Re-ReST: Reflection-Reinforced Self-Training for Language Agents

Anonymous ACL submission

Abstract

Finetuning language agents with reasoning-action trajectories is effective, but obtaining these trajectories from human annotations or stronger models is costly and sometimes impractical. In this paper, we investigate the use of self-training in language agents, which can generate supervision from the agent itself, offering a promising alternative without relying on human or stronger model demonstrations. Self-training, however, requires high-quality model-generated samples, which are hard to obtain for challenging language agent tasks. To address this, we present Reflection-Reinforced Self-Training (Re-ReST), which uses a *reflector* to refine low-quality generated samples during self-training. The reflector takes the agent's output and feedback from an external environment (e.g., unit test results in code generation) to produce improved samples. This technique enhances the quality of inferior samples and efficiently enriches the self-training dataset with higher-quality samples. We conduct extensive experiments on open-source language agents across tasks, including multi-hop question answering, sequential decision-making, code generation, visual question answering, and text-to-image generation. The results demonstrate the effectiveness of self-training and Re-ReST in language agent tasks, with self-training improving baselines by 7.6% on HotpotQA and 28.4% on AlfWorld, and Re-ReST further boosting performance by 2.0% and 14.1%, respectively. Our studies also confirm the efficiency of using a reflector to generate high-quality samples for self-training. Moreover, we demonstrate a method to employ reflection during inference without ground-truth feedback, addressing the limitation of previous reflection work. Our code will be publicized upon publication.

1 Introduction

Large language models (LLMs) (Kenton and Toutanova, 2019; Touvron et al., 2023; Achiam et al., 2023) have demonstrated potential in interacting with external environments and addressing practical interactive tasks, resulting in a new class — language agents (Nakano et al., 2021; Yao et al., 2022). Finetuning LLMs for agentic tasks has proven effective, yet existing works rely on data generated by stronger models (e.g., GPT-4) (Chen et al., 2023; Yin et al., 2024), which are not always available (e.g., to improve the strongest model).

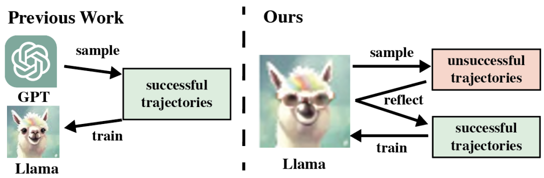

Figure 1: Previous agent training methods (Chen et al., 2023; Yin et al., 2024) distill knowledge from stronger models (e.g., GPT-4) to weaker ones (e.g., Llama-2). In contrast, we adopt self-training and improve it with reflection to improve agents more autonomously, which reduces reliance on external propriety models and maintains a fully open-source framework.

Among the potential techniques to improve agents (Ouyang et al., 2022; Wang et al., 2023b; Li et al., 2024; Chen et al., 2024), self-training holds promise for enhancing agent performance for challenging agentic tasks. The self-training process typically involves refining the model by generating samples, assessing their quality through rewards, and updating the model by training on high-quality samples. Compared with existing agent training methods (Chen et al., 2023; Yin et al., 2024), self-training can autonomously improve agents and reduce the discrepancy between the agent's training data and its original predictions. Additionally, as in Figure 1, self-training can potentially allow for the development of performant agents within a fully open-source framework, without relying on closed-source, proprietary models. Given these benefits, we propose to investigate the use of self-training in language agents in this paper.

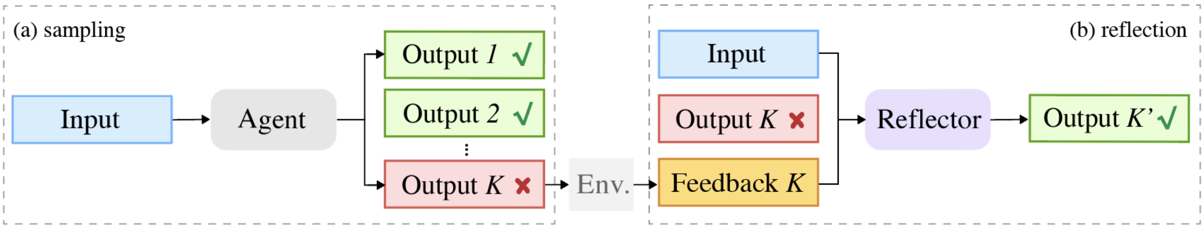

Figure 2: An overview of our Re-ReST method. Our approach incorporates self-training in language agent tasks by sampling multiple outputs from an agent and using positive samples for training. To enhance the effectiveness of self-training in language agents, we introduce a reflector mechanism. If a sample is incorrect, the reflector adjusts the agent's output based on environmental feedback. The corrected sample is then incorporated into the training data, thereby improving the overall self-training process.

However, one significant challenge for applying self-training in language agent tasks lies in the acquisition of high-quality samples to achieve good performance. Specifically, self-training requires a substantial amount of high-quality samples, while relying solely on model-generated samples can be inefficient, particularly for language agent tasks that demand multi-step reasoning and long-horizon planning. As a result, it is challenging to obtain good samples solely through sampling. Moreover, the common practice of discarding low-quality samples neglects their potential for improvement and effective utilization, thus limiting the overall efficacy of self-training methods.

To address these issues, we propose Reflection-Reinforced Self-Training (Re-ReST), which enhances the self-training algorithm using a reflection model. Re-ReST incorporates a *reflector* during self-training, which improves sample quality by utilizing environmental feedback such as execution successes and unit test outcomes. Specifically, the reflector transforms lower-quality samples into higher-quality ones, leveraging the capability of LLMs to self-improve when provided with accurate ground-truth feedback (Huang et al., 2024). Consequently, it enriches the training dataset, enabling more effective bootstrapping. After training, only the agent model is used for inference, ensuring no additional computational burden during testing. Unlike existing self-reflection methods (Madaan et al., 2023; Shinn et al., 2023; Pan et al., 2023), Re-ReST only requires access to feedback during training, not during inference, making our setting more realistic and practical.

