REPORT: A COMPARISON BETWEEN DIFFERENT WEB AGENTS

COHORT 33 - LLM WEB AGENTS TRACK

1 Introduction

The goal of the LLM web agents track of the 33rd cohort of the Fellowship.AI program was the creation of a web agent capable of autonomously navigating through webpages and scraping real-time financial information. Our starting point was the prototype created by the 32nd cohort and our original aim was to expand that prototype to a production-ready web agent. However, after we had gained an overview of the code and run first experiments, it turned out that the prototype was both computationally expensive and ineffective. Furthermore, by the time we started our work, several promising open-source alternatives had been published. For these reasons, throughout the time of the fellowship program, we tested out, compared and worked with several of these alternatives.

The goal of this report is to provide an overview of the different publicly available models and agents we experimented with in order to give the 34th cohort a recommendation on how to proceed. The agents covered in this report are:

- Section 2 - Midscene.js
- Section 3 - UI-TARS Desktop Agent
- Section 4 - Proxy (lite)

In Section 5 we give an extensive overview of the benchmarking results we obtained in particular for the proxy-lite agent in comparison to the 32nd cohort's agent and the UI-TARS agent. Finally, in Section 6 we provide a brief conclusion of our results.

2 Midscene.js

Midscene.js, a Chrome extension, was evaluated as part of our testing of an autonomous web agent for extracting and structuring financial data from a bank's website, leveraging Hugging Face Endpoints. To clarify, Midscene is a web-based agent; that we configured to operate with the UI-TARS model, particularly 7-B SFT. This evaluation drew on the 32nd cohort's approach to agent structuring, though Midscene was not our initial choice, it was one of several options assessed to determine its suitability. The aim was to analyze its ability to streamline financial data collection, utilizing a vision-language model (VLM) to navigate complex webpage layouts and retrieve essential information efficiently.

2.1 Technical Implementation

We configured Midscene via PowerShell with the following parameters:

```
OPENAI_BASE_URL=""
OPENAI_API_KEY=""
MIDSCENE_USE_VLM_UI_TARS=1
MIDSCENE_OPENAI_INIT_CONFIG_JSON='{"defaultheaders": {"Authorization": ""}}'
MIDSCENE_MODEL_NAME="tgi"
```

Initial token allocation:

- **Input tokens** - 40,865
- **Maximum new tokens** - 2,048
- **Total** - 42,913 (exceeding the permitted limit of 32,768)

Sample JSON input

```
{
  "product": "Dell Laptop",
  "price": "$999"
}
```

2.2 BBC Extraction Test (Unsuccessful)

- **Task** - Extract the headline from the BBC homepage.
- **Page content** - about 4,000 tokens
- **Extracted output** - about 500 tokens
- **Outcome** - The total token count exceeded 32,768, resulting in a "Planning422 (Token Limit Exceeded)" error (output below).

BBC_Midscene_Extension.mkv

Reduction attempts included:

1.) max_input_tokens = 4000, max_new_tokens = 1000 – Unsuccessful

2.) max_input_tokens = 2000, max_new_tokens = 1000 – Unsuccessful

3.) max_input_tokens = 1000, max_new_tokens = 500 – Unsuccessful

Each effort was thwarted by excessive token usage, significantly limiting Midscene's effectiveness.

2.3 Comparison and Transition to UI-TARS Desktop

Our task was to assess Midscene's performance, specifically its speed, resource efficiency, and overall capability, against the UI-TARS Desktop agent.

Midscene, operating with the UI-TARS model, showed potential for structured data extraction but faced notable challenges:

- **Token Limit Constraints** - Despite adjustments, it consistently surpassed OpenAI's token threshold, hindering larger extractions.
- **Scrolling Limitations** - It struggled with dynamic web page elements, missing content that required scrolling.
- **Incomplete Outputs** - Token restrictions often led to truncated or partial data retrieval.

These shortcomings prompted a shift to the UI-TARS Desktop agent, also utilizing the UI-TARS model, which outperformed Midscene in key areas, as shown in the flowchart below,

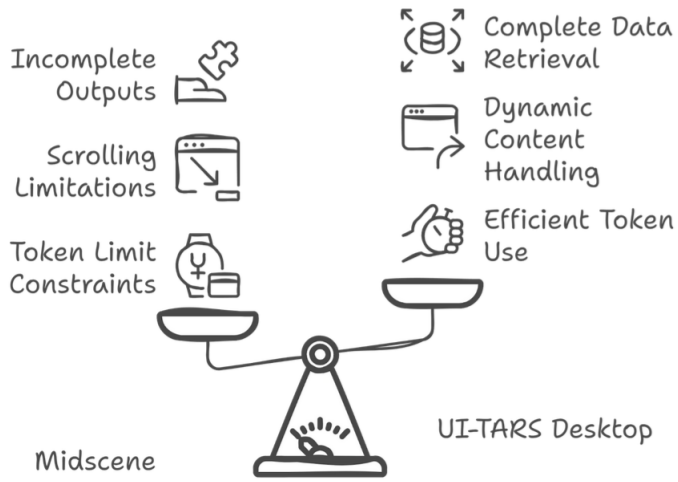

Figure 1: Comparing Midscene and UI-TARS Desktop performance.

