DORA: Dynamic Optimization Prompt for Continuous Reflection of LLM-based Agent

Kun Li[1,2], **Tingzhang Zhao**[1,2], **Wei Zhou**[1*], **Songlin Hu**[1]
[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, University of Chinese Academy of Sciences
{likun2, zhaotingzhang, zhouwei, husonglin}@iie.ac.cn

Abstract

Autonomous agents powered by large language models (LLMs) hold significant potential across various domains. The Reflection framework is designed to help agents learn from past mistakes in complex tasks. While previous research has shown that reflection can enhance performance, our investigation reveals a key limitation: meaningful self-reflection primarily occurs at the beginning of iterations, with subsequent attempts failing to produce further improvements. We term this phenomenon "Early Stop Reflection," where the reflection process halts prematurely, limiting the agent's ability to engage in continuous learning. To address this, we propose the DORA method (Dynamic and Optimized Reflection Advice), which generates task-adaptive and diverse reflection advice. DORA introduces an external open-source small language model (SLM) that dynamically generates prompts for the reflection LLM. The SLM uses feedback from the agent and optimizes the prompt generation process through a non-gradient Bayesian Optimization (BO) algorithm, ensuring the reflection process evolves and adapts over time. Our experiments in the MiniWoB++ and Alfworld environments confirm that DORA effectively mitigates the "Early Stop Reflection" issue, enabling agents to maintain iterative improvements and boost performance in long-term, complex tasks[1].

1 Introduction

Recent studies have highlighted the use of autonomous agents powered by LLMs to make decisions in specific tasks. However, these agents often rely on large, frozen models, making it difficult to adapt to dynamic or changing environments. To address this challenge, the self-improvement framework called "Reflection" has been proposed.

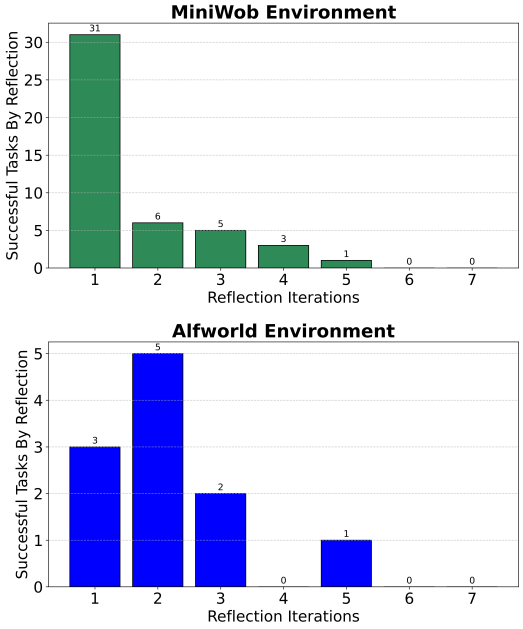

Figure 1: The performance of the agent with reflection in miniwob++ and alfworld environments shows that successful reflections are primarily concentrated in the early stages of the process.

This framework incorporates historical trajectories into an additional LLM, which generates reflection advice to help the agent continuously learn from previous mistakes.

While numerous studies have demonstrated that reflection can enhance performance (Shinn et al., 2023; Huang et al., 2022; Li et al., 2023; Yao et al., 2023b; Chen et al., 2023b; Xi et al., 2024), our deeper investigation find an undesirable pattern in reflection framework: **effective self-reflection occurs primarily at the start of iterations**. As shown in Figure 1, this pattern suggests that initial reflections are impactful, but subsequent iterations fail to drive further improvements.We refer to this issue as "Early Stop Reflection," where the reflection process halts prematurely, limiting the agents' ability to continuously learn from their mistakes,

[*] Wei Zhou is the corresponding author.
[1]Code are available at https://github.com/linkseed18612254945/FineRob

Proceedings of the 31st International Conference on Computational Linguistics, pages 7546–7557
January 19–24, 2025.

especially in complex tasks. This early termination of reflection reduces the potential for iterative improvement and hinders performance in scenarios that require deeper, ongoing adaptation.

A possible reason for this issue is that the LLM used to generate reflection advice is driven by a static, task-independent reflection prompt. Combined with the unchanging nature of the LLM, this often results in repetitive and unhelpful advice after a few iterations. To address this, we introduce the DORA method (Dynamic and Optimized Reflection Advice), which aims to generate more task-adaptive and diverse reflection advice. Specifically, we introduce an external open-source small language model (SLM) to dynamically generate prompts for the reflection LLM. This SLM receives feedback from the agent and optimizes the prompt generation process using a non-gradient Bayesian Optimization (BO) algorithm (Chen et al., 2023c). By refining the prompts based on the agent's performance and advices diversity, the SLM ensures that the reflection process becomes more task-adaptive and effective over time.

We conducted experiments on the MiniWoB++ and Alfworld environments, covering a total of 17 different tasks. The results show that our DORA reflection method increased the average task success rate of agents by 19% in MiniWoB++ and 9% in Alfworld. Additionally, we performed an in-depth analysis to confirm that our method effectively mitigates the "early stop reflection" issue. Our contributions can be summarized as follows:

- We identify a shortfall in the previous reflection framework, which we term "early stop reflection," where continuous improvement is hindered, particularly in long-term or complex tasks.
- We propose a novel reflection framework, DORA, that dynamically generates task-adaptive and diverse reflection advice, improving agents' ability to learn from iterative reflection.
- Extensive experiments validate our method can resolve the premature halting of reflection in complex tasks, allowing for deeper and more effective iterative learning.

