M1 Internship Report

DynPol : Dynamic Policy Library for web agents

Martin CUINGNET

This document is the report of my 5 months internship at UCL in London with the Web Intelligence team under the supervision of Aldo Lipani and in collaboration with Jerome Ramos and Bin Wu.
This work is about how to improve LLM-based web agents using self-improvement and policies to allow better planning and lifelong learning.

December 05, 2025

Contents

Acknowledgements

I would like to thank Aldo Lipani for its supervision as well as Jerome Ramos and Bin Wu for taking part in this project with me. I would also like to thank the whole Web Intelligence team for their support as well as NVIDIA, especially Karin Sevegnani for her advices on the project and for the compute power to run experiments.

General Context

Web agents are complex systems designed to solve tasks in a web-environment. Complex tasks like filling out a form or clicking a sequence of buttons to arrive at a peculiar page can be easy for humans but much harder for agents. In a turn-by-turn fashion, for each webpage state, the agent has to find the right action to complete, like clicking a button or typing text, in order to reach its goal. Our objective will then be to design an agent which selects the right action based on a goal and a webpage state.

Studied Problem

My internship was originally centered on Knowledge Editing and Interpretability in Transformers. However, I changed direction to focus on web agents for two reasons: the original topic wasn't well-aligned with my team's interests, and I found myself drawn to the web agents field due to its compelling literature.

The goal of the internship was to solve the problem of lifelong learning in web agents. Indeed, most state-of-the-art agents use a fixed knowledge base that covers most of the tasks and websites the agent will encounter. However, these techniques limit the versatility of the agent (what if the task or the environment falls outside the knowledge base?) and its potential for improvement. We tried to solve these two problems by allowing the agent to learn from its experiences.

Contribution

I tried to solve this problem by creating a new architecture named Dynpol that allows the agent to learn from its mistakes and to divide complex tasks into smaller subtasks while maintaining and improving instructions on how to solve these subtasks. To accomplish this, I designed components like an Automatic Curriculum for task separation and a Self-Improvement Mechanism that allow the agent to improve its own instructions.

I implemented the architecture in Python (see Section 5) and ran numerous experiments throughout the internship to tweak parameters and make design choices that would improve overall performance. I also conducted large-scale experiments to assess the final performance of the model.

Limitations

Although we managed to get a working architecture that shows better performance than our baseline, the agent is still quite token inefficient, due to the way the environment is passed down as well as being unpredictable due to the prompt optimizations techniques used that are very noisy. Moreover, Dynpol performance are still well behind state-of-the-art techniques. Finally, the number of experiments is quite limited due to performance issues and time constraints, which limits the confidence we can place in the final results.

1. Introduction

Recently, LLM have shown very strong performance on many tasks related to language understanding, making them a great choice to tackle complex challenges in a language based environment like the web. (S. Yao *et al.* [1], X. Wang *et al.* [2]). A web environment can be interacted with by an artificial agent that produces actions based on its observations in order to perform tasks autonomously.
Compared to other language-based environments, web-environments are quite challenging due to their complexity and versatility. (S. Marreed *et al.* [3])

In web environments, the goal for an artificial agent is to complete specific tasks using the elements accessible on the webpage in a turn-by-turn context. The agent has to decide at every step which action to take (clicking on a specific button or typing something in a text field). The task the agent has to solve is written in natural language and covers a diverse set of domains such as shopping, forum, maps, etc. (S. Zhou *et al.* [4])

However, web environments are complex systems with diverse structures and they require in-depth knowledge of website structure and component functionality to navigate correctly. Indeed, when interacting with a website, especially with a computer agent that lacks common sense and experience with the web, it is difficult to know the effect of each button and field *a priori*. Moreover, tasks in web environments are also ambiguous because written in natural language and often require many actions in order to be completed, which requires long-term planning. Our goal will then be to design a polyvalent and independent agent that is able to complete tasks on a large variety of websites not known in advance, able to learn on a lifelong scale and to complete long and complex tasks.

Designing such an agent requires the ability to handle ambiguous, unfamiliar websites and a wide range of possible tasks. The agent needs a way to *learn* and understand how to solve these tasks and how to navigate unknown websites.
The primary way to solve this issue is to use a *knowledge base*. This knowledge base can be composed of trajectories (i.e. sequences of actions and observations produced in order to solve a task), whether they are expert trajectories (Z. Chen, M. Li, Y. Huang, Y. Du, M. Fang, and T. Zhou [5]) or produced by the agent itself (H. Su, R. Sun, J. Yoon, P. Yin, T. Yu, and S. Ö. Arık [6], R. Zhang *et al.* [7]). These trajectories can then be recovered and provided to the agent when needed or can be directly used to finetune the agent. External information can also be provided to the agent to improve its comprehension of the environment (Y. Gao *et al.* [8]). Another way to do it would be to use natural language instructions that explain to the agent how to complete specific subtasks, unveiling the inner working of the website architecture like the SteP method : P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald [9].

Moreover, agents can be faced with complex tasks requiring many actions to finish and thus some form of planning in order to keep track of them.
This planning can be handled using notes that the agent can leave to itself (K. Yang *et al.* [10]), making a plan before tackling a task (J. Shen *et al.* [11]) or dividing the task into smaller subtasks each handled semi-independently, as it is done in works like Voyager (G. Wang *et al.* [12]) or SteP (P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald [9]).

While using trajectories can be a good solution to deal with the first difficulty, trajectories are often quite long and token inefficient, making them ill-suited for in-context learning and finetuning. A solution can be to use general instruction to encompass how to solve tasks in a specific environment. However, these instructions have to be generated on the fly from trajectories created by the agent itself in order for the agent to be able to learn and discover in new environments.

While notes are not optimal for long-term planning (not precise enough and wasting an action), plans made in advance can be a great solution, used successfully by many state-of-the-art techniques (S. Marreed *et al.* [3]).

However, this approach has limited compatibility with natural language instructions. Instead, combining subtask planning with instructions allows specific subtask instructions to emerge organically.

These two techniques (dividing tasks into simpler subtasks and providing subtask specific instructions as a knowledge base) were used in the SteP paper (P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald [9]). This architecture uses a handcrafted set of subtask instructions (called *policies*, as used in Reinforcement Learning M. Ghasemi and D. Ebrahimi [13]) stored in a *policy library*. The agent can then call these policies when needed and handle them as a stack (if a policy calls another policy, the deeper policy is the one being executed, i.e. the agent uses its instructions).
The SteP *static* policy library was handcrafted by the SteP team and contains guidance for 14 subtasks with roughly 3 policies per website.

Despite its interesting architecture, SteP stays quite limited in terms of adaptability to new environments because its policy library is *static* and its policies *handcrafted*.

This is why we introduce Dynpol, a new architecture based on SteP that uses subtask decomposition and natural language instruction created from its own trajectories, all handled in a *dynamic* policy library. Instead of having a fixed number of policies, we will allow the agent to create new policies when needed and improve upon existing policies. The agent will then be able to adapt to new environments and better their performance by correcting itself.

Drawing inspiration from Voyager's approach (G. Wang *et al.* [12]) to lifelong learning in embodied agents, we introduce a novel framework adapted for the distinct challenges of autonomous web navigation. Our framework integrates two symbiotic components: an *Automatic Curriculum* that decomposes complex web tasks to prospectively generate novel policies, and a *Self-Improving Mechanism* that leverages execution feedback to iteratively refine and enhance the instructions of each policy.

2. Method

The goal of Dynpol is, based on the current environment, to issue actions to achieve a specific objective on the website. The agent has an internal state and carries informations between tasks (*lifelong memory*)

To this end, Dynpol features 3 main components :
- The **Main Loop** (notably composed of a *policy stack* and a *dynamic policy library*)
- The **Self-Improvement Mechanism**
- The **Automatic Curriculum**

The core principles behind this architecture where layed down at the end of the first month of the internship after reading papers like SteP (P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald [9]) or Voyager (G. Wang *et al.* [12]). The components were refined during the whole duration of the internship by trying different alternatives (examples of scrapped ideas : call the Automatic Curriculum every time a policy finishes, do not use a critique LLM, provide the whole trajectory to the rewriting model,...). The architecture as presented here was truly settled around May.

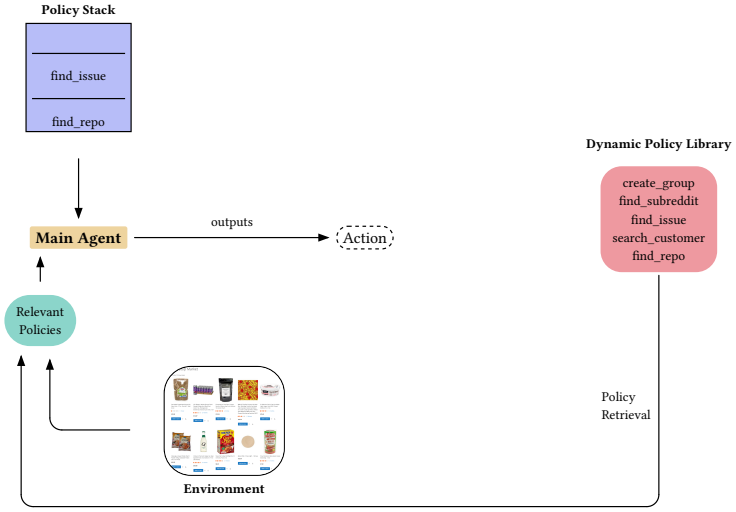
2.1. Main Loop

Based on his environment and available polices, the agent issues actions. These actions can either interact with the environment or change the state of the *policy stack* and the *policy library*.

In reinforcement learning, a policy defines an agent's strategy for selecting actions given environmental states to maximize expected rewards. Similarly, a *policy* here represents dedicated instructions for solving specific subtasks, such as searching lists or navigating web environments.

These policies are stored in the *policy stack* (a policy can call another policy), whereas the instructions of the policy on top of the stack (currently active policy) are provided to the **Main Agent** to choose which action to take.

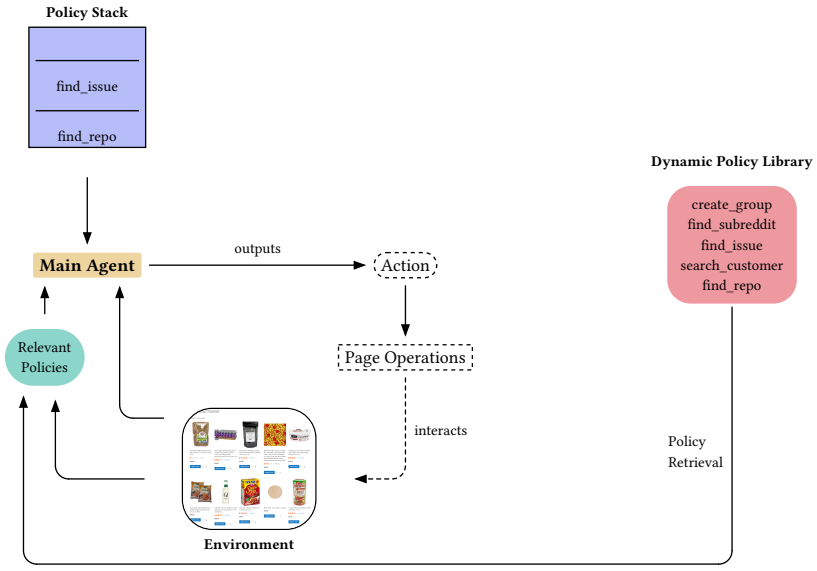
All of these policies are stored in *Dynamic Policy Library* and can be stored and reused at anytime during the interaction.



