tau_k \mid k = 1, \ldots, |T|\}$ is the finite set of tactics. Finally, two performance metrics, $T_L$ and $\overline{N_P}$, represent the last observed tactic (i.e., rightmost in the session) and the average number of interactions per session, respectively.

Each step $p_i$ refers to an attacker action or command (for example, a Modbus packet) that reaches the honeypot. Once the honeypot detects the ICS tactic $t_i$, it selects an action $a_i \in A$, where $A = \{\alpha_j \mid j = 1, \ldots, |A|\}$ is the predefined action set. In practice, this means that at each step $i = 1, \ldots, N_P$, the honeypot chooses one option $\alpha_j$ from $A$, and the selected option becomes the action $a_i$. The environment then returns an output $O_i$ (the honeypot's crafted or modified packet) and a reward $r_i$, reinforcing or penalizing the honeypot's choice.

Similarly, tactics are defined by the set $T = \{\tau_k \mid k = 1, \ldots, |T|\}$. For each interaction $p_i$, the honeypot maps the input to one observed tactic $t_i \in T$. Because an interaction may be compatible with multiple ATT&CK tactics, the mapper returns a single (primary) tactic label by leveraging session context (e.g., the previous tactic $t_{i-1}$ and protocol/function fields); if ties remain, we select the earliest plausible tactic (lowest-index). Across the session, the deepest observed tactic is denoted $T_L$, which serves as a crucial indicator of engagement depth: reaching a larger-value tactic (for example, $\tau_{10}$ vs. $\tau_6$) implies deeper infiltration into the ICS kill chain stages.

**TABLE 3. Notations.**

| Category | Symbol | Term |
|---|---|---|
| ICS Honeypot | $P = (p_1, p_2, \ldots, p_{N_P})$ | Ordered sequence (trace) of attacker interactions in a honeypot session; interactions may repeat, where $N_P$ is the number of attacker interactions (length of $P$), and $p_i$ is the $i$-th attacker interaction in temporal order. |
| | $F_i$ | Extracted feature vector from interaction $p_i$ |
| | $A = \{ \alpha_j \mid j = 1, \ldots, \lvert A \rvert \}$ | Set of predefined honeypot actions |
| | $a_i \in A$ | Action selected by the honeypot in response to the $i$-th interaction $p_i$ |
| | $s_i$ | The $i$-th honeypot state |
| | $r_i$ | The $i$-th honeypot reward |
| | $O_i$ | Honeypot response output to the $i$-th interaction $p_i$ |
| MITRE ATT&CK | $T = \{ \tau_k \mid k = 1, \ldots, \lvert T \rvert \}$ | Set of MITRE ATT&CK tactics for ICS |
| | $\tau_k$ | The $k$-th tactic in MITRE ATT&CK for ICS |
| | $t_i \in T$ | Detected tactic for the $i$-th interaction $p_i$ |
| | $T_L \in T$ | Last (rightmost) detected tactic in the honeypot session |
| Performance | $\overline{N_P}$ | Average number of interactions per session (over all sessions) |

### B. MODEL DESCRIPTION

An ICS honeypot session consists of attacker interactions $P = (p_1, p_2, \ldots, p_{N_P})$. Each interaction $p_i$ contains messages from the ICS protocol. The honeypot intercepts $p_i$ through a proxy, extracting relevant features such as function code, message payload, or session context, and identifying the attacker's current MITRE ICS tactic $t_i$. This tactic label becomes part of the honeypot state representation. After the tactic mapping step, the honeypot selects an action $a_i \in A = \{\alpha_j \mid j = 1, \ldots, |A|\}$. These actions may include 'Allow', 'Block', 'Delay', 'Modify', or 'Inject false data', simulating advanced ICS deception or disruptions. The honeypot then sends an output $O_i$ back to the attacker—for instance, a modified PLC register value—thus potentially steering the attacker deeper into the kill chain. The environment, shaped by ICS protocols and attacker behavior, yields a reward $r_i$, which may be positive if the attacker advances to a higher tactic or negative if the attacker disconnects or detects the honeypot.

We treat each interaction $p_i$ as a discrete time step in an RL problem. The state $s_i$ comprises (1) the current MITRE ICS tactic $t_i$, (2) the type of ICS protocol, and (3) the function code. The action $a_i$ is chosen from $A$. The reward $r_i$ typically encourages deeper kill chain progression or sustained engagement and penalizes early exit or honeypot detection. Formally, if $t_{i+1}$ surpasses $t_i$ (for example, from $\tau_5$ to $\tau_6$), the honeypot obtains a positive increase. In contrast, if the attacker terminates the session or stalls at the same tactic, the reward is reduced. Over repeated interactions, an RL agent learns a policy $\pi$ mapping each state $s_i$ to an action $a_i$. This policy aims to maximize the final tactic $T_L$ reached, as well as the average number of interactions $\overline{N_P}$.

Designing an RL-based ICS honeypot must address real-time and safety constraints. ICS commands require timely replies: excessive delay may reveal the honeypot's artificial nature or disrupt real processes (if integrated too closely with production ICS). The action set $A$ remains finite, with each action $a_i \in A$ tested for consistency with ICS logic to ensure safe behavior. Since ICS messages control physical processes, the honeypot should never issue responses that could endanger actual equipment or operators.

### C. PROBLEM STATEMENT

The goal of this study is to develop an RL-driven ICS honeypot that maximizes both the final tactic depth $T_L$ reached in the MITRE ATT&CK for ICS matrix and the average number of attacker interactions $\overline{N_P}$. The formal problem definition is as follows:

- Input

- Attacker interactions with the ICS protocol:$P = (p_1, p_2, \ldots, p_{N_P})$
- Predefined honeypot action set: $A = \{\alpha_j \mid j = 1, \ldots, |A|\}$ derived from MITRE Engage

- Output

- Sequence of honeypot actions $(a_1, a_2, \ldots, a_{N_P})$, with each $a_i \in A$
- Final observed tactic $T_L \in T$, indicating the deepest stage reached by the attacker

- Objective

- Maximize $T_L$ (depth of the ICS kill chain)
- Maximize $\overline{N_P}$ (average number of attacker interactions per session)

- Constraints

- The action space $A$ is finite
- Each selected action $a_i$ must belong to $A$

By framing ICS honeypot deception as an RL problem, the system can learn an optimal policy that prolongs attacker engagement and reveals advanced tactics. This contrasts with static honeypots, which often terminate sessions prematurely and yield limited intelligence. The RL formulation thus provides a learning setup under which the honeypot can learn, through training, a policy that yields deeper tactical progression and richer threat insights to support industrial defense strategies.

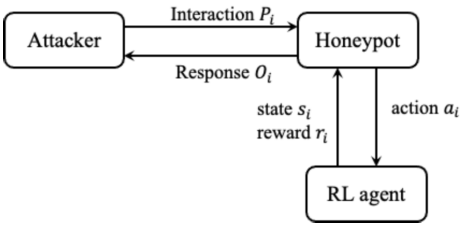


**Fig. 2.** **Selection of best honeypot action.**

## IV. RL-BASED ADAPTIVE HONEYPOT

This section presents an RL-driven solution architecture for ICS honeypots, aiming to deepen attacker engagement by integrating ICS protocols into an RL framework.

### A. OVERVIEW

The central idea of the solution is to formalize the honeypot as an RL system, where States, Actions, and Rewards are defined using ICS-specific semantics. Both MITRE ATT&CK for ICS and MITRE Engage serve as guiding frameworks for tactic-based deception. Figure 3 illustrates the architectural components and the sequential flow of information, denoted by the numbers (1) through (7), which represent the ordered steps from raw traffic capture to RL-based decision-making and final response execution.

In this architecture, each attacker interaction $p_i$ first passes through a Proxy, which extracts protocol-level *features* such as function code, register address, or message type. These features represent the essential fields of the ICS protocol that allow the honeypot to interpret and react credibly. The Mapper then maps $p_i$ to a corresponding tactic $t_i \in T$ (where $T = \{\tau_k\}$ is the set of MITRE ATT&CK tactics), which becomes part of the RL state $s_i = (t_i, \text{protocol}, \text{function})$.

Based on this state, the RL agent selects an action $a_i \in A = \{\alpha_j\}$. Actions range from simply allowing a request to delaying, blocking, or injecting false sensor data, all aligned with MITRE Engage's deception strategies. The selected action is passed to the ICS Emulator, which generates the corresponding output $O_i$ (for example, a modified coil value or fabricated sensor reading). The environment then provides a reward $r_i$, which is positive if the attacker progresses to a deeper tactic and negative if the attacker disconnects or detects the deception.

It is important to highlight that the reward design in this architecture reflects the operational objective of a honeypot, rather than that of a production inline defense. Unlike IDS or firewall systems, which aim to block attacks as early as possible, the primary goal of the proposed honeypot is deception and intelligence gathering. Consequently, the reward $r_i$ is defined to encourage sustained engagement and tactical progression within the isolated decoy environment, as deeper stages of the MITRE ATT&CK for ICS kill chain often reveal richer attacker behaviors and TTPs. In this context, the term "depth" does not suggest inflicting damage to actual assets. Instead, it serves as an indicator of how effectively the honeypot can sustain interactions and extract valuable intelligence within a controlled environment. The protection of the real ICS site and operational processes remains the responsibility of external security controls deployed around production assets (e.g., network segmentation, firewalls, and IDS), while the RL-driven honeypot operates as an isolated decoy, focusing on maximizing intelligence yield through tactic-aware, stage-dependent deception.

### B. PROXY

The Proxy module is responsible for intercepting ICS traffic and extracting relevant features. This step is crucial in ICS contexts, where messages are binary and tightly coupled with real-time commands (for example, Modbus function codes).

The Proxy analyzes incoming ICS packets to retrieve key metadata (source/destination IP, packet count, timestamps) and protocol fields such as Modbus function code [4]. These details are then compiled into a structured feature set $F_i$ for interaction $p_i$. Unlike IT honeypots that only manage text-based requests, ICS interactions must distinguish normal read queries from suspicious writes or block operations. Parsing function codes is essential for tactic detection and shaping RL decisions, since an attacker attempting advanced ICS commands requires a different response than basic scanning. Proxy filters traffic on relevant ICS ports (for example, 502 for Modbus). The extracted fields (protocol type, function code, session metrics) are passed to the next module with low processing overhead, ensuring that the honeypot can respond with realistic timing without revealing timing artifacts that might raise attacker suspicion.

