No DOM, No Cry: How Grounded VLMs Outperform Specialized CAPTCHA Classifiers

Oguzhan Salman
Istanbul Technical University
salmano21@itu.edu.tr

Kemal Bicakci
Istanbul Technical University
kemalbicakci@itu.edu.tr

***Abstract*—While YOLO-based models achieve high accuracy with high speed in solving CAPTCHA puzzles, their dependance on DOM access to locate UI elements on the screen prevents integration with fully visual GUI agents that operates with screenshots. We introduce a DOM-free, screenshot-only architecture that combines YOLO's fast pattern recognition with advanced reasoning of VLM-based GUI agents. Our system employs a fine-tuned YOLOv8 detector for UI element localization and then dynamically routing extracted UI elements to either specialized YOLO models or a Qwen-7B Vision-Language Model (VLM). Our experiments reveal fundamentally different inductive biases between YOLO and VLM approaches such that YOLO excels at detecting structured patterns and achieves high recall on classes defined by distinctive visual patterns. In contrast, VLM exhibits concrete object bias, often prioritizing discrete entities (even secondary, occluded or hallucinated ones) over dominant surface patterns but it demonstrates superior zero-shot reasoning capabilities. VLM correctly identifies previously unsupported categories by fine-tuned YOLO models and distinguishes structurally similar objects that confuse YOLO models. Combining these strengths of both models, our hybrid strategy achieves 86.85% macro-averaged recall (+24.2%) and 89.30% overall accuracy (+1.6%) without fine-tuning VLM on CAPTCHA images. Furthermore, we find that YOLOv8x-mask outperforms GUI agents on segmentation tasks (F1: 0.818 vs. 0.768) by producing pixel level masks which current GUI agents lack. These findings demonstrate that augmenting VLMs with specialized detectors allows agents to take advantage of high-speed accuracy of discriminative models without sacrificing its semantic reasoning generalization capabilities, suggesting that CAPTCHA is no longer a viable defense against automation in the emerging era of visually grounded AI agents.**

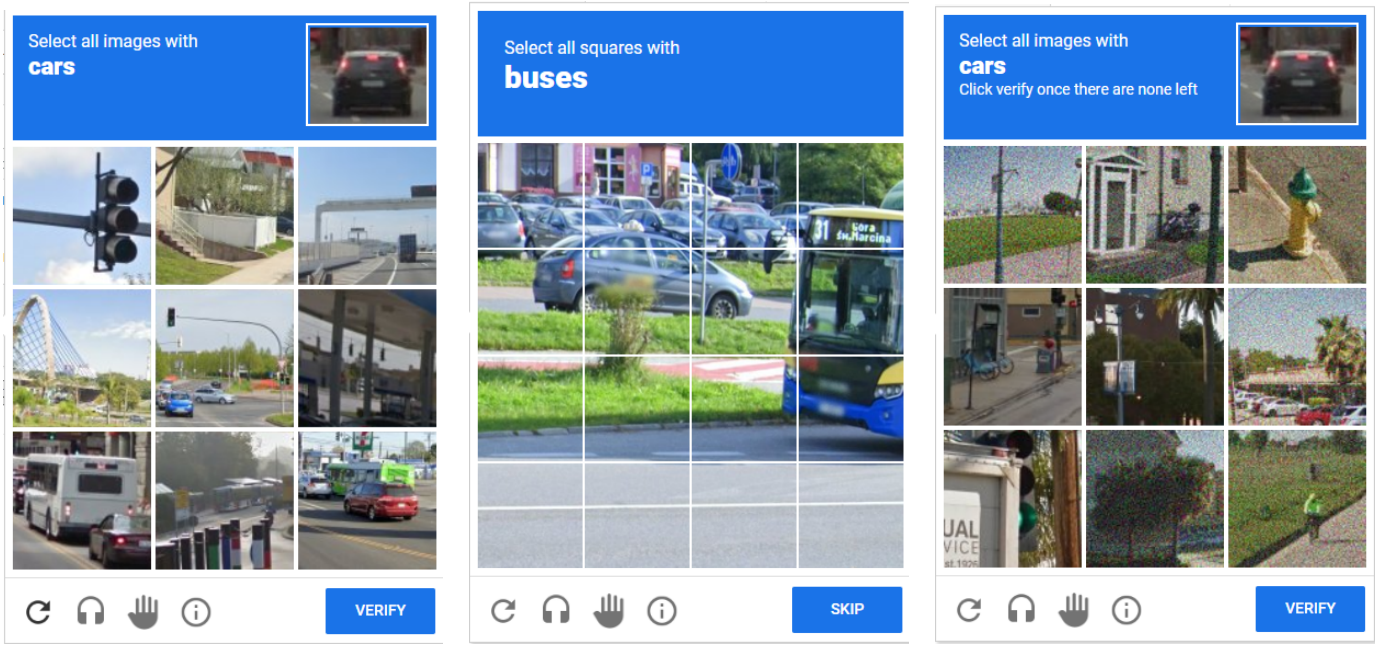
***Index Terms*—CAPTCHA, GUI Agents Vision-Language Models, Generative AI, Object Detection, Web Security**

I. Introduction

CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) [1] remain a primary defense against automated bot activity on the web. Among modern image-based systems, Google's reCAPTCHA v2 [2] is still widely deployed despite the introduction of the implicit behavioral v3 variant of reCAPTCHA [3]. Sites continue to prefer v2 due to its explicit user verification and ease of integration into their products where visible challenge–response mechanisms are desirable. As illustrated in Figure 1, reCAPTCHA v2 offers three visual challenges: (1) classification puzzles where users select 3×3 tiles that contains the target object and each tiles are treated as binary decision; (2) segmentation puzzles where users select corresponding squares on 4×4 grid that contain any part of the target object; and (3) dynamic puzzles where selected tiles are refreshed with new content that requires iterative solving until no target objects remain. These challenges are intended to exploit the gap between human visual perception and current machine-vision capabilities to distinguish legitimate users from automation. However, the arms race between CAPTCHA systems and automated solvers has pushed progress in computer vision, which has led much more sophisticated attack and defense mechanisms.

