Designing CAPTCHA Systems with Reinforcement Learning for Adaptive Defense

Meghana Indukuri , Eman Naseerkhan , Joshua Rose , Martin Tran , Younghee Park *

Posted Date: 16 April 2026

doi: 10.20944/preprints202604.1178.v1

Keywords: CAPTCHA; large language models; reinforcement learning; cybersecurity; proximal policy optimization; bot detection; reinforcement learning agent

 doi:10.20944/preprints202604.1178.v1

Article

Designing CAPTCHA Systems with Reinforcement Learning for Adaptive Defense

Meghana Indukuri, Eman Naseerkhan, Joshua Rose, Martin Tran and Younghee Park *

Department of Computer Engineering, San Jose State University, San Jose, CA 95192, USA
* Correspondence: younghee.park@sjsu.edu

Abstract

CAPTCHA systems remain a widely deployed defense against automated abuse, but advances in machine learning have reduced the effectiveness of traditional challenge-based designs and exposed limitations in proprietary risk-scoring systems. This paper presents an adaptive, reinforcement learning-based CAPTCHA defense framework for high-security web applications. The proposed system formulates bot detection as a partially observable Markov decision process and uses a Proximal Policy Optimization agent with Long Short-Term Memory to analyze streamed behavioral telemetry, including mouse movements, clicks, keystrokes, and scrolling, over sequential interaction windows. Based on accumulated evidence, the agent can continue observing, deploy a honeypot, issue graded CAPTCHA challenges, allow a session, or block it. To complement the sequential agent, the framework also includes an XGBoost classifier that produces a session-level human-likelihood score as a supervised benchmark. Experiments on a simulated ticket-purchasing web application using human-generated sessions and multiple bot tiers, including scripted, replay-based, and LLM-powered agents, show strong preliminary performance. Among the evaluated reinforcement learning variants, Soft PPO achieved the best test performance with two reward structures, with one it reached 98.8% accuracy, 100% precision, and 0.987 F1 score, while with the revised reward structure it reached 96.4% accuracy, 100% precision, and 0.963 F1 score. The XGBoost classifier achieved 99.48% accuracy, 1.000 ROC-AUC, and 0.9919 F1 score. The results indicate that sequential reinforcement learning can support accurate and low-friction bot detection, while the accompanying classifier provides an interpretable and efficient benchmark. Compared with proprietary systems such as Google reCAPTCHA v3, the proposed framework emphasizes transparency, auditability, and explicit sequential decision-making rather than black-box risk scoring. Overall, this work introduces an open and adaptive CAPTCHA-defense framework that offers a promising alternative for studying and deploying behavior-based bot mitigation.

Keywords: CAPTCHA; large language models; reinforcement learning; cybersecurity; proximal policy optimization; bot detection; reinforcement learning agent

1. Introduction

Malicious bots are a pervasive problem. They are responsible for a variety of attacks across online platforms, such as ticket scalping, credential stuffing, spam, large-scale data scraping and more. As of 2025, over 10.6 million websites deploy Google's reCAPTCHA service as their primary line of defense against these threats [1], reflecting the scale at which of this issue. Bot detection has become a fundamental requirement of modern web security. In 2003, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) was formally introduced by von Ahn et al. [2] as a class of AI-hard (i.e., difficult for automated systems but frictionless for humans) challenge-response tests designed to block automated programs from abusing online services, while ensuring they remain accessible to human users. The foundation of CAPTCHA design is to exploit

 doi:10.20944/preprints202604.1178.v1

tasks that are computationally trivial for humans, but onerous for machines, providing a mechanism for distinguishing legitimate users from bots that is both reliable and scalable.

Due to the advent of artificial intelligence, and more specifically machine learning, traditional CAPTCHA schemes have been defeated convincingly. Comprehensive literature reviews, such as the one from Dinh and Hoang [3], divide CAPTCHA into specific security schemes. One such scheme is text-based CAPTCHA, where humans must decipher distorted text and numbers with the aim of blocking bots from accessing webpages. However, researchers have developed a modified variant of convolutional neural networks (CNN) to quickly solve any form of text-based CAPTCHA [4]. Image-based CAPTCHAs follow a similar logical flow but instead use graphical puzzles. In a similar vein, with computer vision algorithms, CNNs, Support Vector Machines and other methods, image-based CAPTCHA security measures have also been compromised [3]. Audio CAPTCHAs, which were introduced to improve accessibility, have proven equally vulnerable. Bock et al. [5] demonstrated that an automated speech recognition ensemble could defeat reCAPTCHA's audio challenge with over 85% accuracy. Video-based CAPTCHAs, which typically involve moving text or an animated object, create difficulty for bots due to their motion and temporal elements. However, they are not widely used due to issues with bandwidth requirements, accessibility and excessive friction for human users [6] [7].

Across all of its versions, a core limitation of CAPTCHA remains the same: high-performing systems remain proprietary. Google's reCAPTCHA v3 and hCaptcha Enterprise both incorporate risk-based scoring, yet their implementations are entirely proprietary, making them impossible to audit, validate, or extend [8,9]. Current open-source systems apply a fixed detection approach and a fixed challenge response that cannot adapt to adversarial behavior over time. Open-source alternatives such as ALTCHA utilize cryptographic proof-of-work mechanisms to raise the cost of bot attacks, but lack the behavioral intelligence to distinguish bots from humans in real time [10]. To our knowledge, no existing open-source system treats bot detection as a sequential decision-making problem, where a defender continuously observes user behavior and adaptively selects both its detection strategy and challenge response based on accumulated evidence. This project directly addresses that gap.

This paper proposes a silent, reinforcement learning-based CAPTCHA system designed for high-security web applications. The system frames bot detection as a Partially Observable Markov Decision Process (POMDP), in which a Proximal Policy Optimization agent with a Long Short-Term Memory architecture (PPO+LSTM) makes sequential classification decisions over a sliding window of behavioral telemetry events. The system collects mouse movements, keystrokes, scroll events, and click data silently as users navigate a simulated ticket-purchasing web application, encoding these signals into a 26-dimensional feature vector per window. At the terminal state of each session, the agent adaptively deploys an easy, medium, or hard CAPTCHA challenge, issues a honeypot, blocks the user, or passes the user without friction, depending on its classification confidence. Generally, honeypots are referred to as deceptive decoy systems designed to attract attackers [11]; in our context, honeypot deployment refers to embedding fields that are visually hidden from human users but exist in the backend, enabling the detection of automated agents that interact with these otherwise invisible elements. Furthermore, we also introduce a companion XGBoost classifier which provides a holistic, session-level human-likelihood score, allowing direct comparison between the two approaches. Overarchingly, the key contributions of this work are:

- This paper proposes a new RL-based CAPTCHA system with dynamic feature engineering in real time.
- We design a new feature vector space by considering users' real-time behaviors using keystrokes and mouse movements.
- We implement the proposed system and evaluate the performance of the RL agent and XGBoost classifier.

The remainder of this paper is organized as follows: Section 2 reviews related work in CAPTCHA systems, their vulnerabilities, behavioral biometrics and the use of reinforcement learning in cybersecurity. Section 3 describes our methodology, including the web application, data collection procedure,

 doi:10.20944/preprints202604.1178.v1

the adversarial bot framework, RL agent architecture and XGBoost classifier design. Section 4 presents experimental results for both the RL agents and the classifier. Section 5 discusses our key findings, system limitations, and directions for future research.

2. Related Work

2.1. CAPTCHA Systems and their Evolution

CAPTCHA was introduced as an online security mechanism designed to prevent automated fraud and abuse, wherein unique challenges are used to distinguish bots and humans [2].The first generation consisted of text-based CAPTCHAs, which required users to identify and transcribe distorted alphanumeric characters rendered against noisy backgrounds. These were followed by image-based CAPTCHAs, which presented users with visual puzzles requiring the identification of objects belonging to a specified category, a format that was used in Google's reCAPTCHA v2 image grid challenges. Both generations relied on the premise that the visual recognition tasks involved were computationally intractable for automated systems.

The advancements in machine learning has brought about additional evolutions in CAPTCHAs. One such methodology tracks user behavior on a webpage, distinguishing between humans and malicious automation through behavioral and sensor metrics [3,12]. This invisible CAPTCHA works in the background, collecting information about the user without a direct puzzle. This approach has become prevalent due to its ease on users and its security performance.

Beyond purely invisible approaches, some systems have explored hybrid designs. hCaptcha Enterprise mixes hidden behavioral scoring with adaptive challenges, deploying harder visual puzzles only when its risk model identifies elevated bot probability [9]. Open-source alternatives such as ALTCHA couple cryptographic proof-of-work with behavioral signals to raise the cost of automated abuse in a transparent manner [10]. However, a key concern with all of these systems is that their detection strategies are fixed at the start of the deployment. Furthermore, research has highlighted an inherent tradeoff in CAPTCHA design: increasing difficulty to deter bots correspondingly raises friction and error rates for legitimate users [13].

2.2. Bots & AI-Based CAPTCHA Attacks

The vulnerability of text-based CAPTCHAs to machine learning attacks has been well established. Tang et al. [4] demonstrated that CNNs could break text-based CAPTCHAs deployed by the top 50 most popular international websites with high accuracy, targeting schemes that utilize noise, distortion, and anti-segmentation techniques.

Image-based CAPTCHAs have proven equally susceptible. Sukhani and Chitaliya [14] showed that a multi-class CNN model could solve image-grid puzzles with 92.98% accuracy, while Sivakorn et al. [15] defeated Google's image reCAPTCHA at scale by combining deep learning-based object recognition with cookie manipulation to influence the risk analysis system. Plesner et al. [16] showed that deep learning models such as the YOLO (You Only Look Once) v8 model can solve reCAPTCHAv2 image challenges with 100% success. Audio CAPTCHAs, introduced to improve accessibility, have fared no better: Bock et al. [5] showed that an automated speech recognition ensemble could mount a large-scale attack against reCAPTCHA's audio challenge with an attack success rate of 85.15%.

In particular, Akrout et al. [17] demonstrated that RL agents can be trained to bypass reCAPTCHA v3 by learning mouse movement patterns that achieve a high risk score. The agent formulated the problem as a grid-world and achieved a success rate of 97.4%. Together, these studies make a compelling case that both challenge-based and behavioral CAPTCHAs are vulnerable to targeted AI attacks, motivating the need for adaptive defenses.

2.3. Behavioral Biometrics

The high data cost associated with this project required that our selected data be simple to collect, widely available in environments similar to ours, and effective for bot detection. To meet these data

 doi:10.20944/preprints202604.1178.v1

constraints, we track a variety of mouse and keyboard interaction telemetry. This data can be collected using straightforward front-end web code that can be deployed to most websites. Nearly all web interaction require some combination of mouse and keyboard use to proceed, making this telemetry broadly applicable.

To ensure the effectiveness of bot detection, we turned to various research cases to identify the most valuable telemetry to track. Mouse dynamics, including movement and clicking behavior, were used by [18] to detect bots with up to 99.20% accuracy. Similarly, [19] used real and synthetic mouse trajectory to supplement other other approaches (e.g., Google's reCAPTCHA) to achieve 98.7% accuracy. In another study, [20] a variety of mouse and keyboard inputs were used to detect bots in online video games with 99% accuracy and negligible performance overhead.

These studies show that bot interaction with online environments is often simple and contains many obvious differences from human data. Mouse movements are often overly smooth while keyboard inputs are unrealistically fast or uniformly timed compared to human inputs. Based on these findings we conclude that a variety of mouse movement and keyboard inputs combined provide a low-cost, widely applicable and effective approach to bot detection.

2.4. Reinforcement Learning for Cybersecurity

Reinforcement learning has emerged as an increasingly viable approach for cybersecurity problems because many defensive tasks are inherently sequential. Nguyen and Reddi [21] provide a comprehensive survey of deep RL (DRL) applied to cybersecurity, covering its application to intrusion detection systems (IDS), cyber-physical system defense, and multi-agent game-theoretic simulations of attacker-defender dynamics. Their review highlights an advantage of RL over static classifiers: while supervised models are trained once on historical attack patterns, RL agents can continuously refine their policies, making them well suited to adversaries that evolve their strategies over time.

Within intrusion detection specifically, RL-based methods have been used to detect anomalies in network traffic and respond to attack patterns that fall outside the distribution of training data. This robustness to distributional shift is valuable in security contexts, where adversaries actively probe for the gaps in a deployed system's knowledge. The adversarial framing, in which the RL agent is cast as a defender operating against an implicit attacker, maps to problems like bot detection, where the goal is not simply to classify known bots, but to maintain a reliable policy as bot strategies evolve [21].

2.5. Existing Dynamic CAPTCHA Systems

The current industry standard for invisible bot detection is Google's reCAPTCHA v3 [8]. It assigns a risk score between 0.0 and 1.0, based on passive behavioral signals collected during a user session. Rather than presenting explicit challenges, reCAPTCHA v3 operates in the background, leaving the site operator to decide what action to take based on the score. These actions may consist of allowing the user through, blocking them, or deploying various types of CAPTCHA puzzles. Google Cloud documentation further describes 11 score levels, although only four are exposed in the free version, and indicates that site-specific performance can improve through observation of production traffic and post hoc assessment annotation [22], [23]. In addition, reCAPTCHA supports the use of backend contextual signals, including IP address, user agent, JA3, and JA4 fingerprints, and policy-based challenge keys that permit deterministic challenge escalation at configured score thresholds [24],[25]. These design choices indicate that reCAPTCHA v3 is not merely a fixed puzzle system, but a mature proprietary risk-scoring service with substantial deployment experience and operational scale [26].

hCaptcha Enterprise [9] adopts a hybrid approach, combining hidden behavioral scoring with adaptive visible challenges: users with high probability of bots are presented with harder puzzles in the image-grid, while low-risk users may pass with minimal friction. Both systems incorporate behavioral signals and adaptive risk thresholds, representing a significant step beyond static challenge-based CAPTCHAs.

Despite their prevalence, a critical limitation of both reCAPTCHA v3 and hCaptcha Enterprise is that their detection logic is proprietary. Their models cannot be audited or validated against novel bot

 doi:10.20944/preprints202604.1178.v1

strategies by researchers or operators. Open-source alternatives exist but address different problems: ALTCHA [10] uses a cryptographic proof-of-work mechanism to raise the computational cost of bot attacks. It provides transparency and audit-ability, but does not perform behavioral bot detection and cannot adapt to adversaries that are willing to absorb the added computational overhead.

More recently, Aura-CAPTCHA [27] proposed an adaptive multi-modal CAPTCHA system that combines GANs for dynamic challenge generation with a Q-learning agent for real-time difficulty adjustment based on user behavior. The system achieved a human success rate of 92% and a bot bypass rate of 10% in preliminary evaluations. However, Aura-CAPTCHA still operates at the challenge level, it adjusts the visual difficulty of a displayed puzzle, rather than framing the entire session as a sequential decision-making problem. No existing open-source system treats bot detection as a POMDP in which the defender continuously observes behavioral evidence and selects both its detection strategy and challenge response over time.

2.5.1. Qualitative Comparison with Proposed System

This distinction is especially important when comparing prior systems with the framework proposed in this paper. A useful qualitative point of reference is reCAPTCHA v3. Both approaches operate in the background and based on behavioral evidence rather than requiring immediate visible challenge. However, reCAPTCHA v3 ultimately exposes a proprietary risk score, leaving downstream intervention logic to the site operator or to Google's closed infrastructure. By contrast, our framework, which will be further discussed in the paper, formulates bot defense as a partially observable Markov decision process (POMDP) and learns an explicit sequential policy over continued observation, honeypot deployment, graded CAPTCHA challenges, allow, and block actions. The difference, therefore, is not simply whether an adaptation exists, but where that adaptation resides: in reCAPTCHA v3, it is embedded within a proprietary service, whereas in our system the intervention policy itself is learned, auditable, and modifiable. At the same time, reCAPTCHA v3 remains substantially stronger in deployment maturity, ecosystem integration, and scale in the real-world. Since Google does not publish public precision, recall, and false-positive benchmarks that are directly comparable to the data set and task formulation used in this study, this comparison should be understood as architectural and operational rather than as a strict head-to-head performance evaluation. Consequently, the contribution of the present work is not to claim benchmark superiority over reCAPTCHA v3, but to introduce an open, auditable, and sequentially adaptive CAPTCHA-defense framework with strong preliminary internal results.

2.6. Gap Analysis and Contribution

As highlighted in the previous sections, challenge-based CAPTCHAs have been repeatedly defeated by deep learning, behavioral scoring systems remain vulnerable to adversarial agents, and most deployed defenses are proprietary and static. No existing open-source system treats bot detection as a sequential decision-making problem in which a defender continuously adapts its detection strategy and challenge response based on accumulated behavioral evidence. This study addresses these gaps by proposing the first open-source, reinforcement learning-driven CAPTCHA framework that formulates bot detection as a POMDP, combines a PPO+LSTM sequential agent with a companion XGBoost classifier, and adaptively calibrates its challenge response to its real-time classification confidence, creating a closed-loop adaptive defense that can improve continuously after deployment.

3. Methodology

3.1. Web Application

For this project a web application was developed to simulate a real world high-security environment requiring anti-bot security measures. The system records per-session user inputs, which is used by a silent RL agent that evaluates activity for each unique session.

 doi:10.20944/preprints202604.1178.v1

User interaction with the application spans three primary pages: concert selection, seat selection, and checkout. These pages and the CAPTCHA challenges delivered to the user are shown in Figure 1. Additionally, the application includes a developer dashboard that provides session telemetry and displays RL agent decisions.

