Research Square

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

# Adversarial Self-Play for Adaptive Cyber Deception Against Reinforcement Learning-Based Bots in SDN-Enabled Edge Networks

**Yannick Florian YANKAM**

yyankam@yahoo.fr

Fotso Victor University Institute of Technology, University of Dschang

**Kerol Kafack Wamba**

Université de Dschang

**Article**

**Keywords:** Reinforcement learning, Cyber deception, Multi-agent stochastic game, Software-defined networking, Edge computing

**Posted Date:** November 4th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11171486/v1

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

**Additional Declarations:** No competing interests reported.

# Adversarial Self-Play for Adaptive Cyber Deception Against Reinforcement Learning-Based Bots in SDN-Enabled Edge Networks

Yannick Florian YANKAM[1*] and Kerol Kafack Wamba[2]

[1*]Department of Telecommunication and Network Engineering, IUT-FV, University of Dschang, 134, Bandjoun, Cameroon.
[2]Research Unit in Fundamental Computer Science, Engineering and Applications (URIFIA), Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon.

*Corresponding author(s). E-mail(s): yyankam@yahoo.fr;
Contributing authors: kerolwamba239@gmail.com;

**Abstract**

Static defenses such as firewalls, signature-based detection, and fixed-rule honeypots assume an adversary whose techniques are known in advance. That assumption no longer holds: bots trained by reinforcement learning probe, fail, adapt, and retry, so static countermeasures are eventually learned and bypassed. This article proposes a deception framework in which the defender also learns. A defender agent trained with proximal policy optimization observes the network through a software-defined controller and orchestrates, in real time, honeypot deployment, moving target defense mutations, and flow isolation. The confrontation is formalized as a two-agent stochastic game decomposed into two coupled Markov decision processes, and both policies are trained symmetrically by adversarial self-play against a pool of past opponents, so neither agent ever faces a frozen target. Because decoys and mutations consume bandwidth and energy, the reward explicitly charges the cost of each defensive action, keeping the learned policies deployable on constrained edge nodes. The framework is implemented on an emulated SDN/edge infrastructure built with Mininet and Ryu, then evaluated on two attack scenarios targeting the sensor subnetwork and the edge servers, respectively. Against a no-defense baseline and a static honeypot baseline, dynamic defense raises the defensive success rate to about 98% on the IoT subnetwork and 97% on the edge servers, extends the time to compromise from about 10 to 50 seconds in the first scenario and to 65 to 80 seconds in the second, and drives the attacker reward to near zero while keeping reconfiguration latency below 50ms.

**Keywords:** Reinforcement learning, Cyber deception, Multi-agent stochastic game, Software-defined networking, Edge computing

## 1 Introduction

Edge computing and software-defined networking (SDN) have converged into the dominant architecture for latency-sensitive services. The former

pushes storage and processing next to the data sources, whether sensors, cameras, or industrial terminals [1]; the latter decouples the control plane from the data plane and turns the network into a programmable substrate [2]. Together they support industrial automation, connected vehicles, and smart hospital systems [3].

This agility, however, comes at the cost of increased exposure. Peripheral nodes are numerous, resource-constrained, and often physically reachable, so an adversary no longer needs to reach the network core: a single poorly protected sensor is enough to gain a foothold [4]. Worse still, the adversary itself has changed. Bots endowed with reinforcement learning (RL) now plan intrusions autonomously, revising their strategy at every interaction [5, 6].

This exposes a structural weakness of classical defenses. Firewalls, access control lists, and signature-based detection recognize behaviors that have already been catalogued, whereas an RL bot complies with no fixed profile: it probes, fails, learns, and starts again differently [7]. Cyber deception offers a different bargain. Rather than blocking the adversary, it misleads it, steering it toward fictitious resources so as to consume its time and expose its intentions without exposing the real assets [4]. Yet most deployed deception systems are themselves static or rule-driven, which merely displaces the problem: once the decoy has a recognizable fingerprint, it is avoided [8].

The question we address follows directly. *How can a deception system be designed that learns and adapts fast enough to counter an RL bot inside an SDN-orchestrated edge environment?* Three subproblems structure the answer: how to model the defender–attacker interaction so that an optimal deception policy can be learned; which deception resources maximize the delay imposed on the attacker at acceptable operational cost; and how far SDN orchestration can reconfigure the environment in real time without disturbing legitimate traffic.

Our contributions are fourfold.

- We formalize the confrontation as a two-agent stochastic game with adversarial self-play, decomposed into two coupled Markov decision processes with explicit states, actions, and antagonistic rewards.
- We design a deception architecture in which the SDN controller acts at once as sensor and actuator, orchestrating honeypots, moving target defense (MTD) mutations, and flow isolation under a transparency invariant that preserves legitimate traffic.
- We charge the operational cost of every defensive action inside the reward and the observation vector, so that the learned policies remain sustainable on constrained edge nodes.
- We evaluate the framework against an RL attacker of realistic complexity on an emulated SDN/edge/IoT testbed, using a three-phase protocol that isolates the specific contribution of online adaptation.

Section 2 reviews the literature and extracts four gaps; Sections 3 and 4 present the architecture and the game formulation; Section 5 reports the experiments; Sections 6 and 7 discuss and conclude.

# 2 Related Work and Positioning of the Proposed Approach

## 2.1 Deception and Mobility Without Learning

Deception began with honeypots. Low-interaction decoys such as Honeyd [9] emulate a handful of services, while high-interaction ones expose a full operating system and reveal the intruder's procedures in detail. A recent survey of honeypots deployed in IoT and cyber-physical settings [10] nevertheless converges on a single verdict: production solutions remain static, so an attacker that has once fingerprinted the decoy sidesteps it thereafter.

MTD was formalized to correct the asymmetry between attacker and defender [11]. Random Host Mutation [12] periodically rotates IP addresses, an idea later optimized for SDN where the controller can hop addresses on a per-transmission basis [13], and extended into self-shielding architectures that combine mutation, machine shuffling, and decoys [14]. All share the same limitation: the mutation schedule is fixed a priori and never reacts to what the adversary actually does.

Game theory was the natural next step. Zero-sum [15] and Stackelberg [16] formulations let the

defender calibrate the frequency and amplitude of randomization, and trilateral games extend the reasoning to cloud environments while accounting for defensive cost [17]. Their contribution is conceptual, since they establish that a well-tuned randomized policy can be optimal in the Nash sense. The price they pay is the set of assumptions that tractability demands, namely perfect rationality, small state spaces, and common knowledge of rewards. Pragmatic heuristics such as cost-effective shuffling [18] or low-delay randomization for industrial control [19] drop those assumptions but still reason on a frozen model of the adversary, which recent surveys consider increasingly untenable [20].

## 2.2 Learning-Based Approaches: When Both Sides Adapt

The offensive side matured first. Penetration testing was cast as a partially observable Markov decision process solved by Q-learning [21], then scaled with deep Q-networks coupled to attack graphs [22] and hierarchical architectures [23]. An RL attacker starting from minimal knowledge learns to scan, select services, exploit, and pivot, outperforming conventional automated tools in both coverage and speed [6]. CyberBattleSim [24] provided a shared platform, and the deception study built on it [25] delivers the key observation for our purpose: decoys slow an RL attacker markedly, but the attacker eventually learns around them whenever the deception policy stays frozen.

Defensive RL followed. Single-agent formulations select mutation type and frequency with a double deep Q-network [26], degrade reconnaissance accuracy by up to 73 percent [27], or drive SDN countermeasures from OpenFlow feedback [28]. Each of these, however, treats the environment as stationary, which amounts to training the defender against an immobile target. Multi-agent Markov games [5, 29] and self-play [30] lift that assumption by letting both policies co-evolve, and a comparison of DQN, DDQN, PPO, and SAC on decoy deployment [7] reports that PPO offers the best stability–performance trade-off under short training budgets. Hybrid RL and game-theoretic schemes [31] together with joint honeypot/MTD evaluations [32] complete the picture, the latter showing that coupling deception with mobility yields more than the sum of the two mechanisms.

## 2.3 SDN-, Edge-, and IoT-Specific Deception Approaches

A third strand exploits SDN programmability directly. Virtual topologies present each attacker with a different view of the network [33], in-network honeypots redirect suspicious traffic toward logical decoys without dedicated hardware [34], and self-adaptive MTD tunes decoy mutation to the observed rate of intrusion attempts on IoT networks [8]. Network function virtualization has been combined with SDN to deploy self-healing honeynets against IoT botnets [35], while threat-intelligence-driven honey-of-things placement targets edge nodes without resorting to RL [36]. Closest to our concerns, a PPO agent deploys honeypots across 5G slices under a reward that penalizes excessive resource usage [37], and CYBERSHIELD [38] models IoT energy limits and data-plane latency explicitly. In every case, though, the adversary remains scripted.

Two properties of this ecosystem shape what a defense can reasonably do. The first is heterogeneity: IoT devices differ in protocol, firmware, and update policy, which is precisely why uniform countermeasures underperform there and why botnets such as Mirai spread as far as they did [8]. The second is centralization: the SDN controller's global view is what makes sub-second, service-preserving reconfiguration feasible in the first place, yet that same view turns the controller into a single point of failure, since an adversary that captures it can reroute traffic at will [3]. Our design leans deliberately on the first property and, as stated in Section 3, assumes the second away.

