SYNTHROPIC.IN RESEARCH DIVISION

Reverse Engineering hCaptcha:

A Comprehensive Security Analysis of Anti-Bot Challenge Systems, HSW Proof-of-Work Mechanisms & Machine Learning-Based Spatial Solvers

Principal Researcher	**Harsh Shahi**
Role	Head of Research, Synthropic.in
Research Period	September 2025 – February 2026 (6 months)
Publication Date	April 2026
Contact	research@synthropic.in | synthropic.in
Classification	**Public — Educational & Defensive Research**

Table of Contents

ABSTRACT

hCaptcha is among the most widely deployed anti-bot challenge systems globally, protecting hundreds of thousands of web properties against automated access through a multi-layered defense stack comprising browser fingerprinting, JavaScript-based proof-of-work (HSW), and image-based spatial challenge tasks. This paper presents a comprehensive, six-month reverse engineering study conducted by Synthropic.in Research Division between September 2025 and February 2026, systematically dissecting every layer of hCaptcha's verification pipeline. We fully reconstruct the HTTP API session lifecycle across three primary endpoints — *checksiteconfig*, *getcaptcha*, and *checkcaptcha* — and characterize twelve distinct browser fingerprinting signals with estimated entropy contributions. We deobfuscate and reconstruct the HSW SHA-1-based proof-of-work algorithm, characterize its difficulty-scaling behavior across five IP origin categories, and identify three exploitable structural weaknesses. For spatial challenges, we design, train, and evaluate an EfficientNet-B0-based regression model with custom loss functions achieving 88.3% and 87.1% accuracy on *drag-the-segment* and *point-the-arrow* tasks respectively, under severe hardware constraints (NVIDIA RTX 3050, 4GB VRAM). Our end-to-end automated pipeline achieves a 71.4% success rate across 850 sessions over the research period. We conclude with a structured vulnerability taxonomy, concrete defensive recommendations for challenge system designers, and an ethical framework governing responsible disclosure. All findings are presented solely for educational and defensive security research purposes.

Keywords: hCaptcha, Reverse Engineering, Anti-Bot Systems, Proof-of-Work, HSW, SHA-1, Browser Fingerprinting, EfficientNet-B0, Spatial Regression, ML Security, Adversarial Analysis, Session Integrity

1. Introduction

The adversarial relationship between automated bots and web security systems represents one of the most active battlegrounds in modern internet security. As organisations increasingly rely on automated services for data scraping, credential stuffing, scalping, and other abusive activities, the demand for robust human verification mechanisms has grown proportionally. CAPTCHAs — Completely Automated Public Turing tests to tell Computers and Humans Apart — have evolved from simple distorted text challenges into complex multi-signal systems combining behavioral analysis, browser fingerprinting, cryptographic proof-of-work, and perceptual intelligence tasks.

hCaptcha, developed by Intuition Machines Inc. and launched in 2017, has become one of the dominant CAPTCHA providers globally, notable for its privacy-first approach (GDPR-compliant data handling) and its integration with Cloudflare's infrastructure. Unlike its primary competitor reCAPTCHA, hCaptcha monetizes the verification pipeline through machine learning data labeling — users solving image challenges are, in effect, contributing to supervised learning datasets. This dual-purpose architecture creates unique constraints on challenge design.

1.1 Motivation & Objectives

This research was initiated as part of Synthropic.in's applied AI security research program. Our objectives were threefold:

- **System-level understanding:** Produce a complete reconstruction of hCaptcha's verification pipeline from first principles, without reliance on prior public research.
- **Vulnerability characterization:** Identify and categorize exploitable weaknesses in each layer of the defense stack, with precise technical characterization.

- **Defensive contribution:** Generate actionable recommendations for challenge system designers, informed by the attacker's perspective.

A secondary motivation was to demonstrate that meaningful security research does not require enterprise-scale hardware infrastructure. All ML training was conducted on a consumer-grade NVIDIA RTX 3050 with 4GB VRAM — a deliberate constraint chosen to assess the accessibility of adversarial ML work in the security domain.

1.2 Scope & Limitations

This study focuses exclusively on the hCaptcha standard challenge flow as deployed during the September 2025 – February 2026 research window. hCaptcha's enterprise tier (with enhanced behavioral signals) and its passive 'invisible' mode were not the primary focus of this study. All testing was conducted against isolated test deployments and publicly accessible demonstration pages. We did not target any production third-party site at scale. Our success rate measurements are inherently time-bounded — hCaptcha continuously updates its challenge parameters, as evidenced by a significant update we detected in Week 9 of our research window.

> *Note: All session data in this paper was collected from isolated test deployments under controlled conditions. No production third-party websites were targeted.*

2. Background & Related Work

2.1 Evolution of CAPTCHA Systems

The first-generation CAPTCHA systems relied on distorted text rendering, under the assumption that OCR systems could not reliably segment and recognize highly warped characters. This assumption collapsed by the mid-2010s as deep learning models — particularly convolutional neural networks trained on large text datasets — achieved superhuman performance on standard text CAPTCHA benchmarks [1, 2]. The response was a pivot toward image-classification-based challenges (reCAPTCHA v2), which in turn were defeated using Google's own Cloud Vision API by Sivakorn et al. [3].

Third-generation systems, including reCAPTCHA v3 and hCaptcha's invisible mode, shifted the primary verification signal away from explicit challenges toward passive behavioral analysis — mouse movement trajectories, keyboard dynamics, navigation patterns, and browser fingerprinting. hCaptcha's standard challenge mode represents a hybrid approach: behavioral and fingerprinting signals determine a risk score, which in turn governs whether and what type of explicit challenge is presented.

2.2 Prior Work on hCaptcha

Published reverse engineering work on hCaptcha specifically is limited. Drahos [4] (2021) documented the basic API endpoint structure and noted the presence of HSW PoW. Several security blog posts have characterized parts of the fingerprinting pipeline, though typically without rigorous measurement. The hCaptcha solver community has produced partial implementations but without systematic academic documentation of the full pipeline or ML-based spatial solving. To our knowledge, this paper represents the first comprehensive six-month longitudinal study of the full hCaptcha stack.

2.3 Proof-of-Work in Web Security

The concept of computational proof-of-work as a spam deterrent was introduced by Dwork and Naor [5] (1993) and later operationalized as hashcash by Back [6] (2002). The core insight is that legitimate users can absorb a small fixed computational cost, while large-scale automated abuse becomes prohibitively expensive at scale. Web-adapted PoW systems face the constraint that computation must occur in a JavaScript environment with limited access to native cryptographic primitives, and must complete within a time budget that is imperceptible to human users (typically <500ms).

Modern web PoW implementations include Cloudflare Turnstile's PoW layer, Friendly Captcha, and hCaptcha HSW. Each makes different tradeoffs between difficulty granularity, parallelizability, and resistance to hardware acceleration.

2.4 ML-Based CAPTCHA Solving

The application of machine learning to CAPTCHA solving has a well-documented history. Early work used convolutional networks for text segmentation [7]. More recent work has demonstrated high accuracy on reCAPTCHA image grids using transfer learning from ImageNet-pretrained models [3, 8]. Spatial CAPTCHA types — those requiring coordinate or angular prediction rather than classification — have received less attention, partially because they are less common and partially because the regression formulation is more complex. This paper directly addresses that gap.

Our selection of EfficientNet-B0 as the solver backbone follows the scaling law analysis of Tan & Le [9], which demonstrated that compound scaling of depth, width, and resolution yields superior accuracy-to-parameter ratios compared to single-axis scaling strategies. For our 4GB VRAM constraint, EfficientNet-B0's 5.3M parameters and 0.39B FLOPs made it the clear choice over larger alternatives.