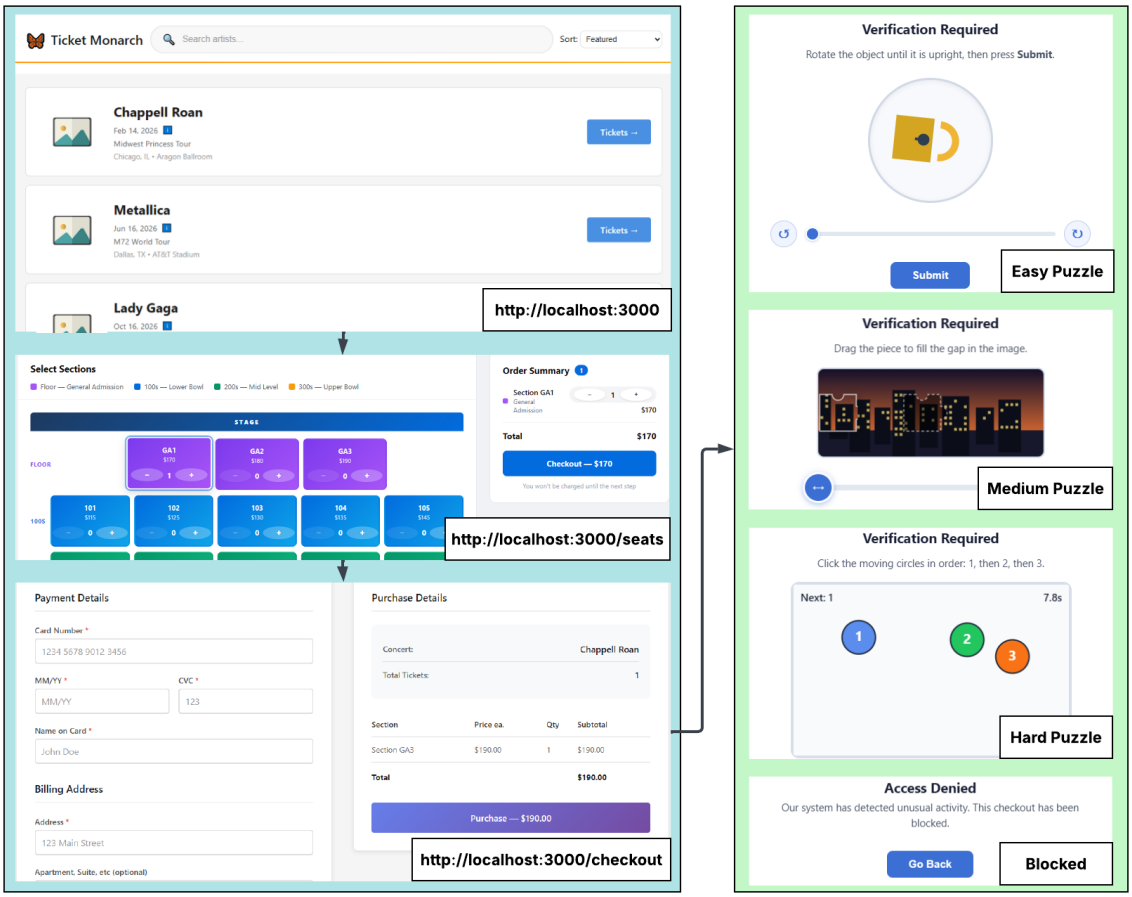

Figure 1. Ticket Monarch Web Environment.

Given that the application represents a mock e-commerce website, it is assumed that, in a real world deployment, operators would prefer to challenge bots, rather than risk blocking human users and losing transactions. Accordingly, the application deploys one of three CAPTCHA puzzles, rather than a hard block, if the user is suspected to be a bot. The "easy" and "medium" puzzles follow image-based sliding CAPTCHA designs. The easy puzzle involves rotating an asymmetric object to an upright position, with generous tolerance to ensure high success rates for human users. The medium is a jigsaw slider challenge that increases difficulty by introducing visual ambiguity and reduced tolerance. The "hard" puzzle is reserved for highly suspicious sessions and significantly increases complexity. It is a sequential click task in which users must select moving objects in a specified order under a time constraint; failure in two attempts will result in blocking.

3.2. Data Collection

The dataset contains two categories; human-generated and bot-generated interactions. Human data was collected through direct interaction with the application. Bot data was generated using LLM agents and replay bots operating in the same environment. To supplement and vary the bot data, a data augmentation process was applied. This process is detailed in Section 3.3. Overall, the data set contains 639 original sessions (203 human, 436 bot).

For each session, telemetry data included keystrokes, mouse movements, scroll events, and button presses. Keystrokes, scrolls, and clicks were logged as discrete events, whereas mouse movements

 doi:10.20944/preprints202604.1178.v1

were recorded at 15 millisecond intervals. A detailed summary of the collected features is provided in Table 1.

Table 1. Telemetry Data.

Signal	Rate	Data
Mouse Movement	15 ms sampling	Position (x, y), Timestamp
Mouse Clicks	Every click	Position (x, y), Target Element, Time Delta
Key Strokes	Every key press	Field ID, Type (Down / Up), Timestamp, Special Keys (Backspace, Tab, etc...)
Mouse Scrolls	Every scroll	Scroll X, Scroll Y Delta, Time Delta

The distinctions between human and bot activity are illustrated in Figures 2 and 3 which present heatmap visualizations of interaction data. These figures map the X-Y pixel positions of all recorded interactions, with the mean position of each distribution indicated by a white marker. For mouse movements, the mean positions are approximately (505, 358), (593, 420), and (594, 419), while for mouse clicks they are (425, 379), (483, 451), and (483, 451).

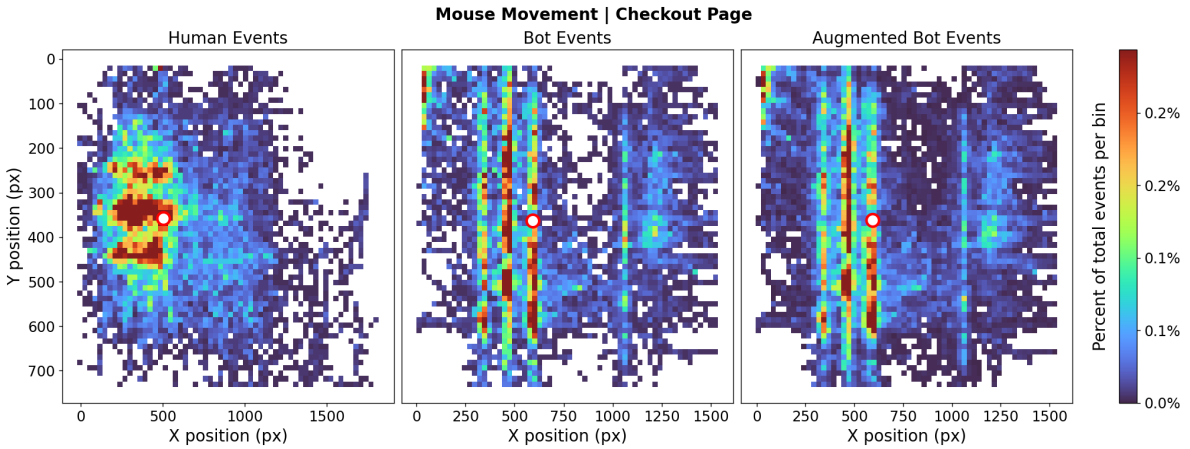

Figure 2. Mouse Movement Telemetry Heatmaps.

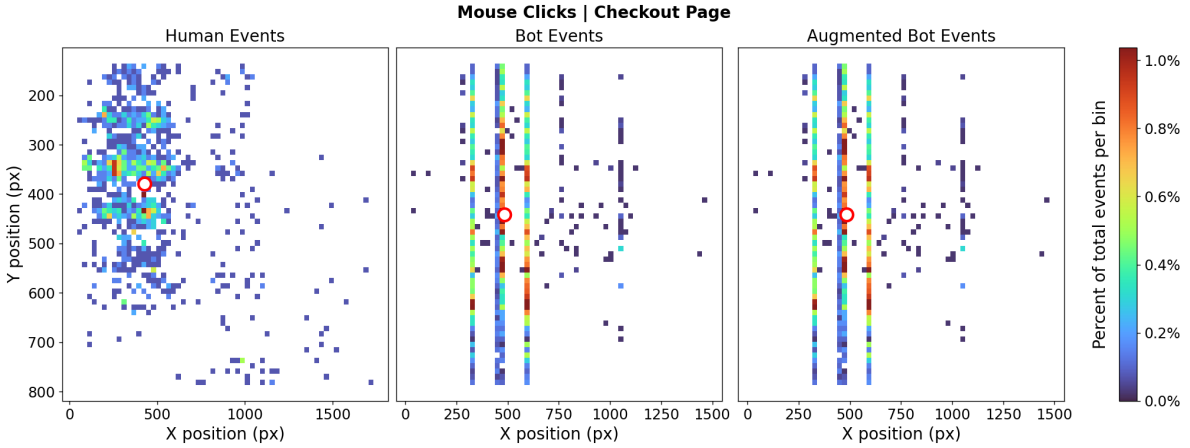

Figure 3. Mouse Click Telemetry Heatmaps.

These visualizations highlight clear behavioral differences between human and bot activity. Human interaction exhibits broad spatial dispersion across the interface, indicating a more exploratory behavior. In contrast, bot interactions are highly structured, with limited horizontal variance and a

 doi:10.20944/preprints202604.1178.v1

small number of unique x-axis positions. This results in the distinct vertical bands of activity observed in both movement and click heatmaps. The mouse clicks heatmaps emphasize this distinction. The standard deviation along the x-axis for human clicks is approximately 1.9 times greater than that of bot data. Additionally, human clicks spanned 559 unique x-axis positions while bot clicks are confined to only 73. Human click interactions also occupy a substantially larger number of spatial bins, covering approximately 17% of the provided space while bot activity only covered about 9%. This data also shows the behavior differences between mouse movements and clicks. Human mouse movements occupied approximately 65% of the environment space while clicks occupy only about 17%. This demonstrates the highly exploratory nature of mouse movements and positioning versus more deliberate and targeted nature of clicking.

In total, the dataset includes 39,401 human mouse movements events and 28,712 bot movement events, which were augmented to produce an additional 171,988 samples. For mouse clicks, the dataset contains 1,376 human events and 4,193 bot events, with an additional 25,116 augmented samples. Notably, the augmented bot dataset closely mirrors the statistical properties of the original bot data across all evaluated metrics, suggesting that augmentation preserves the underlying spatial interaction patterns rather than introducing human-like variability.

3.3. Adversarial Bot Tier Framework and Augmentation

To capture a broad and realistic spectrum of adversarial behavior on websites, we organize our bot implementations into a five-tier difficulty hierarchy. Each tier reflects increasing levels of behavioral complexity, ranging from simple scripted automation to highly realistic human-mimicking agents, including those powered by LLMs.

 doi:10.20944/preprints202604.1178.v1

Table 2. Adversarial Bot Tier Hierarchy.

Tier	Name	Bot Types	Description
1	Commodity	linear, tabber, speedrun	These bots have obvious automation patterns such as constant cursor speed, tab-only navigation, and minimal mouse movement. Their behavior is easy to distinguish from human interaction.
2	Careful Automation	scripted, stealth, slow, erratic, replay	These bots attempt to mimic human behavior using Bezier curve mouse trajectories [28], randomized delays, and natural pacing. While less predictable than Tier 1, they remain detectable through timing variance and movement regularity.
3	Pseudo-Semi-Automated	semi_auto	These bots combine scripted navigation with simulated human-operator handoffs. One phase is automated (e.g., browsing and seat selection) while another uses human-like interaction patterns (e.g., checkout), creating sessions with mixed behavioral signatures that are harder to classify with a single global decision.
4	Trace-Conditioned	trace_conditioned	These bots replay recorded human mouse trajectories with added Gaussian noise and simulate human-profiled typing intervals, producing sessions that statistically resemble real user behavior.
5	LLM-powered	Claude [29], GPT-4o [30], Gemini [31]	These bots are autonomous AI agents navigating web pages via browser-use [32]. They perceive the page through screenshots, Document Object Model (DOM), or an accessibility tree via Chrome DevTools Protocol (CDP) [33]. They then decide actions through a ReAct-style observe-reason-act loop [34], where the LLM observes the current page state, selects an action (click, type, scroll), and receives updated observations after each interaction.

Development-wise, for Tiers 1 through 4, browser interactions are implemented using Selenium WebDriver to simulate automated behaviors of varying complexity [35]. In general, we expect detection performance to decrease from Tier 1 to Tier 5, reflecting increasing adversarial difficulty. Deviations from this trend may indicate that certain behavioral strategies are more effective at evasion than their tier ranking suggests, or that specific agents are better suited to particular adversary types. The corresponding results are presented in Sections 4 and 5.

Adversarial Augmentation

A key challenge with bot detection is that trivially separable features (e.g., Selenium's $\sim$1 ms key-hold durations) can inflate model accuracy without learning genuinely discriminative behavioral patterns. To address this, we draw inspiration from adversarial training [36], which improves model robustness by exposing it to perturbed inputs during training. We adapt this principle to the behavioral bot detection domain by introducing a novel adversarial augmentation procedure — the `HumanProfiler` pipeline — that progressively *humanizes* bot sessions at three difficulty levels, directly mapping to the bot-tier framework in Section 3.3. To our knowledge, no prior work has

 doi:10.20944/preprints202604.1178.v1

applied progressive humanization of bot telemetry as a data augmentation strategy for behavioral bot detection; this pipeline is a novel contribution of this work.

A `HumanProfiler` first learns statistical profiles from real human sessions across six signal categories: key-hold duration, mouse inter-event Δt, jitter ratio, mouse speed, direction-change frequency, and event-type ratios. Let $\mu_h^{(f)}$ and $\sigma_h^{(f)}$ denote the human mean and standard deviation for feature f. Bot sessions are then transformed as follows:

- **Easy augmentation:** Fixes the most obvious giveaways. Key-hold durations are resampled from a clipped Gaussian centered on the human profile, and micro-jitter is injected into mouse trajectories:
$$d'_{\text{hold}} \sim \mathcal{N}\left(\mu_h^{(\text{hold})},\, \sigma_h^{(\text{hold})}\right), \quad \mathbf{p}'_t = \mathbf{p}_t + \boldsymbol{\epsilon}_t, \;\; \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0},\, \sigma^2_{\text{jitter}}\mathbf{I}). \tag{1}$$
- **Medium augmentation:** Applies all easy transforms and additionally humanizes timing distributions by compressing mouse Δt values toward human rates, and applies exponential smoothing to reduce abrupt direction changes in mouse paths:
$$\Delta t'_k = \beta\, \Delta t_k + (1-\beta)\, \mu_h^{(\Delta t)}, \qquad \mathbf{p}'_t = \alpha_s\, \mathbf{p}_t + (1-\alpha_s)\, \mathbf{p}'_{t-1}, \tag{2}$$
where β controls timing compression and α_s is the smoothing coefficient.
- **Hard augmentation:** Applies all transforms with tighter parameters (smaller σ_{jitter}, stronger smoothing, narrower timing compression), producing near-human sessions that are challenging to distinguish from real users.

The specific parameter values used at each difficulty level are summarized in Table 3.

Table 3. Adversarial Augmentation Parameters by Difficulty Level.

Parameter	Easy	Medium	Hard
σ_{jitter} (px)	3.0	2.0	1.0
Timing compression β	—	0.7	0.4
Path smoothing α_s	—	0.8	0.6
Fix key-hold durations	✓	✓	✓

For each original bot session, multiple augmented copies are generated (default: 2 copies $\times$ 3 levels $=$ 6 augmented sessions per bot). This forces the model to learn subtle behavioral signals rather than relying on trivially separable artifacts, directly strengthening robustness against higher-tier adversaries.

3.4. Reinforcement Learning for Adaptive CAPTCHA Defenses

3.4.1. Reinforcement Learning Formulation

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns to make decisions through trial-and-error interactions with an environment. The agent's objective is to learn a policy that maximizes the expected cumulative reward over time [37]. RL formally stems from Markov Decision Processes (MDPs), which are defined as a tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma),$$

where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $P(s' \mid s, a)$ the transition probability function, $R(s,a)$ the reward function, and $\gamma \in [0,1)$ the discount factor that determines the relative importance of future rewards [37,38]. A policy is defined as

$$\pi(a \mid s),$$

 doi:10.20944/preprints202604.1178.v1

which specifies the probability of selecting action a given state s [37]. The value function is given by

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s],$$

where

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

This represents the expected discounted return from state s under policy π [37]. Similarly, the advantage function is defined as

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s),$$

which measures the relative benefit of taking action a compared to the expected value of the policy [37,39]. These components lay the foundation for modern policy gradient methods such as Proximal Policy Optimization (PPO) and actor-critic algorithms [40–42], which commonly use Generalized Advantage Estimation (GAE) [39] to reduce variance in gradient updates.

In the context of our problem (i.e., CAPTCHA defenses), this RL formulation can be mapped to user interaction sessions on a web application. The environment corresponds to a user session, and the state space ($\mathcal{S}$) captures user interaction behavior represented as windowed telemetry features (e.g., mouse movement, click events, keystrokes, and scrolling patterns). On the other hand, the action space ($\mathcal{A}$) consists of the RL agent's possible interventions such as continuing observation, deploying a honeypot, issuing challenges, allowing the user, or blocking the session. The reward function (R) is defined based on the outcome of these actions, assigning positive rewards for correctly identifying bots and legitimate users, and penalties for incorrect decisions or user friction. The transition dynamics (P) are governed by how users (human or automated) respond to these interventions over time, but are not modeled explicitly. This is because the RL agent is trained in a model-free setting, learning directly from observed interaction data without access to the transition probabilities [37]. In our setting, the initial state (s_0) corresponds to the start of a user session, prior to observing any interaction behavior. A terminal state (s_t) is reached when a final action is taken, such as allowing the user, blocking the session, or issuing a challenge. The overarching objective of the RL agent is to learn a policy (π) that maximizes the expected cumulative reward by accurately distinguishing between bots and legitimate users while minimizing user friction.

