EXPLORING THE PRE-CONDITIONS FOR MEMORY-LEARNING AGENTS

Vishwa Shah * **Vishruth Veerendranath***
Graham Neubig **Daniel Fried** **Zora Zhiruo Wang**
Language Technologies Institute, Carnegie Mellon University
{vishwavs, vveerend, gneubig, dfried, zhiruow}@andrew.cmu.edu

ABSTRACT

Digital agents supported by large language models (LLMs) have demonstrated potential in real-world tasks such as web navigation, especially with growing memory by learning from past experiences and applying them in later tasks (Wang et al., 2024). Nonetheless, it is unclear if arbitrary agents can benefit from this memory adaption procedure, or in other words, what are the pre-conditions for such adaptive-memory approaches to show effect. In this work, we first ask: *Does memory-learning agents necessitate certain model capabilities?* We apply AWM to top open-weight LLAMA and DEEPSEEK-R1 models and observe minimal gains; nonetheless, adopting or transferring memory induced by strong GPT increases success by 64–87%, suggesting that the memory induction module is critical and has stricter capability requirements. We further ask: *Can we optimize memory design to loosen the model capability constraint?* We propose to induce increasingly granular workflows and schedule their integration into agent memory via curriculum learning. While it shows little correctness improvement, it reduces the computation cost by 58.2%. Overall, we reveal the pre-conditions of effective memory-learning agents on memory design and induction quality.

1 INTRODUCTION

Large language models (LLMs) are increasingly used in agentic applications in the digital world such as web browsing (Yao et al., 2022a) and navigation (Zhou et al., 2023; Deng et al., 2024). While current LLM-based agents still face challenges in solving complex, long-horizon tasks effectively, incorporating autonomous self-improving modules such as planning (Koh et al., 2024b), reasoning (Yao et al., 2022b), perception augmentation (Zheng et al., 2024) and adaptive memory Wang et al. (2024) often show promising improvements.

Building agents with growing memory has becoming a promising way to realize self-improving agents. As it enables agents to store their previous experiences as precisely recorded (Zheng et al., 2023) or in further abstracted formats (Wang et al., 2024), and then learn from them in non-parametric (Wang et al., 2024; Zheng et al., 2023; Wang et al., 2023) or parametric (Patel et al., 2024; Murty et al., 2024a) means, to improve their performance autonomously. In particular, AWM (Wang et al., 2024) shows improvements in the *online* setting, where memory is produced and integrated by the agent autonomously during inference without any labeled data. While AWM is shown effective with strong GPT models, this method seems to pose certain quality requirements on the agent LLM backbone, especially in memory induction and utilization processes; therefore making it unclear if AWM can be readily extrapolated to agents built upon weaker LLMs.

Therefore, in this work, we first raise the question: *Does building adaptive-learning agents necessitate certain model capabilities?* (§3) To study this, we adopt agent workflow memory (AWM, (Wang et al., 2024)) as a representative for the memory-learning agent, and first apply it to presumably weaker open models — LLAMA and DEEPSEEK-R1. We find that solely applying the weaker model backbone in all three components (memory-using task-solving, trajectory evaluation, and memory induction), AWM no longer offers improvements in success rate (§3.1). We further hypothesize that the memory induction module has the most strict requirement, and explored adopting

*Equal Contribution

or transferring from the memory induced with GPT-4o model, which can effectively improve the overall success rate by 64–87%. This investigation offers a clearer understanding of pre-conditions for AWM to make effect — a strong enough memory induction module (§3.2).

After identifying the importance of the memory induction module, we then ask: *Can we optimize memory design to loosen the memory capability constraint?* (§4) Given that simpler workflows may be easier for weaker models to learn from, we first propose n-gram workflows, by extracting re-occurring n-step sub-trajectories from past successful experiences. To facilitate agent learning as its abilities improve, we design a curriculum learning procedure that produces and integrates increasingly complex (i.e., increasing n) workflows into the agent memory.