## 2.4 Summary of Research Gaps

Four limitations therefore recur. *G1, static defenses.* A large share of deployed solutions still relies on rules fixed at design time and becomes obsolete once mutation patterns are memorized [4, 20]. *G2, a stationary adversary.* Even when RL sits on the defender side, the attacker is frequently assumed fixed [5, 30]. *G3, operational cost ignored.* Decoys and mutations consume bandwidth, energy, and maintenance, yet the cost is rarely written into the reward, which yields policies that are optimal on paper and awkward to deploy. *G4, no benchmark against a reference RL bot.* To our knowledge, no shared evaluation

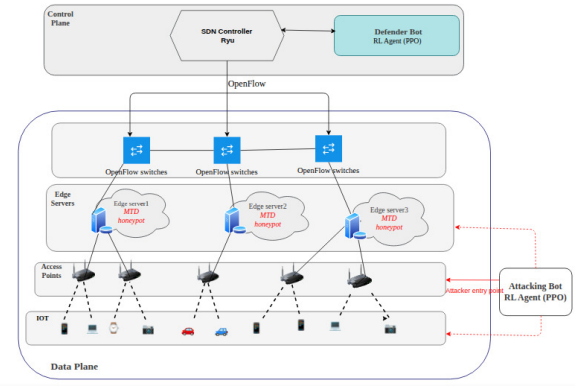



**Fig. 1** Overall SDN/edge/IoT architecture. The defender sits in the control plane beside the Ryu controller; the attacker enters through an access point or an edge segment.

confronts an RL-driven deceptive defender with an RL attacker of realistic complexity inside an instrumented SDN/edge/IoT environment.

Our framework sits precisely at the intersection of the four: the defender learns from repeated interaction with an attacker that is itself learning, the environment reproduces the SDN/edge/IoT stack, the reward charges the cost of deception, and both agents evolve simultaneously during evaluation.

# 3 System Architecture

## 3.1 A Three-Plane Architecture: Control, Data, and Edge Application

The framework rests on the classical SDN separation, enriched with an edge application plane dedicated to services and decoys (Fig. 1). The *control plane* hosts the Ryu controller, which holds the global view through OpenFlow, and the defender bot, whose decisions become flow rules. The *data plane* carries the OpenFlow switches interconnecting edge servers, access points, and IoT devices. The *edge application plane* runs the legitimate services and hosts, on demand, honeypots and configuration mutations.

This split is what makes fast and transparent reconfiguration possible. The controller is simultaneously the *sensor* of the defensive loop, since OpenFlow `Packet-In` messages continuously report throughput, active flow counts, and source-address entropy, and its *actuator*, since every decision materializes as a `Flow-Mod` rule pushed to the switches. It is therefore the pivot of the whole system, and, by the same token, its critical point of failure.

Two node classes deserve emphasis. Edge servers concentrate proximity processing and aggregate the data of every device attached to them, which makes a single compromise unusually costly; they also execute the application-level decoys. IoT devices, conversely, are the most exposed layer, weakly authenticated and physically accessible, and thus the natural entry point.

## 3.2 Threat Model and Operating Assumptions

Before turning to the mechanisms, it is worth stating who the adversary is and what each side is allowed to do, since the results that follow are only valid within those bounds.

The attacker is external and starts with no knowledge of the topology: it must discover hosts by probing, and its only view of the network is what its own actions return. It can scan a node, exploit a vulnerability, pivot from an already compromised host, or wait; it cannot, however, reach the control plane, read $T_{\mathrm{MTD}}$, or observe the defender's internal state. Its goal is a single successful exfiltration, and it is indifferent to the noise it makes except insofar as detection costs it time.

The defender, conversely, sees the network globally through OpenFlow but acts only on the network layer. It installs, modifies, and removes flow rules and spawns decoys; it does not patch vulnerabilities, restart hosts, or alter application code, so a vulnerable node stays vulnerable throughout an episode. We further assume the controller itself is trusted and uncompromised, which is the standard hypothesis in SDN-based defense [28, 33], and we treat detection as an imperfect, externally supplied signal rather than a component we contribute; both assumptions are revisited in Section 6. Finally, legitimate traffic is a hard constraint, not an objective to be traded off: no defensive action may interrupt it, a constraint addressed in the following subsection.

**Table 1** Deception mechanisms available to the defender.

| Mechanism | Layer | Type | Effect on the attacker |
|---|---|---|---|
| MTD (IP / port) | Data plane | Reactive | Invalidates the mapped target and forces a new reconnaissance phase |
| Honeypot | Edge application plane | Proactive | Bait data raising an alert as soon as they are accessed |
| Isolation | Data plane | Reactive | Transparently redirects traffic toward a sinkhole |

### 3.3 Preserving Traffic Transparency Under Reconfiguration

Every reconfiguration obeys one invariant: legitimate traffic must never notice it. The controller therefore maintains a mapping table

$$T_{\mathrm{MTD}} : \mathrm{IP}_{\mathrm{fict}} \to \mathrm{IP}_{\mathrm{real}}. \tag{1}$$

When a mutation fires on node $i$, the controller draws a fresh address $\mathrm{IP}^{(i)}_{\mathrm{fict}}$, registers it in $T_{\mathrm{MTD}}$, and installs a `Flow-Mod` rule that rewrites destination headers on the fly. Legitimate packets thus still reach $\mathrm{IP}^{(i)}_{\mathrm{real}}$ without interruption, whereas the attacker, which holds only the fictitious address, is left with an outdated map from that instant onward. The asymmetry created by Eq. (1) is the mechanical source of every delay reported in Section 5.

### 3.4 Deception Mechanisms

The defender has two complementary families, summarized in Table 1. MTD acts reactively on the data plane by invalidating the map the attacker has patiently built, forcing a fresh reconnaissance phase. Honeypots act proactively by redirecting suspicious traffic toward a fake server whose database holds only fictitious records, which both protects the real assets and reveals the intruder's techniques. As a last resort, the controller isolates a malicious flow toward a sinkhole, silently, without emitting any error that might alert the intruder.

## 4 Adversarial Game and Learning

### 4.1 Formulation as a Two-Agent Stochastic Game

To capture the adversarial dynamics faithfully, we represent the confrontation as the stochastic game formalized in Eq. (2):

$$\mathcal{G} = \big(\mathcal{S},\, \mathcal{A}_{\mathrm{atk}},\, \mathcal{A}_{\mathrm{def}},\, T,\, R_{\mathrm{atk}},\, R_{\mathrm{def}},\, \gamma\big), \tag{2}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}_{\mathrm{atk}}$ and $\mathcal{A}_{\mathrm{def}}$ the action spaces, $T : \mathcal{S} \times \mathcal{A}_{\mathrm{atk}} \times \mathcal{A}_{\mathrm{def}} \to \Delta(\mathcal{S})$ the transition kernel, and $\gamma \in [0, 1)$ the discount factor. The game is nearly zero-sum, that is $R_{\mathrm{atk}} + R_{\mathrm{def}} \approx 0$: up to a cost term, what slows the attacker rewards the defender. This is precisely the structure under which adaptive strategies emerge in self-play [30].

From each agent's viewpoint the opponent is a non-stationary part of the environment, so the game splits into two coupled Markov decision processes,

$$\mathcal{M}_{\mathrm{atk}} = \big(\mathcal{S}_{\mathrm{atk}}, \mathcal{A}_{\mathrm{atk}}, T^{\pi_{\mathrm{def}}}, R_{\mathrm{atk}}, \gamma\big), \tag{3}$$

$$\mathcal{M}_{\mathrm{def}} = \big(\mathcal{S}_{\mathrm{def}}, \mathcal{A}_{\mathrm{def}}, T^{\pi_{\mathrm{atk}}}, R_{\mathrm{def}}, \gamma\big). \tag{4}$$

It is this decomposition, Eqs. (3)–(4), that makes the problem tractable by standard RL applied to each agent, while preserving their interaction.

### 4.2 Attacker Model: Observations, Actions, and Reward

The attacker perceives only what it has itself discovered, which its state vector reflects, as expressed in Eq. (5):

$$s_{\mathrm{atk}} = \big(o_{\mathrm{scan}},\, o_{\mathrm{resp}},\, n_{\mathrm{tgt}},\, t_{\mathrm{ep}},\, h_{\mathrm{atk}}\big)^{\top}. \tag{5}$$

Its components appear in the upper half of Table 2. Its action space, defined in Eq. (6), gathers the primitives of a progressive intrusion:

$$\mathcal{A}_{\mathrm{atk}} = \big\{\mathrm{scan}(i),\ \mathrm{exploit}(i, v),\ \mathrm{pivot}(i \to j),\ \mathrm{wait}\big\}, \tag{6}$$

where $\mathrm{exploit}(i, v)$ denotes exploitation of node $i$; the vulnerability $v$ is service-dependent, but it is resolved by the environment rather than selected