Critically, this problem is more accurately characterized as a partially observable MDP (POMDP) [43], since the agent does not directly observe whether the user is a bot or human. Instead, it must infer this hidden state from observable behavioral signals (e.g., mouse movements) and interaction patterns over time. Thus, due to this partial observability, sequence models, specifically Long Short-Term Memory (LSTM) networks, were utilized in this paper to keep track of past information over time [44]. Concretely, LSTM networks capture temporal dependencies and aggregate information across multiple timesteps, enabling the agent to approximate the underlying hidden state (i.e., whether the user is a bot or human). A visual representation of the POMDP for this problem is shown in Figure 4.

 doi:10.20944/preprints202604.1178.v1

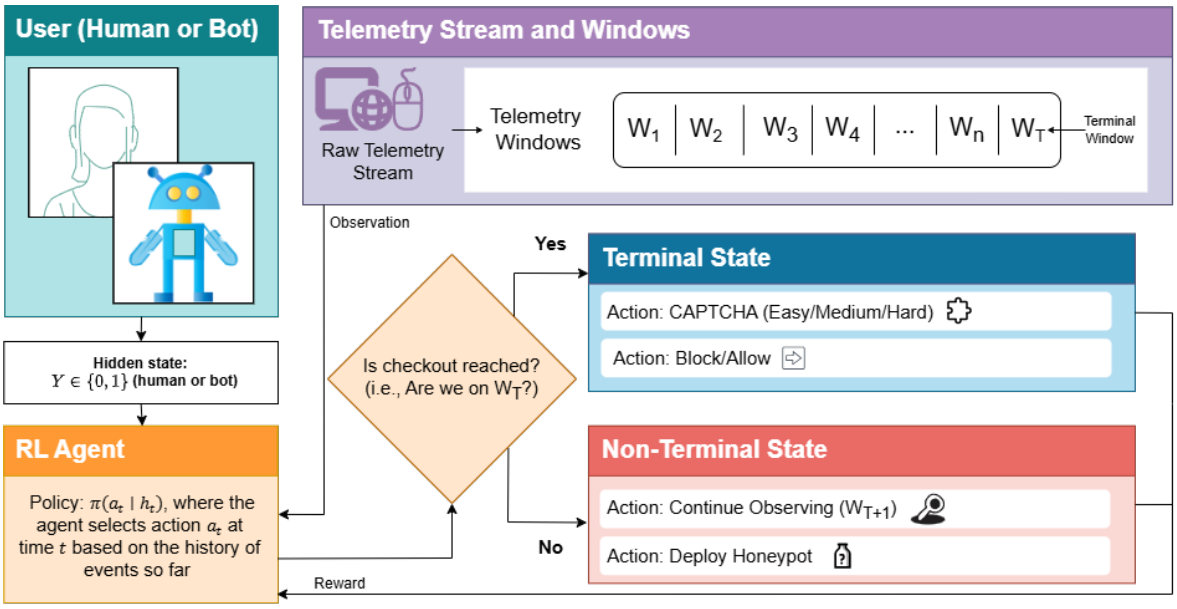

Figure 4. POMDP Diagram of CAPTCHA Defense Problem.

3.4.2. Observation, State, Action, and Reward Space

Observation Space

As discussed, the RL agent cannot directly observe the true state of the environment (i.e., whether the user is a bot or a human). Instead, it receives observations derived from user interaction telemetry. These observations consist of raw behavioral signals, including mouse movements, click events, keystrokes, and scrolling activity, collected during a user session. Raw telemetry is used to construct a chronological event timeline, where the initial state corresponds to the user opening the web page and the terminal state corresponds to checkout completion or session termination. To prevent mouse movement data from dominating the representation, mouse events are sub-sampled by retaining every fifth sample. The resulting event stream is then segmented into fixed-length windows of 30 events with a 50% overlap, producing a sequence of observations for each session, a standard technique for preserving temporal structure in sequential data [45].

State Representation

The true underlying state $s \in \mathcal{S}$ corresponds to whether the user is a bot or a human, which is not directly observable. Instead, the agent must infer this hidden state from the sequence of observations. Each observation window is encoded into a 26-dimensional feature vector capturing behavioral characteristics such as motor dynamics, spatial coverage, temporal patterns, and interaction metadata. This feature vector is shown in Table A1. Since a single observation window is insufficient to determine the user type, the agent utilizes an LSTM-based architecture [44] to aggregate information across multiple time steps (t). The LSTM maintains a hidden state that summarizes past observations, enabling the agent to create a latent representation that approximates the underlying user state (i.e., bot or human).

Action Space

The action space consists of seven discrete actions corresponding to different intervention strategies, including continuing observation, deploying a honeypot, issuing CAPTCHA challenges of varying difficulty, allowing the user, or blocking the session. Since certain actions (e.g., allow or block) should only be taken at the terminal stage of a session, invalid action masking [46] is applied to restrict the set of available actions at each timestep t, ensuring that only valid actions are selected.

 doi:10.20944/preprints202604.1178.v1

Reward Function

The reward function is designed to reflect the asymmetric costs of CAPTCHA decision-making. Positive rewards are assigned for correctly identifying bots and legitimate users, while penalties are applied for false positives, false negatives, and unnecessary user friction. Additionally, a CAPTCHA success probability model and associated cost function are incorporated during training to simulate the likelihood that a user successfully completes a challenge. During training, this probability model is used to determine the outcome of challenge actions and assign corresponding rewards or penalties.

Reward Structure and Challenge Outcome Formulation

We define the concrete reward configuration for terminal actions. Terminal actions consist of three puzzle difficulties (easy, medium, hard), as well as `allow` and `block`. For puzzle actions, outcomes are sampled using fixed success probabilities: humans are more likely to pass easier challenges, while bots are less likely to pass more difficult ones (Table 4). The resulting reward depends on both the user type and the outcome. Humans who pass incur a small friction cost that increases with difficulty, while humans who fail receive a false-positive penalty. Bots who pass incur a slip-through penalty, whereas bots caught by a puzzle receive a positive reward scaled by difficulty, incentivizing harder challenges for higher-confidence detections. Direct actions follow a simpler structure: allowing a human yields a positive reward, while allowing a bot incurs a false-negative penalty. Blocking a bot yields a moderate positive reward ($+0.7$), deliberately set below the puzzle-catch rewards ($+0.8$ to $+1.2$) to encourage evidence-based detection over opaque blocking. Blocking a human incurs the largest penalty (-1.5) to reflect the high cost of incorrectly denying access. The full revised reward mapping is provided in Table 5.

Table 4. Challenge pass probabilities used in training.

Difficulty	Human pass	Bot pass
Easy	0.95	0.40
Medium	0.85	0.15
Hard	0.70	0.05

Table 5. Revised terminal reward mapping. Puzzle outcomes are sampled stochastically with no per-action costs.

Scenario	**Reward**
Puzzle actions	
Human passes easy puzzle	-0.05
Human passes medium puzzle	-0.20
Human passes hard puzzle	-0.40
Human fails puzzle	-1.00
Bot passes puzzle	-0.50
Bot caught by easy puzzle	$+0.80$
Bot caught by medium puzzle	$+1.00$
Bot caught by hard puzzle	$+1.20$
Direct actions	
Allow human	$+0.50$
Allow bot	-1.00
Block bot	$+0.70$
Block human	-1.50

 doi:10.20944/preprints202604.1178.v1

Non-terminal actions include `continue`, which applies a small per-step penalty (-0.001), and `deploy_honeypot`. Honeypots trigger with tier-dependent probabilities for bots (ranging from 85% for Tier 1 commodity bots down to 5% for Tier 5 LLM-powered agents) and a fixed 1% probability for humans. When a honeypot is triggered by a bot, an information bonus ($+0.5$) is applied in the following timestep, encouraging their use as a signal-gathering mechanism without assuming perfect knowledge at deployment.

Evolution of the Reward Design (Legacy Baseline vs. Revised Schedule)

The legacy reward schedule, used for initial experiments and ablation comparisons, treated direct blocking and puzzle-based detection nearly identically. Blocking a bot directly yielded $+1.0$, the same base reward as catching a bot via any puzzle. However, puzzles also incurred per-action costs (easy: 0.1, medium: 0.3, hard: 0.5), making the net reward for a hard puzzle catch only $+0.50$ compared to $+1.0$ for a direct block. This created a perverse incentive: the agent was actively discouraged from using harder puzzles, since direct blocking was strictly dominant in expected reward. The legacy schedule also used a single aggregate honeypot trigger probability (0.60) for all bot tiers rather than tier-stratified rates, and assigned a smaller information bonus ($+0.3$). The full legacy mapping is shown in Table 6.

Table 6. Legacy terminal reward mapping (ablation baseline). Puzzle rewards are shown net of per-action costs (easy: 0.1, medium: 0.3, hard: 0.5). The legacy environment does not distinguish human pass/fail; it assigns a single expected penalty per difficulty.

Scenario	Reward
Puzzle actions (net of action cost)	
Human issued easy puzzle	-0.15
Human issued medium puzzle	-0.45
Human issued hard puzzle	-0.80
Bot passes easy puzzle	-0.50
Bot passes medium puzzle	-0.70
Bot passes hard puzzle	-0.90
Bot caught by easy puzzle	$+0.90$
Bot caught by medium puzzle	$+0.70$
Bot caught by hard puzzle	$+0.50$
Direct actions	
Allow human	$+0.50$
Allow bot	-0.80
Block bot	$+1.00$
Block human	-1.00

The revised schedule (Table 5) addresses these issues by removing per-action costs on puzzles, scaling catch rewards by difficulty ($+0.8/+1.0/+1.2$), and setting the direct block reward ($+0.7$) below all puzzle catches. This makes puzzle-based detection the higher-reward strategy, encouraging evidence gathering over opaque blocking. The penalty for incorrectly blocking a human increases from -1.0 to -1.5, further discouraging false positives. Honeypot trigger probabilities are stratified by bot tier to reflect realistic differences in bot sophistication, and the information bonus increases from $+0.3$ to $+0.5$. The impact of these changes is directly observable in the evaluation results: under legacy rewards, PPO and Soft $\text{PPO}_{\text{noaug}}$ deploy zero puzzles and rely entirely on direct blocking, while under revised rewards all agents shift to near-universal puzzle-based detection (Section 4.1.1). Importantly, the underlying outcome labels (e.g., `bot_blocked_puzzle`, `correct_block`) remain unchanged across

 doi:10.20944/preprints202604.1178.v1

versions; only the numerical reward assignments differ. Furthermore, the observation, state, action, and reward space are all visualized in Figure 5.

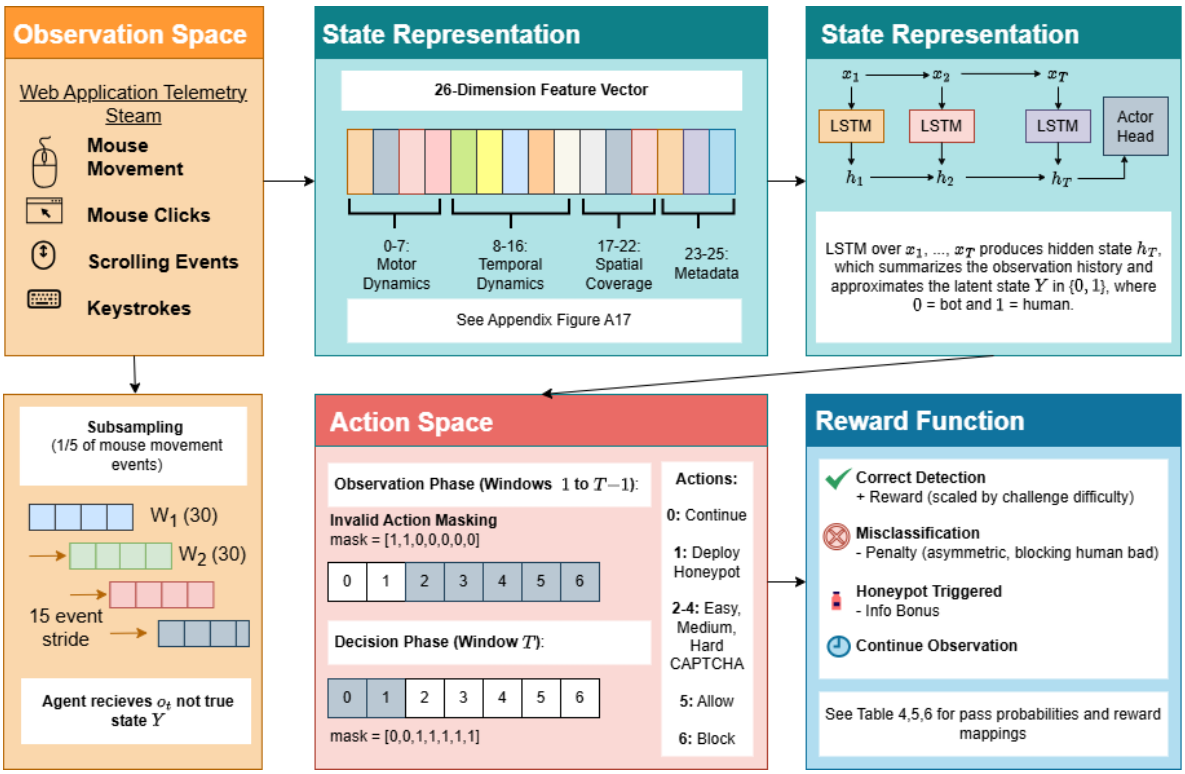

Figure 5. RL-Based Bot-Detection Framework.

3.4.3. Architecture

The architecture we implemented utilizes three main components: LSTM, Actor Head, and Critic Head. Our actor-critic architecture, originating from [41], utilizes an LSTM backbone which enables the RL agent to process sequential observations and accumulate evidence over time. This is well suited for our setting because the user's behavior unfolds temporally and early signals must be integrated with later observations to infer whether the user is a bot or human.

LSTM Backbone

The core of the architecture is a Long Short-Term Memory (LSTM) network [44], which captures long-term dependencies in sequential data through gated memory mechanisms. At each timestep t, the network receives a 26-dimensional feature vector $x_t \in \mathbb{R}^{26}$ representing a window of user interaction telemetry, and updates its hidden and cell states (h_t, c_t) accordingly. Intuitively, the LSTM acts as a memory system that maintains a running summary of user behavior over time. At each step, it decides what past information to retain (forget gate), what new information to incorporate from the current observation (input gate), and what information to expose for decision-making (output gate). This allows the model to retain relevant behavioral patterns while discarding noise. By aggregating information across multiple time steps, the LSTM enables the agent to accumulate evidence over time (e.g., combining early navigation behavior with later typing patterns which helps form a more complete understanding of the user). Thus, the hidden state $h_t \in \mathbb{R}^{128}$ serves as a compressed representation of the observation history. Due to the partial observability of the environment, the LSTM hidden state can be interpreted as an implicit belief representation that summarizes past observations and approximates the underlying latent user state (bot or human) [47].

 doi:10.20944/preprints202604.1178.v1

Actor Head

The actor head defines the policy $\pi_\theta(a_t \mid h_t)$ over the discrete action space. It takes the LSTM hidden state h_t as input and maps it to action scores (logits) using the following layers:

$$h_t \to \text{Linear}(128, 128) \to \tanh \to \text{Linear}(128, 64) \to \tanh \to \text{Linear}(64, 7).$$

These logits are then masked to remove invalid actions and passed through a softmax function to produce a probability distribution over actions. During training, actions are sampled from this distribution to allow exploration, while during evaluation, the highest-probability (greedy) action is selected. Invalid action masking is applied by setting the logits of invalid actions to $-\infty$ before the softmax, ensuring they receive zero probability [46].

Critic Head

The critic estimates how good the current state is by predicting the value function:

$$V_\theta(h_t) = \mathbb{E}_\pi[G_t \mid h_t],$$

It takes the LSTM hidden state h_t as input and maps it through a small neural network:

$$h_t \to \text{Linear}(128, 128) \to \tanh \to \text{Linear}(128, 64) \to \tanh \to \text{Linear}(64, 1).$$

The output is a single value representing the expected future reward from the current state. This value is used as a baseline during training, which helps reduce variance in the updates and makes learning more stable [37].

Shared Representation

The LSTM backbone is shared between the actor and critic. This keeps the model smaller and allows both parts of the network to learn from the same representation of user behavior. Since both the actor and critic rely on similar information (i.e., whether the user is a bot or human), sharing the backbone works well in practice and is commonly used in PPO-based methods [48]. This entire LSTM Actor-Critic architecture is visualized in Figure 6. Furthermore, the parameter count for this architecture is organized in Table 7. The compact architecture (∼130K parameters) is small to avoid overfitting on limited training data while also retaining enough capacity for the sequential classification task at hand.

Table 7. Model Parameter Count.

