Fine-tuning Digital Agents with BAGEL Trajectories

Stanford CS224N Custom Project

Alfred Yu
Department of Symbolic Systems
Stanford University
alfredyu@stanford.edu

An Doan
Department of Computer Science
Stanford University
andoan@stanford.edu

Abstract

There is significant interest in developing language model (LM) web agents to perform human tasks in digital environments. Web environments present difficult challenges to these LM agents - oftentimes, they struggle to perform well without human demonstrations to help contextualize the environment and its large action space. The ability to create synthetic demonstrations is of great interest as an alternative to collecting human demonstrations. Thus, instead of collecting expert demonstrations with human supervision, BAGEL Murty et al. (2024) provides a streamlined approach to generating synthetic trajectories, with usage of LMs. The goal of our work is to 1) create an existing version of BAGEL that can be freely shared and 2) hopefully contribute to the growing literature around digital agents and synthetic demonstrations. Our paper differs: instead of the original approach using the synthetic demonstrations to adapt the LM agent via retrieval augmented generation, we use the demonstrations to fine-tune our LM agent. We use a Gemini 1.5 Pro model to perform round trip iterations of trajectory generation and trajectory relabelling via a trajectory-first BAGEL run through, and the resulting trajectories saved are then broken down into states and actions, which are passed in for supervised fine-tuning on a smaller, zero-shot Gemini 1.0 Pro model to assess performance. On average, fine-tuning introduced a 6% improvement in accuracy across all tasks compared to the baseline. On average, introducing zero-shot Chain-of-Thought prompting didn't lead to meaningful improvement on task performance.

1 Key Information to include

- Mentor: Shikhar Murty
- External Collaborators (if you have any): Flor Lozano-Byrne (we worked on the same codebase starting out, specifically the initial exploration and instruction following policies and prompting)
- Sharing project: N/A
- Team Contributions: Alfred Yu contributed on the generating, gathering, and creating data visuals as well as the code to analyze success rates and also wrote the instruction following, exploration policies, and the initial trajectory generator draft. An Doan contributed on the implementation of the demonstration filter, trajectory relabeling policies, and instruction generation policies described in the original BAGEL paper, the round trip trajectory first iterative approach to BAGEL, code for generating and saving trajectories for success rate analysis, and code to evaluate policy specific performance of the demonstration filter and trajectory relabelling policy. Both parties contributed to the writing of this report.

2 Introduction

Recent advancements in natural language processing have been extremely exciting, and many research studies have been conducted about their efficacy in a variety of tasks. One such task is the

ability to be a web agent, able to navigate a website autonomously, without human intervention. Successful versions of web agents have been created, however, a common obstacle to deploying these web agents for usage is the need for human generated data. Web agents have trouble interpreting the vast amount of options available for any given website, and as such, need human generated trajectories to guide them. These trajectories are expensive, and generally difficult to obtain. Thus, the idea of creating synthetic trajectories has been one of great interest. Currently, not much research has been done on this specific area. There has been prior work done on this field, specifically with regard to retroactively labeling trajectories from web agents, and collecting generated demonstrations in that manner Sumers et al. (2023). However, consider that this work comes with the implicit understanding that the agent is familiar with the environments, and in the case of a zero-shot digital agent, this will not be true. This means that there doesn't exist much research done in the domain of zero-shot digital agents and synthetically created data.

The key idea of our approach is to implement each component of BAGEL, with corresponding ReAct prompting, and then use the BAGEL method to generate synthetic trajectories. Then, these synthetic trajectories are taken and used for supervised fine-tuning on a smaller model. Our goal is to then compare the performance of this smaller model on our baselines to analyze the efficacy of BAGEL in fine-tuning. The key points of our results demonstrate that fine-tuning is indeed the right path to go and point the way to future studies on fine-tuning with potentially more data.

3 Related Work

Human-Generated Demonstrations As mentioned previously, work on the area of digital agents has focused significantly on doing so with expert demonstrations, and human labor to generate data. The first few works in AI all involved collaboration with human instructions Allen et al. (2007). Work extending on this has evolved to make use of both a reward function and expert demonstrations, which BAGEL does not make use of Liu et al. (2018).

Synthetic Demonstrations There has also been much less done in the field of agents with synthetic trajectories being generated. A paper by Sumers et al analyzed the efficacy of using VLMs to generate language describing agent behavior, teaching these tasks to embodied agents Sumers et al. (2023). However, this paper, although related to BAGEL in the domain of analyzing web agents based off synthetic data, differs in that it assumes familiarity with the web environment, something not likely to happen for a zero-shot agent. Another interesting piece of research that aims to tackle the same problem was proposed by Gur et. al, in a paper where they propose WebAgent: a combination of HTML-T5 to parse HTML and predict sub-instructions, and Flan-U-PaLM to generate Python code. We found this paper to be similar because it also studies the domain of web agents that don't rely on human generated data, however, it offers a new way to approach the problem of web agents, not relying on the generation of natural language instruction, but rather, the generation of HTML snippets and Python code. Gur et al. (2024).

