Reflect-RL: Two-Player Online RL Fine-Tuning for LMs

Runlong Zhou* and **Simon S. Du†** and **Beibin Li‡**

Abstract

As language models (LMs) demonstrate their capabilities in various fields, their application to tasks requiring multi-round interactions has become increasingly popular. These tasks usually have complex dynamics, so supervised fine-tuning (SFT) on a limited offline dataset does not yield good performance. However, only a few works attempted to directly train the LMs within interactive decision-making environments. We aim to create an effective approach to fine-tune LMs with online reinforcement learning (RL) in these environments. We propose `Reflect-RL`, a two-player system to fine-tune an LM using SFT and online RL, where a frozen reflection model (player) assists the policy model (player). To generate data for the warm-up SFT stage, we use negative example generation to enhance the error-correction ability of the reflection model. Furthermore, we designed single-prompt action enumeration and applied curriculum learning to allow the policy model to learn more efficiently. Empirically, we verify that `Reflect-RL` outperforms SFT and online RL without reflection. Testing results indicate GPT-2 XL 1.56B fine-tuned with `Reflect-RL` outperforms larger open-source LMs, such as Mistral 7B. The benchmarks, dataset, and code involved in this work are publicly available.[1]

1 Introduction

Large language models (LLMs) have shown promising results in problem-solving, coding, and document retrieval (Mialon et al., 2023). While performing these tasks, LLMs exhibit considerable reasoning, planning, and reflection skills, enabled by prompting techniques like ReAct (Yao et al., 2022), Reflexion (Shinn et al., 2023), Chain of Thought (CoT, Wei et al. (2023)), Tree of Thoughts (ToT, Yao et al. (2023a)), and reasoning via planning (Hao et al., 2023). Some recent studies (Magister et al., 2023; Mukherjee et al., 2023; Mitra et al., 2023) also try to improve reasoning capabilities of smaller models to match those of advanced LLMs.

The reasoning and reflection skills enable LLMs to act as agents and interact with real-world environments (Durante et al., 2024; Cheng et al., 2023), including code interpreters, embodied robotics (Shridhar et al., 2021; Ahn et al., 2022; Tan et al., 2024), games (Park et al., 2023), and many other spaces (Vezhnevets et al., 2023). This interaction ability is closely tied to reinforcement learning (RL), where agents can learn optimal behaviors through trial and error within an environment.

1.1 Motivations

This research is motivated by three distinct application domains within the same system, which include: document querying (Izacard et al., 2022), database searching (Floratou et al., 2024), and coding (Chen et al., 2021). In these applications, a chatbot needs to navigate in a file system to read documents, modify files, and execute code to answer users' questions. Central to these tasks is the chatbot's ability to *autonomously explore* within a repository using system commands, such as, `ls`, `cd src/`, `cat main.py`, similar to the paradigm in Yang et al. (2023).

Interactive chatbot for file systems (NVIDIA, 2024), multi-agent frameworks (Wu et al., 2023), tool selection (Karpas et al., 2022; Patil et al., 2023), and many other industrial applications require interactive decision-making capabilities. Even if LLMs can perform these tasks, they are usually trained heavily with offline supervised learning rather than with online training within complex environments. Moreover, some recent studies have found that LLMs might not be able to correct themselves without external feedback during interactions (Huang et al., 2023). On the other hand, online RL training could enable LMs to dynamically adapt and make informed decisions beyond static

*University of Washington. Part of this work done when Runlong was an intern at Microsoft Research, Redmond. Email: `vectorzh@cs.washington.edu`

†University of Washington. Email: `ssdu@cs.washington.edu`

‡Microsoft Research, Redmond. Email: `beibin.li@microsoft.com`

[1] https://github.com/zhourunlong/Reflect-RL

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 995–1015
August 11-16, 2024 ©2024 Association for Computational Linguistics

