

Shoot the Honey, Cloak the Player: Towards Zero-Runtime-Overhead Proactive Defense and Detection for Visual Game Cheating

Jianing Wang
School of Cyber Science and
Technology
Shandong University
Qingdao, China
jianingwang@mail.sdu.edu.cn

Chuqi Zhang
Independent Researcher
Hangzhou, China
zhangchuqi1999@gmail.com

Yuancheng Jiang
School of Computing
National University of Singapore
Singapore, Singapore
yuancheng@comp.nus.edu.sg

Adil Ahmad
School of Computing and Augmented
Intelligence
Arizona State University
Tempe, United States
adil.ahmad@asu.edu

Shanqing Guo*
School of Cyber Science and
Technology, Shandong Key
Laboratory of Artificial Intelligence
Security, State Key Laboratory of
Cryptography and Digital Economy
Security
Shandong University
Qingdao, China
guoshanqing@sdu.edu.cn

Abstract

Visual aimbots have emerged as a serious cheating threat in first-person shooter (FPS) games, as they evade existing anti-cheat defenses by operating only on rendered frames rather than game memory. However, existing defenses fail to provide an end-to-end solution: post-hoc behavior detectors cannot protect match integrity in real time and are increasingly fragile against human-mimicking aimbots, while proactive runtime defenses often lack accountability, incur substantial overhead, or require intrusive system integration.

We present AIMTRAP, the first end-to-end visual-aimbot defense that combines runtime protection with post-game detection through two adversarial texture mechanisms. Adversarial Camouflage Textures (ACT) hide real players from aimbots, while Adversarial Honeypot Textures (AHT) lure aimbots into locking onto fake targets, yielding strong evidence of cheating. AIMTRAP integrates differentiable rendering with Expectation over Renderings for robust 3D texture synthesis and analyzes honeypot-interaction trajectory to facilitate cheating attribution. In real-game evaluation against a visual aimbot, ACT achieves 85.1% defense success, AHT achieves 96.9%. Compared with prior baselines, AIMTRAP also achieves extremely low false-positive rates with negligible runtime overhead, demonstrating a practical end-to-end defense.

*Corresponding author.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

CCS '26, The Hague, Netherlands

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11

<https://doi.org/10.1145/3830454.3846522>

CCS Concepts

• Security and privacy → Domain-specific security and privacy architectures.

Keywords

Game Security; Proactive Defense; Visual-Cheating Detection

ACM Reference Format:

Jianing Wang, Chuqi Zhang, Yuancheng Jiang, Adil Ahmad, and Shanqing Guo. 2026. Shoot the Honey, Cloak the Player: Towards Zero-Runtime-Overhead Proactive Defense and Detection for Visual Game Cheating. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3830454.3846522>

1 Introduction

Global video game revenue is projected to reach \$564 billion by 2026 [48], with first-person shooters (FPS) remaining one of the most popular and lucrative markets [47]. Unfortunately, FPS games are persistently plagued by cheating, which undermines fair competition, degrades player experience, reduces retention, and causes substantial financial losses for the gaming industry [7, 12, 18, 29]. Among these threats, *aimbots* are particularly damaging in competitive FPS games because they automatically aim and shoot opponents to provide cheaters with an unfair advantage [49, 63, 64].

Aimbots broadly fall into two categories: memory-based and visual-based. *Memory-based aimbots* directly read or manipulate game memory to recover precise enemy locations. They are increasingly constrained by existing anti-cheat systems based on memory monitoring, kernel-level protection, and trusted execution support [5, 6, 9, 13, 34, 54]. In contrast, *visual-based aimbots* [2, 32, 42, 50] operate solely on rendered display frames and

therefore evade these defenses by never tampering with game memory. Powered by advanced object detection models such as YOLO [39–41], they can achieve superhuman aiming precision and speed while remaining highly stealthy.

Existing defenses against visual-based aimbots either perform *post-hoc behavior detection* or implement *proactive runtime defense*. Post-hoc detectors [4, 17, 19, 24, 25, 31, 37, 46, 60–64] analyze game-play statistics to identify suspicious behavior after a match, whereas runtime defenses [30, 49, 59] intervene during gameplay by perturbing rendered visuals to mislead the aimbot. However, they face the following fundamental limitations:

L1. Lack of both accountability and prevention. Neither approach delivers an end-to-end solution. Post-hoc detectors offer *accountability without prevention*: they identify cheating only after unfair advantage has already been gained and therefore cannot protect match integrity in real time. Runtime defenses provide *prevention without accountability*: although they may disrupt cheating during play, they cannot provide forensic evidence for post-game enforcement, lack the ability to confidently attribute misconduct, and thus fail to penalize cheaters effectively.

L2. Fragile against advanced human-mimicking aimbots. Visual aimbots do not exhibit many explicit unnatural behaviors of memory-based cheats, such as pre-firing or perfect tracking enemies through obstacles, which existing detectors rely on, as they share the same field of vision as humans. Moreover, modern cheats employ advanced strategies to mimic human aiming behavior, such as mouse smoothing [2, 42, 50], GAN-based trajectory generation [23], or neuromuscular aim assist [20]. This further narrows the statistical gap between skilled legitimate players and cheaters.

L3. Face practical deployment barriers in real FPS games. Prior defenses face three practical barriers. First, display-perturbation defenses impose substantial per-frame overhead, making them unsuitable for latency-sensitive gameplay [30, 49]. Second, texture-perturbation defenses [59] offer limited geometric coverage: they can only apply to 2D planar surfaces and cannot protect general geometry, such as 3D player and non-planar building models. Third, many defenses require intrusive integration into the game stack, including high-frequency game data collection, peripheral logging, or hooks into shaders and graphics APIs [4, 13, 30, 34, 37, 49]. Together, these barriers limit their portability, maintainability, and adoption in production games.

These limitations raise a key question in the defense landscape: can a visual anti-cheat solution simultaneously prevent cheating online, produce actionable post-game evidence, and remain lightweight enough for deployment in modern FPS games?

We answer this question affirmatively with two key insights. First, **we transform visual-aimbot detection from passive behavior classification into an active visual challenge-response problem**. Rather than inferring cheating from noisy and increasingly human-like aiming behavior, we introduce external visual signals that are salient to visual aimbots but unlikely to affect benign players. Second, unlike display-space perturbations that require expensive per-frame processing or prior texture-space defenses that are limited to 2D planar surfaces, **we shift the cost to an offline, 3D-aware adversarial texture synthesis stage**.

To instantiate these insights, we propose two complementary mechanisms (Fig. 1). *Adversarial Honeypot Textures* (AHT) are static

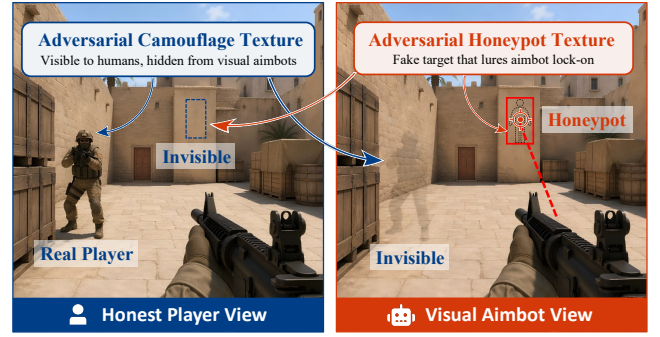


Figure 1: A high-level illustration of AIMTRAP's principles.

decoys placed in game scenes. They appear as high-confidence, player-like objects *only* to visual aimbots, inducing deliberate aiming behaviors (i.e., lock-on), while remaining non-salient to honest players. *Adversarial Camouflage Textures* (ACT) are static noises applied to player models that suppress the aimbot's detection confidence while remaining visually natural to humans. Together, ACT reduces aimbot's ability to detect real opponents, while AHT redirects it toward non-existent targets; *repeated lock-on to such targets then becomes strong evidence of visual cheating*.

Realizing an end-to-end solution based on proposed mechanisms poses two technical challenges. **C1.** It is non-trivial to synthesize *robust* adversarial textures (AHT/ACT) that survive in the 3D rendering pipeline, including UV mapping [57], viewpoint changes, and lighting variations. In addition, secure and scalable texture management is also challenging, as textures must be protected from client-side adversaries and deployed efficiently across millions of online matches. **C2.** Even with honeypot textures, reliably detecting cheating with high accuracy and low false-positive rates remains difficult. The key challenge is to precisely identify sparse honeypot interactions from massive gameplay events while remaining robust to noisy interactions from honest players.

To this end, we present AIMTRAP, a framework that both *proactively defends against* visual aimbots at runtime and *detects* their presence with *strong and explainable evidence*, while preserving practical deployability in real-world FPS games. To enable robust texture synthesis (**C1**), AIMTRAP integrates an end-to-end differentiable rendering pipeline and adopts Expectation over Renderings (EoR) to optimize texture-space perturbations under diverse rendering conditions (§5.2, §5.3). It further incorporates strategies for secure texture management (§5.4) and efficient deployment (§5.5). To enable effective and reliable cheating detection (**C2**), AIMTRAP designs a honeypot-interaction mouse trajectory extraction and abnormal honeypot-interaction identification algorithm (§6).

In real-game settings with real textures, replay logs, and environments, we evaluate AIMTRAP against a state-of-the-art visual aimbot [42]. AIMTRAP achieves strong proactive defense, with ACT reaching an average success rate of 85.1% and AHT 96.9% across diverse rendering conditions (§7.2, §7.5). AHT also transfers to unseen aimbot architectures, reaching 86.9% average decoy success rate across seven detectors when optimized with an ensemble proxy and transfer-enhancing techniques (§7.8). Meanwhile, its cheating detector identifies abnormal honeypot interactions with 98.05%

accuracy and 98.98% recall, and correctly classifies all 40 real-game matches without falsely flagging any of the 20 benign matches (§7.3, §7.4). By aggregating honeypot-interaction evidence at the player level, AIMTRAP further suppresses false positives: requiring more abnormal interactions and consistent evidence across multiple matches reduces the estimated false-positive rate by orders of magnitude (§7.3). The detector also outperforms the state-of-the-art XGUARDIAN [64], which complements AIMTRAP by providing broader coverage of aimbots and wallhacks. Compared with prior perturbation-based defenses [30, 49], AIMTRAP incurs negligible runtime overhead, while preserving perceptual quality and practical deployability (§7.2, §7.7, §7.6). These results demonstrate that AIMTRAP is a practical end-to-end defense against visual aimbots.

Contributions. This paper makes the following contributions:

- (1) We introduce a novel and reliable signal for distinguishing visual aimbots from benign players: adversarial honeypots that induce cheat-specific lock-on behavior.
- (2) We present AIMTRAP, the first end-to-end defense against visual aimbots that unifies proactive runtime protection and post-game detection through two complementary and effective adversarial texture mechanisms, without introducing runtime overhead.
- (3) We evaluate AIMTRAP in real-game environments against an open-source visual aimbot, demonstrating superior effectiveness, explainable detection signals, and practical deployability compared to state-of-the-art baselines.

2 Background and Motivation

2.1 Visual Aimbots: A New Threat in FPS Games

Unlike traditional memory-based aimbots, visual aimbots [2, 32, 42, 50] achieve superhuman precision without tampering with game memory. Instead, they capture real-time display frames and process them through high-speed object detection models, such as YOLO, Faster R-CNN, and NanoDet, to identify on-screen enemies and automatically lock onto targets [39–41]. This vision-based paradigm offers three distinct advantages over memory-based aimbots.

Stealthiness via system-level evasion. Visual aimbots can bypass conventional memory-based anti-cheats [5, 6, 9, 54] as they operate only on legitimate video output. They can also avoid kernel-level monitoring [6, 9] through two-PC setups, where an external machine processes frames through a capture card and returns control signals via simulated peripherals [51]. Recent proof-of-concept systems further show that visual targeting can be coupled with Electrical Muscle Stimulation (EMS) to induce rapid user input, making cheating behavior appear closer to organic human control [20].

