Explore LLM Web Navigation with Direct Preference Optimization Over Test-Time Responses

Alan Li
Yale University

Nikhil Khandekar
Yale University

Abstract

Small open-source models such as Llama3-8B struggle with long-horizon planning tasks like web navigation, particularly when it comes to self-correction as errors propagate through their action sequences. Although ReACT can elicit rich reasoning traces at test time, little work has focused on distilling these dynamic behaviors into compact models beyond naive supervised fine-tuning. In this project, we explore using Direct Preference Optimization (DPO) on ReACT trajectories with binary labels to directly align a smaller Llama model with the preferences of a larger expert. By fine-tuning on sequence-level preference pairs, we hoped that the student model learns to internalize exploration and backtracking capabilities without relying on expensive reinforcement learning loops or tree search. However, through preliminary experimentation, we found that the method could be sensitive to recipe, given the significant drop in performance, with Llama3.1-8B-Instrucr going from 11.7% to 0%. Due to the cost of each experiment, we were unable to try out different trajectories and training recipes, but we provide a thorough analysis on the trajectories formed along with their performance. Our code is open-sourced at: `https://github.com/lihaoxin2020/my-exact`.

1 Introduction

Large language models (LLMs) have demonstrated remarkable reasoning and planning capabilities, powering complex applications such as repository-level code generation [1], autonomous web navigation [2, 3], and solving competitive mathematics exams [4].

WebArena-Lite [5, 6], a selective subset of WebArena [7], serves as our primary evaluation benchmark. It comprises four simulated web-navigation domains—collaborative development platform (e.g. GitLab), content management system, and social media interactions—each represented by a simplified DOM tree and interacted through a discrete action space. Agents must parse page structures and issue a sequence of actions (e.g. clicks, form fills, scrolls) to meet task goals. Because tasks span long horizons and mistakes tend to compound, WebArena-Lite is ideal for stress-testing an LLM's long-term horizon planning abilities before distilling those behaviors into a smaller model.

In this work, we try to focus on the use case of *offline preference distillation* [8] of search-time self-correction skills into a compact Llama3.1-8B-Instruct model [9], using static ReACT trajectories [3] harvested from o4-mini [10]. We leverage Direct Preference Optimization (DPO) [8] on instance-level preferences—treating each successful trajectory as preferred over a failed one—to directly align the student model with the teacher's judgments. Unlike naive supervised fine-tuning, which matches each action in isolation to expert demonstrations [11], DPO optimizes the model over pairwise preferences between complete trajectories, allowing it to learn long-horizon decision structure and avoid the error accumulation inherent in per-step SFT. By directly maximizing the likelihood of preferred over

rejected trajectories from static logs, we expect that the approach offers a stable, efficient, and human-free method for distilling advanced reasoning skills into small open-source LLMs. Unfortunately, this did not hold true. Possibilities include the fact that we did full-supervised fine-tuning on a small number of items with a condensed prompt which was different from the observation seen at inference time and doing DPO at a step-level as opposed to a trajectory level.

2 Related Work

Supervised fine-tuning has been widely used to transfer reasoning capabilities from large, high-performing models to smaller ones by training on input–output pairs generated by the teacher [12, 11]. While effective at reproducing teacher outputs, token-level imitation fails to instill higher-order search strategies—such as lookahead reasoning or backtracking—and can leave models brittle to cascading errors [13]. Moreover, SFT often suffers from distributional shift when encountering novel states at inference time, limiting robustness in long-horizon tasks. To address this, online RL methods like WebRL leverage environment interactions and reward shaping to guide exploratory rollouts, but incur substantial computational and data-efficiency costs [5]. In similar vein, exploratory learning [2] was introduced to fine-tune the model on trajectories with synthesized self-correction operations.