Component	**Parameters**
LSTM	$4 \times (26 \times 128 + 128 \times 128 + 128) = 80{,}384$
Actor head	$128 \times 128 + 128 + 128 \times 64 + 64 + 64 \times 7 + 7 = 25{,}159$
Critic head	$128 \times 128 + 128 + 128 \times 64 + 64 + 64 \times 1 + 1 = 24{,}385$
Total	**129,928**

 doi:10.20944/preprints202604.1178.v1

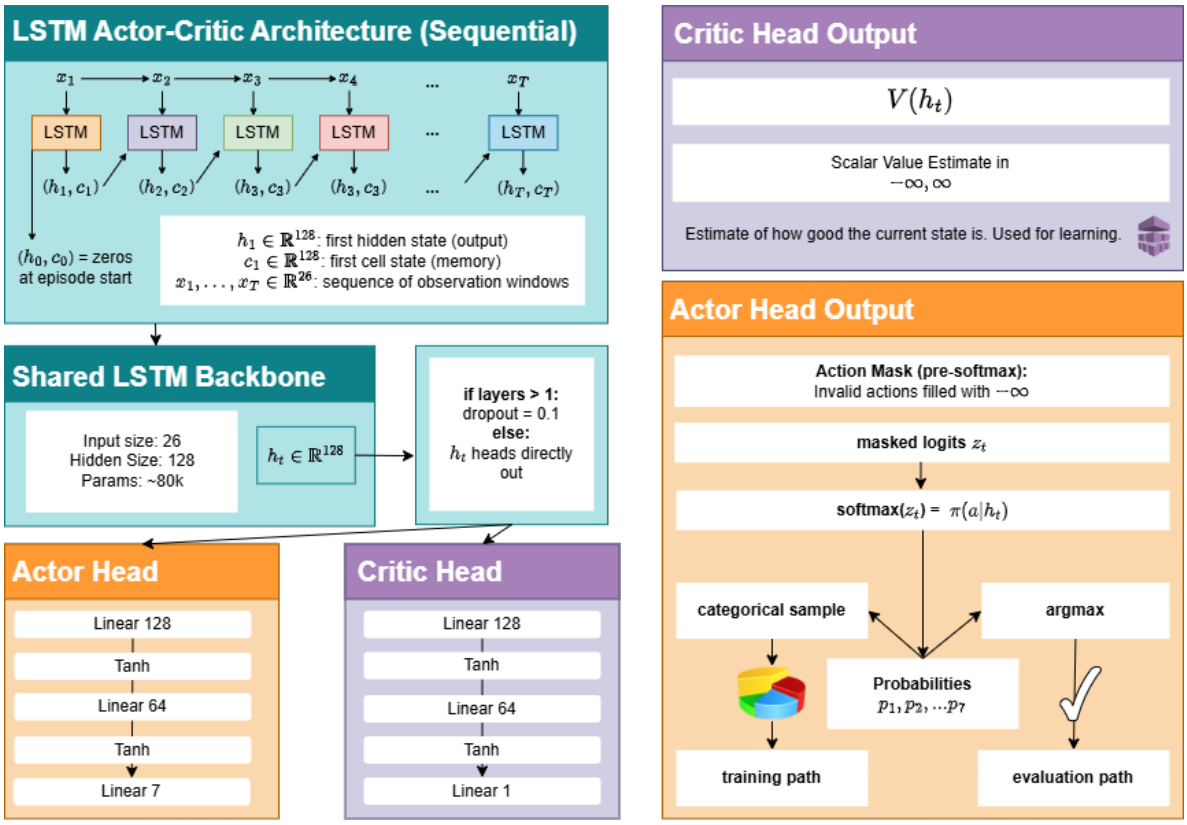

Figure 6. LSTM Actor-Critic architecture.

3.4.4. Training Algorithms

We train three algorithm variants, all sharing the same LSTM architecture and environment. This controlled comparison, illustrated in Figure 7, isolates the effect of the policy optimization method from architectural and environmental factors.

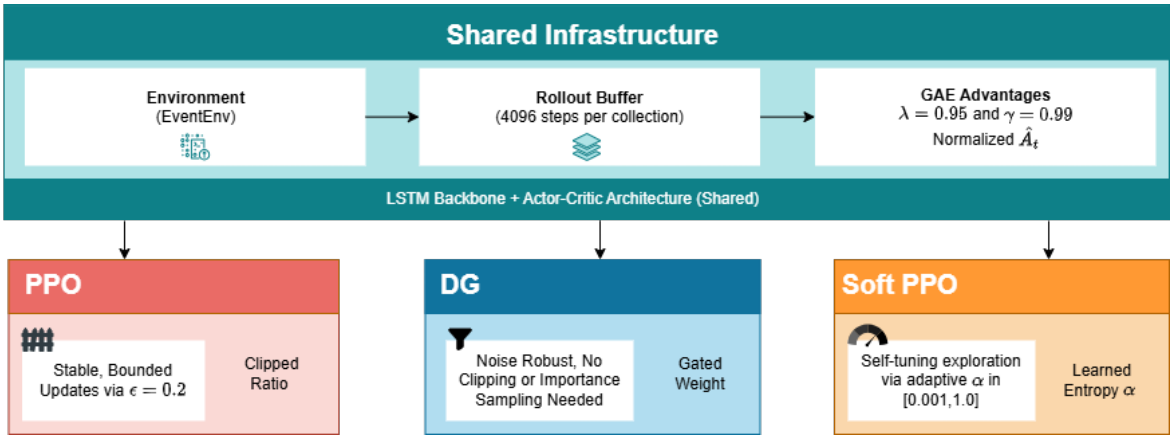

Figure 7. Training Algorithm Comparison.

PPO (Proximal Policy Optimization)

PPO [42] is our primary training algorithm, chosen for its empirical stability and strong performance across a wide range of RL tasks. PPO belongs to the family of policy gradient methods [42] but addresses the instability of vanilla policy gradients through a clipped surrogate objective that constrains how far the policy can change in a single update.

On-policy rollouts of 4,096 steps are collected by executing the current policy in the environment [42]. Each interaction step (transition) stores the observation, action, reward, termination flag, action log-probability, value estimate, and action mask. These stored quantities are required for

 doi:10.20944/preprints202604.1178.v1

computing policy gradient updates, including the likelihood ratio and advantage estimates required for PPO updates. Because the model incorporates an LSTM, the hidden state is reset at the beginning of each episode to prevent information leakage across independent trajectories. The initial hidden state for each sequence is also stored to ensure correct reconstruction of temporal dependencies during training updates. Advantages are computed using GAE [39], where the TD residual (i.e., surprise) is

$$\delta_t = r_t + \gamma V_\theta(h_{t+1}) - V_\theta(h_t)$$

Here, r_t denotes the immediate reward received at timestep t, while $\hat{R}_t$ denotes the discounted return used as the target for value function learning. Accordingly, the advantage is

$$\hat{A}_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}$$

where $\lambda \in [0,1]$ controls the bias–variance tradeoff in advantage estimation [39]. Advantages are normalized across the rollout buffer to stabilize training. The policy is updated by maximizing

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t(\theta)\hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right],$$

where

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid h_t)}{\pi_{\theta_{\text{old}}}(a_t \mid h_t)}$$

Here, θ denotes the parameters of the neural network, including the shared LSTM backbone and the actor and critic heads. The clipping with $\epsilon = 0.2$ prevents excessively large updates and follows standard PPO settings [42]. To stabilize value learning, we use

$$L^V(\theta) = \max\left((V_\theta(h_t) - \hat{R}_t)^2,\ (V_{\text{clip}}(h_t) - \hat{R}_t)^2\right),$$

where

$$V_{\text{clip}}(h_t) = V_{\theta_{\text{old}}}(h_t) + \text{clip}(V_\theta(h_t) - V_{\theta_{\text{old}}}(h_t), -\epsilon, \epsilon).$$

An entropy regularization term encourages exploration:

$$L(\theta) = L^{\text{CLIP}}(\theta) - c_1 L^V(\theta) + c_2 H(\pi_\theta),$$

where $c_1 = 0.5$ and $c_2 = 0.02$. Updates are performed over sequential episode segments to preserve LSTM state consistency. Segments are shuffled across epochs, but transitions within a segment retain temporal order. Gradients are clipped to a maximum norm of 0.5 to prevent instability during LSTM training. The hyperparameters for PPO are highlighted in Table 8.

 doi:10.20944/preprints202604.1178.v1

Table 8. PPO Hyperparameters.

Parameter	Value	Justification
Learning rate	3×10^{-4}	Standard PPO setting [42]
Discount factor γ	0.99	Long-horizon reasoning [37]
GAE λ	0.95	Bias–variance tradeoff [39]
Clip ϵ	0.2	PPO default [42]
Value loss coefficient	0.5	Standard
Entropy coefficient	0.02	Encourages exploration
Max grad norm	0.5	LSTM stability [49]
Rollout steps	4096	On-policy buffer size
Epochs per rollout	4	Standard
Total timesteps	500,000	Convergence budget
Optimizer	Adam	Standard optimizer [42]

DG (Delightful Policy Gradient)

DG [50] is a recent alternative to standard policy gradients that addresses noisy updates and reallocates gradient direction across contexts. Specifically, DG not only reduces variance within a context but also shifts the expected gradient direction across contexts toward a cross-entropy-like objective. The surprisal is

$$\ell_t = -\log \pi_\theta(a_t|h_t),$$

and the gated weight is

$$w_t = \sigma\left(\frac{\hat{A}_t \ell_t}{\eta}\right)$$

The resulting gradient becomes

$$\nabla_\theta J = \mathbb{E}_t\left[w_t \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|h_t)\right]$$

This mechanism emphasizes rare, informative successes while suppressing noisy or uninformative updates. In our CAPTCHA defense setting, this is particularly useful because rare but critical cases, such as complex bots, can have a disproportionate impact on system performance. DG uses only current policy probabilities and does not rely on importance sampling or PPO-style clipping. The hyperparameters for DG are summarized in Table 9.

Table 9. DG Hyperparameters.

Parameter	Value	Justification
Temperature η	1.0	Stable and robust across experiments per [50]

PPO with Adaptive Entropy (Soft PPO)

We extend PPO [42] by learning an entropy coefficient α to automatically balance exploration and exploitation, inspired by Soft Actor-Critic (SAC) [51]. The target entropy is defined as

$$H^* = \rho \log |\mathcal{A}|,$$

and the policy objective becomes

$$L = L^{\text{CLIP}} - c_1 L^V - \alpha H(\pi_\theta).$$

 doi:10.20944/preprints202604.1178.v1

The entropy coefficient is updated via

$$L_\alpha = \alpha\big(H(\pi_\theta) - H^*\big),$$

encouraging higher entropy when the policy becomes too deterministic and reducing entropy when it is overly stochastic. This allows the agent to balance exploration and decision-making automatically without manual tuning. In our implementation, α is optimized in log-space using Adam and constrained to a fixed range for stability. The hyperparameters are summarized in Table 10.

Table 10. Adaptive Entropy PPO Hyperparameters.

Parameter	Value	Justification
Target entropy ratio	0.5	SAC-style entropy scaling [51,52]
α learning rate	3×10^{-4}	Standard RL learning rate [42,51]
Initial $\log \alpha$	-2.0	Common entropy initialization [52]
α range	$[0.001, 1.0]$	Stability constraint (ours)
α optimiser	Adam	Standard optimizer [42,51]

3.4.5. Data Splitting and Training Protocol

All experiments use a stratified 70/15/15 train/validation/test split, partitioned at the session level with a fixed random seed ($s = 42$). Human and bot sessions are split independently to maintain class proportions across all three sets. The partition is determined solely by the original sessions; augmented copies are assigned to the same split as their source session, preventing data leakage across splits. The training set contains $N_{\text{train}} = 447$ original sessions (142 human, 305 bot).

Each algorithm is trained in two configurations: **noaug** (original sessions only) and **advaug** (original sessions plus adversarially augmented bot sessions from the humanization pipeline described in Section 3.3). The advaug training set additionally includes $6 \times |\mathcal{B}\text{train}|$ augmented bot sessions (2 copies $\times$ 3 difficulty levels per training bot), where $\mathcal{B}\text{train}$ denotes the 305 bot sessions in the training split. Evaluation is always conducted with augmented sessions included in the test set, so that all six models (PPO, DG, and Soft PPO, each $\times$ noaug, advaug) are tested against both original and humanized bot behaviors. All six models share the same underlying data split.

Training Loop

Training follows the standard on-policy rollout collection and update cycle. In each iteration, the agent interacts with the environment for 4,096 steps, collecting a rollout buffer of transitions. At each step, the environment samples a session uniformly at random from the training split (with on-the-fly augmentation applied stochastically). The LSTM hidden state is reset at the beginning of each episode. For each transition, the agent receives a 26-dimensional observation window, selects an action according to its current policy π_θ under the applicable action mask, and records the observation, action, reward, termination flag, action log-probability, value estimate, and action mask. Episodes that terminate mid-rollout (via a terminal action or truncation) contribute their final outcome to per-rollout statistics; the remaining budget of steps continues with a new session. If the rollout ends mid-episode, the value of the final observation is bootstrapped to complete the return estimate.

After each rollout, advantages are computed using Generalized Advantage Estimation (GAE; Section 3.4.4) and normalized across the buffer. The policy is then updated over 4 epochs. Within each epoch, the buffer is split into episode-level segments, which are shuffled across epochs to reduce correlation, but transitions within each segment retain their temporal order to preserve LSTM hidden-state consistency. Each segment is processed by passing its full observation sequence through the LSTM from the recorded initial hidden state (h_0, c_0), recomputing logits and value estimates, and applying the clipped surrogate loss. Gradients are clipped to a maximum norm of 0.5 to prevent exploding gradients in the recurrent backbone [49]. This process repeats for $\lfloor 500{,}000/4{,}096 \rfloor = 122$

 doi:10.20944/preprints202604.1178.v1

rollout iterations. Checkpoints are saved every 10 rollouts alongside a deterministic validation pass over 100 episodes (with augmentation disabled), and the final checkpoint is used for evaluation.

On-the-Fly Stochastic Augmentation

To mitigate overfitting on our limited dataset, we apply stochastic augmentation to every training episode at sampling time. Unlike the adversarial augmentation pipeline (Section 3.3), which pre-generates static humanized copies, on-the-fly augmentation applies random perturbations each time a session is drawn, ensuring the agent never observes the identical input twice across training. This is applied independently of and in addition to adversarial augmentation. Three perturbation types are applied with probability $p_{\text{aug}} = 0.5$ per episode:

- **Position noise:** Gaussian noise $\mathcal{N}(0, \sigma_{\text{pos}})$ is added to all mouse and click coordinates, with $\sigma_{\text{pos}} = 15\,\text{px}$ for bot sessions and $\sigma_{\text{pos}} = 5\,\text{px}$ for human sessions. The lighter human perturbation preserves the natural structure of genuine mouse trajectories while still providing regularization.
- **Timing jitter:** Gaussian noise $\mathcal{N}(0, \sigma_t)$ is added to event timestamps, with $\sigma_t = 30\,\text{ms}$ for bots and $\sigma_t = 15\,\text{ms}$ for humans. This prevents the agent from relying on exact inter-event timing, which can vary across hardware and network conditions.
- **Speed warping:** All timestamps are scaled by a uniform random factor $w \sim \mathcal{U}(w_{\min}, w_{\max})$, with $(w_{\min}, w_{\max}) = (0.7, 1.4)$ for bots and $(0.85, 1.15)$ for humans. This simulates variation in overall interaction pace (e.g., a user who is hurried versus one who is browsing slowly/carefree).

Bot sessions receive stronger perturbations than human sessions by design. The asymmetry serves two purposes: (1) it broadens the distribution of bot behaviors the agent encounters during training, improving generalization to unseen bot variants, and (2) it preserves the subtler statistical structure of human sessions, which the agent must learn to recognize as legitimate. On-the-fly stochastic augmentation is disabled during validation and test evaluation to ensure metrics reflect performance on the fixed evaluation distribution. This does not affect whether pre-generated adversarially augmented bot sessions are included in the test split.

3.4.6. Inference and Pseudo-Online Training

At inference time, the agent processes a user session sequentially by constructing a telemetry timeline and passing observation windows through the LSTM-based policy. The LSTM hidden state is reset at the start of each session. The final action selected by the agent determines the intervention shown to the user (e.g., allow, block, or issue a challenge). To enable adaptation to evolving bot behaviors, we implement a pseudo-online training mechanism based on single-session PPO updates. After a session completes and a ground-truth label is obtained via the confirmation endpoint, the full interaction trajectory is replayed through the network with the same action masking used during offline training. The resulting transitions are stored in a single-episode rollout buffer, advantages are computed using the same GAE procedure described in Section 3.4.4, and a PPO update is performed. To ensure stability, the online learning rate is reduced to 60% of the offline rate (1.8×10^{-4}) and updates run for 3 optimization epochs instead of 4. The system evaluates the session both before and after the update, logging whether the agent's decision improved, regressed, or remained unchanged, providing an auditable record of online adaptation. The updated checkpoint is saved immediately after each update.

3.4.7. Evaluation Protocol

To assess performance, all models are evaluated on the held-out test split using deterministic (greedy) policy execution. We evaluate under both the legacy and revised reward structures (Section 3.4.2), producing two independent sets of results across all six agent configurations.

Each evaluation episode corresponds to a single user session processed sequentially through the LSTM under the same action masking used during training. To account for environment stochasticity (puzzle pass rates, honeypot triggers), we run 500 episodes per agent across 5 independent random

 doi:10.20944/preprints202604.1178.v1

seeds and report mean $\pm$ standard deviation. Episodes are sampled with replacement from the unique test sessions, ensuring each session is evaluated multiple times under different stochastic outcomes for stable metric estimates.

3.4.8. Evaluation Metrics

We report the following metrics capturing both classification performance and reinforcement learning behavior:

- **Accuracy, Precision, Recall, F1:** Standard binary classification metrics. Precision measures the fraction of bot predictions that are correct (minimizing disruption to humans), while recall measures the fraction of actual bots detected.
- **Per-tier detection rate:** Recall stratified across the five adversarial tiers (T1: Commodity through T5: LLM-Powered). This is the primary metric for evaluating adversarial resilience as bot sophistication increases.
- **Average reward:** Mean episodic return under the reward function defined in Section 3.4.2, reflecting the RL objective including correct classification, user friction costs, and information-gathering bonuses.
- **Honeypot usage:** Fraction of episodes in which the agent deploys at least one honeypot, capturing the agent's preference for evidence gathering over direct action.

3.5. Classifier

In addition to the RL agent, we developed a supervised machine learning classifier that serves as a critical benchmark. While the RL agent makes sequential decisions over sliding windows of behavioral telemetry, the classifier takes a holistic approach by analyzing an entire session's telemetry after it concludes. It produces a single human-likelihood score $\hat{p} \in [0, 1]$, where $\hat{p} = 1.0$ indicates very high confidence that the session belongs to a human and $\hat{p} = 0.0$ indicates very high confidence that the session belongs to a bot.