2.4 Conclusion

Midscene, initially tested as a Chrome extension agent paired with the UI-TARS model, provided a practical baseline informed by the 32nd cohort's efforts to handle web-based tasks.

However, its limitations quickly surfaced, excessive token consumption bogged down processing, and its struggles with dynamic site interactions, such as handling API rate limits (429 errors) and larger webpages, hindered its reliability for real-world financial scraping. These shortcomings prompted a pivot to the UI-TARS Desktop agent, which utilized the same UI-TARS model (primarily 7B DPO) but offered a standalone GUI approach.

This shift proved advantageous: UI-TARS Desktop demonstrated superior navigation through complex web interfaces, greater efficiency in processing multimodal inputs like screenshots, and enhanced reliability in execution-based tasks, such as extracting loan details from HDFC Bank's site. While not without flaws, namely its resource intensity and occasional failure to complete secondary tasks like saving outputs, it outshone Midscene as a more excellent tool for financial data extraction aligning better with our need for practical, dependable performance.

3 UI-TARS Desktop Agent

This section evaluates the UI-TARS Desktop agent, a GUI-based autonomous tool deployed with the UI-TARS 7B model (and tested with 2B SFT) to assess its capabilities for web navigation and financial data extraction.

Developed by ByteDance Research and hosted at Github (link below), this agent was tested between February and March 2025 following Midscene's limitations.

https://github.com/vishwamartur/UI-TARS-desktop

Our team, Olwin Christian (HDFC Bank tests), Rahul Thakur (benchmarks), and Iremide Oloyede (scraping), explored its performance, runtime stability, and suitability for financial tasks, guided by mentors Feras and Kukesh, and contributions from other team members to refine the model's performance. While UI-TARS showed promise, its complexity and resource demands shaped our final conclusions.

3.1 Technical Implementation

UI-TARS Desktop was installed as a standalone application on Windows/Linux via its GitHub releases page, configured with Hugging Face Endpoints provided by Mike Fuller:

- **Base URL** - https://pktofecpti9fyu52.us-east-1.aws.endpoints.huggingface.cloud/v1/
- **Token** - Standard Hugging Face authentication.
- **Models** - UI-TARS 7B DPO (primary), 2B SFT (secondary).
- **Cost** - $1.8/hour, scaling to zero after 15 minutes, occasionally yielding 503 errors.

Setup followed the repo's deployment guide:

1.) API Service - Launched via

python -m vllm.entrypoints.openai.api_server –served-model-name ui-tars –model < path > .

2.) Local Install - Run

pnpm install && pnpm run dev

after downloading the 7B-DPO model.

Sample task prompt (Olwin's HDFC test):

```
{
  "task": "Calculate HDFC Bank loan interest",
  "url": "https://hdfc_bank_loans",
  "input": "screenshot + rate details"
}
```

3.2 Rahul's Benchmarks

```
model =
↪  Qwen2VLForConditionalGeneration.from_pretrained("bytedance-research/UI-TARS-7B-DPO",
↪  torch_dtype=torch.float16,
↪  quantization_config=BitsAndBytesConfig(load_in_8bit=True))
```

```
tokenizer = AutoTokenizer.from_pretrained("bytedance-research/UI-TARS-7B-DPO")
```

Tests ran on Colab (GPU) and local Windows setups, with 2B facing frequent endpoint issues and 7B causing runtime instability.

3.3 Performance Evaluation - HDFC Bank EMI Extraction

Task: Scrape and calculate EMI from HDFC Bank's loan page using UI-TARS 7B DPO (Olwin).

- **Input** - Screenshot + prompt (about 500 tokens).
- **Expected Output** - EMI for 315,000 at 12.5% over 5 years (about 8,400).
- **Outcome** - Successfully computed 8,412 after about 2 minutes of loading and processing, surpassing Midscene's token-limited failures, but failed to screenshot and save the result to the desktop.

3.4 Iremide's 2B SFT scraping test (Bankrate.com)

```
model =
↪  AutoModelForVision2Seq.from_pretrained("bytedance-research/ui-tars-2b-sft").to("cuda")
inputs = tokenizer("Find mortgage rates on bankrate.com...",
↪  return_tensors="pt").to("cuda")
output = model.generate(**inputs, max_length=1500)
```

- Output:

```
[ { "tool": "Click", "params": { "selector": "text=\"CD rates\"" } } ]
```

- Result: Failed due to OCR errors and 503 timeouts; only 7B handled dynamic pages reliably.

3.5 Rahul's GSM8K benchmark (7B SFT)

Question - Janet's ducks lay 16 eggs daily...

Ground Truth - 18

Prediction - 18

Accuracy - 0.175 (35/200 correct)

3.6 ScienceQA benchmark test

We moreover evaluated the UI-TARS model, together with the proxy-lite model, on the science-related multiple choice tests from the ScienceQA dataset. For more information, we refer to Section 5.3 below.