We find that this presumably optimized memory with increasing granular workflows does not bring substantial improvements in the overall task success rates, reinforcing the critical constraint on model capability in the memory induction module (§4.2). Nonetheless, n-gram workflows with our frequency filtering and memory cache mechanisms bring down the computation cost to a pronounced extent, by 58.2%, demonstrating the potential in building more efficient memory-learning agent pipelines (§4.3).

Overall, our work investigates the pre-conditions for memory-learning agents, and reveals the importance of having a quality-ensured memory-induction module.

2 Background: Memory-Learning Agents

2.1 Problem Statement

In this work, we focus on self-improving agents that learn and utilize memory online during inference time. More concretely, given an agent policy π with an LLM backbone L and instantiated with memory $M_o = \emptyset$, the online learning process passes in a stream of test queries $Q = (q_1, q_2, \cdots, q_N)$. For each given query, the agent policy generates an action trajectory $t_i = \pi(q_i|L, M)$ that contains multiple steps with the action taken and its associated thought process. The query and trajectory constitute an experience $e_i = (q_i, t_i)$ that, if predicted as correct by the experience evaluation module E, will be transformed into one or more workflows $W = I(e_i)$ and augmented to the agent memory $M_t = M_{t-1} + W$ for future task-solving.

Overall, this framework involves three main components (1) the task-solving agent π, (2) the experience evaluator E, and (3) the memory induction module I. By default, these three modules are all instantiated with the same language model, such as GPT-4o in the original AWM work (Wang et al., 2024). In the experiment sections, we will investigate if altering the model choice in all or some critical modules would affect the performance of this memory-learning agent pipeline.

2.2 Benchmark: WebArena

We focus on the web navigation task and adopt one of the most widely-used benchmarks — WebArena (Zhou et al., 2023) for experiments. WebArena consists of 812 realistic web navigation tasks across five websites across various domains — e-commerce (shopping), content management (cms), online social forums (reddit), collaborative software development (gitlab), and maps. Each task specifies a high-level intent in natural language, and the agent is expected to generate a sequence of executable actions to complete the task.

Following the AWM implementation (Wang et al., 2024), we adopt the BrowserGym framework (Drouin et al., 2024), its agent action space, and use the accessibility tree representation for webpage observations.

2.3 Evaluation Metrics

On WebArena, we evaluate agent performance in two dimensions: correctness and efficiency.

Success Rate (SR) SR measures the percentage of tasks that an agent can successfully solve on the entire WebArena benchmark. Using the evaluation script provided by the WebArena benchmark, we can run rigorous execution-based evaluations to check the functional correctness of the agent action

Table 1: Task success rate (SR) on WebArena using LLAMA 3.1 70B and DEEPSEEK-R1 32B. We adopt the GPT-4 success rates as reported from Wang et al. (2024)

Model	Method	Total SR	Shopping	CMS	Reddit	GitLab	Maps
GPT-4	baseline	15.0	17.2	14.8	20.2	19.0	25.5
	w/ AWM	35.5	30.8	29.1	50.9	31.8	43.3
LLAMA 3.1	baseline	**7.90**	**14.1**	2.2	0.0	**6.6**	**16.9**
	w/ AWM	6.17	10.5	**3.3**	0.0	4.6	12.5
DEEPSEEK-R1	baseline	**13.45**	**19.8**	**13.2**	4.5	**10.2**	**17.8**
	w/ AWM	11.55	13.5	10.9	**9.5**	8.2	16.9

trajectory for solving each task. In addition to the final SR after all tasks are completed, we also measure Accumulative Success Rate after each task i.e, SR for the tasks completed upto that point, to observe the correlation between change in memory and SR.

Efficiency and Cost In addition to solving tasks successfully, we also measure the computation cost for the memory learning and agent generation stages, to better gauge the efficiency of the agent pipeline. In particular, we count the *number of tokens in memory* (T) as a proxy metric, and report the average value throughout the inference process.