Early automated solvers relied on traditional optical character recognition techniques for text-based CAPTCHAs [4], [5]. As the CAPTCHA ecosystem shifted toward image-based from text-based challenges, deep learning approaches emerged, initially using convolutional neural networks trained on specific object categories and later YOLO-family detectors [6] that can achieve high accuracy on trained classes much faster than earlier deep learning approaches. Recent work shows that specialized YOLO models can solve reCAPTCHA effectively and these systems use browser automation frameworks such as Selenium, Playwright, Puppeteer, or Cypress to access the Document Object Model (DOM) to locate and interact with UI elements [7], [8], [9], [10].

Concurrently, vision-language models (VLMs) have emerged as powerful general-purpose systems capable of understanding not only predefined tasks but also reasoning over diverse tasks that are not specifically trained to do. Building on the transformer architecture [11] that revolutionized sequence modeling through self-attention mechanisms, large-scale language models [12] demonstrated zero-shot task completion. Today, multimodal models like GPT-4V, Claude 3 and Qwen-VL drive GUI agents such as OpenAI Operator, ChatGPT Atlas, Anthropic Computer Use, WebVoyager and Perplexity Comet [13], [14], [15], [16], [17] that can interact with computers in diverse tasks through user commands. Unlike specialized YOLO models, these VLM systems are capable of understanding and navigating interfaces, interpret screenshots and execute complex workflows without task specific training, which makes them primary candidate for automation. Despite this potential, recent benchmarks identify CAPTCHAs as a primary obstacle that frequently blocks these agents from completing end-to-end workflows [18]. These challenges persist mainly because VLMs face fundamental difficulties with visual grounding and spatial localization without careful prompt engineering [19]. Recent work has shown that prompting strate-



(a) Classification: Select entire tiles containing cars (3×3 grid, independent binary decisions per cell)

(b) Segmentation: Click squares overlapping buses (4×4 grid boundary localization task)

(c) Dynamic: Iteratively select cars until none remain (tiles refresh after each click, requiring multi-round)

Fig. 1: Three types of reCAPTCHA visual challenges. Classification puzzles (a) present distinct images in a 3×3 grid where users select all tiles containing the target object class. Segmentation puzzles (b) present a 4×4 grid image requiring users to identify which squares intersect the target object. Dynamic puzzles (c) refresh selected tiles with new images after each click, creating a sequential decision process that continues until the user verifies no target objects remain. Our system handles all three puzzle types through a unified detection + solving architecture.

gies significantly influence model performance on reasoning tasks [20] and visual interactions similarly require grounding strategies. Without grounding techniques that supply localized crops and structured annotations, LLMs still struggle to identify and interact with small UI elements [21]. To bridge this gap, recent work demonstrates that equipping VLMs with explicit grounding methods, such as Set-of-Mark prompting, enables strong performance in visual interaction tasks, including CAPTCHA solving [22].

From the current landscape of automated solvers, we identify two primary barriers for robust GUI agents. First, specialized classifiers rely heavily on Selenium-based DOM access for element localization, which prevents integration with GUI agents. Second, purely generative VLM approaches underperform not because of limited model capability, but because they lack the visual grounding and structured prompting necessary for reliable classification.

In this work, we address these limitations by coupling fast YOLO-based UI localization with a hybrid classification engine that integrates specialized YOLO discriminators and VLM-based semantic reasoning to achieve higher performance than any single model in isolation. To enable this, we develop a fine-tuned YOLOv8 detector specialized for CAPTCHA UI elements providing accurate localization of the captcha area, grid cells, verify and reload buttons. To evaluate how contemporary GUI agents perform on CAPTCHA tasks, we use an open-weight Qwen-7B-VL model [23] fine-tuned for web interactions [24], demonstrating that such models can effectively handle CAPTCHA challenges when properly grounded. While VLM does not consistently surpass YOLO, it achieves competitive recall on common objects and significantly higher recall on rare or semantically complex categories that YOLO fails to accurately recognize. By integrating these components, our architecture leverages the best of both worlds: it harnesses the real-time classification speed of YOLO for common tasks while utilizing the advanced semantic reasoning of VLM for edge cases. A finite-state controller coordinates this detection, solving, verification and recovery loop, enabling fully visual, DOM-free CAPTCHA solution.

II. Related Work

Early automated CAPTCHA solvers focused on text-based challenges using OCR and segmentation techniques [25], [5], [4], which ultimately led to the development of image-based systems like reCAPTCHA v2. Deep learning approaches demonstrated high accuracy on text-based CAPTCHAs: Noury and Rezaei [26] achieved 98.94% and 98.31% accuracy on numerical and alphanumerical CAPTCHAs respectively using a CNN architecture with parallel Softmax layers trained on 500,000 generated samples. Sivakorn *et al.* [27] demonstrated

large-scale automated attack on reCAPTCHA v2, achieving 70.78% success rate on image challenges using deep learning-based image annotation services and achieving 83.5% on Facebook's image CAPTCHA. Their work revealed vulnerabilities in the risk analysis system and motivated improvements to reCAPTCHA's defenses. Subsequent work targeted image-based reCAPTCHA systems using deep convolutional networks trained on specific object categories. Hossen *et al.* [28] demonstrated that YOLOv3-based object detection can solve Google's image reCAPTCHA v2 with 83.25% success rate, averaging 19.93 seconds per CAPTCHA. Their approach used a bounding box to grid mapping algorithm to convert YOLOv3 detections into cell selections for 3×3 classification puzzles. While effective for COCO-trained object classes (bicycles, buses, cars, traffic lights), their system required puppeteer-firefox for DOM access to extract grid cell coordinates and trigger click events. Weng *et al.* [29] demonstrated systematic attacks across three image CAPTCHA types (selection-based, slide-based, click-based) using CNNs and Fast-RCNN, achieving 79–90% success on selection-based puzzles similar to reCAPTCHA. Plesner *et al.* [30] fine-tuned YOLOv8 on 14,000 reCAPTCHA images across 13 object classes, achieving near-perfect solving rates on image classification puzzles. However, Plesner's system also relies on Selenium for browser automation and remains limited to trained object classes, achieving 0% coverage on unseen categories.

