LASER: LLM Agent with State-Space Exploration for Web Navigation

Anonymous ACL submission

Abstract

Large language models (LLMs) have been successfully adapted for interactive decision-making tasks like web navigation. While achieving decent performance, previous methods implicitly assume a forward-only execution mode for the model, where they only provide oracle trajectories as in-context examples to guide the model on how to reason in the environment. Consequently, the model could not handle more challenging scenarios not covered in the in-context examples, e.g., mistakes, leading to sub-optimal performance. To address this issue, we propose to model the interactive task as state space exploration, where the LLM agent transitions among a pre-defined set of states by performing actions to complete the task. This formulation enables flexible backtracking, allowing the model to recover from errors easily. We evaluate our proposed **L**LM **A**gent with **S**tate-Space **E**xplo**R**ation (LASER) on the WebShop task. Experimental results show that LASER significantly outperforms previous methods and closes the gap with human performance on the web navigation task.

1 Introduction

Large language models (LLMs) such as GPT-4 (OpenAI, 2023) have achieved remarkable performance on a wide range of natural language understanding (NLU) tasks (Brown et al., 2020; Ouyang et al., 2022; Wei et al., 2023). Recently, they have been adapted to interactive decision-making tasks such as virtual home navigation (Yang et al., 2023), text-based games (Lin et al., 2023) or web-navigation (Yao et al., 2023b; Zhou et al., 2023). Previous methods that utilize LLMs to solve interactive tasks often implicitly assume a forward-only execution mode for the model, where they only provide a few oracle trajectories as in-context examples to teach the model how to reason step-by-step (Yao et al., 2023b; Sridhar et al., 2023). In other words, the correct action is selected at every step in those oracle trajectories. This might

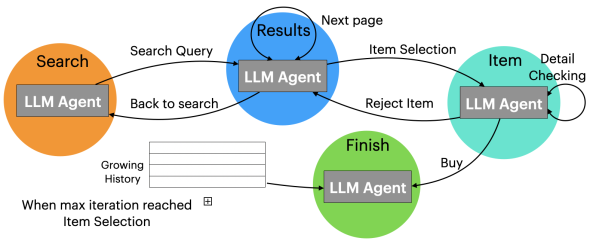

Figure 1: LASER's state transition diagram on the Webshop Task. Each solid circle represents a possible state, and the arrows represent possible state transitions. This formulation enables flexible backtracking and relieves the limitation of forward-only examples, allowing the model to better handle unfamiliar scenarios and recover from errors.

lead to sub-optimal performance because when the model makes an unexcepted mistake at test time, it would not know how to recover from it. At the same time, including many in-context examples to cover all possible scenarios is costly or unrealistic. Moreover, previous methods assume a global action space where the model is free to take any action at any step because they either define the possible actions at the beginning of the prompt or expect the LLM to figure out the possible action from in-context examples automatically. This might further increase the task's difficulty, and the LLM may perform invalid actions in certain cases.

To address the aforementioned issues, we propose to model the interactive tasks as state-space exploration. We first define a set of high-level possible states the LLM agent might encounter during the task execution. Then, we identify the possible action space in each state and the resulting states after performing each action. This formulation effectively converts the LLM agent's exploration in the interactive task as state transitions, where each action takes the agent from one state to another. Naturally, this allows the agent to easily recover from a wrong action: taking another action that would send it back to the previous state.

Moreover, our proposed formulation associates the action space with each individual state, which reduces the task's difficulty and allows the agent to always select the valid action at any step. We study our proposed method on the challenging Webshop (Yao et al., 2023a) task and build an LLM agent for web navigation. We show that our proposed setup enables the LLM agent to navigate effectively in an interactive environment to complete complex user instructions without using in-context examples. Overall, our proposed LASER significantly outperforms all previous baselines and closes the gap with human performance.

2 Methods

This section formally defines our notation for the interactive task and then describes the proposed LLM agent.

