USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Are CAPTCHAs Still Bot-hard? Generalized Visual CAPTCHA Solving with Agentic Vision Language Model

Xiwen Teoh, *Shanghai Jiao Tong University; National University of Singapore;* Yun Lin, *Shanghai Jiao Tong University;* Siqi Li and Ruofan Liu, *National University of Singapore;* Avi Sollomoni and Yaniv Harel, *Tel Aviv University;* Jin Song Dong, *National University of Singapore*

https://www.usenix.org/conference/usenixsecurity25/presentation/teoh

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Are CAPTCHAs Still Bot-hard? Generalized Visual CAPTCHA Solving with Agentic Vision Language Model

Xiwen Teoh[1,2], *Yun Lin*[1]*, *Siqi Li*[2], *Ruofan Liu*[2], *Avi Sollomoni*[3]
Yaniv Harel[3], *Jin Song Dong*[2]
Shanghai Jiao Tong University[1], *National University of Singapore*[2], *Tel Aviv University*[3]

Abstract

Visual CAPTCHAs, such as reCAPTCHA v2, hCaptcha, and GeeTest, are mainstream security mechanisms to deter bots online, based on the *assumption* that their visual challenges are bot-hard but human-friendly. While many deep-learning based solvers have been designed and trained to solve a specific type of visual challenge in a CAPTCHA, vendors can easily switch to out-of-distribution visual challenge of the same type or even new types of challenge with very low cost. However, the emergence of general-purpose AI models (e.g., ChatGPT) challenges the bot-hard assumption of existing visual challenges, potentially compromising the reliability of visual CAPTCHAs.

In this work, we report the first *generalized* visual CAPTCHA solver, Halligan, built upon the state-of-the-art vision language model (VLM), which can effectively solve *unseen* visual challenges in CAPTCHAs without making any adaptation. Our rationale lies in that a visual challenge can be reduced to a search problem where (i) its instruction is transformed into an optimization objective and (ii) its body is transformed into a search space for the objective. With well designed prompts built upon known VLMs, the transformation can be generalized to unseen visual challenges. Our extensive experiments show that Halligan is a game-changer to the known practice of adopting visual CAPTCHAs, which achieves a solving rate of 60.7% on 2,600 challenges belonging to 26 types of visual CAPTCHAs. Further, we use Halligan to infiltrate human-driven CAPTCHA farms, achieving an average solving rate of 70.6% on previously unseen visual challenges from CAPTCHAs in the wild over a 30-day period. Based on the experimental results, we further shed light on puzzle-less anti-bot alternatives in this era.

1 Introduction

Visual CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are challenge-response tests that verify if a user is human or bot [66], which establishes proof-of-personhood by presenting users with interactive puzzles, where solving them is a key step toward passing the CAPTCHA. It has been a conventional web-abuse prevention mechanism for over 20 years [39]. Among the top 1 million websites in 2024, over 256K websites are estimated to use CAPTCHAs, where 94% of the CAPTCHAs are visual CAPTCHAs (see Figure 1) [5]. The effectiveness of CAPTCHAs is based on the assumption that *the challenges are bot-hard but human-friendly*.

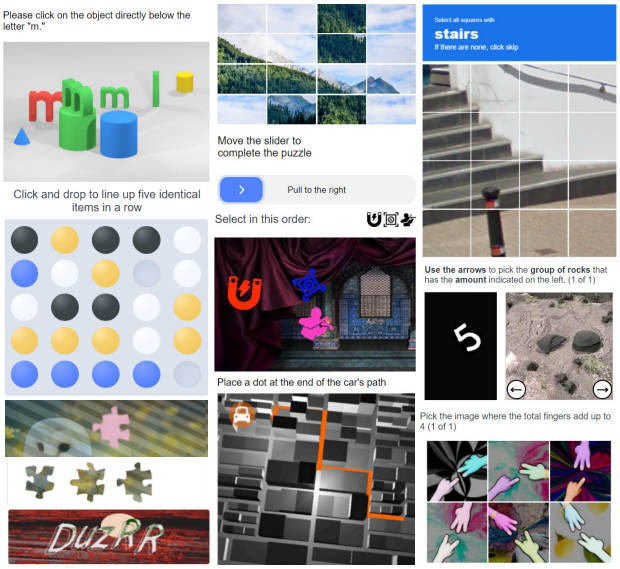


Figure 1: Diverse visual CAPTCHA challenges adopted in practice, expected to be bot-hard but human-friendly.

The security ecosystem of the visual CAPTCHAs has long been a cat-and-mouse game between attackers and defenders [31, 39, 52, 58, 70]. Attackers evolve their solvers [24, 25, 27, 35, 51, 53, 72, 73] from text-based CAPTCHAs (e.g., distorted letters or numbers [66]) to Google's reCAPTCHAv2 (e.g., identifying objects in a 3×3 or 4×4 grid of images) [13] with vision reasoning [36, 37], image classifiers [43, 60, 61],

*Corresponding author

object detectors [44, 55], visual question answering [64], and side-channel attacks [54]. In response, CAPTCHA vendors swiftly evolve their CAPTCHAs with new visual challenges such as AWS WAF [3], Arkose Labs' FunCAPTCHA and MatchKey [1], DataDome [6], GeeTest [7], and Lemin [10]. In such a campaign where the attackers design solvers for a *specific* type of visual challenge, the defenders can quickly adopt an unseen visual challenge, almost nullifying all the efforts of the attackers.

However, with the advancement of AIGC (AI-Generated Content) technologies, such bot-hard assumption of CAPTCHA is challenged. On one hand, vision language models (VLM) can potentially understand almost all known types of visual challenges. On the other hand, the emergence of agentic AI (i.e., systems that can autonomously plan and execute tasks using external tools) [59] further enables the potential development of *generalized* CAPTCHA solving solution that can interact with real-world websites.

Moreover, unlike password authentication, which restricts the number of attempts, CAPTCHA services allow many tries to accomodate user error, creating an opening for abuse. Our survey of CAPTCHA policies [5] shows that 11 services use low thresholds (pass once), 2 use medium thresholds (2–3 passes out of N, depending on factors like VPN use), together covering over 260,000 of the top 1 million websites. 2 services use high thresholds (requiring all N passes), covering 8 sites. In the best-case scenario, with a solving rate of p (e.g., 50%), the success probability after k attempts is $1-(1-p)^k$ (e.g., 96.8% for $k=5$ and $p=50\%$). While stricter policies improve security, they risk degrading website usability. Thus, we argue that the advent of AIGC marks a paradigm shift in which CAPTCHA attackers may, for the first time, outpace defenders by repeatedly using VLMs to solve *unseen* visual challenges in CAPTCHAs.

This work introduces Halligan[1], which is (1) a VLM agent designed as a *general-purpose* visual CAPTCHA solver that remains effective even on previously unseen visual challenges and (2) concrete evidence that visual CAPTCHA challenges are increasingly vulnerable in the AIGC era. We show that a visual challenge can be reduced to a search problem where (i) its instruction (in natural language) can be transformed into an optimization objective and (ii) its body or interface (comprising visible entities) can be transformed into a search space for the objective.

Technically, we design Halligan as a CAPTCHA-to-action system which parses a visual challenge into a sequence of actions (e.g., drag, slide, click, etc.) on CAPTCHA entities (e.g., frames, elements, keypoints, etc.). It uses two high-level ideas (implemented through three steps in Section 3):

- **CAPTCHA Abstraction:** The agent must first abstract the visual challenge into a model comprising entities such as frames, keypoints, and elements. Interactable entities are then visually identified, enabling the VLM to infer precise locations for applying actions.
- **CAPTCHA Solving:** The agent must formulate a search problem based on the visual challenge, generating an optimization objective and constructing a search space through interaction. It can use tools to help evaluate candidate solutions (i.e., CAPTCHA states) against the objective. Starting from an unsolved state, the agent explores the space by applying actions to CAPTCHA entities (e.g., drag frame A, slide element B to C, click keypoint D) until an optimal solution is found, which is considered the solved state.

Halligan achieves a solving rate of 60.7% on 2,600 challenges from 26 types of visual CAPTCHAs. By skillset, it performs well on vision tasks like object detection and visual reasoning (71%) with room to improve in 3D spatial reasoning (17%). By search space, it handles discrete actions like click, select, swap effectively (68%), with potential gains in continuous actions like drag and slide (29%). As a general-purpose CAPTCHA solver, Halligan outperforms both specialized solvers in their target domains [64, 76] and generalist web navigation agents [40, 46]. Further, we use Halligan to infiltrate human-driven CAPTCHA farms, achieving an average solving rate of 70.6% on previously unseen visual challenges from CAPTCHAs in the wild over a 30-day period. Our findings highlight a concerning trend: attacking visual CAPTCHA challenges has become feasible and affordable, which necessitates the call for puzzle-less CAPTCHA alternatives in this AIGC era (See our discussion in Section 5).

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first work to present a *generalized* visual CAPTCHA solver, Halligan, which can effectively solve unseen visual challenges automatically, highlighting vulnerabilities in the visual CAPTCHA ecosystem in practice.
- Technically, we reduce the visual CAPTCHA-solving problem into an optimization problem. By orchestrating a set of VLM agents, Halligan can abstract the visual challenge of any CAPTCHA into a general model and solve the formulated search problem effectively.
- We present the most scalable and diverse interactive CAPTCHA benchmark to date, featuring 26 visual CAPTCHA types and 2,600 unique challenges, to support future research in the community.
- We extensively evaluate Halligan with both close-world and open-world experiments. In the closed-world setting, we evaluate Halligan on 2,600 prepared CAPTCHAs. In the open-world setting, we evaluate Halligan on 3,000 *unknown* CAPTCHAs in the wild by infiltrating human-driven CAPTCHA farms. Both shows that Halligan is a new state-of-the-art in solving visual CAPTCHA challenges.

[1] The name is from a multipurpose tool to breach different locked doors.

