FAST: Finetuning Agents with Synthetic Trajectories

Stanford CS224N Custom Project

Flor Lozano-Byrne
Department of Computer Science
Stanford University
flbyrne@stanford.edu

Abstract

Thanks to the recent advancements in language understanding, large language models (LLMs) have become powerful tools able to understand and control web browsers, in the form of grounded agents. Human demonstrations are needed, however, to ground agents, posing scalability problems and becoming infeasible as more complex tasks can involve hundreds of actions. Synthetic demonstrations present a promising solution to this problem, and have been used to bootstrap LM agents via Retrieval Augmented Generation (RAG). We introduce another way of using synthetic demonstrations, which is for fine-tuning a language model to act as an agent. We synthetically generate high-quality trajectories using MiniWOB++ environments including common building blocks of web interactions, such as filling text boxes, using auto-complete, clicking buttons or check boxes, liking and retweeting, picking dates, etc. and use them to build an observation->instruction labeled dataset. By combining sub-tasks from different environments into a single dataset, which we use to fine-tune a state of the art LLM, we achieve similar performance to STeP in several MiniWOB++ tasks using a smaller fraction of inference turns compared to BAGEL.

Mentor: Shikhar Murty

1 Introduction

The staggering diversity of websites and possible tasks presents a challenge in creating digital agents. Humphreys Humphreys et al. (2023) developed a method centered on reinforcement learning combined with behavioural priors informed by actual human-computer interactions to control computers using natural language guidance. Advancements have been achieved in creating actions by translating images from website observations Shaw et al. (2023) and fine-tuning models for conversational browser control Lù et al. (2024). These methods, however, require human expert demonstrations which are not scalable.

BAGEL Murty et al. (2024) creates synthetic demonstrations and uses them to bootstrap agents using retrieval augmented generation. It developed an iterative method of exploration guided by natural language and relabeling. STeP Sodhi et al. (2024) creates synthetic trajectories by composing dynamic plans.

Various environments for simulating web interactions have been developed, such as WebShop for simulating shopping websitesYao et al. (2022), WorkArena for knowledge worker tasks Drouin et al. (2024), WebArena Zhou et al. (2023) for general web tasks in various types of sites, ToolQA for allowing the use of tools such as calculators, databases and grpahs Zhuang et al. (2023), and MiniWOB++ Shi et al. (2017) including various simplified environments covering a wide range of tasks.

Drawing inspiration and ideas from this work, we propose a method for programmatically defining synthetic trajectories using MiniWOB++ environments, to assemble a dataset for LM finetuning. We require considerably less LM prompting than BAGEL for creating synthetic trajectories and by combining sub-tasks from several environments, we obtain a combinatorial effect similar to STeP. Our fine-tuned model achieves similar performance to STep and BAGEL on MiniWOB++, and has potential for generalizing to other environments.

2 Related Work

Work in the field of digital agents has addressed several issues such as enabling LLM reasoning to create action plans, HTML summarization for allowing its inclusion in prompts, and multimodal web navigation with instruction tuned foundation models.

WebGUM Furuta et al. (2023) creates a multimodal agent that observes both webpage screenshots and HTML pages and outputs navigation actions such as click and type, by training an instruction-finetuned language model and a vision encoder, outperforming prior works by a significant margin. ReAct Yao et al. demonstrated that it's possible to elicit reasoning behavior from LLMs by structuring prompts in two phases, first reasoning and then acting. WebLINX Lù et al. (2024) created a model that efficiently prunes HTML pages by ranking relevant elements. Our prompts draw from those two concepts, introducing modifications to adapt them to our shorter timeframe and we finetune on language only.

3 Approach

Following BAGEL, we selected ten Miniwob++ environments involving common web interactions such as web search, autocomplete, picking dates, clicking tabs and checkboxes, social media, email, booking flights and tic-tac-toe. We then created 100% successful (raw reward =1) synthetic trajectories using a hybrid programmatic Document Object Model (DOM) search and single-action LM prompt approach, which we used to build our labeled dataset, making sure that no repeated random seeds were present. We test the finetuned model using random Miniwob++ seeds, by checking that those seeds were not present in the training data.

Because our prompts required large context windows for DOM and image inputs, not supported by open source language models, we used paid publicly available models of the Gemini Google (2023) family for the baseline, creation of synthetic trajectories and fine-tuning, which we accessed through GCP, .[1]. Although these models suited our needs, using them limited our usage to the credits received, leading us to discard potentially better approaches.

3.1 Other approaches considered

Before setting into our course of action we tried alternate options, which deemed unsuccessful due to the time and resource constrains of this project. Below is a summary discussion of our experimentation, presented here for its potential value in a more resourceful setting.