**Table 2** State variables of both agents.

| Component | Dimension | Description |
|---|---|---|
| *Attacker* | | |
| $o_{\text{scan}}$ | $N$ | Node $i$ answering (1) or silent (0) |
| $o_{\text{resp}}$ | $N_{\text{svc}}$ | Services identified on the current target |
| $n_{\text{tgt}}$ | 1 | Nodes not yet attempted |
| $t_{\text{ep}}$ | 1 | Time elapsed in the episode |
| $h_{\text{atk}}$ | $w \cdot \lvert\mathcal{A}_{\text{atk}}\rvert$ | Sliding window of the $w$ latest actions |
| *Defender* | | |
| $f_{\text{flow}}$ | $4N_{\text{sw}}$ | Per switch: in/out throughput, active flows, source entropy |
| $m_{\text{hp}}$ | $N_E$ | Honeypot deployed (1) on each edge node |
| $m_{\text{mtd}}$ | $N_I$ | MTD active (1) on each node |
| $a_{\text{alert}}$ | $N_E$ | Alert level in $[0, 1]$ per node |
| $c_{\text{res}}$ | 1 | Fraction of edge resources consumed by decoys |
| $h_{\text{def}}$ | $w \cdot \lvert\mathcal{A}_{\text{def}}\rvert$ | Sliding window of the $w$ latest actions |

by the policy, so the agent's actual decision variable is only the target node $i$ once that node has been scanned. Accordingly, the action space grows quadratically with the number of nodes, and its size counts one exploit action per node, not per (node, vulnerability) pair:

$$|\mathcal{A}_{\text{atk}}| = 2N + \tfrac{1}{2}N(N-1) + 1, \tag{7}$$

which weighs on the choice of learning algorithm. Its reward encodes a single objective, to exfiltrate as fast as possible:

$$R_{\text{atk}} = \alpha \cdot \mathbf{1}[\text{success}] + \beta \cdot \mathbf{1}[\text{new information}] - \delta, \tag{8}$$

with a terminal reward $\alpha = 50$, a progression bonus $\beta = 5$ for any action revealing unknown information, and a per-step time penalty $\delta = 1$. The balance is deliberate: the attacker must explore enough to understand the network, but never linger, since an efficient strategy is a fast one.

**Table 3** Reward and threshold constants.

| Symbol | Value | Role |
|---|---|---|
| $c(a_t)$ | 0.2 | Flat cost per defensive action |
| $\tau_{\text{alert}}$ | $[0.3, 0.5]$ | Threshold: observe → deploy honeypot |
| $\tau_{\text{crit}}$ | $[0.6, 1.0]$ | Threshold: trigger MTD mutation |
| $\alpha$ | 50 | Terminal reward (attacker) |
| $\beta$ | 5 | Progression bonus (attacker) |
| $\delta$ | 1 | Per-step time penalty (attacker) |

## 4.3 Defender Model: Observations, Actions, and Reward

Observing from the controller, the defender enjoys a far broader view, captured in Eq. (9):

$$s_{\text{def}} = \big(f_{\text{flow}},\, m_{\text{hp}},\, m_{\text{mtd}},\, a_{\text{alert}},\, c_{\text{res}},\, h_{\text{def}}\big)^{\top}, \tag{9}$$

detailed in the lower half of Table 2. The component $c_{\text{res}}$ matters especially, since it lets the agent factor the cost of its own actions into its decisions. Its action space, shown in Eq. (10), is deliberately tight:

$$\mathcal{A}_{\text{def}} = \big\{\text{honeypot}(i),\ \text{mtd}(i),\ \text{observe}\big\}, \tag{10}$$

where *observe* is more than a null action: it lets the defender gather evidence before committing resources, and it is through this action that the agent learns not to overreact. The corresponding reward mirrors the attacker's objective,

$$R_{\text{def}} = -20 \cdot \mathbf{1}[\text{attack succeeded}] + 1 \cdot \mathbf{1}[\text{attack not succeeded}] - c(a_t). \tag{11}$$

Numerically, we set $c(a_t) = 0.2$ for every defensive action, a flat cost independent of the specific lever chosen. The graduated response of Algorithm 1 is governed by two thresholds on the per-node alert level $a_{\text{alert}} \in [0, 1]$: $\tau_{\text{alert}} \in [0.3, 0.5]$, below which the defender only observes, and $\tau_{\text{crit}} \in [0.6, 1.0]$, beyond which it additionally triggers an MTD mutation. A systematic sensitivity sweep across these ranges is left for future work. Table 3 collects these constants for reference.

The three terms are calibrated jointly rather than chosen arbitrarily. The penalty $-20$ far

exceeds the accumulated +1 increments, so a single breach cancels dozens of successful steps and prevention stays the absolute priority; yet it remains below the attacker's $\alpha = 50$, which preserves the near symmetry of the game. The +1 granted per step for as long as the attacker has not reached its goal amounts to maximizing the time to compromise (TTC). Finally, $c(a_t)$ is what distinguishes our formulation from approaches that optimize effectiveness alone: a policy that kept decoys permanently deployed would score well and deploy badly, and this term penalizes it accordingly, in the spirit of the cost-aware reward proposed for 5G slicing [37]. The residual departure from zero sum stems from that same term and from the asymmetric terminal rewards, an asymmetry that long episodes compensate through the defender's incremental gains.

## 4.4 Rationale for Symmetric PPO-Based Self-Play

We use proximal policy optimization (PPO) for both agents rather than pairing, say, a DQN attacker with a PPO defender, for four converging reasons. First, stability: in a multi-agent game the opponent changes at every episode, and PPO bounds the magnitude of its updates, avoiding the policy collapses that DQN exhibits in such settings [39]. Second, scale: as a policy-gradient method it absorbs the large discrete space of Eq. (7) without the Q-value overestimation of value-based methods. Third, consistency: PPO is on-policy, which suits co-evolving policies, whereas off-policy replay buffers grow stale as soon as the opponent shifts. Fourth, variance: the actor–critic structure estimates the advantage $\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, which accelerates convergence. This choice aligns with comparative studies that rank PPO among the reference algorithms for deceptive resource deployment [7].

Each agent maximizes the clipped surrogate objective stated in Eq. (12):

$$L^{\mathrm{PPO}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(\rho_t \hat{A}_t,\ \mathrm{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big], \tag{12}$$

with the importance ratio $\rho_t$ defined as in Eq. (13):

$$\rho_t = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}, \tag{13}$$

in which clipping the importance ratio prevents any action from becoming abruptly much more or much less probable. Advantages follow the generalized estimator $\hat{A}_t = \sum_{l\geq 0}(\gamma\lambda)^l \delta_{t+l}$ with $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, and the objective actually optimized adds a critic term and an entropy bonus,

$$L^{\mathrm{total}} = L^{\mathrm{PPO}} - c_1 L^V + c_2\, H\big[\pi_\theta(\cdot \mid s_t)\big]. \tag{14}$$

This entropy term plays a decisive role in preventing premature convergence. Without it each agent would converge to a pure policy exploiting its opponent's current weakness, producing policy cycling, a rock–paper–scissors effect, instead of co-evolution. We therefore set a higher bonus for the attacker, $c_2 = 0.02$, to diversify its attack vectors, and a moderate one for the defender, $c_2 = 0.01$, to preserve network stability.

Training proceeds by self-play, as summarized in Algorithm 1. On top of PPO sampling, a graduated response keyed on alert thresholds governs the active levers: below $\tau_{\mathrm{alert}}$ the defender merely observes; between the two thresholds it deploys a honeypot; beyond $\tau_{\mathrm{crit}}$ it additionally triggers an MTD mutation. Embedding these thresholds in the learning loop prevents costly resources from being spent on background noise.

## 4.5 Implementation and Metrics

The environment is emulated with Mininet, controlled by Ryu over OpenFlow, and exposed to the agents through a Gymnasium interface; PPO comes from Stable-Baselines3 and every run is logged with MLflow for reproducibility. Table 4 lists the hyperparameters. Most are shared, but two differ by design: clipping is tighter for the defender, whose actions have persistent effects on the network, and the entropy bonus is higher for the attacker, to sustain strategic diversity. A pool of the five latest versions of each policy is sampled at random every episode, which keeps each agent robust to a variety of opponents instead of overfitting the most recent one.

Evaluation follows established deception and MTD protocols [32]. The primary metric is the TTC, the time the attacker needs to complete an exfiltration; higher is better. We complement it with the defensive success rate, the share of

**Algorithm 1** Adversarial self-play training loop

Initialize $\pi_\theta^{\text{atk}}$, $\pi_\phi^{\text{def}}$ and opponent pools $\mathcal{P}_{\text{atk}}, \mathcal{P}_{\text{def}}$ of size 5
**for** episode $= 1$ to $E$ **do**
  Sample $\bar{\pi}^{\text{atk}} \sim \mathcal{P}_{\text{atk}}$, $\bar{\pi}^{\text{def}} \sim \mathcal{P}_{\text{def}}$
  Reset the emulated topology; clear $T_{\text{MTD}}$
  **while** episode not terminated **do**
    Attacker samples $a_{\text{atk}} \sim \pi_\theta^{\text{atk}}(\cdot \mid s_{\text{atk}})$
    Controller collects flow statistics; updates $a_{\text{alert}}$
    Defender samples $a_{\text{def}} \sim \pi_\phi^{\text{def}}(\cdot \mid s_{\text{def}})$
    **if** $a_{\text{alert}}(i) \geq \tau_{\text{crit}}$ **then**
      `Flow-Mod`: mutate node $i$; update $T_{\text{MTD}}$
    **else if** $a_{\text{alert}}(i) \geq \tau_{\text{alert}}$ **then**
      Deploy honeypot on node $i$; redirect suspicious flows
    **end if**
    Collect $R_{\text{atk}}$ and $R_{\text{def}}$ from Eqs. (8) and (11)
  **end while**
  Update $\theta$ and $\phi$ by maximizing Eq. (14)
  Push current policies into the pools (FIFO)
**end for**

**Table 4** PPO hyperparameters.

| Parameter | Attacker | Defender |
|---|---|---|
| $\gamma$ (discount) | 0.99 | 0.99 |
| $\lambda_{\text{GAE}}$ | 0.95 | 0.95 |
| $\epsilon$ (clipping) | 0.20 | 0.10 |
| Mini-epochs $K$ | 10 | 10 |
| Batch size | 256 | 256 |
| Learning rate | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ |
| Entropy bonus $c_2$ | 0.02 | 0.01 |
| Opponent pool size | 5 | 5 |

episodes in which the attacker fails; the deception rate, counting honeypot captures and MTD invalidations; the overhead, combining consumed resources and induced latency; and the cumulative rewards, which expose the learning dynamics. Latency is subject to an operational ceiling, since edge applications must stay responsive.