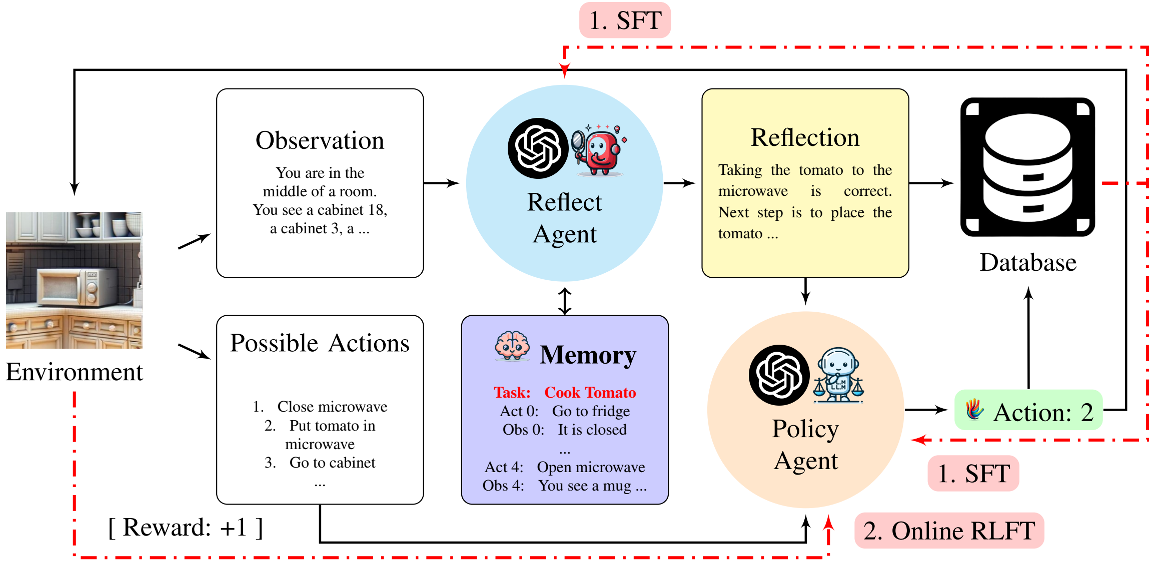

Figure 1: `Reflect-RL` Pipeline. Solid lines represent the forward pass for both data generation and inference. Agents (in circular nodes) are language models capable of generating reflections and making decisions. Red dashed lines represent the loss and gradient calculation during the training periods: the reflection agent is trained with SFT, while the policy agent is trained first with SFT and then with online RLFT. Detailed illustrations for each stage can be found in Appendix B.

datasets.

Some recent studies have incorporated RL to align LMs with human preference and to prompt LM for problem-solving (see Table 1 for details). Szot et al. (2023) and Tan et al. (2024) have started contemporary explorations to integrate LMs within interactive RL environments, but these pioneering studies have not fully utilized the LMs' reasoning capabilities. Motivated by the strength of RL and expansiveness of LLMs, our work aims to fine-tune smaller, faster, and more secure locally-operated LMs that are capable of decision-making and adaptation through *reflection*, which are essential for domain-specific interactive tasks.

1.2 Contributions

In this work, we introduce `Reflect-RL`, **a novel approach to dynamically improve LMs with online RL** (Figure 1), applied with Markov decision processes (MDPs) for *multi-step* decision making. Most of the previous RL-LM works can be categorized into three classes (Table 1): ① treating token-generation as RL, rather than considering embodied tasks, games, or interactive decision making within environments; ② using LMs as agents to augment policy generation with additional textual information, without directly learning from the environment (gradient-free); ③ engaging primarily with single-step bandits rather than multi-step MDPs. Our method seeks to improve multi-step decision making in textual environments by integrating techniques from RL and LMs, enabling LMs to adapt more efficiently to complex environments. We summarize our *key techniques* below.

Key Techniques:

• **Reflection (Section 4.1.3).** We distill reflection abilities for our domain-specific environment from GPT-4 (OpenAI, 2023) through supervised learning. The distilled small LM is frozen and deployed as a reflection model (player) to assist the trainable policy model (player) in decision-making. Reflection accelerates training convergence and improves test performance.

• **Negative example generation (Section 4.2).** The reflection data gathered from GPT-4 is unbalanced, with the majority consisting of positive (near-optimal) decisions. To balance the dataset, we generate negative examples by perturbing the GPT-4 trajectories and optimal trajectories. Negative examples enhance the quality of reflection, ultimately leading to better success rates in the benchmarks.

• **Single-prompt action enumeration (Section 4.3).** We incorporate all possible valid actions into a single prompt, allowing the LM to select the appropriate option using only one token. This approach improves upon the normalization techniques in previous works to generate valid actions and also reduces time complexity.

• **Task-specific curriculum learning (Section 4.4).** The core challenges of RL include planning for a long horizon and sparse reward signals. Vanilla policy optimization methods often fail to obtain sufficient useful trajectories efficiently. We incorporate the idea of curriculum learning into our pipeline, designing a specific curriculum to guide training by giving extra rewards or scheduling the data order.