Prompt Engineering We also find that because BAGEL is heavily reliant on LMs, prompt engineering is essential. As such, we followed the original prompts in BAGEL, which were ReAct inspired Yao et al. (2023). Since this project is heavily reliant on zero-shot agents and prompting, we were also inspired by zero-shot Chain of Thought prompting, for which we created an additional experimental group to analyze difference in results Kojima et al. (2023). We do want to know that Chain of Thought was implemented only in the finetuned model. A future study may analyze its efficacy in potentially improving the process overall when done with trajectory generation as well.

Our paper is mainly inspired and related to Murty's paper introducing BAGEL (Bootstrapping Agents by Guiding Exploration with Language), a novel approach in bootstrapping LM agents while doing away with the need for human generated data. At a high level, we utilize the components of BAGEL to generate trajectories, which we then finetune on a smaller LM. We operate with MiniWoB++, a library consisting of many web interaction environments from tasks ranging to tic-tac-toe, to sending an email to a certain individual with given text Shi et al. (2017).

In this research paper, we contribute to the continuing literature on digital agents by analysis of a model finetuned by BAGEL trajectories, which the original paper didn't have. Furthermore, since

BAGEL isn't an open-source project, we contribute by introducing our own implementation of BAGEL, replicating the original paper's steps, and thus, creating an implementation that has a shareable and usable codebase which may be helpful for future studies.

4 Approach

Our architecture is primarily based off of BAGEL, presented by a paper from Murty et al Murty et al. (2024). As such, we implemented our own versions of each component of BAGEL from scratch, without the aid of an existing code base. We acknowledge that there are possible differences within the code, especially as we didn't have an existing code base to work with, but we followed the general guidelines outlined by the paper. Furthermore, it's worth noting that the actual language model we used was different; we used Gemini 1.5 and 1.0 Pro via the Vertex API, while the original paper used an instruction-tuned PaLM-2 LM.

We began with an implementation of BAGEL following Llama 3-70b, using the Together AI API to access the model in our code. Thus, we began importing our BAGEL implementation to Gemini 1.5 Pro due to token limits. We occasionally modified prompts accordingly to fit in with Gemini's own responses. One such example being that the original prompts given in Murty's paper led to Gemini generating the entire history of action sequences rather than the one best action that was new. As such, an explicit statement to only generate one action was thus added within the prompt to ensure only one action was outputted. We made sure to keep the original essence and meaning of the prompts intact. For our baseline, we used a zero-shot instruction following Gemini 1.5 Pro and Gemini 1.0 Pro to see how well the agents could do at the tasks without given any prior fine-tuning or manipulation outside of the instruction following policy. Essentially, the main goal was to get an idea of where Gemini was before BAGEL fine-tuning was introduced.

For BAGEL, we used Gemini 1.5 Pro as the base LM for our experiments. Since Gemini 1.5 Pro has a high temperature (the default is around 1.0), we modified our temperature to 0, allowing for easier Regex parsing and a more deterministic response. We noticed that even with the modifications to temperature, there were still weird symbols that could appear in Gemini's response. As such, we filtered these symbols out and parsed our responses into a standardized, clean format, then did Regex parsing in order to extract the important information properly. Another important difference is that we made use of Gemini 1.5's multimodality - Gemini 1.5 Pro was provided with images of the websites after the actions that it took with the goal of providing better and more informed trajectories. Gemini 1.0 Pro, however, is not multimodal, so its prompts involved no usage of images. The goal with Gemini 1.5's multimodality usage was to hopefully provide images of the webpages, which would supply our eventually finetuned Gemini 1.0 Model with better trajectories.

