Try-Then-Eval: Equipping an LLM-based Agent with a Two-Phase Mechanism to Solve Computer Tasks

Duy Cao[1,3,4,†], Phu Nguyen[1,3,†], Vy Le[2,3,4,†], Long Nguyen[1,3] and Vu Nguyen[1,3,4,*]

***Abstract*— Building an autonomous intelligent agent capable of carrying out web automation tasks from descriptions in natural language offers a wide range of applications, including software testing, virtual assistants, and task automation in general. However, recent studies addressing this problem often require manually constructing of prior human demonstrations. In this paper, we approach the problem by leveraging the idea of reinforcement learning (RL) with the two-phase mechanism to form an agent using LLMs for automating computer tasks without relying on human demonstrations. We evaluate our LLM-based agent using the MiniWob++ dataset of web-based application tasks, showing that our approach achieves 85% success rate without prior demonstrations. The results also demonstrate the agent's capability of self-improvement through training.**

I. Introduction

Large language models (LLMs) have the potential in automating problem-solving systems due to their natural language comprehension and in-context learning capabilities [1], [2], [3]. Such models can handle cognitive tasks, such as devising plans and generating code, which traditionally required specialized models or certain degrees of human intervention. Their ability to identify and potentially rectify errors through appropriate prompt construction also enhances the robustness and adaptability of autonomous systems [4], [5]. Recent applications utilizing language models as principal components are being rigorously researched, including game agents [6], web agents [7], embodied agents [8], etc.

The development of intelligent agents capable of solving computer tasks using LLMs has also gathered substantial momentum in the research communities [9]. Computer tasks represented by natural language, from clicking a button to filling a complex form (Figure 1), could be automated to improve productivity or to simulate human behaviors for testing purposes. Approaches to this topic can be classified into three categories [10]: reinforcement learning, fine-tuning-based methods, and prompt-based methods. Precedent state-of-the-art prompt-based approaches often require a handful of human demonstrations (e.g., Synapse [11], AdaPlanner [12], RCI [13]).

This paper proposes an LLM-based agent to automatically perform computer tasks given task descriptions without relying on human demonstrations. Our method uses prompt-based mechanisms and adopts the idea from verbal reinforcement [4], prioritizing in-context learning via feedback loop, logic flow, and memory mechanism.

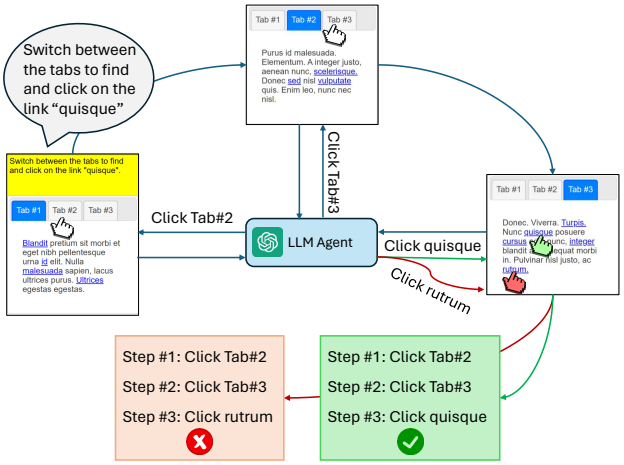


Fig. 1: Example of computer task performed by the LLMs agent. At each step, the LLMs agent receives information to determine the next action, executes it, and updates the state. This cycle repeats until the task succeeds or fails.

Our method consists of two phases, training and evaluation. The training phase concentrates on exploring the environment, observing patterns, and extracting rules that are stored in the long-term memory storage. The evaluation phase uses the memory previously collected to assist the agent in solving computer tasks.

We evaluate our LLM-based agent using the popular MiniWob++ dataset [14] of 43 web-based application tasks, showing that our approach achieves an 85% success rate without prior demonstrations.