Adaptability. Their vision pipeline is inherently game-agnostic. They can seamlessly transfer across different games, platforms, and even cloud gaming environments [28, 33, 45], where traditional memory-based aimbots are infeasible.

Low barrier to entry. Visual aimbots eliminate the need for complex, continuous reverse engineering of obfuscated game binaries. Open-source vision models and LLM-assisted development tools further lower the technical barrier, enabling individuals with minimal technical expertise to develop and deploy effective cheats.

2.2 Existing Defenses Against Visual Aimbots

Existing defenses against visual aimbots can be categorized into the following two paradigms, summarized in Tab. 1.

Post-hoc behavior detection. Such detectors [4, 17, 19, 24, 25, 31, 37, 46, 60–64] analyze gameplay statistics to detect cheating behavior after a match. The statistics can be collected from in-game or system-level data, or extracted from post-game replay logs. They analyze these statistics using machine learning techniques to identify suspicious behavior. For instance, HAWK [63] requires game-specific features (e.g., flash bang affection duration on opponents and teammates) to evaluate player behavior, while XGUARDIAN [64] relies on pitch and yaw angle data to inspect aiming trajectories. Additionally, XGUARDIAN explicitly introduces explainability into its predictions via SHAP [64], whereas other approaches struggle to provide clear evidence for their decisions.

Proactive runtime defenses. Such approaches [30, 49, 59] directly interfere with the aimbot’s vision functionality during gameplay. In particular, display-perturbation methods [30, 49] inject imperceptible adversarial noise into each rendered frame to suppress the functionality of aimbots, but incur significant performance overhead. For instance, Invisibility Cloak [49] applies camouflage perturbations to hide real players. Similarly, Nhu et al. [30] combine camouflage perturbations with decoy perturbations (injecting fake players) to degrade the cheat’s accuracy. Alternatively, texture-perturbation approaches [59] apply noises directly within the texture space to avoid per-frame computational costs.

2.3 Limitations of Existing Defenses

We compare AIMTRAP with state-of-the-art anti-cheat solutions in Tab. 1, identifying critical limitations motivating our approach:

L1. Lack of both accountability and prevention. Neither post-hoc behavior detection nor proactive runtime protection delivers an end-to-end anti-visual-cheating solution (Tab. 1). Post-hoc detectors offer *accountability without prevention*: they identify cheating only after an unfair advantage has been gained, failing to protect match integrity in real time. Runtime defenses provide *prevention without accountability*: although they may disrupt cheating during play, they typically cannot provide reliable forensic evidence for post-game enforcement, lack the ability to confidently attribute misconduct, and thus fail to penalize cheaters effectively.

L2. Existing detectors are fragile against advanced visual aimbots. Visual aimbots can easily evade conventional signature-based detection [5, 6, 9, 54] (§2.1). They also challenge behavioral detectors [4, 17, 19, 24, 25, 60–64] for two primary reasons. First, they do not exhibit explicit unnatural behavior patterns of memory-based cheats, such as pre-firing or perfectly tracking enemies through obstacles. Second, they employ advanced strategies to mimic human aiming trajectories, such as mouse smoothing [2, 42, 50] and GAN-based generation [23], making them difficult to distinguish from highly-skilled legitimate gameplay. Moreover, neuromuscular aim assist [20] (§2.1) produces aiming patterns physiologically consistent with natural muscle responses, further blurring the line between human and machine behavior.

L3. Face practical deployment barriers in real FPS games. First, display-perturbation defenses [30, 49] struggle to meet the strict performance standards of real-time FPS games (144–360 fps) [35,

Table 1: Comparison between AIMTRAP (this work) and state-of-the-art visual anti-cheat solutions.

System	Category	Post-hoc Behavior Detection			Proactive Runtime Protection				Deployment
		Input	Evidence	Anti-Mimic	Space	Camouflage	Decoy	Overhead	
BotScreen [13] (SEC'23)	Detection	✗ Gamedata	✗	✗	N/A	N/A	N/A	N/A	✗ (Legacy Intel SGX)
Nhu et al. [30] (IPCCC'23)	Protection	N/A	N/A	N/A	✗ Display output	✓	✓	✗ Severe	✗ (Display modification)
Invisibility Cloak [49] (SEC'24)	Protection	N/A	N/A	N/A	✗ Display output	✓	✗	✗ High	✗ (Display modification)
AdvMap [59] (TG'25)	Protection	N/A	N/A	N/A	✗ 2D texture	✗	✓	✓ Negligible	✓
HAWK [63] (TIFS'25)	Detection	✓ Replay	✗	✗	N/A	N/A	N/A	N/A	✓
XGUARDIAN [64] (SEC'26)	Detection	✓ Replay	✓	✗	N/A	N/A	N/A	N/A	✓
AIMTRAP (this work)	Both	✓ Replay	✓	✓	✓ 3D texture	✓	✓	✓ Negligible	✓

Note: *Evidence*: whether the method provides human-interpretable evidence for forensic analysis or manual inspection. *Anti-Mimic*: denotes resilience against advanced aimbots designed to mimic human aiming behavior [20, 23]. *Deployment*: whether the method can be deployed transparently in practice without requiring intrusive changes.

Table 2: Runtime overhead comparison of AIMTRAP and prior proactive runtime protection solutions.

System / Baseline	Defense Space	GPU	Resolution	FPS
Gameplay baseline (Casual – Esports)		–	1920 × 1080	144–360
Base hardware baseline (Steam survey [16])		RTX 3060	1920 × 1080	150–350
High-end GPU hardware baseline		RTX 4090	1920 × 1080	400–600
Nhu et al. [30]	Display output	N/R	1920 × 1080	~ 0.12
Invisibility Cloak [49]	Display output	RTX 4090	320 × 320	21–77
AdvMap [59]	2D texture	RTX 4090	1920 × 1080	400–600
AIMTRAP (this work)	3D texture	RTX 4090	1920 × 1080	400–600

Note: N/R denotes hardware not reported. FPS values for [30, 49] come from their papers; [59] and AIMTRAP use online benchmarks, while CPU/memory may differ.

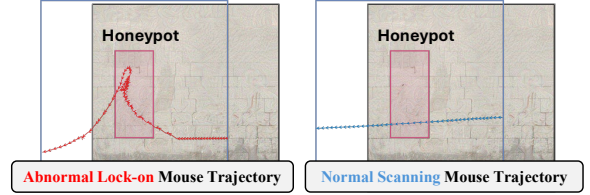
53]. As shown in Tab. 2, Nhu et al. [30] require 8.5 seconds per frame, while Invisible Cloak [49] achieves only 21–77 fps on a high-end RTX 4090 (extrapolating to 9–34 fps on commodity RTX 3060s [14, 16]). Second, texture-perturbation defense [59] reduces runtime overhead but exhibits limited generalizability. Specifically, AdvMap [59] is restricted to 2D planar surfaces and thus cannot be extended to general 3D geometry, including player models and non-planar building models. Last, integrating prior approaches requires disruptive engine or system changes. Behavior-based detectors [4, 13, 37] depend on high-frequency game data collection, peripheral logging, or specialized hardware support (e.g., the legacy Intel SGX). Furthermore, display-perturbation approaches [30, 49] require deep integration with the rendering pipeline, such as hooking shaders or graphics APIs. These intrusions complicate compatibility and maintainability of off-the-shelf games.

2.4 Problem Statement and Threat Model

This paper addresses the emerging threat of *visual aimbots* in FPS games with a defense that (i) proactively disrupts them during gameplay and (ii) provides reliable evidence to identify cheaters.

Assumptions. We assume that standard anti-cheat mechanisms (e.g., EAC and Vanguard) already protect the game client against memory tampering and code injection [5, 6, 54]. We trust the integrity of the game engine, operating system, and network stack. Thus, we specifically focus on external, vision-based cheats.

Adversary Capabilities. Consistent with today’s common practice, cheaters have the following capabilities: (1) *Frame Capture*: Accessing rendered game frames directly or via external hardware (e.g., HDMI capture cards) at runtime to bypass signature-based

**Figure 2: Honeypot-interaction trajectory comparison.**

detectors. (2) *Input Injection*: Generating synthetic mouse and keyboard inputs using OS-level simulation or external USB emulators. (3) *Computer Vision*: Applying models (e.g., YOLO [40]) to captured frames for superhuman aiming precision. (4) *Behavioral Mimicry*: Post-processing aiming trajectories and adding natural delays to evade behavioral detection [20, 23].

3 Approach

3.1 Enabling Observations

Our approach is motivated by three FPS gameplay observations.

Observation 1. *Visual aimbots react to detector confidence rather than human semantic understanding.* Visual aimbots operate on rendered frames and rely on object detectors to identify targets. Therefore, they can react to detector-salient visual cues even when those cues do not correspond to meaningful enemies from a human player’s perspective. This creates an opportunity to manipulate aimbots through visual cues that are recognizable to detectors but imperceptible to legitimate players.

Observation 2. *If a detector-salient but human-imperceptible visual cue induces repeated lock-on, that behavior becomes a cheat-specific signal.* We define such cues as *visual honeypots*. Fig. 2 illustrates the contrast between normal players and visual aimbots. Since normal players cannot perceive honeypots as meaningful targets, they do not intentionally aim at those locations. Their mouse trajectories either pass smoothly across honeypot regions, like scanning, or remain relatively stable at specific points due to normal pre-aiming. In contrast, aimbots are attracted to honeypots and repeatedly lock onto them, exposing automated target selection.

Observation 3. *Relative motion makes static cues dynamic aimbot targets.* FPS gameplay is inherently dynamic: as players move or rotate their view, the perspective projection of the 3D scene continuously changes. By the principle of relative motion, static honeypots continuously shift in screen space and can repeatedly

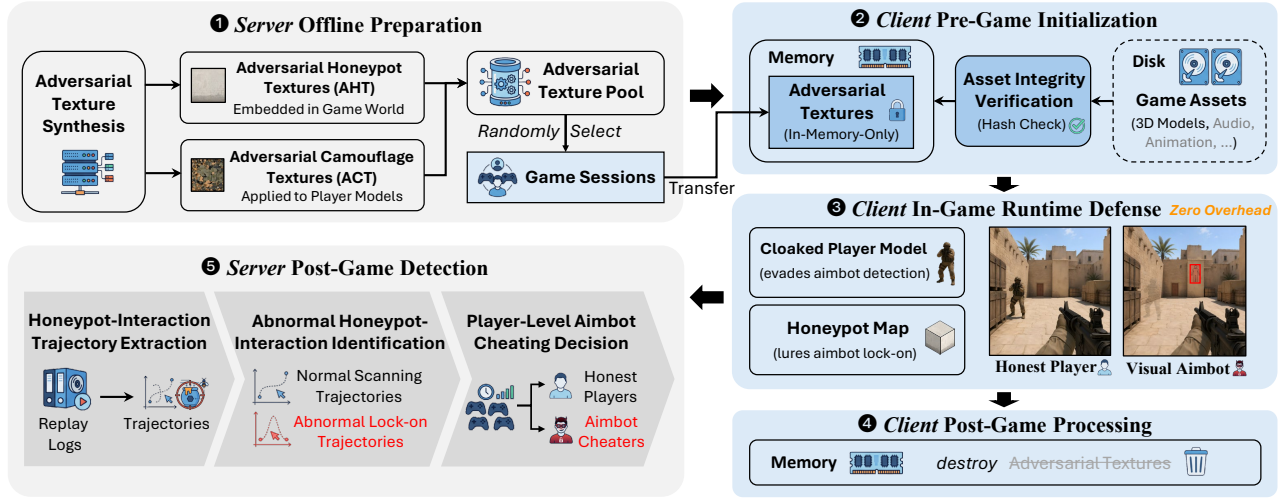


Figure 3: Overview of AIMTRAP.

enter aimbot’s detection region around the crosshair. This makes static honeypots behave like dynamic fake targets from aimbot’s perspective, without runtime animation or rendering modification.