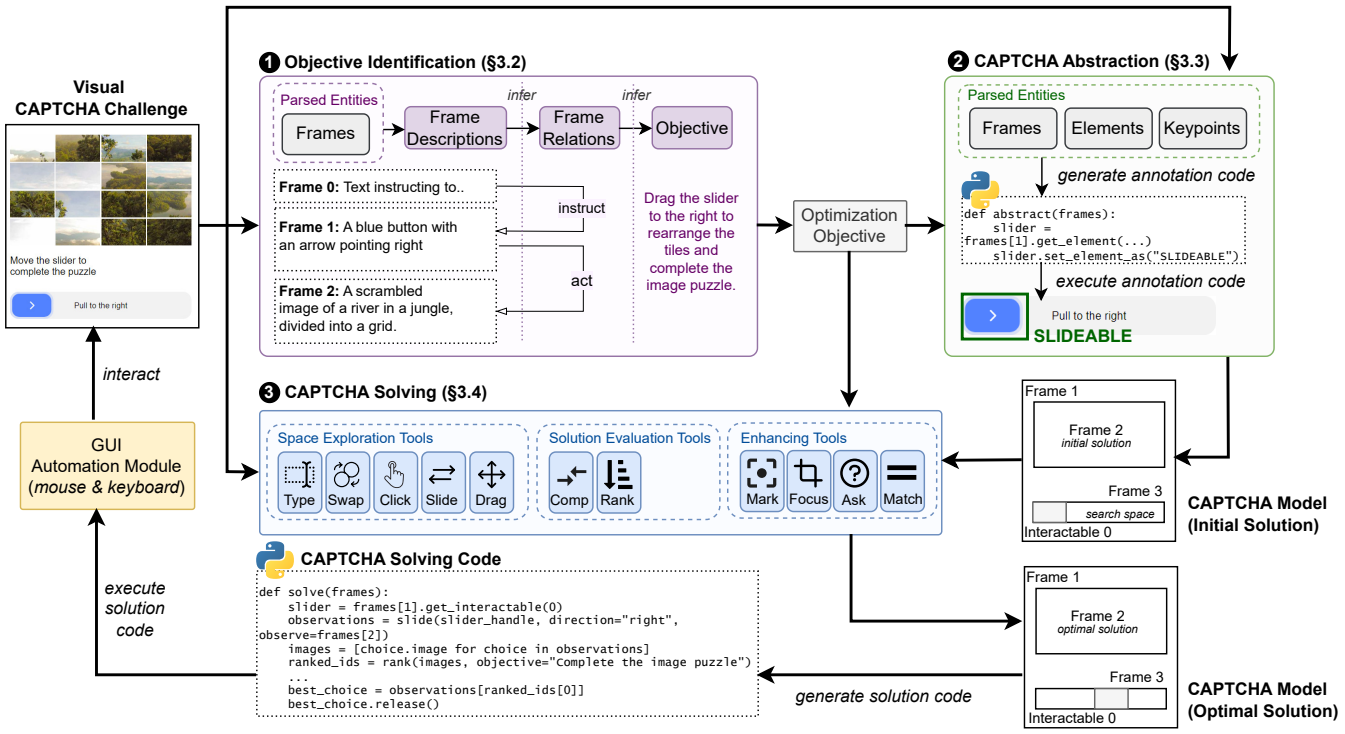


Figure 2: Overview of Halligan, consisting of three steps. ❶ **Objective Identification** step infers the task objective. The ❷ **CAPTCHA Abstraction** step generates an abstract CAPTCHA model from the visual CAPTCHA challenge, where CAPTCHA entities such as frame and interactable UI elements are present. The ❸ **CAPTCHA Solving** step formalizes a search problem to generate a solution, translated into executable Python code to interact with and solve the visual CAPTCHA challenge.

Ethical Research. We acknowledge the risks of a generalized visual CAPTCHA solver to the Internet. To mitigate this, we followed strict ethical guidelines (See section on Ethical Consideration). Our research goal is to highlight the potential risks of VLMs and the vulnerabilities of visual CAPTCHA challenges, rather than create or optimize harmful attacks.

2 Threat Model

Attacker (CAPTCHA Solver). The attacker aims to solve the visual challenge presented in a CAPTCHA. We assume that the attacker has access to local or online visual language models (VLM). In addition, they are free to build agentic programs to orchestrate different VLM agents. There is no hard requirement on computational resources, a consumer-grade laptop that can load models in memory is sufficient to carry out attacks. Besides, the attacker cannot decide how the CAPTCHA is displayed, i.e., web browser, desktop, mobile. The attacker can only operate on vision-level with absolute rendering-device-specific viewport coordinates, but no access to the source code of the CAPTCHA. Finally, we assume that the CAPTCHAs are not vulnerable to side-channel attacks [54] where extra information can be exploited due to flaws in the design or implementation of the CAPTCHA.

Defender (CAPTCHA Service). The defender can evolve novel visual CAPTCHA challenges and employ both visual and prompt-based adversarial techniques (e.g., Gaussian noise and prompt injection).

3 Approach

Figure 2 shows an overview of Halligan, which solves a visual CAPTCHA challenge by orchestrating the interaction between VLM-based agents and a number of external tools to parse and manipulate the CAPTCHA, it has three steps:

Step 1: Objective Identification (Section 3.2). We first design a VLM agent to parse an *optimization objective* in the form of natural language description from the CAPTCHA. To this end, important CAPTCHA entities (and their textual description) as well as their relations are processed to guide the agent to output a relevant objective.

Step 2: CAPTCHA Abstraction (Section 3.3). Next, we abstract the visual CAPTCHA challenge into a *CAPTCHA model* where only relevant information is preserved, such as CAPTCHA layout and interactable GUI elements. A CAPTCHA model is a CAPTCHA solution, laying foundation for constructing the search space.

Step 3: CAPTCHA Solving (Section 3.4). Then, we use two

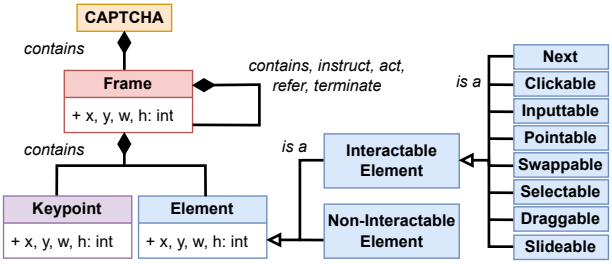


Figure 3: Visual CAPTCHA metamodel.

(a) Frames (b) Elements (c) Keypoints

Figure 4: Examples of CAPTCHA entities such as frame, element, and keypoint.

types of tools to search for the optimal solution for the objective, i.e., (1) *Space Exploration Tools* to apply actions on CAPTCHA models to derive new models (a.k.a., new solution) and (2) *Solution Evaluation Tools* to evaluate whether a new CAPTCHA solution can fit the objective well. The search process converges when Halligan outputs the optimal solution within budget. Finally, we translate the optimal solution into a piece of Python code to apply a sequence of actions to solve the visual CAPTCHA challenge. Given the space limit, all the prompt designs of VLM agents are available at [12].

Next, we first introduce the CAPTCHA model used in this work, followed by the details of each aforementioned step.

3.1 CAPTCHA Meta-model

Figure 3 shows the meta-model of visual CAPTCHA challenges discussed in this work, describing the CAPTCHA entities and their relations. Any instance of the model is a concrete CAPTCHA model used in the following steps. Overall, a visual CAPTCHA challenge consist of a set of *frames* which is a container for other CAPTCHA entities. A frame can contain nested frames, keypoints, and elements. Keypoints are clickable points and elements are visible objects in a frame. Figure 4 shows the examples of CAPTCHA entities.

In addition, a frame can *instruct*, *act*, *refer*, and *terminate* another frame, where Table 1 shows the semantic relations between frames. Figure 5 shows an example where a visual CAPTCHA challenge consists of three frames, with *contain*, *instruct*, and *act* relations. Specifically, frame_1 *act* frame_2 because the interaction with frame_1 can change how frame_2 renders; and frame_0 *instruct* frame_1 because frame_0 provide textual guidance on how to manipulate frame_1. Finally, an element can be interactable or non-interactable. Interactable elements can be *draggable*, *slidable*, *next*, *clickable*, *inputable*, *pointable*, *swappable*, and *selectable*. Table 2 shows the details of each interactable CAPTCHA element.

Table 1: Semantic Relations between Frames

Relation	Description
instruct(A, B)	Frame A provides instructions on how to interact with Frame B.
act(A,B)	Interacting with Frame A produces visible changes in Frame B (e.g., slider, prev/next choice button).
refer(A,B)	Frame A provides additional context for Frame B (e.g., match the pattern or quantity).
terminate(A,B)	Once Frame A is in a state that aligns with the instruction, interact with Frame B to terminate (i.e., submit) the challenge.

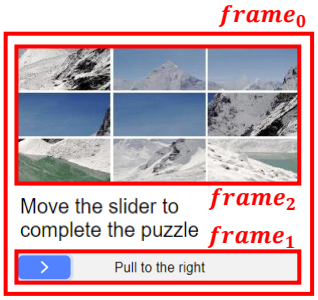


Figure 5: Examples of the semantic relation between frames, where frame_1 *act* frame_2, frame_0 *instruct* frame_1, and frame_0 contains both frame_1 and frame_2.

3.2 Objective Identification

Figure 15 shows our VLM prompt to derive the CAPTCHA objective from a set of parsed frames in a visual CAPTCHA challenge (see Section 3.3 for CAPTCHA abstraction). The objective is in natural language form, serving as a strengthened intention against potential VLM hallucinations that might deviate from the intended task. The input is a set of frames $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ such that (1) the set of all frames form a partition of the original visual challenge space ($\cup\mathcal{F} = C$) and (2) a frame may either be fully contained within another frame or disjoint from it, but they can never partially overlap ($F_i \cap F_j \neq \emptyset \implies (F_i \subseteq F_j \text{ or } F_j \subseteq F_i)$ for $i \neq j$). The outputs are description of each frame, the inferred relation between the frames, and the objective description. We let VLM generate the outputs sequentially, serving as a chain-of-thought to improve the quality of generated objective.

3.3 CAPTCHA Abstraction

Given an arbitrary visual CAPTCHA challenge c, the abstraction is to parse c into an instance of CAPTCHA model according to the meta-model (see Section 3.1). We parse the frames, elements, and keypoints as follows.

Table 2: Classification of Interactable Element

Category	Interactable	Description
UI	NEXT	Can be clicked to submit or skip the task
Discrete	CLICKABLE	Can be clicked to transition a frame
	INPUTTABLE	Can be used to enter text data
	POINTABLE	Can point to location(s) in the region
	SWAPPABLE	Can exchange places with another SWAPPABLE
	SELECTABLE	Can be toggled as an answer choice
Continuous	DRAGGABLE	Can be moved in two dimensions
	SLIDEABLE	Can be moved in one dimension

Frames. A frame extractor $f_{\text{frame}}(I_c) = \{F_1, F_2, \cdots, F_n\}$ takes the screenshot of c, I_c, as input and produces set of frames F_i. We approach this as a connected components problem by pre-processing I_c through median blur, adaptive binary thresholding, and morphological transformations to remove background noise and enhance large foreground structures. Figure 4(a) shows an example. We then identify all disjoint groups of 4-connected pixels with the same value in the binarized image of I_c, order the groups by size. Each group is transformed to a frame.

Elements. An element extractor $f_{\text{element}}(F_i) = \{E_1, E_2, \cdots, E_n\}$ takes any frame F_i as input and produces a set of elements. We approach this as an image segmentation problem and selected the Fast Segment Anything Model (FastSAM) [77], which uses a Convolutional Neural Network (CNN)-based YOLOv8-seg detector and the YOLACT [22] method for instance segmentation. Furthermore, this model is adopted in the industry for parsing GUI elements in Robotic Process Automation (RPA) [11].

To identify CAPTCHA element types (interactable and non-interactable; see Section 3.1), we perform element annotation as shown in Figure 6. Given a frame f, the agent generates natural language descriptions of all elements of interest (e.g., "right arrow button") We then use the Contrastive Language-Image Pre-Training (CLIP) model [57] to encode and retrieve the element in f most relevant to each description. Finally, the agent annotates each element's type using its description and retrieved image. The final output is an executable Python block using `get_element(description: str)` to retrieve elements and `set_element_as(type: str)` to annotate them. The full CAPTCHA abstraction prompt available in [12].

Keypoints. A keypoint extractor $f_{\text{keypoint}}(F_i) = \{K_1, K_2, \cdots, K_n\}$ takes any frame F_i as input and produces a set of keypoints. We treat this as a superpixel segmentation problem and apply Simple Linear Iterative Clustering (SLIC) [17] algorithm to get clusters based on color and proximity similarity. Next, we compute a saliency map and retain clusters with an average saliency above the 50th percentile threshold. The cluster centroids become keypoints.

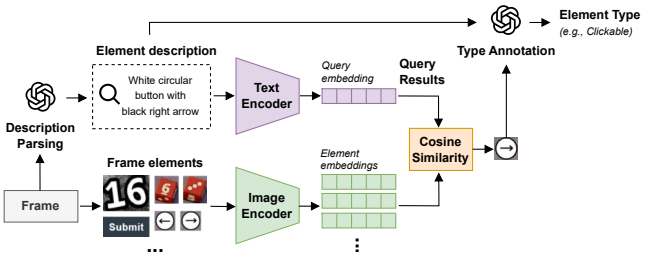


Figure 6: Annotating a parsed element with its type (e.g., clickable, slidable, or draggable).

