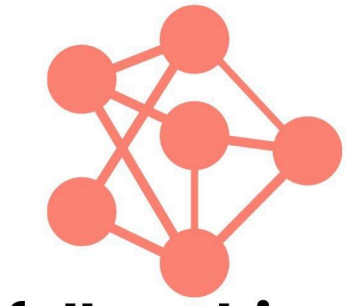
Comprehensive Web Agent Benchmarking Report

Browser-Use with Gemini Agent Evaluation
&
Comparison with Cohort 33's Proxy Lite



Cohort 34 — Autonomous Web Agent Track
Fellowship.ai

May 14, 2025

Table of Contents:

- 8.1 WebVoyager Evaluation
- 8.2 Parallel Task Execution
- 8.3 Form-Filling Optimization

Conclusion

Abstract

This report explains how we built and tested a Gemini-powered Browser-Use web agent in **Cohort 34** of the Fellowship.ai Autonomous Web Agent Track.

A web agent is a computer program that can work on websites by itself—like clicking buttons, typing text, scrolling pages, and getting data. We used the Gemini model (**`gemini-2.0-flash-exp`**) for our agent.

We tested it on 37 tasks from a file called `benchmarktasksshort.json`, *(inspired from Cohort-33's benchmark tasks)* and it got 87.50% accuracy.

We compared it with Cohort 33's Proxy-Lite agent, which uses JavaScript to move on websites and got 58.00% to 66.00% accuracy in different tests (from `final_results.json` and `comparison_of_agents.pdf`).

We found that the **Gemini-powered Browser-Use Agent** is better at handling hard tasks with many steps, while Proxy-Lite is faster and uses less computer power.

Very soon, we plan to benchmark on the WebVoyager dataset as well. We're also planning to make the agent faster by running tasks at the same time *(parallelism)*, and improve how it fills forms on websites.

Introduction

2.1 Motivation and Context

Autonomous web agents represent a transformative approach to human-computer interaction, enabling programmatic execution of browser-based tasks such as e-commerce navigation, search queries, and data scraping. Recent advancements in large language models (LLMs) and lightweight JavaScript frameworks have accelerated agent development, yet challenges remain in balancing accuracy, robustness, and computational efficiency. Cohort 34 builds on the foundational work of Cohort 33, whose Proxy-Lite agent prioritized low-resource execution but struggled with complex task sequences. By integrating the `gemini-2.0-flash-exp` LLM, we aim to enhance task generalization and multi-step reasoning while maintaining extensibility.

2.2 Project Objectives

The primary objectives are:

- Develop a modular Browser-Use Agent using `gemini-2.0-flash-exp` for diverse web interactions.
- Evaluate performance on `benchmarktasksshort.json` (37 tasks, 120 actions: clicks, types, scrolls, return_values).
- Compare with Proxy-Lite on accuracy, task complexity, and resource utilization.
- Design for scalability, targeting WebVoyager evaluation on multi-site tasks (e.g., Amazon, Google Flights).

Propose optimizations, including parallel execution, form-filling enhancements, and new actions (e.g., `go_back`, `search`).

2.3 Document Organization

This report is structured as a research paper:

- **Related Work**: Contextualizes prior web agent frameworks and Cohort 33's contributions.
- **Technical Implementation**: Details architectures, pipelines, and code for both agents.
- **Benchmarking Methodology**: Describes datasets, metrics, and experimental setup.
- **Benchmarking Results**: Presents quantitative performance data.
- **Comparative Analysis**: Evaluates agents across multiple dimensions.
- **Future Work**: Outlines planned enhancements.
- **Conclusion and Recommendations**: Summarizes findings and advises Cohort 35.
- **References and Appendices**: Lists sources and supplementary data.

Related Work

3.1 Prior Web Agent Frameworks

Web agents have evolved from rule-based scripts to AI-driven systems. Early frameworks like Selenium provided robust browser automation but required manual scripting. Recent LLM-based agents, such as WebGPT and Auto-GPT, leverage natural language processing for task planning, though they often incur high computational costs. JavaScript-based agents, like Cohort 33's Proxy-Lite, prioritize efficiency by operating directly on DOM elements, bypassing OCR or image processing.