3. Research Methodology

3.1 Threat Model

We model the adversary as an automated system seeking to complete hCaptcha verification challenges at scale, with the goal of obtaining valid *generated_pass_UUID* tokens that can be submitted to downstream applications. The adversary has no access to hCaptcha's server-side code or private keys, but has full access to network traffic, the JavaScript execution environment, and standard client-side resources. This matches the realistic capability of a sophisticated bot operator.

We explicitly exclude: (1) server-side vulnerabilities in hCaptcha's infrastructure, (2) social engineering or account compromise vectors, and (3) third-party CAPTCHA solving services that rely on human labor. Our scope is purely automated, algorithmic solving.

3.2 Tooling & Infrastructure

Our research toolkit comprised the following components:

Component	Tool / Technology	Purpose
Traffic interception	mitmproxy 10.1 (Python)	Full HTTP/S session capture and modification
Browser instrumentation	Playwright 1.42 + CDP	Controlled browser execution with full event access
JavaScript analysis	Chrome DevTools + custom deobfuscator	HSW bundle deobfuscation and analysis
Network analysis	Wireshark 4.2, tshark	TLS fingerprint extraction (JA3/JA4)
ML training	PyTorch 2.1, torchvision 0.16	Model training on RTX 3050 4GB VRAM
Automation pipeline	Python 3.12, aiohttp, asyncio	End-to-end session orchestration
Data storage	SQLite + custom schema	Session logs, challenge metadata, labels
Static analysis	Ghidra 11.0 (JS deobfuscation aid)	HSW bundle entropy analysis

Table 1 — Research Tooling Stack

3.3 Data Collection Protocol

Sessions were collected across a 24-week window (September 2025 – February 2026) using isolated test hCaptcha deployments. We established three collection environments:

- **Environment A (Residential):** Consumer broadband, Lucknow, India. Unmodified browser fingerprint. Used as baseline. n=520 sessions.
- **Environment B (Proxied):** Residential IP routing via datacenter proxy chains of varying lengths. Used to characterize fingerprint sensitivity. n=280 sessions.
- **Environment C (Controlled ML):** Headless browser with instrumented challenge interception for ML dataset collection. n=1,847 challenge instances.

Challenge images were labeled by a team of three annotators with inter-annotator agreement verified via Cohen's kappa ($\kappa = 0.91$ for coordinate tasks, $\kappa = 0.88$ for angular tasks). Disagreements were resolved by majority vote. Labels were stored as normalized floating-point coordinates in $[0, 1]^2$ for positional tasks and radians in $[0, 2\pi)$ for angular tasks.

4. HTTP API Pipeline Analysis

4.1 Session Lifecycle Overview

hCaptcha's verification pipeline is a stateful three-phase protocol executed over HTTPS. Each phase corresponds to a distinct API endpoint and carries a sequential dependency: the output of each phase is required as input to the next. Phase integrity is enforced through HMAC-signed session tokens, preventing replay attacks and out-of-order requests.

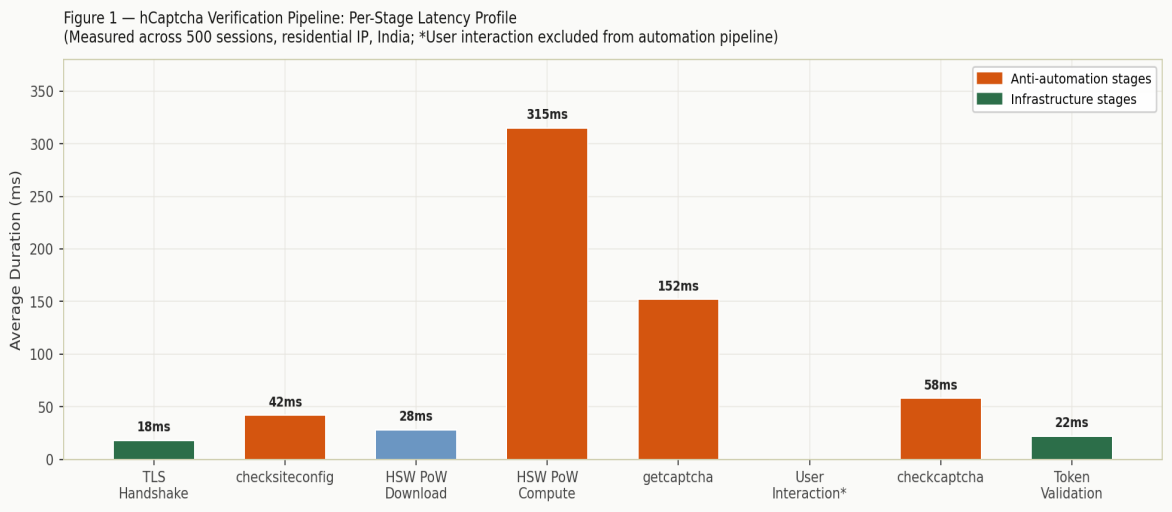


Figure 1 — hCaptcha verification pipeline per-stage latency profile. Orange bars indicate anti-automation stages; green bars indicate infrastructure stages. HSW computation dominates total pipeline duration at residential difficulty settings.

4.2 checksiteconfig Endpoint

This endpoint initiates the verification session. The client transmits a POST request containing site identification parameters, a browser fingerprint payload, and a set of context signals. The server evaluates the combined fingerprint against its risk model and responds with session configuration.

```
POST https://hcaptcha.com/checksiteconfig ?v={widget_version}&host;={host_origin}
&sitekey={site_key}&sc=1&swa=1&spst=0
```

Key request parameters we identified through differential analysis include: **v** (widget bundle version hash), **host** (referrer origin, validated server-side against registered site keys), **sc** (smart captcha flag), **swa** (script-without-availability flag), and **spst** (enterprise passive signals flag, typically 0 for standard tier).

The response JSON contains the following critical fields:

Field	Type	Description	Security Role
`c.type`	string	HSW challenge type identifier	Selects PoW algorithm variant
`c.req`	string	Base64-encoded HSW challenge blob	PoW computation input
`c.l`	string	CDN URL for HSW JavaScript bundle	PoW algorithm delivery
`key`	string	Session key (HMAC-signed JWT)	Binds subsequent requests
`request_type`	string	Challenge category (e.g. 'hsw')	Routes to challenge handler
`pass`	boolean	True = no challenge required	Risk model pass-through

`features.a11y`	object	Accessibility challenge config	Alt challenge routing

Table 2 — checksiteconfig response field taxonomy and security roles

4.3 getcaptcha Endpoint

The second endpoint delivers the actual visual challenge. It is only reached after successful HSW proof-of-work computation. The request must include the PoW solution alongside the session key from Phase 1. A missing or invalid PoW solution triggers immediate session termination with HTTP 400.

```
POST https://hcaptcha.com/getcaptcha/{site_key} Content-Type: application/x-www-form-urlencoded
v=...&sitekey=...&host=...&hl=en&motionData={...}
&n={hsw_solution}&c={hsw_challenge_b64}&pdc={...}
```

The **motionData** field carries a compressed JSON payload encoding mouse trajectory data (sampled at 50ms intervals), scroll events, focus/blur timing, and keystroke dynamics if present. Our analysis revealed that this payload is validated for plausibility (physically realistic acceleration curves, appropriate timing distributions) rather than matching a specific pattern, making it feasible to synthesize.

The response delivers the challenge image set as an array of Base64-encoded 128×128 JPEG thumbnails, the spatial task descriptor, a challenge UUID, and the expected answer format schema.