For more details of the results achieved by the UI-TARS model on the GSM8K and the ScienceQA (as well as the HellaSwag datasets) we refer to Section 5.4 below.

3.7 Key Observations and Limitations

UI-TARS Desktop with 7B excelled in practical tasks but faltered in scalability:

- **Execution** Strength - Outperformed Midscene in EMI extraction, leveraging screenshot-based navigation and OS control.

- **Resource Demands** - 7B required GPU and endpoints, crashing Colab runtimes; 2B suffered 503 errors and poor reasoning (26.5% HellaSwag accuracy).
- **Benchmark Gaps** - Rahul's tests showed 17.5% GSM8K, 26.5% HellaSwag, and ROUGE-1 about 0.14 for summarization, weak without multimodal finetuning.
- **Scraping Issues** - Iremide's 2B OCR approach failed on dynamic content, echoing Scrum's prompting woes.
- **Complexity** - OS-level features exceeded our web-focused needs, per Kukesh's guidance toward lighter options.

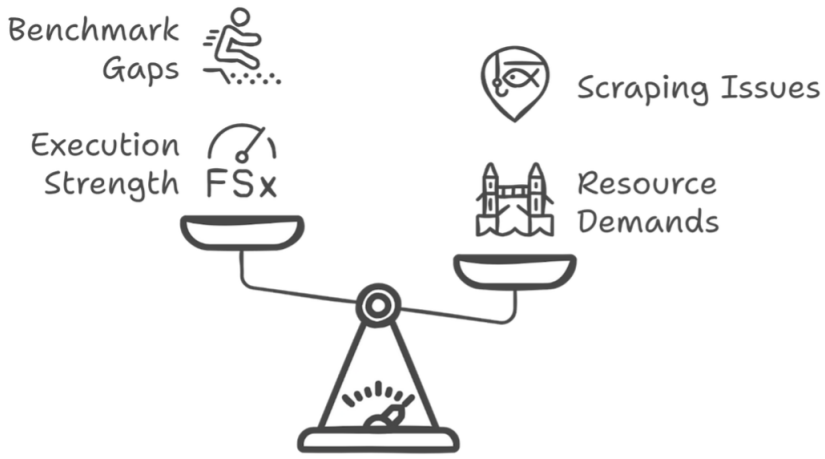

Figure 2: The weaknesses of the UI-TARS model outweigh its strengths with regards to the objective of building a browser-integrated web agent.

3.8 Conclusion

The UI-TARS Desktop agent, deployed with the 7B DPO model, delivered impressive capabilities in web and GUI navigation, notably outperforming Midscene in Olwin's HDFC Bank EMI extraction test by efficiently handling multimodal inputs and dynamic page interactions.

Rahul's comprehensive benchmarks across SQuAD, GSM8K, HellaSwag, and CNN/DailyMail revealed its potential, achieving 17.5% accuracy on math reasoning tasks, yet exposed inconsistent reasoning (10% on HellaSwag) and modest summarization scores (ROUGE-1 about 0.14), suggesting limitations in broader applicability without extensive finetuning.

Iremide's scraping trials with the 2B SFT model further underscored inefficiencies, as OCR errors and endpoint timeouts (503 errors) hampered performance on sites like https://www.bankrate.com/, while the 7B model, though more reliable, demanded significant GPU resources and occasionally failed secondary tasks like saving outputs to the desktop.

These findings highlighted a recurring theme: high computational overhead and complexity that often exceeded our project's web-focused requirements. Guided by mentor Kukesh's recommendation to prioritize efficiency and alignment with our core objectives, we concluded that while UI-TARS Desktop offered powerful tools for financial data extraction, its resource intensity and extraneous features prompted a strategic shift toward a leaner, more tailored solution like proxy-lite to better serve our streamlined, web-centric goals.

4 Proxy (lite)

After struggling with the high computational costs of UI-TARS, we turned our attention to a different framework: the Proxy web agent by Convergence. The Proxy agent appears to be a less computationally expensive, more agile alternative to UI-TARS. This advantage comes at the cost of the agent being only capable of interacting with web interfaces within a web browser (and not the user's entire PC), which, however, is perfectly in alignment with the overall goal of building a (browser-integrated) webscraping agent.

Another specialty of the Proxy agent is that it does not make use of any optical character recognition techniques. Instead it uses JavaScript to detect interactable elements, navigate through the webpages and scrape content.

In order to get a first impression of the (full) Proxy web agent, the Browser version of the agent can be run directly from convergence.ai. In contrast to ByteDance Research (UI-TARS), however, Convergence offers a lite version of their agent, namely, the proxy-lite webagent running on the proxy-lite-3b model, which can be run locally via CPU (instead of a GPU) and does not require the usage of inference endpoints.