2 Related Works

2.1 LLM Agent Reflection

Recent studies highlight that Large Language Models (LLMs) have the potential to create autonomous agents, such as ReAct(Yao et al., 2023a), WebAgent(Nakano et al., 2021), Generative Agents(Park et al., 2023), Voyager(Wang et al., 2023a). Moreover, the reflection strategy enables agents to distill insights from their feedback with LLM as a reflector, including single-step generation (Madaan et al., 2023; Miao et al., 2023; Paul et al., 2023; Xi et al., 2023; Welleck et al., 2023), trajectories analyzing(Shinn et al., 2023; Huang et al., 2022; Li et al., 2023; Yao et al., 2023b; Chen et al., 2023b; Xi et al., 2024). However, recent studies have questioned the effectiveness of LLMs' self-reflection capabilities (Valmeekam et al., 2023), particularly in scenarios lacking external feedback(Huang et al., 2023). The reflection advice from LLMs often appears overconfident or inconsistent with the agent environment(Stechly et al., 2023).In this paper, we also reveal a shortfall of vanilla reflection and deploy a prompter to dynamically build and optimize the reflection prompt to address it.

2.2 Prompt Optimization

When dealing with huge or closed-source LLMs, gradient-descent approaches become less feasible, leading to a shift toward gradient-free prompt optimization methods.These methods typically use iterative sampling, starting with an initial prompt and repeatedly sampling and evaluating candidates to select the most effective one. Various methods have been employed, including: evolutionary algorithms(Xu et al., 2022; Guo et al., 2023; Chen et al., 2023a; Fernando et al., 2023; Yang and Li, 2023), which adaptively evolve prompts over generations; text editing search method (Prasad et al., 2023) that iteratively refine prompts through targeted editing; and reinforcement learning strategy (Deng et al., 2022), where prompts are optimized through interactive feedback.Additionally, recent methodologies directly leveraging the generative capabilities of LLM themselves for prompt optimization(Pryzant et al., 2023; Wang et al., 2023b; Yang et al., 2023). In this paper, We follow the work (Chen et al., 2023c) which employs an open-source language model for prompt generation and applies Bayesian optimization.

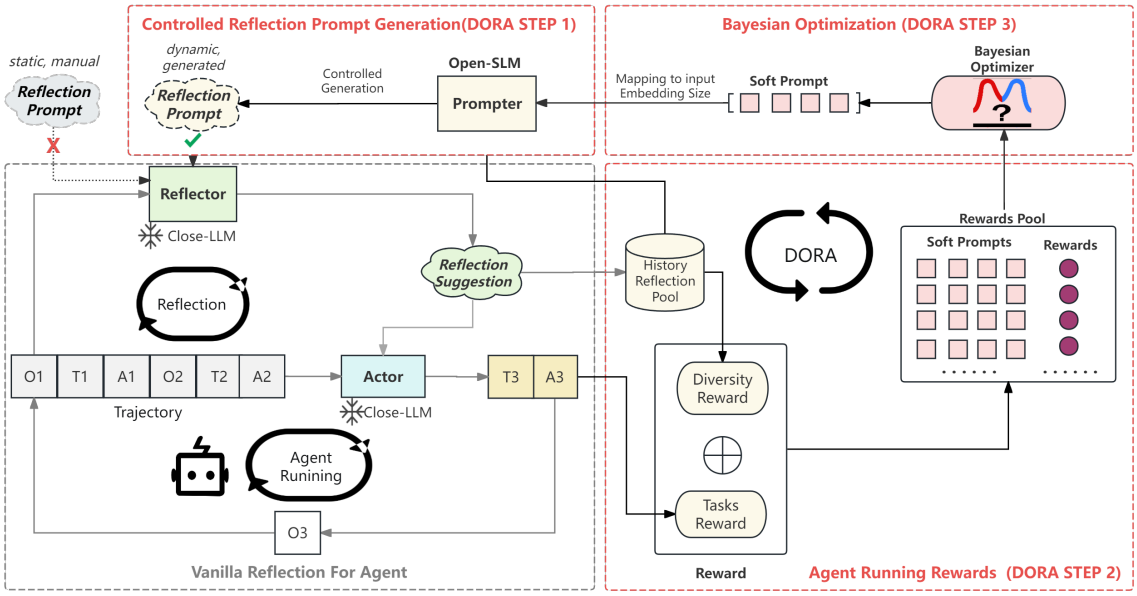

Figure 2: DORA method overview, left-side shows the basic framework of reflection, Agent thoughts(t) and chooses actions(a) based on what it observes(o) as trajectories. In the reflection stage, the **reflector** reviews the agent's trajectories based on a **reflection prompt** and then generates **reflection suggestion** for the agent's future actions and decisions. Right-side shows the process for reflection prompt auto optimization which includes controlled generating, agent running rewards calculating, and Bayesian Optimization for soft prompt

3 Methodology

3.1 Overview

Figure 2 presents an overview of our method. The lower-left section of the figure depicts the standard reflection process for an LLM agent. The key enhancement in the DORA reflection framework is the transition from using a static, manually crafted reflection prompt to a dynamic, generated one. This is achieved through a small open-source language model called **Prompter**.

This model employs techniques from controlled text generation using a few examples of reflection suggestions and a low-dimensional soft-prompt. With the dynamically generated reflection prompt, **Reflector** can analyze the agent's history and produce new reflection suggestions, guiding the agent's decisions in subsequent iterations. Then, we can measure the diversity of reflection suggestions and collect the overall performance of the agent across various similar tasks. These metrics serve as the objectives for optimizing. Finally, we use Bayesian optimization (BO) to update soft-prompt thus controlling the generation of reflection prompts for the next iteration.

3.2 Controlled Reflection Prompt Generation

In the DORA framework, the **Prompter** component is responsible for generating reflection prompts. This module uses an open-source language model with fewer parameters, such as LLama3-8B, which efficiently handles embedding-level inputs. Directly optimizing **Prompter** during the iteration process is not feasible because the final feedback is generated by a black-box agent. Therefore, we employ a non-gradient optimization approach to update the soft-prompts. This method allows us to influence the reflection prompts through a controllable generation process. Specifically, we concatenate the soft-prompt $\mathbf{p_s}$ with input-output examples in the embedding layer of **Prompter**. This process can be represented as follows:

$$prompt \sim LLM_{opensource}(p_s; E), \qquad (1)$$

where $\mathbf{p_s}$ represents the soft-prompt used for generation control, and $\mathbf{E}$ represents the demonstration examples that contain old reflection prompts and suggestions used for generation.