3.5.1. Feature Engineering

Raw JSON telemetry collected during a user session is condensed into a 39-dimensional feature vector organized across eight behavioral groups. Each group targets a distinct aspect of user interaction, chosen based on the behavioral differences between humans and bots identified in Section 2.3.

 doi:10.20944/preprints202604.1178.v1

Table 11. Session-Level Feature Groups (39 dimensions).

Group	Dim	Features	Rationale
Mouse dynamics	9	count, avg/std speed, avg/std Δt, direction-change ratio, straightness, jitter ratio, acceleration std	Bots move in straight lines at constant speed; humans exhibit curved paths, hand tremor (jitter), and variable acceleration.
Click patterns	4	count, avg/std inter-click interval, interactive-element ratio	Bots click at regular intervals and may miss interactive targets.
Keystroke timing	8	count, avg/std inter-key interval, unique fields, field-switch ratio, rhythm regularity (CV), avg/std hold duration	Bots type with uniform timing; Selenium fires near-zero hold durations ($\sim$1 ms vs. human $\sim$80–200 ms).
Scroll behavior	6	count, avg/std Δy, total $\lvert\Delta y\rvert$, avg speed, direction-change ratio	Bots scroll monotonically; humans reverse direction frequently.
Session-level	1	total duration	Bots complete sessions significantly faster than humans.
Event-type ratios	4	mouse / click / key / scroll ratios	Bots produce abnormal event distributions (e.g., no mouse events for tab-only Selenium bots).
Global timing	3	mean / variance / min inter-event Δt	Bots generate unnaturally regular or fast event streams.
Spatial coverage	4	unique x positions, unique y positions, x range, y range	Bots visit fewer unique screen positions with narrower spatial extent.

This 39-dimensional representation aggregates each session into fixed-length statistics that summarize *how* the user interacted rather than what they did, making the classifier robust to differences in page layout or task content. The feature set is intentionally compact and human-interpretable so that XGBoost can train reliably without overfitting and so that feature-importance analysis remains meaningful for research interpretability.

3.5.2. Model Architecture

We selected XGBoost (eXtreme Gradient Boosting) [53] as the classification algorithm based on four considerations. First, the 39 input features are aggregate session-level statistics (i.e., tabular data), a domain in which gradient-boosted decision trees consistently match or outperform neural network approaches [54]. Second, XGBoost handles small labeled datasets effectively through built-in L_1/L_2 regularization and early stopping. Third, the model provides interpretable feature-importance scores via gain-based splitting, which aids in understanding which behavioral signals are most discriminative. Fourth, XGBoost achieves sub-millisecond CPU inference, requiring no GPU, which is critical for real-time deployment alongside the RL agent.

XGBoost constructs an ensemble of K regression trees, where the prediction for session i is given by

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \tag{3}$$

 doi:10.20944/preprints202604.1178.v1

where $\mathbf{x}_i \in \mathbb{R}^{39}$ is the feature vector and $\mathcal{F}$ denotes the space of CART regression trees. The objective function minimized at each boosting round is

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} \ell\Big(y_i,\ \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\Big) + \Omega(f_t), \tag{4}$$

where ℓ is the binary cross-entropy (log loss) defined as

$$\ell(y, \hat{p}) = -\big[y \log \hat{p} + (1 - y) \log(1 - \hat{p})\big], \tag{5}$$

and $\Omega(f_t)$ is the regularization term

$$\Omega(f) = \gamma\, T + \tfrac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|, \tag{6}$$

with T the number of leaves, w_j the leaf weights, and γ, λ, α the complexity, L_2, and L_1 regularization coefficients, respectively [53].

The model is configured with the hyperparameters listed in Table 12. Regularization is intentionally strong ($\alpha = 0.3$, $\lambda = 2.0$, $\gamma = 0.3$, `min_child_weight` $= 5$) to prioritize generalization over memorization on a small dataset.

Table 12. XGBoost Hyperparameter Configuration.

Parameter	**Value**
Number of estimators (K)	200
Max depth	3
Learning rate (η)	0.05
Row subsampling (`subsample`)	0.7
Column subsampling (`colsample_bytree`)	0.7
Min child weight	5
L_1 regularization (α)	0.3
L_2 regularization (λ)	2.0
Min split loss (γ)	0.3
Early stopping rounds	20

3.5.3. Training Pipeline

The classifier is trained with a regularization-heavy pipeline designed to generalize from a small labeled corpus and to resist adversarial mimicry of human behavior.

Data Split

A stratified 70/30 train/test split is applied at the session level, preserving the class distribution between human ($y = 1$) and bot ($y = 0$) sessions. Pre-generated augmented copies (Section 3.3) are appended to the *train* split only, never the test split, so that evaluation reflects the real-world distribution and no augmented sample leaks across the boundary.

Feature Standardization

A `StandardScaler` is fit on the training features and applied to both splits:

$$x_f^{\text{scaled}} = \frac{x_f - \hat{\mu}_f}{\hat{\sigma}_f}, \tag{7}$$

 doi:10.20944/preprints202604.1178.v1

where $\hat{\mu}_f$ and $\hat{\sigma}_f$ are the training-set mean and standard deviation for feature f. This ensures the subsequent noise augmentation is uniform across features regardless of their original scale.

Feature-Space Adversarial Augmentation

For each bot sample, $n_{\text{adv}} = 2$ humanized copies are generated by blending the bot's standardized feature vector toward the mean of the human samples with a per-sample random factor $\beta \sim \mathcal{U}(0.2, 0.6)$, then adding Gaussian noise scaled by the human-feature standard deviation ($\sigma_{\text{adv}} = 0.3$). These copies retain the bot label, forcing the classifier to look beyond surface-level differences.

Feature Noise Augmentation

To prevent overfitting to exact feature values, the training set is duplicated into $n_{\text{copies}} = 3$ noisy versions, each perturbed by Gaussian noise:

$$\mathbf{x}_i' = \mathbf{x}_i + \boldsymbol{\eta}, \quad \eta_f \sim \mathcal{N}(0,\ (\sigma_n \cdot \hat{\sigma}_f)^2), \tag{8}$$

where $\sigma_n = 0.5$ controls the noise scale relative to each feature's standard deviation.

Label Smoothing

Inspired by the regularization principle behind label smoothing [55], we reduce the influence of every training sample by scaling its weight:

$$w_i = 1 - \alpha_{\text{ls}}, \tag{9}$$

with $\alpha_{\text{ls}} = 0.05$, so that each sample's contribution to the loss is multiplied by 0.95. This prevents the model from over-committing to any single training example, producing softer probability estimates without modifying the binary labels themselves.

Class Imbalance Handling

To account for potential class imbalance, XGBoost's `scale_pos_weight` is set to the ratio of negative to positive samples:

$$w_{\text{pos}} = \frac{N_{\text{bot}}}{N_{\text{human}}}, \tag{10}$$

ensuring that the loss contribution of each class is balanced during training.

Optuna Hyperparameter Tuning

An optional automated tuning stage uses Optuna [56] with 5-fold stratified cross-validation and ROC-AUC as the optimization objective. The search space covers XGBoost regularization parameters (`max_depth`, `min_child_weight`, η, `subsample`, `colsample_bytree`, α, λ, γ) as well as the augmentation hyperparameters (σ_n, n_{copies}, α_{ls}, n_{adv}, σ_{adv}). The resulting configuration is then used to retrain the final model on the full training split. The trained model exposes `human_score(x)`, returning a probability in $[0, 1]$ that serves as an interpretable session-level human likelihood score.

3.5.4. Evaluation

The classifier is evaluated on the held-out test set using a comprehensive set of metrics: accuracy, precision, recall, F_1 score, and area under the receiver operating characteristic curve (ROC-AUC). The predicted human-likelihood score $\hat{p}$ is compared against a default decision threshold of $\tau = 0.5$:

$$\hat{y}_i = \begin{cases} 1 \text{ (human)} & \text{if } \hat{p}_i \geq \tau, \\ 0 \text{ (bot)} & \text{otherwise.} \end{cases} \tag{11}$$

In addition to aggregate metrics, we report:

 doi:10.20944/preprints202604.1178.v1

- **Score distribution analysis:** Histograms of $\hat{p}$ for human and bot sessions, assessing the separation between class distributions.
- **Feature importance ranking:** Gain-based importance from XGBoost, identifying which behavioral signals contribute most to classification decisions.
- **Confusion matrix:** Visualizing the trade-off between false positives (humans incorrectly blocked) and false negatives (bots incorrectly allowed), which have asymmetric costs in a CAPTCHA deployment setting, since blocking a legitimate user has greater consequences than admitting a bot.

4. Results

4.1. Reinforcement Learning (RL) Results

For our results, we explore two reward structures as mentioned in Section 3.4.2. The first is the legacy (i.e., original) reward mapping highlighted in Table 6. The second is the revised structure described in Table 5. Furthermore, we evaluate six agent configurations: three algorithms (PPO, DG, and Soft PPO as described in Section 3.4.4) each trained in two data regimes. The base variant (e.g., PPO) trains on original sessions only, while the augmented variant (e.g., PPO+Aug) additionally includes adversarially augmented bot sessions generated by the humanization pipeline described in Section 3.3.

4.1.1. Overall RL Classification Performance

Tables 13 and 14 report RL classification performance for all six agent configurations under the legacy and revised reward structures on the held-out test split, including adversarially augmented bot sessions. Under the legacy reward structure (Table 13), Soft PPO without augmentation achieves the highest accuracy (0.988 ± 0.003) and F1 (0.987 ± 0.003), while DG variants and Soft PPO maintain perfect precision (1.000). PPO variants are the only agents to produce false positives, with PPO noaug dropping to 0.936 precision, suggesting that aggressive entropy decay yields confident but occasionally miscalibrated decisions. Honeypot usage differs sharply across methods: Soft PPO deploys honeypots in 100% of episodes, DG noaug in 98.4%, and PPO noaug in only 53.0%. This pattern is consistent with Soft PPO's higher-entropy exploration strategy, which favors additional evidence gathering before terminal decisions. Despite these behavioral differences, performance is tightly clustered, with even the weakest configuration (PPO noaug) achieving an F1 of 0.940. This suggests that the legacy reward landscape is relatively flat: despite modest differences in terminal-action preferences, all algorithms converge to broadly similar overall behavior and achieve closely clustered classification performance.

 doi:10.20944/preprints202604.1178.v1

Table 13. RL Classification Performance on Test Split (Mean ± Std over 5 Seeds) - Legacy Reward Structure.

Metric	PPO	PPO+Aug	DG	DG+Aug	Soft PPO	Soft PPO+Aug
Accuracy	0.940 ± 0.011	0.974 ± 0.008	0.958 ± 0.010	0.951 ± 0.008	**0.988 ± 0.003**	0.964 ± 0.006
Precision	0.936 ± 0.015	0.973 ± 0.007	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
Recall	0.944 ± 0.010	**0.975 ± 0.014**	0.912 ± 0.021	0.903 ± 0.013	0.975 ± 0.006	0.928 ± 0.012
F1 Score	0.940 ± 0.011	0.974 ± 0.008	0.954 ± 0.012	0.949 ± 0.007	**0.987 ± 0.003**	0.963 ± 0.006
Avg Reward	0.701 ± 0.031	0.773 ± 0.015	**0.962 ± 0.039**	0.828 ± 0.019	0.802 ± 0.020	0.753 ± 0.017
Honeypot %	0.530 ± 0.027	0.618 ± 0.014	0.984 ± 0.008	0.675 ± 0.022	**1.000 ± 0.000**	0.804 ± 0.017

Table 14. RL Classification Performance on Test Split (Mean ± Std over 5 Seeds) - Revised Reward Structure.

Metric	PPO	PPO+Aug	DG	DG+Aug	Soft PPO	Soft PPO+Aug
Accuracy	0.953 ± 0.010	0.941 ± 0.009	0.965 ± 0.008	**0.974 ± 0.008**	0.964 ± 0.005	0.967 ± 0.013
Precision	**1.000 ± 0.000**	0.973 ± 0.007	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
Recall	0.903 ± 0.021	0.903 ± 0.018	0.927 ± 0.015	**0.947 ± 0.017**	0.928 ± 0.010	0.933 ± 0.026
F1 Score	0.949 ± 0.011	0.937 ± 0.008	0.962 ± 0.008	**0.973 ± 0.009**	0.963 ± 0.006	0.965 ± 0.014
Avg Reward	0.961 ± 0.027	0.890 ± 0.011	0.977 ± 0.014	1.003 ± 0.034	**1.010 ± 0.020**	1.002 ± 0.028
Honeypot %	0.966 ± 0.005	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	0.904 ± 0.012

The revised reward structure (Table 14) sharpens reward differentiation while preserving strong classification performance. DG+Aug achieves the highest accuracy (0.974 ± 0.008) and F1 (0.973 ± 0.009). After retraining, DG noaug reaches 0.965 accuracy, indicating sensitivity of the delight-gated objective to initialization under the revised rewards. Five of six configurations achieve perfect precision (1.000); only PPO+Aug shows false positives (0.973 precision), suggesting that the stronger human-blocking penalty (-1.5) reduces false positives in most agents. The clearest behavioral change is in honeypot use: PPO noaug rises from 53.0% to 96.6%, and PPO+Aug from 61.8% to 100%. This near-universal adoption suggests that the revised rewards favor evidence gathering over direct blocking. DG and Soft PPO, already high under the legacy schedule, remain near 100%. PPO noaug improves from 0.940 to 0.953 accuracy, while Soft PPO+Aug increases from 0.964 to 0.967, showing that the revised rewards encourage more cautious policies without reducing detection quality.

The average reward row highlights the structural impact of the reward redesign. Under the legacy schedule, rewards span a narrow band (0.701 to 0.962), while the revised schedule raises rewards across the board (0.890 to 1.010), reflecting higher payoffs for puzzle-based detections. Despite this

 doi:10.20944/preprints202604.1178.v1

shift, classification performance remains stable across both structures (all configurations above 0.937 F1), demonstrating a key advantage of the RL formulation: the reward function can be tuned to prioritize operational goals such as preferring puzzle-based verification over opaque blocking without degrading detection quality.

4.1.2. RL Training Dynamics

Figure 8 shows training dynamics for all six agent configurations under the legacy reward structure across 123 rollouts. In cumulative reward (top-left), PPO achieves the highest final return (0.90), followed by DG (0.80), while Soft PPO plateaus at 0.50 because its learned entropy coefficient favors exploration over reward maximization. In validation accuracy (top-right), however, Soft PPO still reaches 0.99–1.00, matching PPO's 1.00 and showing that lower training reward does not imply worse classification. DG noaug is more variable, ending at 0.93, while DG+Aug stabilizes at 1.00. The clearest difference appears in policy entropy (bottom-left): Soft PPO remains near 0.72 throughout training, DG stays moderate at 0.58–0.64, and PPO decays sharply from 0.73 to 0.10–0.15, producing a confident but potentially brittle policy, consistent with Table 13. In correct decision rate (bottom-right), all algorithms converge to 65–75%, with PPO doing so more consistently and Soft PPO remaining more variable due to sustained exploration. Augmented variants closely track their non-augmented counterparts across all panels, indicating that adversarial augmentation increases distributional diversity without destabilizing training.

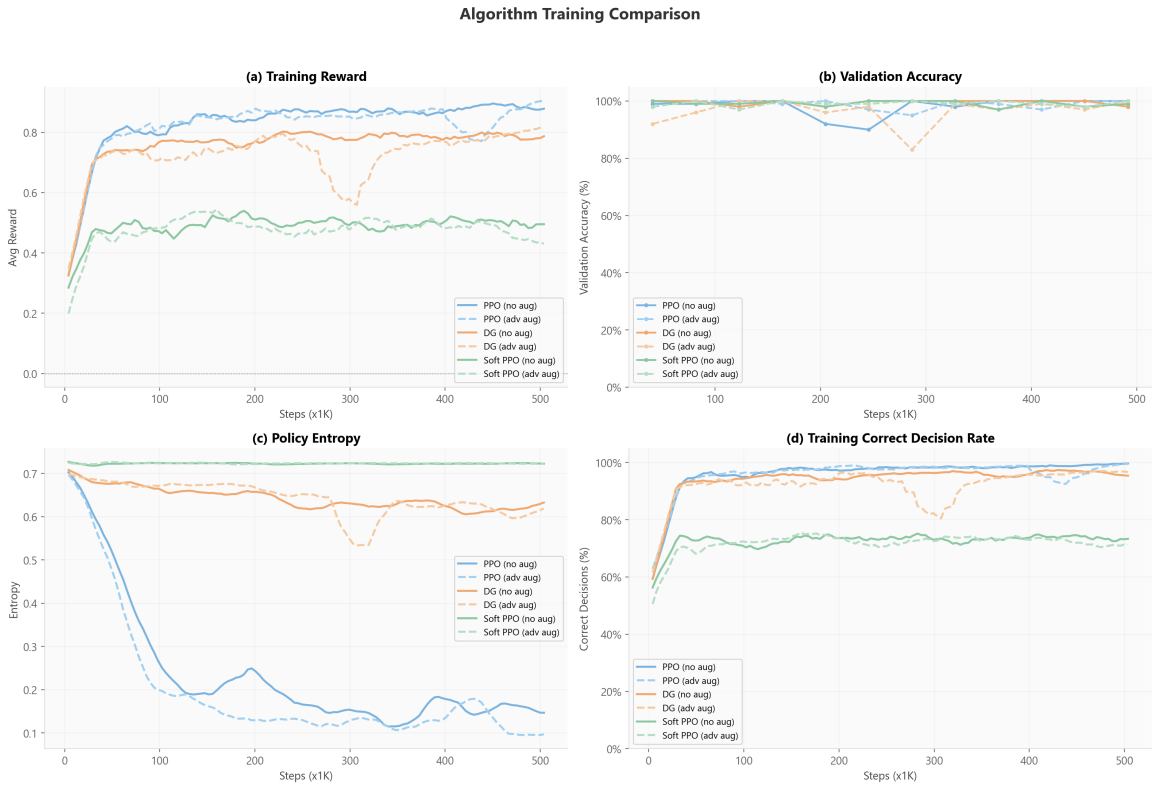

Figure 8. RL Algorithm Training Comparison - Legacy Reward Structure.