3 DOES MEMORY SELF-LEARNING HELP WEAKER MODELS?

In this section, we first reveal the limitations of AWM on weaker open models (§3.1) and then demonstrate the importance of the memory induction module via a memory-transferring test (§3.2).

3.1 APPLYING AWM TO WEAKER AGENTS

The original AWM work demonstrates significant improvements with the GPT-4o model, as shown on the top in Table 1. To study if this approach has certain model capability constraints, we additionally apply AWM to weaker, open-weight models. Concretely, we select LLAMA 3.1 70B and DEEPSEEK-R1 32B, two representative top-performing candidates, for experiments. For implementation, we use the `Llama-3.1-70B-Instruct` and `Deepseek-R1-Distill-Qwen-32B` model checkpoints hosted on the Huggingface serverless API. For this section, we use the same model to instantiate all three modules (task-solving π, experience evaluation E, and memory induction I) as introduced in §2.

Comparing the *baseline* and *w/ AWM* scenarios in Table 1, applying AWM does not bring substantial improvements to either LLAMA 3.1 and DEEPSEEK-R1 models. Furthermore, in contrast to the increased performance with GPT-4, applying AWM decreases the success rate compared to a baseline agent without any memory on most websites. While LLAMA and DEEPSEEK-R1 are fairly capable models, the drop in performance with memory highlights their inability to effectively induce and/or utilize memory in agentic applications.

Sub-optimal Workflow Quality To investigate the potential reason for the ineffectiveness of AWM on agents with weaker LMs, we conduct a small-scale manual analysis and identify two major problematic features in LLAMA-induced workflows — redundant actions and hallucinations.

First, many *redundant actions* inevitably present in workflow memory (Figure 1, left). The agent often produces some incorrect or non-functional actions at the beginning or intermediate steps of a task-solving process, resulting in mixed-quality action trajectories interwoven with useful and redundant actions. For weaker models like LLAMA, the proportion of non-useful steps could be higher and degrade the workflow quality. However, as it is challenging to precisely filter out all redundant actions, these noisy steps end up being incorporated into the agent's memory and bias the agent to similarly incorrect steps, leading to lower success rates overall.

Figure 1: Issues with workflows induced by weaker models. Left: *redundant* actions (in red) mingling with useful actions. Right: *hallucinated* actions not grounded in environment state.

Second, the workflow memory can include *hallucinations* propagated from the agent task-solving process. While a weaker agent often cannot correctly complete the task, it sometimes returns an answer to the user that is not grounded in the environment state. For instance, in Figure 1 (right), the agent returns a message saying that the driving time is 2 minutes, without observing such information presented in the environment. Due to the imperfect neural-based evaluation module of the weaker agent, such trajectories are sometimes predicted as correct and integrated into the memory. The inclusion of such hallucinations in memory further increases the likelihood of undesired hallucinations in similar scenarios.

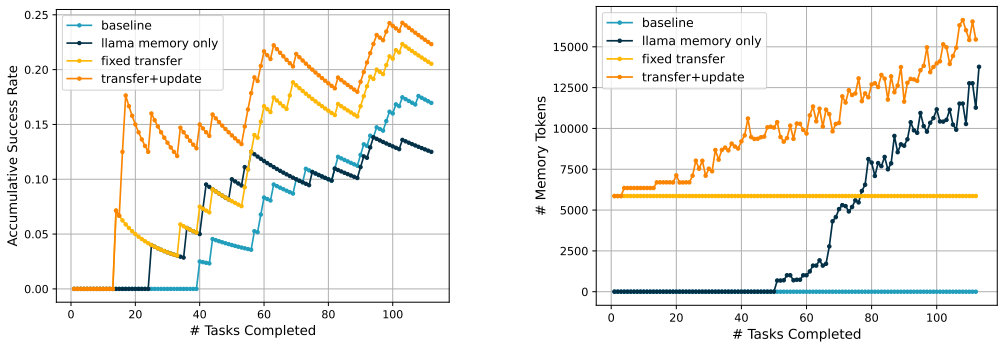

Figure 2: Success rate (left) and memory size (right) of LLAMA-agent when using AWM with GPT-transferred memory, in fixed and updated versions, using the `map` website as an example.