This study provides the following main contributions:

- The use of a verbal reinforcement strategy that incorporates the two-phased mechanism and textual rules to eliminate the need for human demonstrations.
- Employing a token-efficient screen representation to minimize redundant information.
- Proposing an agent that has the capability of self-improvement during training by learning success patterns and rules to devise more accurate actions to take when performing tasks.

[1]Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam
[2]Faculty of Computer Science, University of Information Technology, Ho Chi Minh City, Vietnam
[3]Vietnam National University, Ho Chi Minh City, Vietnam
[4]Katalon Inc.
[†] These authors contributed equally to this research
[*]Corresponding author: nvu@fit.hcmus.edu.vn

II. Background and Related Work

A. Large Language Models

LLMs are deep-learning-based models that are pre-trained on extensive amounts of data. LLMs are intended to have an ample capacity for understanding and generating natural language. With the ability of knowledge-intensive tasks, LLMs were used to solve computer tasks in recent studies. Synapse [11] supplies LLMs with successful trajectories and a current trajectory to make decisions for the current action. This iterative process is called Trajectory-as-Exemplar (TaE) prompting. AdaPlanner [12] uses LLMs in planning and refining its plan by in-plan feedback and out-plan feedback. Recursive Criticism and Improvement (RCI) [13] starts by letting the LLMs generate an initial plan, then has it find the problem with the generated output, and then generates a new and better one. Synapse, AdaPlanner, and RCI require hand-crafted feedback or demonstrations, limiting their ability to handle new or unfamiliar tasks. Li *et al.*[15] eliminate the need of demonstrations by applying *self-reflection* to identify and learn from mistakes from multiple trials. However, this refinement strategy is applied for each action without storage.

B. Reinforcement Learning with Large Language Models

With the advent of LLMs, approaches that leverage common-sense knowledge and effective reasoning abilities of LLMs within the RL environment have been discussed and analyzed [16].

Using LLMs as a reward function. With the common-sense knowledge from massive pre-training data, LLMs are incorporated into the reward design. Xie *et al.* [17] proposed a data-free framework to automatically generate and shape dense reward functions with the leverage of LLMs. LESR [18] uses LLMs to create task-related state representations accompanied by intrinsic reward functions. Kwon *et al.* [19] introduces the approach of using LLMs as a proxy reward function.

Using LLMs as a policy. LLMs are shown to be capable of acting as RL agents with self-critique ability, which enables them to discover and correct the errors in previous outputs. Reflexion [4] has developed a verbal RL approach to strengthen language agents, focusing on linguistic feedback rather than weight updates. ReAct [20] combines reasoning and acting capabilities of LLMs to interact with the environment.

III. Method

Figure 2 illustrates the overview of our autonomous agent. Given a specific web environment and a task goal in natural language, the agent aims to generate a sequence of actions to perform the task. The following sections describe further details of the proposed method.

A. Shortened DOM Representation

Document Object Model (DOM) presents a document underlying a web page. It contains information that may not help determining an action to take. In our approach, we provide the agent with only interactable DOM elements that are visible on the screen (Step 1). More specifically, we extract DOM elements that can be interacted with through user actions and their relevant surrounding context to be stored as JSON objects. Figure 3 shows an example of a task, the current HTML page, and its corresponding JSON objects. These objects are further attached with their corresponding operations including clicks and inputs to form a shortened version of the current DOM. This shortened DOM state is then used as input for *GUI Clickable Describer* (Figure 2) to generate a representation of the shorten DOM in natural language.

B. Iterative Planning

Upon obtaining the refined representation, our agent performs an iterative planning process (Step 2 to Step 5). This process leverages the state representation alongside task requirements to continuously generate actions within a closed-loop framework until the task is accomplished.