Figure 9 shows the same configurations under the revised reward structure, which increases puzzle-catch rewards and penalizes direct blocking more heavily. In the reward panel, DG rises to match PPO at 0.97, up from 0.80 under legacy rewards, while Soft PPO increases from 0.50 to 0.70, indicating that all agents find higher-reward strategies when puzzles are more strongly incentivized. Validation accuracy shows a more nuanced effect: PPO noaug drops to 0.94 from 1.00, and PPO entropy settles at 0.29–0.39 instead of collapsing to 0.10–0.15, suggesting that the revised rewards discourage the aggressive policy compression seen under the legacy schedule. DG remains the least affected, with

 doi:10.20944/preprints202604.1178.v1

validation accuracy of 0.96–0.98 and entropy near 0.65. Soft PPO again reaches 0.97–0.98 validation accuracy despite the lowest reward, reinforcing its tendency to trade reward for generalization. In correct decision rate, PPO and DG rise to about 95%, up from 65–75% under legacy rewards, driven mainly by puzzle-based catches rather than direct blocks, while Soft PPO remains more variable at about 70% because of its broader action distribution. Furthermore, the revised rewards narrow the gap across algorithms, with validation accuracy compressing to 0.94–0.98 from 0.93–1.00 under the legacy schedule. DG also reaches PPO's reward level (0.97) while keeping higher entropy, suggesting the revised structure better rewards information gathering and reduces PPO's earlier advantage from policy compression.

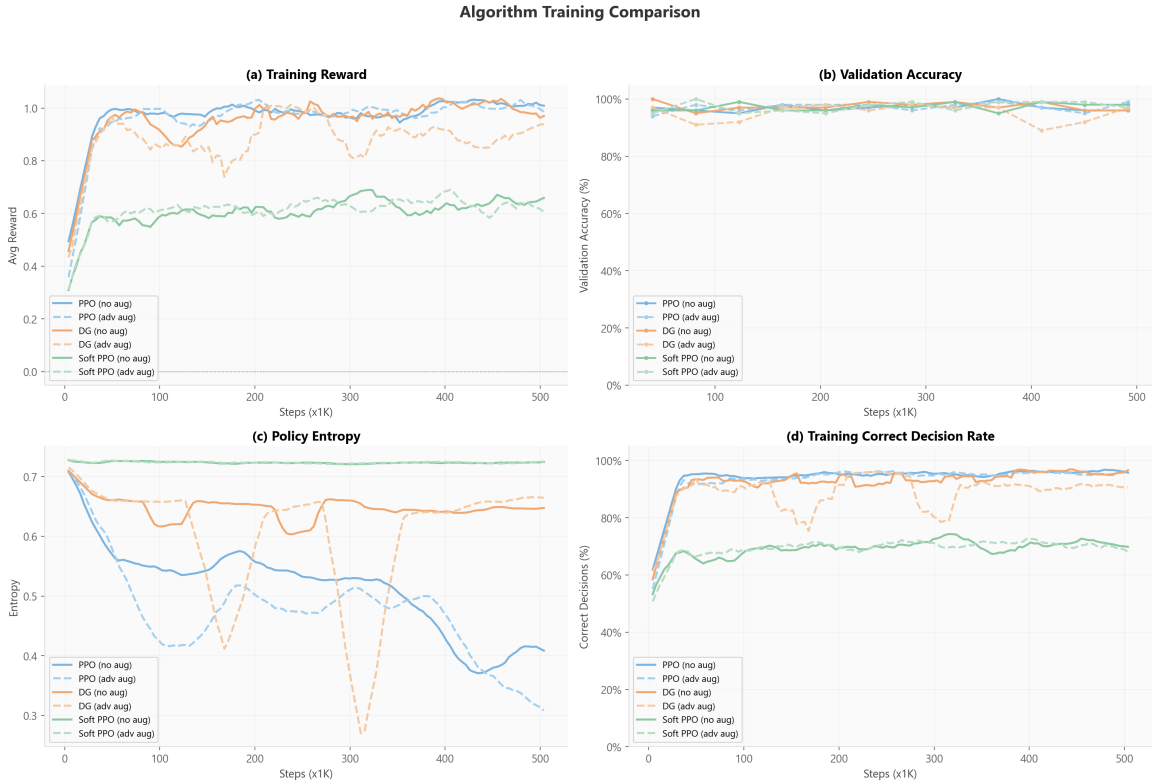

Figure 9. RL Algorithm Training Comparison - Revised Reward Structure.

4.1.3. RL Per-Tier Detection Performance

As shown in Figure 10, for the legacy reward structure, all six agents achieve strong detection on Tier 1 (Commodity), with PPO noaug, Soft PPO noaug, and PPO+Aug at 100% and DG variants at 95.8–98.1%. Tier 4 (Trace-Conditioned) is similarly well-handled: PPO noaug reaches 99.2% $\pm$ 1.7%, Soft PPO noaug and DG+Aug achieve 100%, while DG noaug dips to 95.4% $\pm$ 4.6%. Tier 2 (Careful Automation) stays above 93% across all agents, with PPO+Aug achieving a perfect 100% and Soft PPO noaug close behind at 99.8%. The primary separation between algorithms appears at Tier 3 (Semi-Automated) and Tier 5 (LLM-Powered). At Tier 3, PPO+Aug and Soft PPO noaug both reach 100%, while DG+Aug drops to 84.7% $\pm$ 4.9%, suggesting that the delight-gated objective with augmented data struggles against mixed behavioral patterns at this tier. The most significant differentiation occurs at Tier 5 (LLM-Powered), the most challenging adversary class. PPO noaug achieves the highest detection rate (86.7% $\pm$ 1.4%), followed by Soft PPO noaug (84.3% $\pm$ 6.0%) and PPO+Aug (82.8% $\pm$ 9.8%). DG variants trail at 71.4–72.2%, indicating that the delight-gated objective is least effective against LLM-powered bots whose telemetry closely resembles human patterns. Interestingly, adversarial augmentation does not uniformly improve Tier 5 detection: Soft PPO+Aug drops to 76.2% compared to 84.3% without augmentation, and PPO+Aug shows much higher variance (9.8%)

 doi:10.20944/preprints202604.1178.v1

than PPO noaug (1.4%), suggesting that augmented training data can disrupt the subtle behavioral distinctions needed to identify the most human-like adversaries.

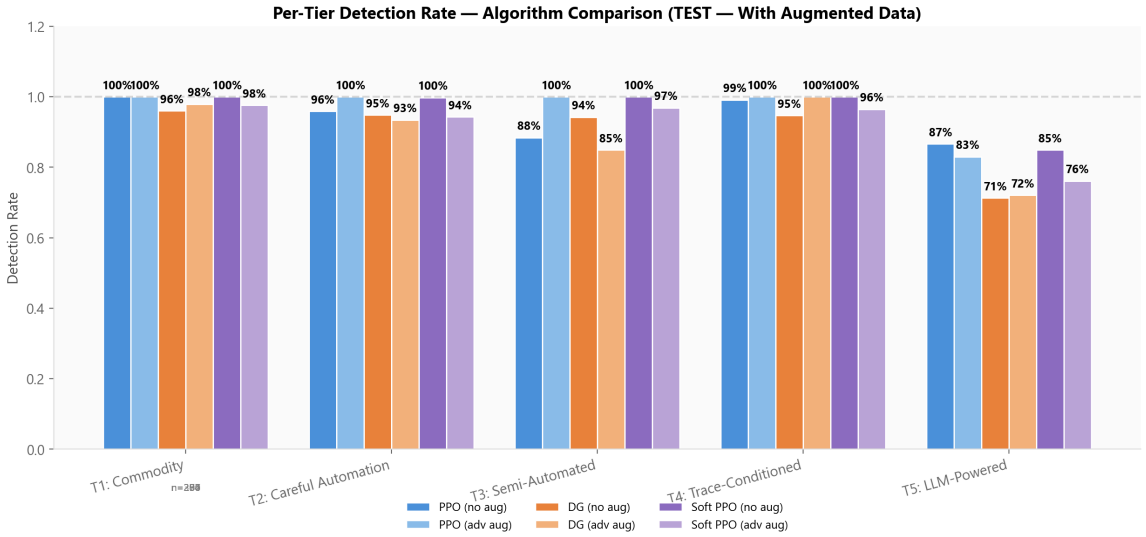

Figure 10. Per-tier Bot Detection Rates Across all Six Agent Configurations on the Test Split - Legacy Reward Structure.

Figure 11 shows per-tier detection under the revised reward structure. Tier 1 (Commodity) remains above 94% for all agents, and Tier 2 (Careful Automation) stays above 90%. The most notable shift appears at Tier 4 (Trace-Conditioned): PPO noaug drops to 78.0% $\pm$ 7.1%, down from 99.2% under legacy rewards, while DG noaug recovers to 96.6% $\pm$ 4.3% (up from 95.4%). This suggests the revised reward's stronger emphasis on puzzle-based detection over direct blocking disproportionately affects PPO's handling of trace-conditioned bots, though PPO+Aug partially recovers to 94.4%. Tier 3 (Semi-Automated) remains strong across the board at 93–96%. The Tier 5 (LLM-Powered) results reveal the most striking change under revised rewards: DG+Aug surges to 96.8% $\pm$ 2.9%, the highest Tier 5 detection rate across both reward structures, dramatically up from 72.2% under legacy rewards. DG noaug similarly improves from 71.4% to 81.4%, suggesting that the revised reward's puzzle-based incentives particularly benefit the delight-gated objective's handling of LLM bots. Soft PPO noaug drops from 84.3% to 74.7%, while Soft PPO+Aug rises from 76.2% to 82.9%, reversing the augmentation penalty observed under legacy rewards. PPO noaug remains stable at 80.9% (down slightly from 86.7%). The reversal of DG's Tier 5 ranking (from worst under legacy to best under revised) demonstrates that reward structure design can have outsized effects on detection of the most sophisticated adversaries.

 doi:10.20944/preprints202604.1178.v1

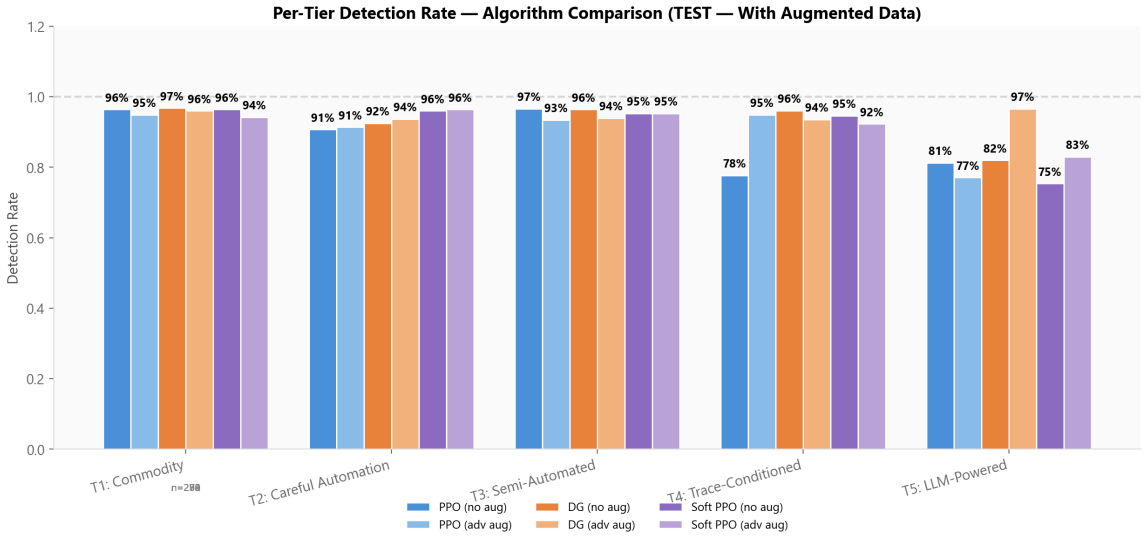

Figure 11. Per-tier Bot Detection Rates Across all Six Agent Configurations on the Test Split - Revised Reward Structure.

4.1.4. RL Decision Behavior and Outcome Distribution

Figure 12 presents the outcome distribution across all six agents under the legacy reward structure (pooled over 5 seeds, 2,500 episodes each). The agents split into two distinct strategies. PPO noaug and PPO+Aug operate almost exclusively with binary allow/block decisions: PPO noaug issues 1,181 correct blocks and 1,168 correct allows with zero puzzle usage, while PPO+Aug similarly relies on direct blocking (1,230 correct blocks) with no puzzles. Both PPO variants are the only agents to produce false positives (81 and 34 respectively), reflecting their low-entropy policies' tendency toward overconfident blocking. In contrast, DG noaug is the heaviest puzzle user, issuing 1,044 hard and 127 medium puzzles, with 1,098 bots caught via puzzles and only 73 slipping through. DG+Aug splits between direct blocking (665 correct blocks) and puzzle-based detection (473 bot catches via puzzle), using 336 hard and 182 medium puzzles. Soft PPO+Aug deploys 639 hard and 66 medium puzzles, placing it between DG's puzzle-heavy and PPO's block-heavy strategies. Soft PPO noaug is notable for achieving the fewest false negatives of any agent (30) while using almost no puzzles (only 12 hard, 4 medium), relying instead on its high-entropy policy to make accurate direct allow/block decisions. No agent ever selects an easy puzzle, indicating that learned policies treat puzzle deployment as an intervention for maximally uncertain sessions. Overall, while most agents under the legacy reward structure still terminate primarily through direct allow/block outcomes, DG distinguishes itself by converting a larger share of suspicious sessions into puzzle-based catches than the other methods, consistent with its objective emphasizing rarer, high-information successes, whereas PPO variants converge more strongly to direct blocking.

 doi:10.20944/preprints202604.1178.v1

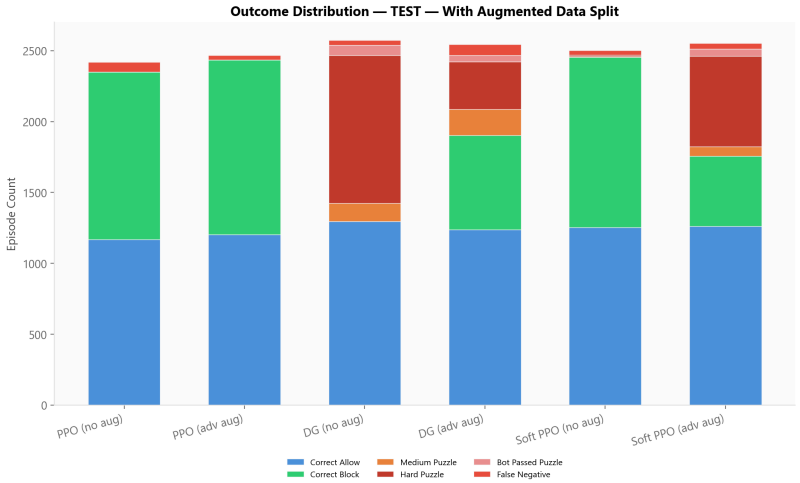

Figure 12. Outcome Distribution with Puzzle Difficulty Breakdown on the Test Split - Legacy Reward Structure.

Figure 13 shows the outcome distribution under the revised reward structure, which increases puzzle-catch rewards and penalizes direct blocking more heavily. The most striking change is the near-universal adoption of puzzle-based detection: PPO noaug, which used zero puzzles under legacy rewards, now issues 1,158 hard puzzles, catching 1,104 bots through challenges rather than direct blocks. Similarly, Soft PPO noaug shifts from 12 puzzles to 1,226 hard puzzles, and DG noaug from 1,044 to 1,171. Every agent now relies primarily on hard puzzles, with correct allows and bot-blocked-via-puzzle as the two dominant outcomes. DG+Aug issues 1,212 hard puzzles with the fewest false negatives (0 false_negative, 64 bot_passed_puzzle), while PPO+Aug is the only agent to still produce false positives (30 fp_puzzle), and the only one to use a mix of hard (1,268) and medium (48) puzzles. The elimination of direct blocking as a strategy across all agents confirms that the revised reward structure successfully steers policies toward evidence-based puzzle challenges. False negatives decrease for DG variants (DG+Aug drops from 123 to 64 total missed bots) but increase for PPO noaug (70 to 118), reflecting the tradeoff between puzzle-based and block-based detection under the new incentive structure.

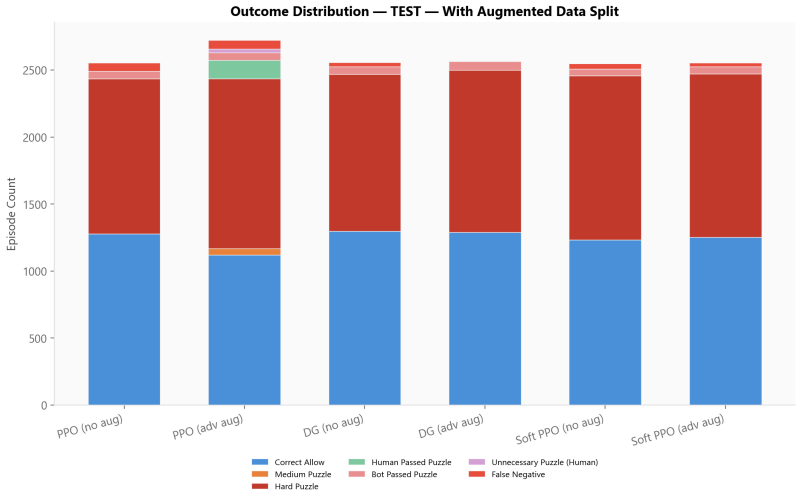

Figure 13. Outcome Distribution with Puzzle Difficulty Breakdown on the Test Split - Revised Reward Structure.