2.1 Problem Formulation

Given a web environment $\mathbf{E}$ and a user instruction $\mathbf{I}$, the agent is instantiated in the environment and provided with an initial observation $\mathbf{O_0}$. The agent is expected to perform a series of actions $\{a_0, a_1, ...a_n\}$ to complete the user instruction, where each a_i produces a new observation $\mathbf{O_i}$ when executed in the environment. S denotes the stopping state where the agent produces an output and stops exploration after reaching it. Finally, the agent's output is compared with the target to compute the metrics.

2.2 LLM Agent

As previously discussed, we would like the agent to be able to handle any novel situations or mistakes that might occur during execution without exhaustively describing them via a large number of in-context examples. Thus, we propose to equip LLM agents with the state-tracking capability. A diagram of the state transitions of our agent is shown in Figure 1. We start by defining a set of possible high-level states the agent might encounter in the environment (§2.3). The LLM agent takes the user input as the overall goal and is initialized in the starting state. At every step, the agent receives state-specific system instruction, current observation, a set of permissible actions in the current states, and the history of past thoughts and actions as inputs. Then, it selects one of the actions as output, which either transitions the agent to a different state or remains in the same state (§2.4). The agent repeats the process until the stopping state or the maximum step is reached.

Notice that with our formulation, we can provide detailed instructions to inform the agent of the possible situations in every state and how to handle them. For example, as shown in Figure 1, at the results state, the current results may or may not be good enough, and we instruct the agent to either select an item, go to the next page, or go back to search depending on its judgment. Hence, these instructions can be very informative to guide the agent while being much more efficient than in-context examples. Next, we describe in detail how we design the state and action spaces.

2.3 State Description

In previous literature, the term *state* is often used to represent the current environment the agent is in. In our work, we use the term *state* on a more generic level, and we consider an agent to be in two different states only if the *structure* of the representation of the current environment is different. In other words, if the agent receives two observations that share the same layout structure but with different details, we consider the agent to be in the same state. This allows us to define only a handful of states to support an agent's exploration in a complex environment fully.

After manually categorizing all possible states in the interactive task, for each state, we write a generic instruction that describes the state in detail. Specifically, we provide a sample layout of the observation the agent would receive in that state and replace all specifications in the layout with placeholders. We also provide a high-level goal and detailed instructions to act in that state. The sample layout combined with state-specific instructions allows us to inform the agent of possible observations it might receive and how to act accordingly. Therefore we no longer need to provide in-context examples to guide the agent. For the WebShop task, we define a total of four states, and the full prompts for search, results, and item states can be found in Table 3, Table 4 and Table 5 in the appendix.

2.4 Action Space

Previous methods often implicitly assume a global action space for the model, i.e. the model is free to take any action without further constraints. Although the LLM is able to figure out valid actions to take most of the time, it might still attempt to take invalid actions in certain cases. Thus after defining all possible states for the task, we further

	Success Rate	Reward
ASH (Sridhar et al., 2023)†	30.2	56.7
ReAct (Yao et al., 2023b)†*	40.0	66.6
ReAct (ours rerun)	34.0	59.7
WebGUM (Furuta et al., 2023)	45.0	67.5
LASER - backup	48.4	71.2
LASER	**50.0**	**75.6**
Human†	59.6	82.1

Table 1: Results on WebShop Task. † Results taken from previous papers. *They used a simplified setting where the number of items shown on each page is limited to 3.

	Success Rate	Reward
LASER	**52.0**	**77.6**
LASER + One-shot	50.0	74.9
LASER - function call	50.0	76.2
LASER (text-davinci-003)	38.5	70.2

Table 2: Ablation Results on the WebShop Task. We evaluated on 200 instead of 500 episodes due to a limited computing budget.

identify the action space for each state to rule out such possibilities. Specifically, we define a set of permissible actions that the agent can choose from for each state, which ensures that the agent always performs valid actions. The state-action mapping for our agent is shown in Table 7 in the appendix. In practice, permissible actions can also be determined heuristically, e.g., identifying all clickable buttons on a webpage.