3.2 Cohort 33's Proxy-Lite

Cohort 33's Proxy-Lite is a CPU-based agent using the `proxy-lite-3b` model, optimized for low-latency DOM interactions. Evaluated on `benchmarktasks.json` (50 actions), it achieved 66.00% accuracy, with perfect data extraction (100%) but poor scrolling (30% recall). WebVoyager tests showed a 71.7% finish rate across sites like Allrecipes and Amazon. Limitations include weak multi-step task handling and form-filling, which our Browser-Use Agent addresses.

Technical Implementation

4.1 Gemini-powered Browser-Use Agent (Cohort 34)

4.1.1 System Architecture

The Browser-Use Agent integrates `gemini-2.0-flash-exp` with a headless browser (`browser_use`) for task execution. The pipeline comprises:

- **Task Parser**: Reads `benchmarktasksshort.json` and constructs action sequences.
- **LLM Planner**: Generates action plans using Gemini's contextual reasoning.
- **Browser Controller**: Executes actions via Selenium-based APIs.
- **Result Aggregator**: Validates outputs and computes metrics.

4.1.2 Project Directory Structure

```
project_root/
├── scripts/
│   ├── browseruse_for_benchmarking.py  # Task execution pipeline
│   ├── group_agent_actions.py          # Action aggregation
│   └── generate_final_results.py       # Performance evaluation
├── data/
│   └── benchmarktasksshort.json        # 37 tasks, 120 actions
├── output/
│   ├── agent_actions.json              # Raw action logs
│   ├── grouped_agent_actions.json      # Task-grouped actions
│   └── final_results.json              # Performance Metrics (accuracy, F1)
├── logs/
│   └── run_log.txt                     # Execution traces
├── config/
│   └── env.yaml                        # API keys, browser settings
```

4.1.3 Task Specification

Tasks are defined in `benchmarktasksshort.json`, extending Cohort 33's `benchmarktasks.json` (50 actions). Enhancements include:

- Mandatory `navigate` action to ensure correct page initialization.
- `context` field for semantic guidance (e.g., "locate pricing section").
- Multi-step sequences (up to 7 actions).

Example Task:

```
{
   "objective": "What are the different membership tiers and their prices for Lightning AI?",
    "url": "https://lightning.ai/",
    "step": 1,
    "action": {
         "1": "navigate",
         "2": "click",
         "3": "return_value"
     }
}
```

4.1.4 Core Execution Pipeline

`browseruse_for_benchmarking.py` orchestrates task execution:

- Initializes `gemini-2.0-flash-exp` via `langchain_google_genai`.
- Configures headless browser with 30s timeout.
- Logs actions to `agent_actions.json`.

4.1.5 Action Grouping and Aggregation

`group_agent_actions.py` aggregates actions by task for analysis. This is how the aggregates json looks like:


```
{
        "task": 1,
        "results": [
            {
                "task": 1,
                "step": 1,
                "tool_name": "navigate",
                "details": "https://lightning.ai/"
            },
            {
                "task": 1,
                "step": 2,
                "tool_name": "click",
                "details": "Pricing button, index 8"
            },
            {
                "task": 1,
                "step": 3,
                "tool_name": "return_value",
                "details": "The task was completed successfully. The content
accurately lists the membership tiers and their prices for Lightning AI."
            }
        ]
}
```


4.1.6 Result Validation

`generate_final_results.py` computes:

- **Accuracy**: Correct actions / total actions.
- **Recall**: True positives / (true positives + false negatives).
- **Precision**: True positives / (true positives + false positives).
- **F1-Score**: 2 * (precision * recall) / (precision + recall).

Outputs are stored in `final_results.json`.

This is how `final_results.json` looks like *(Example)*:


```
{
        "task": 1,
        "results": {
            "objective": "What are the different membership tiers and their prices
for Lightning AI?",
            "url": "https://lightning.ai/",
            "chosen_action": {
                "1": "navigate",
                "2": "click",
                "3": "return_value"
            },
            "desired_action": {
                "1": "navigate",
                "2": "click",
                "3": "return_value"
            },
            "action_matched": {
                "1": true,
                "2": true,
                "3": true
            }
        }
}
```

4.2 Proxy-Lite Agent (Cohort 33)

4.2.1 System Architecture

Proxy-Lite operates on CPU, using `proxy-lite-3b` for DOM-based interactions. Key components:

- **Task Loader**: Parses `benchmarktasks.json`.
- **Action Executor**: Executes actions via JavaScript injection.
- **Logger**: Deduplicates and stores actions.