3.2 Adversarial Texture Mechanisms

Building on these observations, we present two key insights.

Insight 1. We transform visual-aimbot detection from passive behavior classification into an active visual challenge-response problem. Rather than inferring cheating only from noisy and increasingly human-like aiming behavior, we introduce detector-salient visual signals that are unlikely to affect legitimate players.

Insight 2. Static game assets provide a practical carrier for 3D-aware adversarial visual cues. By embedding adversarial cues directly into textures, we can synthesize them in an offline, 3D-aware pipeline and preserve their detector salience across diverse rendering conditions, avoiding expensive per-frame runtime perturbation.

We therefore propose two complementary mechanisms (Fig. 1).

Adversarial Honeypot Textures (AHT) contain static decoys, i.e., honeypots, which are placed in game scenes. They are optimized to lure aimbots into targeting non-existent enemies while remaining visually non-salient to legitimate players. AHT degrades cheat effectiveness by redirecting aimbot precision toward irrelevant targets and provides explainable post-game evidence through abnormal lock-on behavior.

Adversarial Camouflage Textures (ACT) apply static perturbations to player models. They suppress the aimbot’s detection confidence while remaining visually natural to humans, inspired by prior display-perturbation defenses [30, 49]. ACT reduces the aimbot’s ability to engage real players and complements AHT by redirecting target selection toward visual honeypots.

Together, AHT and ACT degrade visual aimbots at runtime while enabling post-game cheat detection, addressing L1. Unlike detectors that infer cheating from noisy human-mimic aiming behavior, our visual honeypots introduce an explicit in-game signal: repeated, intentional lock-on to human-imperceptible decoys, which is cheat-specific and unlikely to arise from benign play, addressing

L2. Both AHT and ACT are synthesized offline and rendered as normal assets, requiring neither per-frame computation nor invasive display-pipeline integration. Additionally, detection is performed server-side using standard in-game replay logs already available in modern FPS games, addressing L3.

3.3 Challenges

Realizing an end-to-end solution presents two technical challenges.

C1: Robust adversarial texture synthesis and management.

Synthesizing static 2D textures that remain adversarial after 3D game rendering is challenging. In games, textures are mapped onto 3D meshes and processed by complex rendering pipelines, while viewpoints and distances vary unpredictably across players. Thus, a single perturbation must remain effective under diverse rendering conditions encountered in practice (§5.2, §5.3). Moreover, since these textures are deployed on the client, adversaries can mount adaptive attacks, for example by probing to infer honeypot locations or extracting the textures to retrain their aimbots for increased robustness. This requires a secure and efficient texture management mechanism that limits attacker predictability (§5.4, §5.5).

C2: Effective honeypot-based detection. Designing a detector that uses honeypot interactions as reliable evidence without causing false positives is also challenging. The system must locate, aggregate, and interpret sparse interactions within massive replay datasets—often the proverbial needle in a haystack. In addition, benign players may occasionally trigger such interactions by chance. Therefore, the detector must tolerate rare benign events while still confidently identifying cheat-driven behavior (§6).

4 System Overview

AIMTRAP is an anti-visual-cheat framework that proactively interferes with aimbots at runtime with zero overhead and detects aimbot cheating behavior. To address outlined challenges (§3.3), AIMTRAP consists of five key phases as illustrated in Fig. 3:

1 Server Offline Preparation. The server synthesizes a diverse pool of adversarial textures offline (§5), including (a) Adversarial

Camouflage Textures (§5.2) for player models to evade aimbot targeting, and (b) Adversarial Honey-pot Textures (§5.3) embedded in the game world to lure aimbots and expose their cheating behavior. We randomize the texture synthesis and per-session selection to limit attacker predictability (§5.4).

② **Client Pre-Game Initialization** (§5.4). In this phase, the client loads and prepares game assets. In detail, (a) adversarial textures are securely transmitted from the server to the client's *memory* before the match starts, and (b) other assets (e.g., 3D models) are loaded from disk and verified against expected hash values. This integrity verification prevents attackers from tampering with game assets to weaken our adversarial textures.

③ **Client In-Game Runtime Defense**. At runtime, our adversarial textures proactively disrupt aimbot's functionality: ACT-camouflaged player models evade aimbot detection, while AHT-based honeypots embedded in the map lure aimbot to lock onto them. As these textures are 3D-robust and static game assets, AIM-TRAP does not need to modify the rendering pipeline at runtime, unlike baselines that inject or alter rendering [30, 49], and therefore introduces no performance overhead during gameplay (§7.6).

④ **Client Post-Game Processing** (§5.4). The client discards all downloaded adversarial textures immediately after the match. This *in-memory-only* lifecycle places them under existing memory-based anti-cheat protections (see Assumptions in §2.4) and prevents any on-disk persistence that could expose them to aimbot developers.

⑤ **Server Post-Game Detection** (§6). AIMTRAP performs an effective and efficient post-game detection on standard replay logs. Note that AIMTRAP requires no instrumentation beyond standard replay logging, incurring no additional recording or storage cost. The detection pipeline has three stages: (i) extracting honeypot-interaction aiming trajectories from tick-level logs via geometric gating and temporal clustering (§6.1); (ii) classifying each interaction based on its aiming pattern to identify abnormal aimbot-like trajectory (§6.2); and (iii) aggregating interaction-level inferences into a player-level decision for the match (§6.3).

5 Adversarial Texture Synthesis

This section presents our adversarial texture synthesis pipeline (Fig. 4), covering preliminaries (§5.1), synthesis process (§5.2, §5.3), and their secure and efficient management strategies (§5.4, §5.5).

5.1 Preliminaries

Expectation over Transformations (EoT). EoT [8, 10, 43, 58] is a training-time strategy for crafting robust adversarial examples by optimizing not for a single input, but for the expected model loss under a distribution of transformations (e.g., scaling, rotation, viewpoint, lighting, noise, compression).

Expectation over Renderings (EoR). When adversarial examples must remain effective after a 3D rendering pipeline, nuisance variables arise from the rendering process that maps 3D assets (e.g., geometry, materials, and textures) and scene conditions (e.g., camera pose, distance, FOV, lighting, background, and motion blur) to the final image. We therefore adapt EoT to *Expectation over Renderings* (EoR), which optimizes asset-space parameters—here, the texture—so that the rendered outputs remain adversarial over a distribution of rendering states. Let M denote the mesh, τ the texture parameters to optimize, $\theta \sim D$ the rendering parameters, R a

differentiable renderer, f the aimbot vision model, and y the attack target (evasion or decoy). The EoR objective is

$$\min_{\tau \in \mathcal{T}} \mathbb{E}_{\theta \sim D} [\mathcal{L}(f(R(M, \tau; \theta)), y)]. \quad (1)$$

Fully Differentiable Rendering Pipeline. To optimize under the EoR objective above, we integrate an end-to-end differentiable rendering pipeline [21, 36]. The renderer $R(M, \tau; \theta)$ maps the textured asset to an image I given a render condition θ . Because rasterization and shading are differentiable in our setup, gradients of the detection loss with respect to the texture follow the chain rule:

$$\tau \xrightarrow{\text{UV Mapping}} M \xrightarrow{R(\cdot; \theta)} I \xrightarrow{f(\cdot)} \text{detections} \xrightarrow{\mathcal{L}} \nabla_{\tau} \mathcal{L}, \quad (2)$$

i.e., $\nabla_{\tau} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial I} \frac{\partial R}{\partial \tau}$.

5.2 Adversarial Camouflage Textures

Adversarial Camouflage Textures (ACT) degrade aimbots' performance by reducing their confidence of game player models, thereby hindering localization and recognition. To ensure robustness across rendering conditions, we optimize ACT under EoR framework.

Rendering Condition Distribution. Let the game-engine rendering condition be $\theta = (d, \phi, \psi, \lambda, b)$, where d is camera distance ($d \in [d_{\min}, d_{\max}]$), (ϕ, ψ) are yaw/pitch angles ($\phi \in [\phi_{\min}, \phi_{\max}]$, $\psi \in [\psi_{\min}, \psi_{\max}]$), λ parameterizes lighting (e.g., ambient/point), and b indexes background assets ($b \in \mathcal{B}$). We define a distribution $\mathcal{D}$ over θ and, at each iteration t , draw a *minibatch* of m rendering states

$$\Theta_t = \{\theta_1, \dots, \theta_m\}, \quad \theta_j \stackrel{\text{i.i.d.}}{\sim} \mathcal{D}. \quad (3)$$

ACT's EoR Objective. We optimize ACT under Eq. (1) using a stochastic estimator over the current minibatch Θ_t :

$$\widehat{\mathcal{L}}_{\text{ACT}}(\tau; \Theta_t) = \frac{1}{m} \sum_{j=1}^m \ell_{\text{det}}(f(R(M, \tau; \theta_j))), \quad \theta_j \stackrel{\text{i.i.d.}}{\sim} \mathcal{D}. \quad (4)$$

This unbiased estimator is differentiated end-to-end through the rendering pipeline (Eq. (2)) to obtain $\nabla_{\tau} \widehat{\mathcal{L}}_{\text{ACT}}$.

Constrained Update. Following [26], we update τ with projected gradient steps in an ℓ_{∞} ball of radius ϵ around the initial texture τ_0 , with per-channel clamping to $[0, 1]$:

$$\tilde{\tau}^{(t+1)} = \tau^{(t)} - \alpha \text{sign}(\nabla_{\tau} \widehat{\mathcal{L}}_{\text{ACT}}(\tau^{(t)}; \Theta_t)), \quad (5)$$

$$\tau^{(t+1)} = \text{clip}_{[0,1]}(\Pi_{\mathcal{B}_{\infty}(\tau_0, \epsilon)}(\tilde{\tau}^{(t+1)})), \quad (6)$$

where $\Pi_{\mathcal{B}_{\infty}(\tau_0, \epsilon)}(z) = \tau_0 + \text{clip}_{[-\epsilon, \epsilon]}(z - \tau_0)$ is the projection onto the ℓ_{∞} ball about τ_0 . At each iteration, we resample Θ_t (viewpoints, distances, lighting, backgrounds), accumulate minibatch gradients via Eq. (4), and apply Eq. (5).

5.3 Adversarial Honey-pot Textures

Adversarial Honey-pot Textures (AHT) are embedded in 3D world to expose aimbot behavior: they are visually natural to human players, yet appear to vision-based aimbots as highly salient, target-like objects. We again use the EoR strategy to ensure robustness under rendering variability.

Rendering Condition Distribution. We follow nearly the same rendering distribution $\mathcal{D}$ as in Eq. (3), excluding the background b term since the honeypot is part of the scene.