At iteration i, *Action Generator* π chooses an action $a_i \in A_i$ where A_i is the list of possible actions on the current DOM state extracted using the *GUI Clickable Describer*.

$$a_i = \pi(A_i, SM_i, S_i, R_i) \tag{1}$$

where $SM_i = ((a_{i-1}, a_{i-2}, ...a_0), T)$ is the previously executed actions (short-term memory) up to iteration i along with the current task T, S_i is the current set of successful trials, R_i is the current set of rules. S_i and R_i will be further described in the next section.

C. Experience Reinforcement

To facilitate the improvement of our agent through iterative learning, we incorporate a long-term memory module "*Experience Knowledge*", which archives historical trials and a set of rules (Step 6 to Step 8). This long-term memory operates within a two-phased mechanism: the training phase and the evaluation phase.

1) Training phase: During the this phase, the memory module updates automatically when the iterative planning completes its episode. *Memory Retriever* is responsible for retrieving essential information to be used as a prompt for *Action Generator* in the subsequent iteration.

Algorithm 1 discusses in detail the above idea. The triplet (S, F, R) constitutes the system's long-term memory. Generally, if the agent's episode is successful, we append it to the set of few-shot examples for the agent's future usage as LLMs are capable of following patterns with the in-context learning capability (lines 1–3). The failed attempts, while not being used explicitly so as to prevent LLMs from being incentivized to follow wrong patterns, are used to extract textual rules expressed in natural language (e.g., *avoid clicking on unrelated elements*) to assist the agent in its subsequent episodes (lines 4–7).

During the training phase, S_i and R_i are dynamically updated after the agent finishes a task. Additionally, we keep track of the average of the correct episodes over a specified number of recent episodes during the training phase (called *moving average*), which is used to determine the optimal

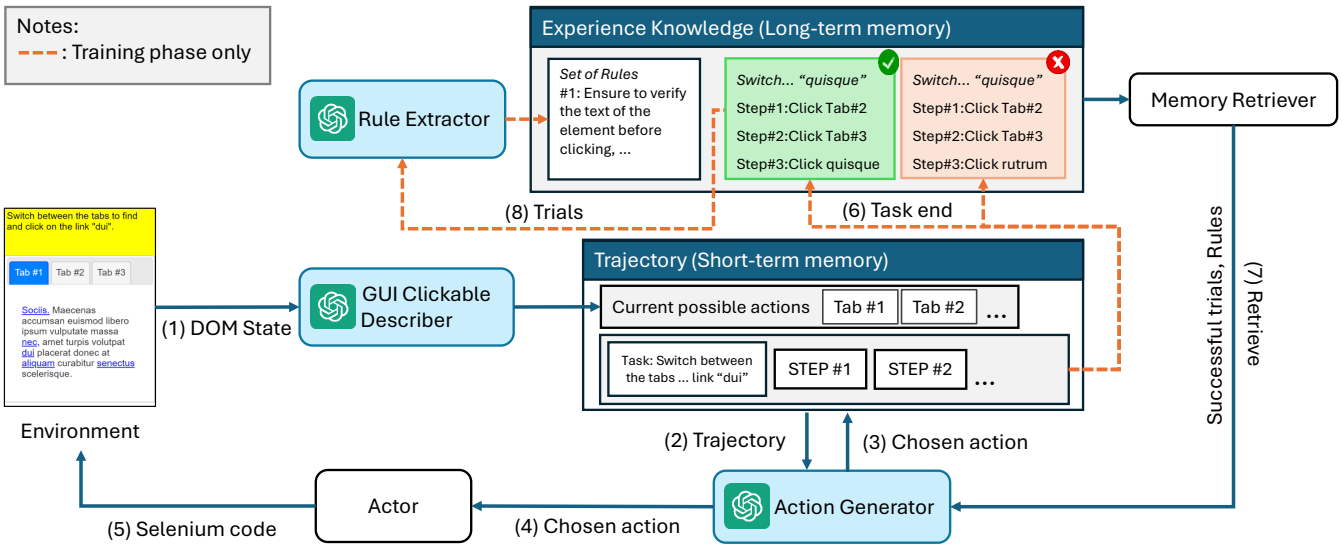
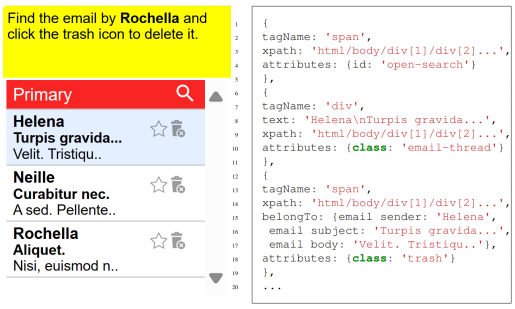