4.1 Technical implementation

In order to run the proxy-lite webagent locally, its Github repository needs to be cloned. Then, for instance, the agent can be run via a Python script of the following form:

1.) The system setup needs to be configured via RunnerConfig.

```python
from proxy_lite import Runner, RunnerConfig
from qwen_vl_utils import process_vision_info
from transformers import AutoProcessor
from proxy_lite.tools import ReturnValueTool, BrowserTool
from proxy_lite.serializer import OpenAICompatibleSerializer

config = RunnerConfig.from_dict(
    {
        "environment": {
            "name": "webbrowser",
            "homepage": "https://www.google.com/",
            "headless": True,
        },
        "solver": {
            "name": "simple",
            "agent": {
                "name": "proxy_lite",
                "client": {
                    "name": "convergence",
                    "model_id": "convergence-ai/proxy-lite-3b",
                    "api_base": "https://convergence-ai-demo-api.hf.space/v1",
                },
            },
        },
        "max_steps": 50,
        "action_timeout": 1800,
        "environment_timeout": 1800,
        "task_timeout": 18000,
        "logger_level": "DEBUG",
    },
)

```

```
proxy = Runner(config=config)
```

2.) The initial prompt and the starting URL need to be assembled and brought into a form that can be processed by the agent.

```
message_history = [
    {
        "role": "system",
        "content": "You are Proxy Lite, a web-browsing agent that can perform searches, extract information, and take actions based on observations. Use the browser tool to interact with the web."
    },
    {
        "role": "user",
        "content": "What's the weather like in London today?"
    },
    {
        "role": "user",
        "content": [
            {"type": "text", "text": "URL: https://www.google.com/ \n- [0] <a>About</a> \n- [1] <a>Store</a>...."}
        ]
    },
]

processor = AutoProcessor.from_pretrained("convergence-ai/proxy-lite-3b")
tools = OpenAICompatibleSerializer().serialize_tools([ReturnValueTool(), BrowserTool(session=None)])

templated_messages = processor.apply_chat_template(
    message_history, tokenize=False, add_generation_prompt=True, tools=tools)
```

3.) Finally, a function needs to be called (asynchronously) to run the agent with the above specifications.

```
async def main():
    response = await proxy.run(templated_messages)
    print(response)

asyncio.run(main())
```

While the proxy-lite-3b agent is a downgraded lightweight version of the actual Proxy agent, it showed strong results during the initial runs carried out by the team. These first results were further backed up by its benchmarking scores. We provide an extensive overview of the benchmarking results achieved by the proxy-lite agent (in comparison to the 32nd cohort's agent) in Section 5 below.

5 Benchmarking

We carried out two types of benchmark tests to compare the performances of the 32nd cohort's agent and the proxy-lite agent. More specifically, we first evaluated the performance over entire runs on the WebVoyagerDataSet. Subsequently, in a more fine-grained approach, we evaluated the ability of the agents to correctly choose actions in individual steps. In addition, we created notebooks for the evaluation of the UI-TARS model's ability to answer questions based on both visual and textual information.

5.1 Evaluation of entire runs on the WebVoyager dataset

The WebVoyager dataset is a dataset created for benchmarking autonomous webagents, consisting of 643 individual tasks distributed over 15 different websites. During benchmarking, each task is passed to the agent together with the corresponding webpage as a starting URL and the following achievements are evaluated:

- Finish: Has the agent finished its task without running into an error?
- Success: If the agent has finished its task, was its final answer correct?
- Steps: How many steps did the agent carry out?

We point out that, while the evaluation of the finish-rate and the number of steps is straight-forward, the evaluation of the success of a run is somewhat subjective and needs to be carried out either through human evaluation or via a vision language model such as GPT-4V.

5.1.1 Results of the proxy-lite agent

The proxy-lite-3b agent had already been evaluated on the WebVoyager dataset by its creators, achieving the following results (cf. the proxy-lite agent's official Hugging Face page).

Webpage	Finish rate	Success rate	Avg. steps
Allrecipes	87.8%	95.1%	10.3
Amazon	70.0%	90.0%	7.1
Apple	82.1%	89.7%	10.7
ArXiv	60.5%	79.1%	16.0
BBC News	69.4%	77.8%	15.9
Booking	70.0%	85.0%	24.8
Cambridge Dict.	86.0%	97.7%	5.7
Coursera	82.5%	97.5%	4.7
ESPN	53.8%	87.2%	14.9
GitHub	85.0%	92.5%	10.0
Google Flights	38.5%	51.3%	34.8
Google Map	78.9%	94.7%	9.6
Google Search	71.4%	92.9%	6.0
Huggingface	68.6%	74.3%	18.4
Wolfram Alpha	78.3%	93.5%	6.1

5.1.2 Results of the 32nd cohort's agent

For our project, we tried to replicate these results with the 32nd cohort's self-built agent, however, we ran into difficulties caused by the high computational costs and the slowness of the agent. For this reason, we planned to evaluate the agent on only ten tasks on each of the webpages from the WebVoyager dataset. The specific set-up we used for the benchmark test consisted of the streamlit version

of the agent (cf. GithubRepository) on a CPU with llama-3.2-90b-vision-preview as the underlying model, llava as the caption model and a maximum of 50 allowed steps.