We conduct extensive experiments with open-source LLMs across a wide range of tasks, including multi-hop question answering, sequential decision-making, code generation, visual question answering, and text-to-image generation. Our results first demonstrate the potential of self-training in language agent tasks, showing improvements over few-shot baselines in long-horizon planning tasks, with gains of 7.6% on HotpotQA and 28.4% on AlfWorld. By incorporating Re-ReST, we further enhance performance significantly by 2.0% and 14.1% on HotpotQA and AlfWorld, respectively, achieving results better or comparable to models relying on commercial APIs. Ablation studies confirm the efficiency of the reflection model in generating high-quality self-training samples. Furthermore, we explore using our reflection model during inference with self-consistency decoding, which improves the model performance while alleviating the need for ground-truth feedback required by previous work (Huang et al., 2024). Additionally, we demonstrate the application of our method in preference optimization objectives.

2 Method: Re-ReST

Self-Training. Formally, given a dataset $U = \{x_i\}_{i=1}^N$, self-training begins by using a base model $\mathcal{M}$ to generate a pseudo-label $\hat{y}_i = \mathcal{M}(x_i)$ for each instance $x_i \in U$. Subsequently, a subset of $\{(x_i, \hat{y}_i)\}_{i=1}^N$ is selected based on a scoring function, and $\mathcal{M}$ is finetuned on this selected subset. For language agents, we define the label y as a trajectory comprising interleaved thoughts and actions, as described in ReAct (Yao et al., 2022). We propose adopting the self-training paradigm by training language agents with their self-generated thought-action trajectories.

Overview of Re-ReST. Obtaining high-quality samples through self-sampling can be challenging, particularly for complex language agent tasks. To address this issue, we introduce Re-ReST, which aims to enhance the pseudo-label generation process in self-training for language agents. As illustrated in Figure 2, we propose improving low-quality samples using a reflection model with external feedback. We then enrich the self-training data by incorporating these corrected generations. This process generates high-quality samples efficiently by correcting low-quality ones with ground-truth feedback during training.

2.1 Components

Our method involves two models, including a language agent $\mathcal{M}$ that generates text and actions, and a reflection model $\mathcal{R}$ that improves a low-quality sample. The reflection model $\mathcal{R}$ has access to an external environment $\mathcal{E}$ that can provide external feedback to a generated sample (e.g. numerical scores and/or verbal error information). We illustrate each of these modules in the following part.

Language Agent. The language agent $\mathcal{M}$ is built upon a large language model (LLM) that is trained or prompted to generate thoughts and actions given a task. Formally, given an instance x_i, the agent $\mathcal{M}$ generates its output $\hat{y} \sim \mathcal{M}(\mathbf{y}|x)$ containing its actions. The agent can first generate its reasoning traces before outputting its actions, which has been demonstrated to improve the model performance and interpretability (Yao et al., 2022).

Reflector. The reflection model $\mathcal{R}$ is also instantiated as an LLM, the goal of which is to improve the language agent's generations given external feedback. We assume that during training, an external environment $\mathcal{E}$ can evaluate a generated sample and provide feedback $\mathcal{E}(x, \hat{y})$ to the agent. The feedback can be a binary success status and/or error information. For example, in code generation tasks, the environment can execute the model-generated code on unit tests, providing information on whether the code has syntax errors and whether it can pass the unit tests. Having access to such an environment is important in our setting, as it has been shown that an LLM cannot perform self-correction without high-quality external feedback (Huang et al., 2024). The reflection model generates a corrected sample $\tilde{y} \sim \mathcal{R}(\mathbf{y}|x, \hat{y}, \mathcal{E}(x, \hat{y}))$ given the task information x, the agent generation $\hat{y}$, and the environmental feedback $\mathcal{E}(x, \hat{y})$. It can optionally first state its reasoning process (e.g., which specific actions could be corrected) before generating the corrected answer.) The use of the reflection model can improve self-training by finding good solutions efficiently because of the additional information provided (i.e., the agent's previous trial and the environmental feedback.) We do not share the model parameters between the agent and reflector in this paper.

2.2 Data Generation

We then describe how we generate self-training data for the language agent $\mathcal{M}$. The data generation process involves two steps, including the initial generation step with the language agent itself and the reflection step with the reflector, and we obtain the agent-generated dataset $\mathcal{D}_{\mathcal{M}}$ and reflector-generated dataset $\mathcal{D}_{\mathcal{R}}$ from the two steps.

Initial Generation. As in the standard setup, given an instance x, we sample k generations $\{\hat{y}^j\}_{j=1}^k$ from the current language agent model $\hat{y}^j \sim \mathcal{M}(\mathbf{y}|x)$. Then, the environment $\mathcal{E}$ scores the generation and provides feedback $\mathcal{E}(x, \hat{y}^j))$. If the score exceeds a threshold, we add the instance to $(x, \hat{y}^j)$ to the training data $\mathcal{D}_{\mathcal{M}}$. In practice, we observe that setting $k = 3$ achieves a good balance between efficiency and effectiveness.

Reflection with Environmental Feedback. The initial generation step only relies on the agent model $\mathcal{M}$ itself to generate data. For a sampled generation $\hat{y}^j$, if the score does not pass the threshold, we will feed it to the reflection model for refinement. The reflector takes as inputs the task information x, the agent's prior generation $\hat{y}^j$, and the environmental feedback $\mathcal{E}(x, \hat{y}^j))$, and then generates the corrected sample $\tilde{y}^j \sim \mathcal{R}(x, \hat{y}^j, \mathcal{E}(x, \hat{y}^j))$. The corrected sample $\tilde{y}^j$ will also be evaluated by the environment and we will add it to the reflector-generated training dataset $\mathcal{D}_{\mathcal{R}}$ if its score exceeds the threshold. While the reflection procedure can be iteratively applied multiple times as per Shinn et al. (2023), in this study, we limit this process to a single iteration for the sake of efficiency. This means that each generated sample $\hat{y}^j$ is allowed a maximum of one refined counterpart $\tilde{y}^j$.