New Benchmark for Online RL Fine-Tuning. Additionally, we introduce `AutoExplore`, *a benchmark inspired by industrial applications*, along with other benchmarks adapted from previous works. These benchmarks are suitable for both *research* and *application* purposes. They can be integrated with either local LMs for training or remote LLMs for in-context inference. Our demonstrations show positive results of LLMs on industrial applications. Both RL training and data generation are made easy by their use.

Paper Overview. This paper begins by discussing LLMs in Section 2 and RL preliminaries in Section 3. Then, we introduce our proposed `Reflect-RL` in Section 4 and benchmarks in Section 5. The results are presented in Section 6. Finally, we discuss the findings and future directions in Section 7.

2 Related Works

Language models (LMs). LMs play a pivotal role in tasks such as sentiment analysis (Zhong et al., 2023; Wang et al., 2023b), machine translation (Gulcehre et al., 2017; Lample and Conneau, 2019), and automated text generation (Chen et al., 2020; Dathathri et al., 2020), showcasing their versatility and capability in handling complex linguistic structures.

LM agents and multi-agent collaborations. Autonomous LM agents (Bran et al., 2023; Park et al., 2023; Wu et al., 2023; Wang et al., 2023a) underscore LMs' capabilities of autonomous and collaborative problem-solving. Such agent collaboration can achieve a level of sophistication and efficiency that is difficult to obtain through solo efforts.

Fine-tuning of LMs. Supervised fine-tuning (SFT, Howard and Ruder (2018); Radford et al. (2019)) and reinforcement learning from human feedback (RLHF) are the most commonly used alignment methods for adapting pre-trained LMs to specific tasks. Additionally, LoRA (Hu et al., 2021), QLoRA (Dettmers et al., 2023), and other parameter-efficient fine-tuning (PEFT) algorithms can facilitate this process.

LMs for interactive decision-making. As summarized in Table 1 and discussed in Section 1.2, only a few studies have applied online RL to LMs for making multi-step decisions. Szot et al. (2023) and Tan et al. (2024) are the two most relevant studies.

3 Preliminaries

Notations. For any set $\mathcal{X}$, we use $\Delta(\mathcal{X})$ to denote the probability simplex over $\mathcal{X}$. Let the tokenizer be fixed throughout the paper. For a string s, we use $|s|$ to denote the number of tokens in s after using this fixed tokenizer.

Markov decision processes (MDPs). Reinforcement learning (RL, Sutton and Barto (1998)) problems are usually formulated as MDPs. They enable agents to learn optimal behaviors through interacting with the environment, without human intervention or labeling. A (finite-horizon) MDP can be described as $\mathcal{M} = (H, \mathcal{S}, \mathcal{A}, \mu, \mathcal{T}, r)$, where H is the planning horizon, $\mathcal{S}$ is the state space, and $\mathcal{A}$ is the action space. $\mu \in \Delta(\mathcal{S})$ is the initial state distribution, which can represent a distribution over tasks. We study *deterministic environments* in this work as the tasks in our motivations are deterministic. The transition function maps a state-action pair to a state $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and the reward function immediately yields a reward $r : \mathcal{S} \times \mathcal{A} \to [-1, 1]$. Given a (Markovian) policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, we define its value function and Q-function as

$$V_h^\pi(s) := \mathbb{E}_\pi\left[\sum_{t=h}^{H} r_t \,\middle|\, s_h = s\right],$$

$$Q_h^\pi(s,a) := \mathbb{E}_\pi\left[\sum_{t=h}^{H} r_t \,\middle|\, (s_h, a_h) = (s,a)\right].$$

The expected return of a policy π is $J^\pi := \mathbb{E}_{s_1 \sim \mu}[V_1^\pi(s_1)]$, and the goal of RL is to find the optimal policy maximizing J^π.

When modeling an application as an MDP, we may encounter the fact that each state s has a separate "valid" action space $\mathcal{A}(s)$. Though we can define $\mathcal{A} = \cup_{s \in \mathcal{S}} \mathcal{A}(s)$, the union could be intractably large. A viable workaround is to define a mapping function f_s at each state, such that $\mathcal{A}(s) \subseteq \{f_s(a) \mid a \in \mathcal{A}\}$. This formulation works smoothly with our approach named "single-prompt action enumeration" (Section 4.3) where $\mathcal{A}$ consists of choices such as 0, 1, 2, $\ldots$, and $f_s(a)$ maps them to detailed actions.