3.4 CAPTCHA Solving

Given the objective o (see Section 3.2), the initial CAPTCHA solution s (see Section 3.3) and the visual CAPTCHA challenge c, we formulate the visual CAPTCHA-solving problem as a search problem by (1) exploring the search space and (2) evaluating the best solution c^* in the space regarding o. For both the space exploration and the solution evaluation, we prepare a set of tools (see Table 3) to derive new solutions (i.e., a CAPTCHA model) or compare the solutions. Next, we use CAPTCHA model and CAPTCHA solution interchangeably.

Search Space Exploration Given a CAPTCHA solution c, we define a set of *Space Exploration* tools (first section in Table 3) to derive new solutions c' such as *typing* an element, *clicking* an element, *swapping* some tiles in a frame, *slide* an element, and *drag* an element on the CAPTCHA solution c. In this work, we construct a compatibility mapping between the type of interactable CAPTCHA element and the types of tools. For example, if only *swappable* elements are detected, only tools relevant to swapping are exposed and made available to reduce tool-misuse. Given the solution c, our VLM agent select any of the compatible tools to generate a set of solutions neighboring c, denoted as C. For example, in a visual CAPTCHA challenge that involves dragging a slider horizontally, we can create a solution space by invoking `slide_x()` to obtain visual CAPTCHA states at discretized intervals along the sliding track.

Search Solution Evaluation Then, for each new solution $c' \in C$, we call our prepared *Solution Evaluation* tools (second section in Table 3) to compare and select the local optima c^*. To achieve this, we design VLM agents to rank and compare the solutions in the space. In addition, a solution can be optionally *enhanced* with additional changes to be evaluated by the VLM agent more reliably. For example, annotating objects with bounding boxes to assist in reasoning or counting, or cropping and zooming in on relevant objects. In another case, when tasked with "*select the choice with the same number and type of icons as the reference image*", the agent can invoke the `ask()` tool to gather details about the reference image for

Table 3: Space Exploration and Solution Evaluation Tools.

Name	Input Description	Output
Space Exploration Tools		
type(x, *text*)	Click on element x and enter text *text*	Solution c'
click(x)	Click on the center of element x	Solution c'
swap(f)	Enumerates all possible swaps of SWAPPABLE components in the frame f	Solutions C
slide(x, *dir*, c)	Click and drag element x in direction *dir* (top / down / left / right), bounded by its parent, observe changes in a CAPTCHA entity c	Solutions C
drag(x,y)	Click and drag element x to element y	Solutions C
Solution Evaluation Tools		
rank(S, o)	(GPT-4o) Rank a set of solutions S according to objective o and produce a ranking R	a ranking R
compare(s_i, s_j, o)	(GPT-4o) Compare two solutions s_i and s_j regarding the objective o, returns 1 if s_i is over s_j; -1 if s_j is over s_i; and 0 if s_i is equivalent to s_j.	an element $e \in \{1, -1, 0\}$
Enhancing Tools (called by Solution Evaluation Tools)		
mark(x, *obj*)	(GroundingDINO) Annotate element x with bounding boxes of detected object *obj*	x'
focus(x, *obj*)	(GroundingDINO) Zoom in (crop) element x around object *obj* to identify small details	x'
ask(x, q, *fmt*)	(GPT-4o) Ask a question q about element x and answer in format $a \in fmt$, $fmt \in \{\text{String}, \text{Bool}, \text{Int}\}$	a
match(x, y)	(Hu moments [45] + color palette) Check if element x and element y are visually similar	True / False

comparison. To support this, we provide a set of *Enhancing Tools* (third section in Table 3) designed to enrich visual input with helpful cues. Finally, the VLM agent autonomously decides when to invoke solution evaluation tools like `rank()` or `compare()`, and selects which enhancing tools to apply on the solution for improved reasoning.

In summary, we design Halligan to find the optimal solution for a visual CAPTCHA challenge using a hill-climbing approach. By iteratively identifying local optima based on the objective o and exploring new solutions, we progressively approach the global sub-optimum, where the solution aligns with the CAPTCHA objective.

Attacker Countermeasures. Lastly, we introduce low-cost pre-processing techniques to improve robustness of solving visual CAPTCHA challenges. Non-local means denoising and Richardson–Lucy deconvolution address noise and blur using fast, non-deep-learning methods. Additionally, a filtering stage prompts the VLM agent to discard irrelevant frames that are not part of the actual challenge. Further analysis of their effectiveness is in Section 4.3.

4 Experiments

We evaluate Halligan with the following research questions:

- **CAPTCHA-Solving Capability (RQ1):** How is Halligan's visual CAPTCHA-solving performance compared to the state-of-the-art visual CAPTCHA solvers?
- **CAPTCHA Objective Identification and Abstraction (RQ2):** What is the performance of each step in Halligan such as CAPTCHA abstraction and CAPTCHA objective identification?
- **Adversarial Study (RQ3):** How do visual transformation, prompt injection, and distraction attacks affect the performance of Halligan?
- **Field Study (RQ4):** What is Halligan's attack success rate on real-world visual CAPTCHA challenges in the wild?
- **User Study (RQ5)** How effective and efficient are attacker-crafted VLM prompts compared to Halligan?

Given the space limit, more experimental details and analysis are available at [12].

4.1 CAPTCHA-Solving Capability (RQ1)

4.1.1 Baselines

In this study, we select the following visual CAPTCHA solvers as our baselines.

GUI Agent (B1). The agent uses the same method as Halligan to identify interactables but applies pure Chain-of-Thought (CoT) [69] reasoning. Formally, it operates as a Partially Observable Markov Decision Process (POMDP) GUI agent [26, 28, 34, 42, 48], selecting actions based on current observations to transition between states. This also serves as an ablation study, highlighting the benefits of framing CAPTCHA solving as a search problem rather than prompting the agent for immediate actions.

WebVoyager (B2) [40]. This GPT-V web navigation agent parses the HTML DOM to annotate interactive elements on a page screenshot. It is the first to use multimodal inputs (screenshot and textual element lists) to generate web actions. We extend its prompt to cover all actions in Table 2.

ShowUI (B3) [46]. This web navigation agent uses Qwen2-VL-2B with high-quality GUI-instructional fine-tuning. Given only a page screenshot, it outputs actions in absolute coordinates. It is the SoTA to date.

VTTSolver (B4) [36]. The authors propose a modular system combining Faster R-CNN, CNN, and BiLSTM to solve visual reasoning CAPTCHAs like Tencent VTT, achieving SoTA performance on these variants.

GeeSolver (B5) [76]. It uses a ViT encoder with masked autoencoder training and a sequential decoder to attack eight real-world text-based CAPTCHAs from Google, Yandex, Microsoft, Wikipedia, Weibo, Sina, Apple, and Ganji. It is selected for its effectiveness and robustness.

PhishDecloaker (B6) [64]. It deploys visual CAPTCHA solvers against CAPTCHA-cloaked phishing websites. It handles 4 types of visual CAPTCHAs: reCAPTCHA v2, hCaptcha, slider, rotation with Faster-RCNN, VQA, template matching, and CNN models respectively. Selected for targeting mainstream visual CAPTCHAs.

4.1.2 Benchmark

Existing visual CAPTCHA datasets [9, 14] consist of samples in the form of static image-label pairs. However, CAPTCHA services issue interactive challenges, which makes static image-label pairs less representative of real-world scenarios. To create a realistic and reproducible test environment, we develop an interactive benchmark with 26 types of visual CAPTCHAs, totaling 2,600 challenges. We select the CAPTCHA types based on their availability (as paid services), popularity (from the top 1 million websites [5]), and AI-completeness (requiring AGI for human-level performance). We exclude technically feasible CAPTCHAs that lack widespread practical use (e.g., video or sensor-based), and those solvable through non-interactive methods (e.g., behavioral-only or proof-of-work CAPTCHAs).

To replicate CAPTCHAs, we use Playwright to crawl 100 challenges per type from demo sites with custom API keys, collecting layout (HTML, CSS, JavaScript) and challenge data (image, instructions). For 5/26 types without demos or APIs, we find real websites to scrape by referencing domains in [31] that correspond to the desired visual CAPTCHA types. Two co-authors labeled separate challenge sets to establish ground truth. For continuous-answer (*drag*, *slide*) tasks, they agreed on a tolerance range. A third author helped resolve any disagreements. We set up challenges on a Flask server with server-side rendering. Upon submission, the server provides JSON feedback (`solved: True/False`) to verify if the state matches ground truth. A demo is available at [4].

4.1.3 Experiment Setup

Halligan and other agents (i.e., B1-B3) are evaluated online through interactive challenges presented sequentially in a web browser. Upon submission, the browser automatically loads the next challenge. The solve rate can be automatically calculated by checking all submissions in the server. Both Halligan and B1 use GPT-4o as the VLM agent due to its low cost, efficiency, and high performance [15]. Halligan uses PyAutoGUI to observe and interact with these challenges. It can propose code solutions with up to 3 retries in case of execution errors. GUI Agent is implemented using Playwright and is equipped with the same action tools as Halligan. It can invoke an action per turn up to 10 turns per challenge, where a turn comprises a complete (state-action-observation) sequence. In contrast, B4-B6, are evaluated offline using static image-label pairs as they do not provide GUI interaction capabilities applicable for our benchmark.

Table 4: Results of the comparison study. **B1**: GUI Agent, **B2**: WebVoyager, **B3**: ShowUI, **B4**: VTTSolver, **B5**: GeeSolver, **B6**: PhishDecloaker. (+*x*) represents change ablated to **B1**. - indicates solver is incompatible with the CAPTCHA.

CAPTCHA Type	Solve Rate (% solved)						
	Halligan	B1	B2	B3	B4	B5	B6
tencent/vtt	23 (+23)	0	0	15	50	-	-
mtcaptcha	66 (+35)	31	69	3	-	4	-
yandex/text	96 (+96)	0	93	0	-	43	-
botdetect	84 (+30)	54	70	47	-	96	-
recaptchav2	68 (+59)	9	0	0	-	-	72
baidu/rotate	71 (+64)	7	0	0	-	-	72
hcaptcha	82 (+78)	4	0	0	-	-	74
geetest/slide	16 (+16)	0	0	0	-	-	99
lemin	13 (+13)	0	0	0	-	-	-
amazon/waf	14 (+13)	1	0	0	-	-	-
funcaptcha/counting	54 (+33)	21	0	19	-	-	-
funcaptcha/hand_number	49 (+26)	23	0	17	-	-	-
funcaptcha/galaxies	100 (+54)	46	0	24	-	-	-
funcaptcha/dice_pair	43 (+26)	17	0	14	-	-	-
funcaptcha/card	82 (+70)	12	0	16	-	-	-
funcaptcha/square_icon	86 (+61)	25	0	19	-	-	-
funcaptcha/rotated	95 (+84)	11	0	15	-	-	-
arkose/3d_rollball	20 (=0)	20	0	0	-	-	-
arkose/dice_match	71 (+61)	10	0	24	-	-	-
arkose/orbit_match	48 (+29)	19	0	25	-	-	-
arkose/rockstack	58 (+37)	21	0	2	-	-	-
arkose/numbermatch	54 (+38)	16	0	14	-	-	-
geetest/icon	46 (+46)	0	0	0	-	-	-
geetest/iconcrush	98 (+92)	6	0	0	-	-	-
geetest/gobang	92 (+73)	19	0	0	-	-	-
yandex/kaleidoscope	47 (+42)	5	0	0	-	-	-

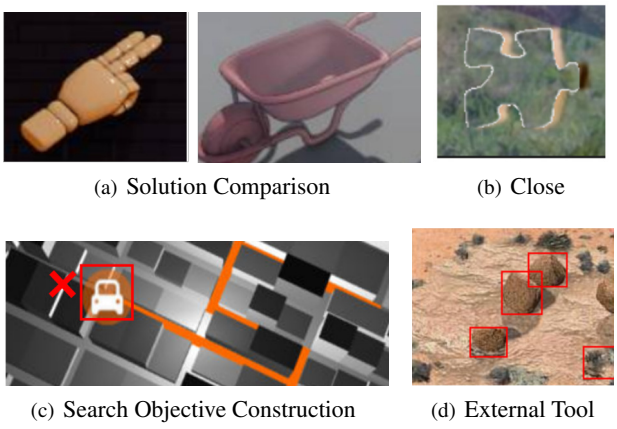
4.1.4 Results & Discussion

Table 4 shows the results. Halligan successfully solved 1,577 out of 2,600 challenges, achieving a solve rate of 60.7%. The results show that Halligan is capable of solving visual CAPTCHA challenges in the benchmark without few-shot examples, pre-training or fine-tuning. In contrast, the deep-learning solvers for specific visual CAPTCHAs, VTTSolver, GeeSolver, and PhishDecloaker can only solve 1/26 (3.8%), 3/26 (11.5%), and 4/26 (15.4%) types of visual CAPTCHA challenges in the benchmark respectively. In addition, Halligan can outperform GeeSolver on text-based CAPTCHAs such as `mtcaptcha` and `yandex/text`, while solving comparable number of challenges on `botdetect`. Furthermore, Halligan exhibits performance on par with PhishDecloaker on mainstream CAPTCHAs like `recaptchav2` and `hcaptcha`.