2.3 Model Training and Inference

We first train the reflector $\mathcal{R}$ parameterized by $\theta_{\mathcal{R}}$ and then use the trained reflector to generate the reflection data $\mathcal{D}_{\mathcal{R}}$. Afterward, we combine $\mathcal{D}_{\mathcal{R}}$ and the agent's self-generated data $\mathcal{D}_{\mathcal{M}}$ to train the agent model $\mathcal{M}$ parameterized by $\theta_{\mathcal{M}}$.

Reflector Training. While base LLMs can perform self-reflection or self-correction without any finetuning given ground-truth feedback (Shinn et al., 2023), we propose to further improve its reflection ability with the self-generated data. First, from the initial generation step, we obtain multiple generations $\{y^j\}_{j=1}^k$ from the agent model $\mathcal{M}$. For each correct generation y^w and incorrect generation y^l with its environmental feedback $\mathcal{E}(x, \hat{y}^l)$ in $\{y^j\}_{j=1}^k$, we will add the instance $\langle x, y^l, \mathcal{E}(x, \hat{y}^l), y^w \rangle$ to the agent-generated dataset $\mathcal{D}_{\mathcal{M}}^{\mathcal{R}}$ for reflector training. In addition, the reflector generates its self-training dataset in a zero-shot manner $\mathcal{D}_{\mathcal{R}}^{\mathcal{R}}$ similar to the agent initial generation step. Combining the two generated datasets, we train the reflector on $\mathcal{D}_{\mathcal{M}}^{\mathcal{R}} \cup \mathcal{D}_{\mathcal{R}}^{\mathcal{R}}$ with the standard maximum log-likelihood objective first before generating the training data $\mathcal{D}_{\mathcal{R}}$ for the language agent:

$$\mathcal{L}_{MLE}(\theta_{\mathcal{R}}) = -\mathbb{E}_{(x,y^l,y^w)\sim\mathcal{D}_{\mathcal{M}}^{\mathcal{R}}\cup\mathcal{D}_{\mathcal{R}}^{\mathcal{R}}} \log p_{\theta_{\mathcal{R}}}(y^w|x, y^l). \tag{1}$$

Language Agent Training. After we have the base language agent to generate the self-training data $\mathcal{D}_{\mathcal{M}}$ and the improved reflector to generate the reflector-generated data $\mathcal{D}_{\mathcal{R}}$, we train the language agent jointly on $\mathcal{D}_{\mathcal{M}} \cup \mathcal{D}_{\mathcal{R}}$:

$$\mathcal{L}_{MLE}(\theta_{\mathcal{M}}) = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\mathcal{M}}\cup\mathcal{D}_{\mathcal{R}}} \log p_{\theta_{\mathcal{M}}}(y|x). \tag{2}$$

Besides the maximum log-likelihood objective, because the reflection training and data generation process involves the use of preference pairs, it is natural to use preference optimization objectives such as DPO (Rafailov et al., 2023) for training, which we will discuss in the experiment section.

Inference. During inference, accessing high-quality environmental feedback is often challenging, which can cause inference-time self-reflection algorithms to fail (Huang et al., 2024). Therefore, we only have the agent $\mathcal{M}$ directly output generations without the reflector during inference. This approach eliminates the need for feedback and avoids any additional computational overhead. A potential method to integrate the reflector into the inference process involves first training a scorer to evaluate the agent's output. If the score falls below a certain threshold, self-correction can then be performed, which we leave as a future direction. Additionally, we propose performing reflection regardless of environmental feedback and employing self-consistency to derive the final results from both the agent's outputs and the reflector's outputs, as shown in the experiment section.

3 Experiments

We experiment with multi-hop reasoning, sequential decision-making, code generation, visual question answering, and text-to-image generation. We present the experimental settings and results for each task. In all our experiments, we advocate for the use of open-source models and aim to avoid black-box, closed-source commercial models whenever possible.

3.1 Multi-Hop Reasoning

Dataset. We use the HotpotQA dataset (Yang et al., 2018), a well-established question-answering dataset featuring multi-hop reasoning and knowledge retrieval. It is constructed based on Wikipedia and an agent needs to retrieve and reason over multiple supporting documents to answer a question. We sample 5,000 training instances randomly for self-training and 500 instances from the development set for evaluation as in Chen et al. (2023).

Model Setup. We build both the agent model and the reflector upon the Llama-2-13B and Llama-3-8B models (Touvron et al., 2023). Note that different from previous work (Shinn et al., 2023; Chen et al., 2023; Yin et al., 2024), we do not employ a stronger language model such as GPT-3.5/4 for data generation or self-reflection, ensuring that the models do not benefit from knowledge distillation. Following Shinn et al. (2023), we use the ReAct (Yao et al., 2022) method where at each step, the agent model first generates its thoughts and then performs an action. The action is chosen from (1) Search[entity], which searches the exact entity on Wikipedia, (2) Lookup[keyword], which localizes a keyword in the retrieved passages, and (3) Finish[answer], which returns the answer and finishes the task. We use a free Wikipedia API[1] for passage retrieval and keyword lookup.

Training and Evaluation Setup. We use 2-shot prompting for few-shot agent and reflector data generation as in Shinn et al. (2023). For each training instance, the agent model samples 3 generations. The generation is evaluated with the exact match metric (i.e., if the generated answer is exactly the same as the ground-truth answer). The retrieval and evaluation results are given to the reflector as the environmental feedback for self-correction. We use Low-Rank Adaptation (LoRA) (Hu et al., 2022) for training the language models for efficiency. The

[1] https://python.langchain.com/docs/integrations/tools/wikipedia

agent and reflector models are trained for 3 epochs with a learning rate of 3e-4.