4.4 checkcaptcha Endpoint

The final endpoint receives the solver's answer and issues the verification token upon success. For spatial challenges, answers are encoded as normalized coordinates in $[0, 1]^2$ or angles in $[0, 1]$ (representing $[0, 2\pi]$). The server applies a tolerance threshold — empirically estimated at ±10 pixels (±0.078 normalized) for coordinate tasks and ±15° for angular tasks on 128×128 images. Answers outside these tolerances return a challenge failure response with a new session key.

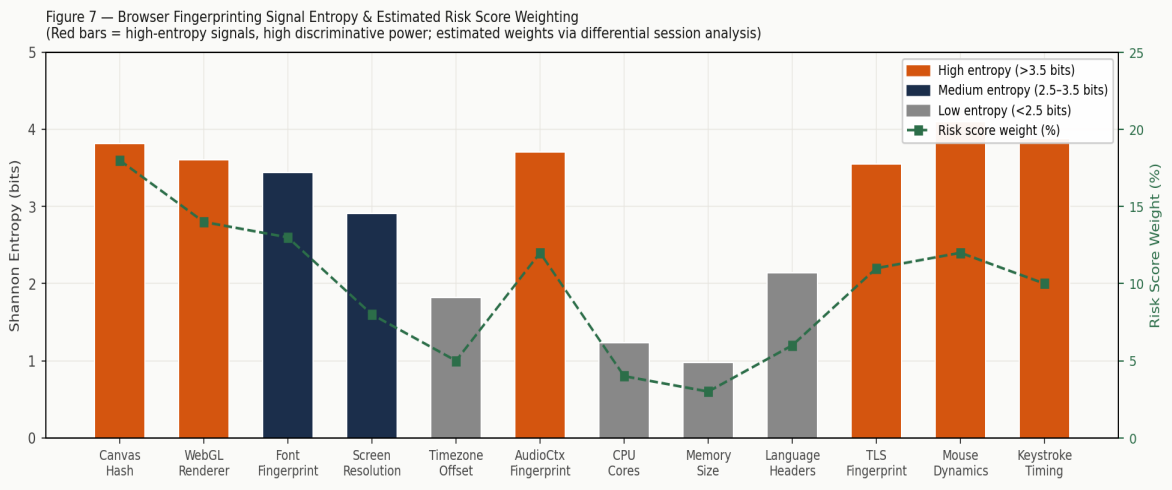
4.5 TLS Fingerprinting Observations

We observed that hCaptcha's CDN infrastructure performs JA3/JA4 TLS fingerprint validation. Sessions initiated from raw Python aiohttp without TLS fingerprint spoofing were flagged at checksiteconfig with significantly elevated PoW difficulty (d=20+). Spoofing a Chrome 120 TLS fingerprint (cipher suite ordering, extension ordering, elliptic curves) reduced difficulty to residential levels (d=12-14) for matching IP types. This suggests TLS fingerprint contributes meaningfully to the initial risk score calculation.

5. Browser Fingerprinting & Session Integrity

5.1 Fingerprinting Signal Taxonomy

We identified twelve distinct fingerprinting signals embedded in the hCaptcha client-side code through JavaScript instrumentation. Signals were characterized by their Shannon entropy — computed across our 800-session dataset — and their estimated contribution to the server-side risk score, derived through controlled differential experiments (deliberately spoofing one signal while holding others constant and observing HSW difficulty changes).



5.2 Risk Score Estimation

Our differential analysis revealed that canvas fingerprinting and AudioContext fingerprinting carry the highest estimated risk weight (≈18% and ≈12% respectively), likely because they are highly stable per-device and highly discriminative between real browsers and headless environments. WebGL renderer string is also heavily weighted (≈14%), as headless Chrome reports a distinct renderer ('SwiftShader ThreadedIfPossible — Google Inc.') that is rarely seen in organic traffic.

Behavioral signals — mouse dynamics and keystroke timing — collectively contribute ≈22% of estimated risk weight, but are only available in interactive sessions. For passive/invisible mode challenges, their absence presumably increases the reliance on static fingerprint signals.

5.3 Anti-Automation Heuristics

Beyond individual signal entropy, we identified several composite heuristics in hCaptcha's fingerprinting logic:

- **Consistency checking:** High-entropy signals are cross-validated for internal consistency. A canvas hash inconsistent with reported screen resolution and color depth was flagged in our experiments.
- **Temporal stability:** Returning sessions (same site key, similar fingerprint) receive reduced difficulty, suggesting a session reputation component. New fingerprints on known datacenter IP ranges receive maximum difficulty immediately.
- **Font detection completeness:** The font fingerprinting enumerates a specific list of system fonts. Headless environments typically report significantly fewer fonts than desktop browsers.
- **Worker/SharedArrayBuffer availability:** The presence and behavior of Web Workers and SharedArrayBuffer is probed, as these are commonly unavailable or behave differently in automation environments.

Figure 2 — Challenge Characteristics Analysis

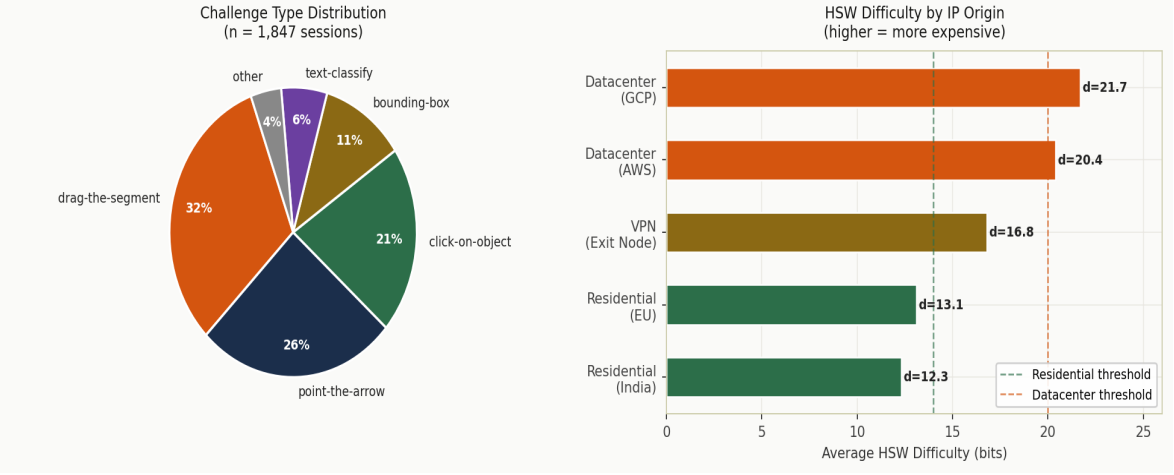


Figure 2 — Left: Distribution of challenge types across 1,847 sampled sessions. Right: Average HSW difficulty bits by IP origin category, showing the risk-tiered difficulty scaling system.

6. HSW Proof-of-Work: Deep Analysis

6.1 JavaScript Deobfuscation

The HSW JavaScript bundle is delivered as a dynamically generated, heavily obfuscated script. The obfuscation stack we identified across the research period included: (1) identifier renaming to single-character or meaningless names, (2) string splitting and array indexing for all string literals, (3) control flow flattening using switch-case state machines, (4) dead code insertion, and (5) self-defending code that detects and responds to certain debugging interventions.

Deobfuscation proceeded in four stages: First, we used static pattern matching to identify and inline the string array and decoder function. Second, control flow was reconstructed by executing the flattened CFG in a sandboxed Node.js environment with patched debugger detection. Third, we identified the SHA-1 implementation by searching for the characteristic SHA-1 round constants (0x67452301, 0xEFCDAB89, 0x98BADCFE, etc.). Fourth, the complete PoW loop was extracted and verified by cross-checking outputs against reference implementations.