It is also worth asking what the loop costs once training is over, since a policy that only runs offline would be of limited use. Per decision step the controller aggregates statistics over $N_{\text{sw}}$ switches, which is linear in the number of switches; the actor performs a single forward pass, whose cost is fixed by the network width and the $|\mathcal{A}_{\text{def}}|$ output logits and is therefore independent of the episode length; and enforcing the chosen action amounts to one `Flow-Mod` message plus one entry in $T_{\text{MTD}}$, both constant-time. The loop is thus linear in the size of the data plane and constant in everything else, which is what allows the latency reported in Section 5 to stay flat as episodes lengthen. The quadratic growth of Eq. (7) concerns the attacker's action space and the training budget, not the defender's runtime.

One theoretical caveat is worth stating. For finite zero-sum two-agent games, gradient-ascent policy sequences converge on average to a mixed Nash equilibrium. Our game is not strictly zero-sum because of $c(a_t)$, so the guarantee does not transfer literally; empirically, however, PPO self-play converges to stable policies, both in the literature [30] and in our runs.

# 5 Experimental Evaluation

## 5.1 Scenarios and Protocol

Two complementary scenarios were retained, differing by target and by the severity of a breach.

### 5.1.1 Scenario Description: Intrusion into the IoT Subnetwork

The bot targets the sensors, whose weak protection makes them convenient entry points, and aims to exfiltrate data by climbing toward the edge servers. The cycle of Fig. 2 chains four stages: reconnaissance, where each positive probe enriches the state and earns the progression bonus; target selection through the PPO actor, each fruitless step being charged in time; exploitation of the chosen node; and then one of two outcomes, either the controller detects the intrusion and raises an alert that activates the defender, or the attack passes unnoticed and exfiltration cashes in the terminal reward.

Once alerted, the defender may fire mtd($i$), mutating either the address or the listening port of the targeted node. By Eq. (1) the edge server retains the node's true identity, so legitimate traffic is untouched; the attacker, by contrast, sees its

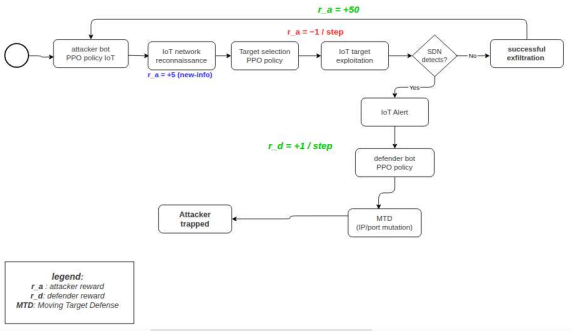



**Fig. 2** Workflow of scenario 1 (IoT attack); $r_a$ and $r_d$ denote the attacker and defender rewards.

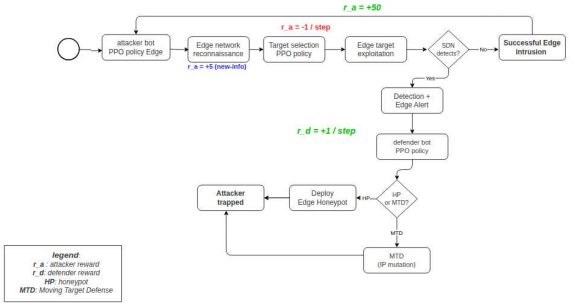



**Fig. 3** Workflow of scenario 2 (edge attack). The defender combines honeypot deployment and MTD mutation depending on the alert level.

patiently built map turn stale and must start over, which mechanically lengthens the TTC.

### 5.1.2 Scenario Description: Intrusion into the Edge Servers

The second scenario (Fig. 3) aims higher. The attacker enters through an access point, reaches the segment leading to the edge servers, and seeks the IoT databases they host. A breach is correspondingly costlier, since it exposes at once the aggregated data of every object attached to the compromised server.

The offensive sequence is analogous, but the defensive repertoire is richer. Against a detected edge intrusion the controller may install a fake edge server, identical in ports and appearance but backed by a fictitious database, so that the attacker reveals its techniques without ever touching real data; or, when the alert crosses $\tau_{\text{crit}}$, mutate the server itself; or, in the last resort, sink the attacker's traffic silently. Here too, after any mutation only the attacker is left with a stale map.

### 5.1.3 Training and Evaluation Protocol

Both agents were trained over 1000 episodes per scenario under a three-phase protocol that serves as the comparative frame throughout. *Phase 1* removes all defense and measures the attacker's raw speed. *Phase 2* introduces a static defense reproducing the dominant non-adaptive paradigm. *Phase 3* activates our PPO-driven dynamic defense. Comparing the three isolates the specific contribution of online adaptation.

The static baseline is specified precisely so that a third party can reproduce it: at each `reset()`, $\lfloor N_E/2 \rfloor$ honeypots, that is roughly half the edge servers, are placed at random. The draw is renewed every episode but never depends on the attacker's ongoing behavior, and no mutation or reassignment occurs once the episode has begun.

Reproducibility was treated as a constraint rather than an afterthought. Every run is logged with MLflow, the hyperparameters of Table 4 are fixed across scenarios except where the table states otherwise, and each configuration is repeated over ten independent runs differing only in random seed; this is what the confidence intervals reported in Section 5.4 are based on. The emulated topology, 32 nodes distributed over switches, edge servers, access points, and IoT devices, is regenerated identically at every `reset()`, so that differences between phases cannot be attributed to topology drift.

## 5.2 Results for the IoT Subnetwork Attack Scenario

### 5.2.1 Time to compromise

Fig. 4 shows three distinct patterns. Without defense the TTC decays steadily, from about twenty seconds down to ten, as the attacker memorizes an immobile topology and prunes its exploitation paths. Static defense holds a higher plateau near twenty seconds, but it erodes too, because a regular mutation schedule is itself a pattern: the attacker learns to wait for the vulnerability windows. Dynamic defense reverses this trend: the TTC exceeds fifty seconds. The reason is co-evolution: whenever the attacker finds an exploitable regularity, the defender reads it in

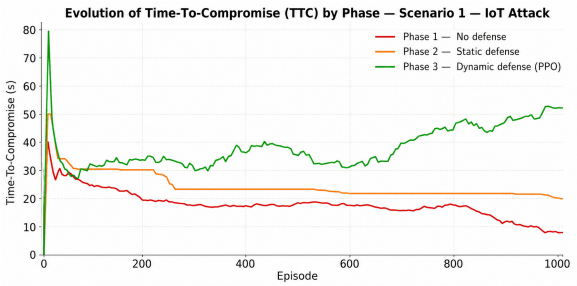



**Fig. 4** Time to compromise per phase, scenario 1.

the OpenFlow statistics and retunes the frequency and target of its mutations, so each invalidation forces a full reconnaissance restart rather than letting the TTC settle on a floor.

### 5.2.2 Reward Dynamics

Fig. 5 confirms this from the reward side. Undefended, the attacker's cumulative reward climbs well above the neutral line, the signature of repeated exfiltrations. Under static defense it stays positive but modest, since fixed rules block some attempts while their regularity leaves a nonzero success rate. Under dynamic defense it collapses toward zero and turns negative late in training, and Eq. (8) explains why: when mutations multiply fruitless reconnaissance cycles, the accumulated $\delta$ penalties eventually outweigh the terminal reward, so most episodes end at a loss. The defender's curve rises symmetrically, reflecting episodes increasingly won, as the agent learns to fire its mutations at the right moment, namely as soon as the alert level betrays scanning, before exploitation is even attempted.

### 5.2.3 Defensive success rate

The crossing visible in Fig. 6 is among the clearest results of the study. Static defense starts respectably near 0.4, then collapses toward zero, a direct consequence of its own stationarity: since its rules never change, the PPO attacker ends up encoding their invariants, the fixed mutation period and the predictable decoy positions, as exploitable information rather than as obstacles. Dynamic defense follows the opposite trajectory and stabilizes close to 0.98. Two effects compound there. On one hand $c_{\text{res}}$ lets the defender modulate mutation intensity against available resources, avoiding a costly overreaction and so sustaining continuous coverage. On the other, the pool of five past opponents forces robustness across a range of offensive behaviors instead of a response tuned to one tactic. The crossing point marks the moment where adaptivity definitively outweighs rigidity.