Main Results. We list the main results in Table 1. As shown in the table, self-training can significantly improve the model performance from an EM score of 20.0% to 27.6% for Llama-2 and from 30.0% to 34.4% for Llama-3. However, only 37.1% and 48.3% of the training instances are correctly solved by the agent model and are used for self-training respectively. By integrating our reflector model into the process, the agent can solve more training instances and thus have more data for training the agent model, increasing the EM scores significantly. In addition to our implemented models, following previous work (FireAct (Chen et al., 2023) and LUMOS (Yin et al., 2024)) that use GPT-3.5/4 for data generation and model finetuning, we employ GPT-4 to generate 0.5k instances and first train the agents with the GPT-4 generated data before self-training. Results demonstrate that 1) self-training is a stronger baseline than FireAct under a fair setting where the same QA tool is used; 2) we can achieve comparable or better performance of our model than these methods, even though both of them use strong knowledge retrieval models (i.e., SerpAPI[2] for FireAct and GPT-4 for LUMOS), which are costly and non-scalable. By contrast, we use the free Wikipedia API.

3.2 Sequential Decision-Making

Dataset. We also assess the proposed approach on sequential decision-making using ALFWorld (Shridhar et al., 2021). ALFWorld comprises a collection of text-based settings designed to test an agent's ability to complete multi-step tasks across diverse interactive environments. Following Yao et al. (2022); Shinn et al. (2023), we operate under the assumption that the agents are devoid of any access to successful trajectories, relying solely on a binary indicator of task success or failure. Our evaluation encompasses testing the agent across 134 previously *unseen* environments, spanning six diverse tasks. These tasks range from locating concealed items and transporting objects to interacting with objects using other items.

Model Setup. We build the agent and the reflector upon the Llama2-7b (Touvron et al., 2023). At each step, the agent can either contemplate its next move or generate admissible actions for execution as in Yao et al. (2022). Following the heuristics outlined by Shinn et al. (2023), we trigger the reflector model for self-reflection if the agent repeats an action with the same response over three cycles, or if it performs over 30 actions in an environment.

Training and Evaluation Setup. We use one-shot prompting instead of the two-shot prompting in Shinn et al. (2023) for the models so that we can better fit a trajectory into the context window of Llama-2. We train the agent and reflector models on the collected trajectories for 2 epochs with a learning rate of 2e-5 using LoRA.

Results. As shown in Table 2, it is evident that the base Llama model faces challenges in adapting to the experimental environment, but self-training can significantly improve the model performance. A significant point to highlight is that the model operates without access to complete trajectories during the experiment. Despite this limitation, it demonstrates a notable improvement in performance within unseen environments—increasing the success rate from 8.9% to 37.3% through the utilization of self-augmented trajectories. Furthermore, the implementation of the reflector contributes a 14.1% uplift in success rates, which affirms the efficacy of our proposed method.

3.3 Programming: Code Generation and Visual Question Answering

Dataset. For code generation, we experiment with the Python code writing task on MBPP (Austin et al., 2021) and visual programming on GQA (Hudson and Manning, 2019). The MBPP benchmark consists of around 1,000 Python programming problems, with each problem paired with unit test cases. We follow its official split for the training and test data. The availability of the training set and its provided unit test cases make it suitable for our reflector to reflect and correct the model-generated code. For GQA, we randomly sample a subset of 5,000 data points for training and 1,000 data for testing.

Model Setup. We build both the agent model and the reflector upon the CodeLlama-13B model (Roziere et al., 2023). For MBPP, following Roziere et al. (2023), the agent model is given the unit test cases during code generation. Similarly, the reflection model is given the agent generation and its unit test results as the environmental feedback, and then generates a corrected version.

[2] https://serpapi.com/

Model	QA Tool	#Train Data (Self/GPT-4 Generated)	EM
Llama-2-13B Agents			
Few-Shot	WikipediaAPI	-	20.0
Self-Training	WikipediaAPI	2k/0	27.6
Re-ReST	WikipediaAPI	2.5k/0	29.6
Llama-2-13B Agents w/ GPT-4-Generated Data			
FireAct (Chen et al., 2023)	SerpAPI	0/0.5k	34.4
LUMOS (Yin et al., 2024)	GPT-3.5	0/20k	31.4
LUMOS (Yin et al., 2024)	GPT-4	0/20k	36.3
FireAct	WikipediaAPI	0/0.5k	32.2
Self-Training	WikipediaAPI	2.5k/0.5k	34.2
Re-ReST	WikipediaAPI	3k/0.5k	35.8
Llama-3-8B Agents			
Few-Shot	WikipediaAPI	-	30.0
Self-Training	WikipediaAPI	2.4k/0	34.4
Re-ReST	WikipediaAPI	3k/0	36.8

Table 1: On HotpotQA, our method enables a better usage of the training data compared with self-training and improves self-training for LLama-2/3-based agents. Also, adding only 0.5k GPT-generated data enables our agents with the free Wikipedia API to achieve comparable or better performance than methods with commercial APIs.

Model	Sample Acc.	Success Rate
Few-Shot	-	8.9
Self-Training	11.2	37.3
Re-ReST	48.0	51.4

Table 2: Results on the ALFWorld dataset. Re-ReST substantially increases the sampling accuracy and outperforms self-training in terms of success rate even upon employing a reflector.

Model	MBPP		GQA	
	Sample Acc.	P@1	Sample Acc.	Score
Zero-Shot	-	48.6	-	40.9
Self-Training	66.9	54.5	44.7	41.9
Re-ReST	77.3	56.4	55.7	42.6

Table 3: Re-ReST improves self-training on code generation and visual programming tasks.

For GQA, following Surís et al. (2023), we build the agent by providing a pre-defined set of visual APIs (e.g. object detection) and prompt the model to generate code using the APIs.

Training and Evaluation Setup. For MBPP, we use zero-shot and three-shot prompting for zero-shot agent and reflector data generation. For GQA, we follow the prompt in Surís et al. (2023) for the model for sample generation. For both datasets, the agent model samples 3 generations per training instance as before. We do not use the provided ground truths for MBPP training for consistency with the other experimental settings. The agent and reflector models are trained for 3 epochs with a learning rate of 3e-4 using LoRA.