3.2 TRANSFERRING MEMORY FROM STRONGER GPT

Given the limitations of weaker models to produce high-quality workflows to effectively augment agent memory and realize capability bootstrapping, we explore transferring memory induced by a stronger agent, i.e., based on the GPT-4 model, to the weaker task-solving LLAMA agent.

More concretely, we experiment with memory transfer in two ways. First, *fixed transfer*, where we apply the fixed, full set of workflows induced by GPT for running inference on all test tasks. The second is *transfer + update*, where we initialize the memory of the LLAMA agent with GPT-induced workflows, while allowing LLAMA agent to continually induce and add new workflows to its memory through its own experiences during the inference time.

We present the success rate and memory sizes (measured in the number of tokens) of the LLAMA agent throughout the inference process in Figure 2.

First, both memory transfer mechanisms effectively improve over the baseline with no memory by 20.6–31.2% and over using LLAMA-induced memory by 64.0–78.4%.

Further comparing the two transfer approaches, we find that *transfer+update* achieves 8.8% higher in end success rate than the *fixed transfer* approach, suggesting that with proper memory instantiation, LLAMA can induce some useful workflows the further boosts the ability of task-solving agent. However, if examine the two memory-transfer methods based on their computation cost, *transfer+update* also incurs significantly more computation (i.e., more tokens to consume in memory) as it additionally adds $3\times$ more tokens to memory (Figure 2, right).

Overall, we find that an effective self-improving agent with AWM necessitates certain capabilities in the memory induction module, that can be fulfilled with GPT but not weaker LLAMA models.

4 CAN OPTIMIZING WORKFLOW MEMORY LOOSEN MODEL CONSTRAINTS?

While it may be challenging for weaker models to produce high-quality trajectory-level workflows from scratch, they may be capable of identifying useful sub-trajectories and inducing workflows accordingly, thus benefiting the task-solving agent in finer granularity.

In this section, we ask: *can we optimize the memory design to loosen the constraint on the memory induction module?* We first propose sub-trajectory workflows (§4.1) and the curriculum learning memory scheduling (§4.2), then study its effectiveness across WebArena websites in correctness and efficiency dimensions (§4.3).

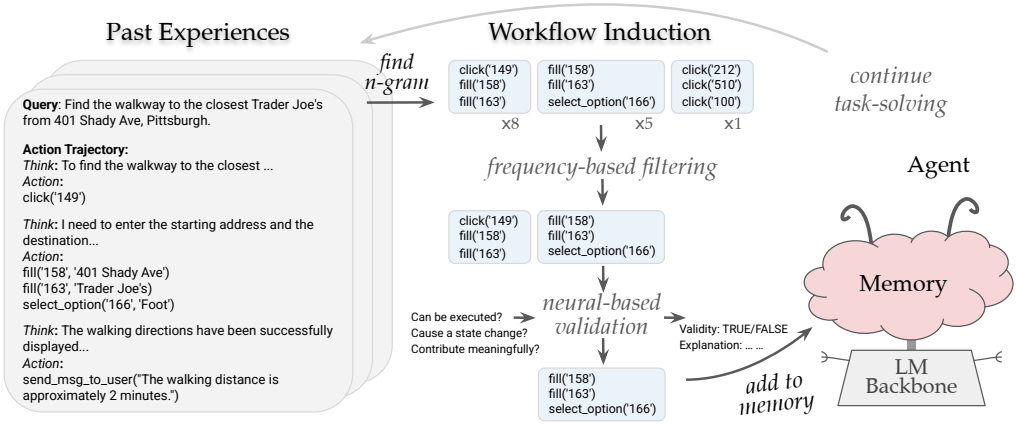

Figure 3: Pipeline of extracting n-gram workflows from past experiences (left), filtering workflows by frequency and validity (middle), and augmenting agent memory for continual task-solving (right).