Fig. 2: Overview of our proposed method.



(a) (b)

Fig. 3: Task goal, HTML page (a), and JSON objects representing a shortened version of the page's DOM (b)

combination of successful trials and rules for the agent in the evaluation phase.

2) Evaluation phase: In the evaluation phase, the memory module remains fixed, and we solely rely on *Memory Retriever* to retrieve the most optimal successful trials and rules, which exhibit an upward trend and stable accuracy from the training phase, decided by the *moving average*. Currently, the optimal sets are manually selected based on human observation, without building demonstrations from scratch in comparison with other prompt-based approaches. The iterative planning algorithm remains the same but uses a fixed optimal set of successful trials and rules, which serve as part of the prompt for *Action Generator* π.

$$a_i = \pi(A_t, SM_t, S_{optimal}, R_{optimal}) \tag{2}$$

3) Rule extraction: As mentioned in III-C.1, we utilize *Rule Extractor* equipped with LLMs to deduce a set of textual rules by feeding them with a set of successful and failed trials. The intuition behind this approach is to apply a higher level of abstraction to LLMs, which has been shown effective in [21]. This behavior is also natural, as humans tend to learn from reasons why they fail in addition to failures.

Algorithm 1 Experience Reinforcement Algorithm

Input: short term memory SM of the current episode, success status $isSuccess$, current set of success trials S_{i-1}, current set of failed trials F_{i-1}, current set of rules R_{i-1}, rule extraction agent γ

func ExperienceReinforce

if $isSuccess(SM)$ **then**
  $S_i = S_{i-1} \cup (SM)$
  $F_i = F_{i-1}$
else
  $S_i = S_{i-1}$
  $F_i = F_{i-1} \cup (SM)$
  $R_i = R_{i-1} \cup \gamma(S_i, F_i, R_{i-1})$
end if
return S, F, R

IV. Experiment

A. MiniWoB++ dataset

We conducted the experiment on the MiniWoB++ dataset [14], which contains a collection of over 100 web interaction environments. Each environment is specifically defined for a task, consisting of a description (e.g., *Choose an item from a drop-down list*), a set of utterances (e.g., *Bobine, Betty*) that will be randomly chosen for every episode to form a complete natural language task (e.g., *Select Betty from the list and click Submit*). Figure 1 demonstrates an example in the MiniWoB++ dataset.

Due to the equivalent comparison to other approaches, we only selected a subset of tasks (43 tasks) and classified them into two categories: 1) 1-State tasks: 29 tasks that perform actions without changing the state, typically easy tasks; 2)

Parameters	Training	Evaluation
#episode	[20, 150]	25
#success_trials	8	8
#rules	3	3
#trials_extract_rule	[2, 6]	N/A

TABLE I: Training & Evaluation phase hyper-parameters.

N-State tasks: 14 tasks that perform actions that can change states (e.g., expanding drop-down, opening a hidden tab).

The reason for classifying these tasks is to assess the exploration capability of LLMs agents. The former category is simple and deterministic, while the latter requires an adaptive plan when the state changes after a certain action.