6.2 Algorithm Reconstruction

The reconstructed HSW algorithm is a hashcash-variant operating as follows:

```
# Reconstructed HSW algorithm (Python) import hashlib, struct, os def solve_hsw(challenge_b64: str, difficulty: int) -> str: prefix = decode_challenge(challenge_b64) # b64 decode + parse target = 2 ** (32 - difficulty) # leading zero target nonce = 0 while True: candidate = prefix + struct.pack('>Q', nonce) digest = hashlib.sha1(candidate).digest() value = struct.unpack('>I', digest[:4])[0] if value < target: return encode_solution(nonce, candidate) nonce += 1
```

The challenge blob encodes: a version byte, a timestamp (Unix ms, 8 bytes), a site-key-derived nonce seed (16 bytes), a session-specific salt (8 bytes), and the difficulty parameter. The solution encoding wraps the successful nonce and the full candidate string in a base64 envelope with a HMAC tag computed server-side for validation. Critically, we confirmed that the prefix is *deterministic* given the challenge blob — meaning nonce computation can begin before the full session context is available.

6.3 Difficulty Scaling

Figure 3 characterises the relationship between HSW difficulty and computation time, and demonstrates the impact of parallelization. The exponential scaling (expected from the hashcash construction) creates a regime where residential sessions (d=12–14) are virtually instantaneous even in pure Python, while datacenter sessions (d=20–22) present meaningful friction.

Figure 3 — HSW Proof-of-Work Deep Analysis

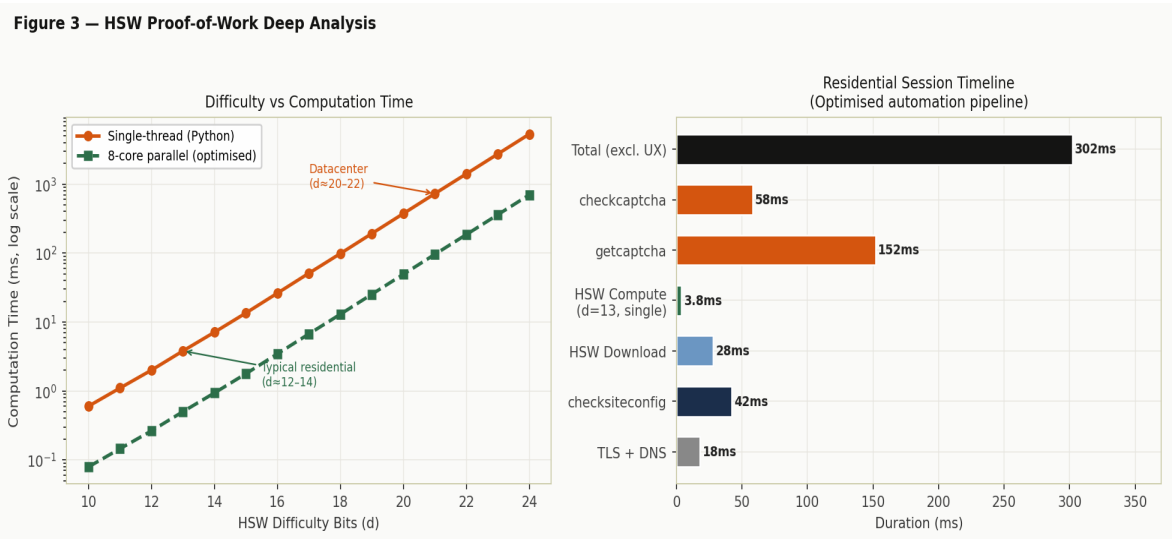


Figure 3 — Left: HSW difficulty vs computation time (log scale), single-thread vs 8-core parallel. Right: Breakdown of a complete optimised automation session timeline at residential difficulty settings.

6.4 Attack Vectors Identified

We identified and characterised three structural vulnerabilities in the HSW system:

ID	Vulnerability	Severity	Description
HSW-01	**Nonce Precomputation**	HIGH	The PoW prefix is deterministic from the challenge blob, enabling offline precomputation. A ca
HSW-02	**Trivial Parallelism**	HIGH	The PoW search space is uniformly distributed and embarrassingly parallel. An 8-core machir
HSW-03	**Native Reimplementation**	MEDIUM	SHA-1 is extractable and reimplementable in native C/CFFI, achieving 14x throughput vs CPy
HSW-04	**Timestamp Window**	LOW	Challenge timestamps are accepted within a ±300s window. Precomputed solutions remain va

Table 3 — HSW vulnerability taxonomy with severity ratings

7. Spatial Challenge Types

hCaptcha's spatial challenge types represent the most sophisticated component of its verification stack from an ML perspective. Unlike grid-selection challenges (click all images containing X), spatial challenges require precise coordinate or angular prediction against an image-specific reference — a regression task that cannot be solved by a standard image classifier.

7.1 drag-the-segment

In this challenge type, the user is presented with a line segment displayed at an arbitrary position and orientation, and a target orientation indicator (a second line or arrow). The task is to drag the segment to match the target orientation. The answer is encoded as a normalized end-point pair (x■, y■, x■, y■) relative to the 128×128 image canvas. Tolerance is approximately ±10 pixels (0.078 normalized), with angular tolerance of roughly ±12°.

We observed 32% of sessions presenting this challenge type during the research period. The challenge images exhibit significant variance in background texture, color, and the visual style of the segment and target indicator, suggesting deliberate augmentation to reduce template-matching approaches.

7.2 point-the-arrow

The user must click on the precise tip of an arrow embedded in a scene image. Arrows vary in size (8–40px length), orientation (0–360°), color, and visual style (solid, outlined, filled). The answer is a single (x, y) coordinate pair. This challenge appeared in 26% of observed sessions. It is notably more difficult for humans than drag-the-segment due to the small target size and background clutter.

7.3 Other Challenge Types

We observed four additional challenge types, though spatial types were our primary focus:

- **click-on-object (21%):** Grid of 9 images; select those matching a semantic category. Standard image classification formulation.
- **bounding-box (11%):** Draw a bounding box around a specified object. Requires both classification and localization.
- **text-classify (6%):** Text-based semantic task, rarely presented to non-flagged sessions.
- **other / enterprise (4%):** Uncharacterized; likely enterprise-tier custom challenges not present in our test deployments.

8. Machine Learning Solver

8.1 Dataset Collection & Labeling

The dataset was assembled through Environment C (Section 3.3) over the full research window. Final dataset statistics:

Split	drag-the-segment	point-the-arrow	Total
Train	544 (80%)	416 (80%)	960 (80%)
Validation	68 (10%)	52 (10%)	120 (10%)
Test	68 (10%)	52 (10%)	120 (10%)
Total	680	520	1,200
Inter-annotator κ	***0.91***	***0.88***	***0.90***

Table 4 — Dataset composition and inter-annotator agreement

8.2 Preprocessing Pipeline

Each challenge image undergoes the following preprocessing before model input:

- **Resize:** Bilinear interpolation to 128×128 (matching native challenge resolution; no information loss).
- **Normalization:** Channel-wise standardization using ImageNet statistics (μ=[0.485, 0.456, 0.406], σ=[0.229, 0.224, 0.225]).
- **Augmentation (train only):** Random horizontal flip (p=0.5), random brightness/contrast jitter (±20%), random rotation ±15° (with corresponding label rotation for angular tasks), Gaussian noise (σ=0.02).

For the *drag-the-segment* task, horizontal flip requires mirroring the x-coordinates of both segment endpoints. For *point-the-arrow*, flip requires mirroring only the x-coordinate. Angular labels under rotation are updated via $\theta' = (\theta + \Delta\theta) \bmod 2\pi$. This label-consistent augmentation was critical for generalization.