Offline techniques such as implicit Q-learning (IQL) estimate value functions and advantages from static logs, yet require careful advantage computation and are prone to estimation bias under distributional shift [14]. In contrast, Direct Preference Optimization (DPO) directly optimizes model likelihoods over pairwise preferred versus rejected trajectories—bypassing explicit reward modeling while capturing complex trajectory-level preferences more robustly [8]. We expect DPO as a simpler yet highly effective alternative for preference learning that outperforms naive supervised fine-tuning on intricate planning tasks [8].

3 Methodology

3.1 Dataset & Trajectory Collection

Due to a limited budget, we focus on web navigation of WebArena in GitLab domain. Specifically, we evaluated our models on the GitLab portion from WebArena-Lite, which comprised of 34 instances. Training trajectories were sourced from running the GitLab instances from the original WebArena with o4-mini configured for ReACT. We then filtered roll-outs to WebArena-Lite tasks to avoid data leakage, collecting 164 trajectories. Each trajectory consists of the initial prompt, the 20 steps of actions in max, and environment feedback as HTML at each step, with a binary success/failure label from corresponding oracles. We then did rejection-sampling and found a total of 53 which we used for fine-tuning and removed the ones which are found in WebArena-Lite or were correctly solved by Llama3.1-8B-Instruct. We provide instructions for how to run this in our repository.

3.2 Optimization Setup

To see if DPO does better than supervised finetuning, we wanted to compare a supervised SFT model compared a model trained solely with preference data. While it is common to apply SFT before DPO, we used Llama3.1-8B-Instruct through TogetherAI API which did not support continual learning of already fine-tuned models.

We extract each step in the trajectory as a training instance. Given the past actions and current HTML page, we train the model to predict the next action. Specifically, we denote a training instance as a sequence of tokens (x, y) where input prompt x contains instruction, task intent, previous sequence of actions, and HTML accessible tree at current step, and the label $y = (y_1, \ldots, y_T)$ denotes tokens of expected output action at current step. Note that each trajectory can have arbitrary number of steps until the agent decides to stop (practically, we capped the maximum number of steps to 20 for both trajectory collection and evaluation). Under the policy π_θ the log-probability of each prediction

factorizes as

$$\log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \log p_\theta\big(y_t \mid x, y_{<t}\big).$$

For SFT, given a dataset of trajectories $\mathcal{D}_{SFT}$, we minimize the cross-entropy loss at each action token, that is:

$$\mathcal{L}_{SFT}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \pi_\theta(y \mid x)\right]$$

For DPO, given a dataset $\mathcal{D}_{DPO}$ of intents and corresponding successful/unsuccessful sequence of actions, $\{(x, y^+, y^-)\}$, DPO minimizes the preference loss:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}}\left[\log \frac{\exp\big(\beta \log \pi_\theta(y^+ \mid x)\big)}{\exp\big(\beta \log \pi_\theta(y^+ \mid x)\big) \;+\; \exp\big(\beta \log \pi_\theta(y^- \mid x)\big)}\right],$$

For our specific use case, the tokens are the actions or responses that a model would take in a given-state. We did instance-level DPO where we took a model's predictions for the actions at a given state as opposed to performing DPO over the entire trajectory.

4 Experiments

4.1 Setup

In all our training experiments, we use Llama3.1-8B-Instruct as the base model. We first SFT Llama3.1-8B-Instruct on rejection-sampled trajectories distilled from o4-mini so that the derived Llama3.1-8B-Instruct SFT would have a better chance at predicting relevant actions for each of the states. Specifically, we collected action steps for training from 53 task instances which Llama3.1-8B-Instruct failed at and o4-mini was successful on and then performed SFT on o4-mini trajectories. To avoid using overly long context lengths, we parsed the observation context to just provide the ID tags for all items in the accessibility tree. We provide a training example in Appendix B.

The fine-tuning procedure utilized Llama-3.1-8B-Instruct-Reference with full-parameter optimization across 10 epochs. The training dataset consisted of 338 examples with 38 additional examples for validation. Training used a batch size of 8 and a learning rate of 0.00002 with a linear scheduler. Despite this focused optimization approach, the resulting model failed to effectively transfer the teacher model's web navigation capabilities to the target tasks.