These prior CAPTCHA solvers share a critical limitation: reliance on browser automation frameworks (Selenium, Playwright) to access the Document Object Model (DOM) for element localization and interaction. Recent comparative studies of browser automation tools highlight well-documented limitations of DOM-dependent approaches [31]. Despite their effectiveness in controlled environments, these approaches fail when DOM access is restricted, require browser-specific drivers and remain vulnerable to anti-bot countermeasures such as automation signatures (e.g., WebDriver detection, browser fingerprinting).

YOLO-family detectors have been extensively used for UI element detection in mobile and web interfaces, achieving reliable localization of buttons, icons, and other general components. OmniParser [21] further demonstrated that pretrained YOLO detectors can accurately localize even small UI icons, but it did not consider CAPTCHA layouts or puzzle-specific grid structures. In parallel, multimodal vision–language models such as CLIP [32], Flamingo [33], LLaVA [34], and InstructBLIP [35] have greatly improved high-level screen understanding and visual reasoning. Yet prior evaluations of VLM-based GUI agents [18] (e.g., GPT-4V, Qwen-VL, Claude 3) consistently report poor performance on CAPTCHAs, reflecting limited fine-grained grounding and a lack of structured output mechanisms needed for grid-style selection tasks. Together, these results suggest that neither generic UI detectors nor current VLM agents alone directly address CAPTCHA-specific perception and solution.

Most recently, Teoh *et al.* [22] introduced Halligan, a VLM-based CAPTCHA solver that achieves 60.7% success rate across 26 CAPTCHA types including reCAPTCHA v2, hCaptcha and custom enterprise challenges. Halligan employs a search-based solving strategy using GPT-4 Vision [36] with CAPTCHA abstraction layers and hybrid YOLO + VLM routing similar to our approach, though their work did not systematically compare VLM performance against specialized YOLO classifiers. Similarly, Deng *et al.* [37] demonstrated VLM-based solving of reasoning CAPTCHAs using task decomposition with GPT-4V and Gemini. Complementary to these browser-based agents, Zhang *et al.* [38] proposed UFO, a Windows UI-automation agent that interacts with applications via the Windows UI Automation API instead of Selenium. Although it removes the dependency on a web browser, UFO still assumes programmatic accessibility hooks and thus does not operate in a purely screenshot-based setting.

III. Method

In this section, we present our hybrid architecture designed to solve visual CAPTCHA challenges without DOM access. The proposed system consists of a fine-tuned object detector for UI localization, a deterministic finite-state machine for process control and a dual-path solving mechanism that leverages the complementary strengths of specialized YOLO classifiers and Vision-Language Models.

A. Detection Backbone: YOLOv8

We fine-tune YOLOv8 from OmniParser's [21] pretrained icon detection model [39] to detect five CAPTCHA-specific classes: captcha area, cell, robot checkbox, reload button, submit button. OmniParser provides strong foundation model for small UI element localization, trained on diverse webpage screenshots. The model processes 512×512 RGB images and we train with AdamW optimizer, multi-scale training and early stopping following established fine-tuning practices.

B. Finite-State Controller

We coordinate detection, solving and verification through a deterministic finite-state machine $\mathcal{M} = (S, s_0, \Sigma, \delta)$ with states S covering initialization, puzzle analysis, solving, verification and recovery. Transitions $\delta : S \times \Sigma \to S$ are triggered by visual events (checkbox detected, grid appeared, CAPTCHA cleared). The FSM handles both static and dynamic puzzles where selected tiles refresh with new images through re-analysis and verification. Algorithm 1 presents the complete solving procedure integrating FSM control with hybrid YOLO/VLM solving.

C. Classification and Segmentation Pipelines

After extracting all CAPTCHA-related UI elements using our YOLOv8 detector, we use Tesseract OCR to detect the instruction text within the CAPTCHA area and extract the target class. With the target identified, individual cells cropped from the detected captcha area and images are sent either to a YOLO classifier/segmentation [40] or to fine-tuned Qwen-7B-VL GUI agent [41]. For VLM inference, we use the following prompt template:

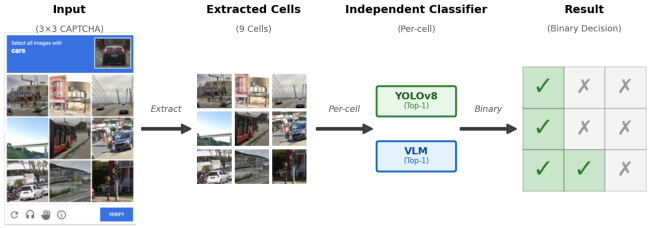


Fig. 2: Classification puzzle solving pipeline for 3×3 CAPTCHAs. The system extracts 9 individual cells from the detected CAPTCHA area, classifies each cell independently using either YOLOv8 (top-1) or VLM (top-1) and produces binary decisions indicating which cells contain the target object (cars in this example). Both models operate in top-1 prediction, committing to exactly one class label per cell.

```
Choose ONLY ONE number from the following list:
0: Bicycle, 1: Bridge, 2: Bus, 3: Car, 4: Chimney,
5: Crosswalk, 6: Hydrant, 7: Motorcycle, 8: Mountain,
9: Other, 10: Palm, 11: Stairs, 12: Traffic Light,
13: Boat, 14: Taxi, 15: Tractor

Answer with ONLY the number (0-15) of the class
you see with the highest confidence.
Do not include any explanation.
```

This indexed format enforces deterministic top-1 predictions and minimizes hallucinations. Figure 2 illustrates the complete classification pipeline: each 9 cells are extracted from 3×3 grid and independently classified by either YOLO or VLM to produce a top-1 prediction.

For segmentation puzzles, 4×4 detected cells are concatenated into an image and sent to the models and we retrieve bounding boxes/masks from YOLO or region descriptions from VLM for the target object. For VLM inference, we use the following structured prompt template that requests bounding box of all target objects:

```
Find all {object_name} in this image and create
bounding boxes that encompass the entire {object_name}.

Requirements:
- Include ALL visible parts of the {object_name}
- Capture the complete {object_name} from top to
  bottom and side to side
- Make sure the entire {object_name} is contained
  within the bounding box

For each {object_name}, return coordinates:
{Object_name} 1: [x1, y1, x2, y2]
{Object_name} 2: [x1, y1, x2, y2]

Where (x1, y1) is top-left corner and (x2, y2) is
bottom-right corner.
```

Predictions are mapped back to individual cell coordinates using a 1% overlap threshold: any cells whose bounding box intersects the predicted region by $\geq$1% of its area is selected for clicking. Figure 3 illustrates the complete segmentation pipeline: the 4×4 grid is extracted and concatenated into a single image and then, either YOLO (producing masks or bboxes) or VLM (producing bboxes only) detects all instances of the target objects.