### 5.2.4 Deception Pressure and Its Operational Cost

Fig. 7 pairs the benefit with its cost. The count of MTD invalidations naturally stays at zero without defense, since no mutation is active and the attacker enjoys an uninterrupted map. Under static defense it peaks near eight per episode around episode 50, then decays back toward zero, which is the same structural limitation seen again: a predictable schedule is progressively anticipated, so fewer probes land on an invalidated target. Under dynamic defense it settles around twenty per episode. Crucially, this is not blind overtriggering; because $c(a_t)$ charges every action, the agent has learned to mutate only once the alert crosses $\tau_{\text{alert}}$. The resulting pressure denies the attacker the temporal stability it needs, forcing it to spend a growing share of its actions on reconnaissance.

The price appears on the right of the same figure: latency grows with reconfiguration intensity and reaches about 45 ms at the end of training. This follows from the nature of `Flow-Mod` messages, since each mutation adds a rule and switch processing time grows with the rule count. Resource consumption, by contrast, stays moderate and comparable to the static baseline, because MTD touches only the nodes actually under alert rather than the whole network.

### 5.2.5 Shift in Attacker Behavior

Fig. 8 makes the mechanism visible behaviorally. Undefended, the bot devotes 74 percent of its actions to exploitation against 24 percent to scanning: having mapped an immobile network, it goes straight for known targets. Static defense shifts this only slightly, to 55 against 40 percent. Under dynamic defense the hierarchy is inverted, with scanning at 60 percent and exploitation at 37 percent. That reversal is the behavioral counterpart of the TTC curve: each unpredictable mutation invalidates the map and reallocates effort from exploitation to reconnaissance. Pivot and wait

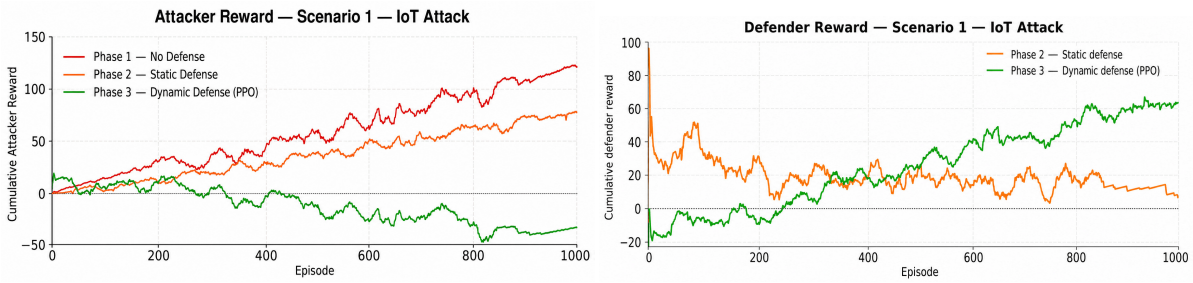



**Fig. 5** Cumulative rewards in scenario 1: attacker (left) and defender (right).

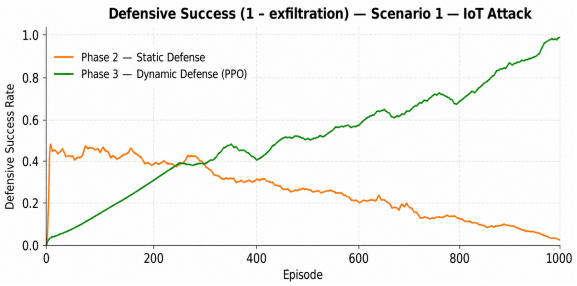



**Fig. 6** Defensive success rate, scenario 1.

remain marginal throughout, between 2 and 5 percent.

## 5.3 Results for the Edge Server Attack Scenario

### 5.3.1 Time to compromise

The harder scenario yields a sharper contrast (Fig. 9). Both without defense and under static defense the TTC settles at barely ten seconds, because an edge server concentrates the data of all its attached objects and is therefore worth aiming at directly, without the intermediate pivots that scenario 1 required. Static defense fails for the familiar reason, its invariant rules exposing usable windows between mutation cycles. Dynamic defense, conversely, drives the TTC to a steady 65–80 seconds. The gap comes from combining both levers: honeypots absorb the first attempts and expose the attacker's techniques, while mutations invalidate whatever map is rebuilt between captures. This combined pressure restarts the reconnaissance cycle more often than MTD alone did on IoT nodes, which accounts for the higher TTC observed here despite the target being intrinsically harder to protect.

### 5.3.2 Reward Dynamics

Fig. 10 shows the attacker's reward growing without defense, growing more slowly under static defense, and hovering near equilibrium under dynamic defense. That it stalls at equilibrium rather than plunging reflects the difficulty of the target: the edge attacker still succeeds occasionally, notably during transitions between two defensive reconfigurations, and those punctual wins offset the many losing episodes.

The defender's side is more instructive still. With no defense there is no reward signal at all. Under static defense the cumulative reward starts modestly and then turns negative, and $c(a_t)$ explains the sign change: honeypots and rules are deployed unconditionally, irrespective of the actual alert level, so resources drain without a proportional defensive effect and the policy becomes counterproductive as training proceeds. Under dynamic defense the reward stabilizes at a clearly higher level, because the agent engages its levers only when the controller's signals warrant it, sustaining the defensive effect while keeping cost in check.

### 5.3.3 Defensive success rate

Fig. 11 confirms the collapse of the static baseline, whose rate falls from about 0.4 to zero within two to three hundred episodes and never recovers. The speed of that decline follows from the target's nature: unlike the many IoT nodes, an edge server is a unique, stable target whose position never varies, so a defense that does not move it unpredictably is learned within a few dozen episodes and simply bypassed. Dynamic defense instead climbs to an oscillating band between 0.6 and 1.0, centered around 0.7 to 0.8 with peaks at 0.97. The greater variance is not a learning failure

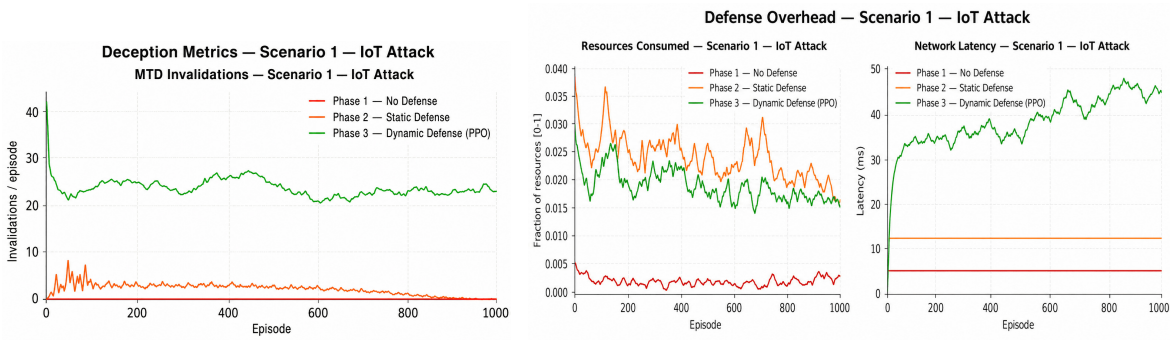



**Fig. 7** Scenario 1: MTD invalidations per episode (left) and defense overhead, consumed resources and latency (right).

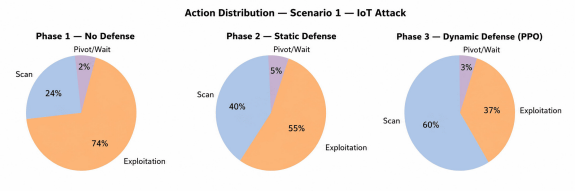



**Fig. 8** Distribution of attacker actions per phase, scenario 1.

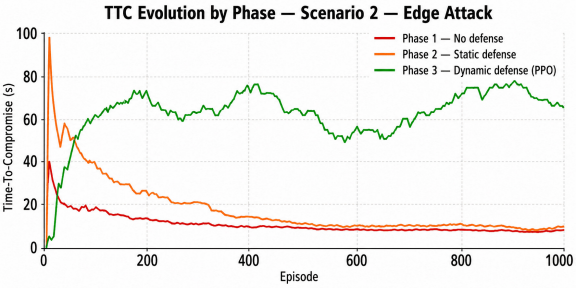



**Fig. 9** Time to compromise per phase, scenario 2.

but the visible trace of self-play: as the defender improves, the attacker strings together failures, then adapts in turn, briefly depressing the rate before the defender readjusts. A residual exposure window also remains structurally, since an edge server cannot be hidden entirely behind a decoy without interrupting legitimate service.

### 5.3.4 Deception Pressure and Its Operational Cost

In Fig. 12, honeypot captures and MTD invalidations both ramp up in phase 3, and the way the defender doses them reveals an emergent strategy: a moderate alert calls for a honeypot, which absorbs the attacker without disclosing the server's real address, whereas a critical alert adds a mutation that invalidates whatever map was rebuilt after the decoy was identified. This complementarity is precisely what the honeypot-only static baseline cannot reproduce, since a decoy whose position has been memorized is thereafter avoided.

The cost is correspondingly higher, with the resource fraction of phase 3 around 0.095, since each honeypot mobilizes a fake server instance while each mutation adds a `Flow-Mod` rule. Latency nevertheless oscillates between 40 and 45 ms, marginally below scenario 1, because honeypots intercept part of the exploitation attempts and thereby reduce the number of mutations needed per episode. Staying under the operational ceiling despite this intensity is what makes the approach deployable.