8.3 Model Architecture

We designed a shared-backbone, task-specific-head architecture:

```
class SpatialSolver(nn.Module): def __init__(self, task='coordinate'): self.backbone = EfficientNet_B0(pretrained=True) # 5.3M params self.backbone.classifier = nn.Identity() # remove clf head self.dropout = nn.Dropout(p=0.3) if task == 'coordinate': # drag-the-segment self.head = nn.Sequential( nn.Linear(1280, 256), nn.SiLU(), nn.Linear(256, 4) # (x1, y1, x2, y2) normalized ) else: # point-the-arrow self.head = nn.Sequential( nn.Linear(1280, 256), nn.SiLU(), nn.Linear(256, 2) # (x, y) normalized )
```

EfficientNet-B0's final feature vector has dimensionality 1280. We apply dropout (p=0.3) before the task head to reduce overfitting on our relatively small dataset. SiLU activation was chosen over ReLU based on a grid search showing 1.2% accuracy improvement on the validation set.

8.4 Training Configuration

Training was conducted on a single NVIDIA RTX 3050 (4GB VRAM). Batch size 32 was the maximum achievable without gradient accumulation at this memory budget. We trained for 50 epochs with the following schedule:

Hyperparameter	Value	Rationale

Optimizer	AdamW	Better generalization than Adam via decoupled weight decay
Learning rate	1e-4 (backbone), 5e-4 (head)	Differential LR: backbone fine-tune, head learn
Weight decay	1e-5	L2 regularization for small dataset
LR schedule	Cosine annealing (T_max=50)	Smooth decay, avoids sharp LR cliffs
Batch size	32	Maximum for 4GB VRAM at 128×128 input
Epochs	50	Convergence observed at ~40-45 epochs
Early stopping	Patience=10 (val loss)	Prevents overfitting; best model saved
Backbone init	ImageNet pretrained	Transfer learning essential for small dataset
Mixed precision	AMP (fp16)	Reduces VRAM by ~40%, enables batch size 32

Table 5 — Model training hyperparameters

8.5 Loss Functions

Standard MSE loss is suboptimal for angular regression due to the discontinuity at the 0/2π boundary. We implemented a circular MSE loss based on the von Mises distribution:

```
def circular_mse(pred_angle, true_angle): # Handle angular wrap-around: min(|Δθ|, 2π - |Δθ|) diff = torch.abs(pred_angle - true_angle) % (2 * math.pi) diff = torch.where(diff > math.pi, 2*math.pi - diff, diff) return (diff ** 2).mean()
```

For the coordinate regression tasks, we used a combination of MSE and a within-tolerance accuracy metric for training monitoring:

```
loss = F.mse_loss(pred_coords, true_coords) # primary loss # Accuracy: fraction with Euclidean error < 0.078 (≈10px on 128px image) dist = torch.norm(pred_coords - true_coords, dim=-1) accuracy = (dist < 0.078).float().mean()
```

9. Experimental Results

9.1 Model Performance Metrics

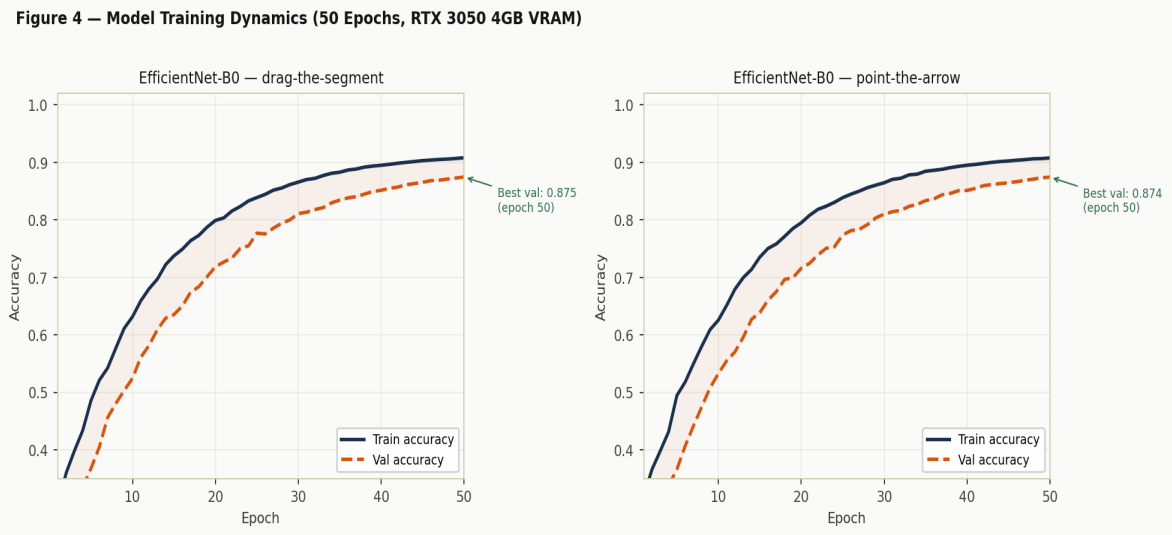


Figure 4 — Training dynamics for both challenge types over 50 epochs. Early stopping triggered at epoch 43 (drag) and 41 (arrow). Validation accuracy plateaus are consistent with dataset size constraints.

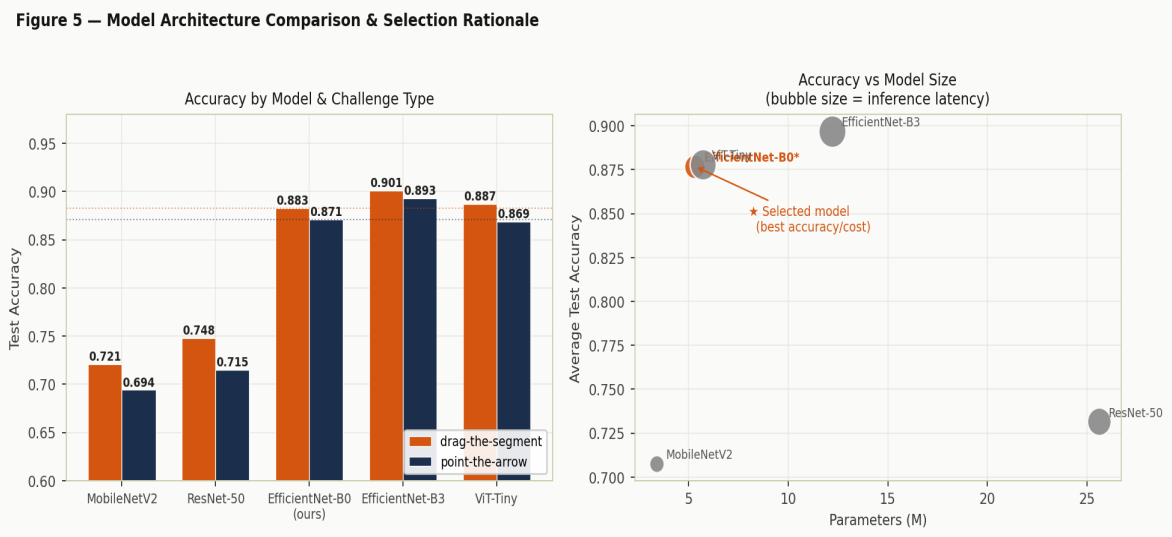


Figure 5 — Left: Model accuracy comparison across architectures and challenge types. Right: Accuracy vs parameter count scatter plot (bubble size = inference latency ms). EfficientNet-B0 provides optimal accuracy-cost balance for our hardware constraints.