### C. MAPPER

After the proxy completes its analysis, the Mapper determines the kill chain stage (tactic) of each attacker interaction by referencing the MITRE ATT&CK framework for ICS. Using the extracted feature vector $F_i$, the mapper translates protocol-specific details into a tactic label. Formally, each interaction $p_i$ is mapped to an observed tactic $t_i \in T = \{\tau_k \mid k = 1, \ldots, |T|\}$. When multiple tactics are plausible for the same interaction, the mapper uses session context (including the previous predicted tactic $t_{i-1}$ and protocol/function metadata) to select a single primary tactic for the RL state. If ambiguity persists, we select the earliest plausible tactic (lowest-index) and optionally record alternative candidates for post-analysis.

The mapper can be implemented as a large language model (LLM) prompt that infers the ICS tactic from the combination of features $F_i$, register usage, and session metrics. This allows more flexibility for novel attacks [26]. A simpler rule-based approach relies on if-else logic keyed to known function codes or indicators of compromise (IoCs) triggers. Although reliable for known patterns, this method lacks adaptability to unknown threats.

We emphasize that the LLM is not used to provide a human-facing interface. Instead, it serves as a semantic

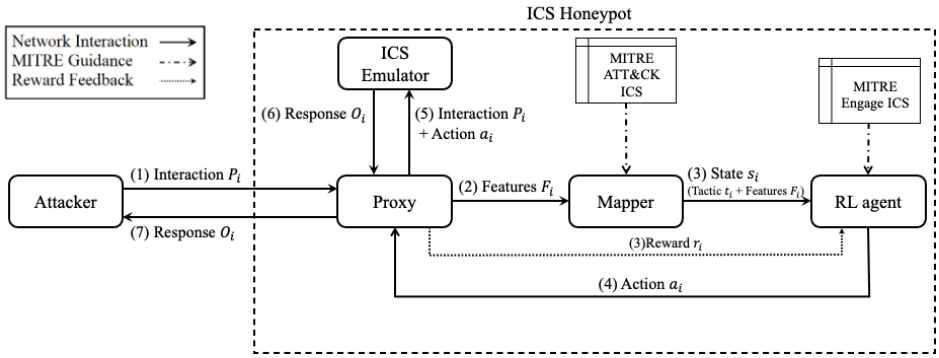


**Fig. 3. Solution overview. Note: The numbers in parentheses (1)–(7) indicate the chronological order of operations in a single interaction cycle.**

mapper that converts structured, low-level ICS features into a discrete tactic label $t_i$ used by the RL agent as part of its state. Although ICS traffic is binary, we encode the extracted fields into a short textual prompt purely as a machine-to-model input format required by current LLMs, enabling the mapper to leverage the MITRE ICS tactic definitions and protocol semantics.

Research conducted by [27] demonstrates that LLMs, such as ChatGPT, can map low-level security inputs to high-level MITRE ATT&CK techniques. Although their study focused on enterprise NIDS rules, it illustrates the broader capability of language models to reason over protocol semantics and attack behaviors. In our design, we extend this approach by enriching the prompt context with the full MITRE ATT&CK for ICS matrix and tactic definitions, customized specifically for industrial protocols.

Furthermore, we evaluated the mapper using a balanced validation set of 120 synthetic ICS interactions designed to cover all twelve ICS tactics and diverse combinations of protocol semantics (e.g., function codes, command payload patterns, arguments, and session metrics), and observed accurate stage classification in more than 95% of cases. In this context, each case represents one invocation of the interaction-level mapper derived from proxy-extracted features. A mapping is considered correct if the mapper outputs the intended single tactic label $t_i \in T$. This reinforces the mapper's effectiveness as a semantic state extractor for the RL agent, enabling real-time, tactic-aware deception in ICS environments. In practice, since the mapper outputs only a single tactic index, the inference latency is on the order of tens of milliseconds, which is reasonable for online honeypot operation and does not become a bottleneck in our pipeline.

While the dataset is synthetic, it enables controlled ground-truth labeling and full tactic coverage, which is difficult to obtain from existing ICS datasets that lack interaction-level MITRE annotations. The purpose of this evaluation is to verify that the mapper can reliably translate low-level ICS protocol features into the correct tactical category.

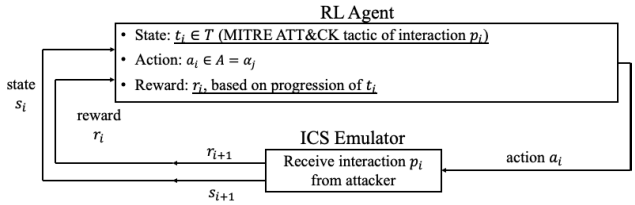


**Fig. 4. RL agent.**

To mitigate hallucination and ensure reliable outputs, we adopt constrained prompt engineering. The prompt provides the full MITRE ATT&CK for ICS tactic set and definitions as strict context, and the output is restricted to selecting a single tactic index ($t_i \in T$) rather than generating free-form text. Moreover, the LLM does not operate in a vacuum; it is grounded by structured proxy features ($F_i$), including protocol type, function codes, command payload, arguments, and session metrics. By tying the inference to observable network telemetry and restricting the output space to a finite set of valid tactics, the risk of producing non-existent tactics is substantially reduced.

By placing $t_i$ in the RL state, the agent gains visibility into the attacker's position in the kill chain and can adapt its strategy, employing subtle deflection in early stages or more disruptive responses as the adversary escalates. This alignment between semantics of the protocol layer and adversarial progression strengthens the honeypot's realism and intelligence-gathering potential.

### 1) LOAD-HANDLING SCOPE UNDER CONCURRENT TRAFFIC

Beyond the reliability of the inferred tactic label, the Mapper is also designed with operational robustness in mind. In particular, computationally expensive mapping/inference is invoked only for semantically meaningful and protocol-compliant ICS interactions, rather than for arbitrary high-rate traffic at the network layer. The Proxy module performs protocol parsing and validation and extracts only minimal ICS semantics (e.g., protocol type, Modbus function codes,

**TABLE 4.** MITRE engage driven action space.

| Action | Engage Activity | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Baseline | Isolation | Network Manipulation | Hardware Manipulation | Security Controls | Lures | Attack Vector Migration | Introduced Vulnerabilities |
| No action / Allow | - | - | - | - | - | - | - | - |
| Restore to Original State | ✓ | - | - | - | - | - | - | - |
| Delay Response | - | - | ✓ | - | - | - | - | - |
| Modify Response | - | - | ✓ | ✓ | - | - | - | ✓ |
| Block Traffic | - | ✓ | ✓ | - | ✓ | - | - | - |
| Simulate System Error | - | - | - | ✓ | - | ✓ | ✓ | - |
| Introduce System Jitter | - | - | ✓ | - | - | ✓ | - | - |

and relevant fields) before forwarding a request to the Mapper. As a result, high-frequency non-protocol-compliant traffic (e.g., malformed packet floods or raw SYN floods) can be handled or dropped by the underlying OS/network stack and Proxy layer without triggering the Mapper/LLM pipeline. This design ensures that LLM resources are reserved for validated ICS interactions that contribute to deception and threat intelligence, while reducing the risk that volumetric noise traffic can translate into unbounded inference load.

### D. RL AGENT

Figure 4 illustrates the structure of the RL agent in our honeypot framework. The agent receives the current state $s_i = (t_i,\ \text{protocol},\ \text{function})$, where $t_i \in T$ is the MITRE ICS tactic mapped by the mapper, and the protocol/function fields capture low-level details (e.g., Modbus vs. S7comm; read, write, or diagnostic operations). Based on this state, the agent selects an action $a_i \in A = \{\alpha_j \mid j = 1, \ldots, |A|\}$, aligned with deception strategies from the MITRE Engage framework [17]. These responses emulate realistic ICS behavior—for example, forging sensor readings, injecting timing jitter, or restoring register states. Table 4 summarizes the full list of supported actions.

We acknowledge that mapping each interaction to one of the 12 MITRE ATT&CK for ICS tactics is a deliberate semantic abstraction that may compress some low-level packet detail. This choice is motivated by tractability and interpretability for online honeypot policy learning, where the tactic label provides a phase-aware representation of attacker progression and enables stage-dependent deception without state explosion. Importantly, our framework is not restricted to a "12-state machine." In the *small-state* setting, we use $s_i = t_i$ to capture only the tactical stage for maximal compactness. In the *big-state* setting, we augment the state with protocol semantics, i.e., $s_i = (t_i,\ \text{protocol},\ \text{function})$, so that the agent can condition actions not only on the tactical stage but also on key ICS operation context (e.g., Modbus vs. S7comm; read/write/diagnostic function codes). Moreover, the interaction dynamics are not a fixed deterministic state machine: an attacker can repeat actions, remain within a tactic, branch to different behaviors, or terminate the session, and the honeypot's chosen action affects subsequent observations and rewards. We therefore position this formulation as a practical RL abstraction for adaptive deception rather than a full game-theoretic model of attacker intent.

The agent then executes the selected action and observes the result of the interaction. A reward $r_i$ is calculated based on progression through the kill chain: advancing to a deeper tactic yields a positive bonus, while disconnection or early detection incurs a penalty. This reward shaping encourages the agent to prolong the engagement and lure attackers further into the ICS lifecycle. Over time, this mechanism allows the honeypot to autonomously learn a deception policy that maximizes the depth and duration of the session, producing more valuable telemetry for defenders.

We adopt tabular Q-learning, as shown in Algorithm 1, to implement the honeypot's adaptive behavior. This method is chosen because the state–action space is relatively small, enabling rapid updates without the need for deep learning, experience replay, or GPU acceleration [28]. In our setting, the RL state is intentionally discrete and compact (tactic-driven, with optional protocol/function context in the big-state configuration), and the action set is also small and predefined; thus, a tabular representation is sufficient and avoids approximation error introduced by function approximators. Compared to a Deep Q-Network (DQN), tabular Q-learning provides more stable and reproducible learning behavior under this compact state space, and it eliminates additional design choices and tuning requirements (e.g., network architecture, replay buffer, target-network updates, and sensitivity to random seeds) that could confound controlled evaluation. Moreover, tabular learning keeps both training and inference lightweight, reducing operational complexity and minimizing timing artifacts that could arise from neural inference in an online honeypot pipeline.