Targeted Detection with Uniqueness. We want a *single* decoy in aimbot's detection area, suppressing competitors; otherwise multiple decoys can affect aimbot's decision-making process, leading to unstable aiming behavior. Let p_1 be the top detection score and

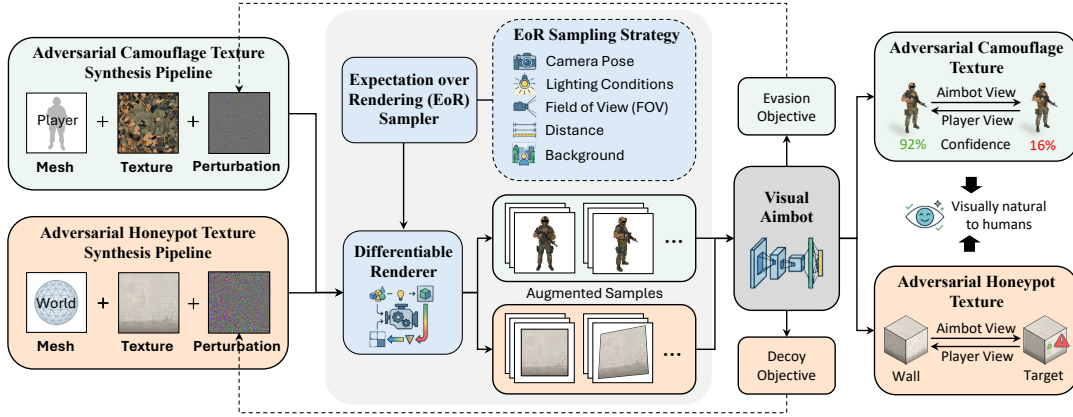


Figure 4: Adversarial texture synthesis.

p_2 the highest-confidence *non-overlapping* competitor. We enforce this with a margin and confidence calibration term:

$$\mathcal{L}_{\text{rank}} = a \text{ReLU}(m - (p_1 - p_2)) + b(1 - p_1)^2, \quad (7)$$

where m is the desired margin and a, b weight the terms.

Geometric Consistency. To reduce unstable locks under view-point changes, we regularize the top box toward desired normalized dimensions. Let (w, h) be the width/height of the top box, and (W_T, H_T) be the reference extents for normalization under θ_j . Define $w_{\text{frac}} = w/W_T$, $h_{\text{frac}} = h/H_T$ and targets $w_{\text{frac}}^*, h_{\text{frac}}^*$. The geometric loss is

$$\mathcal{L}_{\text{geo}} = \mu |w_{\text{frac}} - w_{\text{frac}}^*| + \nu |h_{\text{frac}} - h_{\text{frac}}^*|, \quad (8)$$

with $\mu, \nu > 0$ weighting width/height deviations.

AHT's EoR Objective. We optimize AHT using a stochastic estimator over Θ_t :

$$\widehat{\mathcal{L}}_{\text{AHT}}(\tau; \Theta_t) = \frac{1}{m} \sum_{j=1}^m \left[\lambda_r \mathcal{L}_{\text{rank}}^{(j)} + \lambda_g \mathcal{L}_{\text{geo}}^{(j)} \right], \quad \theta_j \stackrel{\text{i.i.d.}}{\sim} \mathcal{D}, \quad (9)$$

which is differentiated end-to-end through the pipeline (Eq. (2)) to obtain $\nabla_{\tau} \widehat{\mathcal{L}}_{\text{AHT}}$. Finally, we update the texture τ following the same method described in Eq. (5).

Cross-Model Transferability Optimization. We enhance the cross-model transferability of AHTs by combining proxy-model ensembling (ENS), momentum iterative optimization (MI), and translation-invariant optimization (TI). At each EoR-PGD step, ENS averages the attack loss across an ensemble of proxy models. TI then smooths the resulting texture gradient using a 7×7 Gaussian kernel, reducing sensitivity to spatial shifts. Finally, MI accumulates the ℓ_1 -normalized smoothed gradient with momentum factor $\mu = 1.0$. The sign of the accumulated gradient is used in the projected update (Eq. (5)), encouraging the optimization to identify perturbation directions shared across model decision boundaries and thereby improving cross-model transferability (§7.8).

5.4 Secure Adversarial Texture Management

This section details the security guarantees of adversarial textures, enforced by the following three mechanisms.

Randomized Synthesis and Per-Session Selection. Following well-established practices to counter predictability, such as address

space layout randomization (ASLR) [44], we randomize both texture synthesis and per-game-session selection. In detail, we synthesize a *diverse* pool of adversarial textures offline by varying the synthesis initialization (e.g., random seeds). Then we *randomly* sample a subset from the pool for each game session (①, Fig. 3). Such randomization obscures both which textures are adversarial and the specific perturbation patterns in each texture, limiting an aimbot developer's ability to anticipate or adapt across sessions.

In-Memory-Only Deployment. By default, CS2 stores normal textures on disk, and our design preserves this behavior for normal assets. For *adversarial* textures, however, we depart from disk storage: they are transmitted over the network from the server, kept solely in the client's memory, and then cleared immediately in memory when the match ends (②, ④, Fig. 3). This in-memory-only deployment keeps the adversarial textures within the protection scope of existing memory-based anti-cheat mechanisms (see Assumptions in §2.4). By avoiding any persistent disk storage for these textures, it prevents an aimbot developer's ability from extracting, reverse-engineering, or adapting to our textures.

Asset Integrity Verification. Because our textures are applied to existing client-side 3D models, an adversary might attempt to degrade effectiveness by altering models or UV mappings [57]. To prevent such tampering, we verify the integrity of all game assets loaded from the client's disk (③, Fig. 3). This integrity verification follows standard practice already enforced by vanilla clients (e.g., Valve's VPK and Unreal Engine's PAK file integrity check). Thus, our approach introduces negligible integration overhead.

5.5 Efficient Adversarial Texture Deployment

This section explains how we achieve efficient texture deployment through the following three design choices.

Offline Synthesis. We pre-synthesize a diverse pool of adversarial textures on the server, entirely *offline*, so no on-the-fly synthesis is incurred during live gameplay. The average synthesis time for AHT is 12 s and ACT 75 s (§7.6).

Cross-Session Reuse. The server reuses each texture across multiple sessions to amortize the synthesis cost. As reported by [38], CS2 has averaged 902k online players over the past two years, peaking at 1.8M. Assuming 10 players per match, this corresponds to

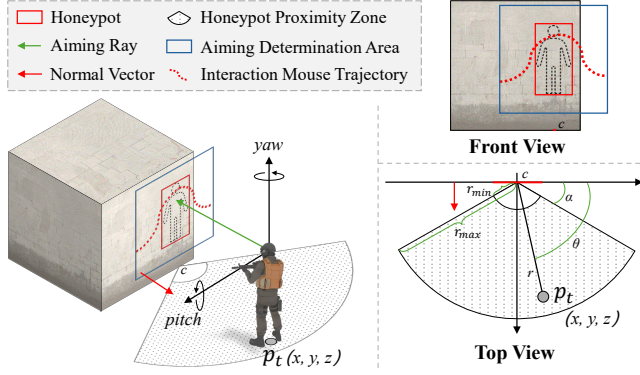


Figure 5: Adversarial honeypot for aimbot detection

roughly 90k concurrent matches on average. At this scale, reusing each texture for hundreds to thousands of sessions still yields ample variety while keeping costs manageable. Suppose each texture is reused for 1,000 sessions, the amortized per-session cost is ≈ 0.012 s (AHT) and ≈ 0.075 s (ACT), both < 0.1 s. Such overheads are readily affordable for production servers.

Lightweight Network Transmission. Adversarial textures are lightweight image files, far smaller than heavyweight assets, such as 3D models. In our implementation, AHTs average 0.83 MB and ACTs 7.48 MB (§7.6). Thus, per-session download introduces negligible bandwidth and latency overhead to the whole system.

6 Aimbot Cheating Detection

This section presents our post-game aimbot detection, comprising three phases: (i) honeypot interaction trajectory extraction (§6.1), (ii) abnormal trajectory identification (§6.2), and (iii) player-level cheating detection (§6.3).

6.1 Honeypot-Interaction Mouse-Aiming Trajectory Extraction

Extraction Overview. We convert tick-level replay logs (details: §C) into precise honeypot-interaction mouse aiming trajectories in two stages. First, *geometric gating* retains only spatially-valid ticks: the player must be within the *Honeypot Proximity Zone* and their view (aiming) ray must intersect the *Aiming Determination Area* on the honeypot plane, as illustrated in Fig. 5. Second, *temporal clustering* groups valid ticks to segment contiguous interactions. For each cluster, the intersections between the aiming ray and the honeypot plane define a 2D honeypot-interaction mouse trajectory. **Honeypot Proximity Zone (HPZ).** We define HPZ as the neighborhood around each honeypot in which a player is considered spatially eligible for interaction. Let c be the honeypot center projected to the floor plane, p_t the player's floor-projected position at tick t . Define $r = \|p_t - c\|$ and θ the incidence angle relative to the honeypot surface. We instantiate the HPZ as the circular sector, as illustrated in the top view of Fig. 5:

$$\text{HPZ} = \{(r, \theta) \mid r_{\min} \leq r \leq r_{\max}, \alpha \leq \theta \leq \pi - \alpha\},$$

excluding viewpoints too close, far, or excessively grazing. Ticks with $(r, \theta) \notin \text{HPZ}$ are discarded to improve accuracy and efficiency.

Aiming Determination Area (ADA). Conditioned on being in HPZ, we further require that the player's view ray intersects a target region on the honeypot plane (see the front view in Fig. 5). The ADA is an axis-aligned 2D box on the honeypot plane, centered at the honeypot. A tick is *aim-valid* if the ray cast from the player's position along their (Yaw, Pitch) direction intersects this box. The resulting aiming trajectory is the ordered sequence of 2D intersection points within ADA. We size ADA slightly larger than the physical honeypot to capture enough aiming patterns.

Temporal Clustering. After HPZ/ADA gating, we cluster the remaining tick indices using 1-D DBSCAN to form contiguous interaction segments. We set the neighborhood radius to $\epsilon = 64$ ticks (≈ 1.0 s) and the minimum samples to $m = 32$ ticks (≈ 0.5 s), so that clusters correspond to sustained interactions rather than spurious single-frame hits. Each cluster yields a 2D mouse trajectory in ADA plane, representing a complete honeypot interaction.

6.2 Abnormal Interaction Identification

Given the extracted honeypot-interaction aiming trajectories, we formulate abnormal trajectory detection as supervised sequence classification. Each trajectory is labeled and encoded with temporal deltas, 2D intersection points, and honeypot context, and a bidirectional LSTM is trained to distinguish abnormal from normal trajectories. Below, we describe our approach in detail.

Dataset. We build a dataset of 3,074 labeled trajectories for training and evaluating our classification model (§7.3).

Sequence representation and normalization. Given a clustered mouse aiming trajectory of length T , we encode each tick into a four-dimensional token

$$s_t = [\Delta t, x_t, y_t, 1\{(x_t, y_t) \in \text{HP}\}] \in \mathbb{R}^4 \quad (t = 1 \dots T),$$

where $\Delta t = (\text{tick}_t - \text{tick}_{t-1})/64$ normalized tick rate, (x_t, y_t) are 2D coordinates in ADA, $[0, 1]$ indicates whether the hit is inside the honeypot HP. In addition, we append the honeypot's normalized box coordinates $[x_{\text{HP}}, y_{\text{HP}}, w_{\text{HP}}, h_{\text{HP}}]$ as static features. Sequences are truncated or padded to $T_{\max} = 320$ with masks.

Model architecture. We encode each sequence with a bidirectional LSTM. Let the final forward and backward states be $h^{\rightarrow}, h^{\leftarrow} \in \mathbb{R}^d$ and define the sequence embedding $h = [h^{\rightarrow} || h^{\leftarrow}] \in \mathbb{R}^{2d}$. We concatenate h with b_{HP} and feed it to a two-layer MLP $g(\cdot)$ with Batch-Norm and Dropout: $z = g([h || b_{\text{HP}}]) \in \mathbb{R}$, where z is the logit for the positive class. We train at the sequence level using class-weighted binary cross-entropy with logits and ℓ_2 regularization. For a batch $\{(S_i, b_{\text{HP},i}, y_i)\}_{i=1}^N$ with $y_i \in \{0, 1\}$ and logits $z_i = g([h_i || b_{\text{HP},i}])$,

$$\text{BCELogit}(z_i, y_i) = \log(1 + e^{z_i}) - y_i z_i,$$

$$\mathcal{L}_{\text{data}} = \frac{1}{N} \sum_{i=1}^N w_{y_i} \text{BCELogit}(z_i, y_i),$$

where $w_1 = \rho$, $w_0 = 1$, and $\rho = N_{\text{neg}}/N_{\text{pos}}$ when positives are rarer. The final objective is

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda \|\theta\|_2^2,$$

with θ denoting all trainable parameters and λ set via weight decay in the optimizer.