We began the test by running the agent on the first ten tasks of the first webpage (Allrecipes) of the WebVoyager dataset, achieving the following results, where the success of each run was humanly evaluated:

Prompt	Finish?	Success?	Steps
Provide a recipe for vegetarian lasagna with more than 100 reviews and a rating of at least 4.5 stars suitable for 6 people.	Yes	Yes	10
Find a recipe for a vegetarian lasagna that has at least a four-star rating and uses zucchini.	No	No	13
Find a recipe for a vegetarian lasagna under 600 calories per serving that has a prep time of less than 1 hour.	No	No	7
Locate a recipe for vegan chocolate chip cookies with over 60 reviews and a rating of at least 4.5 stars on Allrecipes.	No	No	4
Find a recipe for Baked Salmon that takes less than 30 minutes to prepare and has at least a 4 star rating based on user reviews.	No	No	3
Search for a popular Pasta Sauce with more than 1000 reviews and a rating above 4 stars. Create a shopping list of ingredients for this recipe.	No	No	4
Search for a vegetarian lasagna recipe that has at least a four-star rating and over 500 reviews.	No	No	1
Find a popular recipe for a chocolate chip cookie and list the ingredients and preparation steps.	No	No	1
Search for a recipe for Beef Wellington on Allrecipes that has at least 200 reviews and an average rating of 4.5 stars or higher. List the main ingredients required for the dish.	Yes	No	14
Find a high-rated recipe for vegetarian lasagna, list the key ingredients required, and include the total preparation and cook time stated on the recipe.	No	No	8

These results show a finish-rate of 20.0% and a success-rate of 10.0%. While the run of the agent on these tasks took several hours and thus already significantly exceeded the expected time, the run on the tasks for the second website(Amazon) turned out even more problematic. Indeed, after taking over an hour for the second one of these tasks, the agent crashed the local machine on which it was run due to the high computational costs.

Prompt	Finish?	Success?	Steps
Search an Xbox Wireless controller with green color and rated above 4 stars.	No	No	2
Search for women's golf polos in m size, priced between 50 to 75 dollars, and save the lowest priced among results.	No	No	-

After this failed attempt we decided to abort the benchmark tests due to the problems described above as well as the clearly worse results the agent showed in comparison to the proxy-lite agent.

5.1.3 Observations

Throughout the benchmark tests described above, the proxy-lite agent clearly outperformed the 32nd cohort's agent in terms of both the finish-rate and the success rate. Indeed, while the proxy-lite agent achieved a finish-rate of 87.8% and a success-rate of 95.1% the corresponding rates for the 32nd cohort's agent amount to a mere 20.0% and 50.0%. In addition, while we did not integrate a precise timer for measuring the duration of the 32nd cohort's agent's runs, individual runs exceeded a time of one hour. Corresponding durations were never observed in runs of the proxy-lite agent. Furthermore, in the final run of the benchmark test, the 32nd cohort's agent crashed the local machine on which it was run due to overheating.

We notice that while an evaluation of the 32nd cohort's agent on a larger dataset would be necessary to gain more detailed insights, it seems highly unlikely that in such a benchmark test the agent could come close to the performance of the proxy-lite agent.

5.2 Evaluation of individual actions

For a second, more fine-grained, benchmarking test we decided to evaluate the agents' abilities to correctly choose an action in individual situations.

5.2.1 Results of the 32nd cohort's agent

This approach was inspired by the 32nd cohort's work, who had given their agent a set of tasks to be performed on specific websites, together with a set of *desired* actions which the agent was expected to carry out in the steps of the task. The specific dataset that was used in this benchmark test as well as the obtained results can be found in the 32nd cohort's Github repository (cf. dataset, results). These results translate to the following scores in terms of Recall, Precision and F1-Score:

Action	TP + FN	TP + FP	TP	Recall	Precision	F1-Score
Click	20	25	14	0.7000	0.5600	0.6222
Type	11	10	5	0.4545	0.5000	0.4761
Scroll	3	0	0	0.0000	0.0000	0.0000
Return Value	6	1	1	0.1667	1.0000	0.2858

In this table, TP, FN and FP stand for *True positive*, *False negative* and *False positive*, respectively. For example in case of the action "Click", TP is the number of times the agent chose the action "Click" in a step where "Click" was also specified as the desired action. Correspondingly, FN is the number of times the agent chose a different action even though "Click" was the desired action and FP is the number of times the agent chose "Click" even though a different action was desired. Consequently, TP + FN is equal to the overall number of steps in which "Click" was the desired action and TP +

FP describes the amount of time the agent chose the action "Click". The metrics Recall and Precision in this table are calculated as

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}},$$

i.e. in the case of the action "Click", a recall of 0.7000 means that 70 percent of the overall desired "Click" actions were carried out correctly, while a precision of 0.5600 shows that 56 percent of the carried out "Click" actions were actually desired.
The F1-Score in the above table is determined as the harmonic mean of precision and recall,

$$\text{F1-Score} := \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}.$$

Summing up the above results, we obtain the following overall accuracy for the 32nd cohort's web agent

Total predictions	Correct predictions	Wrong predictions	Accuracy
40	20	20	0.5000

5.2.2 Results of the proxy-lite agent

For the evaluation of the proxy-lite agent in a corresponding benchmark test, a modified dataset had to be used. This is due to two reasons: First, some parts of the 32nd cohort's dataset are not accessible on Github anymore. And second, the proxy-lite agent - when run locally - oftentimes runs into cookie banners, which it typically cannot overcome due to the agents' limited abilities to use the tool "Scroll". Consequently, for the evaluation of the proxy-lite agent a dataset of webpages had to be chosen on which cookie banners either do not show up or can bypassed with a simple click.