- The method we initially planned to use for creating synthetic trajectories was BAGEL, an iterative procedure to relabel a seed set of trajectories obtained from unconditioned exploration using various LM policies. However, its iterative nature and reliance on inference by the various policies for every action, as well as reliance on other packages such as pix2act Shaw et al. (2023), made it too time consuming and costly for the scope of this project.
- The environment we initially chose for creating the trajectories was WebShop, an online shopping simulation environment, in order to see how BAGEL could generalize to environments other than the ones it targeted, namely Miniwob++ and ToolQA. However, the installation of all packages and dependencies proved to be non-trivial and couldn't fit in the timeframe of this project.
- Other interesting environments we considered for generalization and therefore explored were ToolQA, WebArena and WorkArena. ToolQA allows LLMs to use external tools for question answering, such as databases, calculators and graphs. WebArena has a variety of real-life

[1]Also, a credit donation by Google tipped the scales between choosing Gemini vs. its close competitor, OpenAI's GPT-4o.

website environments beyond shopping and what's available on Miniwob++. WorkArena allows agents to interact with company knowledge-bases, forms, service catalogs, etc. to perform knowledge-worker tasks on the ServiceNow platform ser. Unfortunately, all of them required non-trivial installation timeframes, which exceeded this project's scope.

- For reducing DOM sizes we considered Dense Markup Ranking (DMR) and other methods proposed in WebLINX Lù et al. (2024). DMR converts HTML into a simplified DOM representation to allow prompting of smaller LMs, which wouldn't be possible with a complete HTML file. Since Miniwob++ environments don't provide HTML updates and only DOM are available for each action, adapting it to our need of just reducing the DOM size would have required modification of their large codebase, rendering it outside of the scope of this project's timeframe. In 3.3 we develop a simplified method.

We also tried using a Llama3-70B AI@Meta (2024) model, but switched to Gemini because of its additional capabilities.

3.2 Synthetic Trajectories

We produced 60 baseline trajectories and started with 100 training trajectories for each Miniwob++ environment. We used Gemini 1.5 Pro, which allows multi-turn conversations, thus the context was kept for the duration of the trajectory. We also leveraged multimodal prompts including images.

For simplicity, we limit the action space of our trajectories to two actions: click and type, being these the basic building blocks for any web interaction.

3.2.1 Baseline Trajectories

The baseline trajectories were created with a ReAct inspired prompt similar to the instruction-following policy BAGEL prompt, using the Miniwob++ random utterances generated when resetting the environment. Unlike in BAGEL, where the same prompt is used for every action in a given trajectory, we start with a detailed prompt and follow up with simpler prompts, since our LM can keep the context of the conversation. Also, we added a set of rules of interaction to help ground the agent.

A trajectory starts with an initial observation of the environment which is incorporated into the prompt in the form of an image and a list of DOM elements, as shown in A.1.1. The model's response is collected in a format specified in the prompt, to allow information extraction using simple regex expressions. The information needed to activate the next state is the type of action (click or type), the element number on which to perform the action, and the actual text to enter, in the case of typing text. Once the new state is activated, we send a follow-up prompt to the LM A.1.2, which includes the DOM elements corresponding to the new current state. This process continues until the task is completed and/or the environment times out and closes.

Despite our efforts to provide rules to the model, it seldom followed them, resulting in poor completion rates and consistently occurring errors in common areas. The environments with the most problems were book-flight, choose-date, search-engine and use-autocomplete. In these environments, our baseline agent had a 0% success rate, missing 83% of possible states. The biggest problem was the failure to click on text boxes before input, found in all 4 environments 96% of the time. clicking on autocomplete was also a problem 98% of the time, clicking on the datepicker 97% of the time, and using arrows to navigate the calendar forward and backward was a problem 100% of the time. Also, for search-engine it wasn't able to count the results to arrive at the desired rank. Some of these errors are illustrated in 1.

Other environments such as social-media, email-inbox and tic-tac-toe had errors too, but those occurred only with certain goals, so the agent was able to complete some of the trajectories for those. In social-media, it failed when the requested option was not on the main screen but a menu needed to be clicked. In email-inbox it did well with liking or starring messages, but never clicked the reply or forward buttons. In social-media-some it did well when only one click was needed, but failed when needing to count more clicks. In tic-tac-toe, it wasn't able to win any game.

