Autonomous Visual Agents for WebUI Testing

Sreeram Sreedhar, Computer Science, BS
Mentor: Dr. Srividya Bansal, Associate Professor
SCAI, Ira A. Fulton Schools of Engineering

Abstract

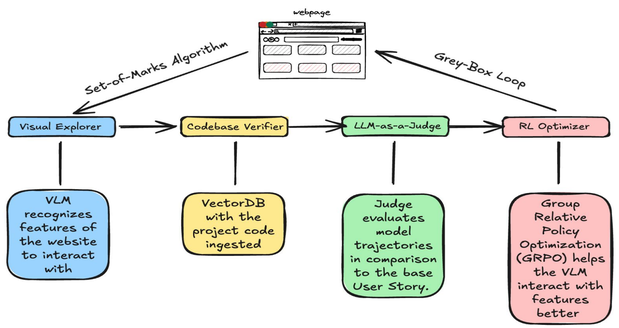
Traditional automated testing is known for being brittle. Even a tiny change to the DOM can break a script, leading to a constant cycle of maintenance. We're tackling this by moving towards a **"Grey-Box" architecture**. Instead of relying solely on code, our system uses Vision-Language Models (VLMs) to "see" the UI and Codebase to understand the backend logic.

By using a **Set-of-Marks (SoM)** algorithm and an **LLM-as-a-Judge**, the system interacts with the app like a human would while solving the "Oracle Problem" (knowing if a test actually passed). We've also integrated **Reinforcement Learning** so the agent gets smarter at exploring the app and pinpointing exactly where a fault occurred. Ultimately, we wanted to see what drives these results: is it the visual understanding of the UI, or the smarter decision-making powered by RL?

References

WebRL: "WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning." *ICLR Proceedings*, 2025
TOGLL: "Correct and Strong Test Oracle Generation with LLMs." *arXiv*, 2024.
WebJudge: "An Illusion of Progress? Assessing the Current State of Web Agents (Online-Mind2Web)." *arXiv*, 2025.
AutoWebGLM: "Bootstrap and Reinforce a Large Language Model-based Web Navigating Agent." *KDD*, 2024.
Reflexion: "Language Agents with Verbal Reinforcement Learning." *arXiv*, 2023.
Economic Analysis: "AI Automation vs Selenium: Complete Technical Guide." *Virtuoso QA*, 2026.

Methodology



The architecture implements a continuous feedback loop that bridges the gap between visual symptoms and source code logic through four specialized agents:

1. Visual Explorer (The Policy): This VLM-driven agent perceives the environment by capturing screenshots overlaid with a **Set-of-Marks (SoM)** algorithm. These marks assign unique IDs to all interactive elements, allowing the agent to navigate based on "rendered reality" rather than brittle DOM selectors

2. Codebase Verifier (The Grounding): To prevent silent failures or "hallucinations," this agent captures network requests and execution logs in real-time. It utilizes a **Codebase RAG** system to query a vector database of indexed source code, mapping UI actions back to specific backend controllers and frontend components.

3. Semantic Judge (The Reward Model): An LLM evaluates the entire session trajectory against high-level human **User Stories**. It performs cross-modal consistency checks (e.g., verifying if the text on screen matches API returned data) to output a dense scalar reward and a natural language critique.

4. Policy Optimizer (The Learner): Utilizing **Reinforcement Learning (RL)**, specifically techniques like **Group Relative Policy Optimization (GRPO)**, this module updates the Explorer's policy. Successful trajectories are reinforced, while failed attempts are corrected via the Judge's feedback (Reflexion), allowing the agent to evolve its exploration strategy over time.

Comparative Analysis

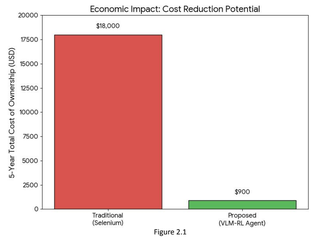


Figure 2.1

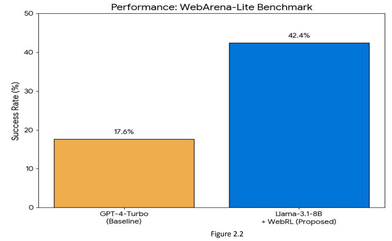


Figure 2.2

Figure 2.1 illustrates the economic disparity between testing paradigms, where traditional Selenium-based automation incurs a five-year Total Cost of Ownership (TCO) of approximately **$18,000** per test compared to **$900** for agentic frameworks.

Figure 2.2 highlights performance gains in long-horizon navigation, demonstrating that the **Llama-3.1-8B + WebRL** model achieves a **42.4%** success rate on the WebArena-Lite benchmark, significantly outperforming the **17.6%** baseline set by GPT-4-Turbo

Conclusion and Future Work

This research replaces brittle scripts with an autonomous **"Grey-Box"** paradigm using VLMs and RAG to resolve the **"Oracle Problem"**. By mapping visual failures directly to source code, the system achieves a ten-fold improvement in bug detection over traditional methods. This self-healing architecture targets a **90% reduction** in maintenance costs, offering a scalable and resilient model for modern software quality assurance.
For future work -

- **SUDOKN Integration:** Deploying the framework to validate complex knowledge graph renderings and supply chain discovery tools.
- **Industry Standardization:** Establishing a "self-healing" protocol as a standard for enterprise CI/CD pipelines to decouple QA costs from feature growth.
- **Safety-Aware RL:** Implementing reward penalties to prevent unintended autonomous actions, such as data deletion, during exploration.

FURI

Ira A. Fulton Schools of Engineering
Arizona State University