Inspired by the success of the reason-then-act method (Yao et al., 2023b), we also ask the agent to produce a thought at every step and then select an action based on its thought. The agent keeps repeating the thought-and-action process until it reaches the stopping state or the maximum step is reached. We also define a memory buffer to store the intermediate results (the items examined but considered non-matching) during the exploration. This is similar to human behavior in that people typically find a few backup options before finding the desired item. When the agent is forced to stop after the maximum number of steps, it selects one of the intermediate results as the final output, and we call this the backup strategy.

3 Experiments

We conduct our experiments on the WebShop task (Yao et al., 2023a). We used 500 test set instructions for evaluation and adopted reward and success rate as metrics following previous works (Yao et al., 2023a). We used GPT-4-0613 to power LASER and its function-calling ability to implement action selection step. More detailed experimental setup is discussed in Appendix B. We compare against the following baselines: ReAct (Yao et al., 2023b) is a prompting method designed for interactive decision-making tasks. At every step, the LLM agent receives an observation and can either produce a thought or an action. The agent accumulates all of the past observations, thoughts, and actions in its prompt, using a full trajectory of exploration as an in-context example. The original ReAct uses PaLM (Chowdhery et al., 2022) as its LLM backbone. To make a fair comparison, we also rerun the ReAct method with GPT-4-0613. ASH (Sridhar et al., 2023) builds on top of React and adds a summarization step that condenses the agent observation and acts based on the condensed information. WebGUM (Furuta et al., 2023) is a supervised method that finetunes FlanT5-XL model (Chung et al., 2022) on 1K human demonstrations provided by the WebShop task.

4 Results

The overall results of our experiments are shown in Table 1. Our early experiments showed that the ReAct agent often produces invalid actions. For example, when it selects an item that doesn't match the instruction, it tries to go to the next page before backing to the results page. Also, the ReAct agent often got stuck in a certain action and failed to produce output. For example, the agent keeps going to the next page until the maximum step is reached. We added detailed instructions as the system prompt to try to address the issue. Despite our best efforts, the agent still makes invalid actions in some cases and achieves worse results than the original paper. On the other hand, LASER outperforms baselines by large margins on both metrics, showing the effectiveness of our approach. We further removed the backup strategy of LASER (the agent would receive a 0 score when the maximum budget runs out) to make a more fair comparison with ReAct. We see that our method still outperforms baselines by very large margins.

4.1 Analysis

We first conduct ablation studies to understand the important design decisions of our agent.

Zero-shot vs Few-shot We used state-specific in-

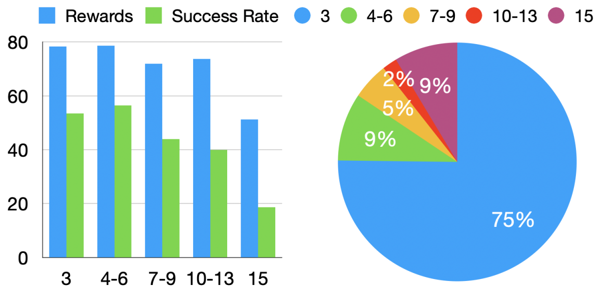

Figure 2: Left: LASER's performance for test set episodes of different lengths. Right: The distribution of the number of steps LASER takes to complete 500 test-set instructions.

structions only to guide our agent's exploration in the environment, whereas previous works often adopt in-context examples. To investigate if the agent can further benefit from in-context examples, we experimented with a one-shot setting: for every prompt in LASER, we added one example input-output pair between our system instructions and current inputs, and the rest of the agent remains the same. Due to the limited computing budget, we only ran our ablation studies on 200 episodes. The results are shown in Table 2. We see that adding an in-context example actually leads to worse performance. Since LASER already performs valid actions 100% time, we hypothesize that the agent understands the task well without in-context examples and the added example is actually distracting the agent in some cases.