B. Evaluation

We use the *success rate* (SR) [22] as the evaluation metric. It is used to evaluate the method's performance by other methods [7], [12], [13], [15], [23]. This metric is calculated as the proportion of successful episodes over the total executed episodes. An episode is considered successful if it receives a positive reward, and failed if it either receives a negative reward or exceeds a specified step limit.

C. Setup

The detailed setup for each training and evaluation phase can be found in Table I. We first train the agent to run between 20 and 150 episodes for a task. Notably, the number of episodes varies during the training phase due to a stopping criterion. Then, we run 25 episodes in the evaluation phase to align with Li *et al.*'s results. The number of textual rules that are being used is 3, which was deduced from 2 to 6 successful/failed trials during the training phase. Moreover, the trials needed to extract rules are not available in the evaluation phase as we do not extract rules in this phase.

D. Model comparison

We use the InstructGPT-3 + RLHF (*gpt3.5-turbo-1106*) model via the OpenAI API for all agent operations and conduct experiments on a shared set of tasks. The Li *et al.* approach [15] employs FLAN-PaLM2 L [24], which is a proprietary LLM, making direct comparison difficult. For the supervised learning and RL approaches, we selected WebN-T5 [25] and CC-Net [23], as they represent the best methods within their categories.

E. Results

The performance of our method for each category of tasks, compared with other baseline methods is reported in Table II. Incorporating a training phase before the evaluation phase boosts the average success rate across all tasks by 10%, with the most significant improvement observed in N-State tasks rather than 1-State tasks. Moreover, we outperform WebN-T5-3B by 28%, a model that fine-tunes the encoder-decoder architecture on 12K human-annotated demonstrations. Our method performs lower than CC-Net (RL+SL) by 11%. However, it is important to note that CC-Net benefits from 2.4M expert demonstrations sourced from 77 human participants for supervised learning. Additionally, our performance is 10% lower than that of RCI and AdaPlanner. These prompt-based approaches leverage prior hand-crafted demonstrations, including step-by-step plans, in either plain text or code format as few-shot examples for a task. Although our approach yields lower results, it eliminates the reliance on expert demonstrations. For prompt-based approaches without demonstrations, Li *et al.* achieves a remarkable 94% success rate without exemplars, employing the FLAN-PaLM2 L model and structured reflection. A significant performance gap of 13% is observed in 1-State tasks, which will be further discussed in the next section.

Method	Category	Avg. Success Rate		
		1-State	N-State	Total
WebN-T5	SL	0.74	0.29	0.57
CC-Net	RL + SL	0.96	0.97	0.96
RCI	Prompt-based	0.97	0.89	0.95
AdaPlanner	Prompt-based	0.97	0.90	0.95
Li *et al.*	Prompt-based	0.98	0.86	0.94
Ours (w/o training)	Prompt-based	0.81	0.59	0.74
Ours (w/ training)	Prompt-based	0.85	0.87	0.85

TABLE II: Performance comparison with other methods. The Category column shows the methods classified into Supervised Learning, Reinforcement Learning, or Prompt-based with LLMs.

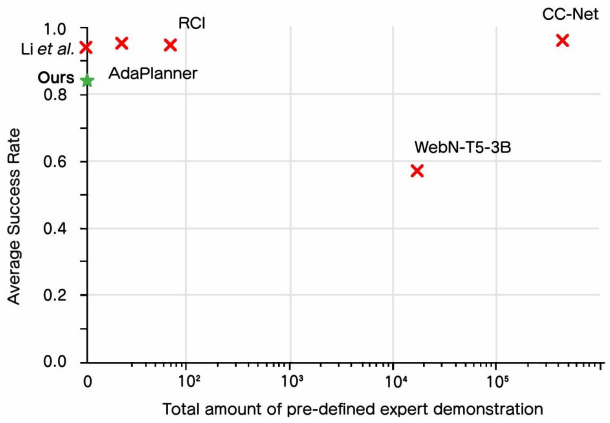


Fig. 4: Relationship between the success rate and amount of total expert training data.