4.1.5. RL Decision Timing

Table 15 reports the average number of observation windows each agent processes before issuing a terminal decision. Across all configurations and both reward structures, human sessions require approximately 21 windows and bot sessions approximately 15–16 windows, with negligible variation

 doi:10.20944/preprints202604.1178.v1

between algorithms. This uniformity confirms that decision timing is governed by session length rather than agent strategy: despite substantial differences in entropy, puzzle usage, and honeypot deployment, all agents observe the full sequence of windows before committing to a terminal action on the final window, as enforced by the action masking described in Section 3.4.2.

Table 15. Average Observation Windows Before Terminal Decision (Pooled over 5 Seeds).

Metric	PPO	PPO+Aug	DG	DG+Aug	Soft PPO	Soft PPO+Aug
Human (Legacy)	21.6	21.6	21.2	21.6	21.3	21.4
Bot (Legacy)	15.4	15.6	15.6	15.5	15.7	15.5
Human (Revised)	21.4	21.3	21.1	21.3	21.1	21.3
Bot (Revised)	15.5	15.8	15.5	15.8	15.8	15.4

4.2. XGBoost Classifier Results

To isolate the contribution of hyperparameter tuning and adversarial augmentation, we trained four configurations of the classifier on the same stratified 70/30 train/test partition. All four share the feature pipeline and label-smoothing setup described in Section 3.5.3 and are evaluated on identical held-out sessions; only Optuna tuning and the inclusion of `HumanProfiler`-augmented bot copies are toggled.

Model Configurations

- **xgb_v1**: Optuna-tuned hyperparameters *and* adversarial augmentation.
- **xgb_v1_noaug**: Optuna-tuned hyperparameters, no augmentation.
- **xgb_v2**: Default `ClassifierConfig` hyperparameters with adversarial augmentation.
- **xgb_v2_noaug**: Default hyperparameters, no augmentation (baseline).

Table 16 reports the headline metrics for all four configurations on the held-out test split at the default decision threshold $\tau = 0.5$.

Table 16. XGBoost classifier evaluation across four configurations ($\tau = 0.5$, test $n = 192$: 61 human, 131 bot). **Aug.** indicates whether `HumanProfiler`-augmented bot sessions were included; **Tuned** indicates Optuna-based hyperparameter selection (5-fold CV). All errors are false negatives (bots misclassified as human); no configuration blocked a legitimate user.

Model	Tuned	Aug.	Accuracy	F1	ROC-AUC	FN
xgb_v1	✓	✓	0.9896	0.9839	1.0000	2
xgb_v1_noaug	✓		0.9948	0.9919	1.0000	1
xgb_v2		✓	0.9948	0.9919	1.0000	1
xgb_v2_noaug			0.9896	0.9839	1.0000	2

Key Observations

1. **Perfect separability across configurations.** All models achieve a ROC-AUC of 1.000, indicating complete separability between human and bot score distributions at some threshold. The 39-dimensional feature representation is sufficient to capture behavioral differences within the current dataset, while regularization (label smoothing, noise augmentation, class balancing, and L_1/L_2 penalties) prevents overfitting.
2. **No false positives.** No configuration misclassifies a human as a bot at $\tau = 0.5$. All errors are false negatives. From a deployment perspective, this is critical: legitimate users are never blocked, and

 doi:10.20944/preprints202604.1178.v1

missed bots can be handled downstream via other interventions. Performance differences are minimal: **xgb_v1_noaug** and **xgb_v2** each miss one bot, while **xgb_v1** and **xgb_v2_noaug** miss two.

3. **Marginal gains from tuning and augmentation.** Hyperparameter tuning and adversarial augmentation each provide slight improvements over the baseline, but their combination does not yield further gains. Optuna selected deeper trees (`max_depth` $= 5$ vs. 3), a moderate learning rate (≈ 0.099), and stronger L_2 regularization (≈ 5.37). However, differences across configurations amount to at most one test sample, suggesting that default settings are already well-aligned with the feature space.

Figures 14–17 show evaluation plots for each configuration. Feature importance analysis indicates that predictive power is distributed across feature groups rather than dominated by a single signal.

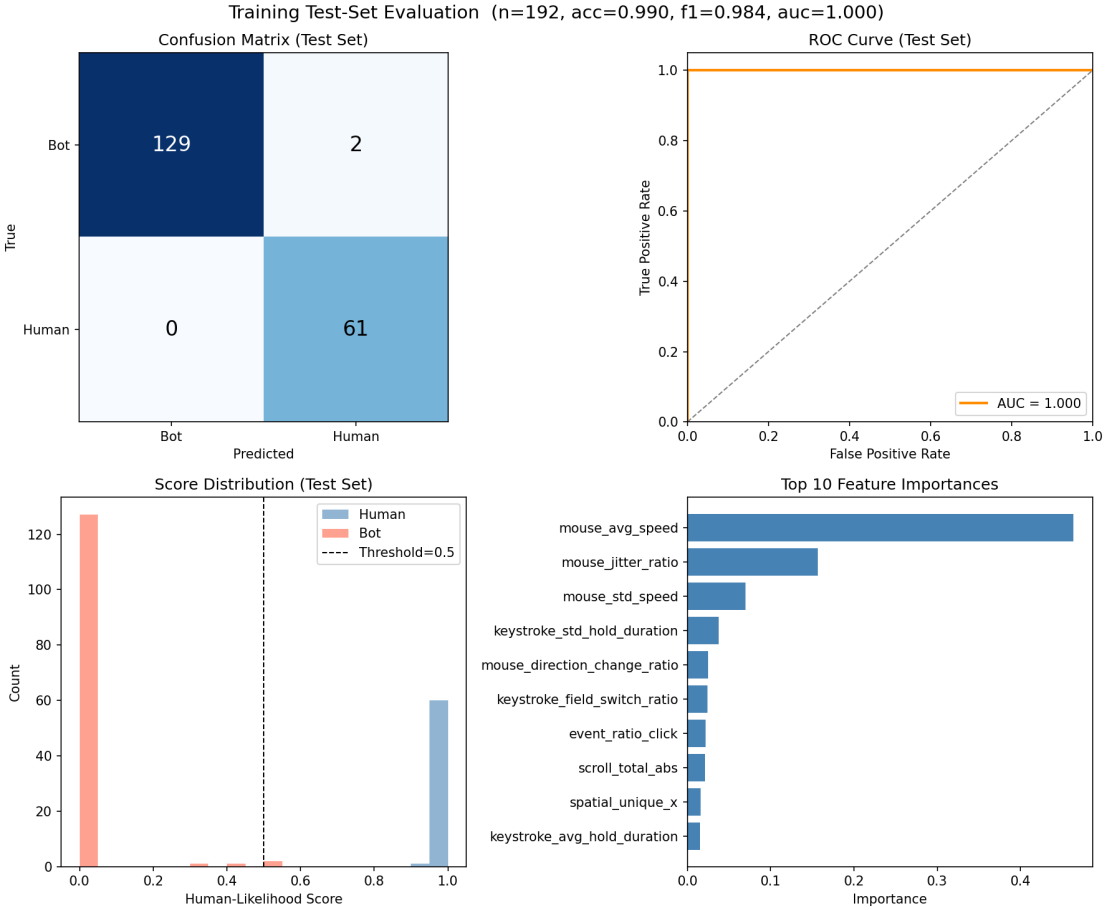

Figure 14. xgb_v1 (tuned + augmentation). 2 false negatives, 0 false positives. Top feature: `mouse_avg_speed`.

 doi:10.20944/preprints202604.1178.v1

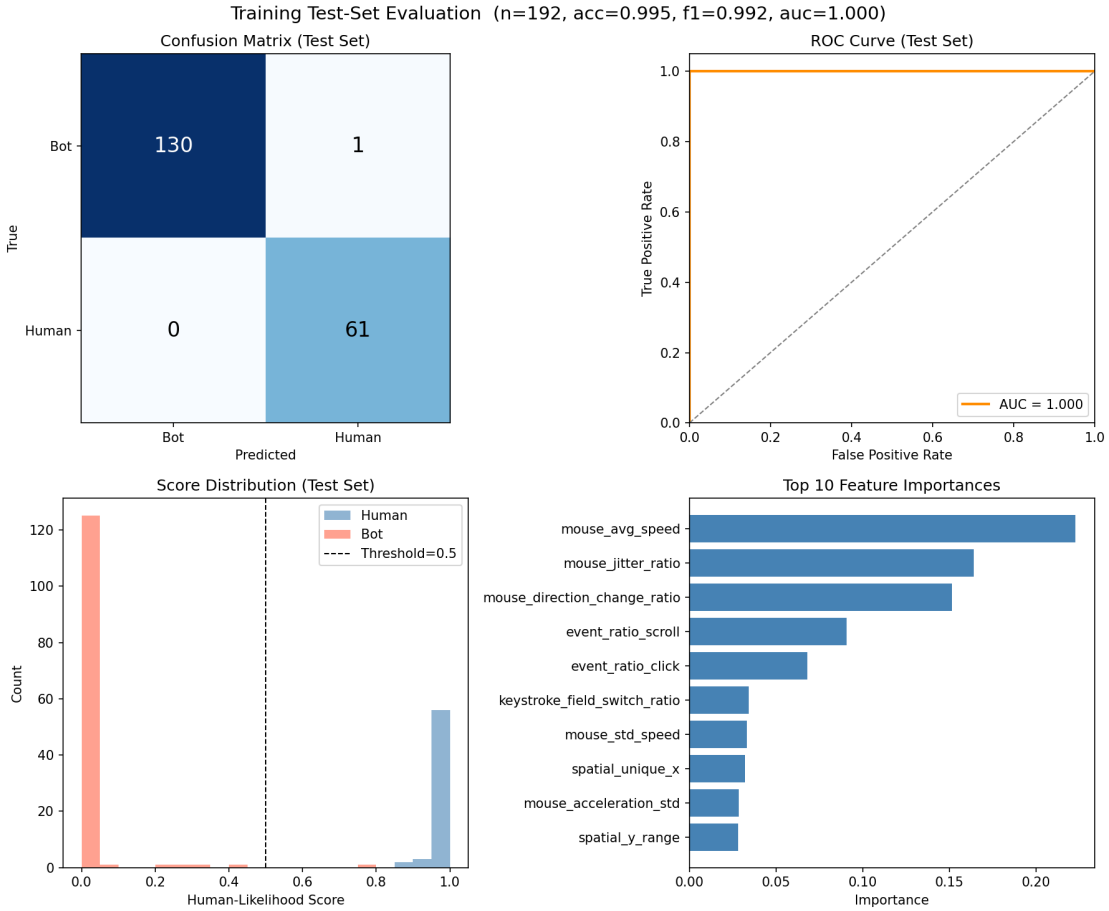

Figure 15. **xgb_v1_noaug** (tuned, no augmentation). 1 false negative, 0 false positives. Top feature: `mouse_avg_speed`.

 doi:10.20944/preprints202604.1178.v1

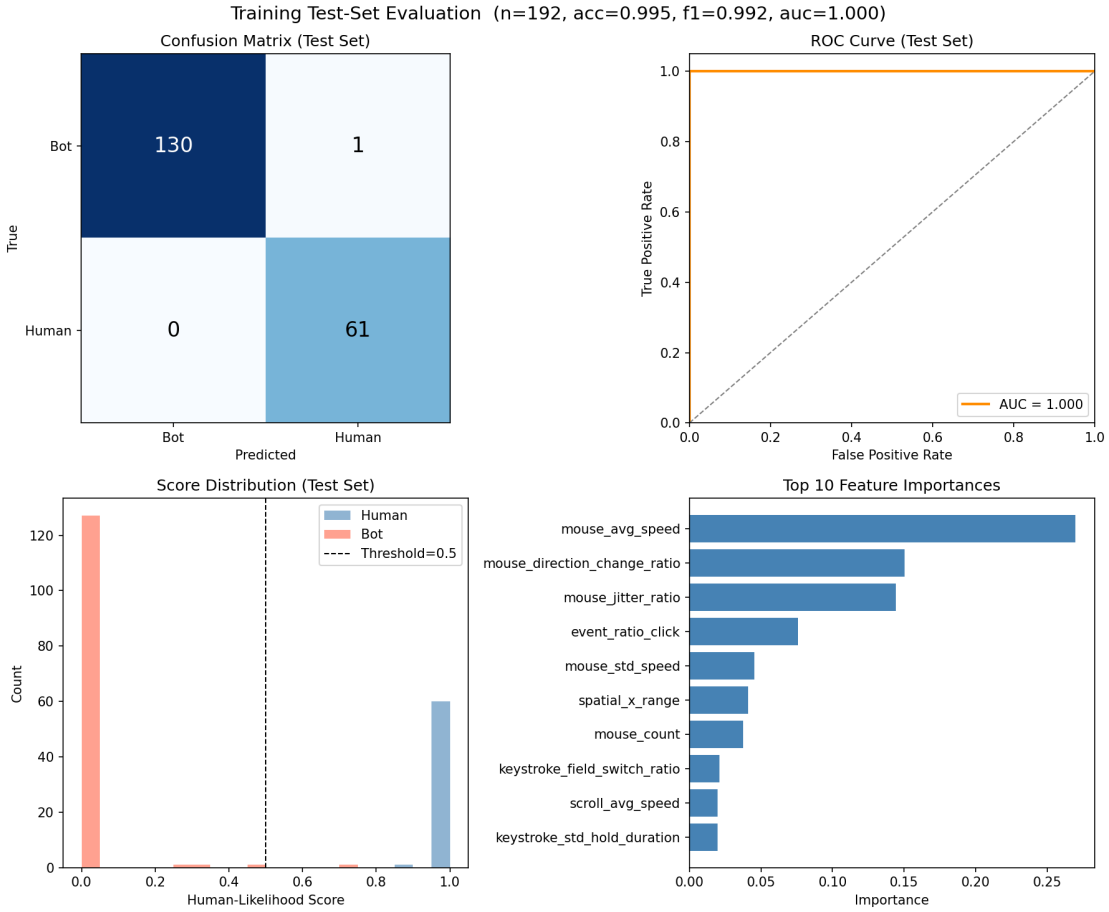

Figure 16. xgb_v2 (default + augmentation). 1 false negative, 0 false positives. Top feature: `mouse_avg_speed`.

 doi:10.20944/preprints202604.1178.v1

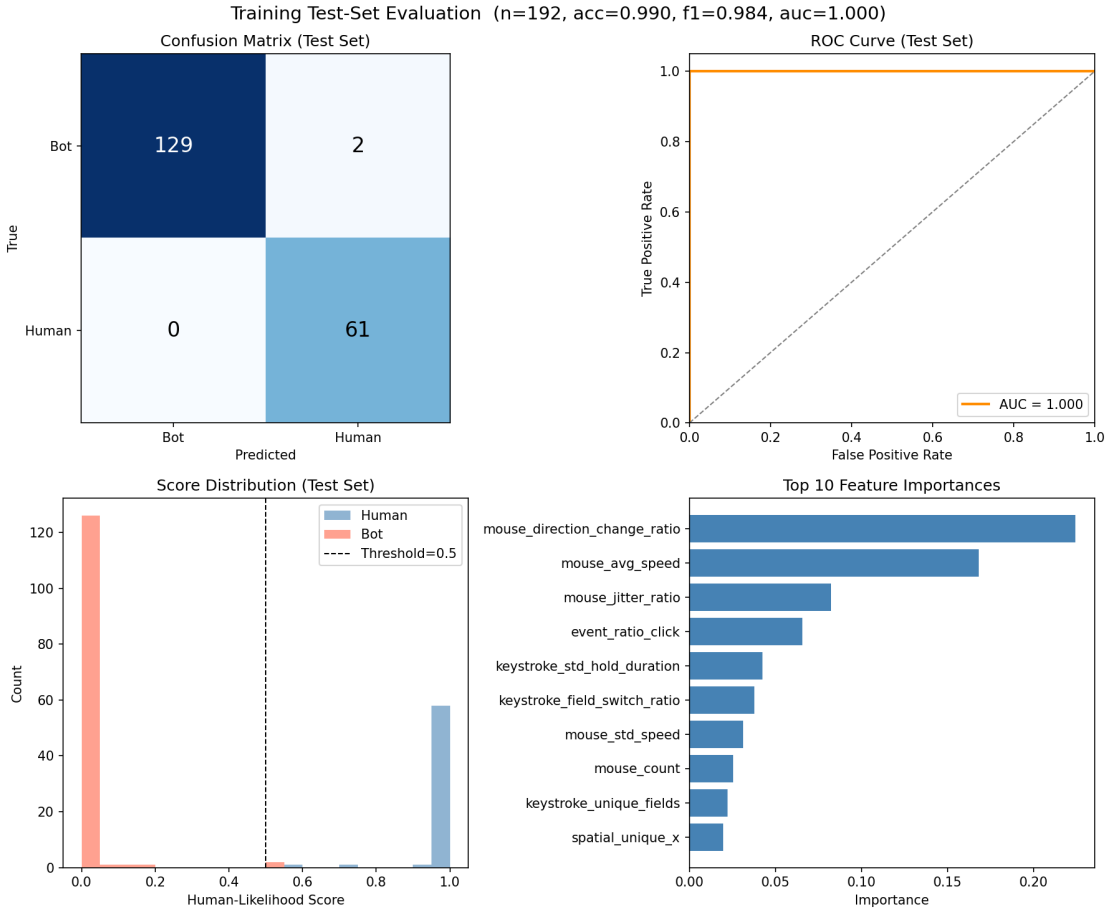

Figure 17. **xgb_v2_noaug** (default, no augmentation). 2 false negatives, 0 false positives. Top feature: `mouse_direction_change_ratio`.