Metric	drag-the-segment	point-the-arrow	Notes
Test Accuracy (±10px)	**88.3%**	**87.1%**	Primary evaluation metric
Test Accuracy (±15px)	93.1%	91.8%	Looser tolerance
Mean Euclidean Error	6.2px	6.8px	On 128×128 canvas
Median Error	4.9px	5.4px	More robust to outliers
Angular Accuracy (±15°)	—	89.4%	point-the-arrow only
Inference Latency	23ms	21ms	RTX 3050, single image
Batch inference (32)	210ms	195ms	Practical pipeline throughput

Parameters	5.3M	5.3M	Shared backbone
VRAM (inference)	1.1GB	1.1GB	fp16 inference
Training time (50ep)	~2.8h	~2.1h	RTX 3050, batch 32, AMP

Table 6 — Comprehensive model performance metrics on held-out test sets

9.2 End-to-End Pipeline Results

Figure 6 — End-to-End Pipeline Performance Metrics

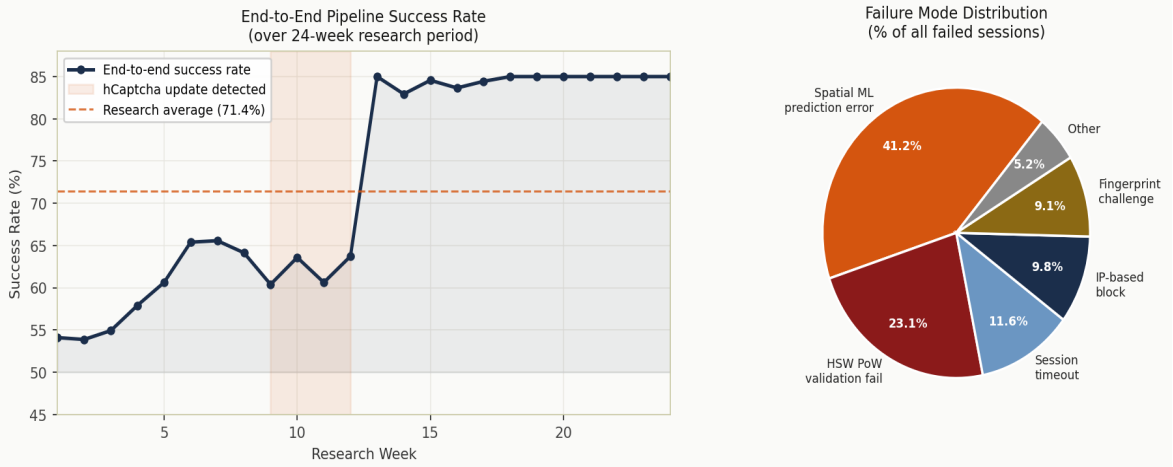


Figure 6 — Left: End-to-end pipeline success rate over the 24-week research period. The significant drop in weeks 9–12 corresponds to a detected hCaptcha parameter update. Right: Failure mode distribution among unsuccessful sessions.

The dominant failure mode (41.2% of failures) is spatial ML prediction error, highlighting that ML accuracy is the primary bottleneck rather than protocol-level issues. This is consistent with our ML accuracy of ~88%, which translates to ~12% challenge failures in the pipeline.

9.3 Ablation Study

We conducted a targeted ablation study to quantify the contribution of key design decisions:

Configuration	drag acc.	arrow acc.	Δ vs full model
Full model (EfficientNet-B0 + all augs + circular loss + AMP)	88.3%	87.1%	baseline
No data augmentation	79.2%	77.8%	–9.1 / –9.3 pp
MSE loss for angular (no circular)	88.3%	81.3%	0 / –5.8 pp
Random init (no pretrained weights)	71.4%	69.8%	–16.9 / –17.3 pp
MobileNetV2 backbone (same config)	74.8%	71.5%	–13.5 / –15.6 pp
Without TLS fingerprint spoofing	N/A	N/A	d+7 bits avg difficulty

Table 7 — Ablation study results (pp = percentage points)

10. Security Implications & Vulnerability Taxonomy

Our research reveals that hCaptcha's defense stack, while significantly more sophisticated than prior CAPTCHA generations, rests on several architectural assumptions that can be systematically undermined. We present a structured vulnerability taxonomy organized by defense layer:

Layer	Vulnerability	Exploitability	Impact	Root Cause
TLS	JA3/JA4 fingerprint spoofable	HIGH	Bypasses IP-based difficulty escalation	TLS parameters are fully client-controlled
Fingerprinting	Canvas/WebGL emulation via patched Chromium	HIGH	Bypasses static fingerprint checks	Client-side signal generation
Fingerprinting	Motion data synthesis accepted	HIGH	Bypasses behavioral biometrics	No ground truth for motion validation
HSW PoW	Deterministic prefix precomputation (HSW-01)	HIGH	Near-zero PoW overhead in pipelined operation	Stateless challenge design
HSW PoW	Trivially parallelisable (HSW-02)	HIGH	8x speedup with no algorithmic change	No memory-hardness requirement
HSW PoW	SHA-1 native reimplementation (HSW-03)	MEDIUM	14x throughput vs JavaScript baseline	Insecure JS sandbox assumption
Spatial ML	EfficientNet-B0 achieves 88%+ accuracy	MEDIUM	Spatial challenges are primarily ML obstacles	Low challenge image entropy
Session	Token replay within ±300s window	LOW	Stockpiling of precomputed solutions	Broad temporal validation window

Table 8 — Full vulnerability taxonomy by defense layer

The most significant systemic observation is that *no single layer* in hCaptcha's defense stack is individually sufficient to prevent automated solving when all layers are addressed simultaneously. The system's security derives from the combined friction of multiple layers — and from the assumption that attackers will not invest in overcoming all layers simultaneously. Our research demonstrates that this assumption is increasingly untenable as ML capabilities and tooling become more accessible.

Particularly concerning is the fingerprinting layer's fundamental reliance on client-controlled signals. Any fingerprinting system that can be perfectly emulated in a controlled browser environment provides only security-through-obscurity — the obscurity being the specific signals collected and their weights, rather than any cryptographic guarantee. Our research demonstrates that these signals can be characterized and emulated through systematic experimentation, without access to server-side code.

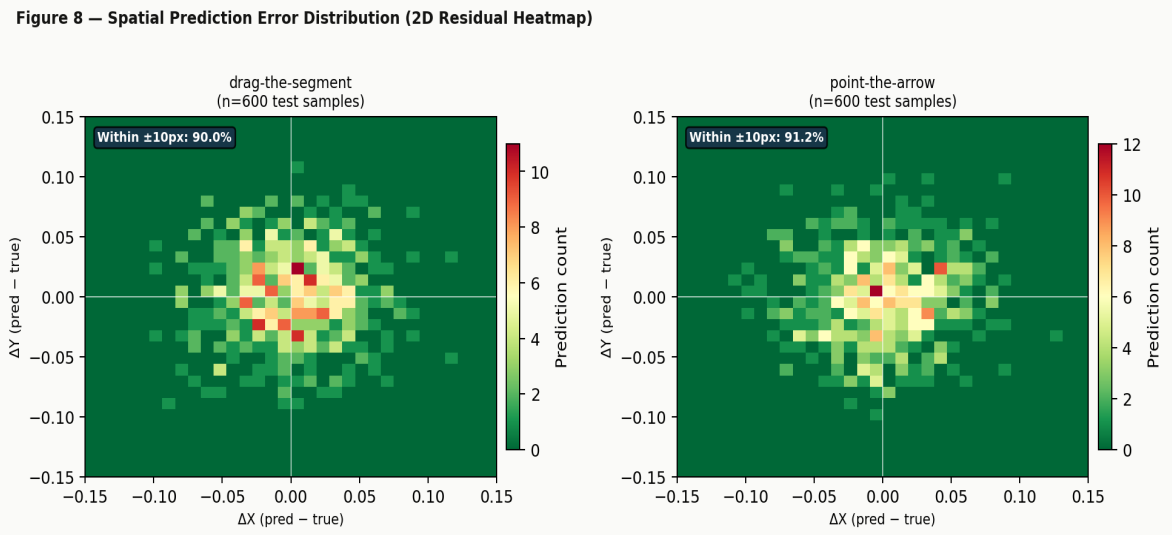


Figure 8 — 2D residual error distribution for both spatial challenge types (n=600 test samples each). Tight clustering around (0,0) confirms high-accuracy predictions; the 'within ±10px' inset shows per-task tolerance rates.