1) Performance on 1-State tasks: Figure 5 illustrates the performance of our method on 1-State tasks. Overall, our agent achieves a 1.00 success rate in 76% of tasks, even without a training phase, where the agent plans without any history of successful trials and rules. However, ambiguity exists in two tasks, namely *click-widget* and *click-dialog-2*. For instance, in the task "Click the button in the dialog box labeled 'x'" of *click-dialog-2*, the button 'x' is represented in the HTML code as <button ... title="Close">. Similarly, the requirement to click on the "text" and "text area" widgets in the task *click-widget* is also ambiguous for the agent. These issues are resolved by incorporating successful trials and rules, resulting in a 1.00 success rate for our agent. The 76% of tasks achieve a 1.00 success rate also indicates why

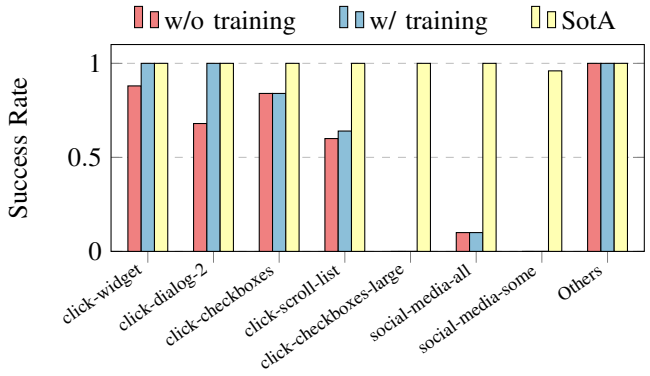


Fig. 5: Performance on 1-State tasks. SotA is state-of-the-art in prompt-based methods.

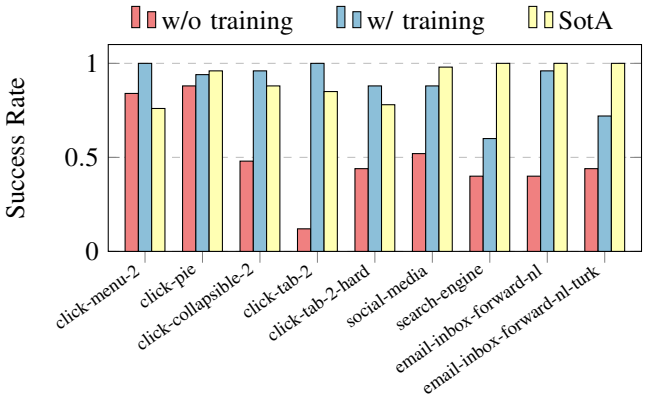
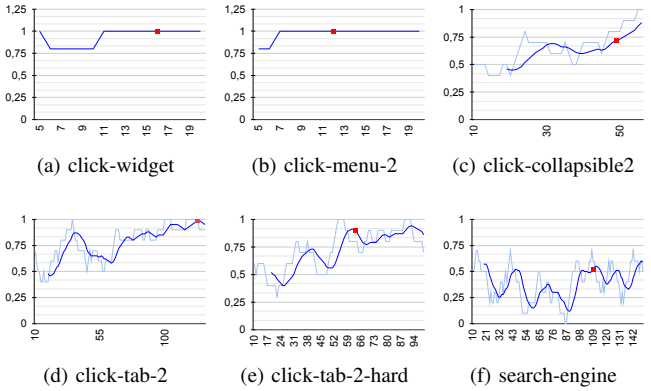


Fig. 6: Performance on N-State tasks. SotA is state-of-the-art in prompt-based methods.

the gap between performance with and without training is not significant in the 1-State task. Additionally, we observe two main failures of our agent, which contribute to the significance gap in success rate compared to the prompt-based state of the art: (i) handling long trajectories (e.g., *click-scroll-list*, *click-checkboxes-large*) that require multiple steps, and (ii) mathematical reasoning (e.g., *social-media-all*, *social-media-some*) where the agent has to count and keep track of the number of items on the web state.