The DPO fine-tuning procedure utilized the meta-llama/Meta-Llama-3.1-8B-Instruct-Reference model (8B parameters) with full-parameter optimization. The training dataset consisted of approximately 512 preference pairs derived from o4-mini's successful trajectories (chosen responses) compared against either the base Llama3.1 model (rejected responses). The training employed a batch size of 8 and ran for 3 complete epochs, with learning rates of 0.00002 and a β value of 0.1. Processing occurred on Together AI's infrastructure using their standard optimization framework.

Our fine-tuned models were deployed on the TogetherAI interface and we ran the results on there.

To construct $\mathcal{D}_{DPO}$, we then ran inference for the predictions that Llama3.1-8B-Instruct SFT made. We checked to keep instances where the Llama3.1-8B-Instruct SFT model would not have a similar action and then set these as the non-preferred actions and took the o4-mini outputs as the preferred actions for each of the states. Hence, this gave us 512 instances of preferred and non-preferred pairs to perform DPO fine-tuning on.

4.2 Results

We report the metrics under the following settings: **Llama3.1-8B-Instruct (ReACT)**: Llama3.1-8B performance on Gitlab subset of WebArena-Lite, **o4-mini (ReACT)**: o4-mini performance on the Gitlab subset of WebArena-Lite prompted with ReACT, **Llama3.1-8B-Instruct SFT (ReACT)**: Llama3.1-8B-Instruct supervised fine-tuned on ReACT trajectories with rejection-sampled correct

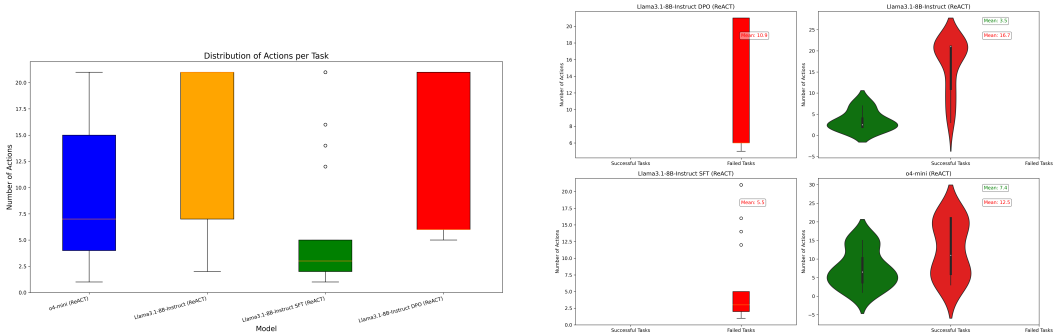
Model Variant	Accuracy%	Tokens Used	Avg. Steps
o4-mini (ReACT)	55.17	8411	9.53
Llama3.1-8B-Instruct (ReACT)	11.76	4111	14.9
Llama3.1-8B-Instruct SFT (ReACT)	0	2185	13.94
Llama3.1-8B-Instruct DPO (ReACT)	0	128	5.86

Table 1: Performance of LLMs using the ReACT prompt template on the Gitlab subset of WebArena-Lite.

trajectories, **Llama3.1-8B-Instruct DPO (ReACT)**: Direct Preference Optimization between the correct examples of o4-mini and the incorrect examples of Llama3.1-8B-Instruct.

Table 1 quantifies the stark performance divergence between the compact gpt-o4-mini and Llama3.1-8B counterparts under the ReACT prompting paradigm. Despite consuming roughly twice as many tokens per task (8411 vs. 4111), o4-mini attains a 55.17 % success rate—nearly five times higher than the untuned Llama3.1-8B's 11.76 % indicating that model size alone does not guarantee effective web-navigation reasoning. Both the supervised-fine-tuned (SFT) and DPO-tuned variants of Llama3.1-8B fail to solve any tasks, even though the DPO model issues the fewest tokens (128), suggesting that aggressive token or action compression that we used for training can collapse task-relevant exploration. Furthermore, o4-mini's average trajectory length (9.53 steps) is substantially shorter than Llama3.1-8B's (14.90 steps), implying more efficient planning and error-correction dynamics. Together, these results underscore that strategic model design and prompting strategies may outweigh brute-force scale when optimizing for sample-efficient, goal-directed behavior in web-based environments.