### 5.3.5 Shift in Attacker Behavior

Fig. 13 reaches the same conclusion with a wider initial amplitude. Undefended, the bot spends 79 percent of its actions on exploitation against 19 percent on scanning, a single fixed target being memorized within a few episodes. Static defense barely disturbs that ratio, 60 against 36 percent, which is consistent with the faster collapse of its success rate. Dynamic defense forces the reallocation once more, to 58 percent scanning against 39 percent exploitation, since every capture or invalidation returns the attacker to reconnaissance. Pivot and wait again stay between 2 and 4 percent.

### 5.3.6 Ablation: Isolating Honeypot and MTD Contributions

Scenario 2 combines both levers, and Section 5.3.4 describes their interaction qualitatively, but

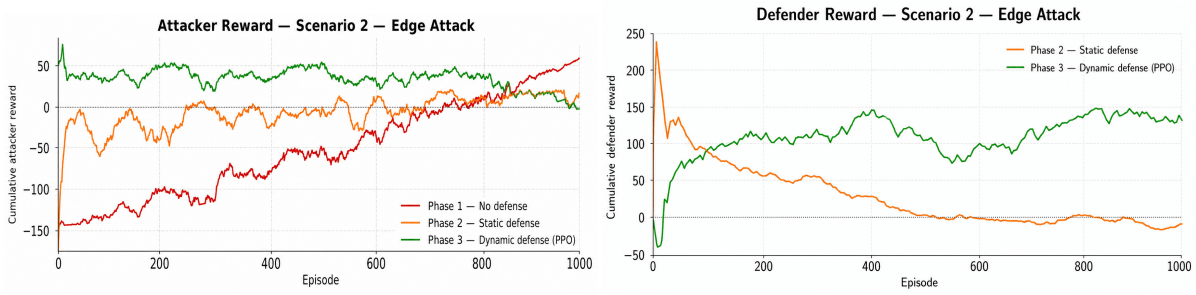



**Fig. 10** Cumulative rewards in scenario 2: attacker (left) and defender (right).

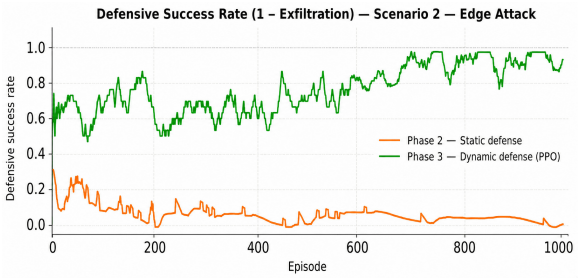



**Fig. 11** Defensive success rate, scenario 2.

**Table 5** Dynamic defense across the two scenarios, with 95% confidence intervals over ten independent runs.

| Metric (dynamic defense) | Scenario 1 (IoT) | Scenario 2 (Edge) |
|---|---|---|
| Mean TTC (s) | $49.3 \pm 4.1$ | $70.5 \pm 8.7$ |
| Defensive success rate | $0.98 \pm 0.02$ | $0.97 \pm 0.02$ |
| Dominant lever | MTD | Honeypot + MTD |
| Consumed resources | $\sim$0.03 | $\sim$0.095 |
| Latency | $\sim$45 ms | $\sim$40–45 ms |

Table 5 does not quantify the marginal contribution of either mechanism alone. A dedicated ablation, evaluating or retraining the defender with only honeypots or only MTD available, is left for future work and would sharpen the claim that combining both levers yields more than the sum of the two.

## 5.4 Cross-Scenario Comparison

Table 5 places the two scenarios side by side. Dynamic defense dominates both baselines in either case, but the shape of the gain differs with the target. On the IoT subnetwork the success rate is high and stable at a moderate resource cost, because MTD alone usually suffices to disorient an attacker aiming at weakly protected nodes. On the edge servers the central tendency sits lower and the variance is markedly wider, for the structural reason already given, and the resource cost roughly triples since both levers must run together. In both cases, however, latency remains under the ceiling, which is the result that matters for deployability: adaptivity is not bought at the price of service degradation.

The intervals were obtained from ten independent training runs per scenario, the steady-state value of a metric being its mean over the last 200 episodes and the interval following Student's law with nine degrees of freedom. They quantify the stability difference just described. The phase comparison, for its part, rests on gaps wide enough to survive that variance, since the phase-3 TTC exceeds the phase-2 TTC at every episode beyond 200 in Figs. 4 and 9.

To complement the interval-overlap criterion above, a paired statistical test on a metric less sensitive to episode-to-episode noise than raw TTC, e.g., a Wilcoxon signed-rank test on the per-seed steady-state TTC (last 200 episodes, $n = 10$ seeds) between Phase 2 and Phase 3, would formally confirm that the observed gap exceeds what run-to-run variance alone could produce. We report this test as future work, contingent on completing the full ten-seed protocol described in Section 5.1.3 for both scenarios.

## 5.5 Generalization Beyond the Self-Play Pool

The evaluation above measures the defender against opponents drawn from its own self-play pool, which raises the standard question of overfitting to co-evolved adversaries. A rigorous answer requires evaluating the trained defender against

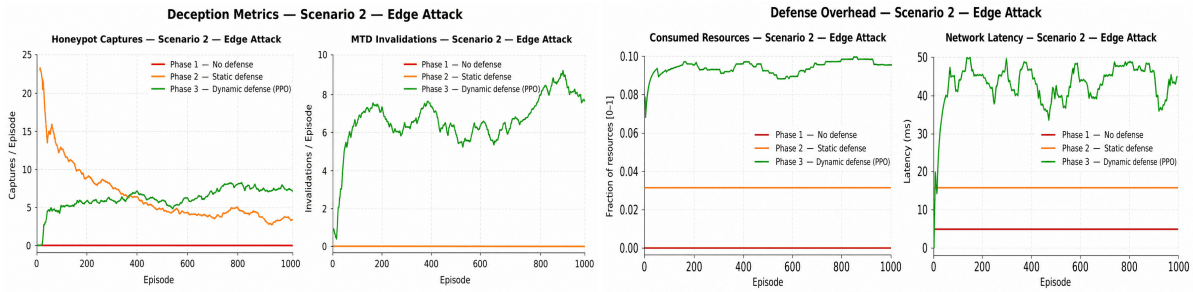



**Fig. 12** Scenario 2: honeypot captures and MTD invalidations (left); defense overhead, consumed resources and latency (right).

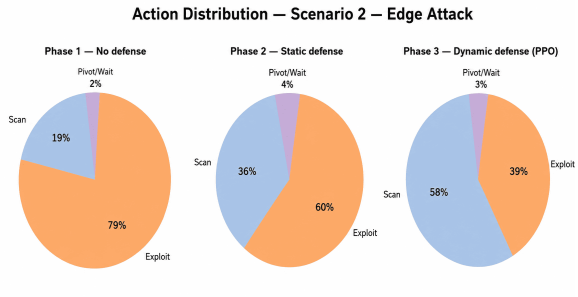



**Fig. 13** Distribution of attacker actions per phase, scenario 2.

an attacker that never participated in its training loop, for instance a policy frozen at the end of single-agent training (no co-evolution), or a scripted attacker following a fixed strategy. We did not run this hold-out evaluation for the present study and leave it, together with the ablation of Section 5.3.6 and the formal significance test discussed above, as a direct extension of this work. We expect the defensive success rate to degrade against a genuinely out-of-distribution attacker, since the graduated response of Algorithm 1 was tuned against the alert patterns produced by co-evolved opponents; quantifying that degradation is necessary before the deployment recommendations of Section 6.3 can be taken as more than indicative.

# 6 Discussion

## 6.1 Positioning Relative to the State of the Art

Table 6 arranges a representative sample of the literature along two axes, the algorithms each side runs and the results reported. The reading is unambiguous: none of these works combines co-evolving agents, SDN orchestration, and an edge/IoT context. Because the underlying topologies, threat models, and detection assumptions differ substantially across these studies (a virtual enterprise network with real CVEs [40], generic topologies [5], and SDN-only or 5G-slicing deployments [28, 37]), the reported metrics are not directly comparable in a controlled sense. Table 6 should therefore be read as indicative of relative trends rather than as a strict, head-to-head quantitative benchmark.

Three families come close without covering the whole. Symmetric-agent and multi-agent approaches, whether PPO self-play [30], multi-agent Markov games [5], or the framework of Charpentier *et al.* [40] that trains an attacker and then a PPO defender on virtual machines carrying real vulnerability entries, integrate neither SDN orchestration nor an edge/IoT context, and their sequential training again amounts to facing an immobile target. Approaches that do integrate SDN, and sometimes the edge [28, 37], confront the defender only with a scripted adversary, which overstates defensive effectiveness the moment an adaptive bot appears. Bringing the three dimensions together is, to our knowledge, what remained to be done.