Although we implemented both trajectory first and instruction first versions of BAGEL iterations, our mentor suggested we choose one, and we chose to go with trajectory first BAGEL for our iterations. Ablative studies in the original paper showed that a Trajectory-First approach typically outperformed an Instruction-First approach, especially in MiniWoB++ environments. For trajectory-first BAGEL, we sampled $\tau^0 \sim p_{\text{explore}}(\cdot)$, that is, we first use the exploration policy to get a trajectory τ. We then successively relabel our iterations until a satisfactory instruction g and trajectory pair τ is determined, so continuous iterations of $g^t \sim p_{\text{label}}(\cdot \mid \tau^t)$ and $\tau^{t+1} \sim p_{\text{agent}}(\cdot \mid g^t)$. Essentially, we sample a new instruction based off of our trajectory, and then sample a new trajectory based off of our new instruction. We also have a demonstration filter s, which takes in an instruction, trajectory pair and rates it on a discretized scale from 1 to 5. In other words, $s(g, \tau) \in \{1, 5\}$. We have a very strict standard for accepting a pair, only doing so if they score 5 on the demonstration filter. This continuous relabeling process is then ran for 5 iterations, or if our demonstration filter ever returns 5 for a particular pairing, similar to the original paper by Murty. Murty et al. (2024)

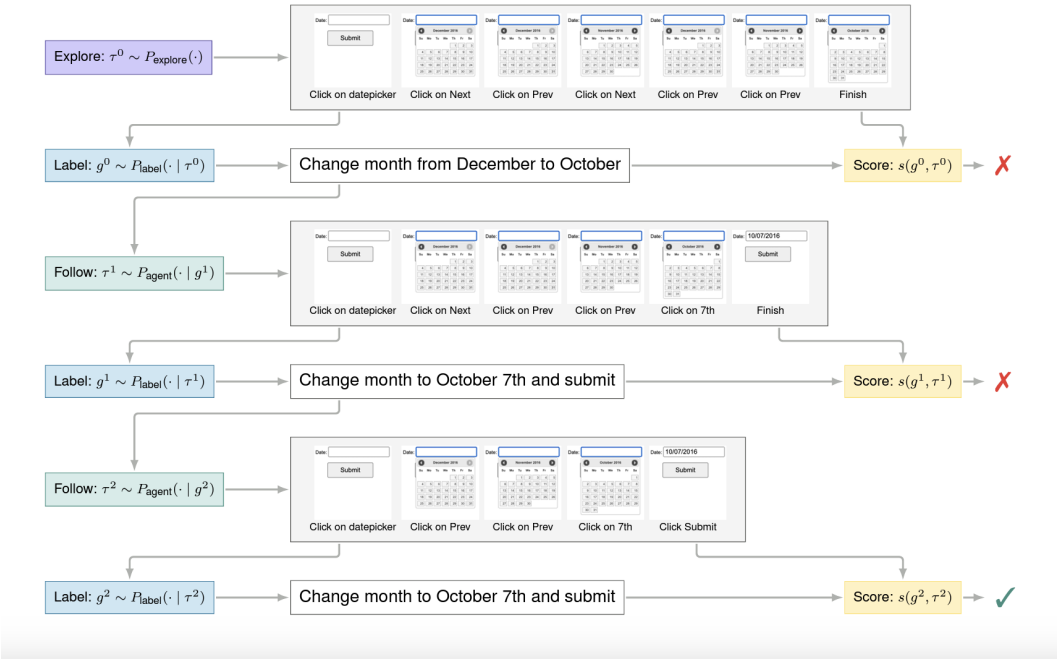


Figure 1: An example of how the trajectory-first policy works in BAGEL.Murty et al. (2024)

5 Experiments

5.1 Data

5.1.1 MiniWoB++

We use the nine web interaction environments from the MiniWoB++ Shi et al. (2017) library as web benchmarks for our research: 'book-flight-nodelay', 'choose-date-nodelay', 'social-media', 'email-inbox', 'click-checkboxes-soft', 'click-tab-2-hard', 'social-media-some', 'use-autocomplete-nodelay', and 'search-engine'. Among the task environments in MiniWoB++, we specify the shared action space in natural language similar to the original BAGEL implementation Murty et al. (2024). (See Appendix). We further removed 'tic-tac-toe' from our consideration of web benchmarks, since that task is an outlier in its scope (it's unlikely a web agent would ever need to play tic-tac-toe) and our demonstration filter would sometimes refuse to return a score, instead just saying the game was won. We never observed this output with any other task.

5.1.2 Finetuning Data

The dataset we're using involves trajectories generated through repeated iterations of our own implementation of BAGEL. We began with 5 iterations of trajectory-first version of BAGEL - using the Demonstration Filter to only keep trajectories that were scored a 5 by the LM. We aimed for a training data of 20 trajectories for all tasks, with the exception of book-flight-nodelay (due to the exceptional time it took to generate trajectories, sometimes around 1-2 an hour, and the fact that most trajectories were similar in format, we opted to generate only 9). From this set of trajectories, we generated 412 training data points and 102 validation data points. Each data point was a dictionary containing a time, action, dom element. We finetuned our model to output the next action based off of what the current dom element state, what the web page looked like after we that action would be performed, all based off the given goal.