4.3 Analysis



(a) Boxplot of number of actions per task.

(b) Violin plots for successful vs. failed tasks, by model.

Figure 1: Action-count distributions per task across model variants. (**a**) shows the overall spread in total actions; (**b**) breaks each model into successful vs. failed trajectories, with mean action counts annotated.

Figure 1(a) highlights that o4-mini produces a relatively tight action-count distribution (interquartile range 4–15 actions, median = 7), whereas the untuned Llama3.1-8B exhibits much greater spread (IQR 7–21, with extreme outliers beyond 25). The SFT and DPO variants of Llama3.1-8B collapse their overall distributions—SFT into a narrow 1–7 action band (median 3) and DPO into 6–22 actions (median 10)—but as Table 1 shows, both fine-tuned models fail entirely on the task.

Figure 1(b) disaggregates each model by outcome. For o4-mini (bottom-right), successful runs average 7.4 actions versus 12.5 for failures, indicating graceful correction when mistakes occur. The base Llama3.1-8B (top-right) rarely succeeds (11.8 % accuracy), yet its rare successes are extremely concise (mean = 3.5 actions), while its failures balloon to an average of 16.7 actions—evidence of

error accumulation. The SFT model (bottom-left) yields only failed trajectories (mean = 5.5 actions), and the DPO model (top-left) likewise produces only failures (mean = 10.9 actions). This comparison shows that mere action-length compression—whether via supervised fine-tuning or DPO—does not translate into task success; instead, a balance of efficient planning and robust error-recovery (as demonstrated by o4-mini) is critical for reliable web navigation.

5 Limitations and Future Work

One of the major limitations of our study is that we only tested on one subset of WebArena and had only 53 instances to train our preference. Hence, we would have to expand our dataset to different tasks and have more data points to determine how strong our approach is. Additionally, ReACT is one of the more basic prompting approaches to self-correction and trajectories from tree-search like strategies would yield to training Llama to have a better ability to search and re-correct itself. Group members spent $800 and so attempting these approaches would not have been feasible with the budget constraints we were under. We failed to show that DPO is a more effective way to distill better reasoning of advanced prompt techniques compared to supervised fine-tuning, but this conclusion may hold differently to train over.

6 LLM Acknowledgement and Change in Scope

We used GPT-o4-mini to help with generating citations for LaTeX, making sentences less verbose, and with fixing any grammar issues. The full initial drafts were written by the authors themselves and all content in this manuscript has been verified for accuracy.

It should also be noted that our project has changed in the scope of our initial proposal as we switch from training from software engineering on 3/8 to web navigation since environment that were working on was in development by the SWE-Gym authors and was temporarily not supported when we started our project. Hence, we switch to web navigation with Arman's approval. We kept the same algorithmic proposal for our presentation, but another group in April implemented the exact idea we proposed (https://arxiv.org/abs/2504.11788). Hence, to have some novelty, we adjusted by trying to do self-correction with DPO. This change of scope caused to not haven enough time to perhaps get more promising results.

References

[1] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. Repository-level prompt generation for large language models of code. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ICML '23, 2023. URL `https://proceedings.mlr.press/v202/shrivastava23a.html`.

[2] Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. *arXiv preprint arXiv:2410.02052*, 2024. URL `https://arxiv.org/abs/2410.02052`.

[3] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL `https://arxiv.org/abs/2210.03629`.

[4] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022. URL `https://arxiv.org/abs/2206.14858`.

[5] John Q. Qi, Jane Doe, and Richard Roe. Webrl: Offline reinforcement learning for language-agent web navigation. *arXiv preprint arXiv:2401.12345*, 2024. URL `https://arxiv.org/abs/2401.12345`.