Results. As in Table 3, for MBPP, because CodeLlama is trained on a large amount of code generation corpus, the base CodeLlama model can achieve a decent performance without any fine-tuning. The high pass rate results in many of the training instances being used for self-training. After self-training on the MBPP training data, the model performance can be improved from 48.6% to 54.5%. The reflector model can generate more self-training data and the pass rate can be improved with the reflector-generated data. For GQA, similar improvements can be seen, indicating that our method is also applicable in visual programming.

3.4 Text-to-Image Generation

Dataset. We also conduct experiments in text-to-image generation. Specifically, we use the dataset constructed by Cho et al. (2023). Their dataset evaluates the model's generated images in multiple dimensions and has training data for the spatial,

Model	Sample Acc.	VPEval Skill Score Count	Spatial	Scale
VPGen	-	72.2	56.1	26.3
VPGen w/ Self-Training	57.6	74.7	54.5	29.3
VPGen w/ Re-ReST	67.6	75.0	58.2	30.1

Table 4: Re-ReST can outperform self-training in text-to-image generation when applied to VPGen and evaluated with VPEval (Cho et al., 2023) on multiple dimensions.

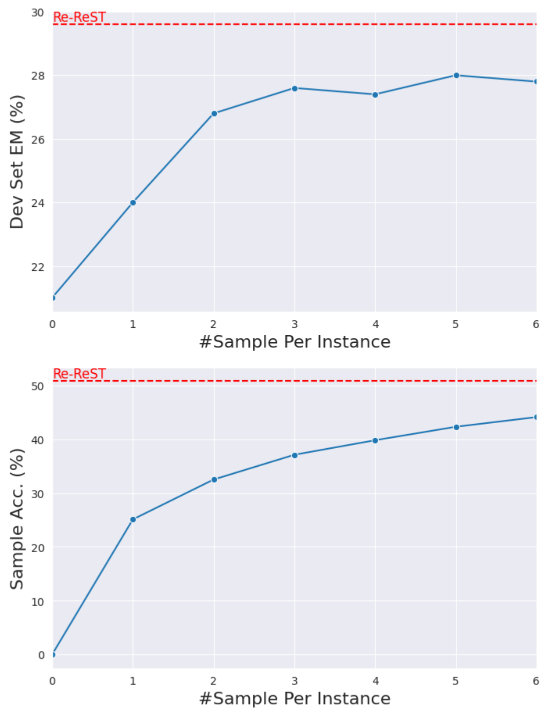

Figure 3: In self-training, increasing the number of generations per instance initially improves model performance, but this effect plateaus. Additionally, both model performance and the number of solved training instances are lower than with Re-ReST, indicating our reflector can efficiently and effectively generate high-quality self-training data.

scale, and count dimensions. For each dimension, the evaluation set consists of 1,000 instances. The training dataset consists of 36,920/18,200/1,560 instances for the spatial/scale/count dimensions.

Model Setup. We use VPGen in Cho et al. (2023) as our base model, which is based on Vicuna-13B (Chiang et al., 2023) and is finetuned for text-to-layout generation on multiple constructed image-text datasets. The generated layouts are fed into an external model (i.e., GLIGEN (Li et al., 2023b)) for image generation. We build both the agent and reflector upon the VPGen model.

Training and Evaluation Setup. We use VPGen to perform inference on their training data, and evaluate the generations using VPEval (Cho et al., 2023). Specifically, during evaluation, a visual question answering model (BLIP-2 (Li et al., 2023a)) is used to determine if the generated images correctly capture the input text information. The BLIP-2 generated results are treated as the environmental feedback for the reflector. We do not use zero-shot reflection results to train the reflector because LLMs cannot perform this task without finetuning. The agent and reflector are trained for 2 epochs with a learning rate of 1e-5 using LoRA.

Model	Sample Acc.	EM
Self-Training	37.1	27.6
Re-ReST w/o Ref. Train.	43.7	28.8
Re-ReST	50.8	29.6

Table 5: While directly using a pretrained LLM as our reflector improves self-training, training the reflector specifically for self-correction further improves the agent performance.

Model	EM
Base	27.6
S.C. (6 agents)	30.8
S.C. (3 agents + 3 reflectors)	32.0
Oracle (3 agents + 3 reflectors)	36.8

Table 6: Previous work relies on ground-truth feedback for test-time reflection (Oracle). In contrast, we propose to use self-consistency (Wang et al., 2023a) to enable our reflector to be applied during inference without ground-truth feedback and achieve improvements, demonstrating the potential of applying our method during the test time.

Results. As shown in Table 4, our method continues showing improvements over baselines in the text-to-image generation task. The baseline VPGen model's performance is enhanced when self-training is applied, further improved significantly with our Re-ReST method across all the dimensions. The results demonstrate promising applications of our model in the multimodal generation domain with a language agent as a backend.

Model	HotpotQA EM	MBPP Pass@1	VPEval Score Count	Spatial	Scale
Self-Training	27.6	54.5	74.7	54.5	29.3
Re-ReST	29.6	56.4	75.0	58.2	30.1
Self-Training w/ DPO	28.0	54.9	74.6	56.7	30.0
Re-ReST w/ DPO	31.0	56.4	75.4	58.5	31.0

Table 7: Our method is compatible with direct preference optimization (DPO) (Rafailov et al., 2023), and integrating DPO into our method can generally improve the model performance.

3.5 Analysis

Re-ReST v.s. Self-Training with More Samples. We investigate if we can simply sample more generations from the language agent for self-training and achieve comparable performance with our reflector-augmented method. Specifically, we try to sample k generations for each instance, where k is set to $1, 2, 3, 4, 5, 6$, and use the generated samples for self-training. As shown in Figure 3, if we keep sampling more generations from the language agent, the agent can indeed solve more instances and we can obtain an increasing amount of data for self-training. However, 1) the number of solved instances is still lower than the number of reflector-solved instances, demonstrating that the reflector can find the correct solutions more efficiently than sampling; 2) the model performance is not always improved with more training data and it cannot outperform our method even when trained with more generated samples, indicating that the quality of the self-training data is also important and our reflector can generate training data effectively for the agent.