6.3 Player-Level Aimbot Cheating Detection

This section formalizes our player-level decision rules based on trajectory-level identification.

Cheating Decision Making. Trajectory-level identification (§6.2) does not directly translate into player-level cheating detection. Instead, we aggregate predictions across multiple honeypot-interaction trajectories into a *player-level decision*: a player is flagged if they exhibit at least T abnormal honeypot interactions in a match. **Player-Level Cheating Detection.** We model each honeypot interaction within a match as independent, as honeypots are placed at distinct locations and interactions occur at different times. Let TPR_T and FPR_T denote per-trajectory true-positive and false-positive rate respectively. For a match with n interaction opportunities, the number of abnormal interactions follows a Binomial distribution, and a player is flagged when at least T of the n interactions are classified abnormal. The resulting player-level true-positive rate TPR_P and false-positive rate FPR_P are the corresponding binomial tails:

$$\text{TPR}_P(n, T) = \sum_{k=T}^n \binom{n}{k} \text{TPR}_T^k (1 - \text{TPR}_T)^{n-k},$$

$$\text{FPR}_P(n, T) = \sum_{k=T}^n \binom{n}{k} \text{FPR}_T^k (1 - \text{FPR}_T)^{n-k}.$$

Tunable Thresholds. Developers can tune the aggregation threshold T to trade recall for precision: increasing T lowers FPR_P at the cost of TPR_P , and decreasing T does the opposite (Tab. 6 in §7.3; §C details the ablation study of n). Because mis-flagging legitimate players undermines the anti-cheat system’s *trustworthiness*, real-world deployments typically prioritize a low false-positive rate over occasional misses. In our real-game evaluation (§7.4), we empirically find thresholds $T \in [3, 5]$ provide a practical operating range, accurately identifying player-level cheating while keeping no false alarm.

7 Evaluation

We answer the following research questions (RQs):

- **RQ1.** How effective is AIMTRAP in proactively defending against vision cheating during gameplay (§7.2)?
- **RQ2.** How does our detection model perform in distinguishing cheating honeypot-interaction trajectories from normal trajectories? How does it perform at player-level? Does the visualized mouse trajectory reveal a strong signal? (§7.3)
- **RQ3.** How effective is our detection model in real-game scenarios, and how does its performance compare against state-of-the-art baselines? (§7.4)
- **RQ4.** How robust is our approach under diverse game rendering conditions (§7.5)?
- **RQ5.** What is AIMTRAP’s performance overhead (§7.6)?
- **RQ6.** Do adversarial textures degrade real-world player experience (e.g., distraction or visual strain) (§7.7)?
- **RQ7.** How well does our approach transfer across different aimbot models (§7.8)?

7.1 Experimental Setup

Implementation. Implementation details are in the §C.

Texture Assets. We evaluate scalability by synthesizing adversarial textures from real-game assets and 3D generation pipelines [1, 22, 27]. AHT uses 24 official CS2 textures across 7 material types. ACT uses Tripo [1] to generate 4 male and female player models (§C).

Table 3: Effectiveness of Adversarial Camouflage Textures (ACT) for evading aimbots

Original Texture Example	Male Character		Female Character	
	Texture	Aimbot	Texture	Aimbot
		 64%		 67%
Adversarial Camouflage Texture Example	Texture	Aimbot	Texture	Aimbot
		 30%		 29%

	Male	Female	Total
# of Textures	2	2	4
# of ACT	20	20	40
# of EoR Sampling	25,600	25,600	51,200
Original Confidence Score	67.7%	56.8%	62.3%
ACT Confidence Drop	↓ 49.9%	↓ 42.5%	↓ 46.3%
ACT Confidence Score	17.8%	14.3%	16.0%
SSIM	81.5%	80.5%	81.0%
Evasion Success Rate	84.1%	86.0%	85.1%

Map Construction and Honeypot Deployment. We use Valve’s Hammer editor [15] to build a custom CS2 map, whose structure is inspired by the classic CS1.6 map, iceworld (details: §C). This map deploys 16 AHTs at diverse locations.

Aimbot and Proxy Models. We evaluate a widely used open-source YOLOv5-based aimbot [42] (2.1k GitHub stars) at its default confidence threshold of 0.4. Its default mouse smoothing applies 40% of the target offset per step, producing human-like trajectories instead of instant snaps (configuration: §C). Following prior work [49], we optimize against YOLOv5n as the proxy and primarily evaluate on YOLOv5s as the attacker. We evaluate AHT transfer boosters across YOLOv5/8, Faster R-CNN, and NanoDet (§7.8).

7.1.1 Evaluation Metrics. We assess AIMTRAP using the following four metrics.

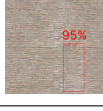
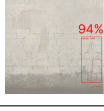
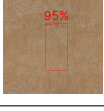
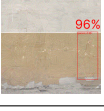
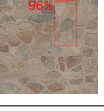
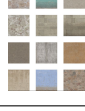
Decoy Success Rate (DSR). Measures how effectively AHTs *attract* the aimbot. We consider a successful deception when aimbot’s top-1 detection is a honeypot with confidence ≥ 0.4 (default threshold §7.1). We evaluate DSR across various conditions, including different camera distances and angles (§7.5).

Evasion Success Rate (ESR). Measures how effectively ACTs *hide* players from the aimbot. We consider a successful evasion if no player-aligned detection exceeds 0.4. ESR is reported under the same rendering variations.

Uniqueness Rate (UR). Measures whether AHT yields a *single*, unambiguous attractor (§5.3). We consider it unique if exactly *one* honeypot exceeds detection threshold. High uniqueness reduces multi-target ambiguity for both attraction and detection.

Structural Similarity Index (SSIM) [55]. Quantifies perceptual similarity between original textures and adversarial textures (AHT/ACT) on RGB maps. We report mean SSIM in $[0, 1]$, where higher values indicate closer resemblance to the original asset.

Table 4: Effectiveness of Adversarial Honeypot Textures (AHT) for misleading aimbots

Material	Brick	Cinderblock	Concrete	Mudbrick	Plaster	Sand	Stone	Total
Original Texture Example								
Adversarial Honeypot Texture Example								
# of Textures	4	2	3	3	6	2	4	24
# of AHT	40	20	30	30	60	20	40	240
# of EoR Sampling	20,480	10,240	15,360	15,360	30,720	10,240	20,480	122,880
Confidence Score	82.5%	75.4%	72.7%	72.1%	79.3%	87.8%	84.2%	79.3%
Uniqueness Rate	98.3%	99.3%	93.3%	95.3%	94.7%	100.0%	99.8%	96.9%
SSIM	70.9%	63.7%	72.5%	70.4%	67.3%	75.3%	77.3%	71.0%
Decoy Success Rate	98.3%	99.3%	93.3%	95.6%	94.7%	100.0%	99.9%	96.9%

7.2 Proactive Defense Effectiveness

To answer **RQ1**, we evaluate the two defenses provided by AIMTRAP: (i) Adversarial Camouflage Textures (ACT) for *evasion* (Tab. 3), and (ii) Adversarial Honeypot Textures (AHT) for *deception* (Tab. 4). **ACT (Evasion)**. We assess ACTs using Evasion Success Rate (ESR), the aimbot’s confidence score, and visual fidelity (SSIM). We consider four distinct player models (covering both male and female characters), and for each model we synthesize ten ACT variants, yielding 40 ACTs in total. To ensure robustness, we adopt Expectation-over-Rendering optimization, randomly sampling 1,280 combinations of various conditions (detailed in §7.5) per ACT (51,200 EoR samples overall). We evaluate the corresponding *original* textures under identical conditions as a baseline.

Tab. 3 shows that ACTs substantially depress the aimbot’s confidence in detecting the player, reducing mean confidence from 62.3% to **16.0%** (a 46.3% drop), well below the default detection threshold 40%. Concurrently, ACTs maintain high perceptual quality with a mean SSIM of 81.0%, indicating strong visual fidelity. The average ESR is **85.1%** across rendering conditions, demonstrating effective proactive evasion against vision-based aimbots.

AHT (Deception). We measure AHTs using Decoy Success Rate (DSR), Uniqueness Rate, aimbot confidence on honeyspots, and SSIM. Following §7.1, we curate 24 real-game textures spanning seven material types (e.g., brick, cinderblock) and synthesize ten AHT variants per texture, producing 240 AHTs in total. For each AHT, we sample 512 EoR condition combinations (122,880 samples overall); condition details are provided in §7.5.

AHTs reliably draw the aimbot’s top-1 detection to the honeypot with high confidence (mean 79.3%), yielding a **DSR of 96.9%**. Moreover, AHTs exhibit a **Uniqueness Rate of 96.9%**, indicating that, in almost all views, exactly *one* honeypot surpasses the threshold—providing a clean, unambiguous signal for downstream cheat detection. Perceptual quality remains acceptable (mean SSIM 71.0%), enabling AHTs to blend into the scene without distracting human players (real player impact study is in §7.7).

7.3 Cheating Honeypot Interaction Identification Performance

Table 5: AIMTRAP’s performance on honeypot-interaction cheating trajectory identification

Accuracy	Precision	Recall	F1	AUC-ROC
98.05%	96.53%	98.98%	97.74%	99.29%

Setup. Following §6, we set the honeypot proximity area parameters to $r_{min} = 60.0$, $r_{max} = 520.0$, and $\alpha = 30.0$, and the aiming determination area to 256×256 pixels.

Dataset. We record honeypot interactions extracted from replay logs on our custom map (§7.1) from two controlled sources: human-honeypot interaction and aimbot-honeypot interaction (configuration: §C). Labels are therefore determined by the recording condition rather than by manual annotation: trajectories from aimbot-assisted interactions are labeled *aimbot* (positive), and those from human-only interactions are labeled *human* (negative). From these logs, we extract honeypot-interaction trajectories with the pipeline of §6.1, i.e., HPZ/ADA geometric gating followed by temporal clustering, yielding 3,074 aiming trajectories in total: 1,326 *aimbot* and 1,748 *human*. We split the data into 2,151/461/462 sequences for training/validation/testing.

Trajectory-Level Detection Results. Tab. 5 summarizes AIMTRAP’s performance on trajectory-level cheating detection: accuracy **98.05%**, precision **96.53%**, recall **98.98%**, and F1 **97.74%**, with **99.29%** AUC-ROC indicating excellent separability. Errors are rare: a small number of false alarms (7/462), while missed cheating trajectories are extremely uncommon (2/462). Consistently, the trajectory-level false positive rate is **2.64%** and the false negative rate is **1.02%**, supporting the model’s suitability for high-recall screening of abnormal honeypot-interaction trajectories.

Player-Level Detection Performance. As discussed in §6.3, we aggregate trajectory-level predictions and flag a player once they exceed the threshold T . Tab. 6 summarizes player-level cheating

Table 6: Player-level cheating detection performance.

T	One-Match		Two-Match		Three-Match	
	TPR _P	FPR _P	TPR _P	FPR _P	TPR _P	FPR _P
1	99.0	2.6×10^{-2}	98.0	7.0×10^{-4}	97.0	1.8×10^{-5}
2	98.0	7.0×10^{-4}	96.0	4.9×10^{-7}	94.0	3.4×10^{-10}
3	97.0	1.8×10^{-5}	94.0	3.4×10^{-10}	91.2	6.2×10^{-15}
4	96.0	4.9×10^{-7}	92.1	2.4×10^{-13}	88.4	1.1×10^{-19}
5	95.0	1.3×10^{-8}	90.3	1.6×10^{-16}	85.7	2.1×10^{-24}

Note: TPR_P in %. We report the compact case $n=T$ (other n : §C). A k -match decision flags a player only when flagged in all k matches.