Policy optimization for MDPs. Policy optimization is an approach to solve MDPs using parameterized policies. Policy optimization techniques for MDPs surround the class of policy gradient (PG, or REINFORCE algorithm, Sutton et al. (1999)) methods, which directly adjust the parameters of the policy in a way that maximizes J^π. Let π_θ be a policy parameterized by θ, then the policy gradient is computed as

$$\nabla_\theta J^{\pi_\theta} = \sum_{h=1}^{H} \mathbb{E}_{s,a \sim d_h^{\pi_\theta}}\left[Q_h^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s)\right].$$

Here $d_h^{\pi_\theta}$ is the distribution of (s,a) pairs at step h under policy π_θ. An update step using policy gradient is $\theta_{t+1} = \theta_t + \eta \nabla_\theta J^{\pi_{\theta_t}}$.

<table>
<tr><th>Category</th><th>Works</th><th>Direct Interaction</th><th>Bandit or MDP</th><th>Reflection</th><th>Training Method</th></tr>
<tr><td>Token-generation as RL</td><td>Lu et al. (2022), Ramamurthy et al. (2023), Luong et al. (2024), Yuan et al. (2024)</td><td>Yes</td><td>MDP</td><td>No</td><td>RL</td></tr>
<tr><td rowspan="3">LMs as agents</td><td>Park et al. (2023), Zhang et al. (2023), Shinn et al. (2023)</td><td>Yes</td><td rowspan="3">MDP</td><td rowspan="2">Yes</td><td rowspan="3">None</td></tr>
<tr><td>Huang et al. (2022), Yao et al. (2022), Yao et al. (2023b), Du et al. (2023)</td><td rowspan="2">No</td></tr>
<tr><td>Ahn et al. (2022)</td><td>No</td></tr>
<tr><td>RLHF</td><td>Ziegler et al. (2020), Stiennon et al. (2022), Bai et al. (2022), Ouyang et al. (2022)</td><td>Yes</td><td>Bandit</td><td>No</td><td>RL</td></tr>
<tr><td>SFT</td><td>Shridhar et al. (2021)</td><td>Yes</td><td>MDP</td><td>No</td><td>Supervised</td></tr>
<tr><td rowspan="2">RL Fine-tuning</td><td>Szot et al. (2023), Tan et al. (2024)</td><td rowspan="2">Yes</td><td rowspan="2">MDP</td><td>No</td><td rowspan="2">RL</td></tr>
<tr><td>This work</td><td>Yes</td></tr>
</table>

Table 1: Comparison between works involving LMs and RL. "Direct interaction" indicates whether the LM plays the role as the policy model directly interacting with the environment, so a "No" means it plays indirectly by assisting another non-language policy model. "Bandit or MDP" indicates whether the environment is a single-step bandit or a multi-step MDP. "Reflection" indicates whether this work elicits the reasoning ability of the language model to generate reflections and help with planning in RL. "Training method" indicates whether the LM is being trained and if yes, the method.

Proximal Policy Optimization (PPO, Schulman et al. (2017)) is another exemplary method applied in this field, whose details are deferred to Appendix A.

4 Methodology

4.1 `Reflect-RL`

Here, we propose `Reflect-RL`, an online reinforcement learning fine-tuning method for LMs in MDPs.

4.1.1 LM as an RL policy

We use a language model as an RL policy $\pi_\theta(a|s)$ where $s = (s_1, s_2, \ldots, s_L) \in \mathcal{S}$ is the current state (represented by tokens) and $a = (a_1, a_2, \ldots, a_K) \in \mathcal{A}(s)$ is the generated token sequence (also represented by tokens). Let $a_{:k}$ denote the subsequence $(a_1, a_2, \ldots, a_k)$. We apply policy model to multi-step RL tasks, where the language model reads s in the input prompt, and then generate a in the completion.

In environments where states are not represented in natural languages, we need a function $p(s)$ to convert the original state s to make it a legal input for an LM. For instance, p can be a ViT (Dosovitskiy et al., 2020) for images, as used in LLaVA (Liu et al., 2023)); or, p can be a text representation for simple graphs. Naturally, for $s_1 \neq s_2$, we require $p(s_1) \neq p(s_2)$. With a little bit abuse of notations, prompt $p(s)$ and state s are equivalent throughout our paper.