We thus used a dataset consisting partially of tasks taken from the 32nd cohort's dataset and partially from self-created tasks on different webpages, taken mostly from the WebVoyager dataset. In the set-up of the test, we further faced the following issue: If the agent chooses one wrong action in an extensive task, all the subsequent actions can be expected to not match the desired actions as well, even though they might be correct. For this reason, we decided to only evaluate the first up to three steps of each run and choose the tasks and starting URLs in such a way that a sufficient variety of actions is covered within these first steps. We point out that evaluating only the first step of each run did not seem feasible mostly due to the action "type", which typically requires a click on a specific input field before it can be carried out. The full dataset we used can be found here.

The results achieved by the proxy-lite agent on our dataset are summarized in the following two tables:

Action	TP + FN	TP + FP	TP	Recall	Precision	F1-Score
Click	20	22	12	0.6000	0.5455	0.5714
Type	10	13	5	0.5000	0.3846	0.4348
Scroll	10	4	3	0.3000	0.7500	0.4286
Return Value	10	10	9	0.9000	0.9000	0.9000

Total predictions	Correct predictions	Wrong predictions	Accuracy
50	29	21	0.5800

5.2.3 Key observations

We notice that the proxy-lite agent achieves a slightly higher overall accuracy of 58% in comparison to the 32nd cohort's agent with an accuracy of 50%. With regards to the actions "Click" and "Type" both agents achieve similar scores, with the 32nd cohort's agent slightly outperforming the proxy-lite agent in the F1-Score.

The most interesting results concern the actions "Scroll" and "Return Value". In case of the "Scroll" action, we notice that the 32nd cohort's agent has not chosen to carry out the action in any task, resulting in a Recall, a Precision and an F1-Score of 0.00%. We also notice, however, that the 32nd cohort's agent was evaluated on a dataset containing only three desired "Scroll" actions and hence this value should not be overinterpreted. While the proxy-lite agent clearly showed stronger results with respect to correctly choosing the "Scroll" action, we notice that this agent also had seven false negative predictions. This is in accordance with the observation that the agent often seems to not consider the option of scrolling, which, for example, appears to be the cause for its inability to bypass many cookiebanners.

A potential explanation for the proxy-lite agent's tendency to not use its "Scroll" action can be found in the agent's scraping functionality. The scraper is designed so that the agent clicks on the first relevant link it encounters rather than evaluating multiple options; i.e. the agent prioritizes immediate interactions rather than searching the page thoroughly. In particular, if the prompt passed to the agent is vague, the agent may struggle to find the right information and proceed by following the predefined behavior of clicking the first clickable object.

For the "Return Value" action, while the 32nd cohort's agent achieves a Precision of 100%, its Recall and F1-Score amount to merely 16.67% and 28.58%, respectively. The high number of false negatives is in accordance with what was observed in the evaluation of the agent's overall runs in Section 5.1.2 above. Indeed, in that benchmark test it was already observed that less thant 20% of the agent's runs were finished, while the remaining runs resulted in errors. The low Recall value recorded in the present benchmark tests seems to confirm that the agent is incapable of recognizing when its goal has been achieved. The proxy-lite agent, in turn, shows particularly strong results with regards to the "Return Value" action, achieving 90% in both Recall and Precision and hence also its F1-Score.

While the proxy-lite agent did not achieve a dramatically higher overall accuracy, its stronger performance in the cases of the "Scroll" and "Return Value" actions might be an indicator of why its runs were observed to be far more successful than the 32nd cohort's agent's attempts.

5.2.4 Potential future research

In the benchmark test there remains room for improvement to achieve more precise results. In particular, the following points can be addressed in potential future research:

- Instead of only evaluating the correctness of the agent's chosen actions, it should also be evaluated (in case of the "Click" action and the "Type" action) at which interactive element the action is performed. For example, most webpages contain more than one interactive button that can be clicked, however, only one of these buttons typically corresponds to the desired action.
- The agents are capable of more actions than the four actions that we evaluated in the above benchmark test, such as for example the action of going back or the action of performing a Google search. While those actions are potentially more difficult to evaluate, they should be included in a complete benchmark test.
- As explained above, the evaluation of individual actions chosen by the agents was carried out on a different dataset for each of the two agents due to the reasons explained in Section 5.2.2.