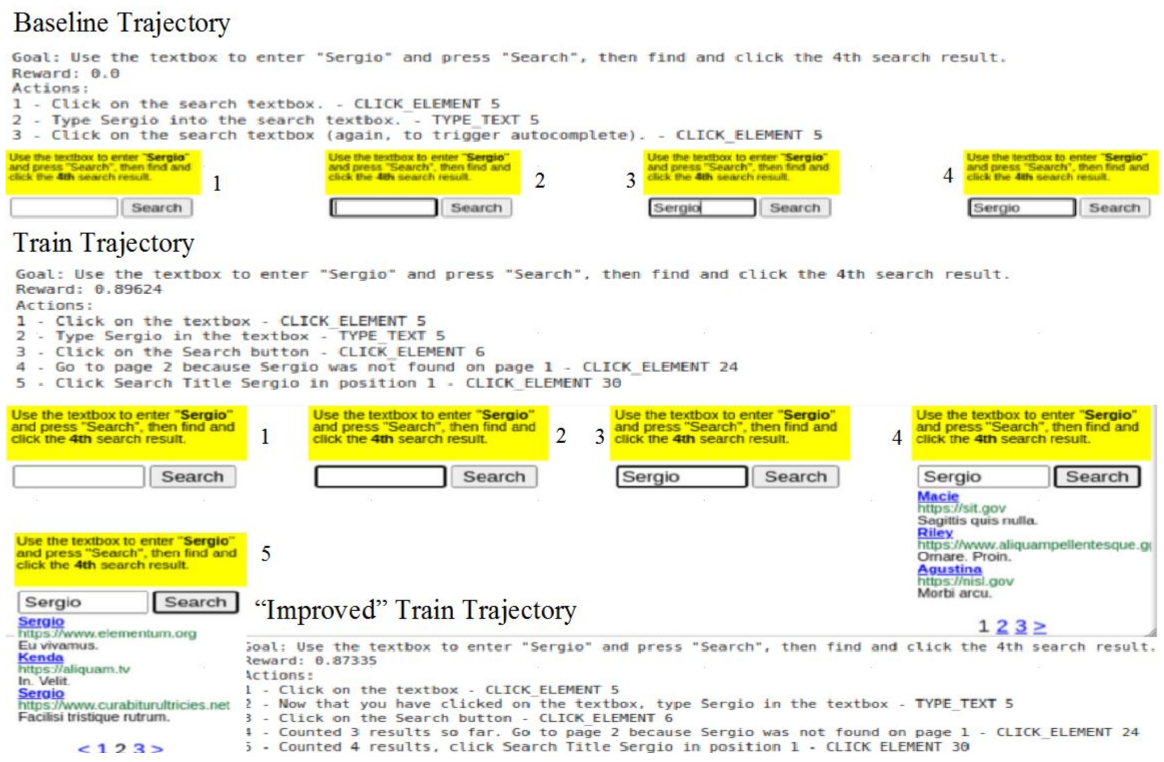
The complete results and comparison are in 1.

3.2.2 Training Trajectories

We aim to produce training data that includes all possible states for every environment, and that means a large number of successful synthetic trajectories. For example, if the agent never clicks on the "Forward" button, it would never produce the state in which the email writing screen is shown, so that state will be missing in our training dataset. Having discarded BAGEL as a possibility for creating synthetic trajectories for this project, we used the following procedure for each environment to ensure that we can reach our goal of a large number of successful training trajectories:

- Identified the actions needed to complete the task
- Took into account the information from the errors found in the baseline to guide successful completion
- Created simple python code to dynamically search and identify the DOM elements involved in any given random instruction
- Prompted the LM including the DOM elements for single tasks that couldn't be predicted, required judgement, or required complex search algorithms A.1.3

This method resulted in FAST creation or synthetic trajectories with 70% less prompting than the baseline and at least 80% less prompting than BAGEL.


Baseline Trajectory

```
Goal: Use the textbox to enter "Sergio" and press "Search", then find and click the 4th search result.
Reward: 0.0
Actions:
1 - Click on the search textbox. - CLICK_ELEMENT 5
2 - Type Sergio into the search textbox. - TYPE_TEXT 5
3 - Click on the search textbox (again, to trigger autocomplete). - CLICK_ELEMENT 5
```

Train Trajectory

```
Goal: Use the textbox to enter "Sergio" and press "Search", then find and click the 4th search result.
Reward: 0.89624
Actions:
1 - Click on the textbox - CLICK_ELEMENT 5
2 - Type Sergio in the textbox - TYPE_TEXT 5
3 - Click on the Search button - CLICK_ELEMENT 6
4 - Go to page 2 because Sergio was not found on page 1 - CLICK_ELEMENT 24
5 - Click Search Title Sergio in position 1 - CLICK_ELEMENT 30
```

"Improved" Train Trajectory

```
oal: Use the textbox to enter "Sergio" and press "Search", then find and click the 4th search result.
eward: 0.87335
ctions:
1 - Click on the textbox - CLICK_ELEMENT 5
2 - Now that you have clicked on the textbox, type Sergio in the textbox - TYPE_TEXT 5
3 - Click on the Search button - CLICK_ELEMENT 6
4 - Counted 3 results so far. Go to page 2 because Sergio was not found on page 1 - CLICK_ELEMENT 24
5 - Counted 4 results, click Search Title Sergio in position 1 - CLICK ELEMENT 30
```

Use the textbox to enter "Sergio" and press "Search", then find and click the 4th search result.
1
2
3
4
5
Search
Sergio
Macie
https://sit.gov
Sagittis quis nulla.
Riley
https://www.aliquampellentesque.g
Ornare. Proin.
Agustina
https://nisl.gov
Morbi arcu.
1 2 3 >
Sergio
https://www.elementum.org
Eu vivamus.
Kenda
https://aliquam.tv
In. Velit.
Sergio
https://www.curabiturultricies.net
Facilisi tristique rutrum.
< 1 2 3 >