Each update modifies only a single Q-table entry, preserving interpretability, which we define as the transparency of the decision-making logic—i.e., a human defender can inspect why a particular deception action is preferred under a given tactical state. Specifically:

**Algorithm 1** Q-Learning for the ICS Honeypot

**Require:** $T, A, \gamma, \alpha, \epsilon_0, \epsilon_{\min}, \rho, \tau_{\mathrm{T}}, E$
**Ensure:** Learned action-value table $Q \in \mathbb{R}^{|T|\times|A|}$.

1: Initialize $Q \leftarrow \mathbf{0}_{|T|\times|A|}$;
2: **for** episode = 1 to $E$ **do**
3: $\epsilon \leftarrow \epsilon_0$; $t \leftarrow \tau_1$; {start at initial tactic}
4: **while** $t \neq \tau_{\mathrm{T}} \ \wedge \ SESSIONACTIVE()$ **do**
5: Receive packet $p_i$ and extract features $F_i$;
6: $t \leftarrow MAPPER(F_i)$; {current state key $t \in T$
7: **/* $\epsilon$-greedy action selection */**
8: With probability $\epsilon$: sample $a \sim \mathrm{Uniform}(A)$;
9: Otherwise: $a \leftarrow \arg\max_{a' \in A} Q[t, a']$;
10: Proxy executes $a$;
11: **/* handle disconnect immediately */**
12: **if** $\neg SESSIONACTIVE()$ **then**
13: $r \leftarrow -5$; {early termination / detection penalty}
14: **break**
15: **end if**
16: Observe $p_{i+1}$, extract $F_{i+1}$, and set $t' \leftarrow MAPPER(F_{i+1})$;
17: **/* reward shaping */**
18: **if** $t' = \tau_{mathrmT}$ **then**
19: $r \leftarrow |T| + 1$; {terminal bonus: full kill-chain coverage}
20: **else if** $t' > t$ **then**
21: $r \leftarrow \mathrm{index}(t') + 1$; {progress bonus (no cap needed)}
22: **else**
23: $r \leftarrow 1$; {small reward to sustain engagement}
24: **end if**
25: **/* Q-update (see lines 21-22) */**
26: $y \leftarrow r + \gamma \mathbb{1}[t' \neq \tau_{\mathrm{T}}] \ \max_{a' \in A} Q[t', a']$; {TD target
27: $Q[t, a] \leftarrow Q[t, a] + \alpha\big(y - Q[t, a]\big)$;
28: $\epsilon \leftarrow \max(\epsilon \cdot \rho, \ \epsilon_{\min})$;
29: $t \leftarrow t'$;
30: **end while**
31: **end for**
32: **return** $Q$;=0

- **Auditability:** Each $Q(s, a)$ entry explicitly represents the expected long-term utility of action $a$ under state $s$, enabling direct inspection of action pReferences.
- **Traceable updates:** Each interaction updates only one cell $Q(s, a)$, so changes to the policy are localized and easy to track over time.
- **Human-in-the-loop validation:** Defenders can review and, if needed, constrain or override specific state–action preferences to ensure alignment with operational requirements.

In the current design, the RL state is intentionally lightweight and discrete (tactic-driven, and optionally augmented with protocol/function context) to support tabular Q-learning with low overhead and high auditability. Although we do not explicitly encode fine-grained timing histories, multi-step behaviors are partially captured at a higher semantic level through the MITRE ATT&CK for ICS tactic sequence $\{t_i\}$, which abstracts the attacker's progression across multiple steps. Consequently, we do not explicitly model richer temporal patterns (e.g., inter-packet/inter-action timing distributions), multi-step sequence history beyond the current session step, or cross-session attacker strategies in the state.

At the beginning of training, the Q-table is initialized to zero. Each episode simulates a complete interaction session between the attacker and the honeypot, starting with the initial tactic $t = \tau_1$. At every step, the honeypot intercepts the incoming packet $p_i$, extracts the feature vector $F_i$ through the proxy layer, and passes it to the mapper to determine the current tactic state $t \in T$.

The agent then selects an action using an $\epsilon$-greedy policy: with probability $\epsilon$, it explores by choosing a random action $a \in A$; otherwise, it exploits its current knowledge by selecting the action with the highest Q-value for the current state. The chosen action is executed, modifying the honeypot's next response accordingly. After applying the action, the system observes the resulting packet $p_{i+1}$ and uses the mapper again to determine the next tactic $t'$.

Rewards are calculated based on session progress. If the attacker disconnects, a penalty of $-5$ is applied. If the attacker reaches the final absorbing tactic $\tau_{\mathrm{T}} \in T$ (e.g., the "Impact" stage), a large positive terminal reward of $|T|+1$ is given, reflecting maximum engagement depth. If the attacker progresses to a higher tactic without yet reaching $\tau_{\mathrm{T}}$, a scaled positive reward of $\mathrm{index}(t') + 1$ is assigned. If the tactic does not advance, a small reward of 1 is given to sustain the session.

The Q-value update follows the standard Bellman formulation, as shown in lines 21–22 of Algorithm 1. The target $y$ is computed as the sum of the immediate reward and the discounted maximum Q-value in the next state (unless the terminal state is reached), and the corresponding Q-table entry $Q[t, a]$ is updated accordingly. After each step, the exploration rate $\epsilon$ is decayed by multiplying it with the decay factor $\rho$, ensuring a gradual shift from exploration to exploitation. The agent then proceeds with the next tactic $t'$ and repeats the loop until the session ends.

### 1) TRAINING VS. DEPLOYMENT

Algorithm 1 describes the learning phase, where the agent updates the Q-table online after each interaction and uses an $\epsilon$-greedy policy ($\epsilon$ initialized to $\epsilon_0$ and decayed by $\rho$) to balance exploration and exploitation. After training converges (or after a fixed number of episodes), the learned Q-table can be exported and reused. In the deployment/use phase, the honeypot loads the learned Q-table and selects actions primarily by exploitation (e.g., $\epsilon \approx 0$), while Q-updates can be disabled to ensure stable, predictable behavior unless online adaptation is explicitly enabled.

**TABLE 5. Open-source tools and libraries.**

| Category | Name | Description & Functionality |
|---|---|---|
| Tools | Conpot [29] | Static Honeypot with emulation of protocols and PLC |
| | MITRE Campaigns [30] | Detailed attacks on ICS with mapped TTPs |
| | Malcolm and Wireshark [31] [32] | Network traffic analyzer for ICS specific protocols |
| | ChatGPT-4o-mini API [38] | Tactic mapping |
| | Metasploit ICS modules [33] | Launching of attack |
| | Kali Linux ICS [34] | |
| Python Libraries | PyTorch [36] | Implementing of RL algorithm |
| | Scapy [35] | ICS protocols modification |
| | Matplotlib [37] | Data Virtualization |

During training, the agent follows an $\epsilon$-greedy policy, which forces exploration of non-greedy actions even when the attacker's behavior is predictable. This built-in randomness reduces the risk of converging to a rigid, script-specific action sequence and encourages learning action preferences that generalize across similar tactical contexts.

### E. ICS EMULATOR

The final module is the honeypot emulator, which executes the RL agent's chosen action and returns a corresponding ICS response. Built on systems like Conpot, it simulates realistic PLC registers and process operations while logging interaction data.

The emulator translates actions (e.g., `delay 2s`, `modify coil #3`, `block write request`) into protocol-compliant responses. For instance, a Modify action may insert a plausible sensor reading or adjust a register value without introducing inconsistencies such as invalid packet lengths or checksums. Because ICS protocols operate under strict timing, each response is validated to avoid delays or anomalies that could expose the honeypot.

After applying action $a_i$, the emulator updates its internal state and replies to the attacker. The subsequent attacker reaction—whether escalation, suspicion, or new commands—completes the feedback loop, informing future policy updates.

This emulator, integrated with the Proxy, Mapper, and RL Agent, enables real-time tactic-aware deception that prolongs adversary engagement and yields richer ICS-specific threat intelligence.

## V. IMPLEMENTATION

This section describes how we built and deployed our RL-driven ICS honeypot on top of Conpot, integrating packet parsing, tactic mapping, and the Q-learning agent. We first review the open source tools and libraries used (subsection V-A), then detail the customizations made to Conpot and the embedding of our RL module (subsection V-B), and finally present the testbed configuration and deployment steps (subsection V-C).

### A. OPEN SOURCE TOOLS AND LIBRARIES

Table 5 summarizes the open source components used in the system. The foundation is Conpot, an ICS honeypot that emulates Modbus, S7comm, and other protocols [29]. It serves as the base simulation platform, preserving realistic register-level behaviors while incorporating hooks for RL-driven deception.

Attacker behavior is reproduced using MITRE Campaign Scenarios, scripted sequences of ICS commands derived from real incidents such as the 2015 Ukraine power grid attack [30]. These provide structured, repeatable attacker flows. For monitoring and validation, Malcolm, an ICS-aware analysis suite, and Wireshark are employed [31], [32]. Malcolm's Modbus parsers cross-check the RL proxy's feature extraction. Attack sequences are launched through Metasploit ICS modules and Kali Linux tools, creating an automated environment for known ICS exploits [33], [34]. We adopt these MITRE-aligned scripted campaigns because they provide a structured and reproducible ground truth for evaluating how an RL-driven honeypot handles a multi-stage ICS kill chain, enabling controlled comparison across configurations and reliable measurement of engagement depth and interaction length.

The implementation is orchestrated in Python, supported by Scapy for packet parsing, PyTorch for Q-table updates, and NumPy/Matplotlib for data processing and visualization [35], [36] [37].

Furthermore, we also reviewed public ICS datasets (e.g., SWaT and WADI). While these datasets provide valuable real-world traffic, they typically do not include the tactical labels required by our setting (e.g., mapping interaction steps to the MITRE ICS tactics) and are commonly provided as passive traces rather than closed-loop attacker–honeypot interactions. This limits their direct applicability for training and benchmarking a tactic-driven RL agent, as well as for computing our key metrics (e.g., tactic depth $T_L$ and session length $N_P$) under controlled conditions.