Effect of Training the Reflector. As illustrated, we propose to first train the reflector before using it to generate the self-training data. In this part, we investigate if we can use the reflector to perform self-correction in a zero-shot manner and then train the language agent. As in Table 5, we find that while the reflector can perform self-correction without any finetuning and improve the performance of the language agent, further improvements can be made if we specifically train the model for self-correction, demonstrating the effectiveness of our proposed reflector training strategy.

Test-Time Reflection without Ground-Truth Feedback. Previously, our reflector functions only during training and is not used during inference because it is often impossible to obtain ground-truth feedback, which is required for reflection methods to work (Huang et al., 2024). In this section, we propose employing self-consistency (Wang et al., 2023a) to enable test-time reflection and address this limitation. Self-consistency is a decoding technique that combines multiple model predictions by sampling various reasoning paths and then selecting the most consistent answer through a majority vote. This approach allows us to apply the reflector during inference. Specifically, we sample multiple answers from our model and perform reflection on each output, regardless of correctness. We then aggregate all the answers using self-consistency. As in Table 6, integrating our reflector with self-consistency (3 agent samples and 3 reflection samples) achieves improvements over baseline (self-consistency with 6 model samples). This demonstrates the potential application of our method during inference, overcoming the current limitation of requiring ground-truth feedback for reflection methods.

Re-ReST with Direct Preference Optimization. Our reflector turns incorrect samples into correct ones, naturally making negative-positive pairs suitable for preference optimization objectives such as DPO. In this part, we investigate the application of DPO in our method. As in Table 7, integrating DPO into our method can generally improve or achieve comparable performance with training models only with supervised training on positive samples, indicating our compatibility with DPO.

4 Conclusion

Our study studies the applications of self-training in language agents and improves it with Reflection-Reinforced Self-Training (Re-ReST), an approach that efficiently obtains high-quality samples for self-training with a reflector. Our experiments demonstrate that Re-ReST outperforms self-training methods across various tasks, confirming the efficiency and effectiveness of incorporating a reflection mechanism. Within the proposed framework, in the future, we can improve the reflection mechanism and develop better training paradigms for the agent and reflector.

Limitations

Our approach is predicated on the availability of ground-truth feedback during the training process. While this assumption holds true for many language agent tasks, it presents challenges when applied to broader contexts. Specifically, acquiring accurate ground-truth feedback can be difficult in diverse, real-world scenarios. This limitation underscores a key aspect of our study: it is primarily concentrated on language agent tasks, thereby neglecting the potential applications and implications within the broader scope of general language modeling. This suggests the need for future research to explore and address the complexities of applying our methods to general language modeling tasks, where ground-truth feedback may not be as readily accessible or reliable. Another potential risk of the method is that through self-training, the biases encoded in LLMs can be amplified, and careful calibrations should be conducted before the deployment of our method.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 technical report. *arXiv preprint*.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint*.

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

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Jaemin Cho, Abhay Zala, and Mohit Bansal. 2023. Visual programming for text-to-image generation and evaluation. *NeurIPS*.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, SHUM KaShun, and Tong Zhang. 2023. RAFT: Reward ranked finetuning for generative foundation model alignment. *TMLR*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. *ICLR*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. 2023. Reinforced Self-Training (ReST) for language modeling. *arXiv preprint*.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2019. Revisiting self-training for neural sequence generation. In *ICLR*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *EMNLP*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. *ICLR*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*.

Drew A. Hudson and Christopher D. Manning. 2019. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*.

Jiaming Ji, Boyuan Chen, Hantao Lou, Donghai Hong, Borong Zhang, Xuehai Pan, Juntao Dai, and Yaodong Yang. 2024. Aligner: Achieving efficient alignment through weak-to-strong correction. *arXiv preprint*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *NeurIPS*.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023a. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. 2024. Self-alignment with instruction backtranslation. In *ICLR*.

Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. 2023b. GLIGEN: Open-set grounded text-to-image generation. In *CVPR*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *NeurIPS*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *NeurIPS*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2022. Show your work: Scratchpads for intermediate computation with language models. In *Deep Learning for Code Workshop*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *NeurIPS*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code Llama: Open foundation models for code. *arXiv preprint*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. ToolFormer: Language models can teach themselves to use tools. *NeurIPS*.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. AlfWorld: Aligning text and embodied environments for interactive learning. *ICLR*.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. ViperGPT: Visual inference via python execution for reasoning. In *ICCV*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023a. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *ACL*.

Ziqi Wang, Le Hou, Tianjian Lu, Yuexin Wu, Yunxuan Li, Hongkun Yu, and Heng Ji. 2023c. Enable language models to implicitly learn self-improvement from data. *arXiv preprint*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*.

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*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. In *ICLR*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. LUMOS: Learning agents with unified data, modular design, and open-source llms. *ACL*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. *NeurIPS*.

A Related Work

In this section, we first overview the research progress in language agents, then briefly describe self-training and self-correction methods for improving language agents. We also summarize the major differences between our work and previous language agent methods in Table 8.

Language Agents. Language agents refer to language models that interact with the world in general. It has been demonstrated that LLMs can perform actions by generating specific commands (Nakano et al., 2021; Huang et al., 2022; Ahn et al., 2022) and calling external tool APIs (Lu et al., 2023; Schick et al., 2023; Gou et al., 2024). By integrating the model reasoning and acting abilities, ReAct (Yao et al., 2022) asks an LLM to first generate reasoning traces and then act accordingly, which is then improved by follow-up works through inference-time techniques such as reflection (Shinn et al., 2023) and planning (Yao et al., 2023). Recently, finetuning agents (Chen et al., 2023; Yin et al., 2024) have attracted attention from the research community. However, most of the existing works attempt to distill knowledge from a relatively strong LLM (e.g., GPT-4) to a weaker LLM (e.g., LLaMa-2). By contrast, our work bootstraps a language agent's performance by utilizing its own reflective ability without using external models.