4.1.2 Training stages of `Reflect-RL`

We propose a two-stage training pipeline for the above-mentioned language model policy. An illustration is shown in Figure 1.

Stage 1. Supervised fine-tuning (SFT). The tasks included in this work all require the instruction-following capability to a certain degree: for any valid state s, the generated action a should follow an instructed format. For example, the model should output a paragraph reflecting on previous decisions before making the next action, with two parts separated by a special token. For these tasks, we fine-tune LMs with a dataset $\mathcal{D}$ comprised of strings which follow the instruction. This process only calculates losses on the completion part.

Stage 2. Reinforcement learning fine-tuning (RLFT). We use reinforcement learning to fine-tune a pretrained language model π_{θ_0}, which can either be a publicly available LM or the one after SFT. This stage proceeds in T update steps. In step $t \in \{0, 1, \ldots, T-1\}$, we use π_{θ_t} to sample a batch of B trajectories from the environment, estimate Q-functions for each step, then perform updates using the policy optimization algorithm.

4.1.3 Training details

Reflection-aided decision-making. As demonstrated in previous works (Yao et al., 2022, 2023b; Shinn et al., 2023), generating reflection is helpful for improving the decision-making performance, which inspires us to incorporate reflection in RL. We combine the idea of reflection with both SFT and RLFT. Specifically, we first assume access to an independent reflection model R to generate reflections before the policy model π_θ makes decisions. Upon observing state s, R generates the reflection $R = R(s)$

which possibly includes analyses of current situation and plans of future steps. Then, the policy model generates the action after taking both s and R as inputs. The reflection model R is independent of π_θ: it can be either a local, pretrained language model, or a publicly-hosted LLM such as GPT-4 or Gemini (Gemini Team, 2023). One illustration can be found in Appendix F.1.

In our work, we train local LMs in SFT stage using data collected from Azure OpenAI GPT-4 (details in Appendix C.3) to serve as the reflection model $\widehat{R}_\phi$ (Line 21 of Algorithm 1). $\widehat{R}_\phi$ is frozen (denoted as $\widehat{R}$) throughout the RLFT stage. The policy model is SFTed using data containing the reflection (Line 22 of Algorithm 1). Formally, let $\mathcal{D} = \{(s_i, R_i, \alpha_i, a_i) : 1 \leqslant i \leqslant N\}$ be the dataset (Line 20 of Algorithm 1), with $|R_i| = L_i$ and $|a_i| = K_i$, then the loss functions are

$$\mathcal{L}_{\text{reflect}}(\phi) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{L_i} -\log \widehat{R}_\phi(R_{i,j}|s_i, R_{i,:j-1}),$$

$$\mathcal{L}_{\text{policy}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K_i} -\log \pi_\theta(a_{i,j}|s_i, R_i, \alpha_i, a_{i,:j-1}).$$

Here $\alpha_i = \alpha(\mathcal{A}(s_i))$ and α is the action enumeration function defined in Section 4.3.

In RLFT stage, we first query $\widehat{R}$ for the reflection, then incorporate this reflection into the policy model's input (Lines 28 and 29 of Algorithm 1). The probability of the action is

$$\pi_{\theta_t}(a|s) = \prod_{j=1}^{K} \pi_{\theta_t}(a_j|s, \widehat{R}, \alpha, a_{:j-1}).$$

Two-player design simplifies the training process. Splitting responsibilities to two players (reflection and policy) can simplify the RLFT stage because the gradients of the policy model do not affect the reflection model. We experimented using the same model for reflection and policy, while computing gradients only on the policy part. Observations (in Appendix F.2) show that such implementation greatly degraded the reflection ability. An alternative single-player approach is to perform RL and SFT concurrently so that the reflection ability can be retained, but this strategy would complicate the training process.

4.2 Generating Reflection for Training

Two components are essential in reflection generation:

• **Logical consistency.** We want a trajectory to be logically consistent, in that the action a_h at step h logically follows the reflection R_h at step h. This requirement is critical for the policy model π_θ to derive the correct action from the reflection.