### B. IMPLEMENTATION DETAILS

We extend Conpot's network handlers by inserting a proxy layer that activates immediately after packet reception. This proxy intercepts each Modbus request and extracts relevant features, such as function code, register address, and timestamps, using Scapy, before logging them into the RL pipeline. More generally, the proxy extracts a set of ICS interaction features: (i) *protocol* (e.g., Modbus, S7comm), (ii) *command payload*, (iii) *function code*, (iv) *arguments*, and (v) *session metrics*. These features are selected to capture the requested operation semantics (function code and arguments), the concrete on-wire representation (payload), and the interaction context (session metrics), while keeping inline extraction overhead low for real-time honeypot operation. Conpot's built-in response logic is preserved as a baseline; when the RL agent issues a 'Modify' or 'Delay'

action, we override the next response payload or timing accordingly.

Furthermore, we also intentionally used scripted attacks to ensure reproducible and controlled benchmarking across honeypot configurations. In ICS settings, "constraints" primarily refer to *semantic validity* at the protocol/device level: requests and responses must respect protocol semantics, device memory layouts, and boundary functions so that the interaction remains plausible and does not immediately fail. However, these constraints do not imply a single fixed or non-randomizable attacker trajectory. Within valid ICS semantics, attacks can still exhibit substantial variability and uncertainty, including (i) branching choices among multiple valid commands, (ii) different parameterizations and target registers, (iii) timing/ordering variations that remain protocol-consistent, (iv) tool/operator differences, and (v) session-level behaviors such as retries, backtracking, and switching tactics after observing responses.

At the same time, these constraints do rule out *naive* script perturbations that would break ICS semantics (e.g., arbitrary shuffling that violates boundary-function preconditions or device memory conventions). For this reason, we did not apply arbitrary randomization; instead, we first ensured that the honeypot could credibly support the required ICS boundary functions, and treat semantics-preserving workflow perturbations (e.g., safe command/parameter/timing variations) as a principled direction for robustness evaluation.

This context also clarifies the role of RL in our design. Even when each interaction must remain protocol-consistent, the defender still optimizes a *long-horizon* objective with delayed outcomes—sustaining engagement and inducing deeper tactic progression while preserving plausibility—and the best deception response is not always obvious a priori. Moreover, the interaction is inherently policy-dependent: deception actions can influence subsequent attacker progression and session continuation, so a fixed open-loop sequence is not assumed. While a rule-based system or finite state machine can be crafted for specific workflows, it typically requires substantial manual specification and tuning across diverse semantic contexts (e.g., protocol/function/register combinations) and can become brittle under strategy variations. In contrast, Q-learning learns the state–action mapping from interaction feedback; after training, deployment-time inference is a lightweight table lookup, retaining interpretability via explicit state–action values without imposing heavy runtime overhead. To isolate adaptive decision-making gains from differences in protocol coverage and baseline fidelity, we also include static honeypot variants as controlled baselines.

Accordingly, we define two static honeypot variants to (i) establish a clear reproducible baseline and (ii) ensure sufficient ICS protocol coverage for executing the intended multi-stage kill chains. First variant, which we called Static0, refers to the default Conpot distribution as downloaded from the public repository, with no modifications. Static1, on the other hand, incorporates targeted enhancements to expand the coverage of boundary functions required for our industrial attack scenarios. In particular, we extend Conpot's Modbus handler to support uncommon but valid function codes used in lateral movement and sabotage operations, add custom register mappings to emulate more realistic sensor and actuator behaviors, and enable extended address ranges to reflect real-world PLC memory layouts. These modifications ensure that the honeypot can convincingly respond to multi-stage attacks as defined in our ICS kill chains, something that unmodified static honeypots often fail to do due to limited protocol support or hardcoded responses.

To maintain protocol realism, we retain Conpot's default register values and response formatting. Dynamic actions, such as modifying a sensor reading, are applied directly to the in-memory register table immediately before serialization. Meanwhile, Scapy ensures that the TCP/UDP framing remains correct. We also measure round-trip times to enable the injection of sub-second delays without exceeding typical Modbus timing thresholds.

While our evaluation metrics focus on engagement depth and interaction length, our implementation includes concrete mechanisms intended to reduce obvious honeypot inconsistencies that could raise attacker suspicion. Specifically, we preserve Conpot's native response logic as a baseline, and the RL agent only applies bounded interventions (e.g., selective payload modification or delay) rather than synthesizing unconstrained responses from scratch. Additionally, modifications are constrained to protocol-valid structures, and timing actions are bounded to prevent unrealistic stalls. Lastly, the proxy supports response-time shaping (bounded delay with small jitter), which helps avoid timing artifacts in either direction (artificially slow or unrealistically fast responses) that could raise attacker suspicion.

The mapper module processes feature tuples from the proxy and applies an LLM-based prompt to assign a MITRE ATT&CK ICS tactic index $\tau_i$. Combined with the protocol type and function code, this forms the RL state vector for each packet.

The Q-learning agent then executes the decision cycle. It reads the current state ($\tau_i$, protocol, funcCode), selects an action $a_i$ using an $\epsilon$-greedy policy on the Q-table, and passes the action to Conpot for execution. After observing the next tactic $\tau_{i+1}$ and the associated reward $r_i$, the Q-value is updated according to the Bellman formulation shown in lines 21–22 of 1. The exploration rate $\epsilon$ is subsequently decayed using the rule defined in the same algorithm.

The overall architecture proceeds as illustrated in Figure 5: packets are captured by the proxy, mapped to tactics, processed by the RL agent for action selection, and finally reintegrated into Conpot's ICS emulation stack before returning through the proxy. This modular pipeline

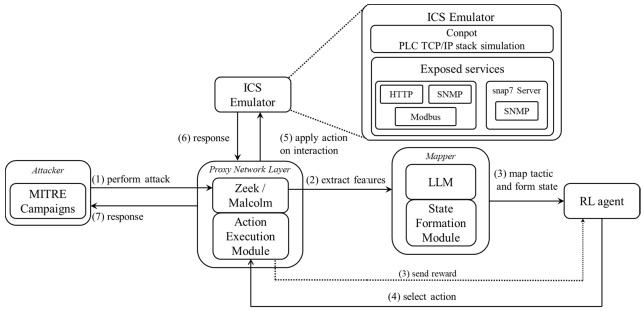


**Fig. 5.** System architecture.

clearly separates packet interception, tactic detection, decision logic, and protocol emulation.

All parameters, including port numbers, supported protocol types, RL hyperparameters ($\alpha$, $\gamma$, $\epsilon_0$, $\rho$), and action definitions, are specified in a YAML configuration file. This enables flexible experimentation and deployment via both Docker and native Python environments.

#### 1) DEPLOYMENT CONSIDERATIONS

In addition to modularity and configurability, we emphasize that the system is designed for deception and intelligence gathering rather than production-grade service availability. In practice, the honeypot can be deployed in multiple placements depending on the defender's objective. In one common pattern, it is intentionally exposed as an edge/DMZ decoy to attract probing and absorb adversarial effort; under this setting, if an attacker chooses to overwhelm the decoy via packet flooding, the honeypot has still achieved a defensive objective by diverting attacker time and resources away from critical production assets. In another pattern, the honeypot is placed behind perimeter or segmentation controls (e.g., firewall/IDS protecting the production network), where volumetric floods and non-protocol-compliant traffic can be mitigated upstream before reaching the honeypot.

### C. TESTBED SETUP

Our RL-driven ICS honeypot system was deployed within a virtualized testbed running on a host machine equipped with an Intel Core i7-10700 CPU at 2.90GHz and 16 GB of RAM. All experiments were conducted without GPU acceleration to emphasize the lightweight nature of the proposed approach. The host OS ran VirtualBox, within which two main virtual machines were configured: the honeypot VM (Ubuntu 22.04) and the attacker VM (Kali Linux 2024), connected via a bridged network interface.

We intentionally adopt this two-VM setup to ensure repeatability and to isolate the effect of the learned RL policy under consistent network conditions. This controlled baseline enables fair, configuration-to-configuration comparison of engagement depth ($T_L$) and interaction length ($N_P$) without confounding factors introduced by background traffic or variable network impairments.

The honeypot setup began by cloning the official Conpot repository and installing the necessary dependencies. To enable dynamic response logic, the Modbus server was patched to route incoming requests through a custom proxy, which extracted packet features and forwarded them to a mapper. The mapper used a backend-configured LLM to assign the corresponding MITRE ATT&CK ICS tactic. The RL agent, running as a systemd service or Docker entry point, performed real-time Q-table updates on live traffic. On the attacker side, Metasploit scripts were executed from the Kali VM to simulate ICS threat behaviors based on MITRE Campaign scenarios. For monitoring and analysis, session traces, Q-table snapshots, and final tactic distributions were recorded, while metrics such as session depth, convergence rate, and tactic coverage were visualized using Python and Matplotlib.

## VI. EVALUATION AND RESULTS

This section presents the evaluation results and assesses how well the RL-driven honeypot addresses the research questions. It first introduces the testbed, honeypot variants, and tracked metrics. Next, it examines whether a tactic-aware Q-learning policy draws attackers deeper than fixed-logic decoys by analyzing depth curves, session length, reward signals, and the learned Q-table. The following part compares a compact state definition with a richer protocol-level one to evaluate the practical benefits of added detail. Finally, it explores parameter sensitivity by contrasting `stable` and `fast` presets, and concludes with key findings and their implications for deployment.

### A. EXPERIMENTAL SETUP

We evaluated the honeypot in a controlled two-VM environment to ensure that any delays stemmed from our code rather than the network. The first VM (Ubuntu 22.04) ran Conpot with three variants: Static 0 (unmodified Conpot), Static 1 (Conpot with simple boundary checks), and Dynamic (with RL). The second VM (Kali Linux) launched scripted attacks using Metasploit ICS modules and Python scripts covering phases such as network discovery, unauthorized writes, and firmware manipulations modeled on the 2015 Ukraine power attack. Both VMs shared a bridged LAN with sub-millisecond latency, ensuring that any added delays reflected honeypot processing rather than packet transit.