4.1 INDUCING n-GRAM WORKFLOWS

N-grams as Sub-trajectories The workflow induction starts with a set of agent past experiences, each with a natural language instruction and a thought-action trajectory. Each task performed by the agent is represented by a set of alternating thought and corresponding actions pair as shown in Figure 3. Each action span consists of multiple steps or API calls that include the function, element and the argument involved (e.g., `fill('154','Pittsburgh Airport))`. For the desired workflow granularity n, we first extract all n-gram step sub-trajectories from the set of past trajectories, as shown in Figure 3 (left). More concretely, we remove the input arguments and only consider the element string (i.e., consider `fill('154')` instead of `fill('154','Pittsburgh Airport))` to allow certain flexibility in context changes among similar procedures. We consider two sub-trajectories to have a shared action sequence as long as they match on the action and element strings (i.e., `click('149')` $\rightarrow$ `fill('158')` $\rightarrow$ `fill('163')`). This gives us flexibility in varying n to capture increasing complexity of workflows.

As this rule-based matching process can be computed quickly without querying neural models, we can flexibly extract workflows with varying n specifications without adding substantial computation costs to the pipeline.

Filtering Workflows by Occurrence Threshold To leverage the maximally useful n-gram workflows, we filter out workflows that have appeared less than k times among the past experiences. As the agent continuously performs tasks, we also increase the threshold in accordance with the increasing number of past experiences, to avoid extracting irrelevant sub-trajectories which become more frequent relative to the initial threshold.[1]

This frequency-based filtering avoids inducting workflows which include task-specific non-generalizable procedures and also significantly reduces the number of sub-trajectories that are validated for quality using an LM, which is an expensive process. For instance, frequency thresholding filters 49 out of 97 2-gram sub-trajectories and 56 out of 276 3-gram sub-trajectories, reducing quality validation calls and retaining generalizable sub-trajectories.

Workflow Quality Validation From the filtered set of n-gram workflows $S = \{s\}$, we further validate their correctness and usefulness, as illustrated in Figure 3 (middle bottom). We use the same LM backbone as the task-solving agent (LLAMA in our experiments), and ask it to evaluate if each s (1) can be independently executed, that it does not rely on missing previous step(s) to be meaningfully completed; (2) causes a change in the state of the environment, to avoid n-grams with invalid steps that have no effect, and (3) contributes meaningfully and is not likely to cause hallucination.

More specifically, we prompt the LM with s (n-gram) and a randomly selected past example trajectory t_{ex} in which s is present. Find the exact prompt and an example one-shot in Appendix Figure 6. The model is expected to output: the validity evaluation (`True` or `False`), an explanation for the evaluation result, a high-level description d of the workflow in natural language d, and an example call of s directly extracted from the ex- ample trajectory t_{ex}. If s is valid, we induce the workflow w which consists of the description d and the concrete use-case example of s, and add the workflow w ($w : \{d, s\}$) to memory.

4.2 MEMORY CURRICULUM

As the agent capability improves, it presumably can consume and leverage more complex workflows, i.e., workflows that contain more steps. While only providing workflows of the same granularity may lead to a plateau in performance and capability improvements, we introduce a curriculum schedule with increasing workflow granularity to gradually increase memory complexity.

More specifically, we induce simpler workflows (e.g., 2-gram workflows) during the first K tasks of test inference. We update memory complexity by instead using $(2 + m)$-gram workflows after K tasks have been completed by the agent. We conduct this memory upgrade by iteratively increasing n by m after every K tasks, given the pre-determined values for hyper-parameters m and K. Empirically, we experiment with $m = 1$ and $K = 20$. Upon this curriculum with increasingly complex workflows, the memory moves from easy to hard, which potentially aligns better with the agent's overall capability in solving tasks.

As the complexity of the sub-trajectories increases, at each change in n, we also increase the occurrence threshold for induction to ensure the memory is reliable, as longer sequences of actions are also more unpredictable, following the same occurrence threshold schedule as discussed in §4.1.