## 6.2 Revisiting the Four Research Gaps

*G1* falls by construction, since our defender applies no predefined rule but learns its policy, and the results bear this out: the static baseline collapses in both scenarios where the dynamic one improves. *G2* is addressed by self-play, our defender facing a PPO attacker that is itself evolving; the sustained mutation pressure and the

**Table 6** Algorithms, environments, and key results against the state of the art. A dash (–) marks a metric the corresponding work does not report. Metrics are not directly comparable across rows due to differing topologies, threat models, and detection assumptions (see Section 6); the table indicates relative trends only.

| Reference | Attacker algo. | Defender algo. | TTC | Defensive rate | Latency | Environment |
|---|---|---|---|---|---|---|
| Hammar and Stadler [30] | PPO | PPO | ∼30 s | ∼80% | – | Self-play, simple topologies |
| Yao *et al.* [5] | DDQN | DDQN | ∼25 s | ∼72% | – | Generic topology |
| Charpentier *et al.* [40] | PPO | PPO | – | 97% | – | Virtual enterprise network (VMs, CVEs) |
| Han *et al.* [28] | – | DDQN | – | – | <80 ms | SDN (OpenFlow) |
| Alnfiai [37] | – | DQN | – | ∼95% | <70 ms | 5G networks (slicing) |
| **This work** | **PPO** | **PPO** | **over 50 s (IoT); 65–80 s (edge)** | **approx. 98% (IoT); 97% (edge)** | **below 50 ms** | **SDN/edge/IoT** |

observed reallocation of attacker actions show that both agents genuinely co-evolve. *G3* is met by writing $c(a_t)$ into the reward and $c_{\text{res}}$ into the state, and the overhead measurements show the concern is not merely formalized but effectively controlled. *G4* is what the three-phase protocol fills, since confronting the defender with an RL attacker of realistic complexity in an instrumented environment measures robustness far more representatively than a scripted opponent would.

## 6.3 Deployment Considerations

The measurements above say what the policy achieves; they say less about how it would be operated, and three points deserve to be made explicit.

First, the trained defender plugs in as a controller application rather than as a replacement for one: it consumes the statistics Ryu already exports and emits `Flow-Mod` messages the controller already knows how to install, so no change to the data plane is required. Second, because the policy is learned against a specific topology and a specific alert model, it should be warm-started from the emulated environment and then refreshed as the deployed network evolves, rather than trained once and frozen; the self-play machinery of Algorithm 1 is precisely what makes such periodic refreshes cheap, since the opponent pool can be repopulated from recent traffic traces. Third, a fail-safe path matters more here than in a purely detective system: since every deceptive action rewrites forwarding state, the controller must be able to flush $T_{\text{MTD}}$ and revert to plain forwarding on demand, which the transparency invariant of Eq. (1) makes straightforward because the real addresses are never discarded.

## 6.4 Limitations

Three limits should temper these conclusions.

*Emulation is not deployment.* Everything was run on Mininet and Ryu, which buys experimental control and reproducibility at the price of abstracting away hardware latency, throughput variability, unforeseen failures, and radio interference on wireless devices. The reported figures are therefore an upper bound on operational performance.

*Scale.* Our topology comprises 32 nodes, which keeps training times reasonable, but Eq. (7) grows quadratically, so single-agent RL will not transfer as such to networks of several hundred or several thousand nodes.

*Reward calibration.* The constants of Eqs. (8) and (11) were chosen to satisfy the ordering argued in Section 4.3 rather than tuned by search, and we did not measure how far the results survive a change of that ordering. A systematic sensitivity study, in particular on the ratio between the breach penalty and the per-step gain, would

sharpen the claim that the observed behavior follows from the structure of the reward rather than from one fortunate parameterization.

*Detection is modeled, not learned.* Our formulation presumes an SDN detector that raises alerts with some reliability, whereas real detectors produce false positives and negatives. Concretely, $a_{\text{alert}}$ is not produced by a trained or signature-based module: it comes from a Bernoulli draw whose probability depends only on the attacker's action type, namely scanning ($p = 0.10$), IoT exploitation ($p = 0.45$), edge exploitation ($p = 0.65$), and pivoting ($p = 0.30$), independently of $f_{\text{flow}}$. When it fires, the node's alert level is incremented and then decays absent new detections. The statistics $f_{\text{flow}}$ are computed separately, with noise correlated to the action type, and thus form an observable but noisy signal distinct from detection proper. This deliberate simplification decouples detection, a field in its own right, from the decision layer we contribute, consistently with prior work that likewise consumes aggregated controller statistics without training a dedicated detector [26–28].

# 7 Conclusion

This paper proposed a dynamic deception framework for SDN-orchestrated edge environments facing autonomous RL bots. We described a three-plane architecture in which the Ryu controller hosts the defender and orchestrates honeypots, MTD, and isolation transparently for legitimate traffic; we cast the confrontation as a two-agent stochastic game decomposed into two coupled Markov decision processes; and we justified symmetric PPO with adversarial self-play as the learning scheme.

Across two scenarios, one aimed at IoT devices and the other at edge servers, the conclusion is consistent: dynamic defense outperforms both the absence of defense and the static baseline. It lengthens the time to compromise durably, raises the defensive success rate to about 0.98 and 0.97 respectively, and pushes the attacker's reward toward zero, all while keeping latency below the 50 ms operational threshold. Against the state of the art, the framework appears to be the first to combine symmetric PPO agents, SDN orchestration, and an edge/IoT context, thereby answering the four gaps identified in Section 2.

Three directions follow naturally. Validation on a physical testbed would confront the learned policies with the energy variability, real delays, and hardware failures that emulation abstracts away. Scaling up calls for hierarchical policies, one selecting the subnetwork to defend and another the precise mechanism, or for cooperative multi-agent defenders. Finally, feeding cyber threat intelligence into the defender's observation would sharpen its decisions against emerging attack patterns.

# Declarations

**Funding:** The authors received no specific funding for this work.

# Appendix A Notation

Table A1 collects the symbols used throughout the paper, in the order in which they are introduced.

# References

[1] Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun. Surveys Tuts. **19**(3), 1628–1656 (2017) https://doi.org/10.1109/COMST.2017.2682318 . 3rd Quart.

[2] Hussain, M., Shah, N., Amin, R., Alshamrani, S.S., Alotaibi, A., Raza, S.M.: Software-defined networking: Categories, analysis, and future directions. Sensors **22**(15), 5551 (2022) https://doi.org/10.3390/s22155551

[3] Abdi, A.H., Audah, L., Salh, A., Alhartomi, M.A., Rasheed, H., Ahmed, S., Tahir, A.: Security control and data planes of SDN: A comprehensive review of traditional, AI and MTD approaches to security solutions. IEEE Access **12**, 69941–69980 (2024) https://doi.org/10.1109/ACCESS.2024.3393548

[4] Beltrán López, P., Gil Pérez, M., Nespoli, P.: Cyber deception: State of the art, trends, and open challenges. arXiv:2409.07194 (2024). https://doi.org/10.48550/arXiv.2409.07194

**Table A1** Summary of notation.

| Symbol | Meaning |
|---|---|
| $T_{\mathrm{MTD}}$ | Controller table mapping fictitious to real addresses |
| $\mathcal{G}$ | Two-agent stochastic game |
| $\mathcal{S}$ | State space of the system |
| $T$ | Transition kernel of the game |
| $\gamma$ | Discount factor |
| $N$ | Number of nodes in the topology |
| $N_{\mathrm{sw}}$, $N_E$, $N_I$ | Numbers of switches, edge servers, IoT nodes |
| $s_{\mathrm{atk}}$, $s_{\mathrm{def}}$ | Attacker and defender state vectors |
| $\mathcal{A}_{\mathrm{atk}}$, $\mathcal{A}_{\mathrm{def}}$ | Attacker and defender action spaces |
| $R_{\mathrm{atk}}$, $R_{\mathrm{def}}$ | Attacker and defender reward functions |
| $\alpha$, $\beta$, $\delta$ | Terminal reward, progression bonus, time penalty |
| $c(a_t)$ | Operational cost charged to a defensive action |
| $c_{\mathrm{res}}$ | Fraction of edge resources consumed by decoys |
| $a_{\mathrm{alert}}$ | Per-node alert level reported by the controller |
| $\tau_{\mathrm{alert}}$, $\tau_{\mathrm{crit}}$ | Intermediate and critical alert thresholds |
| $w$ | Length of the sliding action-history window |
| $\pi_\theta$, $\pi_\phi$ | Attacker and defender policies |
| $\mathcal{P}_{\mathrm{atk}}$, $\mathcal{P}_{\mathrm{def}}$ | Self-play opponent pools |
| $\rho_t$ | Importance ratio of the PPO update |
| $\epsilon$ | Clipping parameter |
| $\hat{A}_t$ | Estimated advantage |
| $\lambda$ | Bias–variance parameter of the advantage estimator |
| $V(\cdot)$ | State-value function estimated by the critic |
| $H[\cdot]$ | Policy entropy |
| $c_1$, $c_2$ | Critic and entropy coefficients |


[5] Yao, Q., Wang, Y., Xiong, X., Wang, P., Li, Y.: Adversarial decision-making for moving target defense: A multi-agent Markov game and reinforcement learning approach. Entropy **25**(4), 605 (2023) https://doi.org/10.3390/e25040605

[6] Ahmed, A.M., Nguyen, T.T., Abdelrazek, M., Aryal, S.: Reinforcement learning-based autonomous attacker to uncover computer network vulnerabilities. Neural Comput. Appl. **36**, 14341–14360 (2024) https://doi.org/10.1007/s00521-024-09668-0