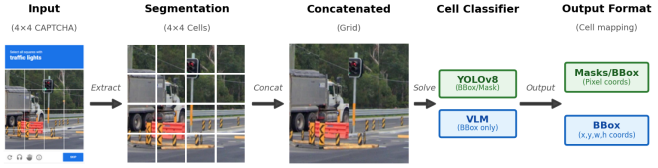


Fig. 3: Segmentation puzzle solving pipeline for 4×4 CAPTCHAs. The system extracts and concatenates 16 cells into a unified image, runs either YOLOv8 segmentation (producing pixel-accurate masks or bounding boxes) or VLM (producing bounding boxes only) and maps detected regions back to cell coordinates. Any cell overlapping the predicted region by $\geq$1% is selected.

Algorithm 1 Hybrid CAPTCHA Solver

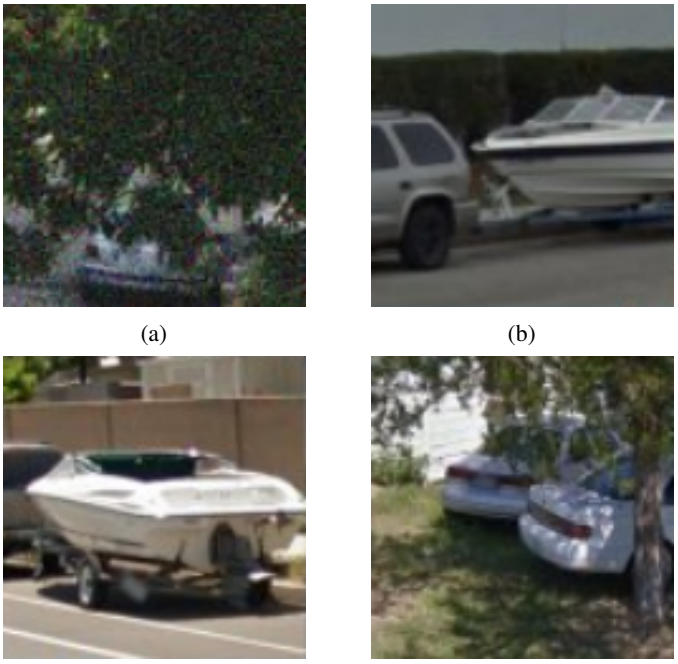
Require: screenshot I
$B \leftarrow$ detect_UI_elements(I) ▷ YOLOv8n
$I_{grid} \leftarrow$ extract_captcha_area(I, B)
$t, type \leftarrow$ OCR(I_{grid}) ▷ Target & type
$R \leftarrow$ detect_cells(I_{grid})
if $type =$ `classification` **then**
  for all $r \in R$ **do**
    $y_r \leftarrow$ YOLO_classify(r, t) or VLM_classify(r, t)
  end for
  $C \leftarrow \{r : y_r = t\}$
else ▷ Segmentation
  $I_{merged} \leftarrow$ concatenate(R)
  $M \leftarrow$ YOLO_segment(I_{merged}, t) or VLM_segment(I_{merged}, t)
  $C \leftarrow$ map_to_cells(M, R)
end if
click(C), wait(200 ms)
click(verify_button)
return check_success()

IV. Data and Metrics

To accurately measure the performance of our hybrid solver, we constructed a comprehensive dataset reflecting real-world class distributions that contain challenging edge cases. This section details the composition of our dataset and defines the specific evaluation metrics used to measure performance across both classification and segmentation puzzle types.

A. Dataset Composition

Our dataset comprises 15.172 samples as total, collected from public reCAPTCHA dataset [42] and CAPTCHA samples obtained during our experiments. The dataset covers 16 object classes: Bicycle, Boat, Bridge, Bus, Car, Chimney, Crosswalk, Hydrant, Motorcycle, Mountain, Other, Palm, Stairs, Taxi, Tractor and Traffic Light. Among these classes, three of these (Boat, Taxi, Tractor) are not supported by the published YOLO classifier. We encountered these object categories during our



(a) (b) (c) (d)

Fig. 4: All four cells are labeled as Car in the ground truth, but our VLM predicts Boat in each cases. In (b)–(c), a boat on the road is clearly visible. In (a)-(d), the cars are heavily occluded by environment or appear adjacent to water, making a prediction of Boat a semantically plausible interpretation. These examples illustrate how semantically reasonable predictions can still be counted as errors in our evaluation.

experiments and include them in our evaluation to reflect real-world scenarios.

B. Evaluation Metrics

We use different primary metrics for classification and segmentation puzzles based on their task requirements. For 3×3 classification puzzles, we prioritize recall because these types of CAPTCHA puzzles require identifying all tiles that contain target object and missing a required tile increases risk score.

Our evaluation for the classification task differs from live CAPTCHA sessions in that we assess models on a static dataset of puzzle tiles from all 16 classes, requiring the model to discriminate the dominant object across all possible categories. In contrast, real reCAPTCHA presents context-specific puzzles where positive examples contain obvious instances of the target and negative examples contain clearly different objects. This makes our evaluation more challenging than live solving because models must handle images that contain multiple categories such as distinguishing dominant bridges from dominant cars in images where both may be present as occluded, out of context or secondary elements.

In this static evaluation setting, recall reliably measures whether the model identifies the ground-truth dominant object, while precision can be artificially deflated when models detect genuine but secondary or occluded objects in the scene, as illustrated in Figure 4. We report per-class precision, recall and F1-score, along with overall accuracy (frequency-weighted, where frequent classes dominate) and macro-averaged accuracy (where all classes contribute equally).