- What types of visual CAPTCHA challenges Halligan can and cannot solve well? We grouped results by interaction type: discrete actions such as *click* and *swap*, and continuous actions such as *drag* and *slide*, and computed average solve rates using Table 4. Halligan performs well on discrete tasks (e.g., swap, click, point) with an average of 65%. In the best case scenario, 65% translates to a 96% success rate within

3 attempts and 99% within 5, which is alarming in practice. In addition, we further categorize Halligan's failure cases for each type of visual CAPTCHA challenge as follows (see Figure 7):

- **External Tool**: Failures caused by errors in tools like the object detector used by `mark()` and `focus()` in Table 3.
- **Search Objective Construction**: The failures are caused by identifying the wrong CAPTCHA objective.
- **Solution Comparison**: The failures are caused by the CAPTCHA solution evaluation does not return suboptimal solution. In addition, we observe that some suboptimal solution (with CAPTCHA unsolved) is very close to the true solution (see see Figure 7(b) for example), we further label them as a *Close* subcategory, which can be addressed with additional engineering efforts.



(a) Solution Comparison (b) Close

(c) Search Objective Construction (d) External Tool

Figure 7: Examples of failure cases: (a) **Expected:** Object faces the direction of hand. **Actual:** Object has a opposite direction. (b) **Expected:** Puzzle piece fits perfectly in slot. **Actual:** Near-correct solution with gap in slot. (c) **Expected**: Point to the end of the car's path. **Actual:** The agent assumes the car has arrived at its destination (end path) and points at the car. (d) **Expected:** All rocks are marked. **Actual:** 1 false negative (rock) and positive (grass marked as rock).

Figure 8 shows the distribution. Overall, the *External Tool* category takes 218 out of 1023 (21.3%), the *Search Objective Construction* category takes 488 out of 1023 (47.7%), the *Solution Comparison* category (with non-*Close* category) takes 37 out of 1023 (3.6%), and the *Solution Comparison* category (with *Close* category) takes 280 out of 1023 (27.4%). Our investigation shows that the *External Tool* failures can be caused by bad outputs from `mark()` and `focus()` (i.e., using object detection model), leading to faulty reasoning and incorrect solutions downstream. For example, mistaken bounding box object annotations, or focusing on wrong regions. Similarly, *Search Objective Construction* failures can be arise

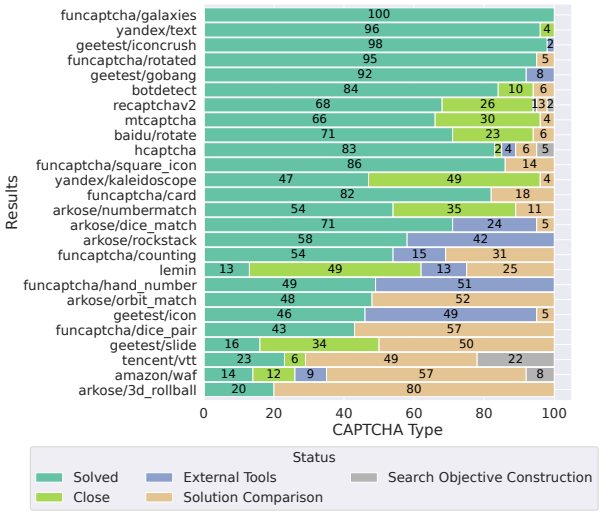


Figure 8: Categorization of failure cases by type of visual CAPTCHA challenge.

from `ask()`, `rank()`, and `compare()`, which use gpt-4o for visual question answering. Nevertheless, with the advance of VLM in the coming years, we expect those mistakes can be mitigated in a way. Nevertheless, with the advance of VLM in the coming years, we expect those mistakes can be mitigated.

- How does Halligan outperform web navigation agents (B2, B3)? Overall, Halligan outperforms the web navigation agents (B2, B3). Notably, WebVoyager and ShowWUI achieve solve rates of 8.9% (232/2600) and 9.8% (254/2600), respectively. To understand the performance gap, we randomly sampled 10 failure cases per CAPTCHA type for both agents and reviewed their execution logs. The key reasons for failure are as follows: (1) *Limitations in identifying interactable elements*: WebVoyager detects interactable elements by parsing the HTML DOM tree. This is effective in benchmarks like MiniWob, where agents manipulate DOM structures (e.g., dragging an item by moving its <li> tag to the third <li> position). However, CAPTCHAs often counteract such methods by rendering their content within a single iframe or canvas element, collapsing multiple UI components into a single non-interactive DOM element. This explains WebVoyager's consistent 0% solve rate on services like `funcaptcha`, `arkose`, `geetest`, `recaptchav2`, `hcaptcha`, though it performs well on unobfuscated text CAPTCHAs requiring only <input> field. In contrast, Halligan operates at the visual level, using computer vision to detect and interpret entities. This allows it to remain effective even when traditional DOM parsing is thwarted by obfuscation or anti-bot defenses. (2) *Overfitting to Common Web GUI Instructions*: Like Halligan, ShowUI operates at the visual level using screen coordinates, but it does not require annotation of interactable elements. However, we observed that ShowUI is heavily fine-tuned on standard web navigation tasks that focus on the next immediate ac-

tion. This short-term, step-by-step optimization often neglects broader visual reasoning, leading to degraded performance on `funcaptcha` and `arkose`. Additionally, the UI grounding (2.7M elements) and navigation (137K samples) datasets are biased toward single-click interactions (e.g., buttons, checkboxes, links), limiting the agent's ability on complex actions such as dragging, swapping, sliding. In contrast, Halligan is not modeled as a POMDP web navigation agent. Instead, it frames CAPTCHA solving as a search problem, systematically exploring and evaluating possible actions against the objective to identify the correct solution.

- Why and when baseline GUI agent is not effective? Halligan consistently surpasses the baseline GUI agent (B1) across all evaluated visual CAPTCHA challenges. Upon investigation, the fundamental reason lies in that Halligan's formulation to a search problem introduce external information (e.g., tools and external object detectors) to VLM to mitigate its otherwise hallucination. We report our observation in details as follows. *Misinterpretation of tasks*: For example, in tasks to find 1 matching pair (e.g., dice, cards, patterns) from 6 options, B1 attempts to create pairings from 2 choices instead of selecting 1. This emphasizes the need to for proper problem modeling. Halligan splits multichoice frames into subframes for individual reasoning. (2) *Misunderstanding action outcomes*: In a task to cycle through choices using left/right arrows to find the correct dice sum, B1 treats arrows as adjusting the sum by 1. This highlights the importance of proper search space formulation, where Halligan treats arrow buttons as generic transitions. (3) *Noisy set-of-mark prompts*: B1 often interacts with the wrong elements, as seen in challenges with dense layouts. For example, in `yandex/text`, the agent mistakenly types in the hint box rather than the input field, while Halligan avoids this by only annotating relevant interactive elements. (4) *Incorrect tool usage*: In `geetest/iconcrush`, to match 3 identical items in a row, B1 uses `drag()` to swap items instead of simply clicking them, whereas Halligan selectively exposes tools like `swap()` based on annotated interactables. (5) *Inability to handle continuous challenges*: The agent struggles with dragging, pointing, and sliding tasks, as shown by a near 0% solve rate across 5 challenge types (i.e., `amazon/waf`, `tencent/vtt`, `geetest/slide`, `geetest/icon`, `yandex/kaleidoscope`). It fails to determine precise distances or coordinates, while Halligan formulates these tasks as discretized combinatorial searches and uses a coarse-to-fine strategy to iteratively refine the solution.

- What is the overhead of solving a visual CAPTCHA challenge? The median cost to attack a challenge with Halligan is $0.024[2], which is generally acceptable. In addition, it takes a median time of 21.8s to solve a challenge with Halligan. This is the end-to-end time from objective identification to solution composition. In the last stage, it takes a median time of 4.5s to compose the solution and 6.3s to execute it. Figure 10 shows the distributions of solution generation and execution times. However, by leveraging amortization once again, this time can be reduced to just the code execution phase, resulting in a 3.46x speedup or a 71.1% time reduction. Halligan is on par with existing human (3.1 – 42s) and ad hoc bot (0.016 – 17.5s) solving times [58].

[2]Following OpenAI's pricing of $2.50 per 1M input, $10.00 per 1M output tokens (2025)

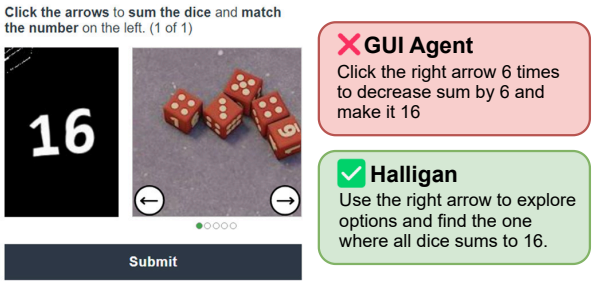


Figure 9: Example of baseline agent underperformance: failure to construct the correct search space due to lack of task objective and annotated interactables.

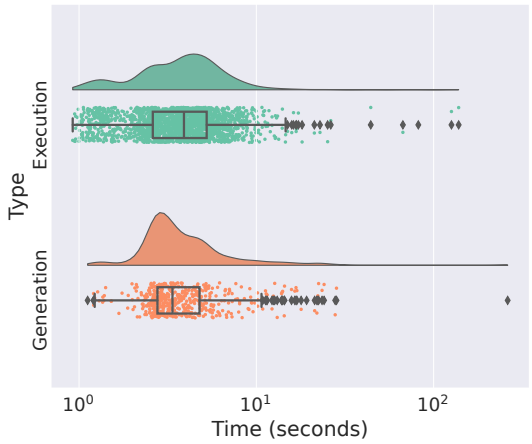


Figure 10: Distribution of code generation and execution time in solution composition stage.

4.2 CAPTCHA Objective Identification and Abstraction (RQ2)

To assess how intermediate failures affect CAPTCHA solving (Section 3.4), we evaluate two steps: (1) CAPTCHA abstraction (Section 3.3) and (2) objective identification (Section 3.2). We sample 10 challenges per type to create a 260-challenge test set, follow the setup in Section 4.1.3, and manually analyze outputs from (1) and (2). Below, we present our analysis and key findings.