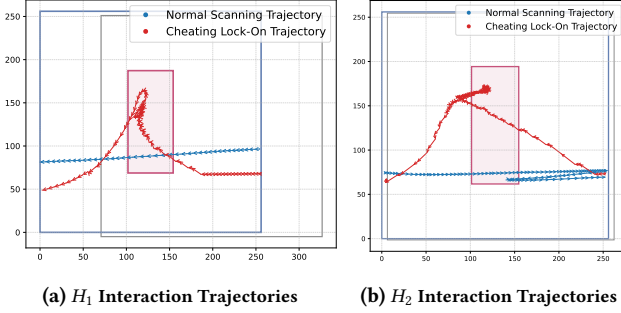


Figure 6: Typical mouse trajectories of aimbots and normal players on two distinct honeypots.

detection for $T \in [1, 5]$ for the compact case $n=T$, given trajectory-level rates (TPR_T = 98.98%, FPR_T = 2.64%). Increasing T sharply lowers the player-level FPR (from 6.97×10^{-4} at $T=2$ to 1.28×10^{-8} at $T=5$ for a single match), and aggregating two matches reduces it by further orders of magnitude, at a modest cost in recall. Because a real match typically yields more than T qualifying interactions, the operational FPR generally exceeds this compact case; we ablate the rates over the realistic range $n \in \{5, 10, 15, 20\}$ in §C, where recall saturates near 100%. At the observed benign mean $n=15$, for instance, a two-match decision at $T=3$ still yields FPR_P = 4.4×10^{-5} while retaining $\approx 100\%$ recall, which we take as AIMTRAP’s operating point. Developers can tune T and the number of matches to balance precision and recall.

Trajectory Visualization and Case Studies. Fig. 6 visualizes representative honeypot-interaction mouse trajectories for aimbot cheaters and normal players. Red rectangles mark two honeypots (H_1 , H_2); gray denotes the wall; blue outlines the Aiming Determination Area (§6). Colored dots plot the aiming trajectory with red denoting aimbot and blue denoting human trajectories. The arrow represents the mouse moving direction. These figures reveal a clear signal: *aimbot trajectories exhibit deliberate lock-ons* that converge tightly on the honeypots, whereas human trajectories pass smoothly through the scene without sustained fixation. Additional visualization cases are available in §C.

7.4 Real-Game Cheating Detection Performance

To answer RQ3, we evaluate AIMTRAP in a real-world CS2 setting and report XGUARDIAN [64] as a complementary reference.

Real-world Gaming Environment Setup. We conducted a paid gameplay user study on our map (§7.1) in CS2’s *Competitive* mode.

Twenty participants each played two matches: one without cheating and one assisted by the aimbot (configuration: §C) to simulate cheating. The participants were blind to the study’s purpose and were recruited from a diverse pool of CS2 players with varying experience levels (from novices to veterans). In total, we collected 40 matches (20 non-cheating, 20 cheating). The configuration of XGUARDIAN is described in the §C.

Overall Results. Tab. 8 summarizes the real-world detection statistics for AIMTRAP. Across all 40 matches, AIMTRAP **correctly** classified every non-cheating and cheating match.

(1) *Non-cheating matches.* Across 20 non-cheating matches, participants triggered $I_A = 306$ honeypot interactions (mean = 15 per match). Only one interaction (by player R) was misclassified, yielding a trajectory-level false positive rate of $1/306 \approx 3.27 \times 10^{-3}$, well below the 2.64% measured on the held-out test set. At the deployment threshold $T \geq 3$, this single stray interaction never accumulates into a player-level flag: AIMTRAP produced **zero** false-flagged players across all 20 benign matches (empirical player-level FPR 0/20). Feeding the observed trajectory FPR into the aggregation model of §6.3 at the match-average $n=15$ projects a player-level FPR of $\approx 1.5 \times 10^{-5}$ at $T=3$ (n -ablation: §C), consistent with observing no false-flagged players. These results provide strong evidence of AIMTRAP’s precision under real-world gameplay.

(2) *Cheating matches.* Across 20 cheating matches, AIMTRAP records $I_A = 325$ honeypot interactions (mean = 16), including $I_C = 202$ cheating trajectories (mean = 10). All players exceeded the threshold $T = 3$ ($I_C = 5-21$), confirming its effectiveness for real-game detection. The gap between I_A and I_C is expected, as the honeypot proximity zone (§6) is recall-oriented and may include interactions where the honeypot is too distant or off-axis for aimbot lock-on.

Complementary-reference Results. When applied to our honeypot dataset, XGUARDIAN (XG) detected 3 of the 20 cheating matches and correctly classified 14 of the 20 non-cheating matches; several additional matches could not be processed successfully. These results highlight the benefit of incorporating a visual-aimbot-specific challenge signal on honeypot maps.

As discussed in §8.2, however, AIMTRAP and XGUARDIAN differ substantially in scope and deployment requirements. XGUARDIAN is trained on large-scale datasets covering diverse cheat tools and configurations, including memory-based and visual aimbots as well as wallhacks. In contrast, AIMTRAP specifically targets visual aimbots by analyzing player trajectories relative to developer-deployed honeypots. Because XGUARDIAN’s datasets contain neither honeypot locations nor honeypot-triggered challenges, AIMTRAP cannot be evaluated on them. Conversely, applying the released XGUARDIAN model to our controlled dataset measures its one-way compatibility under distribution shift rather than establishing a universal comparison between the two systems. Accordingly, the results should not be interpreted as diminishing XGUARDIAN’s broader applicability: unlike AIMTRAP, it requires no deployed honeypots and is designed to detect a wider range of aim-assist and wallhack behaviors.

7.5 Robustness

We assess AIMTRAP’s robustness across various conditions.

Table 7: 3D rendering robustness of Adversarial Camouflage Textures

Yaw		$[-60^\circ, -45^\circ]$	$[-45^\circ, -30^\circ]$	$[-30^\circ, -15^\circ]$	$[-15^\circ, 0^\circ]$	$[0^\circ, 15^\circ]$	$[15^\circ, 30^\circ]$	$[30^\circ, 45^\circ]$	$[45^\circ, 60^\circ]$
									
CD ₀	Conf	36% → 3%	45% → 8%	49% → 8%	61% → 12%	64% → 13%	58% → 9%	62% → 15%	68% → 31%
	ESR	99.2%	94.4%	95.4%	94.7%	91.7%	92.1%	88.2%	66.7%
CD ₁	Conf	62% → 17%	60% → 13%	63% → 17%	71% → 27%	72% → 22%	70% → 18%	75% → 26%	76% → 43%
	ESR	81.2%	83.1%	82.3%	70.9%	78.2%	82.5%	72.4%	39.1%

Note: CD₀ denotes camera distances $\in [0.8, 1.0]$, while CD₁ $\in [1.0, 1.2]$. Conf denotes the change in the aimbot's confidence score before and after the application of ACT. ESR denotes Evasion Success Rate. Camera Pitch is set to $[-2^\circ, 2^\circ]$.

Table 8: Comparison of visual-cheat detection in real-game settings. Green indicates correct predictions.

Player		No Cheating				Cheating			
ID	FPS Exp.	AIMTRAP		XG		AIMTRAP		XG	
		I _A	I _C	Pred	Pred	I _A	I _C	Pred	Pred
A	300–1000	20	0	F	F	25	12	T	F
B	> 1000	17	0	F	F	11	5	T	⊥
C	300–1000	7	0	F	F	16	12	T	F
D	300–1000	14	0	F	⊥	16	9	T	⊥
E	300–1000	12	0	F	F	13	7	T	T
F	< 100	10	0	F	T	10	5	T	F
G	300–1000	12	0	F	F	18	10	T	⊥
H	300–1000	12	0	F	⊥	23	12	T	⊥
I	> 1000	18	0	F	⊥	26	13	T	F
J	100–300	11	0	F	T	11	7	T	F
K	< 100	29	0	F	F	11	8	T	F
L	100–300	19	0	F	F	14	8	T	F
M	300–1000	13	0	F	F	15	12	T	T
N	> 1000	16	0	F	F	23	19	T	F
O	300–1000	7	0	F	F	10	5	T	F
P	< 100	17	0	F	F	16	11	T	F
Q	100–300	10	0	F	T	14	9	T	⊥
R	300–1000	19	1	F	F	17	11	T	T
S	> 1000	24	0	F	F	10	6	T	F
T	100–300	19	0	F	F	26	21	T	F
Average		15	0			16	10		
Overall		306	1	20	14	325	202	20	3

Note: XG is the state-of-the-art detector XGUARDIAN [64]; FPS Exp represents total gameplay experience (hours); I_A, I_C denote total honeypot interactions and cheating-specific interactions per match, respectively; Pred is model prediction output, where $Pred \in \{T, F, \perp\}$. T denotes a cheating classification, F denotes a non-cheating classification, and $\perp$ indicates a processing failure.

(1) *Camera Distance* (d): distance from camera to target (player model for ACT; honeypot for AHT). Default $d = 1.0$; we sweep $d \in [0.8, 1.2]$ (smaller $\Rightarrow$ closer).

(2) *Camera Yaw* (ψ): horizontal rotation with default $\psi = 0^\circ$ (target facing camera). For ACT, we sweep $\psi \in [-60^\circ, 60^\circ]$ for wide lateral views; for AHT, $\psi \in [-20^\circ, 20^\circ]$ suffices since honeypots lie on near-planar walls that require limited yaw range.

(3) *Camera Pitch* (ϕ): vertical rotation. Default $\phi = 0^\circ$. For ACT we sweep $\phi \in [-2^\circ, 2^\circ]$; for AHT we use $\phi \in [-20^\circ, 20^\circ]$ to include steeper up/down views of wall-mounted textures.

Table 9: 3D rendering robustness of AHT.

Yaw		$[-20^\circ, -10^\circ]$	$[-10^\circ, 0^\circ]$	$[0^\circ, 10^\circ]$	$[10^\circ, 20^\circ]$
Pitch $[-20^\circ, 0^\circ]$					
	CD	DSR / Confidence (%)			
CD	[0.8, 0.9]	98.4 / 79.6	97.2 / 79.2	97.2 / 79.6	95.4 / 76.5
	[0.9, 1.0]	97.5 / 79.4	97.2 / 80.4	96.6 / 79.9	96.0 / 78.5
	[1.0, 1.1]	98.8 / 80.2	97.6 / 80.6	96.7 / 79.6	96.3 / 78.0
	[1.1, 1.2]	96.9 / 78.8	96.8 / 79.6	97.2 / 79.0	95.4 / 77.2
Yaw		$[-20^\circ, -10^\circ]$	$[-10^\circ, 0^\circ]$	$[0^\circ, 10^\circ]$	$[10^\circ, 20^\circ]$
Pitch $[0^\circ, 20^\circ]$					
	CD	DSR / Confidence (%)			
CD	[0.8, 0.9]	96.8 / 79.4	97.4 / 79.8	96.1 / 79.1	98.1 / 80.3
	[0.9, 1.0]	96.4 / 77.0	96.5 / 79.9	97.2 / 79.7	97.4 / 78.9
	[1.0, 1.1]	95.7 / 76.2	96.9 / 79.9	97.1 / 80.0	96.9 / 78.4
	[1.1, 1.2]	96.2 / 76.2	95.7 / 78.3	97.0 / 79.7	96.7 / 76.5

Note: CD denotes Camera Distance. DSR denotes Decoy Success Rate. Confidence denotes the aimbot's confidence score.