5.2 Evaluation method

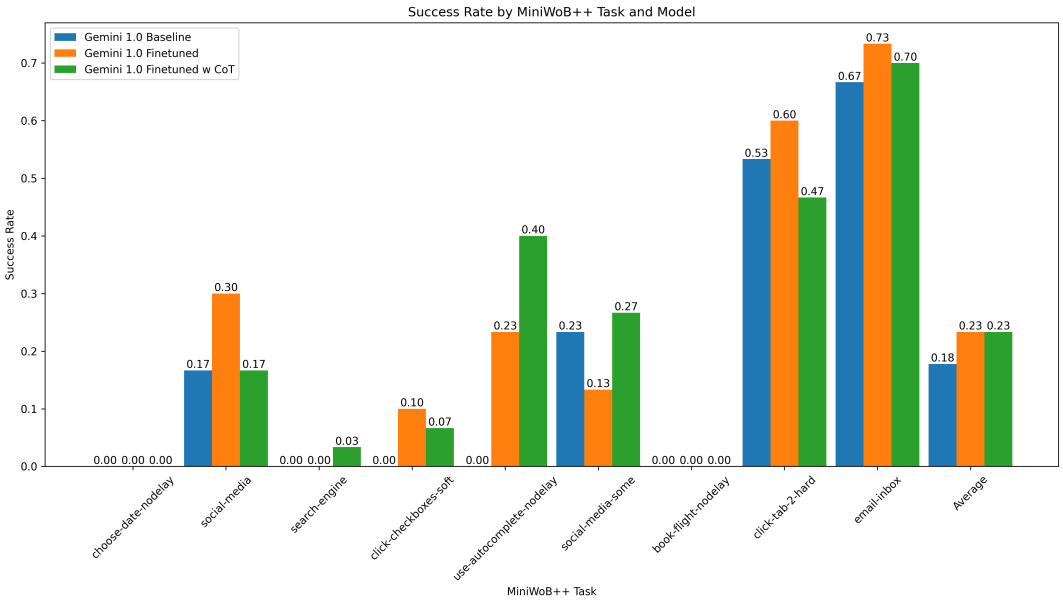
We evaluated the task based off of binary reward. The MiniWoB++ environment assigns a task a reward of either -1 or 1 (-1 for failure, 1 for success), and we count the success rate by measuring the

the count of 1s over the total runs. With these metrics, we hoped to gain a sense of how the model would do on the tasks and how performance would work. These metrics are similar to ones reported in prior papers, and gave us a sense of how well the model was performing on new tasks by purely focusing on whether a successful trajectory was generated or not.

5.3 Experimental details

Report how you ran your experiments (e.g., model configurations, learning rate, training time, etc.)
We ran our experiments with the Vertex AI API, choosing Gemini 1.5 Pro with a temperature of 0. We then chose a smaller model to fine-tune, opting for Gemini 1.0 Pro. We then ran the trajectory relabeler, aiming to generate 20 trajectories for each task, with the exception of book-flight-nodelay. Generating trajectories was a massive time block. The shortest trajectories took 3-4 hours, the longer ones could take upwards of 7+. book-flight-nodelay was extremely excruciating, and took around 12 hours to generate 9 trajectories. We and our mentor hypothesize that a large contributor to this was Gemini's slow speed, and the innate time sleep we introduced to prevent our models from overrunning the quota rate per minute.
We ran our fine-tuning with the Vertex AI API, running fine-tuning for 4 epochs across our dataset. The Vertex AI API uses PEFT (parameter efficient fine tuning), opting to fine-tuning a small subset of parameters, but other details are abstracted away from the user. The fine-tuning took around 4 hours. We set the learning rate multiplier to 1, and prepared our dataset into json-lines before finetuning.

5.4 Results



:

Figure 2: The success rates of our model.

These results are all from a sample pool of 30 trajectories per task. We first recorded our initial baseline with Gemini 1.5 (see Appendix). We then recorded a baseline for Gemini 1.0 in order for comparison with our fine-tuned models. We then also introduced a zero-shot Chain of Thought prompting experimental group, with the words "Let's think step by step." appended to the end of the prompt.