Effect of function-calling LASER takes advantage of the function-calling functionality that is enabled only for GPT models after June 13th. Thus, we are interested to see the effect of replacing this design with regular text generation. To do so, instead of passing the permissible actions as a list of functions, we convert each action as a Python dictionary describing its purpose and arguments and then append them to the prompt. We then ask the LLM to generate output in JSON format to represent the action it selects with appropriate arguments. The results are shown in Table 2. Again, the agent without function calling performs slightly worse on these 200 episodes. It shows that the function calling functionality can be leveraged to boost performance when building interactive agents, suggesting a direction for building future LLMs.

Performance vs trajectory length Here, we are interested in seeing the length of LASER's trajectories and their effect on the overall performance. We plot the distribution of trajectory length in Figure 2 and the agent's performance for each length group. We notice that most of the time, the agent only took three state transitions to reach the finish state, which is search-select-buy. From the left figure, the agent's performance generally decreases as the trajectory gets longer. However, the drop is less significant compared to the observation made for ReAct and ASH agent (Sridhar et al., 2023), which further shows the effectiveness of our agent. Finally, for the length 15 group, for which the agent is forced to stop and select from the browsing history, the performance is much lower than other groups. While not surprising, it has a non-zero success rate, showing that there are cases where the agent found a matching item but failed to recognize it as the target in the first pass.

Generalization to different LLMs We leverage the most powerful LLM to date to build LASER, and we are interested to see if it can transfer well to another LLM. We adopted the text-davinci-003 model to see our agent's performance with a less powerful non-chat model. Since this model does not support function-calling, we adopted the approach described earlier to prompt the model to generate JSON output to represent actions. The results are shown in Table 2. Although switching to text-davinci-003 leads to a large drop in performance, our model still achieves better results than the baselines. It shows that our proposed agent can be easily adapted to other LLMs with different capabilities. With more powerful models in the future, our agent could potentially surpass human performance on this task. We also conducted case studies to inspect the failure modes of LASER and additional results are in Figure C. We discuss related works in Appendix A.

5 Conclusions

In this work, we proposed an LLM agent, LASER, that models interactive web navigation tasks as state-space exploration. Our formulation allows the agent to handle novel situations, easily backtrack from mistakes, and always perform valid actions. Guided solely by the state-specific instructions without any in-context examples, LASER outperforms all previous baselines on the WebShop task by large margins. Furthermore, our analysis shows that LASER is also more robust to longer trajectories and generalizes well to other LLMs.

Limitations

In this work, we have only conducted experiments on the Webshop task. Despite its challenging nature, the websites hosted in this task are still simplified. For future work, it would be interesting to apply our LASER to more challenging benchmarks (Zhou et al., 2023) and real-world shopping websites [1] to test its ability. Also, it would be interesting to equip LASER with more tools such as a knowledge retriever (Ma et al., 2023) or a calculator (Gao et al., 2022), so that it can handle more complex instructions.

Our LASER requires manual annotation of possible states in the environment and their corresponding descriptions. Because of this, our method might only be suitable for building agents for specific domains (rather than open-world web agents), where only a handful of states are required, e.g. e-commerce or travel booking. For future directions, we envision a hierarchical multi-agent system, in which each specific domain is governed by an agent like LASER, and a general open-world agent just collaborates with other domain agents to complete various user instructions.

Regarding potential risks of our work, we think extra caution and testing are required before deploying LASER to real-world scenarios. Since we only conduct experiments in a simulated environment, we allow the agent to take any action permitted in the environment. However, certain actions may have hard-to-recover consequences in the real world. For example, clicking the buy button in a real shopping site. As LASER's success rate is still far from being perfect, it might require addtional human verification before proceeding with actions that have high-stakes.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2022. Understanding html with large language models.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet,

[1] https://www.amazon.com/

Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner monologue: Embodied reasoning through planning with language models.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks.