An alternative design would be to report top-k accuracy or probability-threshold metrics (e.g., treating a prediction as correct if Car appears in the top-3 classes with probability at least 10%). We deliberately refrain from doing so in order to maintain a fair evaluation between YOLO and VLM. The YOLO classifier outputs a full softmax distribution over classes, so top-k and probability thresholds are well defined. The VLM, in contrast, does not expose class probabilities: in our setup, it produces a single class index token (0–15), and when instructed to output "probabilities" it returns unnormalized, inconsistent or even hallucinated values that often do not sum to 100%. These values reflect the model's language-generation behaviour rather than reliable posterior probabilities and there is no principled way to construct a top-k metric for the VLM that is comparable to YOLO's softmax scores. To avoid this asymmetry, we discard the additional confidence information provided by YOLO and evaluate both models under the shared metric which is single top-1 label per tile.

For 4×4 segmentation puzzles, we use F1-score as the primary metric, since boundary localization requires balancing both under-selection (low recall) and over-selection (low precision), making F1-score appropriate for measuring segmentation performances.

V. Experiments

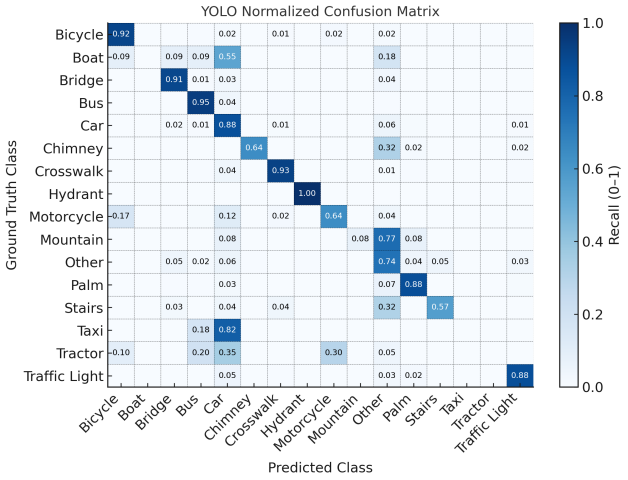
We evaluate discriminative and generative models through a series of experiments designed to highlight the distinct capabilities. We first analyze the performance of classification puzzles, followed by segmentation performances and conclude with detailed latency analysis of each models.

A. Classification Experiments

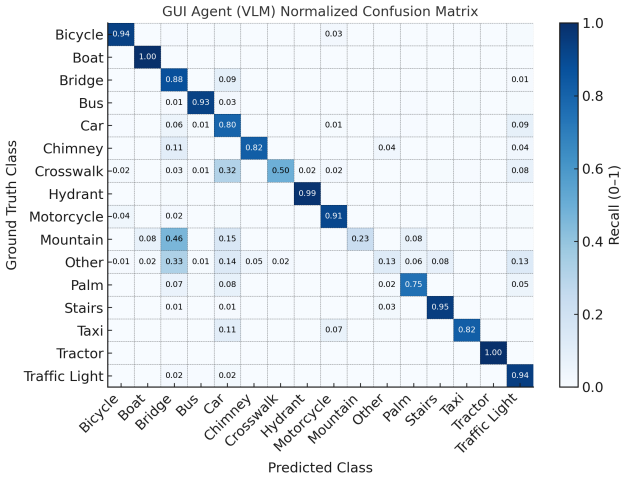
To evaluate the solvers on classification-type puzzles, we compare the published YOLO classifier and our fine-tuned Qwen-7B-VL GUI agent on identical reCAPTCHA challenges. This analysis focuses on cases where each grid cell is to be classified independently.

Following the evaluation protocol described in Section IV-B, we construct a test set comprising 11.908 images spanning 16 object classes and require both models to produce a single top-1 prediction per image. For YOLO, we take the argmax over its softmax scores and for the VLM, we use the numbered 0–15 prompt described in Section III-C, which constrains it to return exactly one class index.

The test set exhibits class imbalance reflective of real-world reCAPTCHA distributions. Frequent classes (e.g., Car: 3,578 samples, Bridge: 1,518 samples) dominate the overall accuracy metric, while rare classes (e.g., Boat: 11 samples, Tractor: 7 samples) are better reflected in macro metric. We report both overall and macro-averaged metrics to provide balanced insight and overall accuracy captures deployment performance, while



(a) YOLO normalized confusion matrix.



(b) GUI Agent (VLM) normalized confusion matrix.

Fig. 5: Comparison of per-class recall for YOLO vs VLM GUI agent. Darker diagonals indicate stronger recall; off-diagonals correspond to confusion.

TABLE I: Per-class Precision, Recall and F1 scores.

Class	YOLO			VLM (GUI Agent)		
	Prec	Rec	F1	Prec	Rec	F1
Bicycle	95.2%	91.9%	93.5%	91.8%	**93.6%**	92.7%
Boat†	0.0%	0.0%	0.0%	20.4%	**100.0%**	33.8%
Bridge	78.4%	**91.1%**	84.3%	36.6%	87.5%	51.6%
Bus	91.9%	**94.5%**	93.2%	92.9%	93.3%	93.1%
Car	90.3%	**87.6%**	88.9%	78.1%	79.7%	78.9%
Chimney	78.3%	64.3%	70.6%	35.4%	**82.1%**	49.5%
Crosswalk	94.4%	**93.0%**	93.7%	94.1%	49.6%	65.0%
Hydrant	99.3%	**99.7%**	99.5%	96.3%	98.7%	97.5%
Motorcycle	67.0%	64.4%	65.7%	48.4%	**91.1%**	63.2%
Mountain	50.0%	7.7%	13.3%	33.3%	**23.1%**	27.3%
Palm	86.8%	**88.3%**	87.6%	86.5%	75.2%	80.5%
Stairs	36.4%	57.3%	44.6%	33.0%	**94.7%**	49.0%
Taxi†	0.0%	0.0%	0.0%	32.4%	**82.1**%	46.5%
Tractor†	0.0%	0.0%	0.0%	44.4%	**100.0**%	61.5%
Traffic Light	86.0%	87.9%	87.0%	52.4%	**94.3**%	67.4%
Other	72.6%	**74.3**%	73.4%	78.2%	12.7%	21.9%
Macro Avg.	64.1%	62.6%	61.8%	60.1%	78.6%	61.4%

†Unsupported by YOLO Classification Model

macro accuracy treats all object types equally, preventing high-frequency classes from masking failures on rare categories.

a) YOLO Classifier Performance: We first evaluate the published YOLO classifier [30]. As shown in Table II, the model achieves an overall accuracy of 0.8773 but a macro accuracy of only 0.6262. Looking at the per-class performances in Table I, YOLO achieves high recall on common classes such as *Car* (87.6%), *Bus* (94.5%) and texture-heavy classes like *Crosswalk* (93.0%). However, it suffers significantly on unsupported classes (0% recall) and structurally similar objects like *Motorcycles* (64.4% recall), which are frequently misclassified as Bicycles in the normalized confusion matrix (Figure 5a).