Our quantitative results were a little underwhelming. They were worse than we expected - we think that this is because of the little data we had for finetuning. Unfortunately, since trajectory relabeling took significantly longer than we anticipated, we weren't able to generate that many trajectories in time for the project. The general improvement in fine-tuning though, tells us that our approach is in the right direction and we might just need more data to exhibit full results. What was interesting was the discrepancy between Chain of Thought prompting and the finetuned results. We

hypothesize that if CoT was implemented in earlier parts of the project as well (such as trajectory relabeling), there would be a more demonstrable performance increase. Interestingly enough, with CoT implemented at the end of the prompt, we noticed that the model would make actions that didn't entirely make sense in tasks like book-flight-nodelay (in which it would type in location specified by the "to" field into the "from" field), and click-checkboxes-soft (where it would occasionally check checkboxes of words that were completely different in definition). Similarly, the model did poorly for book-flight-nodelay across all conditions, and we note that sometimes it would no nonsensical actions like click the wrong element. We hypothesize that having additional words in the prompt may confuse the model due to Gemini 1.0's limited context length. We noticed that adding or removing other elements from the prompt also affected model behavior in interesting ways, where sometimes telling the model to emphasize one aspect of the prompt would lead to the model doing nonsensical behavior like clicking the submit button immediately (which usually leads to a failure on the task). Experiments done on a larger language model with greater context length would need to be done to verify our hypothesis.

6 Analysis

For a further understanding of how BAGEL demonstrations work or fail to achieve improvement in agent performance, we analyze specific synthetic trajectories as examples and their actions and evaluate how policies in our implementation of BAGEL can contribute to success and failure of our final fine-tuned model.

6.1 Considerations of Synthetic Trajectories

6.1.1 Effect of Limitations on Trajectory Generation

One limitation is the inherent reliance of DOM elements, as well as the models available for us to use. Due to time and resource constraints, we opted to finetune the smaller and less advanced Gemini 1.0 Pro model. Due to this, we also had issues with token counts once again, and had to opt between removing tokens that were extraneous or not having evaluations on certain datasets. We thus had to remove information from DOM datasets that didn't allow for as granular an analysis as we would have liked. Information like the color of the DOM elements was removed, and floating point numbers were converted to integers to reduce tokens. We didn't want to compromise any more information in our DOM, and we couldn't find anything else extraneous to parse out, but still occasionally faced token errors. We suggest that future studies approach this project with HTML parsing, and a method to bypass including the DOM elements. Alternatively, as multimodal LMs become more prevalent and advanced, including an image within the prompt is something that may suffice.

6.2 Policy Specific Analysis

In this subsection, we focus on how policies in our implementation of BAGEL perform and how that can affect overall results of how synthetic demonstrations are generated and how this affects the overall fine-tuned model. Specifically, we delve into details of the demonstration filter and the trajectory relabelling filter defined in the original paper Murty et al. (2024), and evaluate their performance.

6.2.1 Performance of the Demonstration Filter

As specified in the original BAGEL paper, the demonstration filter takes as input a synthetic demonstration (g, τ) and makes a binary judgement based on how well τ corresponds to the instruction g, and will save the demonstration based on its judgement. The exact prompting for our implementation of the policy can be seen in the Appendix.
As mentioned, the demonstration filter is the element in the BAGEL pipeline that determines whether or not the synthetic demonstration is good enough to save. In the context of this research, the performance of the demonstration filter determines which trajectories are saved and used for fine-tuning our LM.

We evaluate our demonstration filter through success metrics of our Gemini 1.5 Pro baseline generated trajectories. In our implementation of the demonstration filter, it returns a score of 1-5 depending on

how well the demonstration trajectory performs its goal and the reasoning for giving that score. Using our previously generated trajectories from the MiniWoB++ environment, we have access to the task goal and the reward (whether or not the task succeeds). We define the success of the demonstration filter as returning a score of 5 when the reward is 1, and a score less than 5 when the associated reward of the trajectory is -1.

Task	Accuracy	Precision	Recall	F1
book-flight-nodelay	1.0	N/A*	N/A*	N/A*
choose-date-nodelay	1.0	N/A*	N/A*	N/A*
social-media	0.149	1.0	0.024	0.048
social-media-some	0.561	0.842	0.516	0.640
email-inbox	0.125	1.0	0.054	0.103
click-checkboxes-soft	0.841	0.842	0.667	0.741
click-tab-2-hard	0.524	1.0	0.091	0.167
use-autocomplete-nodelay	0.833	0.0	0.0	N/A*
search-engine	0.7	1.0	0.1	0.182
Average	0.599	0.780	0.204	0.323