Self-Training for Language Models. Various self-training algorithms have been proposed to improve language models (He et al., 2019; Huang et al., 2023; Dong et al., 2023; Gulcehre et al., 2023; Yuan et al., 2024), with the general idea being to improve models with self-generated samples in an unsupervised or semi-supervised manner. He et al. (2019) is one early work in applying self-training to generative language models and points out the importance of introducing noises during pseudo-label generation to increase the sample diversity. In the large language model era, Gulcehre et al. (2023) propose Reinforced Self-Training (ReST), where they use a scoring function to select self-generated samples and augment the training data. Similarly, Yuan et al. (2024) proposes self-rewarding that scores samples with the LLM itself and trains the model with direct preference optimization (DPO) (Rafailov et al., 2023) on the scored samples. Self-training has also been employed to improve the chain-of-thought reasoning (Nye et al., 2022; Wei et al., 2022) ability of LLMs (Uesato et al., 2022). For example, Zelikman et al. (2022) propose to ask an LLM to generate rationales given questions and improve the LLM with its own generated reasoning. Re-ReST falls under the self-training paradigm, and different from previous work, our aim is to generate useful samples efficiently for self-training.

Self-Reflection/Self-Correction for Language Models. Several works have used LLMs to reflect on their generations with internal or external feedback and correct their errors (Welleck et al., 2023; Wang et al., 2023c; Shinn et al., 2023; Madaan et al., 2023; Kim et al., 2024; Ji et al., 2024). A majority of this line of research is focused on improving LLMs during inference. For example, Self-Refine (Madaan et al., 2023) proposes to have LLMs iteratively evaluate their generations, based on which they improve their generations. Similarly, Shinn et al. (2023) use LLM agents to reflect on its generations and their environment feedback, then guide the next generation with the generated verbal feedback. As pointed out by Huang et al. (2024), high-quality external feedback is essential for these self-correction models, without which existing techniques actually decrease model performance. However, such high-quality feedback is often unavailable during the test time, thus we propose to use Re-ReST only during training and perform corrections with oracle feedback from environments, ensuring its effectiveness in correcting the model generations. In addition, during the test time, the corrected generations are distilled into the language model, thus directly generating the answer without introducing overhead during inference.

B Prompts

Work	Agent Training		Agent Reflection	
	Finetuning	GPT-Free	w/ G.T. Feedback	w/o G.T. Feedback
FireAct (Chen et al., 2023)	✓	✗	✗	✗
LUMOS (Yin et al., 2024)	✓	✗	✗	✗
Reflexion (Shinn et al., 2023)	✗	✗	✓	✗
Self-Refine (Madaan et al., 2023)	✗	✗	✓	✗
Re-ReST	✓	✓	✓	✓

Table 8: Comparisons with previous language agent methods. We propose to finetune LLMs for language agent tasks with self-generated data, while previous work such as FireAct and LUMOS rely on stronger LLMs such as GPT-4 to perform knowledge distillation. In addition, we propose to use the agent's reflection ability to improve the self-training efficiency, where the reflection can function both with and without ground-truth feedback, addressing the limitation of previous agent reflection methods (Shinn et al., 2023; Madaan et al., 2023; Huang et al., 2024).

Prompt
Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be three types:
(1) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search.
(2) Lookup[keyword], which returns the next sentence containing keyword in the last passage successfully found by Search.
(3) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.

{In-context examples}
Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?
Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.
Action 1: Search[Colorado orogeny]
Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.
Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.
Action 2: Lookup[eastern sector]
Observation 2: (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.
Thought 3: The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.
Action 3: Search[High Plains]
Observation 3: High Plains refers to one of two distinct land regions:
Thought 4: I need to instead search High Plains (United States).
Action 4: Search[High Plains (United States)]
Observation 4: The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m).[3]
Thought 5: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.
Action 5: Finish[1,800 to 7,000 ft]
...

User
Question: {Input question}

Table 9: Prompt template for the HotpotQA agent. {In-context examples} {Input question}

Prompt
Solve a question answering task with a reflection on a previous trial and interleaving Thought, Action, Observation steps. Reflection can reason on a previous trial that failed and develop a new strategy to avoid repeating the same mistake. Thought can reason about the current situation, and Action can be three types:
(1) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search.
(2) Lookup[keyword], which returns the next sentence containing keyword in the last passage successfully found by Search.
(3) Finish[answer], which returns the answer and finishes the task.
You need to reflect on the previous trial and then restart a new trial.

{In-context examples}
Question: When was the subject of the book The Deliberate Stranger executed?
Previous Trial:
Thought 1: I need to search The Deliberate Stranger and find when the subject was executed.
Action 1: Search[The Deliberate Stranger]
Observation 1: The Deliberate Stranger is a book about American serial killer Ted Bundy written by Seattle Times reporter Richard W. Larsen that was published in 1980. The book spawned a television miniseries of the same title, starring Mark Harmon as Bundy, that aired on NBC on May 4, 1986.
Thought 2: The subject of the book, Ted Bundy, was executed in 1989.
Action 2: Finish[1989] Reflection: I should have searched for Ted Bundy first before I finished answering the question.
Thought 1: I need to first search The Deliberate Stranger and find its subject, and then find out which year the subject was executed.
Action 1: Search[The Deliberate Stranger]
Observation 1: The Deliberate Stranger is a book about American serial killer Ted Bundy written by Seattle Times reporter Richard W. Larsen that was published in 1980. The book spawned a television miniseries of the same title, starring Mark Harmon as Bundy, that aired on NBC on May 4, 1986.
Thought 2: The subject of The Deliberate Stranger is Ted Bundy. I need to search for Ted Bundy and find out which year he was executed.
Action 2: Search[Ted Bundy]
Observation 2: Could not find [Ted Bundy]. Similar: ['Ted Bundy', 'Ted Bundy (film)', 'Ted Bundy: American Boogeyman', 'Conversations with a Killer: The Ted Bundy Tapes', 'Murder of Rita Curran', 'Ted Bundy: Falling for a Killer', 'Extremely Wicked, Shockingly Evil and Vile', 'Disappearance of Ann Marie Burr', 'Murder of Georgann Hawkins', 'Zac Efron']
Thought 3: I need to change the keyword for search.
Action 3: Search[Ted Bundy executed]
Observation 3: Theodore Robert Bundy (Cowell; November 24, 1946 January 24, 1989) was an American serial killer who kidnapped, raped and murdered dozens of young women and girls during the 1970s and possibly earlier. After more than a decade of denials, he confessed to 30 murders committed in seven states between 1974 and 1978. His true victim total is unknown.Bundy often employed charm to disguise his murderous intent when kidnapping victims, and extended this tactic vis-vis law enforcement, the media and the criminal justice system to maintain his claims of innocence...
Thought 4: Ted Bundy was executed on January 24, 1989.
Action 4: Finish[January 24, 1989]
...