TABLE II: Classification accuracy comparison on reCAPTCHA object categories.

Metric	YOLO	VLM (GUI Agent)	Hard-Best
Overall Accuracy	0.8773	0.7336	0.8930
Macro Accuracy	0.6262	0.7862	0.8685

b) VLM GUI Agent Performance: Next, we evaluate our fine-tuned Qwen-7B-VL GUI agent. VLM achieves an overall accuracy of 0.7336 and macro accuracy of 0.7862. Unlike YOLO, VLM demonstrates strong zero-shot generalization, achieving 100% recall on rare classes like Boat and Tractor and high recall on semantically distinct categories like Traffic Light (94.3%) and Stairs (94.7%). However, the VLM underperforms

on specific categories. For example, VLM often ignores the primary object like Crosswalk to focus on secondary objects (classifying them as Bridges or Others or hallucinates specific labels for generic backgrounds).

1) Hybrid Strategy and Per-Class Analysis: Table II summarizes the performance of both models alongside a *Hard-Best* hybrid strategy.

We construct the Hybrid Strategy as the following: Let $\text{Acc}_{\text{YOLO}}(c)$ and $\text{Acc}_{\text{VLM}}(c)$ denote the per-class recall for class c. We define a class-wise routing function as the following:

$$r(c) = \arg \max_{m \in \{\text{YOLO,VLM}\}} \text{Acc}_m(c). \quad (1)$$

At inference time, each ground-truth class c is assigned to the model chosen by $r(c)$ and that model's prediction is used for all samples of that class.

A simple *Hard-Best* hybrid strategy, choosing which model performs better per class, achieves 0.893 overall accuracy and 0.8685 macro accuracy, improving the YOLO baseline by +1.6% overall and +24.2% macro accuracy. The macro accuracy gain is particularly significant, reflecting VLM's ability to handle rare and unsupported classes. Table I presents per-class Precision, Recall and F1 scores.

B. Segmentation Experiments

Segmentation puzzles require selecting all grid cells that contain any part of a target object in a 4×4 reCAPTCHA grid. We evaluate multiple YOLOv8 variants (n/m/x with bounding boxes vs masks) and fine-tuned Qwen-7B-VL GUI agent that outputs bounding boxes and a simple hybrid that uses best performing model for the given target object.

Table III summarizes performance on segmentation puzzles with 3.264 cell decisions from 204 puzzles. YOLOv8x-mask is the strongest single model (87.2% accuracy; F1=0.818). The VLM achieves higher recall (80.5%) but lower precision (73.4%) than YOLO masks. Similarly to classification results, a hybrid of YOLOv8x-mask + VLM provides the best overall results (88.9% accuracy; F1=0.851) by combining YOLO's geometric precision with the VLM's coverage on rare/unseen categories.

TABLE III: Segmentation Results

Model	Accuracy	Precision	Recall	F1-Score
YOLOv8n-BBox	79.0%	80.5%	59.1%	0.682
YOLOv8n-Mask	81.2%	89.9%	57.0%	0.698
YOLOv8m-BBox	83.2%	78.7%	76.7%	0.777
YOLOv8m-Mask	86.3%	88.5%	73.6%	0.804
YOLOv8x-BBox	84.2%	78.8%	80.1%	0.794
YOLOv8x-Mask	87.2%	88.9%	75.7%	0.818
VLM (BBox)	81.4%	73.4%	80.5%	0.768
Hybrid (YOLOv8x-Mask + VLM)	88.9%	87.2%	82.9%	0.851

Table IV presents per-class F1-score performances across YOLO segmentation models and VLM. The results reveal that (1) larger YOLO models outperform smaller ones on supported classes, with YOLOv8x-Mask achieving the highest F1-scores (Bus: 0.915, Motorcycle: 0.892); (2) all YOLO models fail completely (F1=0.000) on non-COCO classes (Crosswalk, Stairs, Taxi) regardless of size; (3) the VLM provides full coverage across all 8 classes, achieving competitive performance without any CAPTCHA specific training with F1=0.857 on Taxi, 0.743 on Stairs and 0.636 on Crosswalk—categories where YOLO has zero capability.

Fig. 6: Mask vs bounding box on a motorcycle puzzle. Bounding boxes include background, causing false positives when mapped to grid cells. Masks follow object boundaries, improving precision by 10% across models.

The macro-averaged metrics favor the VLM due to its zero-shot coverage on unsupported classes. However, when aggregating across all 3,264 cell-level decisions (weighted by puzzle frequency), YOLOv8x-Mask achieves superior overall performance: 88.9% precision, 75.7% recall and F1=0.818, compared to the VLM's 73.4% precision, 80.5% recall and F1=0.768. This demonstrates that while YOLO excels on supported classes, which dominate the dataset, VLM provides coverage for rare and unseen categories.

C. Latency Analysis

Table V presents the latency analysis for different models in different puzzle types. The YOLOv8 detector operates in real-time at 2.7 ms per image (370 FPS), enabling immediate UI localization. A significant performance gap exists between specialized YOLO solvers and VLM. The YOLO-only pipeline maintains real-time performance (≈26 ms for segmentation) because YOLO is architecturally optimized for real-time inference with lightweight size (<100M parameters).

In contrast, VLM exhibits a substantial latency cost (225 ms–2251 ms) driven by two factors:

1) **Model Size:** The Qwen-7B-VL model (7 billion parameters) requires significantly higher compute and memory bandwidth per inference than the YOLO models.