As input tokens to an open-source LLM, $\mathbf{p_s} \in \mathbf{R^d}$ usually has a too-high dimension (8192 for llama3) to be handled by black-box optimization approaches. Hence, we instead optimize a

lower-dimensional vector $\mathbf{v} \in \mathbf{R}^{\mathbf{d}'}$, where $\mathbf{d}' \ll \mathbf{d}$ and project it to $\mathbf{R}^{\mathbf{d}}$ using a simple projection. Similar to other black-box optimization methods, we sample the projection matrix $\mathbf{R}^{\mathbf{d}' \times \mathbf{d}}$ from Normal or Uniform distribution(Wang et al., 2016). So the whole controlled reflection prompt generation can be represented as:

$$prompt \sim LLM_{opensource}(W_{\text{proj}} \cdot v; E). \quad (2)$$

where W_{proj} represents a linear projection matrix initialized with a normal distribution and $\mathbf{v}$ represents low dimension vector used for optimization. We investigated how the choice of vector dimensions affects experimental results, refer to Figure 7.

For every generation, we randomly select two example from history pairs of reflection prompts and suggestions as demonstrations and concatenate them with updated soft-prompt to control new prompt generation. Soft-prompt can be considered as an additional "virtual word" connected in front of the embedding of examples. The specific formula is:

$$\text{prompt} = \text{LLM}\left(W_{\text{proj}} \cdot v \oplus \text{Embed}(t_{\text{example}})\right) \quad (3)$$

where e_{example} is the example string, $\text{Embed}(\cdot)$ is the embedding function of the LLM, $\oplus$ denotes concatenation, $\text{LLM}(\cdot)$ represents the forward pass through the large model to generate the prompt.

3.3 Agent Running Rewards

Through the above step, we generate a new reflection prompt that guides the Reflector to produce suggestions, shaping the agent's future actions.

$$suggestion = Reflector(prompt, \langle a_{i-1}, o_{i-1} \rangle), \quad (4)$$

$$a_i = Agent(suggestion, o_i), \quad (5)$$

Where a and o represent the agent's actions and observations in iteration i for the task.

Similar to using loss values in gradient optimization, non-gradient methods use the reward values of agent as prompt optimization target.Firstly, we focus on the agent's performance in the environment as the primary optimization target. In some environments, the agent's actions result in binary outcomes of success or failure (1 or 0), making single-task performance an ineffective reward measure. Therefore, we also consider the agent's success rate across multiple similar tasks as the reward value. The insight here is that an agent's experiences in similar tasks are transferable. This performance reward can be represented as:

$$R_{perfomance} = Environment(\langle a_i, o_i \rangle, task), \quad (6)$$

Secondly, we calculate the textual distance between reflection suggestions generated in two consecutive iterations as an auxiliary reward. This effectively increases the diversity of reflections and avoids repetition. Additionally, this auxiliary reward ensures that prompt optimization can still proceed even when the performance reward is zero. The diversity reward can be represented as:

$$R_{diversity} = Cosine(vector(R_i), vector(R_{i-1})), \quad (7)$$

Where $\mathbf{R_i}$ represents the reflection suggestion text in iteration i.

Finally, we sum the two reward values to form the optimization goal, incorporating an adjustable weight α. We also discuss the influence of α in Figure 8.

$$\mathcal{R} = (1-\alpha)\mathcal{R}_{performnce} + \alpha\mathcal{R}_{diversity}. \quad (8)$$

3.4 Bayesian Optimization for Soft Prompt

Bayesian Optimization (BO) iteratively selects the most promising points (soft prompts) to evaluate a black-box function (the entire agent with reflection can be regarded as a black-box function) by a surrogate model (usually a Gaussian process). The BO process can be divided into four steps:

Surrogate Model Initialization. Firstly, we model the relationship between lower-dimensional vector $\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}\}$ and their corresponding reward values $\mathbf{R} = \{r_1, r_2, \ldots, r_n\}$ using Gaussian Process Regression (GPR), where each $\mathbf{v}_i$ is a vector in d'-dimensional space and r_i is a scalar reward. We hypothesize the existence of a black-box function f such that $r_i = f(\mathbf{v}_i) + \epsilon_i$, where ϵ_i represents independently and identically distributed Gaussian noise, specifically $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. Then

$$f(\mathbf{v}) \sim \mathcal{GP}(m(\mathbf{v}), k(\mathbf{v}, \mathbf{v}')) \quad (9)$$

In our model, the mean function $m(\mathbf{v})$ is set to zero, and the covariance function $k(\mathbf{v}, \mathbf{v}')$ is defined using the Matérn kernel:

$$k_{\text{Matern}}(\mathbf{v}, \mathbf{v}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}d}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}d}{\ell} \right) \quad (10)$$

Here, $d = ||\mathbf{v} - \mathbf{v}'||_2$ represents the Euclidean distance between points $\mathbf{v}$ and $\mathbf{v}'$. The length scale parameter ℓ is sampled from a GammaPrior$(3.0, 6.0)$ distribution, and the smoothness parameter ν is set to 2.5. The function K_ν denotes the modified Bessel function of the second kind.

Training Dataset Build. Secondly, we start with an initial dataset $D_{\text{train}} = \{(v_i, r_i)\}_{i=1}^{n}$, where $\mathbf{v}_i$ represents the i^{th} soft prompt, and r_i is its associated reward. To enhance optimization efficiency from the onset, we randomly initialize 10 soft prompts and obtain their respective rewards to form our initial optimization set. Subsequently, we continuously augment the Bayesian Optimization (BO) training dataset with newly collected pairs of soft prompts v' and their corresponding rewards r'.