We use scripted workflows primarily for reproducibility and controlled comparison across honeypot variants. Importantly, the RL agent does not observe any script identifier or finite-state-machine step; it learns a policy over the defender-side state representation (tactic-driven state and optional protocol/function context), which biases learning toward stage-appropriate deception rather than memorizing a fixed script.

Our dynamic honeypot uses a tabular Q-learner table, a $\epsilon$-greedy policy, and the usual Bellman update as we defined in Section V-B. We start with a small learning rate $\alpha$, moderate discount $\gamma$, and conservative $\epsilon$-decay $\rho$; the

exact defaults are listed as follows: $\alpha = 1 \times 10^{-4}$, $\gamma = 0.90$, $\epsilon_0 = 0.8 \rightarrow \epsilon_{min} = 0.01$ with $\rho = 0.95$. Throughout every session, we recorded five complementary metrics: (1) the maximum MITRE ICS tactic the attacker reached, (2) the duration of the session, (3) the total RL reward earned, (4) the distribution of final tactics over three 400-session learning windows, and (5) a snapshot of the trained Q-table. Collectively, these metrics provide a comprehensive view of the effectiveness of the honeypot: quantifying the depth and duration of attacker engagement, tracking the progression of policy learning over time, and identifying the action preferences that emerge across various states.

#### 1) EVALUATION SCOPE AND THREAT MODEL

Our experiments rely on scripted attack workflows to ensure reproducibility and controlled comparison across baselines. Accordingly, the reported improvements empirically validate adaptivity and deeper tactic progression within the evaluated threat setting, but they do not by themselves quantify performance against a fully adaptive adversary or a completely unseen attacker workflow during deployment. This scope is consistent with our objective of establishing a repeatable benchmark for learning-based deception control in an ICS honeypot environment.

Importantly, our objective and success criteria differ from intrusion detection: deception does not require rigid "detect/not-detect" decisions, and occasional suboptimal actions typically lead to *graceful degradation* (e.g., reduced engagement efficiency) rather than catastrophic failure, as long as the interaction remains plausible and operational. From a policy-design perspective, the learned decision rule is conditioned on semantic defender-observable states (tactic stage with protocol/function context), not on temporal script identifiers. Consequently, perturbations such as reordering attacker commands correspond to different trajectories over the same semantic state space, and the policy remains applicable whenever the resulting interactions induce states covered by the learned abstraction.

### B. ATTACK-DEPTH: STATIC VS. RL-DRIVEN HONEYPOT

To evaluate the effectiveness of our dynamic honeypot compared to static baselines, we conducted 2000 consecutive attack sessions. For each session, we recorded the highest MITRE ICS tactic reached by the attacker. These results are shown in Figure 6, where the x-axis represents the episode index (from 1 to 2000) and the y-axis shows the maximum tactic index (ranging from 1 to 12) reached during that session. The figure compares four configurations: a static default honeypot (Static 0), a static hand-tuned variant (Static 1), and two RL-driven adaptive honeypots. The first RL configuration, denoted *small-state*, uses only the current tactic index as its state representation. The second, denoted *big-state*, augments this with additional protocol-level context such as ICS protocol type and function code.

In contrast to the static honeypots, which level off early at tactic 4 (Static 0) and tactic 7 (Static 1), the RL-driven

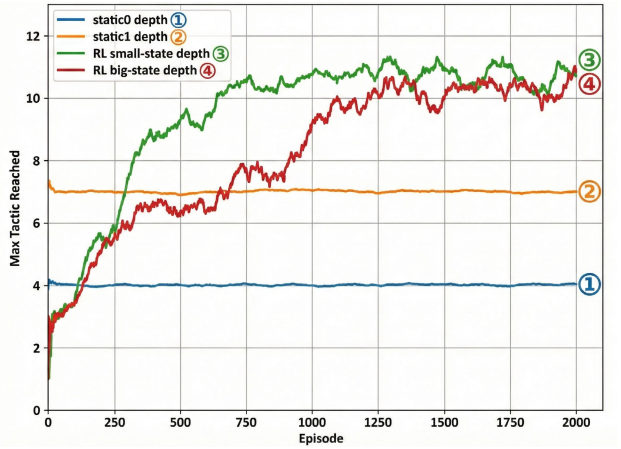


**Fig. 6.** Comparison of attack depth over training episodes.

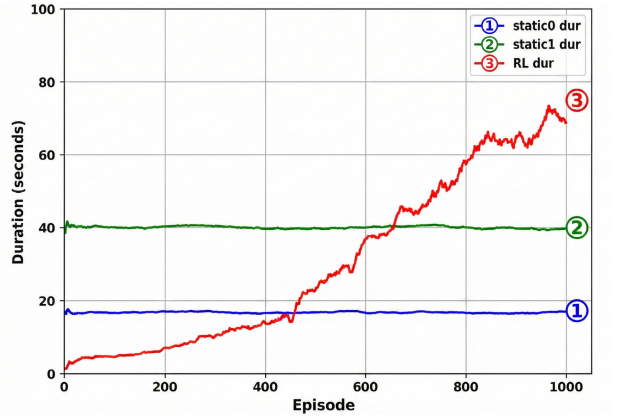


**Fig. 7.** Session duration over training episodes.

approaches show clear upward trajectories over time. During the initial 200–300 episodes, both RL agents remain in exploratory phases, occasionally performing worse than Static 1 due to random or suboptimal actions. As training continues, their performance improves: the small-state variant climbs more quickly, surpassing Static 1 around episode 400 and approaching tactic 11 by episode 1000. The big-state variant rises more gradually, catching up by approximately episode 1400, and thereafter maintains comparable depth.

The increase in attack depth is accompanied by a corresponding growth in session duration, suggesting that the RL-enhanced honeypot is able to sustain attacker engagement for longer periods. Figure 7 illustrates the evolution of the duration of the session in training episodes, plotted as a moving average. The x-axis represents the episode number, and the y-axis indicates the duration of the session in seconds. As shown, the baseline Static 0 configuration maintains a relatively short average duration of approximately 17 seconds. Static 1 improves this to around 40 seconds due to its slightly more credible responses. The dynamic honeypot

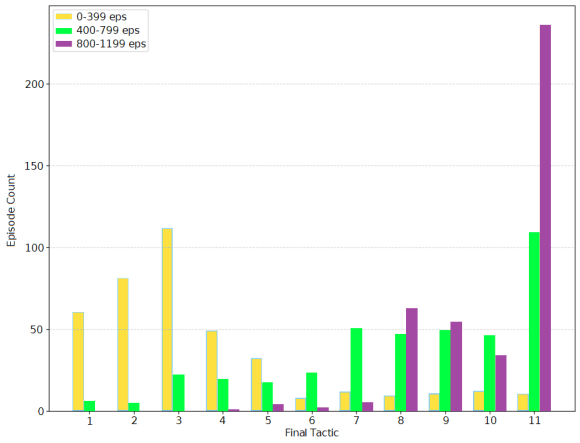


**Fig. 8.** **Distribution of final tactics over learning windows.**

begins with comparable session lengths but demonstrates a consistent upward trend, ultimately surpassing 70 seconds as the agent's policy converges. Analysis of packet-level traffic confirms that this additional engagement time is not idle: attackers use it to issue more advanced and varied ICS commands. These interactions provide defenders with high-value telemetry that captures complex adversarial behaviors, enabling more effective study and modeling of sophisticated ICS threat patterns.

To ensure that these results are not skewed by a small number of outlier sessions, we divide the 1200 total interactions into three consecutive windows of 400 episodes each. Figure 8 presents histograms showing the distribution of the final tactics reached in each window. In addition to illustrating policy convergence over time, the figure also highlights a second benefit: early-stage insight. In the first window (episodes 1-400), approximately 3% of the sessions already reach tactic 11. This implies that even within the initial five minutes of deployment, the dynamic honeypot can capture near-complete kill chains in a meaningful subset of interactions. In practical terms, such an early capability enables defenders to extract valuable intelligence on novel attack sequences almost immediately, without waiting for full policy convergence. As training progresses, performance improves steadily. In the second window (episodes 401–800), 28% of the sessions reach the final tactic, and by the third window (episodes 801–1200), it increases to 60%. These results indicate that the honeypot not only adapts to increasingly complex behaviors but also becomes more effective in sustaining adversary engagement throughout the entire ICS kill chain. The approach thus provides both early actionable intelligence and strong long-term reliability once the policy stabilizes.

Figure 9 compares the cumulative reward with the maximum tactic depth to illustrate how the agent learns over time. The top graph plots the total reward accumulated across episodes, showing a steady linear growth trend. The bottom graph depicts the corresponding increase in tactic depth.