User
Question: {Input question and previous trial}

Table 10: Prompt template for the HotpotQA reflector. {In-context examples} {Input question and previous trial}

Prompt
You are an expert Python programmer, and here is your task: {input task}
Your code should pass these tests:
{unit tests}
Your code should start with a [PYTHON] tag and end with a [/PYTHON] tag.

Table 11: Prompt template for the MBPP agent. {unit tests} {input task}

Prompt
You are an AI Python assistant. You will be given the user input, your past incorrect function implementation, and a series of unit tests. Write your reflection on the function implementation and correct your implementation (copy the function signature and its docstring).

{In-context examples}
[previous impl]:
“‘python
def add(a: int, b: int):
"""
Given integers a and b, return the total value of a and b.
"""
return a - b
“‘
[unit test results from previous impl]:
Tested passed:
Tests failed:
assert add(1, 2) == 3 # output: -1
assert add(1, 2) == 4 # output: -1
[reflection on previous impl]:
The implementation failed the test cases where the input integers are 1 and 2. The issue arises because the code does not add the two integers together, but instead subtracts the second integer from the first. To fix this issue, we should change the operator from ‘-‘ to ‘+‘ in the return statement. This will ensure that the function returns the correct output for the given input.
[improved impl]:
“‘python
def add(a: int, b: int):
"""
Given integers a and b, return the total value of a and b.
"""
return a + b
“‘”’
...

User
{Input task and previous trial}

Table 12: Prompt template for the MBPP reflector. {In-context examples} {Input task and previous trial}

Prompt

```
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Methods
    ——-
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the image matching the object_name.
    simple_query(question: str=None)->str
        Returns the answer to a basic question asked about the image. If no question is provided, returns the answer to "What is this?".
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    best_text_match(string1: str, string2: str)->str
        Returns the string that best matches the image.     crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """
    {Detailed API definition}

{In-context examples}

{Input question}
```

Table 13: Prompt template for the GQA agent. Full prompt is released in https://github.com/cvlab-columbia/viper/blob/main/prompts/benchmarks/gqa.prompt. {Detailed API definition} {In-context examples} {Input question}

Prompt
I am writing code to handle visual question answering tasks by calling computer vision APIs. My code is wrong, and I hope you can help correct it.

{Input question and previous trial}

Your response should start with your reasoning and analysis. Then, you should write the correct code wrapped in ``` python and ```. The correct code should be a function with signature `def execute_command(image) -> str:`

—

Below are the available APIs and some example usages:
```python
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Methods
    ——-
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the image matching the object_name.
    simple_query(question: str=None)->str
        Returns the answer to a basic question asked about the image. If no question is provided, returns the answer to "What is this?".
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    best_text_match(string1: str, string2: str)->str
        Returns the string that best matches the image.    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """
    {Detailed API definition}

{In-context examples}
```

Table 14: Prompt template for the GQA reflector. {Detailed API definition} {In-context examples} {Input question and previous trial}

Prompt
Interact with a household to solve a task. Here is an example.

{In-context examples}
You are in the middle of a room. Looking quickly around you, you see a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a countertop 1, a garbagecan 1, a handtowelholder 2, a handtowelholder 1, a sinkbasin 2, a sinkbasin 1, a toilet 1, a toiletpaperhanger 1, and a towelholder 1.
Your task is to: put some spraybottle on toilet.
> think: To solve the task, I need to find and take a sparybottle, then put it on toilet.
OK.
> think: First I need to find a spraybottle. A spraybottle is more likely to appear in cabinet (1-4), countertop (1), toilet (1), sinkbasin (1-2), garbagecan (1). I can check one by one, starting with cabinet 1.
OK.
> go to cabinet 1
On the cabinet 1, you see a cloth 1, a soapbar 1, a soapbottle 1.
> go to cabinet 2
The cabinet 2 is closed.
> open cabinet 2
You open the cabinet 2. The cabinet 2 is open. In it, you see a candle 1, and a spraybottle 2.
> think: Now I find a spraybottle (2). Next, I need to take it.
OK
> take spraybottle 2 from cabinet 2
You pick up the spraybottle 2 from the cabinet 2.
> think: Now I take a spraybottle (2). Next, I need to put it in/on toilet 1.
OK.
> go to toilet 1
On the toilet 1, you see a soapbottle 2.
> put spraybottle 2 in/on toilet 1
You put the spraybottle 2 in/on the toilet 1.

User
{Input question} Here is the task: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: examine the bowl with the desklamp.
{Reflection Results} think: I was stuck in a loop in which I continually picked up the alarmclock 1 instead of turning on the desklamp.

Table 15: Example Prompt Template on the ALFWorld dataset. A prompt includes (a) {In-context example} which is a complete trajectory from a successful trial. (b) {Input question} describes the initial environment and the instruction of the task, and (c) {Reflection Results} encapsulates the self-reflection results from the reflector model.