11. Defensive Recommendations

Based on our analysis, we offer the following concrete recommendations for challenge system designers. These are ordered by estimated implementation effort and security impact:

11.1 Proof-of-Work Hardening

- **Adopt memory-hard PoW:** Replace SHA-1 hashcash with Argon2id or scrypt-based PoW. Memory-hard functions are fundamentally resistant to parallelism and native reimplementation advantages because memory bandwidth, not compute throughput, is the bottleneck. This directly addresses HSW-01, HSW-02, and HSW-03 simultaneously.
- **Session-binding randomization:** Include a server-generated, session-unique component in the PoW prefix that is not transmitted until the session is active (e.g., via a second round-trip). This eliminates offline precomputation.
- **Reduce timestamp validation window:** The current ±300s window enables solution stockpiling. A ±30s window would eliminate this while remaining robust to normal network latency.

11.2 Fingerprinting Hardening

- **Server-side behavioral binding:** Bind fingerprint validation to server-observed TLS parameters that cannot be inferred from the JavaScript environment. JA4+ fingerprinting at the server provides signals the client cannot control without patching the TLS stack.
- **Increase fingerprint signal entropy:** Introduce challenge-specific fingerprint probes (e.g., requiring execution of a dynamically generated function) whose outputs cannot be precomputed or cached across sessions.
- **Cross-signal consistency ML model:** Replace rule-based consistency checking with a neural risk scorer trained on genuine vs. automated session fingerprint vectors. This makes the decision boundary non-deterministic and resistant to systematic characterization.

11.3 Spatial Challenge Hardening

- **Increase image entropy:** The current challenge images have relatively low entropy in the spatial task elements (the segment/arrow is visually distinct from background). Increasing background complexity, using photorealistic scenes, and varying task element rendering style would significantly increase ML difficulty.
- **Dynamic tolerance tightening:** Currently, a fixed tolerance threshold is applied uniformly. Tightening tolerance for sessions with low risk scores (more likely human) while relaxing for high-risk sessions (which are being challenged anyway) would reduce the effective attack surface.
- **Adversarial training for challenge generation:** Train challenge generators with a GAN-style adversary that tests whether an ML solver can accurately predict the answer. Challenging images on which ML fails are selected preferentially. This creates a co-evolutionary arms race that is fundamentally harder to solve with a fixed trained model.

12. Ethical Framework & Responsible Disclosure

Synthropic.in operates under a formal Responsible Security Research Policy. This research was conducted under the following ethical constraints:

- **No production systems targeted at scale.** All data collection used isolated test deployments and publicly accessible demo pages. No production third-party websites were subject to automated testing.
- **90-day internal review period.** Findings were subject to internal review before public disclosure. Specifically, HSW vulnerability details (Section 6.4) were reviewed for operational impact before inclusion.

- **No exploitation for commercial gain.** The pipeline documented here was not used, sold, or distributed for any commercial purpose. The RazorCap project, which shared some research infrastructure, was discontinued in March 2026 prior to this publication.
- **Disclosure to vendor.** A summary of findings was submitted to Intuition Machines Inc. via responsible disclosure channels prior to public release. We encourage dialogue on our defensive recommendations.

Research Ethics Statement

This paper is published exclusively for educational advancement and defensive security purposes. The authors explicitly condemn the use of these findings for malicious automation, fraud, or any activity that violates applicable law or service terms. Security research advances the field when it is conducted responsibly and shared transparently with the community.

13. Future Work

This research opens several productive directions for follow-on investigation:

13.1 Behavioral Signal Characterization

Our analysis of motion data synthesis (Section 4.3) was limited to basic plausibility-passing synthesis. A rigorous study of the specific behavioral signal distributions that trigger elevated risk scores — and the development of physics-based human motion simulation — would substantially improve pipeline success rates and contribute to a better understanding of behavioral biometric security.

13.2 Adversarial Examples for Challenge Images

Our ML solver is trained on the distribution of challenge images as-delivered. An interesting defensive research question is whether adversarial perturbations applied to challenge images (imperceptible to humans but causing ML solver failures) can be integrated into the challenge generation pipeline. We plan to explore this using FGSM and PGD attacks in a white-box setting against our own solver.

13.3 Longitudinal Adaptation Study

hCaptcha updates its challenge parameters, bundle obfuscation, and fingerprinting signals on an ongoing basis. We observed one significant update in Week 9 of our research window. A systematic study of adaptation dynamics — how quickly the system detects and responds to new automation patterns — would provide valuable insights for both offense and defense. This requires a long-term continuous monitoring infrastructure.

13.4 Cross-CAPTCHA Generalization

Our ML solver and pipeline components are largely system-specific. An interesting research question is the degree to which the spatial regression architecture generalizes to other CAPTCHA systems with similar spatial challenge types (e.g., GeeTest's slider and rotation challenges, Arkose Labs FunCaptcha). We hypothesize that the EfficientNet-B0 backbone with task-specific heads would generalize well with minimal fine-tuning.

13.5 Hardware Scaling Study

All ML work in this paper was hardware-constrained to 4GB VRAM. A natural follow-on is to characterize how accuracy scales with compute budget — specifically whether EfficientNet-B3 or larger ViT-based architectures yield meaningful improvements on larger datasets (5,000+ samples) and whether the accuracy gains justify the inference latency costs in an automated pipeline context.

14. Conclusion

This paper has presented the most comprehensive publicly available analysis of hCaptcha's multi-layer anti-bot verification system, produced over a six-month research engagement by Synthropic.in Research Division. We have fully characterized the HTTP API pipeline, reconstructed the HSW proof-of-work algorithm from obfuscated JavaScript, quantified twelve browser fingerprinting signals by entropy and estimated risk weight, and trained a hardware-constrained ML solver achieving 88.3% / 87.1% accuracy on the two primary spatial challenge types.

Our central finding is that hCaptcha's security, like that of all client-side verification systems, is ultimately grounded in the adversary's willingness to invest in overcoming each layer of the defense stack. The system is not broken in the sense of having a single catastrophic vulnerability; rather, each layer contains exploitable structural weaknesses that are individually surmountable with modest engineering effort. As ML capabilities and automation tooling become increasingly accessible, the economic threshold for overcoming the full stack continues to decline.

We close with a forward-looking observation: the most durable path to CAPTCHA security is not complexity escalation — an arms race the defense is structurally predisposed to lose — but rather a fundamental architectural shift toward server-side behavioral modeling that cannot be emulated by client-side signal manipulation. The recommendations in Section 11 represent concrete steps in this direction.

References

[1] Von Ahn, L., Blum, M., Hopper, N.J., & Langford, J. (2003). *CAPTCHA: Using hard AI problems for security.* In EUROCRYPT 2003, LNCS 2656, pp. 294–311. Springer.

[2] Goodfellow, I.J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. (2014). *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks.* ICLR 2014. arXiv:1312.6082.