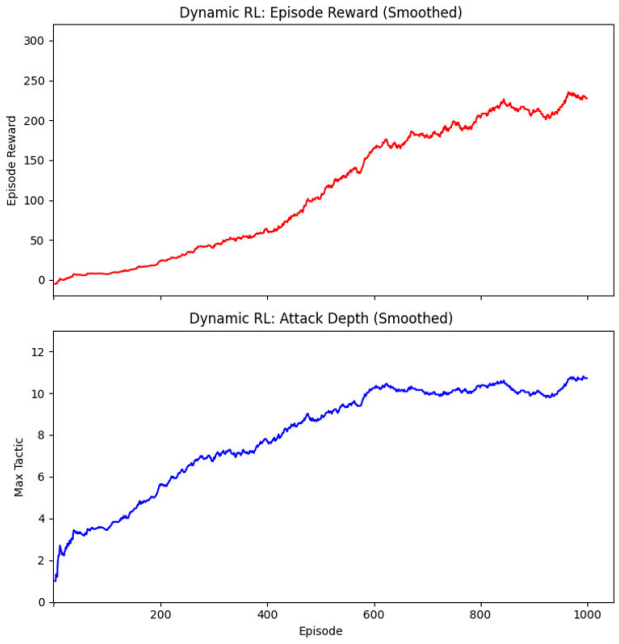


**Fig. 9.** **Reward and tactic.**

**TABLE 6.** **Final Q-table values.**

| Tactic | Allow | Restore | Delay | Modify | Block | System Error | Jitter |
|---|---|---|---|---|---|---|---|
| Initial Access | 10.08 | 2.29 | 3.3 | 3.53 | -4.76 | 2.49 | 3.18 |
| Execution | 11.5 | 6.97 | 2.34 | 2.66 | -4.6 | 9.58 | 2.44 |
| Persistence | 11.12 | 3.08 | 2.53 | 4.55 | -4.07 | -2.12 | 2.45 |
| Privilege escalation | 3.65 | 1.18 | 3.22 | 11.79 | -3.25 | 1.16 | 3.14 |
| Evasion | 13.51 | 4.05 | 2.71 | 3.47 | -3.25 | 0.93 | 2.33 |
| Discovery | 14.44 | 1.3 | 3.51 | 5.31 | -3.58 | 2.94 | 3.54 |
| Lateral Movement | 13.55 | 5.43 | 4.52 | 0.31 | -4.07 | 0.69 | 4.74 |
| Collection | 14.45 | 0.18 | 2.46 | 12.24 | -2.34 | 1.86 | 2.41 |
| C&C | 12.26 | 3.57 | 3.73 | 5.85 | -4.24 | 1.17 | 3.2 |
| Inhibit Response Function | 13.71 | 5.96 | 8.35 | 9.94 | -4.98 | 3.08 | 10.51 |
| Impair Process Control | 14.5 | 5.35 | 10.06 | 9.78 | -4.96 | 7.45 | 8.25 |
| Impact | -4.99 | 11.29 | 3.31 | 2.02 | -4.99 | 4.59 | 2.1 |

Because the reward function grants a bonus of $\tau + 1$ for each tactic advancement and only a single penalty upon disconnection, it naturally incentivizes deeper exploration. This alignment between the reward structure and observed behavior confirms that the agent learns as intended: by maximizing depth through extended attacker engagement.

A sample from the final Q-table is presented in Table 6, providing a human-readable interpretation of the learned policy. The table summarizes which deception actions the agent prefers at different stages of the attack sequence. In the early tactics, the agent assigns the highest Q-values to Allow and Modify, indicating a preference for maintaining session continuity and encouraging the adversary to continue probing. As the attacker progresses into mid-stage tactics (tactics 6 to 9), the agent increasingly favors Modify actions, which involve generating forged sensor values or manipulated register contents that simulate plausible but misleading system behavior, thereby promoting further escalation of the

attack. In the later stages of the kill chain, the preferred actions shift toward Delay and Jitter, which introduce subtle timing disruptions into the communication. These slow down the attacker's progress just enough to prolong engagement without prematurely ending the session or triggering a jump to the terminal Impact phase. Notably, the Block action consistently receives negative Q-values across all states, suggesting that the agent has learned to avoid prematurely terminating sessions, as doing so interrupts progression and results in lower cumulative reward.

Although Allow frequently appears among the highest-valued actions, it does not imply passivity. In our framework, actions are context-sensitive. For instance, in the early phases, it enables trust-building by letting benign-looking interactions pass, while in later phases, it is often combined with internal changes that simulate sabotage or success, without triggering alert thresholds. This balance helps preserve realism and prolong the presence of adversaries without obvious signs of manipulation. These trends confirm that the agent's policy is not biased toward inaction but instead adopts a staged deception strategy aligned with the progression of the adversary. In future extensions, multi-action compositions or probabilistic behavior modulation could introduce even richer response dynamics while retaining the structure and clarity of the learned policy.

To further clarify how deceptive actions affect protocol messages, we provide representative Modbus/TCP examples. A read-holding-registers request (FC = $0 \times 03$) is: `00 01 00 00 00 06 01 03 00 64 00 02`. A baseline response returns two registers: `00 01 00 00 00 07 01 03 04 00 FA 01 30`. Under the *Modify* action, the response remains protocol-valid while register contents are *perturbed relative to the current emulator-derived register state* (rather than synthesized arbitrarily), e.g., `00 01 00 00 00 07 01 03 04 00 F2 01 2A`. Here, "bounded ranges" refers to *bounded interventions* on the current value, implemented as a capped relative/absolute deviation (e.g., within a small percentage band such as $\pm 5\%$ and/or per-register thresholds specified in the YAML configuration), which prevents abrupt, implausible value jumps. For a write-single-register request (FC = $0 \times 06$), `00 02 00 00 00 06 01 06 00 64 03 E8`, the acknowledgement remains protocol-valid (typically echoing the request), while *Modify* is applied as a bounded perturbation to the internal in-memory register table before subsequent serialization. The *Jitter/Delay* action does not change packet bytes; it perturbs the response send time by a bounded delay with small random jitter to avoid timing artifacts. In addition, the reward design penalizes premature session termination, which discourages disruptive register perturbations that would quickly reveal deception and end the interaction.

These findings provide a direct answer to our first research question. After several hundred training episodes, the adaptive honeypot surpasses Static 0 by approximately six tactics and Static 1 by three, while also extending the median session duration by a factor of two. Importantly, these improvements are achieved using a simple tabular Q-learning model without neural networks or experience replay, ensuring that the learned policy remains interpretable and easily adjustable. The agent's ability to apply protocol-aware deception in the early stages and timing-based strategies in the later stages demonstrates an effective, self-improving defensive mechanism capable of eliciting deeper and more diverse attacker behaviors within a short training period.

*In summary, the RL-driven honeypot consistently draws attackers deeper into the ICS kill chain compared to static honeypots. By sustaining longer sessions and revealing more advanced tactics, this approach yields richer intelligence and strengthens its role as a proactive defense strategy for industrial environments.*

### C. STATE-SPACE SIZE: BIG VS. SMALL

To address the second research question, we examine how the choice of state representation influences learning dynamics. We compare two variants of the RL-driven honeypot: the *small-state* model, which tracks only the current ATT&CK tactic index $\tau$ (twelve possible values), and the *big-state* model, which augments this with protocol-level context such as the ICS protocol type and function code. This richer representation enables the agent to distinguish actions such as register reads, writes, and diagnostics, even within the same tactic. Although the big-state Q-table is about three times larger, it remains small by RL standards. In early training, when the agent is mainly advancing through tactics 1–6, both state definitions behave almost the same, and the added features provide little immediate benefit.

As shown in Figure 6, both models behave similarly during the first 300 episodes, typically reaching only around tactic 5 while still in the exploration phase. Between episodes 300 and 1000, however, the small-state agent moves ahead, advancing steadily to tactic levels 10–11 by about episode 1000. The big-state agent progresses more slowly, crossing tactic 9 roughly three hundred episodes later. This slowdown arises because, once the agent reaches deeper tactics ($\geq 7$), it encounters a greater variety of function codes. The big-state agent must refine Q-values across a larger set of substates, such as differentiating a coil read from a register write at the same tactic level. Since each update influences only one of these finer-grained entries, additional episodes are required before stable values emerge, resulting in the flat learning phase observed in the mid-range.

After episode 1250, the big-state agent accumulates enough experience with protocol–function combinations to employ that context more effectively. It begins selecting more precise deception actions—for instance, treating a harmless read differently from a potentially dangerous write—and its depth curve rises to roughly match that of

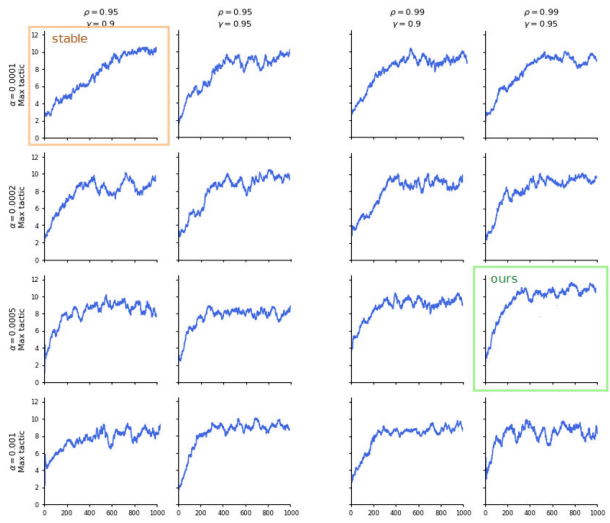


**Fig. 10. Maximum tactic progression across different RL parameter settings.**

the small-state agent. By episode 1400, both versions cluster around tactic 11, with the small-state agent maintaining a slight edge in maximum depth. Thus, the two approaches yield similar progression but differ in emphasis: the compact state offers faster convergence and slightly higher long-term depth, while the protocol-aware state provides finer-grained, context-sensitive deception.

#### 1) IMPLICATIONS FOR ROBUSTNESS BEYOND A FIXED SCRIPT ORDER

Although our evaluation uses scripted workflows for reproducibility, the proposed RL policy is not conditioned on a script-step index. Instead, the agent learns a state-conditioned policy $\pi(a \mid s)$, where $s$ is constructed from defender-observable semantic interaction context (e.g., the inferred tactic stage together with protocol/function context in the Big-State setting), rather than an explicit temporal identifier. Therefore, changes in the attacker's command ordering primarily alter the trajectory through the underlying state-transition graph (i.e., the sequence of visited states), rather than invalidating the decision rule itself. In particular, when the attacker's reordered actions induce states that are already represented by the learned abstraction, the policy remains applicable because the action selection is driven by semantic state content instead of its position in a predetermined sequence.

The Big-State configuration further strengthens this point by substantially enlarging the state-action space via protocol and function-level context, which increases the heterogeneity of interaction situations the agent must handle. The convergence behavior in Fig. 6 indicates that the agent learns stable, context-conditioned state–action mappings in this more granular space, rather than simply replaying a short linear "ladder" of tactic transitions. This provides supporting evidence that the learned behavior is not tied to a single rigid script order, but instead is grounded in semantic interaction states that can be encountered under non-linear attacker trajectories.

Furthermore, the Small-State and Big-State settings highlight a practical trade-off between contextual expressiveness and learning efficiency. Enriching the state with protocol/function context reduces context loss and avoids an overly simplistic state-machine behavior, but it also increases the effective state–action space and can slow convergence due to higher sample complexity. This behavior is consistent with our observation that Big-State converges more slowly while approaching comparable performance once sufficient experience is accumulated.