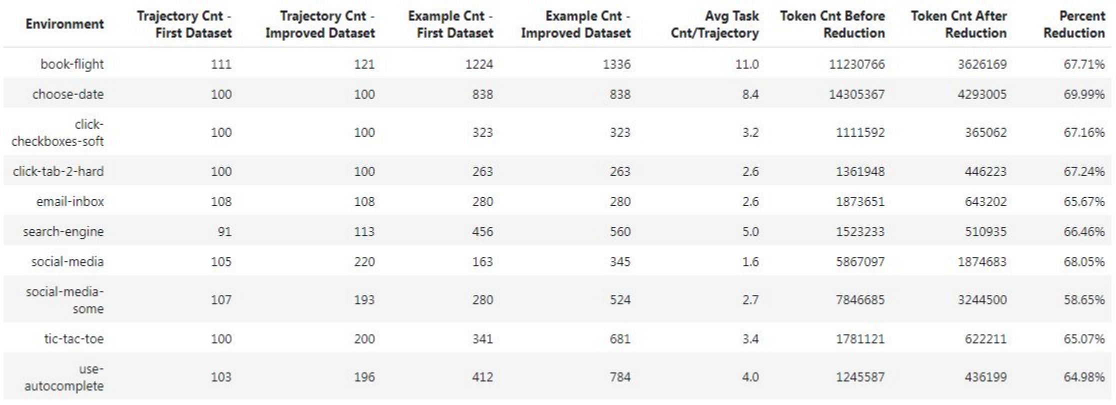
Figure 1: This example illustrates the type of errors in our baseline trajectories, which were obtained by prompting the LM for each action. Comparing it to the train trajectories created programmatically, we see that it's missing states would be needed for training because of failing to click on the search button (this is an uncommon error though, but because the seeds are random, it's hard to get two equal seeds to compare). After we finetuned the first model, we sought to improve a second version by providing actions with more background. Although the improved trajectory makes more sense, it didn't improve the results, see 1

3.3 Dataset Creation

We save the training trajectories individually in pickled python dictionaries for ease or analysis.The supervised fine-tuning task on Gemini 1.0 Pro-002 requires a JSONL dataset in which each JSON line represents a message containing the user prompt and the desired response. Our training prompt consists on a goal sentence paired with the list of DOM elements representing the state of the environment (observation) for input and the corresponding instruction for output. We didn't include images due to the fact that this model, the only one in the Gemini family available for fine-tuning on GCP, is not multi-modal. Each example corresponds to one action of the trajectory, however the overall goal is the same for each individual action of the trajectory. The token limit for the goal and DOM part was 8,192 tokens, and a large percentage of examples our dataset exceeded it.

Aiming to keep all the information required for the task, we removed unneeded elements, achieving an average 66% token count reduction. Removals were as follows:

- Foreground and background color definitions are needed for actions such as typing text, where the border color changes to indicate if the box has been clicked or not, or to indicate incorrect inputs. While colors may be relevant in many of our training environments, we hope that the text part will suffice to clarify those actions.
- Flags are used to to indicate the state of elements, such as if they are clickable or if they have been clicked or not. We hope to include that information in the text part of the example, so we removed flags for most environments. In the social-media-some task however, showing the already clicked buttons is crucial, so we leave the flags and we eliminate the trajectories that exceed the limit, which correspond to web-pages with a larger number of items, and we hypothesize that this may not have any impact on the quality of the dataset (only 7% of trajectories exceeded the limit) To compensate, we include additional trajectories to reach the number of 100, as in the other environments.
- We make all remaining arrays integer to eliminate extra characters and convert all arrays with only one element to scalars.
- After converting the DOM elements from a dictionary to a JSON string, we also eliminate all quotes, convert the arrays to lists and remove variable type names.



Environment	Trajectory Cnt - First Dataset	Trajectory Cnt - Improved Dataset	Example Cnt - First Dataset	Example Cnt - Improved Dataset	Avg Task Cnt/Trajectory	Token Cnt Before Reduction	Token Cnt After Reduction	Percent Reduction
book-flight	111	121	1224	1336	11.0	11230766	3626169	67.71%
choose-date	100	100	838	838	8.4	14305367	4293005	69.99%
click-checkboxes-soft	100	100	323	323	3.2	1111592	365062	67.16%
click-tab-2-hard	100	100	263	263	2.6	1361948	446223	67.24%
email-inbox	108	108	280	280	2.6	1873651	643202	65.67%
search-engine	91	113	456	560	5.0	1523233	510935	66.46%
social-media	105	220	163	345	1.6	5867097	1874683	68.05%
social-media-some	107	193	280	524	2.7	7846685	3244500	58.65%
tic-tac-toe	100	200	341	681	3.4	1781121	622211	65.07%
use-autocomplete	103	196	412	784	4.0	1245587	436199	64.98%

Figure 2: In our first dataset we aimed for 100 trajectories per environment. However, we can see that the number of examples is imbalanced because of the different number of tasks for each trajectory. To mitigate this imbalance, aside from other improvements, we increased the number of examples in the lower performing environments that had the least examples. We also see the results of the token reducing measures presented in this section.