TABLE IV: Segmentation Per-Class F1-Scores

Class	n-BBox	n-Mask	m-BBox	m-Mask	x-BBox	x-Mask	VLM
Bicycle	0.687	0.696	0.722	0.739	0.685	0.715	0.723
Bus	0.866	0.898	0.871	0.902	0.911	0.915	0.813
Crosswalk†	0.000	0.000	0.000	0.000	0.000	0.000	0.636
Fire Hydrant	0.702	0.679	0.933	0.925	0.892	0.906	0.756
Motorcycle	0.670	0.698	0.811	0.859	0.836	0.892	0.791
Stairs†	0.000	0.000	0.000	0.000	0.000	0.000	0.743
Taxi†	0.000	0.000	0.000	0.000	0.000	0.000	0.857
Traffic Light	0.733	0.733	0.876	0.883	0.918	0.898	0.749

†Unsupported by YOLO segmentation models
n=YOLOv8n, m=YOLOv8m, x=YOLOv8x; BBox=bounding box, Mask=segmentation mask.

TABLE V: Component Latency (median; ms per inference)

Component	Latency (ms)	FPS
YOLOv8 Detector	2.7	370
YOLO Classifier (per cell)	4.4	220
YOLOv8n-seg (per grid)	23	43.5
YOLOv8x-seg (per grid)	26	38.2
VLM Classification (per cell)	225	4.1
VLM Segmentation (per grid)	2251	0.4

2) **Autoregressive Generation:** Unlike YOLO's near instantaneous inference, VLM must generate text tokens sequentially. This cost is particularly evident in the segmentation task (2251 ms), where VLM must generate a long sequence of coordinate tokens for every detected object, whereas classification requires generating only a single class index token (225 ms).

Consequently, our hybrid approach trades this latency for semantic robustness only when necessary, defaulting to the real-time YOLO for other categories.

VI. Discussion

In this section, we discuss the distinction between discriminative pattern-matching and open-world knowledge, analyze the issue of semantic over-interpretation and examine the role of context and model capacity.

A. Pattern-Matching vs. Open-World Knowledge

Our results show that these models demonstrate fundamentally different recognition strategies such that YOLO relies on learned silhouettes, while the VLM leverages semantic reasoning from internet-scale pre-training. This difference is evident on visually similar classes. For instance, YOLO frequently confuses Motorcycles with Bicycles based on structural similarity, whereas the VLM distinguishes them through semantic attributes—engine blocks, exhaust pipes—that discriminative training does not explicitly encode, achieving higher recall on both classes. This reasoning capability extends naturally to novel categories without retraining.

B. Semantic Over-Interpretation and Object Bias

The performances on Other, Crosswalk and Mountain reveals a fundamental difference. YOLO acts as a texture-sensitive discriminator, achieving high recall on Crosswalks (93.0%) and correctly identifying the negative class Other (74.3%).

In contrast, the VLM exhibits two distinct failures:

1) **Object-Centric Bias:** The VLM demonstrates a strong preference for discrete objects over surface features. It often prioritizes potentially distinct entities (even if secondary) and treats dominant texture patterns like Crosswalks (49.6% recall) and Mountains (23.1% recall) as contextual background, leading to false negatives.
2) **Semantic Over-Interpretation:** VLM struggles significantly with the Other class (12.7% recall), frequently hallucinating specific labels for generic scenes. Unlike YOLO, which learns a decision boundary for "none of the above," VLM's pre-training forces it to converge to find meaningful concepts in every image. Consequently, it often over-interprets generic roads as Bridges or Crosswalks and background as Palms, unable to accept that an image contains "nothing of interest".

C. The Role of Context and Model Capacity

A significant performance divergence exists between the classification and segmentation tasks for irregular objects like Motorcycles. In classification, the model analyzes 3×3 tiles, leading to a high confusion rate with Bicycles (17%). In segmentation, performance improves significantly (F1=0.892). This improvement is driven by two factors: (1) **Higher Resolution**, where the merged 4×4 grid provides high resolution environmental details (road, rider, scale), which is absent in significantly smaller classification tiles; and (2) **Model Capacity**, as our segmentation pipeline employs larger YOLOv8x backbone compared to the light-weight classification model. The combination of seeing the "whole picture" and having the parameter capacity to model complex boundaries enables the segmentation model to resolve the inherent similarities that confuse smaller classifiers.

VII. Ethics and Responsible Use

This research is conducted for academic purposes to advance understanding of vision-language model capabilities and GUI automation systems. We provide our code upon request to advance GUI automation research and enable reproducibility. Since our work builds upon already published CAPTCHA-solving models [30], we continue this research direction to benefit the research community studying vision-language models and visual automation systems.

As with any security research, our techniques could potentially be misused for unauthorized automation. We do not

endorse such applications and emphasize that circumventing security measures without authorization is unethical and often illegal. However, advancing scientific understanding of AI capabilities requires studying both strengths and limitations of deployed systems.

VIII. Conclusion

This work shows that DOM access—a primary detectability vector for automation—is not strictly necessary for solving reCAPTCHA v2. By replacing DOM-based element localization with visual detection, we obtain a robust, screenshot-only pipeline that reliably defeats reCAPTCHA while remaining fully compatible with GUI agents.

More broadly, our results point to a concrete design pattern for next-generation autonomous GUI agents. Specialized YOLO detectors handle most common visual decisions, providing fast and accurate predictions on supported classes, while VLM is used only when semantically similar or out-of-distribution content demands open-world reasoning. The substantial latency gap between YOLO and VLM solvers (26 ms vs. 2250 ms) makes purely generative agents currently impractical for latency-sensitive CAPTCHA solving, yet our hybrid routing strategy demonstrates that this limitation can be substantially mitigated. Combining the speed of discriminative models with targeted generative reasoning yields an architecture that is both practical and highly effective.

In this study, we instantiate this methodology on widely deployed reCAPTCHA v2 and show that it is sufficient to surpass state-of-the-art YOLO-only solvers by combining the strengths of VLM agents. While our concrete components (detectors, prompts, and finite-state controller) are designed for reCAPTCHA v2, the underlying logic can be adapted to other visual CAPTCHA families. From a security perspective, our findings indicate that image-based reCAPTCHA can no longer be regarded as a reliable defense against automation given the effectiveness of hybrid, DOM-free approaches.

References

[1] L. Ahn, M. Blum, N. Hopper, and J. Langford, "Captcha: using hard ai problems for security," vol. 2656, 05 2003, pp. 294–311.

[2] Google, "reCAPTCHA v2 Documentation," https://developers.google.com/recaptcha/docs/display, 2024, accessed: 2025-01-10.