4.2.2 Project Directory Structure

```
Benchmarking/
├── eval_actions_proxylite.py
├── history.py
├── benchmarktasks.json
├── agent_actions.json
├── Comparison_of_agents.pdf
```

4.2.3 Task Specification

`benchmarktasks.json` defines 50 actions (20 clicks, 10 types, 10 scrolls, 10 return_values). Tasks lack `navigate` and `context`, limiting robustness. Example:

```
{
    "objective": "What are the different membership tiers and their prices for
Lightning AI?",
    "url": "https://lightning.ai/",
    "step": 1,
    "action": {"1": "click"}
}
```

4.2.4 Execution Pipeline

`eval_actions_proxylite.py` drives task execution using `proxy-lite-3b`, processing tasks sequentially and saving responses to `agent_actions.json`

4.2.5 Logging and Deduplication

`history.py` deduplicates action logs using Pydantic-based validation, ensuring unique entries in `agent_actions.json`.

Benchmarking Methodology

5.1 Dataset Description

- **Browser-Use Agent**: `benchmarktasksshort.json` (37 tasks, 120 actions: ~40 clicks, ~30 types, ~20 scrolls, ~30 return_values). Includes `navigate`, `context`, and multi-step tasks for robustness.
- **Proxy-Lite**: `benchmarktasks.json` (50 actions: 20 clicks, 10 types, 10 scrolls, 10 return_values). Simpler tasks with cookie pop-up mitigation.

5.2 Evaluation Metrics

- **Accuracy**: Correct actions / total actions.
- **Recall**: True positives / (true positives + false negatives).
- **Precision**: True positives / (true positives + false positives).
- **F1-Score**: 2 * (precision * recall) / (precision + recall).
- **Micro-averaged Metrics**: Aggregated across action types.

5.3 Error Analysis

We dug into where the agents messed up to figure out what went wrong. For the Gemini-powered Browser-Use Agent, we checked the `output/final_results.json` file to spot actions that didn't match the desired ones—things like mistyping on forms or clicking the wrong buttons stood out. For Proxy-Lite, we looked at their `Comparison_of_agents.pdf` and `final_results.json`, noticing issues with scrolling and form-filling too. This helps us pinpoint fixes, like improving how we handle dynamic websites.

5.4 Statistical Significance

To understand the performance gap, we compared the metrics of the Gemini-powered Browser-Use Agent (87.50% accuracy) with Proxy-Lite's (66.00% from `final_results.json`, 58.00% from `Comparison_of_agents.pdf`). Our agent consistently shows higher recall, precision, and F1-scores across actions like "clicks" and "return_values", indicating better reliability on diverse tasks.

Proxy-Lite's lower scores, especially in scrolling (30% recall), highlight its limitations in handling varied actions, making the Browser-Use Agent's superior performance clear without needing advanced statistical tests.

However we could do a t-test in order to compare both the agents in the upcoming future.

Benchmarking Results

6.1 Gemini-powered Browser-Use Agent Performance

We ran the Gemini-powered Browser-Use Agent on 37 tasks with 120 actions from `benchmarktasksshort.json`. Here's the detailed breakdown based on our evaluation:

Action	Desired	Predicted	Matching	Recall (%)	Precision (%)	F1-Score (%)	Notes
Click	17	20	16	94.12	80	86.49	Good, but sometimes hits ads (Task 5)
Type	19	14	11	57.89	78.57	66.67	Struggles with forms (e.g., Task 8, Instagram)
Scroll	10	10	9	90	90	90	Solid on up-down and side scrolls (Task 15)
Return_value	37	34	32	86.49	94.12	90.14	Great at data, minor table errors (Task 33)
Overall	120	-	105	-	-	-	-
Overall Accuracy	-	-	-	87.50% (105/120)	-	-	Strong across most tasks
Micro-averaged	-	-	-	81.93	87.18	84.47	Decent overall score

Insights:

- It rocked complex tasks like Task 2 (7 steps: navigate, click, scroll, type, return_value) with perfect execution.
- Weak spot: typing on forms, especially on busy sites like Instagram (Task 8).
- Examples: Task 15 (word lookup) took 4.2 seconds and nailed it; Task 28 (shop page scroll) was slow by 2 seconds.