Algorithm 1 Training with `Reflect-RL`

1: **Input and initialize:** Environment E, batch size B, prompting function p, action enumeration function α, SFT data size N, pretrained LM $\mathcal{M}$, LLM to generate reflection data $\mathcal{M}_R$, number of updates T.
2: $\mathcal{D}_{\text{reflect}} \leftarrow \varnothing$, $\mathcal{D}_{\text{negative}} \leftarrow \varnothing$.
3: **for** $n = 1, 2, \ldots, N$ **do**
4: $\quad E$.reset(), $h \leftarrow 1$
5: $\quad$**while** $\neg E$.done **do**
6: $\quad\quad s_h \leftarrow E$.observation()
7: $\quad\quad R_h \leftarrow \mathcal{M}_R(p(s_h), p_{\text{reflect}})$
8: $\quad\quad a_h \leftarrow \mathcal{M}_R(p(s_h), R_h, \alpha(\mathcal{A}(s_h)))$
9: $\quad\quad \mathcal{D}_{\text{reflect}} \leftarrow \mathcal{D}_{\text{reflect}} \cup \{(s_h, R_h, \alpha(\mathcal{A}(s_h)), a_h)\}$
10: $\quad\quad a'_h \sim$ Uniform$(\mathcal{A}(s_h)\backslash a_h)$ // random action
11: $\quad\quad E, E' \leftarrow E$.step$(a_h)$, E.step(a'_h)
12: $\quad\quad h \leftarrow h + 1$
13: $\quad\quad$// Look ahead: reflect after the "wrong" action
14: $\quad\quad s'_h \leftarrow E'$.observation()
15: $\quad\quad R'_h \leftarrow \mathcal{M}_R(p(s'_h), p_{\text{negative}})$
16: $\quad\quad a'_h \leftarrow \mathcal{M}_R(p(s_h), R_h, \alpha(\mathcal{A}(s'_h)))$
17: $\quad\quad \mathcal{D}_{\text{negative}} \leftarrow \mathcal{D}_{\text{negative}} \cup \{(s'_h, R'_h, \alpha(\mathcal{A}(s'_h)), a'_h)\}$
18: $\quad$**end while**
19: **end for**
20: $\mathcal{D} \leftarrow \mathcal{D}_{\text{reflect}} \cup \mathcal{D}_{\text{negative}}$
21: $\widehat{R} \leftarrow$ SFT$(\mathcal{M}, \{(R \mid p(s)) \in \mathcal{D}\})$
22: $\pi_{\theta_0} \leftarrow$ SFT$(\mathcal{M}, \{(a \mid p(s), R, \alpha(\mathcal{A}(s))) \in \mathcal{D}\})$
23: **for** $t = 0, 1, \ldots, T-1$ **do**
24: $\quad$**for** $b = 1, 2, \ldots, B$ **do**
25: $\quad\quad E$.reset(), $h \leftarrow 1$
26: $\quad\quad$**while** $\neg E$.done **do**
27: $\quad\quad\quad s_h \leftarrow E$.observation()
28: $\quad\quad\quad R_h \sim \widehat{R}(p(s_h))$
29: $\quad\quad\quad a_h \sim \pi_{\theta_t}(p(s_h), R_h, \mathcal{A}(s_h))$
30: $\quad\quad\quad E \leftarrow E$.step$(a_h)$, $h \leftarrow h + 1$
31: $\quad\quad$**end while**
32: $\quad\quad \tau_b \leftarrow (s_1, R_1, \mathcal{A}(s_1), a_1; \ldots)$
33: $\quad$**end for**
34: $\quad \theta_{t+1} \leftarrow$Policy_Gradient$(\theta_t, \{\tau_1, \ldots, \tau_B\})$
35: **end for**

• **Negative examples.** Using optimal or oracle actions to train policy models is a well-established strategy in RL. However, employing this strategy to generate training data with LLM may introduce a bias towards producing predominantly affirmative *reflections* on previous actions. If such data are exclusively used for training, the reflection model might merely flatter the decisions made by the policy model, without providing substantive self-reflections. Consequently, the model's ability to generalize to new or sub-optimal actions could be significantly limited. To mitigate this, incorporating negative examples (sub-optimal actions) can help balance the dataset and enhance the error-correcting capabilities of the reflection model.

Accordingly, we use two methods to generate the SFT dataset, with two types of special prompts p_{reflect} and p_{negative}.

At step h, we get the state s_h from the environment and send $(s_h, p_{\text{reflect}})$ to GPT-4. Here p_{reflect} tells GPT-4 to first analyze current situation, plan for the next steps, then generate the action. GPT-4 will generate a response, from which we can easily extract out reflection R_h^{GPT} and action a_h because of GPT-4's high-level instruction-following capability. Next we send a_h to the environment and increment h until termination. The above procedure generates a *logically consistent* trajectory τ. The illustration can be found between Lines 7 and 9 of Algorithm 1 and Figure 1.