[3] ——, "reCAPTCHA v3 Documentation," https://developers.google.com/recaptcha/docs/v3, 2024, accessed: 2025-01-10.

[4] E. Bursztein, J. Aigrain, A. Moscicki, and J. Mitchell, "The end is nigh: Generic solving of text-based captchas," 08 2014.

[5] J. Yan and A. Ahmad, "A low-cost attack on a microsoft captcha," 10 2008, pp. 543–554.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016. [Online]. Available: https://arxiv.org/abs/1506.02640

[7] https://www.selenium.dev/, accessed: 2025-01-10.

[8] https://playwright.dev/, accessed: 2025-01-10.

[9] https://pptr.dev/, accessed: 2025-01-10.

[10] https://www.cypress.io/, accessed: 2025-01-10.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[13] OpenAI, "Introducing operator," https://openai.com/index/introducing-operator/, 2025, accessed: 2025-01-23.

[14] ——, "Introducing chatgpt atlas," https://openai.com/index/introducing-chatgpt-atlas/, 2025, accessed: 2025-10-21.

[15] Anthropic, "Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku," https://www.anthropic.com/news/3-5-models-and-computer-use, 2024, accessed: 2024-10-22.

[16] H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu, "Webvoyager: Building an end-to-end web agent with large multimodal models," 2024. [Online]. Available: https://arxiv.org/abs/2401.13919

[17] Perplexity AI, "Introducing comet: Browse at the speed of thought," https://www.perplexity.ai/hub/blog/introducing-comet, 2025, accessed: 2025-07-09.

[18] Y. Luo, Z. Li, J. Liu, J. Cui, X. Zhao, and Z. Shen, "Open captchaworld: A comprehensive web-based platform for testing and benchmarking multimodal llm agents," 2025. [Online]. Available: https://arxiv.org/abs/2505.24878

[19] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao, "Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v," 2023. [Online]. Available: https://arxiv.org/abs/2310.11441

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[21] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "Omniparser for pure vision based gui agent," 2024. [Online]. Available: https://arxiv.org/abs/2408.00203

[22] X. Teoh, Y. Lin, S. Li, R. Liu, A. Sollomoni, Y. Harel, and J. S. Dong, "Are captchas still bot-hard? generalized visual CAPTCHA solving with agentic vision language model," in *Proceedings of the 34th USENIX Security Symposium*. USENIX Association, 2025. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/teoh

[23] J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou, "Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond," 2023. [Online]. Available: https://arxiv.org/abs/2308.12966

[24] M. Andreux, B. B. Skuk, H. Benchekroun, E. Biré, A. Bonnet, R. Bordie, N. Bout, M. Brunel, P.-L. Cedoz, A. Chassang, M. Chen, A. D. Constantinou, A. d'Andigné, H. de La Jonquière, A. Delfosse, L. Denoyer, A. Deprez, A. Derupti, M. Eickenberg, M. Federico, C. Kantor, X. Koegler, Y. Labbé, M. C. H. Lee, E. L. J. de Kergaradec, A. Mahla, A. Manevich, A. Maret, C. Masson, R. Maurin, A. Mena, P. Modard, A. Moyal, A. N. Kerbel, J. Revelle, M. L. Richter, M. Santos, L. Sifre, M. Theillard, M. Thibault, L. Thiry, L. Tronchon, N. Usunier, and T. Wu, "Surfer-h meets holo1: Cost-efficient web agent powered by open weights," 2025. [Online]. Available: https://arxiv.org/abs/2506.02865

[25] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual captcha," 05 2003.

[26] Z. Noury and M. Rezaei, "Deep-captcha: a deep learning based captcha solver for vulnerability assessment," 2020. [Online]. Available: https://arxiv.org/abs/2006.08296

[27] S. Sivakorn, J. Polakis, and A. D. Keromytis, "I'm not a human: Breaking the google recaptcha," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:26151412

[28] M. I. Hossen, Y. Tu, M. F. Rabby, M. N. Islam, H. Cao, and X. Hei, "An object detection based solver for google's image recaptcha v2," 2021. [Online]. Available: https://arxiv.org/abs/2104.03366

[29] H. Weng, B. Zhao, S. Ji, J. Chen, T. Wang, Q. He, and R. Beyah, "Towards understanding the security of modern image captchas and underground captcha-solving services," *Big Data Mining and Analytics*, vol. 2, pp. 118–144, 06 2019.

[30] A. Plesner, T. Vontobel, and R. Wattenhofer, "Breaking recaptchav2," in *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, Jul. 2024, p. 1047–1056. [Online]. Available: http://dx.doi.org/10.1109/COMPSAC61105.2024.00142

[31] B. García, J. Del Alamo, M. Leotta, and F. Ricca, *Exploring Browser Automation: A Comparative Study of Selenium, Cypress, Puppeteer, and Playwright*, 09 2024, pp. 142–149.

[32] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[33] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, "Flamingo: a visual language model for few-shot learning," 2022. [Online]. Available: https://arxiv.org/abs/2204.14198

[34] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2304.08485

[35] W. Dai, J. Li, D. Li, A. M. H. Tiong, J. Zhao, W. Wang, B. Li, P. Fung, and S. Hoi, "Instructblip: Towards general-purpose vision-language models with instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2305.06500

[36] "Gpt-4v(ision) system card," 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263218031

[37] G. Deng, H. Ou, Y. Liu, J. Zhang, T. Zhang, and Y. Liu, "Oedipus: Llm-enchanced reasoning captcha solver," 2024. [Online]. Available: https://arxiv.org/abs/2405.07496

[38] C. Zhang, L. Li, S. He, X. Zhang, B. Qiao, S. Qin, M. Ma, Y. Kang, Q. Lin, S. Rajmohan, D. Zhang, and Q. Zhang, "Ufo: A ui-focused agent for windows os interaction," 2024. [Online]. Available: https://arxiv.org/abs/2402.07939

[39] https://github.com/microsoft/OmniParser, accessed: 2025-01-10.

[40] https://github.com/aplesner/Breaking-reCAPTCHAv2, accessed: 2025-01-10.

[41] https://huggingface.co/blog/Hcompany/holo1, accessed: 2025-11-14.

[42] https://github.com/ajmandourah/recaptcha-dataset, accessed: 2025-11-14.