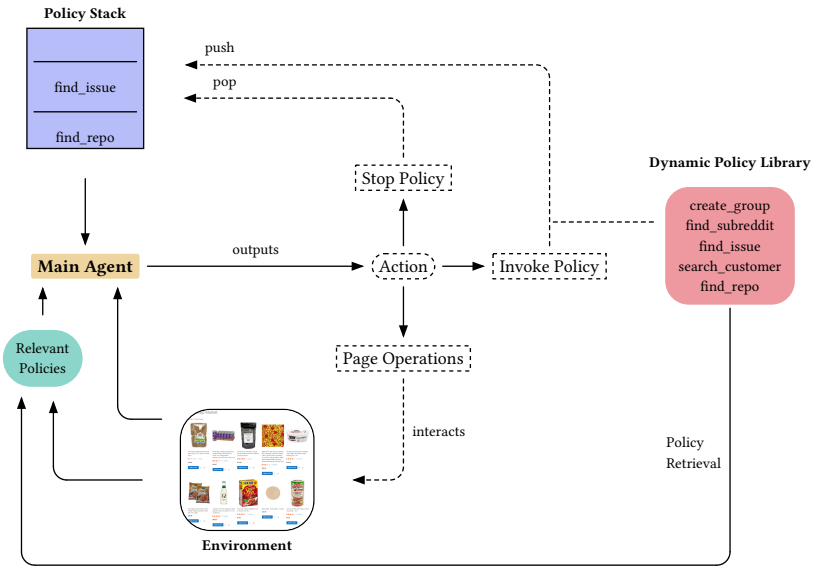
To issue an *action*, the **Main Agent** takes information from its *environment* and *internal state*. Based on the web-environment, the policy on top of the *policy stack* and the relevant polices extracted from the *policy library*, the **Main Agent** *outputs* an action that can fall into three categories.

Relevant Policies are selected from the *policy library* by identifying which policies most closely match the current task. This matching process works by converting both the task description and each policy into text embeddings (numerical vectors), then selecting policies with the smallest vector distances to the task.

The first category of action is a *Page Operation*. This action changes the environment by *interacting* with one of its elements. For instance : `click [420]` or `type [309] [Carnegie Mellon]` where `[320]` is the id of the element on the webpage.



The agent can also *Invoke a policy* from the *Relevant Policies*, directly from the *policy library*. This policy will be added on top of the *policy stack* and can be used in the next actions.
When the agent deems that the policy is finished, it can *Stop the Policy*, popping it off the *policy stack* and providing an answer to the subtask if needed like `stop [42km]`.



After outputting an action, the internal state or the environment of the agent changes and is then used to output the next action. The agent keeps choosing the next action until outputting a *Stop Policy* action when the policy stack is empty.

Every time a policy is popped off the stack, the agent runs the **Self-Improving Mechanism** in order to improve or populate the policy. Based on the trajectory and previous instructions, this mechanism will provide new instructions to avoid future mistakes using this policy.

Finally, before starting any task, the agent uses the **Automatic Curriculum** in order to plan the next trajectory and to add any new policy to the *policy library* if needed.

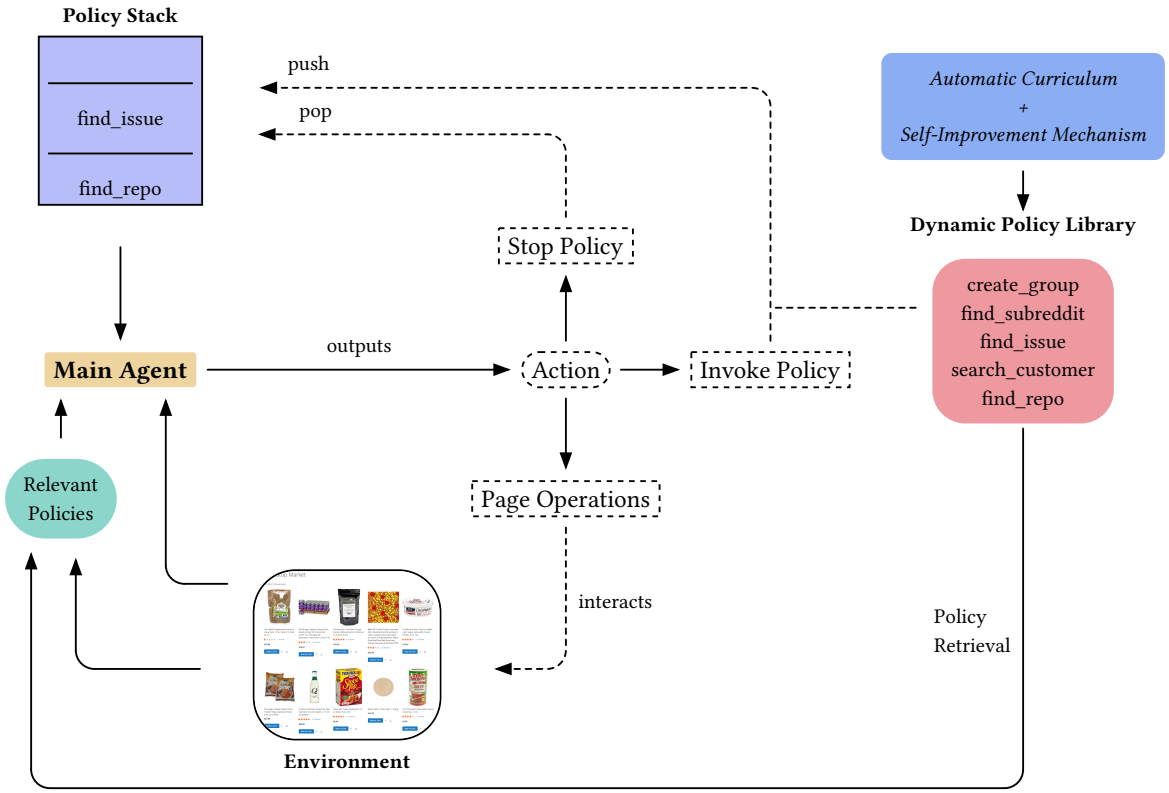
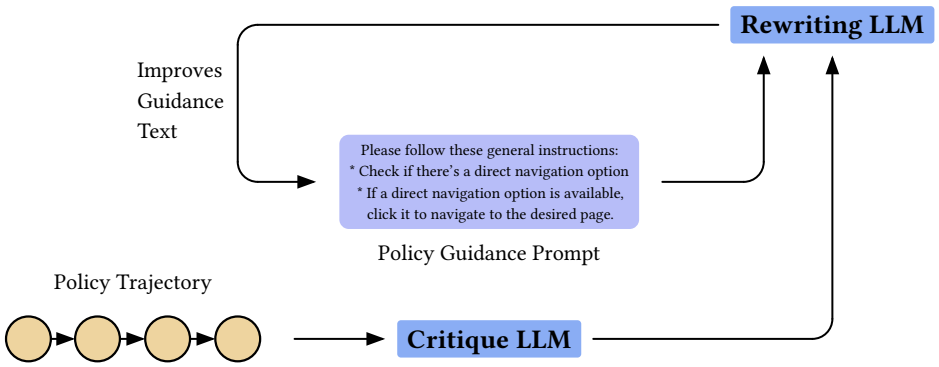


Figure 1: Dynpol Main Loop

You can find a pseudo-code algorithm of the main loop here : Appendix 10.A.

You can see the exact prompt used by the agent to get the next action here : Appendix 10.H.C.A.

2.2. Self-Improvement Mechanism



This component aims at actively improving the quality and accuracy of every policy that is used by the agent.

Figure 2: Self Improvement Mechanism

Based on the current trajectory of the policy (i.e. every action taken when the policy was on top of the stack), and a critique of this trajectory in regards to the policy objective, a **Rewriting LLM** aims at improving the instructions of the policy to yields a better success rate.

The critique is produced by a **Critique LLM** that has to decide if the trajectory was a success or not and identify the main actions of the trajectory that were critical to its success or failure. The **Rewriting LLM** can then focus on these actions to improve the guidance on the most critical points.

Each time a policy is used by the agent, depending on the success rate of the policy (for instance if a policy fails too often), the instructions will be refined by the **Self-Improvement Mechanism**.

You can see the relevant prompts here : Appendix 10.H.C.C and Appendix 10.H.C.D.

2.3. Automatic Curriculum

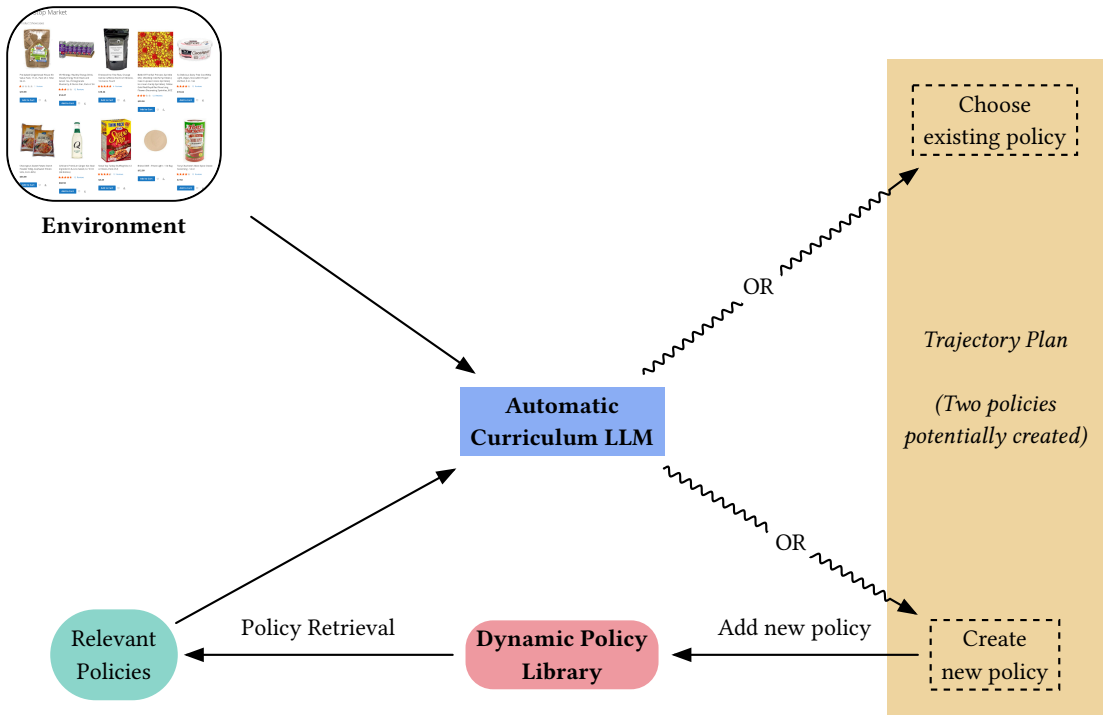


Figure 3: Automatic Curriculum

Based on the state of the starting webpage, the objective of the task and the *relevant polices* (gathered in the same way as in the **Main Loop**), the **Automatic Curriculum LLM** produces a plan to solve the task, alongside key policies that have to be used to follow it, whether they are among the *relevant policies* or not.