*In summary, the comparison highlights a practical trade-off: compact states support rapid deployment and efficient learning, whereas protocol-aware states deliver comparable depth with the added benefit of more realistic and targeted responses.*

### D. FINE-TUNING RL PARAMETERS: IMPACT AND OBSERVATION

To evaluate how our core RL parameters influence both the learning speed and the final performance, we perform a complete $4 \times 2 \times 2$ grid search across three hyperparameters: the learning rate $\alpha \in \{1 \times 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$, the exploration decay rate $\rho \in \{0.95, 0.99\}$ and the discount factor $\gamma \in \{0.90, 0.95\}$. For each configuration, we recorded the maximum tactic achieved over 1000 training episodes. Figure 10 presents the resulting learning curves, where each subplot corresponds to one of the 16 combinations of parameters. The x-axis shows the number of training episodes (from 0 to 1000), and the y-axis indicates the highest tactic reached in each session. The curves capture the agent's ability to escalate attacker engagement across different learning conditions.

Among these parameters, the learning rate $\alpha$ exerts the most noticeable effect on early training dynamics. A higher learning rate $\alpha = 1 \times 10^{-3}$ accelerates initial progress, reaching tactic 8 in fewer than 150 episodes, but then bounces up and down by three tactics. Those large updates overshoot the optimal Q-values, leading the agent to alternate between overly permissive and overly aggressive responses, which sometimes end sessions early. In contrast, a lower rate such as $\alpha = 1 \times 10^{-4}$ produces slower but more stable learning, leveling off around episode 800 without major fluctuations. This lower rate sacrifices speed for consistency, reducing unstable behavior that could alert the attacker or lead to early session termination.

The decay rate $\rho$ determines how long the agent continues to explore before moving towards exploitation. A slower decay of 0.99 keeps the agent exploring for roughly twice as many episodes as $\rho = 0.95$, giving it more opportunities to sample rare transitions, such as late-stage Modbus diagnostics. That extra exploration yields a final depth about one tactic higher on average. A faster decay converges sooner but

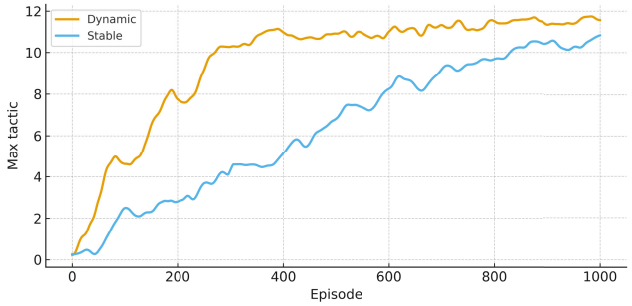


**Fig. 11. Maximum tactic progression using TD-error-based dynamic learning rate.**

stabilizes earlier, since the agent stops testing new actions before uncovering the deepest tactics.

The discount factor $\gamma$ plays a more subtle role. The increase in $\gamma$ from 0.90 to 0.95 shifts the emphasis toward long-term rewards, which slows the convergence by about 100-150 episodes but ultimately reaches a similar depth. A pilot at $\gamma = 0.97$ confirmed that too high a value makes the target term in the Q-update noisy, causing the curve to stall around tactic 8. Thus, $\gamma = 0.95$ offers a reasonable upper limit without destabilizing the training.

Based on these findings, we identify two profiles of practical parameters. For production use where stability and predictability are essential, the recommended configuration is $\alpha = 1 \times 10^{-4}$, $\gamma = 0.90$, $\rho = 0.95$. This profile consistently achieves tactic level 10 within 800 episodes (approximately 12 minutes of traffic) with low variance, making it safe for live ICS environments. For experimental or time-sensitive settings, the faster learning profile $\alpha = 5 \times 10^{-4}$, $\gamma = 0.95$, $\rho = 0.99$ reaches a similar depth in under 300 episodes (around 4 minutes) but shows $\pm 1$-tactic oscillations, acceptable within isolated test environments. In particular, even the slowest trace in our grid surpasses the Static 1 baseline (tactic 7) within 500 episodes.

In addition to these fixed profiles, we introduce a third configuration: a TD-error-based dynamic learning rate schedule. Rather than selecting a fixed $\alpha$ or using a predefined decay curve, the agent monitors its own temporal-difference (TD) error and automatically adjusts $\alpha$ in response. The learning rate begins at $\alpha = 5 \times 10^{-4}$ and is multiplied by 0.8 whenever the running average of the TD-error falls below a moving threshold, indicating that the policy is stabilizing. The exploration decay $\rho = 0.95$ and the discount factor$\gamma = 0.90$ are retained in the stable production profile, ensuring that the exploration window remains broad enough to support late-stage transitions while maintaining a practical bias toward short-term attacker behavior. Figure 11 illustrates the progression of maximum tactic depth under two learning rate profiles: a fixed stable rate and an adaptive dynamic schedule.

During the early phase (episodes 0–400), the dynamic schedule rises more quickly, reaching tactic 11 in roughly 400 episodes, while the stable profile lags behind at lower depths. In the mid-phase (400–700), the stable curve continues to climb gradually, whereas the dynamic agent maintains its lead with smoother gains and fewer oscillations. By the late phase (700–1000), both approaches converge near tactic 11–12, but the dynamic schedule achieves this plateau significantly earlier. Overall, the dynamic learning rate combines faster initial progress with stable long-term performance, outperforming the fixed-rate baseline in both speed and consistency.

*In summary, fine-tuning the RL parameters reveals clear trade-offs between speed and stability. The stable profile provides reliable convergence for production use, while the fast profile accelerates early learning at the cost of higher variance. The TD-error-based adaptive schedule then combines both advantages, achieving faster convergence with stable long-term performance and making the honeypot more resilient for diverse ICS environments.*

## VII. CONCLUSION AND FUTURE WORKS

This paper proposed and evaluated an adaptive deception system for ICS, integrating RL into a dynamic honeypot. The solution combines three modules: a proxy extracting protocol features, a mapper converting interactions into MITRE ICS tactics, and a tabular Q-learning agent selecting actions aligned with MITRE Engage. Together, these transform a static honeypot into a real-time adaptive system.

Results were examined in three dimensions. First, adaptive deception outperformed static variants: in 1200 attack sessions, the RL-driven honeypot reached tactic levels 10–11 within 700 episodes, versus a ceiling of 7 for the best static setup. Median session duration more than doubled, and 60% of runs covered the full kill chain within five minutes of training, showing that even a lightweight learner yields consistent, high-value intelligence.

Second, state-space size shaped performance. The compact state (tactic index only) converged faster, reaching tactic levels 10–11 by about episode 1000. The protocol-aware state (including protocol and function code) required closer to 1400 episodes to reach comparable depth, but offered more context-sensitive and realistic responses. This highlights a trade-off between fast initialization and fine-grained control.

Third, hyperparameter tuning showed trade-offs between speed and stability. A conservative setup reached tactic 11 in under 12 minutes with low variance, while a fast profile achieved the same depth in less than four minutes with minor oscillations. All configurations surpassed the static baseline within 500 episodes. A TD-error-based adaptive learning rate further combined both advantages, reaching tactic 11 in 400 episodes and maintaining stability within $\pm 0.5$—without additional tuning parameters—making the approach robust across attacker behaviors and deployment contexts.

Several extensions can further improve the generalizability and deployment readiness of the approach. First,

future work will extend the RL state with lightweight temporal and longitudinal context—e.g., discretized inter-packet/inter-action timing and cross-session summaries—so the agent can better recognize low-and-slow behaviors and strategies that unfold across sessions, beyond the multi-step progression already captured by MITRE ATT&CK tactics.

Second, although the RL agent was trained on varied interaction patterns, future studies should test generalization to unseen attack workflows. Because the policy is tactic-driven rather than exploit-specific, it should adapt based on progression stage, but constructing new ICS kill chains would provide a stronger assessment of robustness against zero-day scenarios. In addition, the learned policy can be evaluated under more realistic attacker uncertainty by introducing randomized/obfuscated workflows (e.g., varying command order, timing, and target parameters within safe bounds) and by conducting human-driven red-team exercises to assess robustness against non-linear and adaptive adversary behaviors.

Third, an important direction is balancing state expressiveness (to avoid context loss) with learning efficiency (to control sample complexity) when moving from Small-State to Big-State representations. Hierarchical Reinforcement Learning (HRL) is a promising approach, where a high-level policy on a compact representation (e.g., tactic stage) selects a coarse deception intent/action family, while a low-level policy conditioned on richer protocol/function context executes fine-grained responses. Another complementary approach is progressive state expansion (curriculum/transfer learning): training starts from the Small-State formulation for fast coarse policy acquisition and then expands to the Big-State formulation for context-aware refinement, e.g., by warm-starting context-specific value estimates from the corresponding tactic-level estimates before fine-tuning.

Lastly, future work will explicitly address attacker-side deception detection (honeypot fingerprinting and suspicion) by incorporating timing/protocol-consistency cues as additional state features or as a penalty term in the reward, enabling the agent to balance engagement depth with stealth under realistic adversary scrutiny. In addition, we will validate this stealth-aware design under more realistic ICS conditions, including heterogeneous device profiles, mixed benign/background traffic (e.g., trace replay or traffic generation), and network impairments such as congestion-induced delay/jitter/loss, to assess robustness beyond the controlled two-VM environment.

## REFERENCES


[1] E. D. Knapp, *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems*. Amsterdam, The Netherlands: Elsevier, 2024.

[2] R. M. Lee, M. J. Assante, and T. Conway, "*Analysis of the cyber attack on the Ukrainian power grid: Defense use case*," E-ISAC & SANS Institute, Technical.

[3] M. Kaouk, J.-M. Flaus, M.-L. Potet, and R. Groz, "A review of intrusion detection systems for industrial control systems," in *Proc. 6th Int. Conf. Control, Decis. Inf. Technol. (CoDIT)*, Apr. 2019, pp. 1699–1704, doi: 10.1109/CODIT.2019.8820602.

[4] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, "Attack taxonomies for the modbus protocols," *Int. J. Crit. Infrastructure Protection*, vol. 1, pp. 37–44, Dec. 2008, doi: 10.1016/j.ijcip.2008.08.003.