To get negative examples, we start from τ or an optimal trajectory $\tau^\star$ by perturbing each step. For any step h, we first restore the environment to state s_{h-1}, then we randomly pick an action a'_{h-1} from the set $\mathcal{A}(s_{h-1})\backslash\{a_{h-1}\}$. This perturbed action will lead us into another state s'_h. We send $(s'_h, p_{\text{negative}})$ to GPT-4, where p_{negative} tells GPT-4 that the last action a'_{h-1} is sub-optimal, and lets it to find out the reason of sub-optimality, plan for the next steps to correct the mistake, then generate the action a'_h. The reflection generated at this step is $(R_h^{\text{GPT}})'$. We halt at this step, using only $(s'_h, (R_h^{\text{GPT}})', a'_h)$ as a *negative example*.

4.3 Single-Prompt Action Enumeration

The action spaces in the benchmarks are extremely large and *state-dependent*. Moreover, a valid action spans over several tokens, and has constraints on the token combination. For instance, in ALFWorld, the action spaces can differ across tasks or locations, due to variations in the objects that can be interacted with. A typical valid action is "`go to cabinet 10`" which contains 4 tokens, while "`take cabinet 10`" is invalid. However, this valid action may become invalid when presented in another task where "cabinet 10" does not exist. As stated in various works (Ahn et al., 2022; Tan et al., 2024), it is highly possible for the language model to generate a long token sequence that does not meet the constraints.

The remedies proposed by these works share the same spirit. SayCan (Ahn et al., 2022) and Action prompt normalization (Tan et al., 2024) are similar approaches enumerating all the valid actions $a \in \mathcal{A}(s)$, calculating the probability $\pi_\theta(a|s)$, and normalizing over $\mathcal{A}(s)$. Calculating $\pi_\theta(a|s)$ using a Transformer model takes $\Theta((|s| + |a|)^2)$ time. This approach takes $\Theta(\sum_{a\in\mathcal{A}(s)}(|s|+|a|)^2) = \Theta(|\mathcal{A}(s)|\,|s|^2 + \sum_{a\in\mathcal{A}(s)} |a|^2)$ time, which is intractable when $|\mathcal{A}(s)|$ is large. Here we assume $|s| \gg |a|$ as in almost all of the benchmarks. For two benchmarks (`AutoExplore` and `ALFWorld`) considered in our work, we have $|\mathcal{A}(s)| \approx 20$, $|s| \approx 500$, and $|a| \approx 5$ for almost all the states. As a result, action prompt normalization cannot be applied to our benchmarks.

We propose *single-prompt action enumeration* which shares spirit with many language classification tasks (Zellers et al., 2018; Bisk et al., 2019; Hendrycks et al., 2021) to reduce time complexity while enforcing valid actions. This method works on two sides. On the *environment* side, we introduce an extra component: the action enumeration function α. Suppose $a_1, a_2, \ldots$ is an order of actions in $\mathcal{A}(s)$, then we compose $\alpha(\mathcal{A}(s)) = (1, a_1; 2, a_2; \ldots)$ by explicitly writing down the choice letter i and action a_i. α is sent to the policy model as additional input, together with state s and reflection R. On the *model* side, we restrict the policy model to output exactly *one* token, representing the choice in α. We also mask out rows of `lm_head` (neurons of the final output layer) that does not decode into a choice letter. With these combined, we are ensured that the generated action is valid. As a comparison with action prompt normalization, the running time of our approach is $\Theta((|s|+\sum_{a\in\mathcal{A}(s)} |a|)^2) = \Theta(|s|^2 + \sum_{a\in\mathcal{A}(s)} |a|^2)$, which is strictly better. Here reflection R is considered as part of s without loss of generality.

4.4 Curriculum Learning

Curriculum learning (Elman, 1993; Bengio et al., 2009) is a paradigm in machine learning using a *topological ordering* of tasks to help with training. Starting with easy tasks, the model can have a faster convergence on hard tasks compared with directly training on them. In this work, we experiment on a curriculum design called "extra reward signal". For tasks with long horizons and sparse rewards, it is nearly impossible for a policy to sample a trajectory with a meaningful reward signal, thus policy gradient methods will make slow progress. We design the curriculum by manually adding rewards to some "milestones". In experiments of `DangerousTaxi` (see Section 5), which requires to first pick up then drop off a passenger while only giving reward after a successful dropoff, we design the curriculum to give a reward after a successful pickup.