Table 1: We measure accuracy (number of successes from the demonstration filter / total trajectories), precision, recall, and the F1 score for each task, as well as the described metrics for all tasks combined (specified by the 'Average' row). For each *Data cells that are specified N/A means a division by zero (Meaning for precision, there were no trajectories scoring 5, recall means that no trajectories were successful (reward of 1), and for F1 that both precision and recall were 0.

```
{'score': 4, 'reasoning': "The web-agent was trying to find the
email thread by Raven and reply to it with the text Velit, non, urna.
Mauris. It successfully opened up the reply window for Raven's email
and inputted the correct text. However, it did not actually send
the reply. The web-agent successfully found Raven's email thread and
opened the reply window. It also correctly inputted the desired reply
text. However, it failed to send the reply."}

{'score': 4, 'reasoning': "The web-agent was asked to switch to
the tab number 4 and click the link Euismod . Looking at the final
webpage, we can see that the tab 4 is indeed active and the link
Euismod is present. However, there is no indication that the link was
actually clicked.The web-agent successfully switched to the correct
tab, but it's unclear whether it clicked the link Euismod ."}
```

Figure 3: Above shows examples of the parsed response returned by the demonstration filter for the following tasks (in order): 'email-inbox', 'click-tab-2-hard'. **Note** that for each of these trajectories, the web agent did in fact complete the task and resulted in a reward from the MiniWoB environment of 1.

As observed, both qualitatively from the demonstration filter's specific scores and reasoning and from the table of success metrics above (specifically looking at recall), that there are many times were the demonstration filter incorrectly labels a successful trajectory with a subpar score, specifying that it did not fully complete the task (false negative). We can look at the reasoning that the LM generates where it specifies that it is unsure if the web agent was able to complete the final task (either of submission or clicking some other element) because at this point for MiniWoB++, the environment ends and the state of the webpage finishes, so the LM is unable to determine if the last step of the trajectory completed the task. However, as observed from our metrics, the precision of the demonstration filter is much higher than the recall, which is more important in how our synthetic trajectories are generated. This observation gives us insight to why synthetic generation is so time-consuming, because many successful trajectories are likely to not be saved / have a score of 5. However, the saved trajectories accuracy (reward matching the demonstration filter's score) is equivalent to the precision because we are only saving trajectories that have a score of 5 (true positives and false positives), and because these metrics are generally high across the board, we can have faith that the demonstration filter policy is able to filter out a significant portion of the failed trajectories.

6.2.2 Performance of the Trajectory Relabeler

The trajectory relabeler takes as input a synthetic demonstration, and attempts to hypothesize what the instruction is for that specific trajectory. In our iterative approach to BAGEL, the relabeler is fed into an LM agent that now . Here we analyze common patterns of our implementation of the trajectory relabeler and how that may affect the synthetic demonstrations generated through example.

```
{'reasoning': "There are no changes in the HTML structure or content
between the initial and final webpages. This suggests that the
instruction likely involved navigating or interacting with the page in
a way that didn't result in persistent changes.", 'answer': 'Scroll
to the bottom of the list of posts.'}
```

Figure 4: Above shows examples of the parsed response returned by the trajectory relabeling for the task 'social-media'.

One observation that was observed through many examples of trajectory relabelling, including the example above, is that if the original trajectory does not seem to have modified the webpage significantly, the trajectory relabeler will provide an instruction that is not very comprehensive, specific, or informative to the overall task (in this case, the 'social-media' task that depends on liking and reposting posts) that is still saved and used to fine-tune the model, resulting in little improvement through fine-tuning because the synthetic demonstrations are not exhaustive of the web environment. This may only be in the case of the MiniWoB++ environment, because normally, in a webpage environment, you're likely to get feedback in the case you submit.

7 Conclusion

The main conclusion is that finetuning on BAGEL trajectories does indeed improve performance. We suggest future studies analyze finetuning with an improved and more robust dataset.

Potential avenues for future work may involve pursuing other LMs, such as potentially looking at paid options. Unfortunately, our team reached many blockades because we decided to pursue options that wouldn't cost us money, and as such, we faced a lot of limitations. One such blockade was Gemini's Quota Limit via Vertex AI API. Due to this quota limit, we had to restrict our trajectory generation, since we introduced momentary sleep times to avoid prompting Gemini too often. A paid API may bypass this, and thus, allow for greater data generation, and of course, more data results in a likely better fine-tuned model.

Unfortunately, Gemini also proved to have very varied responses. Despite manually setting the temperature to 0, in an effort to better replicate the more deterministic instruction-tuned PaLM model uiltized in the original paper, we noted that Gemini still gave very varied responses.