If one of the policies chosen by the LLM is not in the *relevant policies* list, it is a sign that the *policy library* is missing an important policy to solve a task. This new policy is thus added to the *policy library* and its instructions will be later on populated by the **Self-Improvement Mechanism** when the policy will be called.

You can see the prompt for the automatic curriculum here : Appendix 10.H.C.B.

2.4. Workflow

Thanks to the **Self-Improvement Mechanism** and the **Automatic Curriculum**, Dynpol can improve over time through experience. By making the agent interact with a variety of tasks and environment, the agent will develop a diverse and efficient toolbox of policies to face many situations.

Consequently, to create the best possible policy library, and thus the best agent, we need to run the Dynpol agent on many tasks, and so multiple times in order to create and refine useful policies. In this work, we use **WebArena** (see Section 4), a web agent benchmark that contains 812 tasks across 5 different types of websites.
Starting from a default policy library (for instance an empty one or the policy library from SteP), the agent is ran multiple times on a fixed set of tasks sampled randomly in WebArena.

During the experiments, we chose to run the agent on the same set of tasks 3 times. This makes it easier for the agent to refine the policies and allow for better tracking of the agent's performance.

3. Formalization

Let's model how an agent interacts with its environment as a sequential, step-by-step process. Initially, the agent has an internal state I_0 and it's environment is in a state e_0. The environment is composed of the objective of the task and the current state of the webpage (an observation). At each step of the process, the agent issues an action a_t that will change the internal state to I_{t+1} and environment state to e_{t+1} state.
Therefore, a trajectory consists of an alternating sequence of states and actions:

$$T = (s_0, (I_0, e_0), ..., a_{n-1}, (I_n, s_n))$$

Let's now detail the components of the internal state I :

1. The first component of the internal state of the agent is the policy stack $\Gamma = [\pi_0, ..., \pi_j]$.

   Each policy π has :
   - A name
   - A textual description
   - Instructions that the agent has to follow
   - A query (the sub-objective that the policy needs to achieve like `create_issue [Button not working]`)
   - The history h of actions taken within the policy scope

2. The second component is the policy library Π which is a dictionary $(k : \pi)$ with k being the embedding of π. It contains every policy that can be used by the agent.

3. The last component are the relevant policies for the current environment $R = [\pi'_0, ..., \pi'_k]$. It is created by selecting the top k polices from Π with the embeddings closest to the current query embedding.

The currently active policy (i.e. : the policy giving instructions to the **Main LLM**) is the one at the top of the stack. The policy stack is initialized with an generic root policy that only contains the task's objective as a query.

At each time-step, the action a_t that the agent can perform fall in three categories :

1. Page Operation

A page operation (`click [328]` for instance) will change the environment state $e_t \xrightarrow{a_t} e_{t+1}$ and update the history of actions of the top policy :

$$\Gamma = [\pi_0, ..., \pi_j] \xrightarrow{a_t} [\pi_0, ..., \pi_j < h \leftarrow h.(a_t, s_{t+1}) >]$$

2. Policy Call

The agent can call a new policy π_{j+1} from the relevant policies R along with a query q' and add it on top of the stack :

$$\Gamma = [\pi_0, ..., \pi_j] \xrightarrow{a_t} [\pi_0, ..., \pi_j, \pi_{j+1} < q \leftarrow q' >]$$

3. Policy Termination

The agent can terminate the top policy and return an answer (internally represented as `stop [answer]`) which will be added to the history of the next policy, abstracting the behavior of the top policy :

$$\Gamma = [\pi_0, ..., \pi_j, \pi_{j+1}] \xrightarrow{\text{stop [answer]}} [\pi_0, ..., \pi_j < h \leftarrow h.(\text{stop [answer]}, s_t) >]$$

Self-Improvement Mechanism :
After a policy has failed too many times or succeeded for the first time, its content is rewritten to improve future performance based on the trajectory :

$$\pi \leftarrow \text{Writing}(T, \pi, \text{Critique}(T, \pi))$$

Automatic Curriculum :
Based on the environment e and the relevant policies R, the Automatic Curriculum makes a plan to solve the task along with the policies needed along the way. These policies are either taken from R or new and added to Π.

4. Testing Environment

The testing environment is a self-hosted web environment called WebArena (S. Zhou *et al.* [4]).

This environment aims at emulating real-life websites (from a website architecture standpoint as well as the data populating them) in a closed and controlled environment. Once self-hosted, the 5 websites of the environment can then be accessed normally from any web browser.

In practice, the WebArena instance is ran using the docker environment provided by the WebArena team. This docker environment was ran and setup by Jerome Ramos on an AWS server.

Here are the 5 websites handled by the WebArena instance :
1. A shopping site
2. A shopping admin panel website
3. A map website
4. A Reddit-like forum
5. A GitLab instance

During most of the internship, the map website was broken due to a dependency on Carnegie Mellon University's websites.

Additionally to being a standalone web environment, WebArena is also a web agent benchmark. Indeed, dispatched among the five websites, WebArena provides 812 tasks consisting of a natural language objective and

a starting URL. WebArena also provides the expected answers to these tasks (although some may be imperfect due to the inherent ambiguity of natural language) as well as functions to assess the success of a trajectory.

See Appendix 10.G for examples of tasks in WebArena.

At each step of a trajectory, the agent has access to the state of the website (the axtree : a parsed and simplified HTML representation, an annotated screenshot,...) and can output an action (click on a specific element identified using a specific ID, type in an element, go back, scroll,...).

5. Technical Details and Implementation

To generate the embeddings, we used the `text-embedding-004` model using the Gemini API, which based on a string, outputs a 768-dimensional vector encompassing its semantic.

Multiple LLMs where used during the testing phase. However, all LLMs calls in the final experiments where done using `Llmama 3.3 Instruct 70B`.

Except for Gemini related models, all models used during the internship were ran on MareNostrum, the supercomputer of the Barcelona Supercomputing Center thanks to credits provided by Nvidia for this project on web agents. The models' weights were loaded on the cluster and executed using `sbatch` and a custom script. I spent quite some time understanding the specific prarameters needed for the `sbatch` custom script (especially considering that the models used take quite some time to load due to their important number of weights making the back and forth pretty slow).
All the LLM queries were done using direct HTTP requests to MareNostrum using a reverse SSH tunnel from my computer to the cluster.

Two main libraries were used, `Gymnasium` for handling the web environment and `BrowserGym` for providing useful functions designed for web agents in mind. The codebase is hosted on this Github repo : https://github.com/mcngnt/DynamicPolicyLibrary/° and is written entirely in Python. The project consists of around 2100 lines of codes and prompts and was improved and refined from April until the end of the internship (see Appendix 10.B)

6. Experiments

To assess the performance of our architecture, we ran Dynpol multiple times on a set of tasks, along with other architectures that serve as baselines for comparison.

6.1. Baseline

6.1.1. Single LLM

Here we only use a single prompt to directly generate the next action based on the environment (see the prompt used here : Appendix 10.H.D).

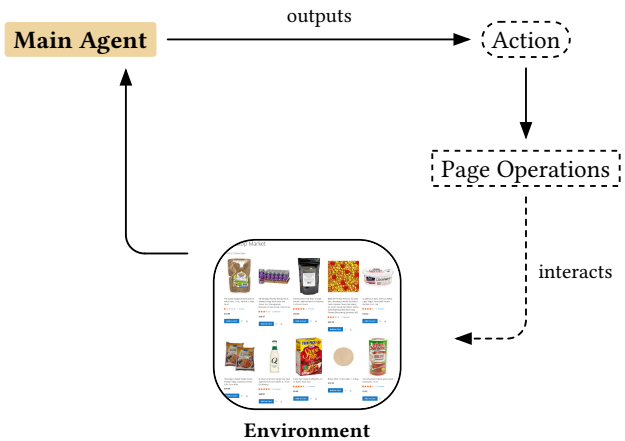


Figure 4: Single LLM Architecture

6.1.2. SteP

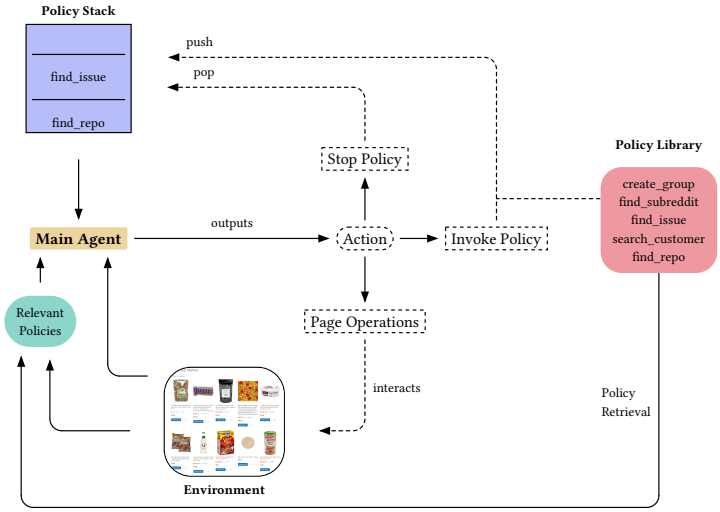


Figure 5: SteP Architecture

By ablating the Dynpol architecture from the **Self-Improvement Mechanism** and the **Automatic Curriculum**, initializing the *policy library* to the library of SteP and making it static, we are able to replicate the SteP architecture. For the sake of comparison, we used the same prompts as in the original paper and used the same LLM as Dynpol (all architectures are using `Llama 3.3 Instruct 70B`)

6.1.3. Dynpol Without Automatic Curriculum

To test the usefulness of the **Self-Improvement Mechanism** by itself, we instantiated the Dynpol agent with the SteP policy library and without the **Automatic Curriculum**, thus preventing the creation of new policies. This agent will only try to improve the original SteP policies.

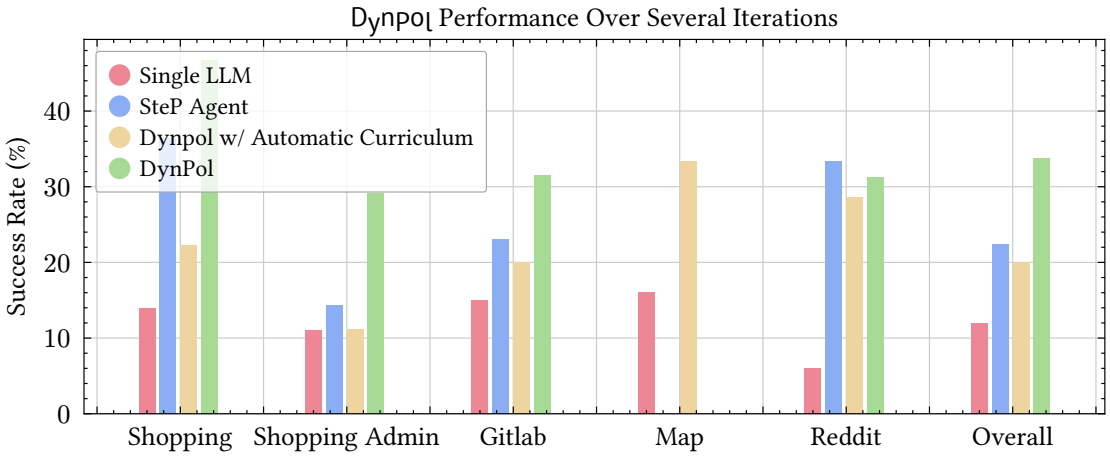
6.2. Results

Each architecture is ran on a set of 50 randomly sampled tasks from WebArena, except for Dynpol which is ran on 80 randomly sampled tasks.