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

Kaixin Ma, Hao Cheng, Yu Zhang, Xiaodong Liu, Eric Nyberg, and Jianfeng Gao. 2023. Chain-of-skills: A configurable model for open-domain question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1599–1618, Toronto, Canada. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Abishek Sridhar, Robert Lo, Frank F. Xu, Hao Zhu, and Shuyan Zhou. 2023. Hierarchical prompting assists large language model on web navigation.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023a. Webshop: Towards scalable real-world web interaction with grounded language agents.

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

Longtao Zheng, Rundong Wang, and Bo An. 2023. Synapse: Leveraging few-shot exemplars for human-level computer control.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. Webarena: A realistic web environment for building autonomous agents.

A Related Works

Interactive decision-making tasks such as web navigation have become popular recently (Liu et al., 2018; Yao et al., 2023a; Deng et al., 2023; Zhou et al., 2023), while some efforts have tried to solve these tasks by finetuning pretrained language models on a large corpus of demonstration data (Gur et al., 2022; Furuta et al., 2023), other attempted to build agents to navigate web environments solely relying on prompting LLMs (Yang et al., 2023). Among the LLM-based approaches, ReAct (Yao et al., 2023b) and InnerMonologue (Huang et al., 2022) equip the LLM with a thought process before producing actions. ASH (Sridhar et al., 2023) and WebAgent (Gur et al., 2023) focus on decomposing complex decision-making steps into a set of simpler steps, e.g. first summarizing the task-relevant content and then act upon it. Most similar to our work, Synapse (Zheng et al., 2023) also proposed to use state-conditional prompts to guide the LLM's action. However, their focus is on decomposing the few-shot examples into atomic parts whereas our agent uses state-specific instructions alone without in-context examples to complete tasks.

Another line of work focuses on the planning stage of LLM agents. Kim et al. (2023) proposed an agent RCI that generates a plan before acting, and then refines its action when encountering errors. Adaplanner (Sun et al., 2023) further enhanced the planning approach by adaptively updating the plan during the agent's execution. Reflexion (Shinn et al., 2023) agent refines its plan and actions by taking environmental feedback through a trial-and-error fashion. These approaches are orthogonal to our work and can be potentially combined with our agent to enhance its performance.

B Experimental Details

The WebShop provides a simulated environment for online shopping, containing 1,181,436 items collected from Amazon shopping sites. Additionally, the task provides human-annotated instructions for purchasing certain items and their corresponding target items. We followed previous works and used the 500 test set instructions to evaluate our LASER and evaluate with rewards and success rate, where the agent is considered successful if the purchased item perfectly matches the target item, otherwise, if the purchased item partially matches the target item, the agent receives a partial reward (scale between 0-100).

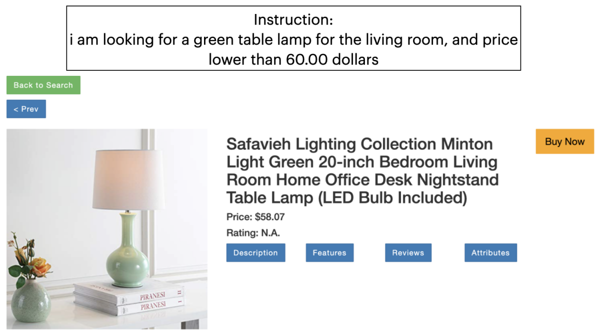

Figure 3: An example of the *Item good enough* error cases, the item selected by the agent is shown and the user instruction is on the top. The reward the agent receives is 0.666.