Workflow Caching The neural-based workflow validation process can be expensive, especially if we validate all possible n-grams after solving each task. Relying on the fact that most n-grams remain the same before and after solving one task, we introduce a *workflow cache* that stores validated n-gram workflows to prevent repetitive validation, and make the induction process more efficient. The cache maps each workflow to a tuple that consists of its validity and resulting workflow ($s \rightarrow (\textit{validity}, w)$). If s is predicted as invalid, then the corresponding w is an empty string.

When the n-gram s is encountered the next time, instead of evaluating it for induction again using an LM, we add the inducted result w from the cache to the memory if *validity* is `TRUE`.

[1]This threshold is decided based on the website as each website has different repetitiveness in actions, as well as the number of tasks completed. We instantiate with a threshold (eg: 2 for `map`, 1 for `shopping`) at time zero and increment the threshold by 1 after completing 20 tasks.

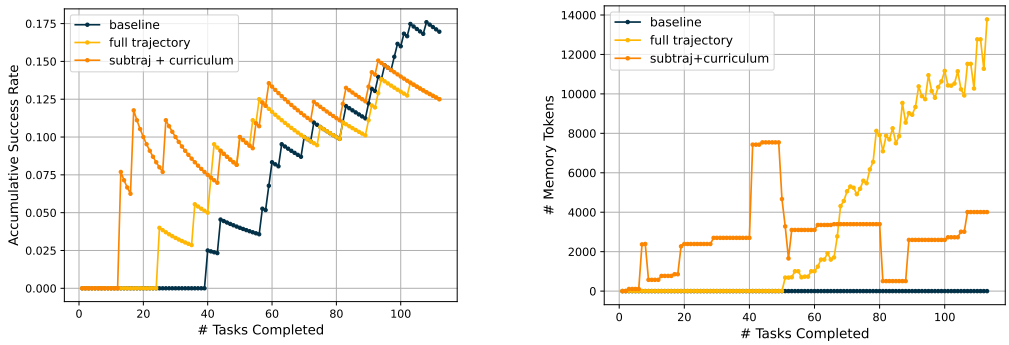

Figure 4: Success rate (left) and memory tokens (right) of LLAMA-agent with n-gram workflow memory curriculum scheduling on the Map benchmark.

In addition to making the induction process cheaper and more efficient, the cache also ensures stability and avoids the randomness of LM outputs during workflow validation. We show an example sub-trajectory memory file induced with the above process for Maps in Figure 7.

4.3 RESULTS AND ANALYSIS

We compare our n-gram workflow curriculum strategy with the baseline without memory and full-trajectory memory as in AWM, using the map website as an example. We experiment primarily with the LLAMA-agent in this section due to the larger performance gap between AWM and baseline on the map website, when compared to the DEEPSEEK-R1-agent. As shown in Figure 4, using n-gram workflows achieves similar end performance with trajectory-level workflows used in AWM, yet does not improve the overall success rate, suggesting the strict requirement on the quality of induced memory.

n-Gram Workflows Enables Easier Start Despite little difference in the end, in the first one-third of the inference process (i.e., 0–40 examples), the agent using our n-gram workflows consistently achieves higher success rates than the agent using full-trajectory workflows. We conjecture that, even the full trajectory can barely solve the task correctly, they often contain n-length sub-trajectories that can do some intermediate steps correctly and repetitively. Our n-gram strategy can extract these more fine-grained workflows and augment them in agent memory. As n increases and the workflows become more complex in the later inference process, the difference between our approach and full-trajectory is less prominent, therefore showing fewer differences.

Reduced Induction Cost In addition to the correctness dimension, we evaluate the total cost in workflow induction (i.e., neural-based validation) and compare it to the cost of inducing workflows using the AWM approach, by calculating the number of tokens of the inference process on the `shopping` website.