Each architecture is executed once per task from the sampled set, except for Dynpol without Automatic Curriculum and Dynpol itself, which are each ran 3 times per task. The results shown below are the performance of the architectures at the final and best iteration.

Tasks	Single LLM	SteP Agent	DynPol without Automatic Curriculum (3 iterations)	DynPol (3 iterations)
Shopping	0.14	0.36	0.22	**0.47**
Shopping Admin	0.11	0.14	0.11	**0.29**
Gitlab	0.15	0.23	0.2	**0.32**
Map	0.16	0	**0.33**	0
Reddit	0.06	0.33	0.29	**0.31**
Overall	0.12	0.22	0.2	**0.34**

Table 1: WebArena Success Rates for 50 (or 80) randomly sampled tasks for several architectures



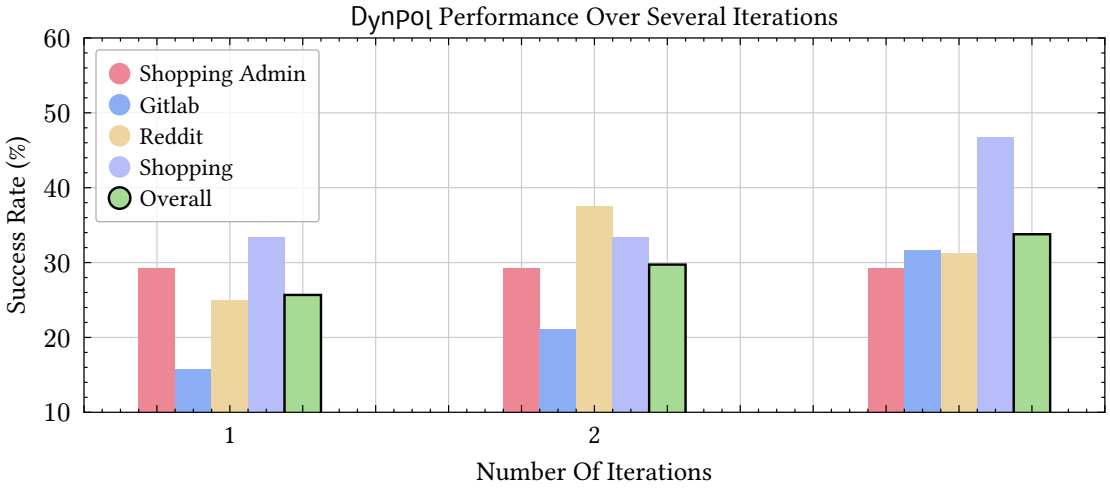
As mentioned in Section 4, the Maps website was down for most of the internship, which made certain experiments on Maps (such as the final experiments on Dynpol) impossible to conduct. The overall scores therefore exclude Maps whenever the website was unavailable.

As shown in Table 1, Dynpol outperforms prior work using the same backbone LLM (`Llama 3.3 Instruct 70B`). The most significant improvements are observed in Shopping (+11%) and Shopping Admin (+15%), where it surpasses the baseline without relying on demonstration trajectories, unlike SteP. These results highlight that the Automatic Curriculum, combined with the Self-Improvement mechanism, enable the agent to significantly outperform both SteP and Dynpol without the Automatic Curriculum. This also shows that the Self-Improvement mechanism alone is not particularly effective.

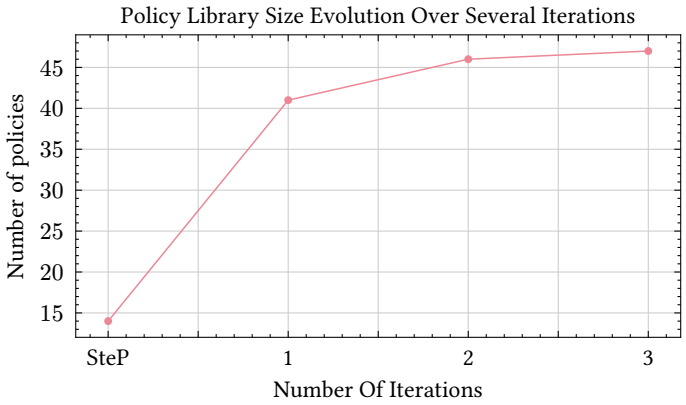
The poor performance of Dynpol without the Automatic Curriculum is likely due to the fact that the SteP policies initializing the library are already highly optimized (handcrafted by the SteP team). In such cases, the Self-Improvement mechanism may actually degrade performance rather than enhance it.

7. Analysis

7.1. Dynpol Evolution Over Several Iterations



As the number of iterations increases, the agent's overall performance improves accod. However, some tasks do not show consistent improvement (such as Reddit), likely due to insufficient sample sizes (for instance, Reddit only has 16 samples).



The number of policies in the *policy library* increases dramatically after the first iteration due to the Automatic Curriculum feature. After this initial surge, this number stabilizes because the Automatic Curriculum recognizes previously encountered tasks and reuses existing policies rather than creating new ones.

7.2. Policy evolution over time

As we have seen, the number of policies stabilizes after the first iteration, meaning almost no policy were added, limiting the impact of the Automatic Curriculum. Nevertheless, the agent continues to improve between iterations 2 and 3, indicating that the Self-Improvement mechanism enhanced the quality of the policies in the policy library.

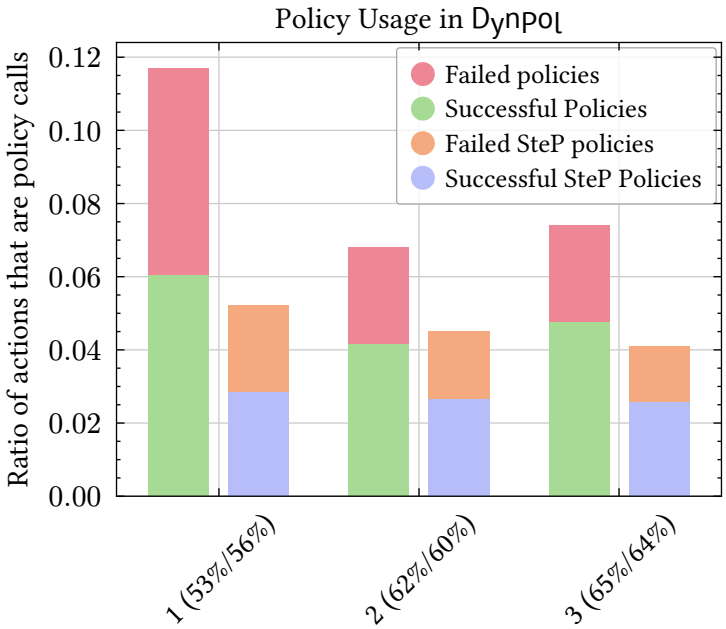
Our qualitative analysis shows that the Self-Improvement mechanism strengthens and refines the policies in three main ways:

- Providing more detailed guidance (e.g., clarifying semi-obvious parts of a trajectory)
- Preventing mistakes observed in previous trajectories (see Appendix 10.H.B.A)
- Adding examples, often based on the trajectory used for rewriting (see Appendix 10.H.B.B)

7.3. Policy usage by the agent

As expected, the improvement shown in Table 1 is also reflected in the success rates of the policies themselves in regard to the subtask they try to tackle. To determine whether a policy is successful or not, we rely on the Critique LLM's assessment, which provides a rough estimate of whether the policy actually succeeded or not.

After the first iteration, the success rate increases dramatically, demonstrating the effectiveness of the Self-Improvement mechanism. We can see that the Self-Improvement mechanism also works with SteP policies, as they show a similar surge in success (even though the results of Dynpol with only the Self-Improvement mechanism were inconclusive).



Number of iterations alongside the success rates of the policies (Left : All Policies, Right : SteP Policies)

You can find additional analysis here : Appendix 10.C.

8. Limitations and Future Work

8.1. Limitations

Despite all the experiments and optimization, this work presents many limitations:

- **Performance gap due to different LLMs**: While Dynpol surpasses the SteP technique in our experiments, the original SteP paper P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald [9] tells a different story. The success rate reported in that paper is 33%, significantly higher than the 22% presented in our work. This difference stems from the use of different LLMs in the two experiments. In this work, we used `Llama 3.3 Instruct 70B`, whereas the SteP paper (and many others) used `GPT-4` or its variants, partially explaining this performance gap. This gap is even more visible on leaderboards where all methods using better LLMs easily surpassing the 34% success rate of Dynpol.
- **Unreliability of LLM-driven prompt optimization**: Our technique relies heavily on LLM-driven prompt optimization. While this approach works well in languages with very restrictive syntax rules (like code, see G. Wang *et al.* [12]), it can be very noisy and unreliable when applied to natural language (i.e., the guidelines of the policy). This can lead to great variability in performance. (See M. Yuksekgonul *et al.* [14])
- **Catastrophic forgetting in self-improvement**: Related to this noisiness, the self-improvement mechanism can easily discard useful information within policies' guidelines, leading to a catastrophic forgetting phenomenon.
- **Limitations of text-only approach**: Our current technique is purely textual, only taking advantage of the accessibility trees (axtrees). However, with complex and dense websites, the axtree can grow very rapidly, sometimes reaching over 30K tokens. With such enormous axtrees, the LLM mainly spends time deciphering the axtree instead of focusing on determining the best action to solve the current objective.
- **Time-intensive execution**: Finally, due to the huge size of axtrees, the performance overhead from working with a web browser, and the high number of steps needed per task (15 actions taken per trajectory on average), completing even a single task can be quite time-intensive (around 2-3 minutes in the current setup). This limited the number of experiments that could be conducted, leading to greater uncertainty in the results presented.

8.2. Future Work

- We first plan to conduct significantly more experiments to gain greater confidence in our results and the performance of Dynpol. We eventually plan to run the agent on every task of WebArena to reach the quality standard required to develop this work into a publishable paper.
- Additionally, we aim to add a visual component to Dynpol to reduce our complete dependence on the textual accessibility tree. Indeed, using techniques like Y. Lu, J. Yang, Y. Shen, and A. Awadallah [15] will allow the agent to access the same information available in the axtree in a denser and more efficient manner.
- Finally, we plan to continue improving the prompts of the different LLMs used by the agent as well as the prompts within the policy library by providing additional examples. We could, for instance, use in-context learning by adding examples to policy prompts extracted from previous trajectories.