Fitting Surrogate Model. Thirdly, in fitting the Gaussian Process (GP) model to the described data, the objective is to identify a set of hyperparameters θ that maximize the marginal log-likelihood, $\log p(r_{\text{train}}|v_{\text{train}}, \theta)$. This is mathematically represented as:

$$\log p(r_{\text{train}}|v_{\text{train}}, \theta) = -\frac{1}{2}\mathbf{r}_{\text{train}}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}r_{\text{train}} - \frac{1}{2}\log|\mathbf{K} + \sigma_n^2\mathbf{I}| - \frac{n}{2}\log 2\pi \tag{11}$$

Here, $\mathbf{K}$ is the covariance matrix computed from the training inputs $\mathbf{D}_{\text{train}}$ using the covariance function $k(\mathbf{v}, \mathbf{v}')$, σ_n^2 represents the noise variance, and θ denotes the model's hyperparameters. n is the number of training data points.For practical implementation, we use botorch[2] Python library to facilitate this process.

Acquisition Next Point. Finally, based on the above model, we need to decide where to sample next. We use the Expected Improvement (EI) acquisition function, which quantifies the expected improvement over the current best reward value. For a maximization problem:

$$\text{EI}(\mathbf{x}) = \mathbb{E}\left[\max(0, f(\mathbf{v}) - f(\mathbf{v}^+))\right] \tag{12}$$

where $f(\mathbf{v}^+)$ is the reward of the current best soft-prompt. The next point to evaluate is chosen by maximizing the EI:

$$\mathbf{v}_{\text{next}} = \arg\max_{\mathbf{v}} \text{EI}(\mathbf{v}) \tag{13}$$

To identify the next point for optimization, we employ the standard Covariance Matrix Adaptation Evolution Strategy (CMA-ES)(Hansen and Ostermeier, 2001; Hansen et al., 2003), a widely recognized algorithm for evolutionary optimization. Through these four steps, we predict a new optimal point, setting the stage for the next iteration in the optimization cycle.

4 Experiments

4.1 Settings

Environment. In our experiments, we select MiniWoB and Alfworld, environments that closely mimic real-world agent operations and better represent the intricate and diverse nature of tasks agents face.**MiniWob++**[3] is a web-based simulation for diverse computer tasks ranging from simple clicks to complex, reasoning-based tasks like booking flight tickets. This environment allows for systematic assessment due to the varying complexity of tasks. It features a unified action space for both keyboard and mouse inputs, centered around HTML code, making it ideal for comprehensive agent evaluation. **Alfworld**[4] offers a collection of over 3,000 text-based interactive environments, challenging agents with multi-step tasks. Agents choose actions from a given list, gaining observations and binary rewards that determine their next state. The suite includes six varied tasks such as locating hidden objects, moving items, and object manipulation. These tasks have been applied in 134 different daily living environments, involving activities like finding a spatula, moving a knife, or chilling a tomato.

Baselines. We compared three major LLM-based agent frameworks as our baseline. Notably, we employ the zero-shot setting to ensure broader applicability and to more effectively showcase the actual capabilities of agents. **ReAct**(Yao et al., 2023a) utilizes the COT(Chain of thought) reasoning capabilities of large language models for agent construction, which doesn't involve reflection or any iterative optimization process. This agent will constantly interact with the environment, analyze the observation results, and make action decisions. **RCI**(Kim et al., 2023) starts with a

[2] https://github.com/pytorch/botorch

[3] https://miniwob.farama.org/index.html

[4] https://github.com/alfworld/alfworld

	1-screen-1-step		1-screen-n-step				n-screen-n-step				
	click-color	click-shape	click-checkboxes-soft	click-option	find-word	navigate-tree	click-collapsible-2	use-autocomplete	click-pie	click-menu	Average
Vicuna-7b											
React	0.33	0.56	0.67	0.11	0.33	0.67	0.0	0.22	0.11	0.33	0.33
RCI	0.33	0.44	0.67	0.0	0.22	0.78	0.0	0.11	0.0	0.22	0.28
Reflexion	0.44	0.67	0.78	0.44	0.33	0.78	0.0	0.89	0.11	0.33	0.48
DORA(w/o BO)	0.33	0.56	0.56	0.33	0.22	0.89	0.22	0.67	0.11	0.33	0.42
DORA(BO)	**0.56**	**0.78**	**0.89**	**0.56**	**0.44**	**1.00**	**0.89**	**1.0**	0.11	0.33	**0.66**
Llama3-8b											
React	0.44	0.67	0.78	0.22	0.33	0.78	0.0	0.22	0.11	0.33	0.39
RCI	0.33	0.44	0.78	0.11	0.22	0.78	0.0	0.11	0.0	0.33	0.30
Reflexion	0.44	0.78	1.0	0.56	0.44	0.78	0.0	1.00	0.11	0.33	0.54
DORA(w/o BO)	0.44	0.56	0.78	0.56	0.56	0.89	0.22	**0.78**	0.11	0.33	0.52
DORA(BO)	**0.56**	**0.89**	**1.0**	**0.67**	0.56	**1.0**	**0.89**	**1.0**	0.11	0.33	**0.70**

Table 1: Experiment results in MiniWoB++ environment, each task was tested across 11 specific scenarios to calculate the **task success rate**, with the best results highlighted in bold. W/o BO means removing the Bayesian optimization process, which means that the prompts used to control the generation of reflection instructions will no longer be updated based on feedback. We conduct the experiment with both Vicuna-7b and LLama3-8b as the SLM prompter.

language model generating a first response with zero-shot prompting. Essentially, RCI is a process where the model improves its work, similar to the Refiner method, but done in a step-by-step manner.
Reflexion(Shinn et al., 2023) is the state-of-the-art language agent architecture, Unlike RCI, the agent benefits from comprehensive trajectory feedback in the environment. But, its optimization goal stays the same and doesn't adapt to the environment. We use this as the baseline method for our primary comparisons and optimization efforts.
Implementation Details. The DORA method utilizes the llama3-8b/Vicuna-7b model as SLM prompter[5] and uses gpt-3.5-turbo-0125[6] to drive agent, selecting two examples from the history reflection pool for input and output. For each task, the React method conducts 40 trials, the Reflexion method undergoes 40 trials, and the DORA method starts with 4 initial instructions. Bayesian optimization is conducted over 5 rounds up to 40 trials.