3.4 Fine-tuning

We perform parameter efficient fine-tuning (PEFT) of Gemini 1.0 Pro-002 from a GCP n1-standard-4 managed notebook, using the supervised fine-tuning module of Vertex AI (vertexai.preview.tuning.sft). We train for 4 epochs with a learning rate of 0.001 and a learning rate multiplier of 1.

4 Experiments

4.1 Data

We initially created a training dataset including approximately 100 successful trajectories for each of the MiniWOB++ environments. We check for duplicate trajectories both at the time of creating the trajectories and at the time of building the dataset (each single run of 100 trajectories checks itself for duplicates, but sometimes we had to stop, or run additional times due to not enough trajectories having positive rewards or other issues). The number of examples per environment varies due to the variable number of tasks in each environment. Our first dataset included 4580 examples, but after seeing the imbalance in number of examples across environments, due to the variable number of tasks per trajectory, we increased the number of examples for the lowest performing environments with the lowest example counts, which yielded a total of 5934 examples in the improved dataset, as shown in 2. More details on the trajectories included and dataset creation methodology can be found in 3.2.2 and 3.3.

4.2 Evaluation method

Following BAGEL, we test the fine-tuned model on 50+ MiniWOB++ random seeds per environment. Before each test we check against the training seeds in order to ensure that each random seed in the test was not used in the training dataset. We analyze the 500+ resulting trajectories, calculate the average completion rate for each environment and compare it to our baseline and available existing benchmarks for fine-tuned models. For the environments where benchmarks are not available, we compare them to our baseline and BAGEL.

Additionally, we perform a qualitative analysis of the lowest scoring environments and discuss common errors.

4.3 Experimental details

We started by training small models on single environments, which allowed us to investigate and correct issues related to those individual environments. We then trained a model with all environments except two of them and tested to see how it generalized to those environments. After analyzing the results of this first round (FAST), we hoped to improve them by providing background from the previous action in each action description (more on this in 5) and mitigate the imbalance by adding more trajectories. We generated improved trajectories for several environments and used them to train an "improved" model (FAST-I).

We trained an ablation (FAST-A), in which we removed all trajectories from two environments, search-engine and click-checkboxes-soft to investigate generalization to unknown environments.

Duration of training was about an hour per individual environment included in the dataset, with a total of ten hours when training for all environments. GCP offered no GPU options in their fine-tuning and the machine configuration used once the job is launched is unknown. The fine-tuned models are hosted on GCP and they can be accessed through GCP's service offerings.

4.4 Results

Table 1 shows a comparison of results including our baseline, BAGEL, STeP, our first dataset (FAST), a second "improved" dataset and ablation.

FAST improved over STeP and came close to BAGEL on email-inbox. Also, it came slightly ahead of BAGEL on tic-tac-toe, use-autocomplete, book-flight and showed a large (69$\%$ absolute) improvement in search-engine. These results show that this approach's worked.

FAST-A aimed to investigate the generalization potential of this approach. The fact that it came close to BAGEL in search-engine is an indication of it being worth more research.

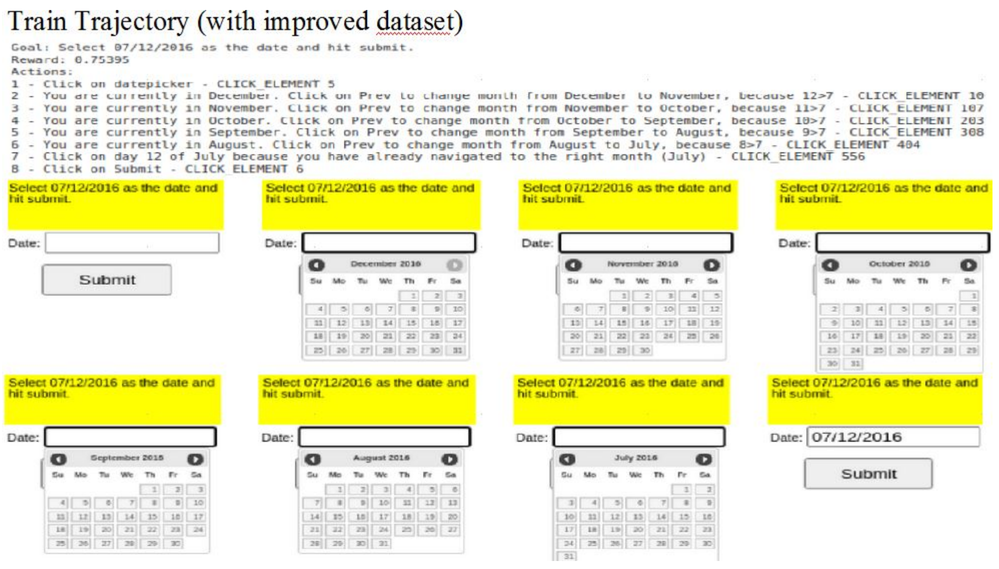
FAST-I came as a result of trying to improve on the first run. The main improvements were doubling the number of trajectories for social-media, social-media-some, tic-tac-toe and use-autocomplete. Also, search-engine had increased trajectories by 20$\%$, all of them in an attempt to have a more balanced dataset. Another improvement was to include precedence in every action, and this was done for book-flight, choose-date, social-media-some, use-autocomplete and search-engine. The tic-tac-toe actions were also improved to make a clear difference when there are already two Xs in a row, instead of having a generalized statement. As we can see, the "improvements" in action precedence resulted in 0 successful trajectories. Also, email-inbox drastically dropped in performance, which we hypothesize could be explained by the drop in its percentage of examples in the dataset.