9. Conclusion

In this work, we introduce Dynpol, a new architecture for web agents, leveraging prompt optimization techniques with experimentation to create an aligned and optimized dynamic policy library that can be tailored easily for any tasks or website. These polices are used in a stack as a natural solution to the task subdivision problem while enabling the agent to reduce noisiness in its judgement.

This internship provided me with valuable experience working in a research team, primarily alongside two PhD students, with regular progress tracked through bi-weekly meetings. During this period, I developed a novel agentic architecture and evaluated its performance. I had the opportunity to aquire useful technical knowledge on LLM and discover the previously unfamiliar field of agentic AI.

Bibliography

[1] S. Yao *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models." [Online]. Available: https://arxiv.org/abs/2210.03629

[2] X. Wang *et al.*, "Executable Code Actions Elicit Better LLM Agents." [Online]. Available: https://arxiv.org/abs/2402.01030

[3] S. Marreed *et al.*, "Towards Enterprise-Ready Computer Using Generalist Agent." [Online]. Available: https://arxiv.org/abs/2503.01861

[4] S. Zhou *et al.*, "WebArena: A Realistic Web Environment for Building Autonomous Agents." [Online]. Available: https://arxiv.org/abs/2307.13854

[5] Z. Chen, M. Li, Y. Huang, Y. Du, M. Fang, and T. Zhou, "ATLaS: Agent Tuning via Learning Critical Steps." [Online]. Available: https://arxiv.org/abs/2503.02197

[6] H. Su, R. Sun, J. Yoon, P. Yin, T. Yu, and S. Ö. Arık, "Learn-by-interact: A Data-Centric Framework for Self-Adaptive Agents in Realistic Environments." [Online]. Available: https://arxiv.org/abs/2501.10893

[7] R. Zhang *et al.*, "Symbiotic Cooperation for Web Agents: Harnessing Complementary Strengths of Large and Small LLMs." [Online]. Available: https://arxiv.org/abs/2502.07942

[8] Y. Gao *et al.*, "Retrieval-Augmented Generation for Large Language Models: A Survey." [Online]. Available: https://arxiv.org/abs/2312.10997

[9] P. Sodhi, S. R. K. Branavan, Y. Artzi, and R. McDonald, "SteP: Stacked LLM Policies for Web Actions." [Online]. Available: https://arxiv.org/abs/2310.03720

[10] K. Yang *et al.*, "AgentOccam: A Simple Yet Strong Baseline for LLM-Based Web Agents." [Online]. Available: https://arxiv.org/abs/2410.13825

[11] J. Shen *et al.*, "ScribeAgent: Towards Specialized Web Agents Using Production-Scale Workflow Data." [Online]. Available: https://arxiv.org/abs/2411.15004

[12] G. Wang *et al.*, "Voyager: An Open-Ended Embodied Agent with Large Language Models." [Online]. Available: https://arxiv.org/abs/2305.16291

[13] M. Ghasemi and D. Ebrahimi, "Introduction to Reinforcement Learning." [Online]. Available: https://arxiv.org/abs/2408.07712

[14] M. Yuksekgonul *et al.*, "TextGrad: Automatic "Differentiation" via Text." [Online]. Available: https://arxiv.org/abs/2406.07496

[15] Y. Lu, J. Yang, Y. Shen, and A. Awadallah, "OmniParser for Pure Vision Based GUI Agent." [Online]. Available: https://arxiv.org/abs/2408.00203

10 Appendix

10.A Pseudo Code of DynPol

Here is a pseudo-code describing the behavior of DynPol in more details :

```python
def get_action(observation, objective, is_first_step):
    if is_first_step:
        # Initialize the policy stack with a site-dependant policy
        policy_stack = Stack.init(root_policy)
        # Automatic Curriculum, seeing if new policies need to be added to the policy library
        relevant_policies = policy_library.retrieve(objective)
        # LLM Call to choose which policies to use
        chosen_policies = automatic_curriculum(objective, observation, relevant_policies)
        for policy in chosen_policies:
            if not (policy in policy_library):
                policy_library.add(policy)

    current_policy = policy_stack.top()
    relevant_policies = policy_library.retrieve(objective)
    # LLM call to determine which action to take
    action = get_action(objective, observation, current_policy, relevant_policies)

    match action.type:
        case PAGE_OPERATION:
            env.interact(action)
        case POLICY_CALL:
            policy_stack.push(action.policy)
        case STOP:
            finished_policy = policy_stack.pop()
            # Self-Improvement Mechanism
            critique = get_critique(finished_policy.trajectory, objective)
            policy_library.update_usage(critique.is_success)
            if rewrinting_needed(policy_library.get_usage(finished_policy)):
                new_guidance = rewrite(finished_policy.guidance, critique, objective, finished_policy.trajectory)
                policy_library.update(finished_policy, new_guidance)
```

10.B Code Repartition

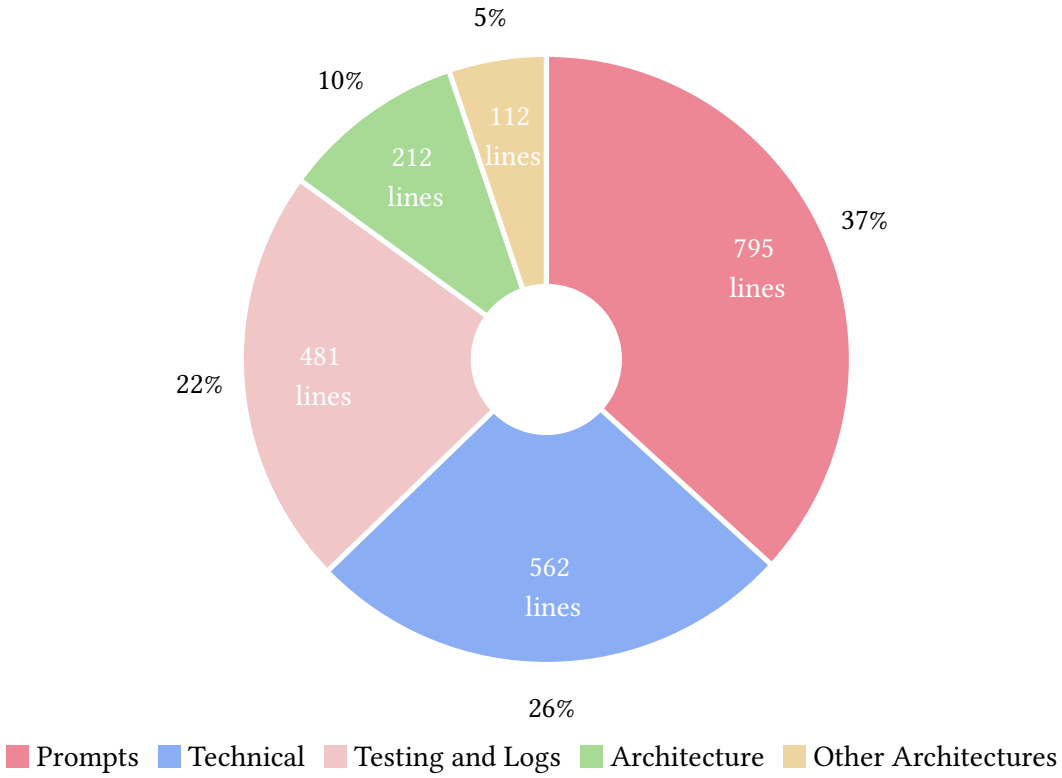
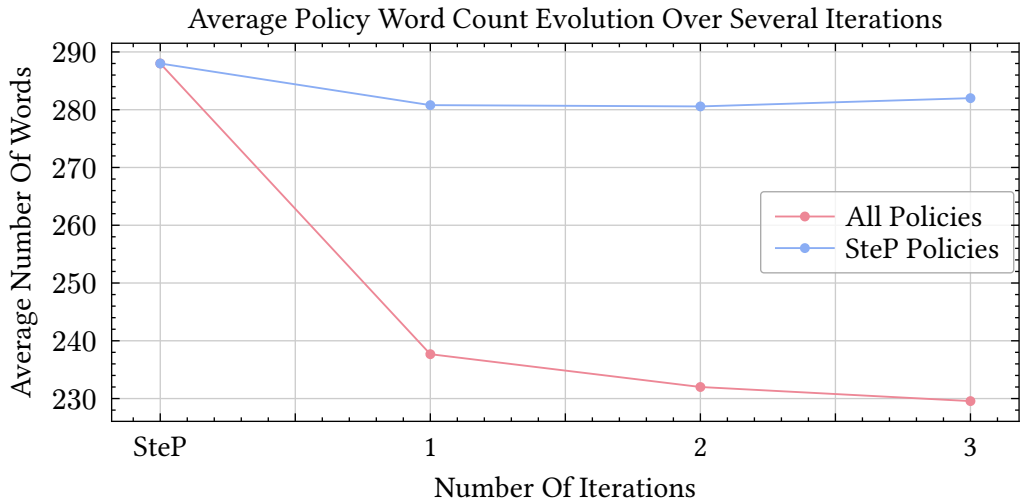


Figure 6: Repartition of the lines inside the codebase

10.C Miscellaneous Analysis

10.C.A Policy size



The size of the policies inside the policy library differ significantly between the policies originating from the SteP papers and the ones created by the Automatic Curriculum. The difference comes from the fact that most SteP policies contain example interactions. These examples are hard to create *ex nihilo* by the Rewriting LLM, especially because it has not access to whole trajectories because of token limits (again caused ny the axtree).

10.C.B Why a dynamic policy library helps over a static library ?

As mentioned in the Agent Occam paper (K. Yang *et al.* [10]), static policies can be misleading for web agents due to a misalignment compared to the policies and the objectives. In fact, the Occam paper found out that that policies can hinder performance when compared to techniques like LLM-as-a-judge.

This misalignment issue was mitigated in DynPol thanks to the Automatic Curriculum and the Self-Improvement mechanism. The Automatic Curriculum allows the agent to develop or select policies that are tailored to the specific tasks at hand, while the Self-Improvement mechanism enables direct re-alignment of a policy based on feedback from its own trajectories.

This can be directly seen in the results comparison in Table 1.

10.D Failed Experiments

10.E Agent not using policies

Converlsy to the SteP method where the policy library is static and so the available policies are known in advanced and directly integrated to the prompt, the policies with DynPol are generated on the fly and harder to correctly integrate to the prompt. Indeed, because the policies are dynamic, its harder to give good examples of how and when to use them. This led to versions of DynPol where the agent would never called any puucsolicies and rely solely on page operation. This was a problem because it made the Automatic Curriculum and the Self-Improvement Mechanism useless, essentially making the agent behave as a single LLM.

10.F LLM not large enough

During the initial experimentations we also tried to use smaller LLMs for quicker inference during the prompt engineering phase. For instance, we tried using the `Llama 3.3 Nemotron 49B`. However these smaller LLMs had real trouble to fullfill the tasks in DynPol, even following the required formatting (the output of the LLMs have to be parsed before being used inside the architecture). The bottleneck probably lies inside the enormous size of the axtree describing the HTML content fed to the LLM to choose an action (probably due to attention problems)

10.G Task Examples

Task ID	Website	Task Objective
181	Gitlab	Open my latest created issue that has theme editor in its title to check if it is closed
298	Shopping Admin	Show the most recent completed order
386	Shopping	What is the rating of Ugreen lightning to 3.5mm cable. Please round to the nearest whole number
643	Reddit	Post a notice on a virtual meetup for racing cars enthusiasts on Oct 21st in the nyc subreddit
767	Maps	Find the walkway to the closest chain grocessory owned by a local business from 401 Shady Ave, Pittsburgh.