[7] Li, C., Zhao, N., Wu, H.: Multiple deception resources deployment strategy based on reinforcement learning for network threat mitigation. Sci. Rep. **15** (2025). Art. no. 16830

[8] Alkasassbeh, M., Khalil, A., Almseidin, M.: Self-adaptive moving target defense with cyberdeception for proactive defense of IoT networks. In: Proc. Int. Conf. New Trends Comput. Sci. (ICTCS), pp. 1–6 (2025). https://doi.org/10.1109/ICTCS65341.2025.10989318

[9] Provos, N.: A virtual honeypot framework. In: Proc. 13th USENIX Secur. Symp., San Diego, CA, USA, pp. 1–14 (2004)

[10] Ilg, N., Duplys, P., Sisejkovic, D., Menth, M.: A survey of contemporary open-source honeypots, frameworks, and tools. J. Netw. Comput. Appl. **220** (2023) https://doi.org/10.1016/j.jnca.2023.103737 . Art. no. 103737

[11] Cai, G.-L., Wang, B.-S., Hu, W., Wang, T.-Z.: Moving target defense: State of the art and characteristics. Frontiers Inf. Technol. Electron. Eng. **17**(11), 1122–1153 (2016) https://doi.org/10.1631/FITEE.1601321

[12] Al-Shaer, E., Duan, Q., Jafarian, J.H.: Random host mutation for moving target defense. In: Security and Privacy in Communication Networks. LNICST, vol. 106, pp. 310–327. Springer, Berlin, Germany (2012). https://doi.org/10.1007/978-3-642-36883-7_19

[13] Chang, S.-Y., Park, Y., Babu, B.B.A.: Fast IP hopping randomization to secure hop-by-hop access in SDN. IEEE Trans. Netw. Service Manage. **16**(1), 308–320 (2019) https://doi.org/10.1109/TNSM.2018.2889842

[14] Yackoski, J., Bullen, H., Yu, X., Li, J.: Applying self-shielding dynamics to the network architecture. In: Moving Target Defense II. Advances in Information Security, vol. 100, pp. 97–115. Springer, New York, NY, USA (2013)

[15] Clark, A., Sun, K., Bushnell, L., Poovendran, R.: A game-theoretic approach to IP address randomization in decoy-based cyber defense. In: Decision and Game Theory for Security. LNCS, vol. 9406, pp. 3–21. Springer, Cham, Switzerland (2015). https://doi.org/10.1007/978-3-319-25594-1_1

[16] Feng, X., Zheng, Z., Mohapatra, P., Cansever, D.: A Stackelberg game and Markov modeling of moving target defense. In: Decision and Game Theory for Security. LNCS, vol. 10575, pp. 315–335. Springer, Cham, Switzerland (2017)

[17] Zhou, Y., Cheng, G., Jiang, S., Dai, M.: Cost-effective moving target defense against DDoS attacks using trilateral game and multi-objective Markov decision processes. Comput. Secur. **97** (2020) https://doi.org/10.1016/j.cose.2020.101976 . Art. no. 101976

[18] Zhou, Y., Cheng, G., Jiang, S., Hu, Y., Zhao, Y., Chen, Z.: A cost-effective shuffling method against DDoS attacks using moving target defense. In: Proc. 6th ACM Workshop Moving Target Defense (MTD), pp. 57–66 (2019). https://doi.org/10.1145/3338468.33568

[19] Etxezarreta, X., Garitano, I., Iturbe, M., Zurutuza, U.: Low delay network attributes randomization to proactively mitigate reconnaissance attacks in industrial control systems. Wireless Netw. **30**, 5077–5091 (2024) https://doi.org/10.1007/s11276-022-03212-5

[20] Zhang, T., Kong, F., Deng, D., Tang, X., Wu, X., Xu, C., Zhu, L., Liu, J., Ai, B., Han, Z.: Moving target defense meets artificial-intelligence-driven network: A comprehensive survey. IEEE Internet Things J. **12**(12), 13384–13397 (2025) https://doi.org/10.1109/JIOT.2025.3533016

[21] Schwartz, J., Kurniawati, H.: Autonomous penetration testing using reinforcement learning. arXiv:1905.05965 (2019). https://doi.org/10.48550/arXiv.1905.05965

[22] Hu, Z., Beuran, R., Tan, Y.: Automated penetration testing using deep reinforcement learning. In: Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW), pp. 2–10 (2020). https://doi.org/10.1109/EuroSPW51379.2020.00010

[23] Li, Q., Hu, M., Hao, H., Zhang, M., Li, Y.: INNES: An intelligent network penetration testing model based on deep reinforcement learning. Appl. Intell. **53**(22), 27110–27127 (2023)

[24] Microsoft Defender Research Team: CyberBattleSim: An Experimentation Research Platform for the Interaction of Automated Agents in a Simulated Abstract Enterprise Network. https://github.com/microsoft/CyberBattleSim (2021)

[25] Walter, E., Ferguson-Walter, K., Ridley, A.: Incorporating deception into CyberBattleSim for autonomous defense. arXiv:2108.13980 (2021). https://doi.org/10.48550/arXiv.2108.13980

[26] Eghtesad, T., Vorobeychik, Y., Laszka, A.: Adversarial deep reinforcement learning based adaptive moving target defense. In: Decision and Game Theory for Security. LNCS, vol. 12513, pp. 58–79. Springer, Cham, Switzerland (2020)

[27] Zhang, T., Xu, C., Shen, J., Kuang, X., Grieco, L.A.: How to disturb network reconnaissance: A moving target defense approach based on deep reinforcement learning. IEEE Trans. Inf. Forensics Security **18**, 5735–5748 (2023) https://doi.org/10.1109/TIFS.2023.3314219

[28] Han, Y., Rubinstein, B.I.P., Abraham, T., Alpcan, T., De Vel, O., Erfani, S., Hubczenko, D., Leckie, C., Montague, P.: Reinforcement learning for autonomous defence in software-defined networking. arXiv:1808.05770 (2018). https://doi.org/10.48550/arXiv.1808.05770

[29] Chowdhary, A., Huang, D., Mahdavi, J.S., Sabur, A., Vadnere, S., Ufuktepe, E., Hewavitharana, U.: SDN-based moving target defense using multi-agent reinforcement learning. In: Proc. 1st Int. Conf. Auton. Intell. Cyber-Defense Agents (AICA), pp. 1–11 (2021)

[30] Hammar, K., Stadler, R.: Finding effective security strategies through reinforcement learning and self-play. In: Proc. 16th

Int. Conf. Netw. Service Manage. (CNSM), pp. 1–9 (2020). https://doi.org/10.23919/CNSM50824.2020.9269092

[31] Datar, M., Dujardin, Y.: Adaptive learning for moving target defence: Enhancing cybersecurity strategies. arXiv:2508.17945 (2025). https://doi.org/10.48550/arXiv.2508.17945

[32] Reti, D., Elzer, K., Fraunholz, D., Schneider, D., Schotten, H.D.: Evaluating deception and moving target defense with network attack simulation. In: Proc. 9th ACM Workshop Moving Target Defense (MTD), pp. 45–53 (2022). https://doi.org/10.1145/3560828.3564006

[33] Achleitner, S., La Porta, T.F., McDaniel, P., Sugrim, S., Krishnamurthy, S.V., Chadha, R.: Deceiving network reconnaissance using SDN-based virtual topologies. IEEE Trans. Netw. Service Manage. **14**(4), 1098–1112 (2017) https://doi.org/10.1109/TNSM.2017.2724239

[34] Lin, H., Zhang, J., Su, Y.: SDN-based in-network honeypot: Preemptively disrupt and mislead attacks in IoT networks. arXiv:1905.13254 (2019). https://doi.org/10.48550/arXiv.1905.13254

[35] Molina Zarca, A., Bagaa, M., Bernal Bernabe, J., Taleb, T., Skarmeta, A.F.: Semantic-aware security orchestration in SDN/NFV-enabled IoT systems. Sensors **20**(13), 3622 (2020) https://doi.org/10.3390/s20133622

[36] Khoa, N.H., Do Hoang, H., Ngo-Khanh, K., Duy, P.T., Pham, V.-H.: SDN-based cyber deception deployment for proactive defense strategy using honey of things and cyber threat intelligence. In: Intelligence of Things: Technologies and Applications. LNNS, pp. 269–278. Springer, Cham, Switzerland (2023)

[37] Alnfiai, M.M.: AI-powered cyber resilience: A reinforcement learning approach for automated threat hunting in 5G networks. EURASIP J. Wireless Commun. Netw. **2025** (2025) https://doi.org/10.1186/s13638-025-02497- . Art. no. 47

[38] Robles-Carrillo, G., Pérez-Vargas, D., Maestre-Vidal, J., Sotelo-Monge, M.A.: CYBERSHIELD: A competitive simulation environment for training AI in cybersecurity. In: Proc. IEEE Int. Mediterranean Conf. Commun. Netw. (MeditCom), pp. 1–6 (2024)

[39] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv:1707.06347 (2017). https://doi.org/10.48550/arXiv.1707.06347

[40] Charpentier, A., Neal, C., Boulahia-Cuppens, N., Cuppens, F., Yaich, R.: Real-time defensive strategy selection via deep reinforcement learning. In: Proc. 18th Int. Conf. Availability, Rel. Secur. (ARES), Benevento, Italy, pp. 1–11 (2023)