4.2 Metric

We use success rate and agent rewards as evaluation metrics, and use rliable[7]library to calculate the aggregate metrics for agent rewards, which include the median, IQM, Mean and Optimality Gap of task rewards. IQM(interquartile mean) measures the degree of dispersion in data distribution which interpolates between mean and median across runs. Optimality Gap represents the gap between the performance of the current strategy and the potential optimal strategy performance(Agarwal et al., 2021).

4.3 Main Results

In this subsection, we compare DORA with the baselines on the miniwob++ and Alfworld environments. The task success rates are presented in Table 1 and Table 2. In the experimental evaluation within the miniwob++ environment, our proposed DORA method demonstrated consistently superior performance compared to the baseline methods (React, RCI, and Reflexion). As illustrated in Table 1, DORA achieved the highest success rate in both simple(such as 'click-tab-2') and difficult tasks(such as 'use-autocomplete'). We can see the 100% success rate in some task types. As the number of iterations increases, reflections guided by dynamically optimized prompts enable the Agent to complete all tasks within a category (10 of 10). This result is rarely achieved with baseline methods. For the Alfworld environment, The DORA method outperforms or is comparable to the basic reflexion method in most tasks, with an average success rate improvement of 11% across all six tasks. Notably, similar to the results in Miniwob++, the RCI method performs poorly, even worse than the original ReACT method. Through careful examination of the agent's historical trajectories, we found that the single-step critic in the RCI method often rejects the original correct reasoning results, leading to suboptimal behavior selection. We also conducted ablation experiments without using BO, the results indicate that the prompts used to control the generation of reflection instructions will no longer be updated based on feedback without the Bayesian Optimization process.

Furthermore, we calculated the aggregation mea-

[5] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
[6] https://platform.openai.com/docs/models/gpt-3-5-turbo-0125
[7] https://github.com/google-research/rliable?tab=readme-ov-file

	pick heat then place	look at obi	pick clean then place	pick two obi	pick and place	pick cool then place	Average
Vicuna-7b							
React	0.33	0.44	0.22	0.11	0.11	0.11	0.22
RCI	0.11	0.11	0.22	0.0	0.11	0.0	0.10
Reflexion	0.33	0.67	0.44	0.11	0.22	0.11	0.24
DORA(w/o BO)	0.33	0.33	0.33	0.11	0.22	0.11	0.24
DORA(BO)	**0.44**	**0.78**	**0.44**	0.11	**0.33**	**0.22**	**0.39**
Llama3-8b							
React	0.33	0.56	0.33	0.11	0.11	0.11	0.26
RCI	0.22	0.11	0.33	0.0	0.11	0.11	0.15
Reflexion	0.44	0.89	0.33	0.11	0.22	0.11	0.37
DORA(w/o BO)	0.44	0.67	0.33	0.11	0.22	0.22	0.33
DORA(BO)	**0.67**	**1.0**	**0.44**	0.11	**0.33**	**0.33**	**0.48**

Table 2: Experiment results in Alfworld environment, each task was tested across 11 specific scenarios to calculate the **task success rate**, with the best results highlighted in bold. W/o BO means removing the Bayesian optimization process, which means that the prompts used to control the generation of reflection instructions will no longer be updated based on feedback.We conduct the experiment with both Vicuna-7b and LLama3-8b as the SLM prompter.

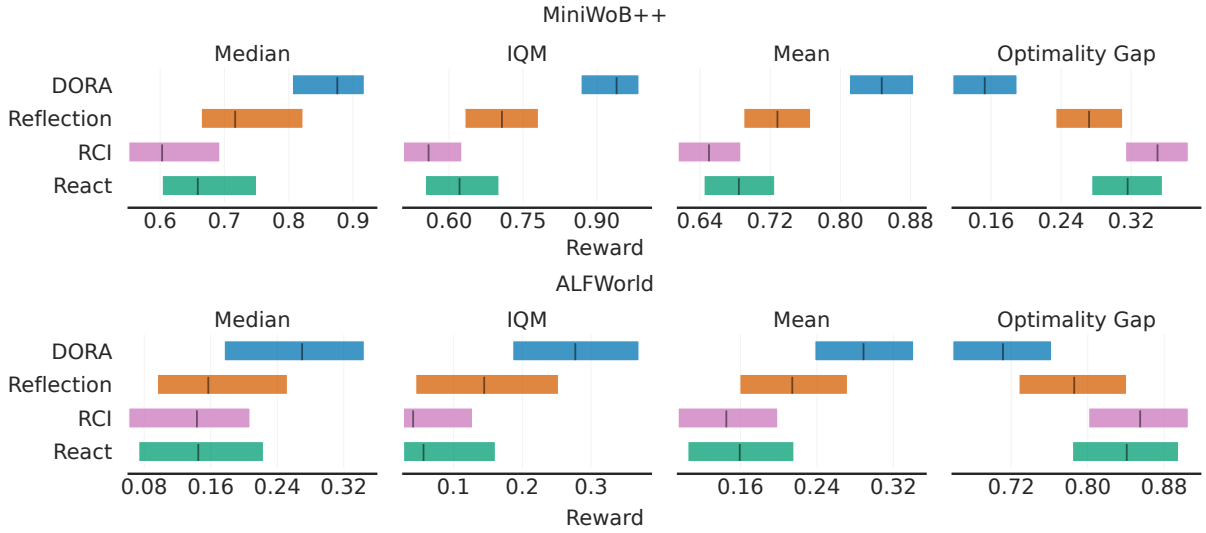

Figure 3: Aggregate metrics for agent rewards on MiniWoB++ and ALFWorld with 95%CIs(Confidence Intervals). The colored rectangle in the figure represents interval estimation, and the black line represents point estimation. The wider the rectangle, the greater the uncertainty of the estimated value. Higher mean, median and IQM scores and lower optimality gap are better.

sure of agent rewards on MiniWoB++ and ALFWorld environments, including the median, mean, interquartile mean (IQM), and optimality gap. Aggregate metrics on MiniWoB++ and ALFWorld with 95%CIs are as shown in Figure3. The median, mean, and IQM of the DORA method are higher than those of other methods, and the optimization gap of the DORA method is also the lowest, which means that the DORA method is more effective and stable. And in the Figure 3 and Appendix A, we provide detailed rewards distribution and compare the probability of improvement of DORA.