Environment	Baseline	BAGEL	STeP	FAST	FAST-A	FAST-I
click-tab-2-hard	32	100	100	90	86	90
social-media	35	70	-	60	63	59
email-inbox	30	100	90	100	98	58
social-media-some	5	80	-	25	24	19
tic-tac-toe	6	40	-	44	39	52
use-autocomplete	0	45	-	54	51	0
book-flight	0	15	90	40	22	0
choose-date	0	40	100	10	2	8
search-engine	0	25	100	94	20	0
click-checkboxes-soft	31	90	54	50	18	60

Table 1: Completion Rates for MiniWOB++ Environments

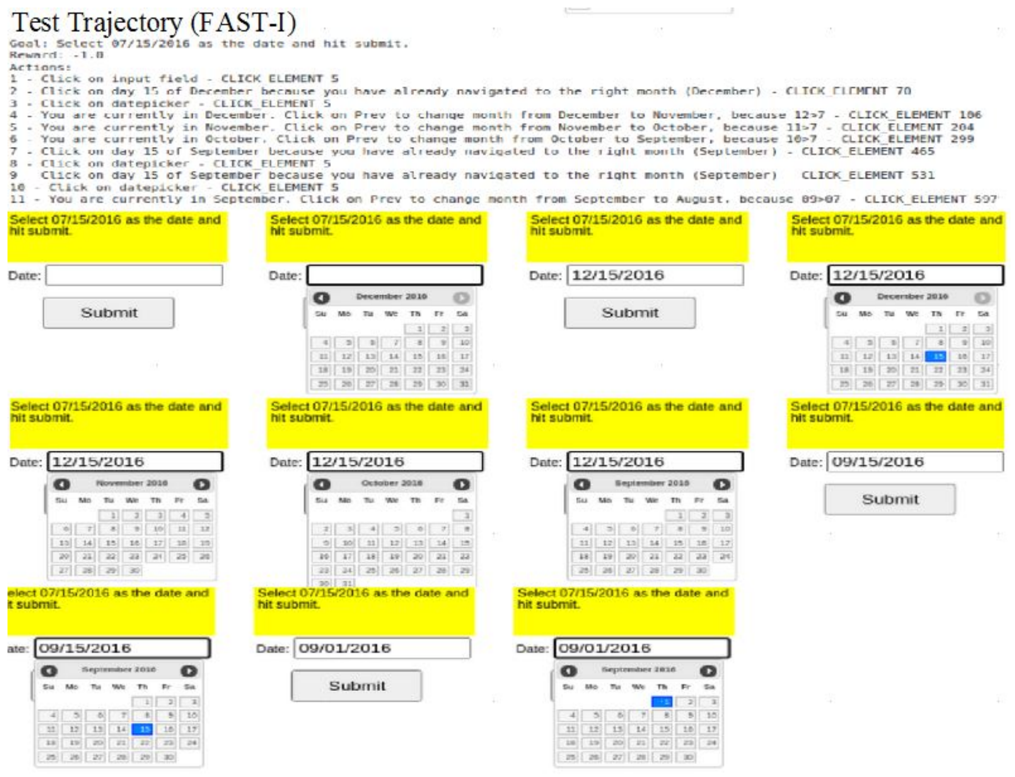
5 Analysis

The worst-performing environment for FAST is choose-date. Looking at the distributions of train and test dates doesn't explain it 5.