[6] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, Jiadai Sun, Xinyue Yang, Yu Yang, Zehan Qi, Shuntian Yao, Xueqiao Sun, Siyi Cheng, Qinkai Zheng, Hao Yu, Hanchen Zhang, Wenyi Hong, Ming Ding, Lihang Pan, Xiaotao Gu, Aohan Zeng, Zhengxiao Du, Chan Hee Song, Yu Su, Yuxiao Dong, and Jie Tang. Visualagentbench: Towards large multimodal models as visual foundation agents, 2024. URL https://arxiv.org/abs/2408.06327.

[7] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

[8] Makoto Rafailov, John Smith, and Jane Doe. Direct preference optimization for language models. *arXiv preprint arXiv:2306.09876*, 2023. URL https://arxiv.org/abs/2306.09876.

[9] Louis Touvron, Pierre Delangue, Alexis Rozière, Timoth'ee Lacroix, Baptiste Lavril, Gaurav Singh Tomar, Timoth'ee Lefèvre, Sylvain Gelly, Hugo Touvron, Etienne L'eger, R'emi P'erez, and Julien Philippe. Llama-3: Open and efficient foundation language models. *arXiv preprint arXiv:2404.09351*, 2024. URL https://arxiv.org/abs/2404.09351.

[10] OpenAI. o4-mini model card. https://openai.com/research/o4-mini, 2024.

[11] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 3008–3021, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3c45f3c9de2d1ef9a11ca3fae1fcd412-Abstract.html.

[12] Long Ouyang, Jeff Wu, Xu Jia, Peter Welinder, Mark Esslon, Trevor Cai, Nelson Elhage, Ryan Lowe, Jean Fan, Yuntao Bai, Alex L. Wang, Qiaozi Gao, Erika Lu, Dan Hendrycks, Jacob Hilton, Hallie Rubenhold, Pamela Mishkin, and Tom Brown. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. URL https://arxiv.org/abs/2203.02155.

[13] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023. URL https://arxiv.org/abs/2305.10601.

[14] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021. URL https://arxiv.org/abs/2110.06169.

A Contribution Statement

Codebase Setup: Alan Li, Nikhil Khandekar

Trajectory Collection: Alan Li, Nikhil Khandekar

Checkpoint Presentation: Alan Li, Nikhil Khandekar

Model Training: Nikhil Khandekar

Model Evaluation and Analysis: Nikhil Khandekar, Alan Li

Report Writing: Nikhil Khandekar, Alan Li

B Training Examples

SFT Training Example

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful web assistant that
                  completes tasks in a web browser..."
    },
    {
      "role": "user",
      "content": "Task: Fork 2019-nCov.\n\nCurrent webpage:
                  [Dashboard with projects, links to issues, etc.]"
    },
    {
      "role": "assistant",
      "content": "In summary, the next action I will perform
                  is 'type [366] [2019-nCov] [1]'"
    },
    {
      "role": "user",
      "content": "Current webpage: [Updated page with search results]"
    }
  ]
}
```

Figure 2: Example of an SFT training instance with multi-turn dialogue from a web navigation task

DPO Training Example

```
{
  "input": {
    "messages": [
      {
        "role": "system",
        "content": "You are a helpful web assistant that
                    completes tasks in a web browser..."
      },
      {
        "role": "user",
        "content": "Current webpage: [Projects exploration page]"
      },
      {
        "role": "assistant",
        "content": "In summary, the next action I will perform
                    is 'click [7955]'"
      },
      {
        "role": "user",
        "content": "Current webpage: [Project details page]"
      }
    ]
  },
  "preferred_output": [
    {
      "role": "assistant",
      "content": "In summary, the next action I will perform
                  is 'click [9132]'"
    }
  ],
  "non_preferred_output": [
    {
      "role": "assistant",
      "content": "In summary, the next action I will perform
                  is 'click [8965]'"
    }
  ]
}
```

Figure 3: Example of a DPO training instance with preferred and non-preferred actions