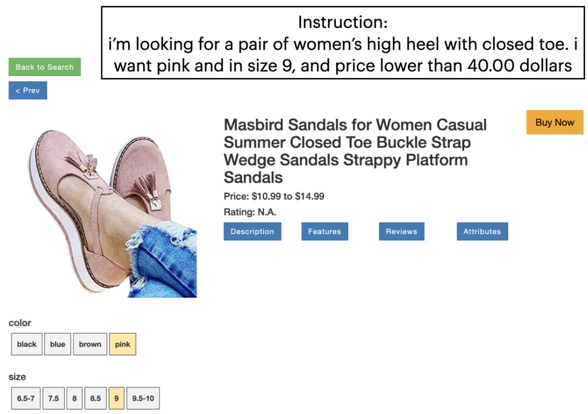

Figure 4: An example of the *Missing details* error cases, the item selected by the agent is shown and the user instruction is on the top. The reward the agent receives is 0.8.

For our method, we used the GPT-4-0613 to power our LASER. We used the function-calling functionality to implement the action selection step. In particular, we write a short description for each action and then pass them as a list to the function-call argument of the LLM to let the model select from. We allow our agent to make 15 state transitions in maximum. In practice, if the agent has not reached the finish state after 13 state transitions, we force it to select from the history to ensure it does not exceed the budget.

C Case Studies

We manually annotated 30 error cases from the Dev set to understand the failure cases of LASER. We broadly categorize the errors into three categories: *Item good enough*: the item selected by the agent meets the user instruction from the authors' perspective but did not receive a full score. We found that 9 out of 30 cases fall into this category, and an example is shown in Figure 3. The item found by the agent is indeed a green table lamp for the living room with a price within the budget, but it is considered incorrect. *Retrieval failure*: none of the items returned by the search engine meets the user requirement, despite that the agent used a suitable query for retrieval. We found 12 out of 30 cases fall into this category. We hypothesize that a more effective retriever or search engine can probably address these issues. *Missing details*: The item selected by the agent indeed does not match the user's instruction on certain details. We found that 9 out of 30 cases fall into this category, and an example is shown in Figure 4. In this example, although the color and size of the selected women's shoes both matched the user instructions, these are not high-heel shoes. This indicates that LASER can make mistakes when encountering items with many matching details, and it would be interesting to see if a self-feedback/verification module can address this issue (Madaan et al., 2023).

D Prompts used in our experiments

E Licenses

The Webshop task and ReAct method are both released under MIT License. They are both released for research purposes, and our experiments are consistent with their intended usage.

You are an intelligent shopping assistant that can help users find the right item. You are given an observation of the current web navigation session, in the following format:

Current Observation:
WebShop
Instruction:
{the user instruction}
[button] Search [button_] (generate a search query based on the user instruction and select this button to find relevant items)

Every button in the observation represents a possible action you can take. Based on the current observation, your task is to generate a rationale about the next action you should take. Note that if an history of past rationales and actions is provided, you should also consider the history when generating the rationale.

Table 3: The system instruction we used for the search state.

You are an intelligent shopping assistant that can help users find the right item. You are given an observation of the current web navigation session, in the following format:

Current Observation:
Instruction:
{the user instruction}
[button] Back to Search [button_] (select this button to go back to the search page)
Page current page number (Total results: total number of results)
[button] Next > [button_] (select this button to go to the next page of results)
[button] {item_id 1} [button_] (select this button to view item 1's details)
{name of item 1}
{price of item 1}
[button] {item_id 2} [button_] (select this button to view item 2's details)
{name of item 2}
{price of item 2}
[button] {item_id 3} [button_] (select this button to view item 3's details)
{name of item 3}
{price of item 3}
{More items...}

At this stage, you want to select an item that might match the user instruction. Note that even if an item has non-matching details with the user instruction, it might offer different customization options to allow you to match. E.g. an item may have color x in its name, but you can customize it to color y later, the customization options are shown after you select the item. Thus if an item name seems relevant or partially matches the instruction, you should select that item to check its details. If an item has been selected before (the button has been clicked), you should not select the same item again. In other words, do not select an item with [clicked button] item_id [clicked button_]. Prepare your response in the following format:
Rationale: the user wanted {keywords of the target item}, and we have found {matching keywords of item x}, thus item {item_id x} seems to be a match.