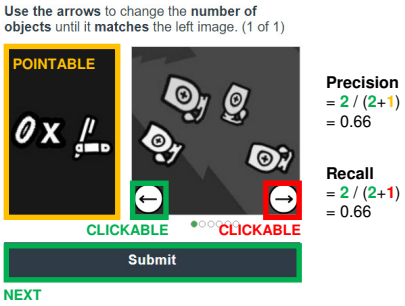


Figure 11: Example of CAPTCHA abstraction step evaluation. Green boxes are correct annotations. Red boxes are false negatives (missed annotations), Orange boxes are false positives (extra annotations).

4.2.1 CAPTCHA Abstraction

Evaluation Metric. We evaluate the precision and recall of interactables annotated by Halligan (see Section 3.3 and Figure 6). Precision is defined as # correct annotations / # annotations by Halligan, and recall as # correct annotations / # ground truth annotations. An annotation is correct if it matches both the interactable type (per Table 2) and the target entity (frame, element, or keypoint).

Results. Halligan achieves an average precision of 0.71 and an average recall of 0.79. Among the 260 challenges, 8/260 (3.1%) fail to execute on the first try due to errors from referencing out-of-index frames and non-existent component attributes, which can be corrected by providing error feedback to the agent. 120 out of 252 challenges (47.6%) reported false positives in their abstraction, while 62 out of 252 challenges (24.6%) reported false negatives.

Analysis of False Positives (FP). An FP occurs when Halligan annotates a non-interactable entity as interactable. Of the 120 errors, 69 (57.5%) are due to irrelevant annotations on instruction and reference frames, 4 (3.3%) are due to irrelevant elements, and 51 (42.5%) are due to incorrect or missing annotations (detailed later). FPs have a negligible impact, as the agent learns to ignore them in CAPTCHA-solving step.

Analysis of False Negatives (FN). An FN occurs when Halligan misses a ground truth interactable entity. Of the 62 errors, 14 (22.6%) are due to missing frame (e.g., multichoice) and 15 (24.2%) to missing element annotations (e.g., buttons), 3 (4.8%) to incorrect annotation locations, 23 (37.1%) to incorrect annotation types, and 7 (11.3%) to incorrect usage of the `split` function (e.g., incorrect rows and columns) for dividing multichoice frames.

Analysis of Wrong Annotation Types. These are cases where the target entity is correct but labeled with the wrong interactable type. Of the 23 errors, 11 (47.8%) misclassified *clickable* as *slideable* (e.g., prev/next arrows), 4 (17.4%) misclassified *next* as *clickable* (e.g., submit buttons), 3 (13.0%) misclassified *selectable* as *pointable* (e.g., multichoice options), 3 (13.0%) misclassified *inputtable* as *pointable*, and 1 (4.3%) misclassified a *draggable* element as *clickable* (e.g., selecting instead of dragging a puzzle piece).

Addressing Wrong and Missing Annotations. While Halligan has both false positive and negative errors, we confirm that it successfully generates a perfect annotation at least once for every type of visual CAPTCHA challenge. We attribute this behavior to the inherent stochasticity of VLM outputs; even with deterministic configurations, some level of error is expected. We can mitigate these errors through two strategies: (1) *Sampling*, generating multiple abstraction candidates and selecting the optimal candidate via majority voting; and (2) *Caching*, storing projection functions in a vector database, indexed by a key derived from the CAPTCHA's frames, relations, description, and objective. The focus of this work is on the security of visual CAPTCHA challenges. We leave engineering improvements for future work.

4.2.2 CAPTCHA Objective Identification

Evaluation Metric. Given the output in Figure 15, we evaluate accuracy by calculating the total score divided by the maximum score. A score of (+1) is awarded for each satisfied criterion: (1) `describe()` provides accurate context for understanding the visual CAPTCHA challenge, (2) `relate()` correctly identifies frame relations (as defined in Table 1), and (3) `objective()` identifies the correct actions, entities, and final goal.

Results. Halligan achieves a score of 1367 out of 1421, resulting in an average accuracy of 96%.

Analysis of Failure Cases. We provide a detailed breakdown of deducted scores. *Frame descriptions* represents 23/54 (42.6%) cases: 18 due to missing details (e.g., dice numbers, icons, symbols) and 5 due to incorrect information (e.g., hand direction, object count). *Frame relations* represents 3/54 (5.6%) cases: 2 due to missing instruction-to-frame relations and 1 for referencing an incorrect frame. *Task objective* represents 28/54 (51.8%) cases: 15 incorrect preemptive answers, 5 ambiguous goals (e.g., matching images should involve matching animals, colors, or icons), 5 incorrect actions, and 3 incomplete action sequences. We observe all of these mistakes do not affect subsequent stages, with the exception of incorrect preemptive answers, which lead the agent to submit a hard-coded answer. We mitigated this by instructing the agent to ignore answers provided in the objective and always use tools to verify the solution.

4.3 Adversarial Study (RQ3)

We design adversarial experiments for Halligan to evaluate its robustness against defender countermeasures aimed at reducing the solver's success rate, both with and without the attacker countermeasures described in Section 3.4.

4.3.1 Defender Countermeasures

We focus on black-box evasion techniques commonly used to subvert ML systems online [19,38]. This decision is supported by real-world evasion attacks on content moderation systems [30,63,75], cloud image services [71], and phishing detectors [19,64]. We introduce three defensive measures grounded in practical scenarios: visual transformation, visual prompt injection, and distraction.

Visual Transformation. We prepare two image augmentations: Gaussian noise, an additive transformation that introduces normally distributed noise, and motion blur, which reduces image clarity and sharpness.

Visual Prompt Injection. Carefully crafted prompts can manipulate VLM agents into disobeying instructions or performing harmful actions [68]. To explore this, we implemented typographic defense [20,56], which misleads agents by superimposing deceptive, CAPTCHA-unrelated instructions onto the CAPTCHA image. In addition, recognizing that agents are trained to reject unethical or harmful requests, we introduced a refusal defense that superimposes warnings onto the CAPTCHA image, emphasizing the unauthorized and unethical nature of solving it, and even threatening potential legal consequences.

Distraction. A VLM agent's limited context window constrains its attention and visual perception. Drawing parallels to how users experience cognitive overload from excessive online information [23], we collected 70 online advertisement banners and randomly injected n banners around the CAPTCHA $(n = 1, 2, \ldots, k)$ to evaluate how VLM agents handle distractions.

4.3.2 Experiment Setup

We follow the same setup as Section 4.1.3 to evaluate Halligan. However, we limit the challenge pool to those solved by Halligan, randomly selecting 10 challenges per visual CAPTCHA type, creating a test set of 260 samples. All attacks are applied during the objective identification stage, Gaussian noise and motion blur transform each frame $\mathcal{F}$ with random intensity. Typographic, refusal, and distraction attacks randomly insert attack frames from respective sets ($\mathcal{F}_{\text{attack}}$), with n frames interleaved with the original frames in $\mathcal{F}$ to form $\mathcal{F}' = \mathcal{F} \oplus A$. The specific attack sets and n values are: 3 for typographic (frames with misleading instructions), 1 for refusal (frames with warning messages), and 4 for distraction (frames that are advertisement banners).

4.3.3 Results & Discussion

Table 5 presents the evaluation results. Defensive measures reduce solve rates by 15.4%–49.6%. However, countermeasures recover the solve rates to 56.5%–85.4%. Regardless, no defense fully prevents the attacks.

Table 5: Results of the adversarial study. **A** and **B** indicate the solve rates with (after) and without (before) countermeasures. **A1**: Gaussian noise, **A2**: Motion blur, **A3**: Typographic, **A4**: Refusal, **A5**: Distraction.

CAPTCHA Type	Challenges Solved									
	A1		A2		A3		A4		A5	
	B	A	B	A	B	A	B	A	B	A
tencent/vtt	3	3	4	4	2	3	3	6	4	5
mtcaptcha	1	9	7	7	0	10	3	10	8	9
yandex/text	0	9	0	3	4	9	3	10	5	10
botdetect	3	9	6	6	4	9	5	9	4	9
recaptchav2	5	6	8	8	4	9	5	7	8	8
baidu/rotate	3	6	7	9	4	10	9	10	7	9
hcaptcha	8	8	7	7	1	6	7	7	8	8
geetest/slide	1	1	4	4	1	4	4	6	4	5
lemin	0	3	3	3	0	8	2	8	3	8
amazon/waf	0	7	3	7	2	10	6	10	5	10
funcaptcha/counting	4	9	4	5	0	10	9	9	10	10
funcaptcha/handnumber	7	7	6	6	0	8	5	6	5	7
funcaptcha/galaxies	3	10	7	10	4	10	7	10	8	10
funcaptcha/dice_pair	6	6	5	5	0	8	5	8	3	8
funcaptcha/card	2	5	2	2	0	10	6	9	2	9
funcaptcha/square_icon	3	10	6	7	1	10	10	10	7	10
funcaptcha/rotated	10	10	9	9	1	10	8	8	9	9
arkose/3d_rollball	4	4	7	7	2	5	5	6	3	6
arkose/dice_match	3	5	3	7	3	9	7	8	6	8
arkose/orbit_match	1	4	2	2	1	7	5	7	7	9
arkose/rockstack	4	4	7	9	2	10	9	10	9	10
arkose/numbermatch	2	8	4	5	2	9	1	10	3	10
geetest/icon	0	6	1	1	1	6	5	5	0	9
geetest/iconcrush	0	4	5	5	1	10	3	10	9	10
geetest/gobang	0	8	3	3	3	10	6	7	6	7
yandex/kaleidoscope	3	6	6	6	3	10	8	9	10	9
Total	76	167	126	147	46	220	146	215	153	222

Analysis of Failure Cases. Figure 12 categorizes the impact of defensive measures. Visual prompt injection has the greatest impact on visual CAPTCHA solving (*Stage 3*). Among them, typographic defenses proved the most effective by exploiting agents' instruction-following behavior. It tricks the agent into solving nonexistent challenges, causing logic errors, undefined operations, and hallucinated functions. This aligns with findings that VLMs are more vulnerable to text than image manipulation [29]. However, it can be mitigated by training VLMs with hierarchical instruction priorities [67]. Refusal attacks causes the agent to submit blank or nonsensical responses. Distraction attacks mislead the agent into referencing wrong frames, causing indexing errors.

On the other hand, visual transformation disrupts CAPTCHA abstraction (*Stage 2*), tool outputs (*Vision Tools*), and the VLM agent's perception during search solution evaluation (*Optimization*). Vision algorithms and models are vulnerable to these transformations, and agents struggle to split frames or accurately identify rows, columns, and tiles. Objective identification (*Stage 1*) is less affected, with errors mainly caused by illegible instructions post-transformation.

Analysis by Challenge Type. Defenses vary across challenges. In refusal defense, task rejection rate is higher when

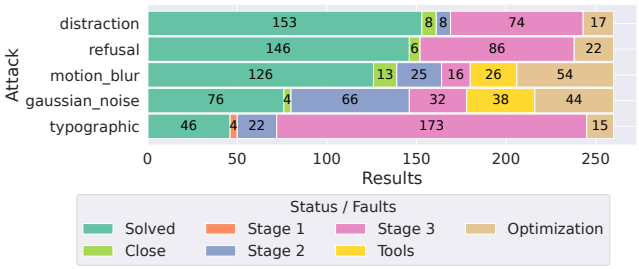


Figure 12: Categorization of failure cases caused by defensive measures.