ACT Robustness. We randomly sample 1,280 EoR combinations per ACT from a Gaussian centered at the defaults (51,200 total). Tab. 7 reports slices over (d, ψ) with fixed pitch $\phi \in [-2^\circ, 2^\circ]$, including aimbot confidence on *original* and *ACT* textures as well as ESR. ACTs are stable across most viewpoints. At extremes (large $|\psi|$ and long distance), ESR drops due to reduced visible surface and foreshortening, suggesting room for further EoR augmentation.

AHT Robustness. We sample 512 EoR combinations per AHT (122,880 renders total). Tab. 9 summarizes honeypot confidence and DSR across representative (d, ψ, ϕ) ranges. AHTs sustain uniformly high deception: all ranges exceed 95% DSR (avg. 96.9%), indicating robust decoys across various rendering conditions.

7.6 Performance Overhead

Server Offline Synthesis Overhead. Tab. 10 shows synthesizing one ACT averages **74.95 s** and AHT **12.05 s**. Since synthesis runs offline to pre-generate texture pools (§4), AIMTRAP does not increase client overhead. Moreover, cross-session reuse (§5.5) amortizes this one-time cost: reusing one texture across 1,000 sessions reduces per-session overhead to < 0.1 s while maintaining sufficient variety.

Client Pre-Game Initialization Overhead. Before matches, the server transmits adversarial textures to clients (§4). Small average

Table 10: Performance overhead on adversarial texture synthesis

	Time Cost			Storage Cost	
	Avg (s)	Min (s)	Max (s)	Resolution	Size (MB)
AHT	12.05	3.08	42.00	640 × 640	0.83
ACT	74.95	28.12	119.98	2048 × 2048	7.48

payloads—0.83 MB per AHT and 7.48 MB per ACT—ensure transfer and asset installation introduce negligible loading time.

Client In-Game Runtime Defense Overhead. AIMTRAP incurs near-zero runtime overhead (Tab. 2). Adversarial textures are standard assets that *replace* existing ones, introducing no additional per-frame computation beyond normal rendering paths. Although file sizes differ slightly (typically smaller than the originals), we observed no measurable latency or FPS degradation.

Server Post-Game Detection Overhead. Evaluated on 40 real-world matches (§7.4), end-to-end post-game detection averages 8.46 s per player per match (8.33 s for trajectory extraction, 0.13 s for cheating inference). This low computational overhead makes AIMTRAP suitable for production-level deployment.

7.7 Real-World Gaming Impact Analysis

We conducted a questionnaire-based user study to assess AIMTRAP’s impact on players in real-world gaming scenarios, covering *Perceptibility*, *Indistinguishability*, and *Naturalness*, including 113 participants with diverse FPS experience (experience breakdown: §C).

Perceptibility. We examine its impact on human perceptibility by asking whether (i) ACT makes player models imperceptible and (ii) AHT misleads them into noticing a human-like target on a wall. Overall, participants reported *no loss of perceptibility* and *no misleading salience*.

Indistinguishability. We tested whether participants could distinguish adversarial textures from the originals. After a brief warm-up with a side-by-side comparison image, participants answered two multiple-choice questions for ACT and AHT with four options: (i) all targets deploy AIMTRAP, (ii) only the first, (iii) only the second, and (iv) none deploy. The correct answers for the two items were (ii) and (i) (green in Fig. 7), respectively. The results show that most participants selected “none deploy,” indicating that the adversarial textures were perceived as visually consistent with the originals. The remaining responses, including the correct options, were dispersed in a near-uniform manner, suggesting that AIMTRAP is difficult to distinguish from standard textures.

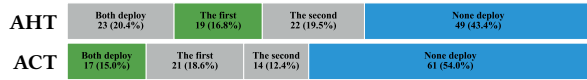


Figure 7: Results of the user study on indistinguishability

Naturalness. Finally, we evaluated naturalness on paired targets: one with AIMTRAP and one without. The participants were asked to compare the AIMTRAP-applied model (both ACT and AHT) to the original. As illustrated in Fig. 8, most participants judged the two as equally natural.

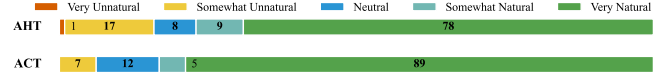


Figure 8: Results of the user study on naturalness

Table 11: Cross-model transferability performance of AHT.

Aimbot Model	AIMTRAP (baseline)		AIMTRAP (ENS+MI+TI)	
	DSR	Conf.	DSR	Conf.
YOLOv5n	98.4	93.6	100.0 (+1.6)	95.2 (+1.6)
YOLOv5s	93.5	79.8	100.0 (+6.5)	95.0 (+15.1)
YOLOv5m	82.6	68.1	91.9 (+9.3)	82.0 (+13.9)
YOLOv5l	81.7	65.0	85.3 (+3.6)	72.2 (+7.1)
YOLOv8n	92.2	76.4	99.2 (+7.0)	82.8 (+6.3)
Faster R-CNN	84.3	74.6	93.3 (+9.0)	90.3 (+15.7)
NanoDet	19.9	10.5	38.9 (+19.0)	21.6 (+11.1)
Average	78.9	66.9	86.9 (+8.0)	77.0 (+10.1)

Note: All values are in %; parentheses show the change from baseline in percentage points, with gains in green and regressions in red. Baseline uses YOLOv5n as proxy, ENS+MI+TI uses YOLOv5n/s.

7.8 Transferability

We evaluate cross-model transferability using *adversarial honeypot textures* (AHTs), since reliable decoying is critical for detecting aimbots in the wild. It is widely recognized that high transferability of adversarial attacks remains difficult [11, 52, 56].

We evaluate AIMTRAP’s transferability under two optimization settings: a baseline setting that uses YOLOv5n as the proxy model, and an enhanced setting that adopts an ensemble proxy of YOLOv5n and YOLOv5s combined with momentum iterative and translation-invariant optimization (ENS+MI+TI, §5.3). For each setup, we synthesize 12 AHTs over 120 fixed optimization iterations without early stopping and evaluate each AHT on 256 EoR samples. The resulting baseline DSR of 93.5% differs from the 96.9% reported in §7.2 because the latter are early-stopped once the proxy is fooled; both of them use YOLOv5n as the proxy model and YOLOv5s as the aimbot model. As Tab. 11 shows, ENS+MI+TI improves average DSR and confidence by 8.0 and 10.1 percentage points, respectively. Every model improves on both metrics under ENS+MI+TI, with the two proxy-family models (YOLOv5n/s) reaching 100.0% DSR. Large DSR gains also appear on Faster R-CNN (+9.0 points) and NanoDet (+19.0 points), the two non-YOLOv5 architectures, although NanoDet remains the hardest target.

8 Discussion

8.1 Adaptive Attack

As with any heuristic- or learning-based defense, AIMTRAP may be subject to adaptive attacks when its design is known. Nevertheless, its complementary mechanisms constrain straightforward evasion strategies and increase the attacker’s cost in realistic settings.

Threshold adjustment provides no simple escape. ACT reduces the aimbot’s detection confidence for real players from 62.3% to 16.0%, whereas AHT raises its confidence for honeypots to 79.3%. These distributions cannot be separated by a single threshold: lowering the threshold to recover ACT-protected players also increases

honeypot lock-on, while raising it to avoid honeypots further suppresses detection of real players.

Retraining is expensive and perishable. First, EoR-based optimization makes adaptation from a single frame ineffective, requiring attackers to collect and label each texture across diverse viewpoints and rendering conditions. Second, in-memory texture delivery limits offline extraction; collecting sufficient observations from live gameplay may itself induce honeypot interactions that are recorded in trajectory logs. Third, randomized initialization seeds, per-session texture selection, and periodic rotation reduce the useful lifetime of harvested training data and require attackers to repeat the adaptation process.

8.2 State-of-the-art Comparison

Comparison with the post-hoc behavior detector XGUARDIAN. XGUARDIAN [64] trains its detection model on large-scale, real-world replay data. Both XGUARDIAN and AIMTRAP analyze player's pitch and yaw data extracted from replay logs. However, AIMTRAP additionally requires maps on which visual honeypots can be deployed. XGUARDIAN provides broad coverage of memory-based and visual aimbots as well as wallhacks. In contrast, AIMTRAP specifically targets visual aimbots through a visual challenge-response mechanism. The two systems are therefore complementary rather than interchangeable: XGUARDIAN offers broader cheat coverage, whereas AIMTRAP provides a targeted detection signal for visual aimbots in honeypot-equipped environments.

Comparison with the proactive runtime defense Invisibility Cloak. Both systems aim to prevent visual aimbots from detecting legitimate players, but use fundamentally different approaches. Invisibility Cloak [49] applies perturbations directly to display-space frames at runtime. By contrast, AIMTRAP's ACT consists of 3D-robust textures that are generated offline, attached statically to player models, and rendered as ordinary game assets, thereby introducing no runtime computational overhead (Tab. 2). The two approaches also impose different deployment costs. Invisibility Cloak requires modifications to the client-side rendering pipeline and incurs additional runtime computation. AIMTRAP requires honeypot-deployed maps to support its visual challenge-response detection mechanism, though the deployment effort is minimal: adversarial textures are standard assets that simply replace their original counterparts, leaving the rendering pipeline untouched.

9 Conclusion

This paper presents AIMTRAP, an anti-visual-aimbot framework that can not only proactively interfere with aimbot behavior at runtime without introducing performance overhead, but also provide honeypot-based detection signals for explainable cheating detection. Our real-game evaluations demonstrate a practical, robust, and deployable anti-visual-cheat pipeline that counters vision-based aimbots without game-engine modifications or runtime cost and offers effective and efficient offline cheating detection.

Acknowledgments

We thank our shepherd and anonymous reviewers for their insightful comments. We thank Jun Yu and Shaofei Li for their help on our experiments. This paper was edited for grammar using ChatGPT

and Gemini. This work was supported by Key R&D Program of Shandong Province, China (No. 2026CXPT269, 2024CXGC010114, 2025CXPT085), National Natural Science Foundation of China under Grant (No. 62372268), Shandong Provincial Natural Science Foundation, China (No. ZR2022LZH013, No. ZR2021LZH007).