Another avenue would be more closely mimicking the prompting of the LM that generates the trajectories by saving a running list of dictionaries containing the: action type, reference, text (if applicable) for fine-tuning data. In Gemini's fine-tuning documentation, it states, "The examples in your datasets should match your expected production traffic. If your dataset contains specific formatting, keywords, instructions, or information, the production data should be formatted in the same way and contain the same instructions." We explored the possibility of just saving the action type recommended, in part because of the token limits of the fine-tuning approach we took (we used the Vertex API's finetuning system, as we believed that would be more efficient since we were already reliant on the Vertex AI API). Unfortunately, we did not anticipate the significant bottleneck that Gemini introduced to our trajectory generation in terms of time, and thus, were forced to proceed with the data we already had.

We're also aware that other people pursued a similar project, however, they did away with the trajectory relabeling aspect of the project. Perhaps future work could also analyze the necessity of each of these individual components, especially because trajectory relabeling took us a long time.

8 Ethics Statement

In this section we present key considerations of the ethical challenges and possible societal risks associated with the development of language model (LM) web agents fine-tuned with synthetic demonstrations.

One potential ethical issue is how the agent could be manipulated by websites for extra profit. Consider the example of a shopping web environment where the agent is given the task of purchasing a purse for a customer, and the agent outputs a trajectory were it purchases the most expensive designer brand purse. It is possible that based on how the policies are prompted or by how the web environment is defined, agents could be intentionally trained on synthetic trajectories that lead to unethical situations acting to the disadvantage of customers. More concretely, the trajectory relabelling policy relies on an LM to relabel an existing iteration of a trajectory to a new task that more closely describes the goal that was attempted by the trajectory, so it is possible that the trajectory relabeler could relabel the goal from "Purchase a purse" to "Purchase the most expensive option when searching for purse," especially depending on the language of its prompting strategy. A mitigation strategy would be to adopt comprehensive ethical principles that guide the development of these LM web agents, starting from having specific and unbiased prompting for all policies that are part of the BAGEL pipeline. In this case, having biased vs. unbiased prompting for the relabelling policy or including an ethical scoring policy in the demonstration filter could mitigate the described ethical issue.

The reliance of BAGEL on LMs for its policies to generate synthetic trajectories can be prone to inherent bias and a societal fear of lack of transparency in how these policies are implemented. Ways that this could be mitigated would be to fine-tune on a varied dataset that explores all possible contexts of the web environment (as a way to potentially mitigate the biases that may come from pretraining on a homogeneous dataset), as well as employing a variety of language models to test out BAGEL. It may be worthwhile to attempt to implement BAGEL with a variety of LMs and compare those results to see if there's any major differences. Potentially, a team with more resources could look into how different models with paid APIs (such as the newly released GPT-4o) generate trajectories. These trajectories between models can then be qualitatively, or potentially even quantitatively, evaluated. The lack of transparency of transparency can be mitigated by providing detailed documentation on how prompting the policies is developed like in the original BAGEL paper Murty et al. (2024), providing additional documentation that clearly explains the decision-making processes and limitations, and implementing streamlined processes for accountability, such as logging and monitoring agent activities to ensure human oversight and intervention when necessary.

There is a concern of privacy and data security, specifically for web agents operating on web environments that hold private information such as websites that hold account logins/passwords, shopping sites that hold credit card or bank account information, etc. Synthetic demonstrations could unintentionally expose sensitive information whilst creating trajectories and exploring the action space, and could even lead to generating demonstrations that exploit this private information such as changing the credit card information or password in an account. Mitigation techniques would be similar to those described above, such as more guidelines to to ensure human oversight, or to manually remove actions that access private information from the web environment's action space (for example, preventing sign in or keeping the web agent in guest mode).

References

James F. Allen, Nathanael Chambers, George Ferguson, and Lucian Galescu. 2007. Plow: a collaborative task learning agent.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration.