2) Performance on N-State tasks: Figure 6 shows how our method performs on more challenging tasks that require our agent to adapt its plan to state changes. In the first two tasks, our agent performs well in both settings, with performance improvement observed in the *click-menu-2* and *click-pie* due to experience knowledge. However, our agent struggles significantly in the last six tasks, which involve multiple states (e.g., multiple sections/tabs/screens) within the tasks. After gaining experience knowledge from the training phase, the success rate increases considerably due to successful patterns and rules, minimizing negative failures. Exceptionally, *search-engine* requires a high level of reasoning to search for ordinal elements (1st, 2nd, etc.) scattered across multiple pages. This task demands the agent to understand both the mathematical positions and how to explore through pages. Consequently, our agent's capability is unable to solve this task in all cases, resulting medium score compared with Prompt-based SotA.

3) Training phase analysis: Figure 7 demonstrates the moving average of the correct episodes over a specified number of 25 episodes during the training phase. The x-axis shows the number of episodes that were run. When tasks perform well even without prior training, the training phase is observed to have a good initial start, reaching a stable maximum reward with the minimal number of episodes. However, with more challenging tasks, the agent either improves through the knowledge it obtains during the training phase or fluctuates otherwise. It also proves that the significant gap in the evaluation phase is due to the improvement over the training phase of our agent (from 7c to 7e), otherwise, the agent is either good enough from the start (7a and 7b), or can not obtain good experience knowledge during training (7f).



(a) click-widget (b) click-menu-2 (c) click-collapsible2

(d) click-tab-2 (e) click-tab-2-hard (f) search-engine

Fig. 7: Moving average of 6 tasks during the training phase. The red rectangles mark the position where successful trials $S_{optimal}$ and rules $R_{optimal}$ are manually chosen.

4) Ablation study: To further investigate the contribution of the experience combined of success trials and rules, we conduct an ablation study where we intentionally omit either success trials or rules in the evaluation phase. The tasks involved in this ablation study are those whose results in low performance when running pure iterative planning without training, which includes all of the N-State tasks and some of the 1-State tasks. Tasks can be handled with ease even without any examples or rules; such tasks cannot be used to demonstrate the influence of examples and rules, and thus excluded in the ablation study.

Figure 8 illustrates the performance of our agent for the

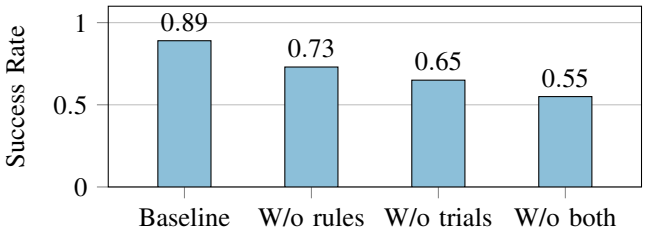


Fig. 8: Ablation study on success trials and rules.

ablation study. It can be observed that our agent is indeed better at following patterns presented in examples than following rules in natural language, and that the combination of both offers a noticeable performance boost of 34% success rate in our agent's performance compared to applying neither.

V. Limitation

Getting LLMs to understand DOM representations, with all of the nuanced features of the scripting language and the underneath JavaScript, is challenging. Our current approach, while prioritizing token efficiency by using interactable elements, assumes that all needed contextual content is in a close proximity relative to these elements. Our approach also relies on descriptions of user interface elements. Our iterative planning process does not handle live events like pop-up modals or loading screens, assuming instead that website changes occur solely from our actions and update instantly. This contrasts with real-world websites, which often involve client-server communication, background tasks, and animations. Finally, a more robust governance over the long-term memory should be applied. Further research is needed in determining good and bad examples/rules for LLMs as well as a mathematical formation to select the most promising set of examples/rules. Moreover, there is also a reasoning limitation within LLMs.