Feature Insights

Mouse dynamics features—including average speed, jitter ratio, and direction-change ratio—consistently rank among the most informative predictors. This aligns with the intuition that human motor behavior exhibits irregular yet bounded patterns that are difficult for simple bots to replicate.

Interpretation

These results represent a performance ceiling for the current bot population rather than a deployment guarantee. Although the test split is held out, it is drawn from the same distribution of bot families as the training data.

5. Discussion and Limitations

5.1. Summary of Key Findings

Across four configurations spanning tuned and untuned hyperparameters and the presence or absence of adversarial augmentation, the XGBoost classifier reaches a ROC-AUC of 1.000 on the held-out test split. All four configurations achieve near-perfect classification at the default decision threshold ($\tau = 0.5$): **xgb_v1_noaug** and **xgb_v2** each misclassify a single bot, while **xgb_v1** and **xgb_v2_noaug** each misclassify two. Crucially, no configuration blocks a legitimate user—every error across all four models is a false negative (a bot scoring above the threshold). Across all four configurations, the top-ranked feature comes from the mouse dynamics group—average speed, jitter ratio, or direction-change ratio—confirming that motor behavior is the most discriminative channel for bot detection in the classifier. The adversarial augmentation pipeline (`HumanProfiler`) contributed to training robustness by forcing the classifier to learn against progressively humanized bot sessions, shifting feature reliance

 doi:10.20944/preprints202604.1178.v1

away from trivially separable artifacts (e.g., Selenium's ∼1 ms key-hold durations) toward deeper behavioral signals that generalize across bot tiers. However, the marginal gains from augmentation are small on the current dataset: the non-augmented models match or exceed the augmented variants, and all four configurations share the same zero-false-positive outcome.

Among the RL agents, Soft PPO noaug achieves the highest accuracy (0.988 $\pm$ 0.003) and F1 (0.987 $\pm$ 0.003) under the legacy reward structure, while under the revised structure DG+Aug takes the lead (0.974 $\pm$ 0.008 accuracy, 0.973 $\pm$ 0.009 F1). Critically, classification quality remains stable across both reward structures, with all configurations exceeding 0.937 F1 and standard deviations $\leq$ 0.014 across five seeds, demonstrating that the reward function can be redesigned without degrading detection reliability. What changes is agent behavior: the revised rewards shift all agents from direct blocking toward puzzle-based challenges, increase honeypot deployment (PPO noaug rises from 53% to 97%), and reverse DG's Tier 5 detection from the worst (72.2%) to the best (96.8%). This decoupling of operational strategy from classification accuracy is a key advantage of the RL formulation over a static classifier.

A binary classifier like XGBoost outputs a fixed score with no mechanism to adjust its decision process after deployment; the RL agent, by contrast, selects from a graded action space (allow, block, easy/medium/hard puzzle, honeypot) and can be steered toward different operational priorities simply by modifying the reward weights, without retraining the underlying behavioral model or recollecting data. Furthermore, because the agent processes sessions sequentially through its LSTM, it naturally supports online deployment where the policy can continue to adapt as new behavioral patterns emerge, providing a self-correcting mechanism against distributional shift that no static classifier can match. The classifier and the RL agents are therefore complementary: the classifier provides a fast, interpretable post-hoc score suitable for audit and logging, while the RL agents enable real-time sequential intervention with adaptive challenge selection and tunable operational objectives.

5.2. Limitations

Several limitations should be acknowledged. First, all data was collected from a single web application (TicketMonarch) with a fixed page layout and user flow, meaning the learned policies may not transfer directly to sites with different interaction patterns or form structures. Second, the human session dataset consists entirely of desktop browser interactions from a small group of university participants, which does not capture the behavioral diversity of mobile users, tablet users, accessibility tool users, or users with varying levels of technical familiarity. The small participant pool also means the classifier's human distribution is narrow, and features like mouse direction change ratio show zero overlap between classes partly because the human sample lacks sufficient variety. Third, training is inherently stochastic: PPO's entropy collapse to 0.10–0.15 makes its final policy sensitive to initialization, and different training runs can converge to meaningfully different local optima, as evidenced by PPO's precision varying across retraining. While we report mean $\pm$ std over five evaluation seeds, training itself was performed with a single seed per configuration, so the reported results may not reflect the full distribution of possible trained policies. Fourth, the evaluation environment simulates puzzle pass/fail rates using fixed probabilities rather than deploying actual CAPTCHA challenges, so the real-world effectiveness of the graduated puzzle intervention strategy remains untested. A bot that can solve hard puzzles at higher rates than assumed would significantly degrade the agents' apparent detection advantage over direct blocking. Fifth, the bot telemetry was generated by five synthetic bot families with hand-designed behavioral profiles rather than captured from real-world bot traffic. While the five-tier difficulty hierarchy covers a range from commodity scripts to LLM-powered agents, real adversarial bots may exhibit behavioral patterns not represented in this taxonomy. Sixth, the 639-session dataset (203 human, 436 bot) is small, and the augmented copies are deterministic perturbations of the original bot sessions rather than independently collected samples. This limits the diversity of the training distribution and may inflate apparent generalization. Finally, the feature set relies exclusively on client-side DOM telemetry (mouse, keyboard, scroll,

 doi:10.20944/preprints202604.1178.v1

click events), which cannot detect bots that operate at the network or browser-engine level without producing distinguishable behavioral signals.

6. Future Work

Future work should explore several directions to strengthen and extend this system. First, developing an RL-based attacker agent that learns evasion strategies against the defender would enable adversarial co-training, where both the attacker and defender continuously improve against each other in a minimax loop. This would produce more robust detection policies than training against a fixed set of bot behaviors. Second, while our current dataset of 436 bot sessions across five tiers and 203 human sessions is sufficient for initial validation, expanding the dataset to include a wider range of real-world user behavior would improve generalization. In particular, users who rely on assistive technologies such as screen readers or switch inputs may produce behavioral telemetry that resembles bot activity, and the current system has not been evaluated against such edge cases. Ensuring low false positive rates for accessibility users is critical for real-world deployment. Third, the 84–90% Tier 5 detection rate suggests that LLM-powered bots remain a partially open challenge. Research into richer feature representations such as sub-window micro-patterns or cross-page behavioral continuity that may capture subtler differences. Finally, integrating the RL agent with browser-level instrumentation beyond DOM telemetry (e.g., GPU rendering timing, network request fingerprinting) could close the remaining detection gap for the most sophisticated adversaries.

Use of Artificial Intelligence: During the preparation of this manuscript, the author(s) used Claude Sonnet 4.6 [57] and Claude Opus 4.6 (Anthropic) [58] for manuscript organization, improving readability, and refining written content. The authors have reviewed and edited the suggestions and take full responsibility for the content of this publication.

Acknowledgments: The code base presented in the study is openly available in Github at adaptive-captcha-defense. The original data set used in this study is openly available in Google drive at: Human Data Set, Bot Data Sets.

Appendix A. RL Window-Level Feature Definitions

Table A1. RL Window-Level Feature Groups (26 dimensions).

Group	Dim	Features
Event composition	4	mouse / click / key / scroll ratios
Mouse dynamics	4	mean speed, speed variance, mean acceleration, path curvature
Timing features	3	mean / variance / min inter-event Δt (log1p normalized)
Click timing	2	mean / variance inter-click interval
Keystroke dynamics	4	mean / variance hold duration, mean / variance inter-key interval
Spatial & contextual	9	scroll magnitude, scroll direction changes, unique cursor positions, spatial x/y range, interactive click ratio, window duration, normalized event count

References

1. BuiltWith Pty Ltd.. reCAPTCHA Usage Statistics. https://trends.builtwith.com/widgets/recaptcha, 2025. Accessed: 2025-02-28.

 doi:10.20944/preprints202604.1178.v1

2. Von Ahn, L.; Blum, M.; Hopper, N.J.; Langford, J. CAPTCHA: Using hard AI problems for security. In Proceedings of the International conference on the theory and applications of cryptographic techniques. Springer, 2003, pp. 294–311. https://doi.org/10.1007/3-540-39200-9_18.
3. Dinh, N.T.; Hoang, V.T. Recent advances of Captcha security analysis: a short literature review. *Procedia Computer Science* **2023**, *218*, 2550–2562. https://doi.org/10.1016/j.procs.2023.01.229.
4. Tang, M.; Gao, H.; Zhang, Y.; Liu, Y.; Zhang, P.; Wang, P. Research on deep learning techniques in breaking text-based captchas and designing image-based captcha. *IEEE Transactions on Information Forensics and Security* **2018**, *13*, 2522–2537. https://doi.org/10.1109/TIFS.2018.2821096.
5. Bock, K.; Hughey, D.; Bhargava, V.; Hoy, N.; Bhatt, D.; Suba, T.; Levin, D. unCaptcha: A Low-Resource Defeat of reCaptcha's Audio Challenge. In Proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, 2017.
6. Xu, Y.; Reynaga, G.; Chiasson, S.; Frahm, J.M.; Monrose, F.; Van Oorschot, P. Security and Usability Challenges of {Moving-Object}{CAPTCHAs}: Decoding Codewords in Motion. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 49–64.
7. Singh, V.P.; Pal, P. Survey of different types of CAPTCHA. *International Journal of computer science and information technologies* **2014**, *5*, 2242–2245.
8. Google. reCAPTCHA v3. https://developers.google.com/recaptcha/docs/v3, 2018. Accessed: 2025-02-28.
9. Intuition Machines, Inc.. hCaptcha Enterprise. https://www.hcaptcha.com/enterprise, 2025. Accessed: 2025-02-28.
10. ALTCHA. ALTCHA: Open-Source CAPTCHA Alternative. https://altcha.org, 2025. Accessed: 2025-02-28.
11. Aggarwal, P.; Du, Y.; Singh, K.; Gonzalez, C. Decoys in Cybersecurity: An Exploratory Study to Test the Effectiveness of 2-sided Deception. In Proceedings of the Proceedings of the IJCAI-21 Workshop on Adaptive Cyber Defense, 2021. https://doi.org/https://doi.org/10.48550/arXiv.2108.11037.
12. Searles, A.; Nakatsuka, Y.; Ozturk, E.; Paverd, A.; Tsudik, G.; Enkoji, A. An Empirical Study & Evaluation of Modern CAPTCHAs. In Proceedings of the 32nd usenix security symposium (usenix security 23). USENIX Association, 2023, pp. 3081–3097.
13. Yan, J.; El Ahmad, A.S. Usability of CAPTCHAs or usability issues in CAPTCHA design. In Proceedings of the Proceedings of the 4th symposium on Usable privacy and security. Association for Computing Machinery, 2008, pp. 44–52. https://doi.org/10.1145/1408664.1408671.
14. Sukhani, K.; Sawant, S.; Maniar, S.; Pawar, R. Automating the bypass of image-based CAPTCHA and assessing security. In Proceedings of the 2021 12th International Conference On Computing Communication And Networking Technologies (ICCCNT). IEEE, 2021, pp. 01–08. https://doi.org/10.1109/ICCCNT51525.2021.9580020.
15. Sivakorn, S.; Polakis, I.; Keromytis, A.D. I am robot:(deep) learning to break semantic image captchas. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2016, pp. 388–403. https://doi.org/10.1109/EuroSP.2016.37.
16. Plesner, A.; Vontobel, T.; Wattenhofer, R. Breaking reCAPTCHAv2. In Proceedings of the 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2024, pp. 1047–1056. https://doi.org/10.1109/COMPSAC61105.2024.00142.
17. Akrout, I.; Feriani, A.; Akrout, M. Hacking Google reCAPTCHA v3 using Reinforcement Learning, 2019, [arXiv:cs.LG/1903.01003].
18. Acien, A.; Morales, A.; Fierrez, J.; Vera-Rodriguez, R. BeCAPTCHA-Mouse: Synthetic mouse trajectories and improved bot detection. *Pattern Recognition* **2022**, *127*, 108643. https://doi.org/https://doi.org/10.1016/j.patcog.2022.108643.
19. Niu, H.; Wei, A.; Song, Y.; Cai, Z. Exploring visual representations of computer mouse movements for bot detection using deep learning approaches. *Expert Systems with Applications* **2023**, *229*, 120225. https://doi.org/https://doi.org/10.1016/j.eswa.2023.120225.
20. Gianvecchio, S.; Wu, Z.; Xie, M.; Wang, H. Battle of Botcraft: fighting bots in online games with human observational proofs. In Proceedings of the Proceedings of the 16th ACM Conference on Computer and Communications Security, New York, NY, USA, 2009; CCS '09, p. 256–268. https://doi.org/10.1145/1653662.1653694.
21. Nguyen, T.T.; Reddi, V.J. Deep reinforcement learning for Cyber Security. *IEEE Transactions on Neural Networks and Learning Systems* **2023**, *34*, 3779–3795. https://doi.org/10.1109/TNNLS.2021.3121870.
22. Google Cloud. Annotate Assessments. https://docs.cloud.google.com/recaptcha/docs/annotate-assessment, 2026. Accessed: 2026-03-31.

 doi:10.20944/preprints202604.1178.v1

23. Google Cloud. Interpret Assessments for Websites. https://docs.cloud.google.com/recaptcha/docs/interpret-assessment-website, 2026. Accessed: 2026-03-31.
24. Google Cloud. Create Assessments for Websites. https://docs.cloud.google.com/recaptcha/docs/create-assessment-website, 2026. Accessed: 2026-03-31.
25. Google Cloud. Install Policy-Based Challenge Keys on Websites. https://docs.cloud.google.com/recaptcha/docs/instrument-web-pages-policy, 2026. Accessed: 2026-03-31.
26. Shackleford, D. SANS Review: reCAPTCHA Enterprise. https://services.google.com/fh/files/misc/sans_review_recaptcha_enterprise.pdf, 2023. SANS Institute Whitepaper, July 2023.
27. Chandra, J.; Manhas, P.; Kaur, R.; Sahay, R. Aura-CAPTCHA: A Reinforcement Learning and GAN-Enhanced Multi-Modal CAPTCHA System. *arXiv preprint arXiv:2508.14976* **2025**. https://doi.org/https://doi.org/10.48550/arXiv.2508.14976.
28. S, S.A.; Venkateshan, P.; Suresh, P.A. Emulating Human-Like Mouse Movement Using Bezier Curves and Behavioural Models for Advanced Web Automation. *International Journal of Innovative Research in Technology* **2025**, *12*, 1364–1370. https://doi.org/10.13140/RG.2.2.20692.10880.
29. Anthropic. Claude 3.5 Sonnet, 2025. Accessed: Feb. 27, 2026.
30. OpenAI. GPT-4o Technical Overview. https://openai.com/research/gpt-4o, 2024. Accessed: 2026-03-31.
31. Google. Gemini: A Family of Highly Capable Multimodal Models. https://deepmind.google/technologies/gemini/, 2024. Accessed: 2026-03-31.
32. browser-use contributors. browser-use. https://github.com/browser-use/browser-use, 2026. Accessed: 2026-04-07.
33. Google. Chrome DevTools Protocol Documentation. https://chromedevtools.github.io/devtools-protocol/, 2026. Accessed: 2026-04-07.
34. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In Proceedings of the International Conference on Learning Representations (ICLR), 2023. https://doi.org/https://doi.org/10.48550/arXiv.2210.03629.
35. Selenium Project. Selenium - Web Browser Automation. https://www.selenium.dev/, 2024. Accessed: 2026-04-07.
36. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples, 2015, [arXiv:stat.ML/1412.6572].
37. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2 ed.; The MIT Press, 2018.
38. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; Wiley Series in Probability and Statistics, John Wiley & Sons: New York, 1994.
39. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.I.; Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.
40. Williams, R.J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* **1992**, *8*, 229–256. https://doi.org/10.1007/BF00992696.
41. Konda, V.R.; Tsitsiklis, J.N. Actor-Critic Algorithms. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS); Solla, S.; Leen, T.; Müller, K., Eds. MIT Press, 1999, Vol. 12.
42. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* **2017**.
43. Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* **1998**, *101*, 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X.
44. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.
45. Harris, F.J. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *Proceedings of the IEEE* **1978**, *66*, 51–83. https://doi.org/10.1109/PROC.1978.10837.
46. Huang, S.; Ontañón, S. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. In Proceedings of the Proceedings of the International FLAIRS Conference, 2022, Vol. 35. https://doi.org/10.32473/flairs.v35i.130584.
47. Hausknecht, M.; Stone, P. Deep Recurrent Q-Learning for Partially Observable MDPs. In Proceedings of the AAAI fall symposia, 2015, Vol. 45, p. 141.
48. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. JMLR.org, 2016, pp. 1928–1937.

 doi:10.20944/preprints202604.1178.v1

49. Pascanu, R.; Mikolov, T.; Bengio, Y. Understanding the exploding gradient problem. *ArXiv* **2012**, *abs/1211.5063*.
50. Osband, I. Delightful Policy Gradient. *arXiv preprint arXiv:2603.14608* **2026**.
51. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the Proceedings of the 35th International Conference on Machine Learning; Dy, J.; Krause, A., Eds. PMLR, 2018, Vol. 80, pp. 1861–1870.
52. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft Actor-Critic Algorithms and Applications. *arXiv preprint arXiv:1812.05905* **2019**.
53. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794. https://doi.org/10.1145/2939672.2939785.
54. Grinsztajn, L.; Oyallon, E.; Varoquaux, G. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data? In Proceedings of the Proceedings of the 36th International Conference on Neural Information Processing Systems. Curran Associates Inc., 2022, pp. 507–520.
55. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture. In Proceedings of the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818–2826. https://doi.org/10.1109/CVPR.2016.308.
56. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-Generation Hyperparameter Optimization Framework. In Proceedings of the Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2019, pp. 2623–2631. https://doi.org/10.1145/3292500.3330701.
57. Anthropic. Claude 4.6 Sonnet, 2026. Accessed: March. 8, 2026.
58. Anthropic. Claude 4.6 Opus, 2026. Accessed: March 8, 2026.