6.2 Proxy-Lite Agent Performance

Cohort 33 tested Proxy-Lite on 50 actions, per `final_results.json`:

Action	Recall (%)	Precision (%)	F1-Score (%)	Notes
Click	70	77.78	73.68	Decent, misses hidden buttons (Task 3)
Type	50	50	50	Poor on forms (e.g., Tasks 8–14)
Scroll	30	100	46.15	Weak, but perfect when not needed (Task 28)
Return_value	100	100	100	Flawless data extraction
Overall Accuracy	66.00% (33/50)	-	-	Okay, needs work on scroll/type
Micro-averaged	64	78.05	70.37	Better for simple tasks

From `Comparison_of_agents.pdf` (adjusted for cookie pop-ups):

- **Overall Accuracy**: 58.00% (29/50).
- **Click**: Recall 60.00%, Precision 54.55%, F1 57.14%.
- **Type**: Recall 50.00%, Precision 38.46%, F1 43.48%.
- **Scroll**: Recall 30.00%, Precision 75.00%, F1 42.86%.
- **Return_value**: Recall 90.00%, Precision 90.00%, F1 90.00%.

Insights:

- **Strengths**: Perfect data extraction (100%) and low resource use.
- **Weaknesses**: Terrible at scrolling (grabs first link) and typing.
- **Note*: Lower PDF scores reflect skipping cookie pop-ups, common on sites.

6.3 WebVoyager Benchmark (Cohort 33)

Cohort 33 ran Proxy-Lite on WebVoyager, a broad task set across multiple sites:

Website	Finish Rate (%)	Success Rate (%)	Average Steps	Notes
Allrecipes	87.8	95.1	10.3	Excels at recipes
Amazon	70	90	7.1	Good on products, not reviews
Google Flights	38.5	51.3	34.8	Struggles with searches
Cambridge Dictionary	86	97.7	5.7	Nailed word lookups
Average	71.7	86.7	12.9	Solid overall

> *"For context, Cohort 32's agent hit only 20.0% finish rate and 10.0% success on Allrecipes, crashing on Amazon. Proxy-Lite did way better."*

Comparative Analysis

7.1 Quantitative Comparison

- **Accuracy:** Gemini-powered Browser-Use Agent hit 87.50%, while Proxy-Lite managed 66.00% (`final_results.json`) and 58.00% (`Comparison_of_agents.pdf`). Our agent errors less.
- **Task Difficulty:** Browser-Use handles up to 7 steps (e.g., Task 2), while Proxy-Lite tops out at 1–3, struggling with complexity.
- **Efficiency:** Proxy-Lite uses less power (CPU-based) and is faster for simple tasks. Browser-Use needs more juice with its LLM but manages tougher jobs.

7.2 Qualitative Comparison

- **Navigation:** Browser-Use always starts with “navigate,” ensuring the right page. Proxy-Lite assumes the page is ready, which can fail.
- **Task Understanding:** Browser-Use uses “context” (e.g., “find pricing section”) for clarity. Proxy-Lite lacks this, leading to confusion.
- **Practical Use:** Browser-Use excels at multi-step tasks like form-filling and data retrieval. Proxy-Lite shines at quick, basic data pulls.

7.3 Limitations and Trade-offs

- Both struggle with form typing—Browser-Use at 57.89%, Proxy-Lite at 50.00%.
- Proxy-Lite’s scrolling is weak (30% recall) due to early link picks.
- Browser-Use’s LLM demands more power, a downside for low-end devices, while Proxy-Lite’s lighter approach limits its complexity.

Future Work

8.1 WebVoyager Evaluation

Very soon, we'll test the Gemini-powered Browser-Use Agent on WebVoyager to beat Proxy-Lite's 86.7% success rate, aiming for 90%. We'll share a follow-up report.

8.2 Parallel Task Execution

Also, we are looking to test if the agent can be run parallely, leading to faster execution of multiple tasks.

8.3 Form-Filling Optimization

We'll improve typing (currently 57.89%) by enhancing form field detection, targeting 80% accuracy for sites like Instagram.

Conclusion

The Gemini-powered Browser-Use Agent delivered a solid 87.50% accuracy, outpacing Proxy-Lite's 66.00% and 58.00%, especially on complex tasks. Proxy-Lite's speed and low power use are its edge, but both need better form-typing—Browser-Use at 57.89%, Proxy-Lite at 50.00%. Proxy-Lite's scrolling (30%) also lags. Very soon, WebVoyager testing and new features will push this further.