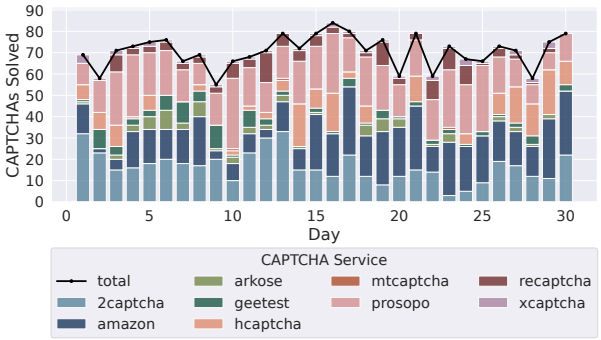


Figure 13: Results of the 30-day field study.

instructions mention "puzzle" or display a jigsaw puzzle, and with traditional text-based challenges. Additionally, visual transformation is more effective on continuous challenges, with all four types showing fewer tasks solved compared to visual prompt injection and distraction attacks.

Nonetheless, no defense can block 100% of attacks. While these measures increase the attack cost, countermeasures can recover the solve rate. Moreover, since CAPTCHA attacks can be attempted repeatedly, even with a drop to 15% ($p = 0.15$) as seen without countermeasures, the probability of success after k attempts is $1-(1-p)^k$. For instance, to achieve over 80% success, we can solve for k: $1-(1-0.15)^k \geq 0.80 \Rightarrow k \geq \frac{\ln(0.20)}{\ln(0.85)} \approx 10$

4.4 Field Study (RQ4)

To evaluate Halligan in realistic settings, we conducted a field study on live, in-the-wild CAPTCHAs that involve behavioral verification beyond solving the puzzle itself. Inspired by [31], we infiltrated 2Captcha, a human-driven CAPTCHA farm.

Evaluation Metric. Halligan's success is measured by its ability to function as a worker and retrieve a CAPTCHA token from the CAPTCHA provider (details in [31]), which 2Captcha then forwards to its customer. If the visual challenge is not solved, 2Captcha would not receive a token. The success rate is calculated as the ratio of total tokens received to total CAPTCHA tasks attempted.

Worker Device Configuration. We used a personal computer (1920×1080 resolution, Win 11 x64) with an active browsing history. A single IP address was used without rotation.

Worker Software Configuration. We ran 2Captcha's worker client during the day and paused it overnight, resulting in a non-uniform task arrival pattern. We used a single worker account for the entire duration of the study. Fortunately, the account was not banned, indicating that Halligan's activity did not arouse suspicion from 2Captcha or its customers

Worker Behavior Configuration. When Halligan interacts with visual challenges, it uses simplified, linear mouse movements. While many CAPTCHA providers analyze mouse behavior [2, 8], the relative weight of this signal compared to others (e.g., identity profiling) remains unclear [55, 60, 61]. To offset Halligan 's minimal behavioral signals, we relied on 2Captcha's infrastructure (e.g., proxies) to inherit customers' identities. Furthermore, we did not consider behavioral-only CAPTCHAs (e.g., reCAPTCHAv3, Cloudflare Turnstile).

Data Collection and Baseline Evaluation. We used *mitmproxy* to intercept 2Captcha traffic and extract task data, including CAPTCHA images and target URLs. Two co-authors manually labeled the tasks for ground truth. Halligan was evaluated online on live tasks, while baseline solvers (B4–B6) were tested offline using the labeled data.

4.4.1 Results & Discussion

We ran the study for a duration of 30 days. In the end, we accepted a total of 3000 CAPTCHA tasks.

Analysis by Time. Halligan successfully attacked the visual challenges in 2,117 out of 3,000 CAPTCHA tasks, with a daily success rate fluctuating between 55% and 84%, and an average of 70.6%. Results are shown in Figure 13. Halligan encountered 9 distinct CAPTCHA providers, 4 of which (`2captcha`, `amazon`, `prosopo`, `xcaptcha`) were not part of the benchmark. We define `2captcha` as a generic service for text-based CAPTCHAs. Some services, such as `2captcha`, `amazon`, and `prosopo`, were targeted daily, while others, like `arkose`, `geetest`, `hcaptcha`, and `recaptcha`, appeared intermittently. Regardless of popularity, any CAPTCHA provider is vulnerable to attack. When analyzing tasks by CAPTCHA type, Halligan encountered 25 unique variants, 17 of which were not present in the benchmark, accounting for 75.2% (2,252 out of 3,000) of the tasks. Despite this diversity, the visual challenges part of the task consistently relied on a limited set of interaction types (e.g., click, drag, slide), which Halligan was able to handle. In contrast, the baseline solvers addressed only 6 variants.

Analysis by CAPTCHA Service. Table 6 shows success rates by CAPTCHA service. Although `recaptcha` has the highest market share, it accounts for 6% of tasks. In contrast, services with frequently updated challenges (e.g., `prosopo`, `hcaptcha`, `arkose`, and `geetest`) make up 50.3% of all tasks. Halligan outperforms or matches all baseline solvers. We analyzed

Table 6: CAPTCHA success rates by CAPTCHA providers.

Service	Tasks	Solve Rate (%)			
		Halligan	B4	B5	B6
prosopo	652	560 (85.9%)	-	-	-
amazon	549	518 (94.4%)	-	-	-
2captcha	672	498 (74.1%)	-	121 (18.0%)	-
hcaptcha	250	197 (78.8%)	-	-	116 (46.4%)
recaptcha	181	130 (71.8%)	-	-	88 (48.6%)
geetest	390	99 (25.4%)	4 (1.0%)	-	0 (0.0%)
arkose	218	72 (33.0%)	-	-	-
xcaptcha	84	40 (47.6%)	-	-	-
mtcaptcha	4	3 (75.0%)	-	2 (50.0%)	-

Table 7: Top 10 targeted domains in field study.

Domain	Category	Rank	CAPTCHA
edge.app	Crypto	65,136	Prosopo
galxe.com	Crypto	29,108	GeeTest
catch.com.au	E-commerce	32,132	Amazon WAF
gopluslabs.io	Crypto	60,181	GeeTest
yahoo.net	News & Media	2,294	reCAPTCHA v2
fazenda.gov.br	Government	3,881	hCaptcha
tjpe.jus.br	Government	86,001	Amazon WAF
micropact.com	Technology	118,073	Amazon WAF
booking.com	Travel	285	Amazon WAF
linkedin.com	Social Media	19	Arkose Labs

the logs of failed cases among baselines and found that: (1) four providers are unsupported by B4–B6; (2) B4 and B5 overfit to benchmark training data and fail under real-world shifts in layout and style, as seen in `geetest`, `mtcaptcha`, and `2captcha`; (3) B6 relies on parsing DOM-level elements in `geetest`, which is infeasible in non-browser environments; (4) B6 underperforms on `recaptcha` due to its assumption of simple 3×3 tasks, while real-world challenges often involve 4×4 grids and multi-step prompts; and (5) B6 struggles with `hcaptcha` due to its evolving pool of tasks requiring fine-grained pointing and complex visual reasoning.

Analysis by Targeted Websites. We traced all 3,000 task requests to 1,146 unique URLs, corresponding to 119 fully qualified domain names (FQDNs) and 106 eTLD+1 domains. Of these, 63 eTLD+1 domains (59.4%) were targeted by repeated CAPTCHA attacks (>1 task). We enriched the domains with categorical and popularity information from Cloudflare Radar and Tranco. Table 7 lists the top 10 most targeted websites. Despite the widespread use of reCAPTCHA v2, it does not dominate the rankings. Instead, half of the domains rely on CAPTCHA services that frequently update, forcing attackers to depend on human-driven CAPTCHA farms. We also categorized the eTLD+1 domains, revealing that CAPTCHA attacks impact websites of all sizes, from global platforms to local government services, as shown in Table 10.

4.5 User Study (RQ5)

To compare Halligan against manually crafted prompts, we invited participants to solve visual CAPTCHA challenges.

Table 8: User Study Results. **P**: Participants 1-5. **H**: Halligan. T_0: Time (min) to first working prompt. T_{tr}: Time (min) to complete training set. S_{tr}: Solve rate on training set. S_{te}: Solve rate test set. $T > 30$: Time limit exceeded (30 min).

Ps.	CAPTCHA A				CAPTCHA B				CAPTCHA C			
	T_0	T_{tr}	S_{tr}	S_{te}	T_0	T_{tr}	S_{tr}	S_{te}	T_0	T_{tr}	S_{tr}	S_{te}
P1	>30	>30	0.00	0.00	10.2	28.0	1.00	0.48	3.7	>30	0.80	0.22
P2	2.9	>30	0.90	0.21	7.8	24.3	1.00	0.16	5.5	>30	0.80	0.34
P3	11.6	>30	0.30	0.18	12.8	>30	0.80	0.17	6.0	>30	0.80	0.34
P4	14.9	>30	0.20	0.18	24.6	>30	0.20	0.21	20.5	>30	0.60	0.31
P5	14.5	>30	0.20	0.28	3.9	16.1	1.00	0.36	5.3	>30	0.10	0.18
H	0.7	1.2	0.70	0.76	0.3	1.2	1.00	0.98	0.3	0.7	0.50	0.54

4.5.1 Experiment Setup

Participants. We recruited 5 Computer Science participants with 2+ years of LLM prompting experience.

Study Procedure. Participants were tasked with designing prompts for solving 3 types of visual CAPTCHA challenges using GPT-4o. Each type included 10 training and 90 testing challenges. The goal is to create prompts that generate parseable JSON outputs (e.g., actions, coordinates) executable on-screen. We selected 3 CAPTCHA types: `baidu/rotate`, `geetest/iconcrush`, and `funcaptcha/counting` (referred to as A, B, and C respectively). These were chosen based on (1) participants' prior familiarity and (2) varying difficulty levels for Halligan (with solve rates of >50%, >70%, and >90%). Participants received a pre-test briefing and completed a short survey. During the test, they completed a warmup exercise and had up to 30 minutes to create prompts for each challenge type. Afterward, we conducted a post-test interview to understand their experience.

Environment. Testing was conducted on a 1920x1080 Windows desktop using Chrome browser, with apps for prompting GPT-4o, capturing screenshots, annotating images, and displaying mouse coordinates.

Evaluation Metrics. We recorded the time each participant took to produce their first working prompt (T_0) and the total time to complete all training challenges (T_{tr}). Prompt effectiveness was measured by the number of training (S_{tr}, out of 10) and testing (S_{te}, out of 90) challenges solved. A challenge was considered solved if the VLM output could be automatically mapped to the correct on-screen action.

4.5.2 Results & Discussion

Table 8 summarizes our results. Users took 11.0, 11.9, and 8.2 minutes on average to craft working prompts for challenge types A, B, and C, while Halligan required just 0.4 minutes per type. User prompts often lacked stability and required further tuning, leading to timeouts (>30 mins) in 80% of cases. In contrast, Halligan generated stable Python code in under a minute that generalized across all challenges. On the test set (S_{te}), Halligan outperformed the best user prompts by 172%,

104%, and 58% on types A, B, and C, respectively.

How do user prompts generalize to test samples? Users often overfit to the training set, resulting in relatively lower test performance. In challenge A, P1–P5 noted that VLMs struggles with image orientation, with P2–P4 attributing this to limited training data. They used overly specific cues (e.g., "this image contains...", "the answer should be more/less than..."), failing to realize that framing the task as a search problem enables semantic reasoning over rotation.

How do users learn to construct prompts? User performance does not transfer across CAPTCHA types, even experienced LLM users exceed the time limit. Top performers on one type (e.g., P1, P5) struggle with others, indicating a steep learning curve despite similar task structures. Participants relied on ad-hoc heuristics like angle-to-distance formulas (P1, P5 on challenge A), elimination rules (P2 on challenge B), or brute-forcing with the same prompt (P3 on challenge C). In contrast, Halligan requires no feedback or heuristics, it compares all choices to find the best solution.