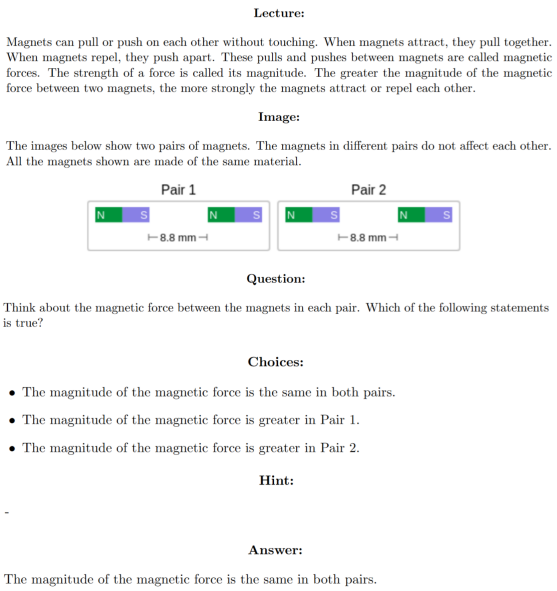

Figure 3: A typical example task from the ScienceQA dataset.

For an exact comparison between the two agents, both agents need to be evaluated on the same dataset.

5.3 (Visual) question answering abilities

The ScienceQA dataset is a comprehensive collection of 21,208 multiple-choice science questions sourced from elementary and high school curricula. Combining up to two textual (*lecture* and *hint*) and a potential additional visual (*image*) resource in each sample, it is designed to test a model's ability to draw conclusions from multiple sources of information. Furthermore, each sample contains a question, between two and five choices and the correct answer. A typical example for a task from the dataset can be seen in Figure 1.

We aimed to evaluate the UI-TARS agent's ability to scratch data from screenshots taken of webpages, to understand the scratched content and to formulate it into a suitable answer to the original prompt. To this end, we tested the agent on a subset of 200 samples from the ScienceQA dataset. In addition, we aimed to carry out a similar benchmark test for the proxy-lite-3b model, which however, cannot process visual input as the proxy-lite agent is designed to navigate through webpages and scrape data via JavaScript instead of via optical character recognition. Thus, for the evaluation of the proxy-lite model, we filtered out samples that include visual data from the ScienceQA dataset and created an alternative dataset consisting of 200 questions related to purely textual information. The models achieved the following accuracy on their respective datasets:

Model	Accuracy
UI-TARS-7b DPO	0.4
proxy-lite-3b	0.395

The agents are approximately on par in their accuracy, however, it is noticable that both of them did not perform much better than pure guessing. Indeed, about 53% of the multiple choice tests in the ScienceQA dataset admit two choices, about 23% admit three and four choices each and approximately 1% admits five choices. This amounts to an expected overall accuracy of 36.8% achieved via pure guessing, which is only about 3% less than both models.

At first glance, these unexpectedly low results seem to contradict the strong performance of both the UI-TARS agent and the proxy-lite agent in webscraping tasks. However, there are possible explanations: While the UI-TARS agent does use OCR techniques to understand screenshots and navigate through webpages and delivers answer based on the knowledge it gains from this, the crucial information in these screenshots is typicall given in the form of text or diagrams. Many of the images in the ScienceQA dataset, instead, are given in the form of actual photos or drawings, which might be harder to understand for the UI-TARS model, as it has not been trained for such tasks.

Furthermore, the questions in the ScienceQA dataset require reasoning skills. These might be neglected in the UI-TARS model and the proxy-lite model as for many webscraping tasks recognizing similarities between the user's prompt and the scraped content can be sufficient to provide a satisfactory answer. For example, when an agent is asked to find a certain interest rate, it is often sufficient to recognize the word *interest rate* in the scraped content and deliver the subsequent number for a correct answer. Consequently, UI-TARS and proxy-lite can perform strongly in their scope of duties as autonomous web agents while at the same time failing the ScienceQA benchmark tests.

In addition to the benchmark test on the ScienceQA dataset, we created further notebooks for evaluating the models on the following similar datasets:

- The Visual Question Answering v2.0 (VQA v2.0) dataset: A dataset for the evaluation of models' abilities to answer questions about given images with a higher focus on visual aspects than the ScienceQA dataset and no additional textual information.
- The COCO Captions dataset: A dataset consisting of images and corresponding captions for the evaluation of models' abilities to correctly describe the content of photos.

The benchmarking notebooks for the latter two datasets have not yet been run as they appear to take up too much disk space for a free Google Colab account. However, we do not necessarily recommend to pay for a Colab Pro account to run these notebooks as there is no reason to expect any insights that have not already been gained from the ScienceQA benchmark tests.

5.4 Further datasets

We moreover evaluated the proxy-lite-3b model on several of the datasets we had used for benchmarking the summarization-, math-, and reasoning-skills. More specifically, we have evaluated the model on 200 tasks of each of the following datasets:

- **GSM8K** - A dataset of linguistically diverse math word problems on elemantary school level.
- **CNN Dailymail** - A dataset of English language news articels, created to evaluate a model's summarization ability.
- **HellaSwag** - A dataset of multiple choice tests testing a model's reasoning skills by asking it to finish sentences in accordance with a given context.