Table 4: The system instruction we used for the result state.

You are an intelligent shopping assistant that can help users find the right item. You are given an observation of the current web navigation session, in the following format:

Current Observation:
Instruction:
{the user instruction}
[button] Back to Search [button_] (select this button to go back to the search page)
[button] < Prev [button_] (select this button to go back to the previous page of results)
{Customization type1}:
[button] option1 [button_]
[button] option2 [button_]
{Customization type2}:
[button] option1 [button_]
[button] option2 [button_]
{more customization options... (if any)}
{Item name and details}
[button] Description [button_] (select this button to view the full description of the item)
[button] Features [button_] (select this button to view the full features of the item)
[button] Reviews [button_] (select this button to view the full reviews of the item)
[button] Buy Now [button_] (select this button to buy the item)

description: (if this is shown, the description button should not be selected again)
{full description of the item (if any) or "None"}

features: (if this is shown, the features button should not be selected again)
{full features of the item (if any) or "None"}

reviews: (if this is shown, the reviews button should not be selected again)
{full reviews of the item (if any) or "None"}

Target item details (what the user is looking for):
keywords: {keywords of the target item}
max_price: {the price of the item should not exceed this}

At this stage, you want to verify if the item matches the user instruction. You should consider the available customization options when deciding whether an item matches the user instruction. If an item can be customized to match the user instruction, or if the customization options cover the user specification, it is also a good match. If the item does not match the user instruction and it does not provide enough customization options, you can go to previous page to view other items. You can also check the item's description, features and reviews to view more details (Note that description, features and reviews could be "None", do not check them again if they are already given). Prepare your response in the following format:
Rationale: the user wanted {keywords of the target item}, and they required the following customization options: {cutomization of the target item}, the item is keywords of the item in the current observation, and it has the following customization options: {options available for the current item}, which {cover}/{not cover the user requirement}, thus we should {buy the item}/{check more details}/{go to previous page to view other items}

Table 5: The system instruction we used for the item state.

You are an intelligent shopping assistant that can help users find the right item. You are given an observation of the current environment and a rationale for the next action to be taken,
in the following format:

Current Observation:
The observation layout from search or result or item state, as shown from Table 3, Table 4 and Table 5

Next action rationale: {the rationale for the next action}

Your task is to perform one of the function calls based on the rationale.

Table 6: The system instruction we used for generating action from thought.

State	Available Actions
Search	{"name": "Search", "description": "Use this function to search for the target item in the inventory based on keywords"}
Result	{"name": "select_item", "description": "Use this function to select one of the items from the search results and check its details"} {"name": "Next", "description": "Use this function to go to the next page of search results to view more items, if none of the items on the current page match the user instruction."} {"name": "Back_to_Search", "description": "Use this function to go back to the initial search page. You should use this function only if you have browsed multiple pages of items and checked multiple items' details in the history, and none of the items match the user instruction."}
Item	{"name": "Description", "description": "Use this function to check the description of the item, if you are unsure if the item perfectly matches the user instruction"} {"name": "Features", "description": "Use this function to check the features of the item, if you are unsure if the item perfectly matches the user instruction"} {"name": "Reviews", "description": "Use this function to check the reviews of the item, if you are unsure if the item perfectly matches the user instruction"} {"name": "Buy_Now", "description": "Use this function to buy the current item, if the current item perfectly matches the user instruction."} {"name": "Prev", "description": "Use this fucntion to go back to the results page, if the current item does not match the user instruction "}

Table 7: The action space of our agent in each state. Each action is implemented as a function call following the guidelines from OpenAI [2], additional parameters used in the function call are omitted here for brevity.