5 Analysis

In this section, we thoroughly analyze the reflection details and evaluate the effectiveness of our method under different settings. We want to answer the following for research questions.

RQ 1: Does DORA help alleviate the "Early Stop Reflection" issue? To address this question, we examined the distribution of successful reflection iterations within the MiniWoB++ environment, as shown in Figure 4. Our analysis reveals that, consistent with our observations, successful reflections using the standard reflection method are concentrated in the early stages, typically succeeding after one or two reflections. In contrast, the distribution of successful reflection rounds with

Iteration	Prompt	Suggestion
turn 1	The goal of reflection is to promote personal growth,self-improvement, and self-discovery.	For screen index=0, you should take a moment to reflect on the task at hand, your current progress, and any potential obstacles you may encounter.Consider how you can approach the task with a clear mind and a positive attitude to achieve the best results.
turn 2	The instruction for reflection is a process in which an individual consciously reflects on their thoughts, feelings, and actions to gain a deeper understanding of themselves and their experiences.	For screen index=0, you should consider using the arrow keys to navigate through the file tree and locate the "Deneen" folder or file.
turn 3	The instruction for reflection was suggest it to click on the element with the reference 8 to access the folders.	For screen index=0, you should click on the \'Alan\' folder in the file tree to find the file or folder named \'Agustina\'.
turn 4	The instruction for reflection was to navigate through the file tree and find and click on the folder or file named "Alan"	For screen index=0, you should consider using the "click_element" action on the folder named "Alan" to navigate deeper into the file tree.

Table 3: Case study for the generated reflection prompt and corresponding reflection suggestions in for turns.

the DORA method is more uniform.We believe that while the randomly initialized prompts used in DORA may not perform as well as manually set static prompts in the initial few rounds of reflection, the iterative feedback from the agents ensures that the DORA method maintains effective reflection and improves overall success rates in subsequent iterations.

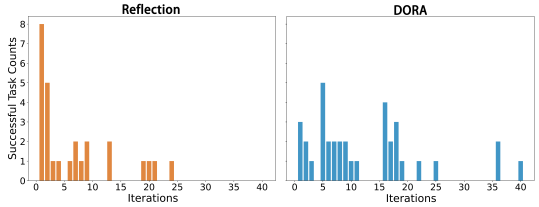

Figure 4: The success reflection rounds distribution in miniwob++. Compared to the baseline method, Dora can achieve a uniform result.

RQ 2: Does DORA help improve the performance of agents on difficult tasks? . The tasks in miniwob++ can be categorized into three levels based on their complexity: **1-screen-1-step tasks**, which are completed with a single action within one screen, such as "click-color" and "click-shape"; **1-screen-n-steps** tasks, which require multiple actions on a single screen to complete, including operations like "click-checkboxes-soft", "click-option", "find-word"; and **n-screen-n-steps** tasks, which involve navigating through multiple screens and performing several steps, including "navigate-tree", "click-collapsible-2". The difficulty of these tasks increases progressively across the categories. In our study, we compared the success rates of the baseline reflection method and the DORA method across different levels of task complexity shown in Figure 5. We found that our method shows improvements in success rates across all levels, with the most significant improvement observed in the most challenging n-screen-n-steps scenarios.

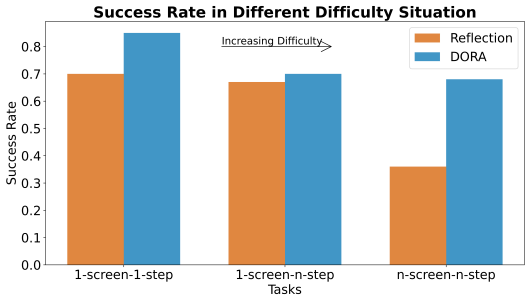

Figure 5: The success rate in three-level task categories.

Question 3: Does DORA also lead to toxic reflection, better or worse? Previous studies have identified a phenomenon called "toxic reflection," which leads to worse performance. To explore this, we compared the REACT baseline method, which involves multiple attempts without reflection, to scenarios involving reflection. Reflection is considered "effective" if it reduces the steps needed for success, "ineffective" if the number of steps stays the same, and "toxic" if the steps increase.The distribution of these three outcomes is shown in Figure 6. The findings reveal that while the DORA method can sometimes result in toxic reflection, its main advantage lies in turning many instances of ineffective reflection into effective ones, thus improving overall performance.

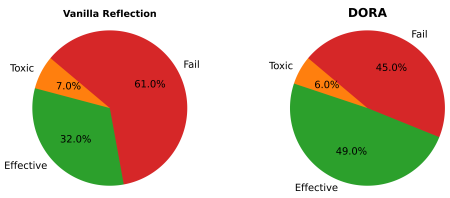

Figure 6: The proportion of effective, failed, and toxic reflections. While DORA did not significantly reduce the proportion of toxic reflections, it effectively addressed the issue of failed reflections.