Train Trajectory (with improved dataset)

```
Goal: Select 07/12/2016 as the date and hit submit.
Reward: 0.75395
Actions:
1 - Click on datepicker - CLICK_ELEMENT 5
2 - You are currently in December. Click on Prev to change month from December to November, because 12>7 - CLICK_ELEMENT 10
3 - You are currently in November. Click on Prev to change month from November to October, because 11>7 - CLICK_ELEMENT 107
4 - You are currently in October. Click on Prev to change month from October to September, because 10>7 - CLICK_ELEMENT 203
5 - You are currently in September. Click on Prev to change month from September to August, because 9>7 - CLICK_ELEMENT 308
6 - You are currently in August. Click on Prev to change month from August to July, because 8>7 - CLICK_ELEMENT 404
7 - Click on day 12 of July because you have already navigated to the right month (July) - CLICK_ELEMENT 556
8 - Click on Submit - CLICK_ELEMENT 6
```

Figure 3: Training Trajectory for choose-date in July



Test Trajectory (FAST-I)

```
Goal: Select 07/15/2016 as the date and hit submit.
Reward: -1.0
Actions:
1 - Click on input field - CLICK ELEMENT 5
2 - Click on day 15 of December because you have already navigated to the right month (December) - CLICK ELEMENT 70
3 - Click on datepicker - CLICK_ELEMENT 5
4 - You are currently in December. Click on Prev to change month from December to November, because 12>7 - CLICK_ELEMENT 106
5 - You are currently in November. Click on Prev to change month from November to October, because 11>7 - CLICK_ELEMENT 204
6 - You are currently in October. Click on Prev to change month from October to September, because 10>7 - CLICK_ELEMENT 299
7 - Click on day 15 of September because you have already navigated to the right month (September) - CLICK_ELEMENT 465
8 - Click on datepicker - CLICK_ELEMENT 5
9   Click on day 15 of September because you have already navigated to the right month (September)   CLICK_ELEMENT 531
10 - Click on datepicker - CLICK_ELEMENT 5
11 - You are currently in September. Click on Prev to change month from September to August, because 09>07 - CLICK_ELEMENT 597
```

Select 07/15/2016 as the date and hit submit.
Date:
Submit

Select 07/15/2016 as the date and hit submit.
Date:

Select 07/15/2016 as the date and hit submit.
Date: 12/15/2016
Submit

Select 07/15/2016 as the date and hit submit.
Date: 12/15/2016

Select 07/15/2016 as the date and hit submit.
Date: 12/15/2016

Select 07/15/2016 as the date and hit submit.
Date: 12/15/2016

Select 07/15/2016 as the date and hit submit.
Date: 12/15/2016

Select 07/15/2016 as the date and hit submit.
Date: 09/15/2016
Submit

elect 07/15/2016 as the date and t submit.
ate: 09/15/2016

Select 07/15/2016 as the date and hit submit.
Date: 09/01/2016
Submit

Select 07/15/2016 as the date and hit submit.
Date: 09/01/2016

Figure 4: Testing trajectory for choose-date in July

Looking at the trajectories and we realize the agent is very confused about the final destination. We hypothesize that attaching the goal to each prompt is most relevant in this particular environment. The training dataset had the goal attached to each input, and logically it should be included in the input when prompting. This presents an opportunity for further work improving this model.

Further analysis is needed to elucidate the exact causes of the sharp decrease in performance to zero success in FAST-I for the affected environments, and this should be prioritized in further work.

6 Conclusion

Our endeavour to create a dataset of synthetic trajectories to fine-tune a language model as an agent was successful, with completion rates on MiniWOB++ paralleling SOTA in many of the environments tested. We found that there are opportunities for improvement by attaching the goal to the testing prompt, which can achieve even better results. Investigating the sharp drop in performance to zero in the affected environments may uncover further opportunities for improvement. A possible explanation may be a "diluting" effect comparable to adding more categories to a classification dataset, and would create the need for an increased number of examples. Future work could test that hypothesis by training on a larger number of examples, as our increases may not have been sufficient.

7 Ethics Statement

The development of digital agents by fine-tuning language models or other means has the potential of replacing millions of workers by making them obsolete, requiring them to re-train for new careers or else become unemployed. Even before that, developing ways to create synthetic trajectories to assemble fine-tuning datasets will result in decreased demand for workers. Governments and organizations can help mitigate these impacts by establishing lifelong learning programs to re-skill and up-skill workers, creating safety nets, job transition programs and providing tax incentives for training. Educational institutions, governments and private companies can establish partnerships for aligning training programs with market needs.

References

Service now: The intelligent platform for digital transformation.

AI@Meta. 2024. Llama 3 model card.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. Workarena: How capable are web agents at solving common knowledge work tasks?

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Shixiang Shane Faust, Aleksandra Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. In *Advances in Neural Information Processing Systems*.

Gemini Team Google. 2023. Gemini: A family of highly capable multimodal models.

Peter Humphreys, David Raposo, Toby Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. 2023. A data-driven approach for learning to control computers.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. 2024. Bagel: Bootstrapping agents by guiding exploration with language.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. 2023. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. In *Advances in Neural Information Processing Systems*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Paloma Sodhi, S.R.K. Branavan, Yoav Artzi, and Ryan McDonald. 2024. Step: Stacked llm policies for web actions.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *ArXiv*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR)*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools.