As shown in Table 2, n-gram sub-trajectory workflows only costs 41.8% of the compute required by full-trajectory workflow memories used by AWM. Although the prompts for inducing full-trajectory workflows have fewer tokens on average, they require $6.4\times$ more calls for induction, thus resulting in a higher number of total tokens consumed. For n-gram workflows, since validating and extracting from more examples uses a longer prompt, high overlap of n-grams along with thresholding and caching, the validation calls are much fewer resulting in a total of $2.4\times$ fewer tokens used for induction.

What is the impact of memory curriculum? We compare using constant sizes of n-gram workflows throughout inference with varying n from 2 to 4. In Figure 5, we see that curriculum scheduling achieves higher end success rate than using constant ns, as fixed-length n-gram workflows can limit the diversity and complexity of the workflows present in the memory. For instance, the model tends to perform better with 2-gram induction in the beginning, when having weaker capabilities;

Method	# Induction Calls	Avg. # Tokens Per Call	Total # Tokens
n-gram workflows	29	1451.2	42,079
full-traj workflows	186	541.3	100,684

Table 2: Comparing the computation cost of inducing trajectory-level workflows (in AWM) and our n-gram workflows.

while 4-gram workflows are harder to bootstrap from the start given the weaker model capabilities, leading to overall inferior performance. This further highlights the challenges with designing an effective memory representation for LLAMA-agent as fixed workflow sizes significantly underperform the baseline agent.

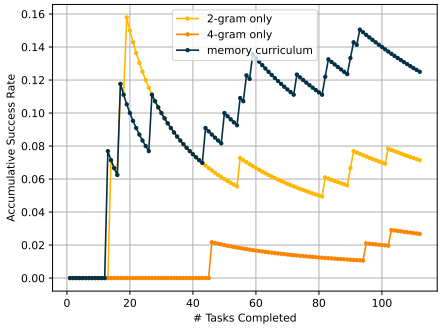

Figure 5: Comparing using n-gram workflows with increasing sizes and fixed sizes (2 and 4) for the LLAMA-agent

5 Related Work

Web navigation benchmarks and agents Among the wide range of tasks that have been proposed to evaluate digital agents Yao et al. (2024); Kapoor et al. (2025); Xie et al. (2024), browsing and navigating through webpages has acted as a popular playground for evaluating LLM-based agents in versatile scenarios such as shopping (Yao et al., 2022a), social media communication (Zhou et al., 2023; Koh et al., 2024a), knowledge work tasks (Drouin et al., 2024), and more (Deng et al., 2024). We hence focus on the web navigation task using the WebArena (Zhou et al., 2023) benchmark.

Agent Learning from Experiences An important thread of work to build autonomously self-improving agents is to learn from experiences. In addition to collecting human (Deng et al., 2024) or synthesized (Ou et al., 2024; Xu et al., 2025) demonstrations, some works gather past experiences through agent-initiated exploration in instruction-driven (Murty et al., 2024b) or trajectory-driven (Murty et al., 2024a) approaches, which could provide as warm starts on the websites of interest. On the other hand, some work aggregate experiences (Wang et al., 2024) or feedback (Qu et al., 2024) during test time on the fly.

Agents Improving via Adaptive Memory Agents aggregate information from various information sources above to augment their memory for self-improvement. Many works propose to learn information in parametric representations via supervised fine-tuning (Murty et al., 2024a), contrastive learning (Song et al., 2024) to prevent recurrent failures, or reinforcement learning (Zhou et al., 2024). On the other hand, non-parametric approaches that directly augment self-reflections (Shinn et al., 2023) or retrieved past experiences (Wang et al., 2023; Zheng et al., 2023) in context. Taking a step forward than raw experiences, Wang et al. (2024) proposes to induce more reusable workflows for memory augmentation. While these methods have been shown effective with strong, closed-source models, it is unclear if they can generally apply to weaker models, or what specific pre-conditions they necessitate to bring benefits. Our work first reveals the memory capability constraint and explores memory optimization strategies to loosen those requirements.