5 Discussion

The Call for CAPTCHA Evolution. Modern CAPTCHA systems are multilayered, combining visual challenges, behavioral biometrics (e.g., mouse, keyboard, touch, motion), and identity profiling (e.g., network, device, fingerprinting, cookies). The rise of VLMs has lowered the barrier for attacking the visual challenge layer, weakening its effectiveness and shifting greater reliance onto other parts of the system. This shift exacerbates the long-standing trilemma in CAPTCHA design: balancing security, usability, and privacy. Referencing [41], we outline eight puzzle-less alternatives: (1) *Behavioral-only CAPTCHAs*: Use passive, frictionless layers (e.g., Cloudflare Turnstile); (2) *Information Disparity*: The user knows something a bot doesn't (e.g., security questions); (3) *Hardware Attestation*: The user has something a bot doesn't (e.g., WebAuthn); (4) *Biometrics*: The user is something a bot isn't (e.g., fingerprints); (5) *Social Networks*: Others vouch for the user (e.g., referrals); (6) *Strong Network*: An authority verifies the user (e.g., KYC, OAuth); (7) *Proof-of-Work*: Use computation to make scaling bot attacks costly (e.g., mCaptcha); (8) *Honeypot*: Traps bots with elements invisible to users (e.g., hidden forms).

Limitations. Currently, Halligan struggles with temporal challenges, which involve time-based dynamic elements independent of user actions [50], as well as challenges obscured by other elements. In the field study, future work could assess the impact of different signals on token validity.

Threats to Internal & External Validity. Internally, we compare Halligan's performance to human solve times from [58], differing challenges may cause discrepancies; Externally, our evaluation is limited to GPT-4o, but the core ideas (CAPTCHA meta-model and search problem) are broadly applicable. We expect newer VLMs to yield even better results. In the field study, successfully solving the visual challenge may be a necessary but not sufficient condition for token validity. Some providers (e.g., Google reCAPTCHA Enterprise) perform additional checks on the network traffic, fingerprint, or behavioral patterns [61, 64, 65] when the token is used on the target site. As such, our results provide a best-effort estimation of Halligan's effectiveness based on (1) solving the visual challenge component of the CAPTCHA and (2) retrieving tokens from the provider. Since 2Captcha customers can report invalid tokens, we also considered the continued activity of the worker account (i.e., not banned) as an indirect and positive signal supporting token validity.

Practical Applications. Prior work [64] uses CAPTCHA solvers to detect CAPTCHA-cloaked phishing sites. In Section 4.1.3, if Halligan outperforms [64], it can uncover more threats. To test this, we used 2,600 phishing kits from DynaPD [47], split into 26 groups (100 each), each assigned a different CAPTCHA type from our benchmark. We compared two setups: PhishDecloaker (1) with and (2) without Halligan. Setup (2) detected 1,226 more phishing sites than (1).

6 Related Work

CAPTCHA Solvers. To the best of our knowledge, we are the first to propose a generalist VLM agent for solving visual CAPTCHAs. A related work, Oedipus [32], focuses on reasoning CAPTCHAs with a domain-specific language but lacks open sourced code, real-world evaluations, and adversarial testing. Previous work has explored CAPTCHA solving using machine learning [24, 35], deep convolutional networks [43, 44, 60, 61], generative adversarial networks [74], transformers [76], and reinforcement learning [18, 65]. Additionally, research has been conducted on audio-based CAPTCHAs and bypass techniques [16, 21, 33, 49, 62].

CAPTCHA Surveys & Studies. Guerar et al. [39] compiled a 20-year timeline of 77 CAPTCHA schemes and proposed a taxonomy of 10 groups, highlighting challenges and opportunities in the CAPTCHA ecosystem. Weng et al. [70] focused on image-based CAPTCHAs, proposing an attack framework and evaluating it against 10 popular schemes, while identifying 152 CAPTCHA-solving services. Motoyama et al. [52] analyzed the behavior of 8 text-based CAPTCHA-solving services. Searles et al. [58] conducted an empirical study with 1,400 participants on 10 CAPTCHA types. Nyugen et al. [31] studied CAPTCHA attacks in the wild through MITM attacks on 2 CAPTCHA farms.

7 Conclusion

We reformulated visual CAPTCHA solving as a search optimization problem, enabling VLMs to effectively address the task. Our VLM-based tool, Halligan, successfully solves 26 types of CAPTCHAs in our benchmark. Extensive adversarial, field, and user studies further demonstrate its robustness.

Acknowledgment

We sincerely thank the Shepherd and reviewers for their time, effort, and valuable feedback, which significantly improved the quality of our work. We are also grateful to Prof. Yehuda Afek for his generous support and guidance. Their expertise and advice have been truly invaluable. This research is supported in part by the Minister of Education, Singapore (MOE-T2EP20124-0017, MOET32020-0004), the National Research Foundation, Singapore and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008-1B), and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

Ethical Consideration

A generalized CAPTCHA solver poses severe negative consequences for the Internet ecosystem. It can undermine community trust, degrade the quality of online experiences, and inflict financial loss on web service providers. We fully understand the severity of this work. Therefore, we strictly adhered to these guidelines when conducting research.

- **Responsiblity and Disclosure.** We actively communicated our findings with AI research organizations (i.e., OpenAI, Anthropic, DeepMind) and CAPTCHA services (i.e., Google, Cloudflare, Datadome, Arkose Labs).
- **Conscientious Sharing:** We restrict our source code to registration only and permit access to researchers with verified credentials. For transparency, we will publicly release media demonstrations and execution traces.
- **Benchmark:** The CAPTCHAs in the benchmark were solved offline with no interaction with the CAPTCHA providers. To construct the benchmark, we register our own API keys with CAPTCHA services or use demo websites. As a last resort, we scrape live websites.
- **Field Study:** Halligan's success is measured by its ability to retrieve a token from the CAPTCHA provider. Ultimately, only the target site that receives this token can determine its validity, as they possess the secret (or private key) to make a validation request to the CAPTCHA provider, and intermediaries such as 2Captcha could not do so. In theory, we could have retrieved a token and tampered with it before forwarding it to 2Captcha, thereby avoiding an active role in facilitating attacks against target sites. However, such tampering could raise suspicion if it resulted in errors reported by 2Captcha's customers, potentially leading to our account being banned. Therefore, we opted for a relatively small-scale study (100 tasks/day) to minimize risk.
- **Affected Stakeholders:** All experiments except the user study do not involve human subjects. Although website users could be indirectly affected by our benchmark collection, the potential impact remains minimal. We limit scraping to a small subset of CAPTCHAs (5 out of 26), which are otherwise unobtainable, capping it at 100 challenges per website. Based on published pricing from reCAPTCHA and hCaptcha, this represents around 0.1% of the typical monthly usage in the enterprise tier. To mitigate potential risks to scraped websites, we sanitize all identifiable information (e.g., domain names and CAPTCHA public keys).
- **User Study:** This study received an exemption from our institutional review board due to the de-identification of participants in interviews and tests, absence of personally identifiable information, and the fact that disclosure of results posed no harm to participants. The study also involved no deception or concealment. Participants were pre-screened for normal or corrected-to-normal vision and no accessibility challenges. Informed consent was obtained, and participants were briefed on the study's purpose, duration (up to 2 hours), and potential discomforts (visual strain from prolonged tasks). Participation was voluntary, with the right to withdraw at any time. After the study, participants received a thank-you gift (unrelated to performance), access to study results, and optional public acknowledgment.
- **Deterrence:** Currently, our prototype is relatively costlier and slower than human-driven CAPTCHA farms. We do not intend to perfect the design, but rather demonstrate its potential destruction when fully weaponized.

Open Science

For more details, please visit our project website https://halligan.pages.dev. Both Halligan's source code and the CAPTCHA benchmark are available at https://doi.org/10.5281/zenodo.15580923.

References

[1] Arkose Labs. https://www.arkoselabs.com.

[2] Arkose Labs Mouse Behavior Analysis. https://www.arkoselabs.com/blog/unlocking-captchas-moving-beyond-deterrence-to-detection/.

[3] AWS WAF. https://docs.aws.amazon.com/waf/latest/developerguide/waf-captcha-and-challenge.html.

[4] Benchmark Demo (Anonymized). https://halligan-benchmark.onrender.com/geetest/slide/1.

[5] BuiltWith Trends. https://trends.builtwith.com/widgets/captcha.

[6] DataDome. https://datadome.co/products/datadome-captcha.

[7] GeeTest. https://www.geetest.com/en.

[8] hCaptcha Mouse Behavior. https://www.hcaptcha.com/post/hcaptcha-technical-architecture.

[9] Kaggle Captcha Datasets. https://www.kaggle.com/search?q=captcha+in%3Adatasets.

[10] Lemin. https://www.leminnow.com.

[11] OpenAdapt. https://github.com/OpenAdaptAI/OpenAdapt.

[12] Project Website (Anonymized). https://halligan.pages.dev.

[13] reCAPTCHA v2. https://developers.google.com/recaptcha/docs/display.

[14] Roboflow Captcha Datasets. https://universe.roboflow.com/search?q=class%3Acaptcha.

[15] What LLM Provider? https://web.archive.org/web/20241214022618/https://whatllm.vercel.app.

[16] Hadi Abdullah, Aditya Karlekar, Saurabh Prasad, Muhammad Sajidur Rahman, Logan Blue, Luke A. Bauer, Vincent Bindschaedler, and Patrick Traynor. Attacks as defenses: Designing robust audio captchas using attacks on automatic speech recognition systems. In *NDSS*, 2023.

[17] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[18] Ismail Akrout, Amal Feriani, and Mohamed Akrout. Hacking google recaptcha v3 using reinforcement learning. *arXiv preprint arXiv:1903.01003*, 2019.

[19] Giovanni Apruzzese, Hyrum S Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin Roundy. "real attackers don't compute gradients": bridging the gap between adversarial ml research and practice. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 339–364. IEEE, 2023.

[20] Hiroki Azuma and Yusuke Matsui. Defense-prefix for preventing typographic attacks on clip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3644–3653, 2023.

[21] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. {unCaptcha}: A {Low-Resource} defeat of {reCaptcha's} audio challenge. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[22] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019.

[23] Giorgio Brajnik and Silvia Gabrielli. A review of online advertising effects on the user experience. *International Journal of Human-Computer Interaction*, 26(10):971–997, 2010.

[24] Elie Bursztein, Jonathan Aigrain, Angelika Moscicki, and John C Mitchell. The end is nigh: Generic solving of text-based {CAPTCHAs}. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[25] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 125–138, 2011.

[26] Anthony R Cassandra. A survey of pomdp applications. In *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, volume 1724, 1998.

[27] Kumar Chellapilla and Patrice Simard. Using machine learning to break visual human interaction proofs (hips). *Advances in neural information processing systems*, 17, 2004.

[28] Qi Chen, Dileepa Pitawela, Chongyang Zhao, Gengze Zhou, Hsiang-Ting Chen, and Qi Wu. Webvln: Vision-and-language navigation on websites. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 1165–1173, 2024.

[29] Shuo Chen, Jindong Gu, Zhen Han, Yunpu Ma, Philip Torr, and Volker Tresp. Benchmarking robustness of adaptation methods on pre-trained vision-language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[30] Mauro Conti, Luca Pajola, and Pier Paolo Tricomi. Captcha attack: Turning captchas against humanity. *arXiv preprint arXiv:2201.04014*, 2022.

[31] Hoang Dai Nguyen, Karthika Subramani, Bhupendra Acharya, Roberto Perdisci, and Phani Vadrevu. Cframe:

Characterizing and measuring in-the-wild captcha attacks. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 197–197. IEEE Computer Society, 2024.

[32] Gelei Deng, Haoran Ou, Yi Liu, Jie Zhang, Tianwei Zhang, and Yang Liu. Oedipus: Llm-enchanced reasoning captcha solver. *arXiv preprint arXiv:2405.07496*, 2024.