10.H Prompts

10.H.A Example of Policy Instructions

Here is an example of guidance provided by a policy after three passes on the same set of tasks. This guidance is for the policy `find_latest_post_and_author` which aims at `Find the latest post and extract its author` :

Based on the plan and explanation, the guidance for the web agent to solve the task is:
* To find the latest post related to a specific topic, start by examining the search results page and looking for the most recent timestamp.
* Use the website's search function or filtering capabilities to narrow down the search and highlight the latest post. Look for options to sort posts by timestamp in descending order (newest first).
* If sorting or filtering options are available, use them to highlight the latest post.
* Clearly identify the latest post and its author by looking for the username associated with the post.
* Verify that the highlighted post is indeed the latest one related to the topic by checking the timestamp and ensuring that the post is relevant to the search query.
* Provide clear evidence in the observation that the task has been completed by highlighting the latest post and its author.

Examples:
* If the website has a search box, type in the keyword to find posts related to the topic, and then look for the most recent timestamp to identify the latest post.
* If the website has filtering options, select the option to filter posts by keyword or topic, and then select the topic to find related posts. Look for options to sort posts by timestamp in descending order.
* If the website displays user posts in a list, look for the most recent timestamp to identify the latest post related to the topic, and then extract the author's username.
* If the website displays user posts in a grid, look for the post with the most recent timestamp, extract the author's username, and verify that the post is relevant to the search query.

10.H.B Comparison of Policies after Self-Improvement

10.H.B.A Preventing previous mistake

Here is a comparison of the evolution of the `filter_and_sort_issues` policy.

The goal of this policy is to sort the issues of a GitLab repo based on predetermined tags.

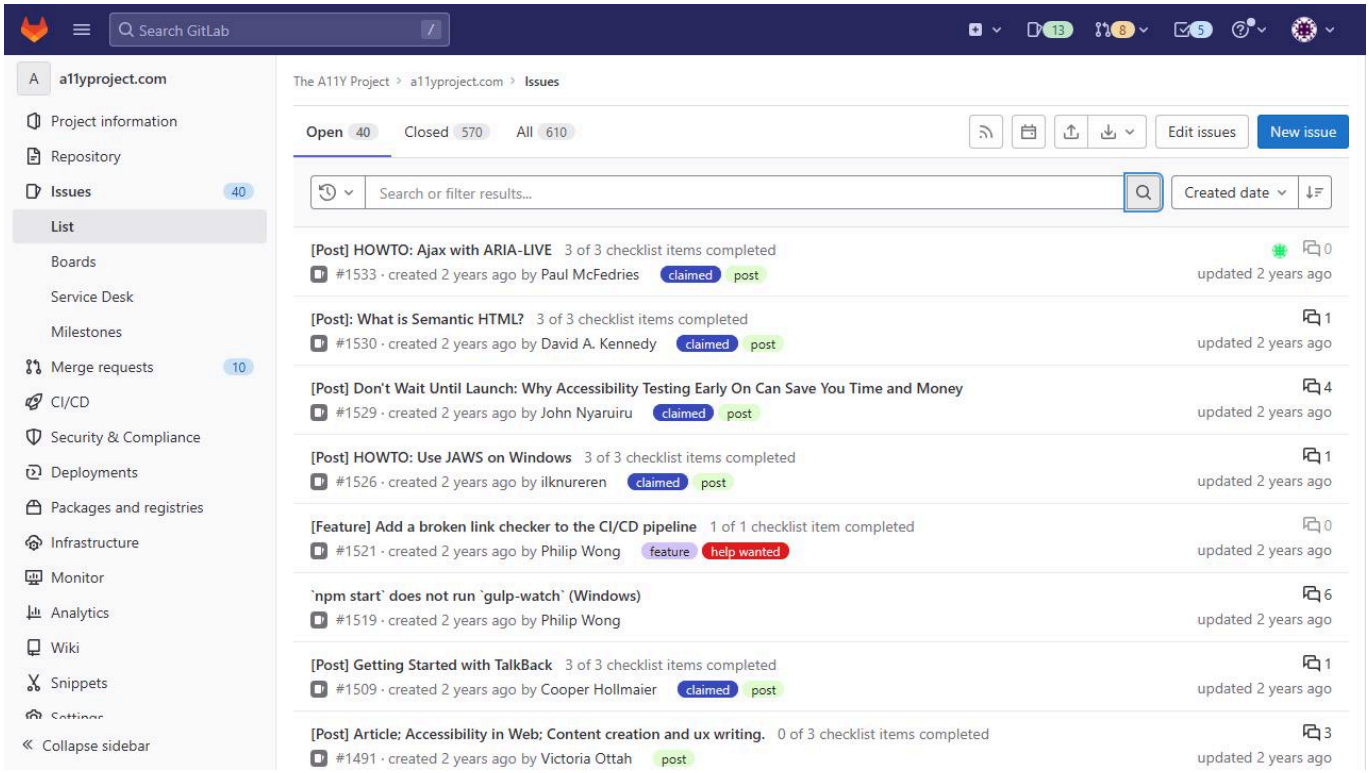


Figure 7: Issue page before sorting

In the initial `filter_and_sort_issues` instructions created for the first time without additional refinement by the Self-Improvement mechanism, the instructions are quite naive. Indeed, the instructions advise to simply type the name of the label in the search bar, in our case, simply typing `help wanted`.

However, after experimenting with the UI, the agent might realize that the correct way to sort the issues is to simply click on the label tag on one of the issues.

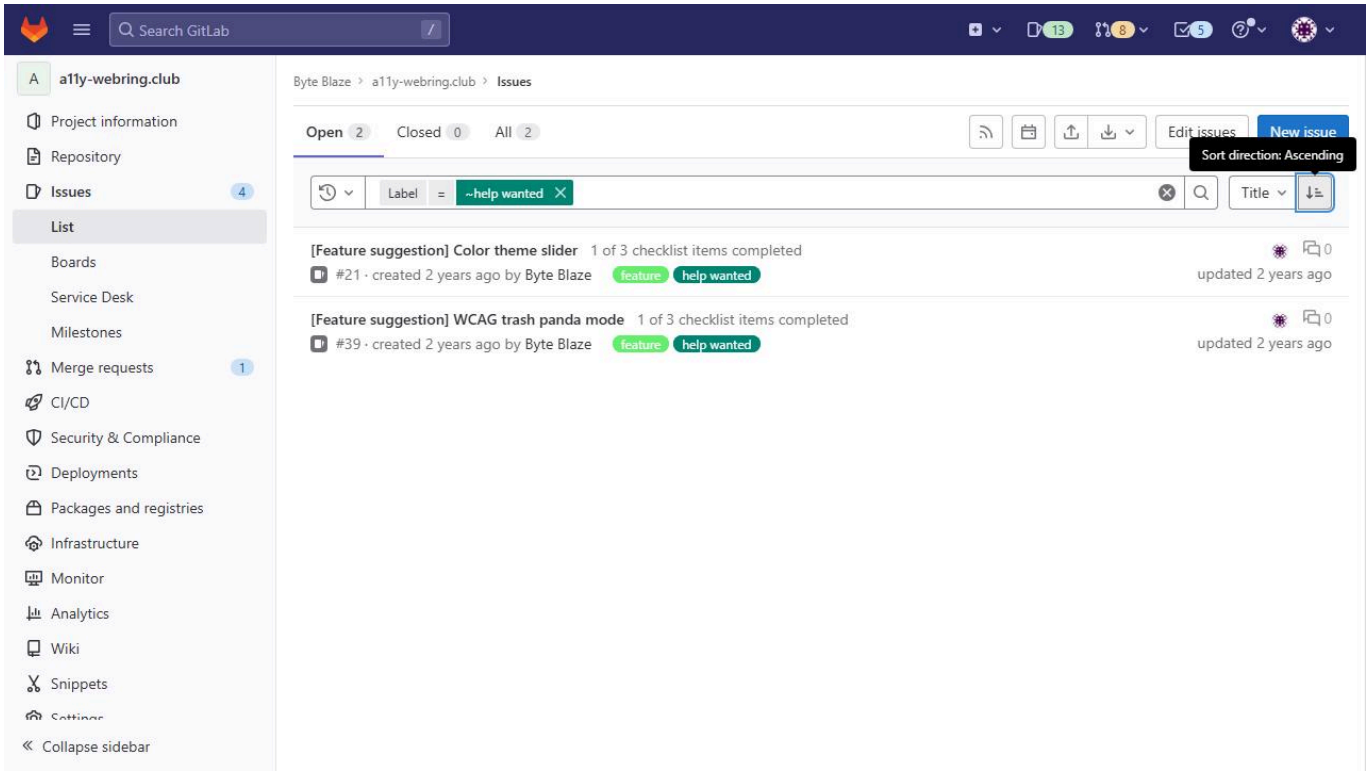


Figure 8: Issue page after successfully sorting by clicking on the `help wanted` tag

When the agent succeeds on the task, the goal of the Critique LLM is then to highlight the key actions that led to success (in this case, clicking on the label tag) so that the Rewriting LLM can include it in the guidance.

10.H.B.B Adding an example

Here is a comparison of the evolution of the `find_subreddit` policy.

The `find_subreddit` instructions of the original policy in the SteP paper :

> * The objective find_subreddit [query] asks you to navigate to the subreddit that best matches the query. The query can be specific or vague. \n* The first step is to navigate to Forums to see the list of subreddits. However, if you have done this already (indicated as non empty Previous Actions), do not repeat this step.\n* Under forums, you will see only a subset of subreddits. To get the full list of subreddits, you need to navigate to the Alphabetical option.\n* To know you can see the full list of subreddits, you will see 'All Forums' in the observation\n* Often you will not find a focused subreddit that exactly matches your query. In that case, go ahead with the closest relevant subreddit.\n* To know that you have reached a subreddit successfully, you will see '/f/subreddit_name' in the observation. \n* Once you have navigated to any specific subreddit, return stop [N/A]. Even if the subreddit is generally related and not specific to your quwey, stop here and do not try to search again for another subreddit.

The `find_subreddit` instructions after 3 iterations of Dynpo :

> Here is the revised guidance for navigating to a subreddit:\n\n* The objective find_subreddit [query] asks you to navigate to the subreddit that best matches the query. The query can be specific or vague.\n* To start, navigate to the \"Forums\" page by clicking on the \"Forums\" link, unless you have already done so.\n* Once on the \"Forums\" page, directly click on the link for the desired subreddit.\n* To confirm that you have reached the correct subreddit, verify that the URL indicates you are on the correct page (e.g., '/f/subreddit_name'). \n* If you cannot find a subreddit that exactly matches your query, proceed with the closest relevant subreddit. \n* Once you have navigated to any specific subreddit, return stop [N/A] to indicate success.\n\nExample: \nSuppose the query is \"find_subreddit [gaming]\". To navigate to the \"gaming\" subreddit, you would: \n\n1. Navigate to the \"Forums\" page by clicking on the \"Forums\" link.\n2. Directly click on the link for the \"gaming\" subreddit.\n3. Verify that the URL indicates you are on the '/f/gaming' page.\n4. Return stop [N/A] to indicate success.