A Appendix

A.1 Baseline Prompts

A.1.1 Modified Bagel Instruction-Following Policy Prompt

You are a web-agent capable of executing the following kinds of tasks on a webpage:

1. Click on *description* : This action will click on element that matches *description*. Examples:

- Click on the red button
- Click on the first result in the autocomplete pop up menu
- Click on a date in the calendar
- Click on the scroll bar to scroll up and down
- Click on a left arrow to go to a previous
- click on a right arrow to go to the next
- Click on an input field before entering text

The action code for the clicking action is CLICK-ELEMENT

2. Type *text* on *description*: This action will type *text* into the web element matching *description*. The action code for typing is TYPE-TEXT

You are given the following goal:
{Random utterance from MiniWOB++ environment's reset}

You observe the following image from the web-page HTML:
{Image from MiniWOB++'s environment state (observation) }

Start by thinking about what action you should take next. Write down all the different choices and then, choose the best answer taking into account the following rules:

- **RULES FOR INTERACTING WITH DOM ELEMENTS**:
- **Datepicker**: If the month displayed is the desired month, select the desired day by clicking on a number. If the month desired is before or after the month displayed, use the right and left arrows to navigate to the correct month.
- **Text input box**: click on it before and after entering text, click on first autocomplete option after typing.
- **Autocomplete**: select an option by clicking on it

Provide your answer for the ONE next action in the following format:

Action description:

Once you identified next action to take, look at the DOM elements from the web-page HTML:
{List of DOM elements from MiniWOB++'s environment state (observation) }

and identify the ref number of the element on which you need to perform the action. Write your answer in the following format:

Action code:(choose either TYPE-TEXT or CLICK-ELEMENT)
Dom element ref number:
Text: (only for TYPE-TEXT action code)

A.1.2 Follow-Up Prompt Within Trajectory

After performing these actions:
{List of actions already performed}

Start by thinking about what action you should take next. Write down all the different choices and then, choose the best answer taking into account the previously given rules.

Provide your answer for the ONE next action in the following format:

Action description:

Once you identified next action to take, look at the DOM elements from the web-page HTML: {List of DOM elements from MiniWOB++'s environment state (observation) }

and identify the ref number of the element on which you need to perform the action. Write your answer in the following format:

Action code:(choose either TYPE-TEXT or CLICK-ELEMENT)
Dom element ref number:
Text: (only for TYPE-TEXT action code)

A.1.3 Single-Task Prompt for Programmatic Trajectories

You are a web-agent capable of reasoning to make decisions based on the information you are given. You are able to click on website buttons to select the best option based on the criterion provided. For example, you are able to compare durations, to select the shortest flight.

You are given the following goal:
{Random utterance from MiniWOB++ environment's reset}

You observe the following DOM elements from the web-page HTML:
{List of DOM elements from MiniWOB++'s environment state (observation) }

Start by comparing the different choices available and then find the best answer. Provide your answer for the ONE next action in the following format:

Action description:

Once you identified the next action to take, identify the ref number of the element on which you need to perform the action. Write your answer in the following format:

Dom element ref number:

A.2 Month Distribution in choose-date

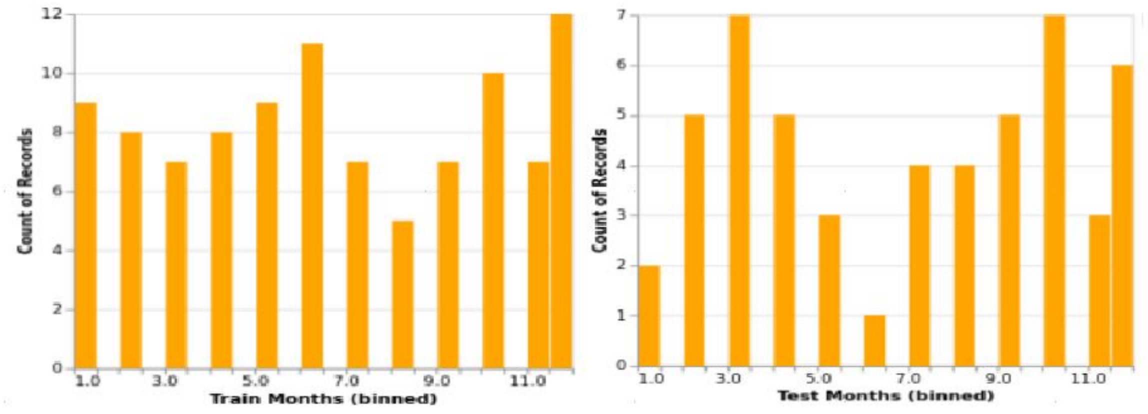


Figure 5: Testing trajectory for choose-date in July