[5] R. Lukas, V. Johnny, H. Daniel, and P. Andrea. (2020). *Conpot ICS/SCADA Honeypot*. [Online]. Available: http://conpot.org/

[6] C. Jianhong, L. Wei, L. Jianjun, and L. Bo, *DiPot: A Distributed Industrial Honeypot System*. Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-73830-7_30.

[7] S. E. Blake et al. (2018). *MITRE ATT&CK: Design and Philosophy*. MITRE Corporation. [Online]. Available: https://www.mitre.org/sites/default/files/2021-11/prs-19-01075-28-mitre-attack-design-and-philosophy.pdf

[8] A. Ghaleb, S. Zhioua, and A. Almulhem, "On PLC network security," *Int. J. Crit. Infrastructure Protection*, vol. 22, pp. 62–69, Sep. 2018, doi: 10.1016/j.ijcip.2018.05.004.

[9] S. East, J. Butts, M. Papa, and S. Shenoi, "A taxonomy of attacks on the DNP3 protocol," in *Critical Infrastructure Protection III*. Berlin, Germany: Springer, 2009, pp. 67–81, doi: 10.1007/978-3-642-04798-5_5.

[10] The MITRE Corporation. (2016). *2016 Ukraine Electric Power Attack, Campaign C0025 — MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/campaigns/C0025/

[11] (2022). *2022 Ukraine Electric Power Attack, Campaign C0034 — MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/campaigns/C0034/

[12] C.-A. Petre and A. Korodi, "Honeypot inside an OPC UA wrapper for water pumping stations," in *Proc. 22nd Int. Conf. Control Syst. Comput. Sci. (CSCS)*, May 2019, pp. 72–77, doi: 10.1109/CSCS.2019.00020.

[13] M. Conti, F. Trolese, and F. Turrin, "ICSpot: A high-interaction honeypot for industrial control systems," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Mauro and Trolese, France, Jul. 2022, pp. 1–4, doi: 10.1109/ISNCC55209.2022.9851732.

[14] M.-R. Zamiri-Gourabi, A. R. Qalaei, and B. A. Azad, "Gas what?: I can see your GasPots. Studying the fingerprintability of ICS honeypots in the wild," in *Proc. 5th Annu. Ind. Control Syst. Secur. (ICSS) Workshop*, Dec. 2019, pp. 30–37, doi: 10.1145/3372318.3372322.

[15] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for Internet of Things, Industrial Internet of Things, and cyber-physical systems," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2351–2383, 4th Quart., 2021, doi: 10.1109/COMST.2021.3106669.

[16] (2022). *Mitre Att&k for Industrial Control Systems (ICS) Matrix*. [Online]. Available: https://attack.mitre.org/matrices/ics/

[17] (2022). *Matrix — MITRE Engage*. [Online]. Available: https://engage.mitre.org/matrix/

[18] A. Pauna and I. Bica, "RASSH–reinforced adaptive SSH honeypot," in *Proc. 10th Int. Conf. Commun. (COMM)*, May 2014, pp. 1–6, doi: 10.1109/ICCOMM.2014.6866707.

[19] A. Pauna, A.-C. Iacob, and I. Bica, "QRASSH—A self-adaptive SSH honeypot driven by Q-learning," in *Proc. Int. Conf. Commun. (COMM)*, Jun. 2018, pp. 441–446, doi: 10.1109/ICCOMM.2018.8484261.

[20] D. S. Lorenzo, "Cannypot: A reinforcement learning based adaptive SSH honeypot," Ph.D. dissertation, Master Sci. Program Comput. Eng., Polytechnic University of Turin, Turin, Italy, 2021. [Online]. Available: https://webthesis.biblio.polito.it/secure/19256/1/tesi.pdf

[21] C. Guan, H. Liu, G. Cao, S. Zhu, and T. La Porta, "HoneyIoT: Adaptive high-interaction honeypot for IoT devices through reinforcement learning," in *Proc. 16th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, May 2023, pp. 49–59, doi: 10.1145/3558482.3590195.

[22] V. S. Mfogo, A. Zemkoho, L. Njilla, M. Nkenlifack, and C. Kamhoua, "AIIPot: Adaptive intelligent-interaction honeypot for IoT devices," in *Proc. IEEE 34th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2023, pp. 1–6.

[23] A. G. Barto, "Reinforcement learning," *Adaptation*, vol. 31, no. 29, p. 5, 1998. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-642-27645-3.pdf

[24] W. Cabral, C. Valli, L. Sikos, and S. Wakeling, "Review and analysis of cowrie artefacts and their potential to be used deceptively," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2019, pp. 166–171, doi: 10.1109/CSCI49370.2019.00035.

[25] A. Vetterl, R. Clayton, and I. Walden, "Counting outdated honeypots: Legal and useful," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2019, pp. 224–229, doi: 10.1109/SPW.2019.00049.

[26] F. Nourmohammadzadeh Motlagh, M. Hajizadeh, M. Majd, P. Najafi, F. Cheng, and C. Meinel, "Large language models in cybersecurity: State-of-the-art," 2024, *arXiv:2402.00891*.

[27] N. Daniel, F. K. Kaiser, A. Dzega, A. Elyashar, and R. Puzis, *Labeling NIDS Rules With MITRE ATT&CK Techniques Using ChatGPT*. Cham, Switzerland: Springer, 2024, p. 76, doi: 10.1007/978-3-031-54129-25.
[28] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annu. Rev. Statist. Its Appl.*, vol. 7, no. 1, pp. 279–301, Mar. 2020, doi: 10.1146/annurev-statistics-031219-041220.
[29] A. Jicha, M. Patton, and H. Chen, "SCADA honeypots: An in-depth analysis of conpot," in *Proc. IEEE Conf. Intell. Secur. Informat. (ISI)*, Sep. 2016, pp. 196–198, doi: 10.1109/ISI.2016.7745468.
[30] MITRE Corporation. (2022). *MITRE Campaigns — MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/campaigns/
[31] G. D. Seth, "Verifying cyber implementation best practices with malcolm," in *A Powerful Network Traffic Analysis Tool Suite*. Idaho Falls, ID, USA: Idaho National Laboratory, 2024. [Online]. Available: https://www.osti.gov/biblio/2480663
[32] V. Ndatinya, Z. Xiao, V. R. Manepalli, K. Meng, and Y. Xiao, "Network forensics analysis using wireshark," *Int. J. Secur. Netw.*, vol. 10, no. 2, p. 91, 2015, doi: 10.1504/ijsn.2015.070421.
[33] O. Valea and C. Oprisa, "Towards pentesting automation using the metasploit framework," in *Proc. IEEE 16th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2020, pp. 171–178, doi: 10.1109/ICCP51029.2020.9266234.
[34] M. Tigner, H. Wimmer, and C. M. Rebman, "Analysis of kali Linux penetration tools: A survey of hacking tools," in *Proc. Int. Conf. Electr., Comput. Energy Technol. (ICECET)*, Dec. 2021, pp. 1–6, doi: 10.1109/ICECET52533.2021.9698572.
[35] R. R. S, R. Rohith, M. Moharir, and G. Shobha, "SCAPY—A powerful interactive packet manipulation program," in *Proc. Int. Conf. Netw., Embedded Wireless Syst. (ICNEWS)*, Dec. 2018, pp. 1–5, doi: 10.1109/ICNEWS.2018.8903954.
[36] A. Stooke and P. Abbeel, "Rlpyt: A research code base for deep reinforcement learning in PyTorch," 2019, *arXiv:1909.01500*.
[37] B. Ekaba, *Matplotlib and Seaborn*. New York, NY, USA: Apress, 2019, doi: 10.1007/978-1-4842-4470-812.
[38] A. Tom and S. Emma, *Overview of the OpenAI APIs*. New York, NY, USA: Apress, 2024, doi: 10.1007/979-8-8688-0885-26.

**YING-DAR LIN** (Fellow, IEEE) received the Ph.D. degree in computer science from the University of California at Los Angeles (UCLA) in 1993. He is currently a Chair Professor in computer science with National Yang Ming Chiao Tung University (NYCU), Taiwan. He was a Visiting Scholar with Cisco Systems, San Jose, from 2007 to 2008; the CEO of the Telecom Technology Center, Taiwan, from 2010 to 2011; and the Vice President of National Applied Research Laboratories (NARLabs), Taiwan, from 2017 to 2018. He co-founded L7 Networks Inc., in 2002 and O'Prueba Inc., in 2018. His research interests include cybersecurity, wireless communications, network softwarization, and machine learning for communications. He has served or serving on the editorial boards for several IEEE journals and magazines, including the Editor-in-Chief for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS (COMST) (2017–2020).

**RASUL MANKAEV** received the Graduate degree from the EECS International Graduate Program, National Yang Ming Chiao Tung University (NYCU), in 2025. His research interests include network security, intrusion detection, and threat expertise.

**DIDIK SUDYANA** (Member, IEEE) received the Ph.D. degree from the Department of Electrical Engineering and Computer Science (EECS), National Yang Ming Chiao Tung University (NYCU), in 2024. He is currently an Assistant Professor with the Computer and Network Center, National Cheng Kung University (NCKU), Taiwan. His research interests include cybersecurity, machine learning, and network design and optimization.

**YUAN-CHENG LAI** received the Ph.D. degree from the Department of Computer and Information Science, National Chiao Tung University, in 1997. He joined as a Faculty Member of the Department of Information Management, National Taiwan University of Science and Technology, in August 2001, and has been a Distinguished Professor since June 2012. His research interests include performance analysis, software-defined networking, wireless networks, and the IoT security.

**REN-HUNG HWANG** (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Massachusetts, Amherst. He is currently the Dean of the College of Artificial Intelligence, National Yang Ming Chiao Tung University (NYCU), Taiwan. Before joining NYCU, he was with National Chung Cheng University, Taiwan, from 1993 to 2022. He has published more than 250 international journals and conference papers. He was as the Dean of the College of Engineering from 2014 to 2017. His current research interests include deep learning, wireless communications, network security, and cloud/edge/fog computing. He received the IEEE Best Paper Award from IEEE UbiMedia 2018, IEEE SC2 2017, and IEEE IUCC 2014.