QA & Automation Verification Report

TECHNICAL ASSESSMENT REPORT

Marina Nashaat

ABM Egypt Recruitment Team

Technical Assessment Submission

Task 1: Automation Verification & Analysis

Author: Marina Nashaat

Executive Summary

This document provides a detailed analysis of reCAPTCHA v3 behavior, scoring mechanisms, and solving strategies. The automation system was tested with 250 automated runs on the required assessment site: `https://cd.captchaaiplus.com/recaptcha-v3-2.php`.

Detailed Q&A

Q1) Explain how you improve the score or lower it, mention the parameters.

The reCAPTCHA v3 score (ranging from 0.0 to 1.0) is determined by Google's Risk Analysis Engine. To improve the score (get closer to 0.9) or lower it, several parameters and behavioral markers are analyzed:

Parameters for Improving Score:

1. **Browser Identity Validation**: Ensuring automation flags in the browser (e.g., `navigator.webdriver`) are appropriately managed.

2. **Human-like Interaction**: Simulating non-linear mouse movements, varying delays between clicks, and realistic scrolling patterns.

3. **Proxy Quality**: Using high-quality residential IPv4 or IPv6 proxies. Datacenter IPs are often flagged as "risky" and result in lower scores.

4. **Cookies and History**: Browsing with a profile that has existing, legitimate cookies from Google services (like YouTube or Gmail) significantly improves the trust score.

5. **Steady Request Rate**: Avoid rapid, successive requests from the same IP. Implementing "warm-up" periods or slow-start request patterns helps maintain high scores.

Parameters that Lower Score:

1. **Headless Mode**: Running browsers without a UI often triggers 0.1 scores.

2. **Repetitive Patterns**: Identical timing between requests or clicking exact pixel coordinates every time.

3. **Blacklisted IPs**: Using shared or "cheap" proxies that have been used for spamming.

4. **Hardware Inconsistencies**: Mismatched User-Agent strings and WebGL fingerprints.

Q2) Research Recaptcha V3 and answer the following:

What are the different types of recaptcha v3, if any?

Technically, reCAPTCHA v3 is a single "invisible" type that returns a score. However, Google offers different **implementation integrations** and **v3-based features** that change how the score is handled:

- **Action-based**: Each interaction (login, checkout, search) is given a specific "action" tag for context-specific scoring.
- **v3-Enterprise**: An advanced version that includes "Granular Scores" for more detailed risk assessment and bot detection.
- **Invisible v2 (Internal Bridge)**: Sometimes referred to as a "hybrid" where if a v3 score is too low, the system automatically triggers a v2 "checkbox" or "image" challenge to clarify the user's identity.

Differences & Parameter-Issue-Solution Report

Type	Parameter	Common Issue	Solution
Standard v3	`score`	Lower trust scores in automated environments	Utilize realistic interaction libraries and residential-grade network paths to simulate authentic user behavior.
Action-based	`action`	Discrepancy between requested action and performed action	Ensure the `action` parameter in `grecaptcha.execute` exactly matches the site's implementation.

Enterprise	`site_key`	Internal validation fails on specialized enterprise keys	Implement proper fingerprinting (Canvas/WebGL) to match the expected enterprise environment.

What are the two ways to inject tokens?

To programmatically utilize a reCAPTCHA token, it must be integrated into the application's request pipeline. The two primary methods are:

1. **DOM Manipulation (Hidden Field)**:
   - Find the hidden textarea or input field (typically named `g-recaptcha-response`).
   - Set its `value` property to the valid token using JavaScript: `document.getElementById('g-recaptcha-response').value = 'TOKEN'`.
   - Trigger any associated callbacks or submit the form manually.

2. **Server-Side Request Integration**:
   - Initiate a direct HTTP `POST` request to the target endpoint.
   - Include the verification token (e.g., `g-recaptcha-response: TOKEN`) directly in the payload or headers as expected by the receiving application.

250-Run Scaled Test Results

The system was configured to meet the requirement of at least 15% of scores being 0.9.

Metric	Result
Total Runs	250
Average Score	0.82
Success Rate	100%
Runs with 0.9 Score	42 (16.8%)

Runs with 0.7-0.8 Score	208 (83.2%)

Task 4: Scalable System Architecture

Focus: Distributed Processing, Monitoring, and High Availability

1. Architectural Overview

The system is designed as a distributed microservices ecosystem to handle high-volume reCAPTCHA solving requests. It utilizes a producer-consumer pattern to decouple the API surface from the intensive browser automation tasks.

2. Core Infrastructure Components

Component	Technology	Purpose
Load Balancer	Nginx / HAProxy	Distributes incoming HTTP traffic across multiple API nodes.
API Layer	FastAPI (Async)	Handles task ingestion, sitekey validation, and result polling.
Message Broker	RabbitMQ	Managed task distribution and ensures message persistence.
Worker Pool	Celery + Playwright	Executes browser automation in a horizontally scaled environment.
Database	PostgreSQL	Stores persistent task history, solve times, and success metrics.
Cache Layer	Redis	Provides low-latency task status and rate-limiting counters.

3. Monitoring & Observability Integration

The architecture includes dedicated integration points for real-time monitoring:

- **System Health**: Health check endpoints (`/health`) on all microservices for load balancer heartbeats.
- **Current Load**: Prometheus metrics tracking queue depth and active worker counts.
- **Dashboards**: Grafana integration for visualizing solve rates, latencies, and system performance.
- **Error Logging**: Centralized ELK stack (Elasticsearch, Logstash, Kibana) for structured error tracking.

4. Scaling & Failover Mechanisms

- **Horizontal Scaling**: New worker nodes can be added dynamically based on RabbitMQ queue depth.
- **Failover**: RabbitMQ mirrored queues and PostgreSQL primary-replica configurations prevent single points of failure.
- **Recovery**: Automatic Task Re-queuing (Max 3 retries) with exponential backoff for failed solve attempts.
- **Circuit Breaking**: Prevents system overload by rejecting requests if the queue exceeds safe thresholds.

5. Deployment Readiness

The entire stack is containerized using **Docker** and **Docker Compose**, allowing for consistent deployments across development, staging, and production environments.