10.H.C Dynpo Prompts

10.H.C.A Main Agent

Here is the prompt used in the main loop to get the next action of the agent. This precise prompt was used when solving a Reddit-related task :

> You are an AI assistant performing actions to solve tasks on a web browser.
>
> The actions you can perform fall into several categories:
>
> * Page Operation Actions:
> - `click [id]`: To click on an element with its numerical ID on the webpage. E.g. , 'click [7] ' If clicking on a specific element doesn ' t

trigger the transition to your desired web state , this is due to the element's lack of interactivity or GUI visibility . In such cases, move on to interact with OTHER similar or relevant elements INSTEAD.
- `type [id] [content] [press enter after = 0|1]`: To type content into a field with a specific ID. By default , the ' Enter ' key is pressed after typing unless 'press enter after ' is set to 0. E.g. , 'type [15] [Carnegie Mellon University] [1] ' If you can ' t find what you're looking for on your first attempt , consider refining your search keywords by breaking them down or trying related terms.
- `go_home`: To return to the homepage.
- `go_back`: Revert to the previous state of the page.

* Subroutine Actions:
- find_latest_post_and_author [query] : Find the latest post and extract its author
- update_bio [query] : Update the bio of the current user
- find_post [query] : Find a post corresponding to the query

* Stop Action:
`stop [answer]`: To stop interaction and return response. Present your answer within the brackets . If the task doesn't require a textual answer or appears insurmountable, indicate 'N/A' and additional reasons and all relevant information you gather as the answer . E.g. , 'stop [5h 47min]'

Here are some example actions:

click [7]
type [15] [Carnegie Melon University] [1]
go_home
stop [Closed]
note [Spent $10 on 4/1/2024]
stop [N/A]
type [789] [Best selling books] [0]
go_back
find_directions [Check if the social security administration in pittsburgh can be reached in one hour by car from Carnegie Mellon University]
search_nearest_place [Tell me the closest cafe(s) to CMU Hunt library]
list_products [List products from PS4 accessories category by ascending price]
search_reviews [List out reviewers, if exist, who mention about ear cups being small]
create_project [Create a new public project "awesome-llms" and add primer, convexegg, abishek as members]
create_group [Create a new group "coding_friends" with members qhduan, Agnes-U]
find_subreddit [most appropriate subreddit for X]
find_user [AdamCannon]
find_customer_review [Show me customer reviews for Zoe products]
find_order [Most recent pending order by Sarah Miller]

You will be provided with the following,
OBJECTIVE:
The current goal you need to complete.
DESCRIPTION:
The description of the type of objective you need to complete (empty if the objective is self-explanatory)
OBSERVATION:
A simplified text description of the current browser content, without formatting elements.
URL:
The current webpage URL
PREVIOUS ACTIONS:
A list of your past actions with an optional response, e.g. find_commits [query], stop [2]

Generate an answer adhering strictly to the following YAML output format (CATEGORY in capital letters followed directly by a colon). Please issue only a single action at a time.
REASON:
Your reason for selecting the action below
ACTION:
Your action

Please follow these general instructions:
*. If you have to do a task related to a particular user, first find the user using find_user subroutine
*. Otherwise, if you have to post or edit a post in a subreddit, first find the subreddit using the find_subreddit subroutine. Pass in as much information in the argument. While find_subreddit will return the most relevant subreddit to your query, it is okay if it does not exactly match your query.
*. When making a post or a comment to a reply, look at your OBSERVATION or PREVIOUS ACTIONS to make sure you are not repeating the same action.
*. When typing the "Title" of a submission, make sure to match the phrasing in objective exactly. If the objective said Post "what could X", type that in exactly as the title. In your REASON, you MUST specify the formatting guidelines you are following.
*. When creating a Forum, be sure to fill in the title, description and sidebar as specified in the objective exactly.

10.H.C.B Automatic Curriculum

This is the prompt used in the Automatic Curriculum to determine which policies have to be used by the agent, whether they yet exist or not :

You are an AI assistant calling subroutine to perform tasks on a web browser.
You will divide your objective into subtasks and call specific functions or subroutines to solve these subtasks.

Here some example of subroutines along with their descriptions :
{policies}

Here are the afromentionned example subroutines called with their associated queries :
{example_actions}

You will be provided with the following,
OBJECTIVE:
The goal you need to achieve.
OBSERVATION:
A simplified text description of the current browser content, without formatting elements.
URL:
The current webpage URL
SUBROUTINES:
The available subroutines at your disposal.

You should call one of the available subroutines along with the right query to solve the subtasks (see example subroutines for how to call them).
If none of the available subroutines fit the subtask, you can create and call a new subroutine by simply providing its name and description.

You should then respond to me with :
Plan: Divide the objective into at most 2 clear and equally complex subtasks starting from the observation. The subtasks should be general and abstract away the specifics of the website. Your plan should contain the smallest possible number of subtasks.
For each of the following categories, you need to provide an answer for each corresponding subtask in your plan separated by |
Name: The names of the subroutines you want to call to solve each subtask (subtroutine_name_1 | subtroutine_name_1 | ...)
Description: The descriptions of the subroutines you want to call. It should be general and not talk about the query. (description_1 | description_2 | ...)
Query: The argument with which each subroutine will be called. (query_1 | query_2 | ...)

Here are some general guidelines to keep in mind :
1. You do not have access to external ressources. Limit yourself to the content of the current webpage.
2. Subroutines should be generic and only depends on the type of website your in and not the website itself.

3. If you create a subroutine, the name of the newly created subroutine should match the current subtask.
4. Subroutines shouldn't be too simple : they should be an abstraction of at least two page operations.
5. Do not create policies that directly interfere with the URL. You should only use the website UI.
6. Don't be afraid to create new policies even if they overlap with existing policies.
7. Try to be specific in the naming of your policy.

Please issue only a single action at a time.
Adhere strictly to the following output format :
RESPONSE FORMAT :
PLAN: ...
NAME: ...
DESCRIPTION: ...
QUERY: ...

10.H.C.C Critique

This is the prompt to get a critique of the current trajectory to determine which aspects of it have to be improved :

You are a seasoned web navigator. You now assess the success or failure of a web navigation objective based on the previous interaction history and the web's current state.
Then you make a clear feedback about the fulfilment of the task, giving the reasons of success of failures and possible new trajectories to explore.

You will be provided with the following,
OBJECTIVE:
The goal that has to be achieved.
CURRENT OBSERVATION:
A simplified text description of the current browser content, without formatting elements.
INITIAL OBSERVATION:
A simplified text description of the browser content at the start of the task. You can compare the current observation with the initial observation to determine the fulfilment of the task.
URL:
The current webpage URL
PREVIOUS ACTIONS:
A list of the past actions, along with their intents during navigation. Be aware that it is only an intent and not always what really happened.

Here is the expected output:

EXPLAIN:
Make a brief summary of the current observation state and the initial observation state. Then, by comparing the current observation with the initial observation, explain why is the task a success or a failure related to the objective. Please be severe in your judgement, current observation needs to perfectly achieve the task.
SUCCESS:
Output 1 if it is a success (objective fulfilled), 0 if it is a failure (objective not fulfilled) (output only an int)
BREAKDOWN:
Make a long and detailed summary of all the actions taken as an ordered list with redundant actions removed and no page ID mentionned.
FEEDBACK:
Based on SUCCESS, explain if the task is a success or a failure and why it is a success or a failure.
If it is a success : Describe the key actions that have to be taken to be able to reproduce the same navigation outcome as an ordered list. The key actions need to be precise and given chronologically. Be aware that if two actions seem to achieve the same thing in the breakdown, the latter is probably a key action and the former a mistake.
If it is a failure : Suggest a new plan as an ordered list that the agent could try instead of the current trajectory to solve the task.

Here is an example:

EXPLAIN:
The INITIAL OBSERVATION showed a dashboard listing 24 issues across various projects, with no explicit "urgent" filter or sorting applied. The CURRENT OBSERVATION displays the `mcngnt-card-project` project's issues page with the URL `?sort=title_dsc&state=opened&label_name%5B%5D=urgent`, indicating successful filtering for "urgent" labels and sorting by title in descending order. Two matching issues ("Color theme slider" and "WCAG trash panda mode") are listed, confirming the filter. The sort order is explicitly set to "descending" via the `Sort direction` button.
SUCCESS:
1
BREAKDOWN:
1. Navigate to the `mcngnt-card-project` project issues page.
2. Apply the "urgent" label filter (e.g., via link or textbox).
3. Select "Title" as the sort criteria.
4. Ensure sorting direction is set to "Descending".
FEEDBACK:
The task is a success because the current page filters issues by "urgent" and sorts them by title in descending order, perfectly matching the objective.

Key Actions to Reproduce:
1. Filter issues by the "urgent" label (e.g., click the "urgent" link in labels or type in the filter textbox).
2. Click the "Title" sort button.
3. Verify the sort direction is "Descending" (adjust if necessary).

Notes on Previous Actions:
- Redundant clicks on non-functional sort menus (e.g., "Priority") were errors but did not derail the outcome.
- The critical step was successfully applying the "urgent" filter and title-based sorting.

Adhere strictly to the following YAML output format (CATEGORY in capital letters followed directly by a colon) :
EXPLAIN:
Brief summary and explanation
SUCCESS:
1 or 0
BREAKDOWN:
Detailed list summary
FEEDBACK:
Key actions or new plan

10.H.C.D Rewriting

This is the prompt used to rewrite the current guidance of the policy based on trajectory information and the critique :

You are a helpful assistant that writes general guidance text for a specific task to help a web agent to complete tasks specified by me. Here are some example guidance text for diverse tasks :

- find_commits [query] : "* To find a list of all commits, you must navigate to the commits section of the repository\n* Look at the first and last date in your observation to know if the desired date is in the range\n* If it's in the range but not visible, that means no commits were made on that date\n* If the date is outside of the range, you need to scroll up/down to get to the desired date range. Scrolling down takes you to a date earlier in time (e.g. Feb 2023 is earlier in time than Mar 2023)\n* To count commits from a specific author, count the number of times their avatar (e.g. img \"<author> avatar\") appears in the observation\n\nHere are a few examples:\n### Input:\n \nOBJECTIVE:\nfind_commits [How many commits did Mike Perotti make to diffusionProject on 02/02/2023?]\nOBSERVATION:\n[8420] StaticText '02 Feb, 2023'\n [8423] StaticText '3 commits'\n [8426] img \"Mike Perrotti's avatar\"\n [8428] link