Question 4: What features of the reflection prompts and suggests produced by the DORA method? We showcased several reflection prompts and corresponding suggestions generated by the DORA framework during its iterative process in Table 3. Initially, the reflection prompts are similar to those set manually, offering general guidance. With the progression of the iteration process, these prompts increasingly relate to the specific context of the task, finally pointing towards targeted solutions. Additionally, by setting diversity as an optimization objective, we observe a decrease in the occurrence of repetitive or redundant suggestions for reflection. However, the limitations of Bayesian optimization may lead to certain iterations producing entirely irrelevant reflection prompts, as shown in Figure 4.

Question 5: What should be the dimension size of soft prompt for Bayesian optimization? The size of the soft prompt used to control the generation of reflective prompts can impact the effectiveness of the method. Generally, when optimizing with non-gradient methods like BO, setting the dimensionality of the soft prompt too high can make optimization challenging and increase computational costs. However, setting it too low risks oversimplifying the problem, and missing critical interactions between variables. We conducted an ablation study on three tasks where improvements are significant, to explore the impact of soft prompt size on the experimental results. The findings of this study are depicted in Figure 7. It is observed that when the dimensionality of the soft prompt is high, the performance of the agent tends to decline. This could be attributed to the long length and complexity of the optimization process, making it more challenging. In the context of the DORA method, optimizing smaller soft prompts appears to be more suitable. Furthermore, when the soft prompt dimensionality is extremely low, such as 2, there is a noticeable decrease in performance, especially in the task "click-color."

Question 6: How do hyperparameters affect the performance of DORA methods? We discovered that relying solely on task completion can lead to very low or even zero rewards in early iterations, causing the loss of the optimization goal. Incorporating diversity rewards helps guide the model to explore a broader range of reflective instructions. In our main experiment, we set alpha to 0.5. Recognizing the importance of systematically evaluating different weights, we conduct additional experiment, as shown in Figure8.

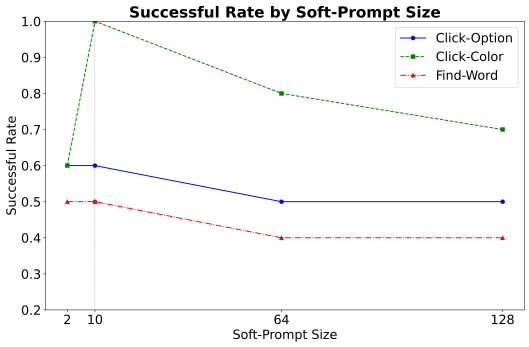

Figure 7: Ablation study results (three task's success rate) on miniwob++. The horizontal axes represent different soft prompt dimensions.

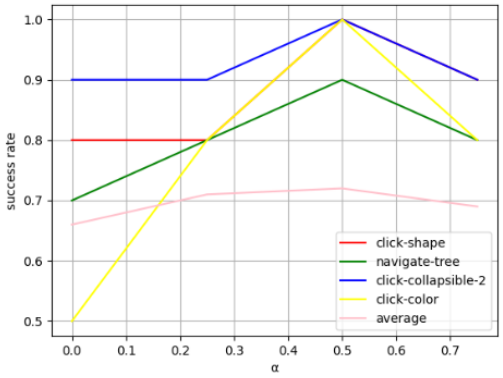

Figure 8: The hyper-parameter alpha balances the weight between diversity rewards and task performance rewards.

6 Conclusion

In this work, we introduces the Dynamic Optimization Reflection prompt for LLM-based Agent (DORA), addressing the "Early Stop Reflection" issue in autonomous agents driven by LLMs. By replacing static reflection prompts with dynamically generated ones and refining these prompts via Bayesian Optimization, DORA significantly improves agents' ability to learn from history mistakes. Our experiments across two environments and 16 distinct tasks have empirically demonstrated DORA's effectiveness in overcoming the "Early Stop Reflection" issue.

Limitations

Firstly, our approach was tested in only two environments, suggesting the potential for future experiments to extend to a broader range of settings.

Secondly, our approach is still affected by the issue of toxic reflections, which we plan to address in the future.

Ethics Statement

The methods presented in this work involve agent operating environments and corresponding datasets that are publicly available. The goal of this research is to enhance the self-improvement ability of agents during long-term operations. In the context of such AGI applications, it is also important to focus on their long-term stability and controllability, ensuring that they do not produce unexpected behaviors that could lead to adverse effects.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320.

Angelica Chen, David M. Dohan, and David R. So. 2023a. Evoprompting: Language models for code-level neural architecture search. *CoRR*, abs/2302.14838.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023b. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2023c. Instructzero: Efficient instruction optimization for black-box large language models. *CoRR*, abs/2306.03082.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *EMNLP*, pages 3369–3391. Association for Computational Linguistics.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *CoRR*, abs/2309.16797.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *CoRR*, abs/2309.08532.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *CoRR*, abs/2310.01798.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner monologue: Embodied reasoning through planning with language models. In *CoRL*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR.

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

Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. 2023. A zero-shot language agent for computer control with structured reflection. In *EMNLP (Findings)*, pages 11261–11274. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *CoRR*, abs/2303.17651.

Ning Miao, Yee Whye Teh, and Tom Rainforth. 2023. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *CoRR*, abs/2308.00436.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *UIST*, pages 2:1–2:22. ACM.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. REFINER: reasoning feedback on intermediate representations. *CoRR*, abs/2304.01904.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *EACL*, pages 3827–3846. Association for Computational Linguistics.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *EMNLP*, pages 7957–7968. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. GPT-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *CoRR*, abs/2310.12397.

Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. 2023. Can large language models really improve by self-critiquing their own plans? *CoRR*, abs/2310.08118.

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

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2023b. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *CoRR*, abs/2310.16427.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. 2016. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Intell. Res.*, 55:361–387.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *ICLR*. OpenReview.net.

Zhiheng Xi, Wenxiang Chen, Boyang Hong, Senjie Jin, Rui Zheng, Wei He, Yiwen Ding, Shichun Liu, Xin Guo, Junzhe Wang, et al. 2024. Training large language models for reasoning through reverse curriculum reinforcement learning. *arXiv preprint arXiv:2402.05808*.

Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Jia Liu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Self-polish: Enhance reasoning in large language models via problem refinement. In *EMNLP (Findings)*, pages 11383–11406. Association for Computational Linguistics.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. 2022. GPS: genetic prompt search for efficient few-shot learning. In *EMNLP*, pages 8162–8171. Association for Computational Linguistics.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *CoRR*, abs/2309.03409.

Heng Yang and Ke Li. 2023. Instoptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators. In *EMNLP (Findings)*, pages 13593–13602. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023a. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. 2023b. Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151*.

Appendix

A Probability of Improvement

Probability of improvement shows how likely it is for X to outperform Y on a randomly selected task. Specifically, $\mathbf{P}(X > Y) = \frac{1}{M}\sum_{m=1}^{M}\mathbf{P}(X_m > Y_m)$, where $\mathbf{P}(X_m > Y_m)$ is the probability that X is better than Y on task m. As shown in Figure9, each row shows the probability of improvement, with 95% bootstrap CIs, given that DORA was claimed to be better than Y . For all algorithms, results are based on multiple runs per task.

B Extra Analysis

Figure 9: Average Probability of Improvement for agent rewards on MiniWoB++ and ALFWorld. Each subplot shows the probability of improvement of DORA compared to all other algorithms.

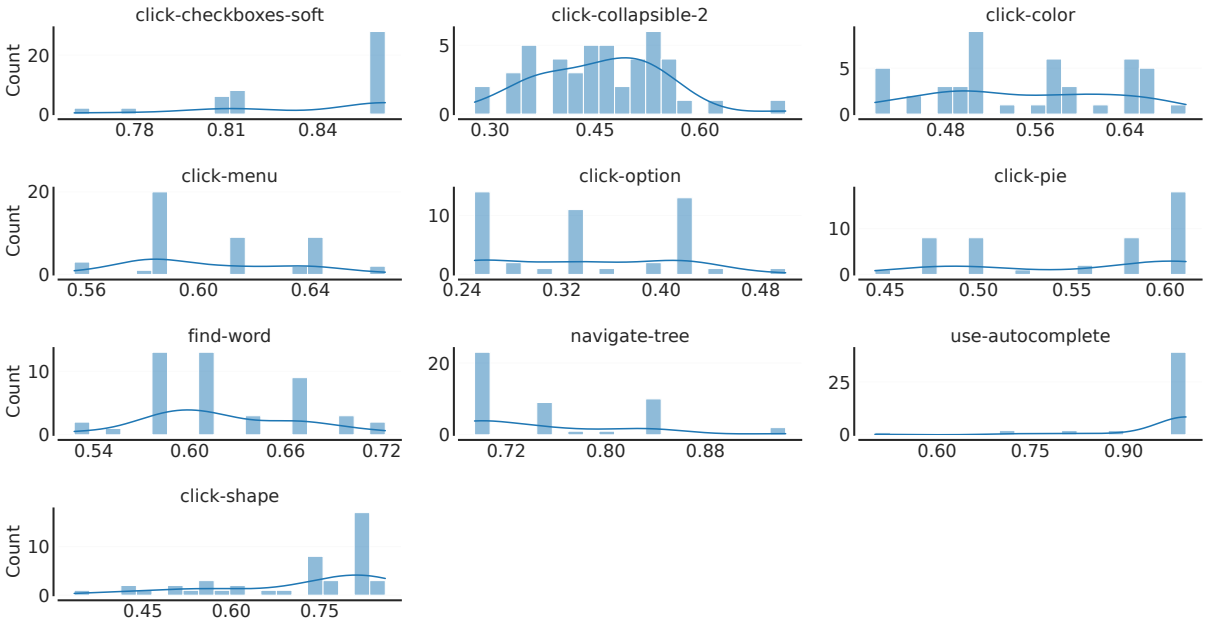

Figure 10: MiniWoB++ agent rewards distributions. Histogram plot with kernel density estimate of reward of DORA on 10 games in the MiniWoB++ benchmark.

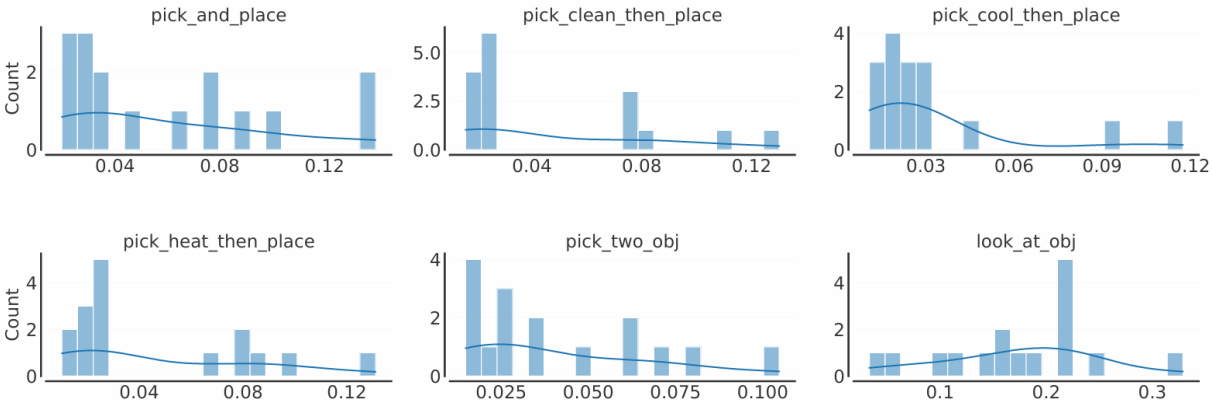

Figure 11: ALFWorld agent rewards distributions. Histogram plot with kernel density estimate of reward of DORA on 6 games in the ALFWorld benchmark.