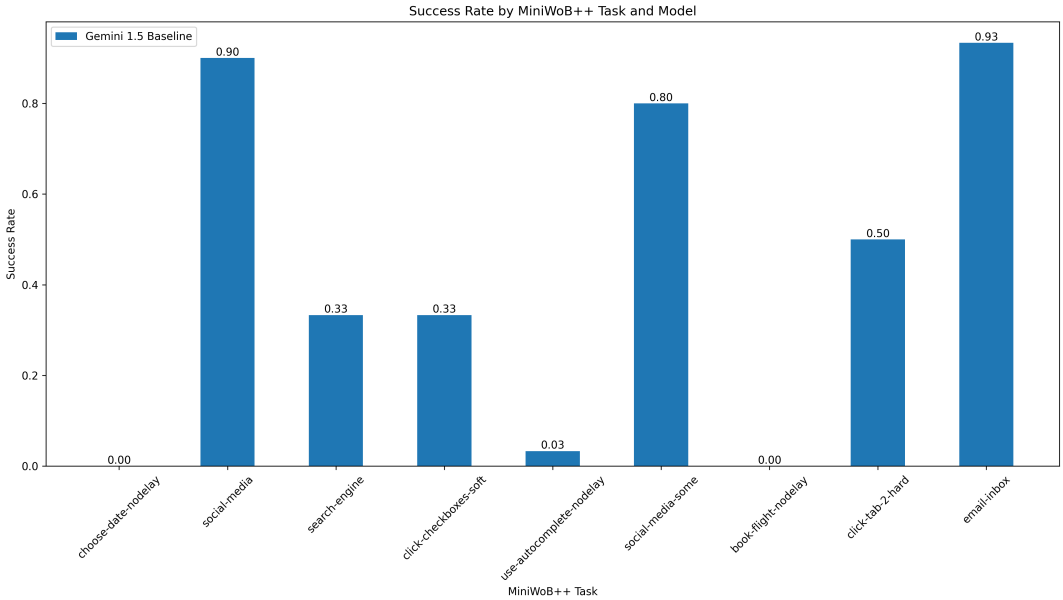
Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. 2024. Bagel: Bootstrapping agents by guiding exploration with language.

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.

Theodore Sumers, Kenneth Marino, Arun Ahuja, Rob Fergus, and Ishita Dasgupta. 2023. Distilling internet-scale vision-language models into embodied agents.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.

A Additional Baseline Plot



Above: The results of the Gemini 1.5 Baseline.

B Prompts

Note: all of the prompts are available within the code we submitted. Also, most of them remain the same from Murty's original paper Murty et al. (2024).

```
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 item, click on a right
arrow to go to the next item
- Click on the scroll bar to scroll up or down
- 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
```

Above: Action space for MiniWoB++. This is then directly used for various prompts as {inventory_str} similar to the prompting in the original paper.

```
A web-agent is given a precise instruction from a human, which it
carries out through a sequence of sub-tasks,
where each sub-task (such as clicking on elements / typing on elements
/ scrolling etc.)  changes the HTML state of the webpage.
You are given the initial webpage (as DOM elements), the final webpage
after all sub-tasks are carried out, as well
as a summary of changes that each sub-task made to the starting HTML
state.

Initial Webpage:
{inital_dom}
Final Webpage:
{final_dom}
Sub-tasks attempted by the web agent:
{subgoa_str}
Summary of changes made to HTML:
observation_changes

Your objective to guess the instruction that was given to the agent.
Ensure that your instructions are concrete and such that every
sub-task meaningfully contributes to fulfiling the instruction.  Start
by providing your reasoning.
Use the following format for your answer:
Reasoning:  your reasoning
Answer:  your guess of the instruction that was given to the agent
```

Above: Trajectory Relabeler prompt.

```
You are given an initial web-page from a website (as DOM elements).
To accomplish some task, a web-agent then interacts with the website,
leading to a final webpage.
Given the task, the initial webpage and the final webpage, your
objective is to judge how well the web-agent carried out this task
by giving it a score from 1 to 5.
Only give a score of 5 if the task is perfectly accomplished and the
final webpage has no errors.

Task:
{goal_str}

Initial Webpage:
{init_dom}

Final Webpage:
{final_dom}

Start by thinking about what the web-agent was trying to accomplish,
and describe how well it was done.  Provide your answer in the
following format:
Thought:
Answer:
Score:
```

Above: Demonstration Filter prompt.

```
You are a web-agent capable of executing the following kinds of tasks
on a webpage:
{inventory_str}
You are given the following goal:
{goal}
You observe the following DOM elements from the web page HTML:
{dom_elements}
Start by thinking about what action you should take next to achieve
the goal.  Write down all the different choices and then, choose the
best answer taking into account the following rules:
{element_rules}
Provide your answer for the ONE next action in the following format
(output *finished* if the instruction has been accomplished by
choosing an item from your inventory):

Action:
Text:
Kind of action and element name:

Once you've identified the next action to take, look at the DOM
elements from the web-page HTML:
{parsed_dom}
and identify the ref number of the element on which you need to
perform the action.
Remember to take into account all past actions, but include ONLY the
newest action in your response, as shown in the format below (where
Action is the action_code (TYPE_TEXT or CLICK_ELEMENT), Text is
whatever is typed in provided the action code is TYPE_TEXT, ref is
the reference number - don't add any random symbols)

Provide your answer in EXACTLY the following format:
Action:  the action_code (TYPE_TEXT or CLICK_ELEMENT)
Text:  whatever is typed in provided the action code is TYPE_TEXT Ref:
the reference number.
```

Above: Instruction following prompt.