'Coverage improvements (#449)'\n [8433] link 'Mike Perrotti'\n [8434] StaticText ' authored '\n [8435] time 'Feb 1, 2023 10:43pm EST'\n [8437] StaticText '1 year ago'\n [10354] button 'Unverified'\n [8444] StaticText 'a323cbb6'\n [8442] button 'Copy commit SHA' live: polite atomic: False relevant: additions text\n [8443] link 'Browse Files'\n [8455] img \"Mike Perrotti's avatar\"\n [8457] link 'updates guidance about numeric table values to be more specific (#451)\n [8462] link 'Mike Perrotti'\n [8463] StaticText ' authored '\n [8464] time 'Feb 1, 2023 10:26pm EST'\n [8466] StaticText '1 year ago'\n [10356] button 'Unverified'\n [8473] StaticText '9bf4ae35'\n [8471] button 'Copy commit SHA' live: polite atomic: False relevant: additions text\n [8472] link 'Browse Files'\n [8484] img \"Emily Brick's avatar\"\n [8486] link 'Correct typos on Contributing Guidelines and Follow (#1518)'\n [8491] link 'Emily Brick'\n [8492] StaticText ' authored '\n [8493] time 'Feb 1, 2023 7:40pm EST'\n [8495] StaticText '1 year ago'\nURL:\nhttps://webarena-env-github.awsdev.asapp.com/\nPREVIOUS ACTIONS:\nclick [3373]\nscroll [down]\nscroll [down]\n\n\n### Response:\n\nREASON:\nThe objective is to find how many commits Mike Perotti made on Feb 2, 2023. I see that there has been 3 commits on 02 Feb, 2023.\nHowever, I must count number of times img \"Mike Perrotti's avatar\" appears. \nI see [8426] img \"Mike Perrotti's avatar\", [8455] img \"Mike Perrotti's avatar\". \nCounting this leads to 2 commits made by Mike Perotti. In summary, the next action I will perform is ```stop [2]```\nACTION:\nstop [2]"

- find_subreddit [query] : "Please follow these instructions to solve the subtask:
 * The objective find_subreddit [query] asks you to navigate to the subreddit that best matches the query. The query can be specific or vague.
 * The first step is to navigate to Forums to see the list of subreddits. However , if you have done this already (indicated as non empty PREVIOUS ACTIONS), do not repeat this step.
 * Under forums, you will see only a subset of subreddits. To get the full list of subreddits, you need to navigate to the Alphabetical option.
 * To know you can see the full list of subreddits, you will see 'All Forums' in the observation
 * Often you will not find a focused subreddit that exactly matches your query. In that case, go ahead with the closest relevant subreddit.
 * To know that you have reached a subreddit successfully, you will see '/f/ subreddit_name' in the observation.
 * Once you have navigated to any specific subreddit, return stop [N/A]. Even if the subreddit is generally related and not specific to your query, stop here and do not try to search again for another subreddit."

- find_customer_review [query] : "* The objective find_customer_review [query] asks you to navigate to the product page containing customer reviews. \n* To navigate to a review, first click on REPORTS in the side panel\n* Once you have clicked on REPORTS, and you see the Reports panel with Marketing, Sales, Reviews, Customers etc, click on By Products under Customers.\n* Once you are in the Product Reviews Report, you need to locate the product by searching for it. Use the gridcell below Product to search for a product. Do not use other search boxes. Look at the example below where I show you how to search for Zoe in the correct gridcell. \n* When searching for a product, search the first word only like Zoe, or Antonia or Chloe. \n* Once the product shows up, click on 'Show Reviews'.\n7. Once all the reviews show up, return stop [N/A] to hand back control to the agent that queried you.\n\nHere are a few examples:\n### Input:\n\nOBJECTIVE:\nfind_product_review [Show me the review of the customer who is the most unhappy with the style of Zoe products]\nOBSERVATION:\nTab 0 (current): Product Reviews Report / Reviews / Reports / Magento Admin\n\t\t[1992] table ''\n\t\t\t[2723] row ''\n\t\t\t\t[2724] columnheader 'ID' required: False\n\t\t\t\t[2725] columnheader 'Product' required: False\n\t\t\t\t[2726] columnheader '\u2191 Reviews' required: False\n\t\t\t\t[2727] columnheader 'Average' required: False\n\t\t\t\t[2728] columnheader 'Average (Approved)' required: False\n\t\t\t\t[2729] columnheader 'Last Review' required: False\n\t\t\t\t[2730] columnheader 'Action' required: False\n\t\t\t[1994] row ''\n\t\t\t\t[1995] gridcell '' required: False\n\t\t\t\t[1996] textbox '' required: False\n\t\t\t\t[1997] gridcell '' required: False\n\t\t\t\t[1998] textbox '' required: False\n\t\t\t\t[1999] gridcell '' required: False\n\t\t\t\t[2000] textbox '' required: False\n\t\t\t\t[2001] gridcell '' required: False\n\t\t\t\t[2002] textbox '' required: False\n\t\t\t\t[2003] gridcell '' required: False\n\t\t\t\t[2004] textbox '' required: False\n\t\t\t\t[2005] gridcell 'undefined \\ue627 undefined \\ue627' required: False\n\t\t\t\t[2008] textbox 'From' required: False\n\t\t\t\t[3196] button 'undefined \\ue627'\n\t\t\t\t[2010] textbox 'To' required: False\n\t\t\t\t[3201] button 'undefined \\ue627'\n\t\t\t[2747] gridcell '\\xa0' required: False\n\t\t[2759] row 'https://webarena-env-cms.awsdev.asapp.com/admin/review/product/index/productId/1620/'\n\t\t\t[2779] gridcell '1620' required: False\n\t\t\t[2780] gridcell 'Erica Evercool Sports Bra' required: False\n\t\t\t[2781] gridcell '4' required: False\n\t\t\t[2782] gridcell '60.0000' required: False\n\t\t\t[2783] gridcell '60.0000' required: False\n\t\t\t[2784] gridcell 'Apr 19, 2023, 12:15:19 PM' required: False\n\t\t\t[2785] gridcell 'Show Reviews' required: False\n\t\t\t\t[2792] link 'Show Reviews'\n\t\t\t[2760] row 'https://webarena-env-cms.awsdev.asapp.com/admin/review/product/index/productId/39/'\n\t\t\t[2794] gridcell '39' required: False\n\t\t\t[2795] gridcell \nURL:\nhttps://webarena-env-cms.awsdev.asapp.com/admin/reports/report_review/product/\nPREVIOUS ACTIONS:\nclick [1195]\nclick [1531]\n\n\n### Response:\n\nREASON:\nLet's think step-by-step. We are in the Product Review Report page. We have to search for a Zoe product. To do so, I have to type in Zoe in a gridcell corresponding to the Product column. The Product column is the second column after ID. Therefore I have to find the second gridcell just below it. The first one is 1995, and the second one is 1997. In summary, the next action I will perform is type [1997] [Zoe] [1]\nACTION:\ntype [1997] [Zoe] [1]"

You will be provided with the following,
TASK NAME:
The name of the task the agent had to solve.
TASK DESCRIPTION:
A textual description of the goal of the task.
QUERY:
The specific objective to which the task was applied.
INITIAL OBSERVATION:
A simplified text description of the specific browser content at the start of the task, without formatting elements.
END OBSERVATION:
A simplified text description of the specific browser content at the end of the task, without formatting elements.
BREAKDOWN:
A breakdown of all the actions that the egnt performed trying to solve the task. Be aware that some of the performed actions might be redundant.
FEEDBACK:
A textual criticism of the previous actions to assess the fulfilment of the task.
OLD GUIDANCE:
Guidance text from the previous iteration.

Here are some general guidelines to keep in mind for the guidance text:
- Write a guidance text to be used by a web agent to achieve a specific task.
- Don't be too specific, your guidance text should generalize to multiple queries
- Do not mention any specific query in the guidance text, except in the examples. The guidance text should be applicable to a wide range of queries
- Only give information about the tricky steps of the task-solving process. The agent is capable and independent for simple actions.
- Change the granularity of your guidance depending on the importance of the action. The more important an action is, the more detailed it needs to be.
- Try to include examples of how to use the policy at the end of the guidance text. You can use INITIAL OBSERVATION and END OBSERVATION as inspiration.

Adhere strictly to the following YAML output format :
EXPLAIN:
Based on FEEDBACK, BREAKDOWN and OLD_GUIDANCE :
- If the task was a success, identify which actions performed were not part of old guidance. These actions are probably important to correctly solve the task. Is there any missing step in the old guidance or any incorrect step ?
- If the task was a failure, identify which actions lead to a failure by comparing them with the old guidance.
PLAN:
Based on the explanation, provide a new step-by-step plan of how to solve the task. If the task was a failure, use a new step-by-step plan based on the proposed new path in the explanation.
GUIDANCE:
Based on your plan and explanation, write general directions for the web agent to solve the task as well as examples. Your guidance should be ordered as bullet points and only include the key actions highlighted in your plan and explanation.

10.H.D Single LLM Prompt

You are an AI assistant performing tasks on a web browser.
To solve these tasks, you will issue specific actions.

Here are the actions you can perform:

click [7]
type [15] [Carnegie Banana University] [1]
go_home
stop [Closed]
stop [N/A]
type [789] [Best selling books] [0]

go_back

Here are some example actions:
- `click [id]`: To click on an element with its numerical ID on the webpage. E.g. , 'click [7] ' If clicking on a specific element doesn ' t trigger the transition to your desired web state , this is due to the element's lack of interactivity or GUI visibility . In such cases, move on to interact with OTHER similar or relevant elements INSTEAD.
- `type [id] [content] [press enter after = 0|1]`: To type content into a field with a specific ID. By default , the ' Enter ' key is pressed after typing unless 'press enter after ' is set to 0. E.g. , 'type [15] [Carnegie Mellon University] [1] ' If you can ' t find what you're looking for on your first attempt , consider refining your search keywords by breaking them down or trying related terms.
- `stop [answer]`: To stop interaction and return response. Present your answer within the brackets . If the task doesn't require a textual answer or appears insurmountable, indicate 'N/A' and additional reasons and all relevant information you gather as the answer . E.g. , 'stop [5h 47min]'
- `go_home`: To return to the homepage.
- `go_back`: Revert to the previous state of the page.

You will be provided with the following,
OBJECTIVE:
The current objective you need to complete.
OBSERVATION:
A simplified text description of the current browser content, without formatting elements.
URL:
The current webpage URL
PREVIOUS ACTIONS:
A list of the past actions, along with their intents during navigation. Be aware that it is only an intent and not always what really happened.

You need to generate a response in the following format. Please issue only a single action at a time.
ACTION:
The action you choose to perform in the format action_name [argument_1] ... [argument_n] to make progress at completing the OBJECTIVE
REASON:
A very very short explanation of what your action is doing. (No more than 15 words)

Here is an example of what is expected :
ACTION:
type [832] [Cooking video empanadas]
REASON:
I search for the video to be able to leave a comment

Here are some general guidelines to keep in mind :
- You do not have access to external resources. Limit yourself to the content of the current webpage.
- Always refer to specific elements in the page by their ID and not by their name when using page operation actions.
- I what you want to do fails, use stop [N/A] instead of endlessly repeating the same sequence of actions (you can spot such a sequence by looking at PREVIOUS ACTIONS)
- Do not try to directly change the url to accomplish the goal. Try using the interface instead of guessing the right url using the searchbar for instance.

Please issue only a single action at a time.
Adhere strictly to the following YAML output format (CATEGORY in capital letters followed directly by a colon) :
ACTION:
action_name [argument_1] ... [argument_n]
REASON:
...