VI. Conclusions

In this paper, we have proposed a two-phase mechanism utilizing the idea of self-reflection to design an autonomous agent that solves computer tasks without prior expert demonstrations. Our experiments on the MiniWoB++ dataset demonstrated that our method exhibits an 85% success rate without prior human-annotated data compared to supervised learning, reinforcement learning, and other prompt-based approaches. Furthermore, we showed that the LLM-based agent can potentially self-improve from the training phase by exploring the state and obtaining useful information. Ablation study on both rules and success trials also showed the effectiveness of each component during the evaluation phase. However, there are still certain limitations of the agent in planning and reasoning which require further work to be done in future research.

VII. Acknowledgements

This research is partially supported by research funding from the Faculty of Information Technology, University of Science, VNU-HCM, Vietnam.

References

[1] P. Ethape, R. Kane, G. Gadekar, and S. Chimane, "Smart automation using llm," *International Research Journal of Innovations in Engineering and Technology*, vol. 7, no. 11, p. 603, 2023.

[2] T. Sumers, S. Yao, K. Narasimhan, and T. Griffiths, "Cognitive architectures for language agents," *Transactions on Machine Learning Research*, 2023.

[3] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, pp. 1–26, 2024.

[4] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[5] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[6] Y. Wu, X. Tang, T. Mitchell, and Y. Li, "Smartplay: A benchmark for llms as intelligent agents," in *The Twelfth International Conference on Learning Representations*, 2023.

[7] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2web: Towards a generalist agent for the web," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2998–3009.

[9] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang *et al.*, "Agentbench: Evaluating llms as agents," in *The Twelfth International Conference on Learning Representations*, 2023.

[10] I. Gur, H. Furuta, A. V. Huang, M. Safdari, Y. Matsuo, D. Eck, and A. Faust, "A real-world webagent with planning, long context understanding, and program synthesis," in *The Twelfth International Conference on Learning Representations*, 2023.

[11] L. Zheng, R. Wang, X. Wang, and B. An, "Synapse: Trajectory-as-exemplar prompting with memory for computer control," in *The Twelfth International Conference on Learning Representations*, 2023.

[12] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang, "Adaplanner: Adaptive planning from feedback with language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[13] G. Kim, P. Baldi, and S. McAleer, "Language models can solve computer tasks," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[14] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," in *International Conference on Learning Representations (ICLR)*, 2018.

[15] T. Li, G. Li, Z. Deng, B. Wang, and Y. Li, "A zero-shot language agent for computer control with structured reflection," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[16] G. Franceschelli and M. Musolesi, "Reinforcement learning for generative ai: State of the art, opportunities and open research challenges," *Journal of Artificial Intelligence Research*, vol. 79, pp. 417–446, 2024.

[17] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2reward: Reward shaping with language models for reinforcement learning."

[18] B. Wang, Y. Qu, Y. Jiang, J. Shao, C. Liu, W. Yang, and X. Ji, "LLM-empowered state representation for reinforcement learning," in *Forty-first International Conference on Machine Learning*, 2024.

[19] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," in *The Eleventh International Conference on Learning Representations*, 2023.

[20] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[21] H. S. Zheng, S. Mishra, X. Chen, H.-T. Cheng, E. H. Chi, Q. V. Le, and D. Zhou, "Take a step back: Evoking reasoning via abstraction in large language models," in *The Twelfth International Conference on Learning Representations*, 2024.

[22] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang, "World of bits: An open-domain platform for web-based agents," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3135–3144.

[23] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhaparia, A. Muldal, J. Abramson, P. Georgiev, A. Santoro, and T. Lillicrap, "A data-driven approach for learning to control computers," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9466–9482.

[24] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[25] I. Gür, O. Nachum, Y. Miao, M. Safdari, A. Huang, A. Chowdhery, S. Narang, N. Fiedel, and A. Faust, "Understanding html with large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 2803–2821.