References

- [1] Tripo AI. 2025. *Create Your First 3D Model with Text and Image in Seconds*. <https://www.tripo3d.ai/>
- [2] ai aimbot. 2025. *AIMr: The best AI Aimbot for Fortnite, Valorant, CS2, R6, COD, Apex, EVERYTHING (Good PC Recommended)*. <https://github.com/ai-aimbot/AIMr>
- [3] AimTrap. 2026. *AimTrap Artifacts*. <https://github.com/Ji4n1ng/AimTrap>
- [4] Hashem Alayed, Fotos Frangoudes, and Clifford Neuman. 2013. Behavioral-based cheating detection in online first person shooters using machine learning techniques. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*. 1–8. doi:10.1109/CIG.2013.6633617
- [5] Easy Anti-Cheat. 2025. *Easy Anti-Cheat - the industry-leading anti-cheat service, countering hacking and cheating in multiplayer PC games*. <https://www.easyanticheat.net/>
- [6] Riot's anti-cheat Vanguard team. 2024. *Vanguard x VALORANT*. <https://playvalorant.com/en-us/news/game-updates/vanguard-x-valorant/>
- [7] Anybrain. 2023. *How Cheating in Video Games Impacts Game Developers*. <https://www.linkedin.com/pulse/how-cheating-video-games-impacts-game-developers-anybraing-1f/>
- [8] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2018. Synthesizing robust adversarial examples. In *International conference on machine learning*. PMLR, 284–293.
- [9] BattlEye. 2025. *BattlEye - The Anti-Cheat Gold Standard*. <https://www.battleye.com/>
- [10] Tom B Brown, Dandelion Mané, Aurko Roy, Martin Abadi, and Justin Gilmer. 2017. Adversarial patch. *arXiv preprint arXiv:1712.09665* (2017).
- [11] Zikui Cai, Xinxin Xie, Shasha Li, Mingjun Yin, Chengyu Song, Srikanth V Krishnamurthy, Amit K Roy-Chowdhury, and M Salman Asif. 2022. Context-aware transfer attacks for object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 149–157.
- [12] Selina Cho, Jonathan Lusthaus, and Ivan Flechais. 2024. Unpacking the Dynamics of Harm in Game Cheating Communities: A Guiding Framework for Cross-Industry Intervention. *ACM Games* 2, 2, Article 12 (Aug. 2024), 17 pages.
- [13] Minyeop Choi, Gihyuk Ko, and Sang Kil Cha. 2023. BotScreen: Trust Everybody, but Cut the Aimbots Yourself. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 481–498. <https://www.usenix.org/conference/usenixsecurity23/presentation/choi>
- [14] Technical CITY. 2026. *GeForce RTX 4090 vs RTX 3060*. <https://technical.city/en/video/GeForce-RTX-3060-vs-GeForce-RTX-4090>
- [15] Valve Corporation. 2025. *Hammer - Counter-Strike 2 Workshop Tools*. https://developer.valvesoftware.com/wiki/Counter-Strike_2_Workshop_Tools
- [16] Valve Corporation. 2026. *Steam Hardware and Software Survey: March 2026*. <https://store.steampowered.com/hwsurvey/videocard/>
- [17] Luca Galli, Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. 2011. A cheating detection framework for Unreal Tournament III: A machine learning approach. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. 266–272. doi:10.1109/CIG.2011.6032016
- [18] GlobeNewswire. 2025. *Cheating in Asian Gaming: The Hidden Costs and Impact on Revenue*. <https://www.globenewswire.com/news-release/2025/04/04/3056006/0/en/Cheating-in-Asian-Gaming-The-Hidden-Costs-and-Impact-on-Revenue.html>
- [19] Mee Lan Han, Jung Kyu Park, and Huy Kang Kim. 2015. Online Game Bot Detection in FPS Game. In *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems - Volume 2*. Springer International Publishing, Cham, 479–491.
- [20] Basically Homeless. 2025. *Neuromuscular Aim Assist*. <https://youtu.be/9aJwQG-Wbk?si=OhXxBYmpDJFYyl5h>
- [21] Matthew Hull, Chao Zhang, Zsolt Kira, and Duen Horng Chau. 2024. Adversarial Attacks Using Differentiable Rendering: A Survey. *arXiv e-prints* (2024), arXiv–2411.
- [22] HYPER3D. 2025. *Rodin AI - Free AI 3D Model Generator For Everyone*. <https://hyper3d.ai/>
- [23] Anssi Kanervisto, Tomi Kinnunen, and Ville Hautamäki. 2023. GAN-Aimbots: Using Machine Learning for Cheating in First Person Shooters. *IEEE Transactions on Games* 15, 4 (2023), 566–579. doi:10.1109/TG.2022.3173450
- [24] Daiping Liu, Xing Gao, Mingwei Zhang, and Haining Wang. 2016. Shoot for the Moon and You Will Never Miss: Characterizing and Detecting Aimbots in Online Games. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science (Antibes Juan-les-Pins, France) (SIGMETRICS '16)*. Association for Computing Machinery, New York, NY, USA,

- 401–402. doi:10.1145/2896377.2901503
- [25] Daiping Liu, Xing Gao, Mingwei Zhang, Haining Wang, and Angelos Stavrou. 2017. Detecting Passive Cheats in Online Games via Performance-Skillfulness Inconsistency. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 615–626. doi:10.1109/DSN.2017.20
- [26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).
- [27] Meshy. 2025. *The Easiest Way to Create 3D Models*. <https://www.meshy.ai/>
- [28] Microsoft. 2025. *Xbox Cloud Gaming*. <https://www.xbox.com/en-US/play>
- [29] National Cyber Crime Unit NCCU. 2017. *National Crime Agency (NCA): Pathways Into Cybercrime*. <https://www.nationalcrimeagency.gov.uk/who-we-are/publications/6-pathways-into-cyber-crime-1/file>
- [30] Anh Nhu, Hieu Phan, Chang Liu, and Xianglong Feng. 2023. A Comprehensive Defense Approach Targeting The Computer Vision Based Cheating Tools in FPS Video Games. In *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. 168–177. doi:10.1109/IPCCC59175.2023.10253881
- [31] Binghua Nie and Bin Ma. 2024. VADNet: Visual-Based Anti-Cheating Detection Network in FPS Games. *Traitement du Signal* 41 (02 2024), 431–440. doi:10.18280/ts.410137
- [32] NTUYWANG103. 2025. *APEX AIMBOT: This is a YOLOV7 based APEX and CSGO Aimbot*. https://github.com/NTUYWANG103/APEX_AIMBOT
- [33] NVIDIA. 2025. *GeForce NOW - The Next Generation in Cloud Gaming*. <https://www.nvidia.com/en-us/geforce-now/>
- [34] Seonghyun Park, Adil Ahmad, and Byoungyoung Lee. 2020. BlackMirror: Preventing Wallhacks in 3D Online FPS Games. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, USA) (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 987–1000. doi:10.1145/3372297.3417890
- [35] Apex Gaming PCs. 2025. *A Guide to FPS in Gaming: Why It Is Important*. <https://apexgamingpcs.com/blogs/apex-support/what-is-fps-gaming>
- [36] Camilo Pestana, Naveed Akhtar, Nazanin Rahnavard, Mubarak Shah, and Ajmal Mian. 2022. Transferable 3d adversarial textures using end-to-end optimization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 88–97.
- [37] José Pedro Pinto, André Pimenta, and Paulo Novais. 2021. Deep learning and multivariate time series for cheat detection in video games. *Machine Learning* 110, 11–12 (Dec. 2021), 3037–3057. doi:10.1007/s10994-021-06055-x
- [38] ACTIVE PLAYER. 2025. *Counter-Strike 2 Live Player Count and Statistics*. <https://activeplayer.io/counter-strike-2/>
- [39] RangLiYu. 2020. *NanoDet-Plus: Super fast and lightweight anchor-free object detection model*. <https://github.com/RangLiYu/nanodet>
- [40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.
- [41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).
- [42] RootKit-Org. 2025. *AI Aimbot: World's Best AI Aimbot - CS2, Valorant, Fortnite, APEX, every game*. <https://github.com/RootKit-Org/AI-Aimbot>
- [43] Shouwei Ruan, Yinpeng Dong, Hang Su, Jianteng Peng, Ning Chen, and Xingxing Wei. 2023. Towards viewpoint-invariant visual recognition via adversarial training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4709–4719.
- [44] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. 2004. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (Washington DC, USA) (CCS '04)*. Association for Computing Machinery, New York, NY, USA, 298–307. doi:10.1145/1030083.1030124
- [45] Sony. 2025. *PlayStation Plus Premium*. <https://www.playstation.com/en-sg/ps-plus/>
- [46] Ruan Spijkerman and Elizabeth Marie Ehlers. 2021. Cheat Detection in a Multiplayer First-Person Shooter Using Artificial Intelligence Tools. In *Proceedings of the 2020 3rd International Conference on Computational Intelligence and Intelligent Systems (Tokyo, Japan) (CIIS '20)*. Association for Computing Machinery, New York, NY, USA, 87–92. doi:10.1145/3440840.3440857
- [47] Statista. 2025. *Share of video gamers worldwide who have played games in select gaming genres in the past 12 months as of 3rd quarter 2024*. <https://www.statista.com/statistics/240990/global-online-games-genre-breakdown>
- [48] Statista. 2025. *Video gaming market size worldwide 2022-2032*. <https://www.statista.com/statistics/292056/video-game-market-value-worldwide/>
- [49] Chenxin Sun, Kai Ye, Liangcai Su, Jiayi Zhang, and Chenxiong Qian. 2024. Invisibility Cloak: Proactive Defense Against Visual Game Cheating. In *33rd USENIX Security Symposium (USENIX Security 24)*. 3045–3061.
- [50] SunOner. 2025. *Sunone Aimbot: Aim-bot based on AI for all FPS games*. https://github.com/SunOner/sunone_aimbot
- [51] SwissChili. 2023. *Putting the AI in Aimbot*. <https://swisschili.sh/putting-the-ai-in-aimbot>
- [52] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. 2019. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 0–0.
- [53] YouWei Trade. 2025. *What Hz do pro gamers use?* <https://youweitrade.com/blogs/blog/what-hz-do-pro-gamers-use>
- [54] Valve. 2025. *VAC - Valve Anti-Cheat System*. <https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74>
- [55] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.
- [56] Xingxing Wei, Siyuan Liang, Ning Chen, and Xiaochun Cao. 2018. Transferable adversarial attacks for image and video object detection. *arXiv preprint arXiv:1811.12641* (2018).
- [57] Wikipedia. 2025. *UV Mapping*. https://en.wikipedia.org/wiki/UV_mapping
- [58] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. 2020. Adversarial t-shirt! evading person detectors in a physical world. In *European conference on computer vision*. Springer, 665–681.
- [59] Kai Yan, Xianyi Chen, Qi Cui, Haoqin Yuan, and Zhenshan Tan. 2025. AdvMap: Crafting Adversarial Maps to Counter AI Aimbot in First-Person Shooter Games. *IEEE Transactions on Games* (2025), 1–12. doi:10.1109/TG.2025.3537824
- [60] S.F. Yeung, J.C.S. Lui, Jiangchuan Liu, and J. Yan. 2006. Detecting cheaters for multiplayer games: theory, design and implementation. In *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference*, 2006., Vol. 2. 1178–1182. doi:10.1109/CCNC.2006.1593224
- [61] Su-Yang Yu, Nils Hammerla, Jeff Yan, and Peter Andras. 2012. Aimbot detection in online FPS games using a heuristic method based on distribution comparison matrix. In *Proceedings of the 19th International Conference on Neural Information Processing - Volume Part V (Doha, Qatar) (ICONIP'12)*. Springer-Verlag, Berlin, Heidelberg, 654–661. doi:10.1007/978-3-642-34500-5_77
- [62] Su-Yang Yu, Nils Hammerla, Jeff Yan, and Peter Andras. 2012. A statistical aimbot detection method for online FPS games. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 1–8. doi:10.1109/IJCNN.2012.6252489
- [63] Jiayi Zhang, Chenxin Sun, Yue Gu, Qingyu Zhang, Jiayi Lin, Xiaojiang Du, and Chenxiong Qian. 2026. Identify as a Human Does: A Pathfinder of Next-Generation Anti-Cheat Framework for First-Person Shooter Games. *IEEE Transactions on Information Forensics and Security* 21 (2026), 240–255. doi:10.1109/TIFS.2025.3635024
- [64] Jiayi Zhang, Chenxin Sun, and Chenxiong Qian. 2026. XGuardian: Towards Explainable and Generalized AI Anti-Cheat on FPS Games. *arXiv:2601.18068 [cs.CR]* <https://arxiv.org/abs/2601.18068>

A Ethical Considerations

The Institutional Review Board (IRB) approved this study, and all participants gave informed consent. We minimized risk, ensured equitable treatment, and collected only self-reported FPS gameplay hours to assess robustness across experience levels; no sensitive data were recorded. All experiments involved only enrolled participants on a private game server, preventing bystander exposure. We neither developed nor endorse the evaluated aimbot; it was used only under controlled conditions for research purposes.

B Open Science

Artifacts of AIMTRAP are available at [3]. Due to legal and ethical constraints, we cannot release the official CS2 textures or raw replay logs containing sensitive account identifiers. We instead provide asset metadata, synthetic replacement textures, de-identified replay data, derived trajectories with labels, and the necessary processing scripts.

C Full Version

Due to space constraints, we omit some technical details from this paper. The full version, including the omitted material in the appendix, is available at [arXiv:2606.25734](https://arxiv.org/abs/2606.25734).