6 CONCLUSION

In this paper, we study the pre-conditions of memory-learning agents on model capabilities. While an effective induction module necessitates strong models, weaker models can still self-improve by transferring memory from stronger models. In addition, we design a learning curriculum for memory across sub-trajectories of different granularities that significantly reduces the cost of memory while marginally improving performance. We hope that this study provides insight into the downfalls of memory-learning agents with weaker models and facilitates better memory designs.

REFERENCES

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178. Springer, 2025.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024a.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024b.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D. Manning. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator, 2024a. URL https://arxiv.org/abs/2410.02907.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. Bagel: Bootstrapping agents by guiding exploration with language, 2024b. URL https://arxiv.org/abs/2403.08140.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale, 2024. URL https://arxiv.org/abs/2409.15637.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*, 2024.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve, 2024. URL https://arxiv.org/abs/2407.18219.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

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*, 2022b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2023.

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

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. *arXiv preprint arXiv:2412.13194*, 2024.

A EXPERIMENT DETAILS

We here provide the specific prompt we use for sub-trajectory evaluation (Figure 6) and exemplar workflows induced by our method (Figure 7).

```
Given the below trajectories for a web navigating agent
including its thoughts and actions - each action is composed
of a certain number of steps.

We provide a sub-trajectory where each step is denoted
by its corresponding elements/button which are present
in these trajectories, decide if this is a valid sub-trajectory
given the complete trajectory.

Output if it is a valid sub-trajectory or not and the suitable
explanation. If it is a valid sub-trajectory,
create a suitable 1-line natural language query for the
sub-task which is solved by the sub-trajectory
- this should be a generalized description of the
sub-trajectory and can be adapted from the main query.

Also extract all the think and steps of the corresponding
sub-trajectory.If the subtrajectory spans multiple thought
and action pairs, copy all of them. If it spans only a subset
of steps in a single action, modify the thought of
that action to only be based on the steps in the sub-trajectory.
Output all of the steps present in the subtrajectory given
below, but do not include any extra steps.

Conditions of a valid sub-trajectory:
1. It contributes meaningfully to the
trajectory to attain the final goal
2. It is a valid and complete set of actions in itself,
is not a continuation or relies on the previous step to
be meaningfully completed
3. There is a change in state of the web environment,
i.e. the action ends up displaying a new page,
sending some input etc. If there is a change,
it will be reflected in the model's next think step
as it observes the updated change in the web environment.
```

Figure 6: Prompt for sub-trajectory evaluation.

```
## Subtrajectory Examples


Workflow: To find 'X' near 'Y' location
by clearing previous inputs and searching again

Example:
<think>
To find restaurants around CMU ArtPark Lab, I need
to perform a search for "restaurants" in the vicinity. I'll
enter "restaurants" in the search box and execute the search.
</think>
<action>
clear('145')
fill('145', 'restaurants near CMU ArtPark Lab')
click('147')
</action>



Workflow: To find driving directions from location 'X' to
location 'Y' using the directions feature

Example:
<think>
To find the estimated driving time from Homewood Suites
Southpointe to PPG Paints Arena, I need to enter "Homewood
Suites Southpointe" in the "From" field and "PPG Paints Arena"
in the "To" field. Then, I'll calculate the route using the
"Car (OSRM)" option.
</think>
<action>
fill('158', 'Homewood Suites Southpointe')
fill('163', 'PPG Paints Arena')
click('171')
</action>



Workflow: To enter two locations and select a
walking route for calculating directions

Example:
<think>
To find the walking time from the University of Pittsburgh
to Starbucks on Craig Street, I need to
enter these locations in the "From" and "To" fields and
select the walking  option. I'll start by
filling in the "From" and "To" fields.
</think>
<action>
fill('158', 'University of Pittsburgh')
fill('163', 'Starbucks, Craig Street')
select_option('166', 'Foot (OSRM)')
</action>
</action>
```

Figure 7: Example sub-trajectory memory file for maps website (3-grams).