[33] Valerie Fanelle, Sepideh Karimi, Aditi Shah, Bharath Subramanian, and Sauvik Das. Blind and human: Exploring more usable audio {CAPTCHA} designs. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 111–125, 2020.

[34] Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, et al. Assistgui: Task-oriented pc graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13289–13298, 2024.

[35] Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang, and Jiawei Li. A simple generic attack on text captchas. In *NDSS*, 2016.

[36] Yipeng Gao, Haichang Gao, Sainan Luo, Yang Zi, Shudong Zhang, Wenjie Mao, Ping Wang, Yulong Shen, and Jeff Yan. Research on the security of visual reasoning {CAPTCHA}. In *30th USENIX security symposium (USENIX security 21)*, pages 3291–3308, 2021.

[37] Philippe Golle. Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542, 2008.

[38] Kathrin Grosse, Lukas Bieringer, Tarek R Besold, and Alexandre M Alahi. Towards more practical threat models in artificial intelligence security. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4891–4908, 2024.

[39] Meriem Guerar, Luca Verderame, Mauro Migliardi, Francesco Palmieri, and Alessio Merlo. Gotta captcha'em all: a survey of 20 years of the human-or-computer dilemma. *ACM Computing Surveys (CSUR)*, 54(9):1–33, 2021.

[40] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[41] Scott Hollier, Janina Sajka, Jason White, Michael Cooper, and Matt May. Inaccessibility of captcha: Alternatives to visual turing tests on the web. https://www.w3.org/TR/turingtest/, December 2021. W3C Group Draft Note.

[42] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290, 2024.

[43] Md Imran Hossen and Xiali Hei. A low-cost attack against the hcaptcha system. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 422–431. IEEE, 2021.

[44] Md Imran Hossen, Yazhou Tu, Md Fazle Rabby, Md Nazmul Islam, Hui Cao, and Xiali Hei. An object detection based solver for {Google's} image {reCAPTCHA} v2. In *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)*, pages 269–284, 2020.

[45] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE transactions on information theory*, 8(2):179–187, 1962.

[46] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*, 2024.

[47] Ruofan Liu, Yun Lin, Yifan Zhang, Penn Han Lee, and Jin Song Dong. Knowledge expansion and counterfactual interaction for {Reference-Based} phishing detection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4139–4156, 2023.

[48] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[49] Hendrik Meutzner, Santosh Gupta, Viet-Hung Nguyen, Thorsten Holz, and Dorothea Kolossa. Toward improved audio captchas based on auditory perception and language understanding. *ACM Transactions on Privacy and Security (TOPS)*, 19(4):1–31, 2016.

[50] Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C Van Oorschot,

and Wei-Bang Chen. A three-way investigation of a game-captcha: automated attacks, relay attacks and usability. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 195–206, 2014.

[51] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[52] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Re:{CAPTCHAs—Understanding}{CAPTCHA-Solving} services in an economic context. In *19th USENIX Security Symposium (USENIX Security 10)*, 2010.

[53] Gabriel Moy, Nathan Jones, Curt Harkless, and Randall Potter. Distortion estimation techniques in solving visual captchas. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE, 2004.

[54] Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, and Gene Tsudik. {CACTI}: Captcha avoidance via client-side {TEE} integration. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2561–2578, 2021.

[55] Andreas Plesner, Tobias Vontobel, and Roger Wattenhofer. Breaking recaptchav2. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1047–1056. IEEE, 2024.

[56] Maan Qraitem, Nazia Tasnim, Kate Saenko, and Bryan A Plummer. Vision-llms can fool themselves with self-generated typographic attacks. *arXiv preprint arXiv:2402.00626*, 2024.

[57] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[58] Andrew Searles, Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, Gene Tsudik, and Ai Enkoji. An empirical study & evaluation of modern {CAPTCHAs}. In *32nd usenix security symposium (usenix security 23)*, pages 3081–3097, 2023.

[59] Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, et al. Practices for governing agentic ai systems. *Research Paper, OpenAI, December*, 2023.

[60] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. I am robot:(deep) learning to break semantic image captchas. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 388–403. IEEE, 2016.

[61] Suphannee Sivakorn, Jason Polakis, and Angelos D Keromytis. I'm not a human: Breaking the google recaptcha. *Black Hat*, 14:1–12, 2016.

[62] Saumya Solanki, Gautam Krishnan, Varshini Sampath, and Jason Polakis. In (cyber) space bots can hear you speak: Breaking audio captchas using ots speech recognition. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 69–80, 2017.

[63] Florian Stimberg, Ayan Chakrabarti, Chun-Ta Lu, Hussein Hazimeh, Otilia Stretcu, Wei Qiao, Yintao Liu, Merve Kaya, Cyrus Rashtchian, Ariel Fuxman, et al. Benchmarking robustness to adversarial image obfuscations. *Advances in Neural Information Processing Systems*, 36:42830–42865, 2023.

[64] Xiwen Teoh, Yun Lin, Ruofan Liu, Zhiyong Huang, and Jin Song Dong. {PhishDecloaker}: Detecting {CAPTCHA-cloaked} phishing websites via hybrid vision-based interactive models. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 505–522, 2024.

[65] Ilias Tsingenopoulos, Davy Preuveneers, Lieven Desmet, and Wouter Joosen. Captcha me if you can: Imitation games with reinforcement learning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 719–735. IEEE, 2022.

[66] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 294–311. Springer, 2003.

[67] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

[68] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36, 2024.

[69] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[70] Haiqin Weng, Binbin Zhao, Shouling Ji, Jianhai Chen, Ting Wang, Qinming He, and Raheem Beyah. Towards understanding the security of modern image captchas and underground captcha-solving services. *Big Data Mining and Analytics*, 2(2):118–144, 2019.

[71] Qixue Xiao, Yufei Chen, Chao Shen, Yu Chen, and Kang Li. Seeing is not believing: Camouflage attacks on image scaling algorithms. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 443–460, 2019.

[72] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Twenty-Third annual computer security applications conference (ACSAC 2007)*, pages 279–291. IEEE, 2007.

[73] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554, 2008.

[74] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 332–348, 2018.

[75] Kan Yuan, Di Tang, Xiaojing Liao, XiaoFeng Wang, Xuan Feng, Yi Chen, Menghan Sun, Haoran Lu, and Kehuan Zhang. Stealthy porn: Understanding real-world adversarial images for illicit online promotion. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 952–966. IEEE, 2019.

[76] Ruijie Zhao, Xianwen Deng, Yanhao Wang, Zhicong Yan, Zhengguang Han, Libo Chen, Zhi Xue, and Yijun Wang. Geesolver: A generic, efficient, and effortless solver with self-supervised learning for breaking text captchas. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1649–1666. IEEE, 2023.

[77] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023.

A Appendix

A.1 CAPTCHA Attack Cost

Table 9: CAPTCHA attack cost (Q1 2025 pricing)

Name	Cost per CAPTCHA (cents ¢)
Halligan (gpt-4o, end-to-end)	2.40
Halligan (gpt-4o, amortized)	0.60
2captcha.com	0.05 – 5.00
9kw.eu	0.13 – 0.33
anti-captcha.com	0.05 – 0.50
azcaptcha.com	0.04 – 0.10
truecaptcha.org	0.03
capsolver.org	0.08 – 0.30
deathbycaptcha.com	0.06 – 0.29

A.2 Field Study Statistics

Table 10: Commonly targeted domain categories in field study.

Category	Examples
Crypto	bybitglobal.com, nexo.com, bybit.com
Social Media	tiktok.com, x.com, mewe.com
Gaming	roblox.com, hoyoverse.com
Government	la.gov, justice.gov, hackney.gov.uk
Marketplaces	avito.ru, leolist.cc
Travel	airbnb.com, uber.com, amadeus.com
Technology	google.com, nsis.ru, giantpartners.com, trustcommerce.com, uxlink.io, baidu.com
E-commerce	popmart.com, shopify.com, albertsons.com
Finance	vermittlerregister.info

A.3 Interactables

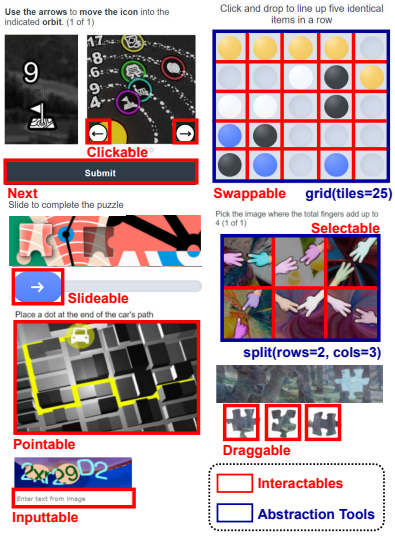


Figure 14: Examples of interactables types (red) and abstracted structure during CAPTCHA abstraction stage (blue).

A.4 Prompts

Input:
1. CAPTCHA image
2. Frame images with captions ("Frame *i*")
System prompt: *You are given n frame(s). First, provide a one-sentence visual description for each frame. Second, identify the relationships between the frames. Third, using the gathered information, determine the sequence of events and the final visual criteria that lead to solving the task.*
Output: A Python block with
1. describe(frame_id: int, description: str)
2. relate(frame_id_a: int, frame_id_b: int, relation: str)
3. objective(steps: str)

Figure 15: Objective identification prompt.

Input:
1. CAPTCHA image
2. Task objective
3. Frame images with captions ("Frame *i*")
4. Frame relations ("Frame *i* to *j*: *relation*")
5. Frame descriptions ("Frame *i*: *description*")
System prompt: *Write a Python script to identify all element/frame-level interactables using these tools:*
⟨Description Parsing Tools⟩
⟨Type Annotation Tools⟩
⟨Abstraction Tools⟩
Output: A Python function using the tools above.

Figure 16: CAPTCHA abstraction prompt.

Input:
1. CAPTCHA image
2. Task objective
3. Non-interactable frame images with captions ("Frame *i*")
4. Interactable frame images with captions ("Frame *i*: *interactable_type*")
5. Frame relations ("Frame *i* to *j*: *relation*")
6. Interactable element images with captions ("Frame *i* Element *j*: *interactable_type*")
System prompt: *Solve tasks using your vision, coding, and language skills. When using code, the user cannot provide any other feedback or perform any other action beyond executing the code you suggest. Do not suggest incomplete code which requires users to modify. You can perform these actions:*
⟨Space Exploration Tools⟩
⟨Solution Evaluation Tools⟩
⟨Enhancing Tools⟩
Output: A Python function that uses the tools to compose solution exploration, evaluation, and execution.

Figure 17: CAPTCHA solving prompt.

A.5 Example

```python
def solve(frames: list[Frame]):
    # Get the slider handle from Frame 2
    slider_handle = frames[2].get_interactable(0)

    # Slide the handle to the right while observing changes in Frame 1
    observations = slide_x(slider_handle, direction="right", observe_frame=frames[1])

    # Extract images from observations
    images = [choice.image for choice in observations]

    # Rank the images based on the task objective: completing the puzzle
    ranked_ids = rank(images, task_objective="Complete the image puzzle")

    # Select the best choice based on the highest rank
    best_choice = observations[ranked_ids[0]]

    # Refine the search around the best choice
    refined_choices = best_choice.refine()

    # Further rank the refined choices
    refined_images = [choice.image for choice in refined_choices]
    refined_ranked_ids = rank(refined_images, task_objective="Complete the image puzzle")

    # Select the best refined choice
    final_choice = refined_choices[refined_ranked_ids[0]]

    # Release the slider at the best position
    final_choice.release()
```

Figure 18: Solution for `yandex/kaleidoscope` CAPTCHA generated by Halligan, which involves dragging a slider to rearrange a grid of tiles and complete the puzzle. The CAPTCHA is visualized in Figure 2 and Figure 5.