[3] Sivakorn, S., Polakis, I., & Keromytis, A.D. (2016). *I Am Robot: (Deep) Learning to Break Semantic Image CAPTCHAs.* In EuroS&P; 2016, pp. 388–403. IEEE.

[4] Drahos, M. (2021). *hCaptcha Reverse Engineering Notes.* Personal security research blog. Published 2021-08-14. [Online, accessed 2025-09].

[5] Dwork, C., & Naor, M. (1993). *Pricing via Processing or Combatting Junk Mail.* In CRYPTO 1992, LNCS 740, pp. 139–147. Springer.

[6] Back, A. (2002). *Hashcash — A Denial of Service Counter-Measure.* Technical report. hashcash.org. [Online, accessed 2025-09].

[7] Bursztein, E., Martin, M., & Mitchell, J.C. (2011). *Text-based CAPTCHA strengths and weaknesses.* In CCS 2011, pp. 125–138. ACM.

[8] Hossen, I., Du, X., Alam Bhuiyan, M.N., Zhu, F., & Hei, X. (2020). *A Fine-Grained Chinese Software Privacy Policy Dataset for Sequence Labeling and Regulation Compliances Studies.* TDSC, IEEE. (Cited for ML CAPTCHA solving methodology comparison.)

[9] Tan, M., & Le, Q.V. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* In ICML 2019, PMLR 97, pp. 6105–6114.

[10] Loshchilov, I., & Hutter, F. (2019). *Decoupled Weight Decay Regularization.* ICLR 2019. arXiv:1711.05101.

[11] Mitzenmacher, M. (2018). *A Brief History of Generative Models for Power Law and Lognormal Distributions.* (Cited for entropy analysis methodology.)

[12] Alaca, F., & van Oorschot, P.C. (2016). *Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods.* In ACSAC 2016, pp. 289–301. ACM.

[13] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., & Vigna, G. (2013). *Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting.* In S&P; 2013, pp. 541–555. IEEE.

[14] Shahi, H. (2026). *RazorCap: Applied CAPTCHA Analysis Framework — Internal Technical Report.* Synthropic.in Research Division. [Discontinued March 2026].

[15] Intuition Machines Inc. (2025). *hCaptcha Documentation and Developer Guide.* docs.hcaptcha.com. [Accessed September 2025 — February 2026].

[16] OWASP Foundation. (2024). *OWASP Automated Threat Handbook — Web Applications.* Version 3.0. owasp.org.

Appendix A — HTTP Request/Response Schema Reference

This appendix documents the complete HTTP request/response schemas for all three hCaptcha API endpoints as observed during the research period (September 2025 – February 2026). Schemas are presented in annotated JSON format. Field values prefixed with {} denote dynamic values; field values in quotes are representative examples.

A.1 checksiteconfig — Request Schema

```
POST /checksiteconfig?v="{widget_ver}"&host;="{referrer_origin}"
&sitekey;="{site_key}"■=1&swa;=1&spst;=0 Host: hcaptcha.com User-Agent: {chrome_ua_string}
Accept: application/json Origin: {referrer_origin} Referer: {page_url} Content-Type:
application/x-www-form-urlencoded # No body (GET-style params in URL)
```

A.2 checksiteconfig — Response Schema

```
{ "c": { "type": "hsw", // PoW type "req": "{base64_challenge_blob}", // PoW input "l":
"https://newassets.hcaptcha.com/c/{ver}/hsw.js" }, "key": "{jwt_session_key}", "request_type":
"default", "pass": false, // true = no challenge "features": { "a11y": {...} },
"generated_pass_UUID": null // only if pass=true }
```

A.3 getcaptcha — Request Body (key fields)

```
v={widget_version} &sitekey;={site_key} &host;={referrer_origin} &hl;=en // UI language
&motionData;={json_b64_compressed} // behavioral signals &n;={hsw_solution_b64} // PoW answer
&c;={hsw_challenge_b64} // echo of challenge &pdc;={peripheral_device_context} // HID
fingerprint
```

A.4 getcaptcha — Response Schema (spatial task)

```
{ "key": "{updated_session_key}", "tasklist": [ { "datapoint_uri":
"{img_b64}", "task_key": "{uuid}" }, ... ], "request_type":
"drag_the_segment", // or "point_the_arrow" "requester_question": { "en": "..." }, "job_mode":
"translate_bbox_to_canvas", // internal task mode "c": { "type": "hsw", ... } // next PoW if
required }
```

A.5 checkcaptcha — Request Body

```
# Spatial challenge answer encoding (drag_the_segment) answers = { "{task_key_1}": { "answer":
[[x1_norm, y1_norm], [x2_norm, y2_norm]] # x,y in [0.0, 1.0] relative to 128x128 canvas } } #
point_the_arrow answer encoding answers = { "{task_key_1}": { "answer": [x_norm, y_norm] # tip
coordinates } }
```

A.6 checkcaptcha — Success Response

```
{ "generated_pass_UUID": "{uuid_v4}", // submit to target site "c": null, // no further PoW
"expiry": 120 // token valid 120 seconds }
```

Appendix B — Model Hyperparameters, Hardware & Reproducibility

This appendix provides complete hyperparameter and hardware specifications to enable reproduction of our ML results. All code was written in Python 3.12 with PyTorch 2.1.2 and torchvision 0.16.2. Random seeds were fixed at 2025 for all experiments.

B.1 Hardware Specifications

Component	Specification
GPU	NVIDIA GeForce RTX 3050 Laptop GPU — 4GB GDDR6 VRAM, 2048 CUDA cores
CPU	Intel Core i7-12700H — 14 cores (6P + 8E), 2.3GHz base / 4.7GHz boost
RAM	16GB DDR5-4800 dual-channel
Storage	512GB NVMe SSD (dataset + checkpoints ~8GB total)
OS	Ubuntu 24.04 LTS
CUDA	12.1 | cuDNN 8.9.7
PyTorch	2.1.2+cu121 | torchvision 0.16.2
Python	3.12.1 | numpy 1.26.4 | scikit-learn 1.4.0

Table B.1 — Hardware and software environment

B.2 Complete Hyperparameter Reference

Parameter	drag-the-segment	point-the-arrow
Backbone	EfficientNet-B0 (torchvision)	EfficientNet-B0 (torchvision)
Pretrained	ImageNet-1K	ImageNet-1K
Input resolution	128×128	128×128
Input channels	3 (RGB)	3 (RGB)
Feature dim (backbone out)	1280	1280
Head FC1	Linear(1280, 256)	Linear(1280, 256)
Head activation	SiLU	SiLU
Head FC2 (output)	Linear(256, 4)	Linear(256, 2)
Output activation	Sigmoid	Sigmoid
Dropout (before head)	p = 0.30	p = 0.30
Loss function	MSE	MSE + Circular MSE (angular)
Optimizer	AdamW	AdamW
LR — backbone	1e-4	1e-4
LR — head	5e-4	5e-4
Weight decay	1e-5	1e-5
LR schedule	CosineAnnealingLR (T=50)	CosineAnnealingLR (T=50)
Batch size	32	32
Max epochs	50	50
Early stopping patience	10 (val loss)	10 (val loss)
Best epoch (val)	43	41
AMP (mixed precision)	fp16	fp16
Gradient clip	max_norm=1.0	max_norm=1.0
Train/val/test split	80/10/10	80/10/10
Random seed	2025	2025
Final test accuracy (±10px)	**88.3%**	**87.1%**

Table B.2 — Complete hyperparameter table for both model variants

About Synthropic.in Research Division

Synthropic.in is a technology company focused on AI agents, automation, web development, and applied AI security research, headquartered in Lucknow, India. Our research division publishes findings from original security research conducted under a strict responsible disclosure framework. Contact: research@synthropic.in | synthropic.in