The results the proxy-lite model achieved (in comparison to the UI-TARS model's results) are summarized in the following table.

Model	GSM8K	CNN Dailymail	HellaSwag
UI-TARS-7b DPO	Accuracy: 0.175	Rouge1: 0.142, Rouge2: 0.078, RougeL: 0.104, RougeLsum: 0.117	Accuracy: 0.265
proxy-lite-3b	Accuracy: 0.010	Rouge1: 0.096, Rouge2: 0.047, RougeL: 0.072, RougeLsum: 0.082	Accuracy: 0.265

In this table, for the evaluation of the models' summarization skills on the CNN Dailymail dataset, the Rouge (Recall-Oriented Understudy for Gisting Evaluation) metric is used. In this metric, Rouge-n measures the overlap of n-grams (typically unigrams (n=1, each word) and bigrams (n=2, each pair of consecutive words)) between the generated and reference summaries, capturing lexical similarity. RougeL evaluates the longest common subsequence (LCS) to account for sentence structure and fluency. RougeLsum is a variant of RougeL designed specifically for multi-sentence summaries, applying LCS at the summary level rather than sentence-by-sentence. These metrics help assess how well a generated summary retains key information from the reference while considering both exact matches and structural similarities.

We notice that the results achieved by both UI-TARS and proxy-lite on the datasets GSM8K, CNN Dailymail and HellaSwag are rather low. While the UI-TARS model showed certain mathematical abilities in the GSM8K test (accuracy 17.50%), the same could not be said for the proxy-lite model (accuracy 1.00%). Furthermore, both models performed rather poorly in the summarization test with overall rouge scores of roughly 0.1. The apparent reason for these low rouge scores is that both models showed the tendency of reproducing the original texts almost word by word rather than actually summarizing them. A heuristic explanation for this behavior is that in webscraping tasks it is perfectly fine for an agent to only retrieve the original text, so the models never learned to summarize. In case of the HellaSwag dataset, the results are even worse. An accuracy of 26.50% corresponds almost exactly to the accuracy expected for pure guessing on the multiple choice tests (consisting of four choices each) in the HellaSwag dataset. And indeed, a closer inspection of the models' answers showed that both responded exclusively with option (A) throughout the entire 200 questions. After ruling out the possibility of an error in the setup of the test, we came to the conclusion that this is in accordance with the behaviour observed by the proxy-lite web agent during Google searches: The agent has the strong tendency to click the first the first link it encounters rather than considering further options (cf. Section 5.2.3) - a strategy that can be useful for an autonomous agent focussed on web navigation but terribly fails in multiple choise tests.

In conclusion, it can be said that the GSM8K, CNN Dailymail and HellaSwag datasets are concipated rather for the evaluation of LLMs, which UI-TARS and proxy-lite are not. Hence, there is no contradiction in the fact that both agents perform strongly in webscraping tasks while their models failed these benchmark tests.

6 Conclusion

For the conclusion of this report, we briefly recap the results discussed above. The agents we had considered and their performances can be summarized as follows:

- **The 32nd cohort's agent:** This agent was found to be highly computationally expensive and slow, making it impractical for large-scale web scraping tasks. It exhibited low finish and success rates, with frequent failures and crashes due to excessive resource demands. Due to these limitations, further development was abandoned in favor of more efficient alternatives.
- **The Midscene.js agent:** Tested with UI-TARS as its core model, the Midscene agent struggled with token limitations, dynamic page elements, and incomplete extractions, making it unreliable for real-world financial scraping. Despite initial promise, it frequently encountered errors such as exceeding OpenAI's token threshold and failing to retrieve full webpage data. These shortcomings led to a transition to the UI-TARS Desktop agent.
- **The UI-TARS Desktop agent:** While significantly more capable than Midscene.js, the UI-TARS Desktop agent proved to be resource-intensive, requiring GPU support and Hugging Face endpoints for optimal performance. It excelled in complex web navigation and data extraction, however, its computational overhead and complexity ultimately led the team to seek a more lightweight, web-focused solution.
- **The proxy-lite agent:** This agent emerged as the most efficient choice, offering a lightweight alternative with strong performance in web-based interactions using JavaScript rather than OCR. It outperformed the 32nd cohort's agent in benchmark tests and showed performances comparable to UI-TARS with significantly lower computational costs than both of these agents.

Consequently, our strong recommendation to the 34th cohort is to proceed with the proxy-lite model. This can be done in one of two ways. The first way is to use only the model itself and develop a self-built webagent around it, possibly replicating individual functionalities of the original proxy-lite agent. This has been attempted by our cohort but has not lead to a functioning agent as in particular the web-navigation via JavaScript turns out to be a delicate issue. In case the 34th cohort wants to further pursue this approach, we provide the (unfinished) project here (insert link later).

The second, more feasible option, is to use the original proxy-lite webagent as made available in its Github repository and simply finetune its underlying model on the desired financial scraping tasks. For the finetuning, our cohort further provides some readily available notebooks (insert link later), which, however, have not yet been successfully run due to high computational costs.