5 Benchmarks

Motivated by the LLF-Bench (Cheng et al., 2023), we have created a natural language environment base class (`NatLangEnv`) that is compatible with the OpenAI Gym framework, characterized by its unique approach of utilizing textual representations for both observations and actions. This adjustment allows us to effectively train and test language models.

`AutoExplore`. To verify our methodology of `Reflect-RL` on the exploration example mentioned in Section 1.1, we built a complete benchmark for autonomous exploration. This benchmark contains three components: a `AutoExploreSandbox` for file protection, a multi-agent system `AutoExploreCopilot` for interactive decision-making, and a labeled dataset for performance assessment. The `AutoExplore` environment enables LMs to interact with the file

	Model	AutoExplore Depth 1	AutoExplore Depth 2	DangerousTaxi Pickup	DangerousTaxi +Dropoff*	ALFWorld
Open Source	Mistral 7B	34%	3%	7%	0%	0%
	Llama2 7B-chat	2%	1%	3%	0%	0%
	Orca-2 7B	6%	1%	1%	0%	0%
SFT Only	GPT-2 XL 1.56B	4%	9%	7%	0%	0%
RLFT Only	GPT-2 XL 1.56B	12%	3%	2%	0%	0%
SFT+RLFT (w/o reflection)	GPT-2 XL 1.56B	20%	4%	6%	0%	66%
SFT+RLFT (w/o negative)	GPT-2 XL 1.56B	33%	12%	-	-	-
Reflect-RL (Ours)	GPT-2 XL 1.56B	**36%**	**17%**	**58%**	**29%**	**74%**

Table 2: Testing performance (average success rate) of open source models (Jiang et al., 2023; Touvron et al., 2023; Mitra et al., 2023), GPT-2 XL fine-tuned with baselines, and with Reflect-RL. ReAct and memory mechanism, as shown in Figure 1, have been incorporated to improve performance. For conciseness, we have not performed prompt optimization for the open-source models, and their performance could potentially be improved with different prompting techniques in the future. **Explanation for baselines:** "SFT+RL (w/o reflection)" means the policy model is the only model involved, and the reflection field is removed from SFT data. "SFT+RL (w/o negative)" means there are no negative examples in SFT data, so both the reflection model and the policy model are trained on expert demonstrations. We only ran this ablation on AutoExplore. **Explanation for tasks:** For AutoExplore, we tested on 44 user queries, each with 10 runs. "Depth i" includes the tasks with target file depth exactly i. For DangerousTaxi, we ran on 100 random maps. "Pickup" computes the success rate of picking up the passenger, and "+Dropoff" computes the overall success rate. For ALFWorld, we tested on 4 tasks, each with 25 runs.

system safely, with the ultimate goal of answering a natural language question specified by users. The labeled dataset is composed of several real-world and synthesized repositories, with over 2500 trajectories. See Appendix C for more details.

This exploration task draws inspiration from Retrieval Augmented Generation (RAG) (Lewis et al., 2020) and InterCode (Yang et al., 2023). RAG's performance is linearly dependent on the amount of content (e.g., number of files) in the search space, presenting scalability challenges. In contrast, InterCode utilizes a tree-structured search methodology, requiring merely logarithmic space and time. This approach is notably beneficial for expansive search spaces or environments prone to frequent updates (e.g., Docker environments, customized systems). By integrating online RL training into InterCode, our proof-of-concept environment aims to create code interpreter designed for large code repositories.

During interaction with AutoExploreCopilot, each step the agent receives -1 reward as the cost of time. After 15 steps or the agent explicitly terminates, if the correct file is identified, a $+15$ reward is given; otherwise a -15 reward is given.

DangerousTaxi. We extended the OpenAI Gym's Taxi environment to introduce a higher level of challenge, thereby creating the "DangerousTaxi " environment. This game concludes prematurely if the player commits any invalid action, such as colliding with a wall, or incorrectly picking up or dropping off passengers at unauthorized locations. This modification crucially elevates the task's difficulty by eliminating the opportunity for the model to correct its mistakes after a wrong decision—a common allowance in the standard environment.

We applied curriculum learning to DangerousTaxi. In the designed pickup curriculum, we assign a positive reward 20 and terminate the environment after the driver successfully pickup the passenger. In the dropoff stage, the pickup reward is retained, but the driver needs to further dropoff the passenger at destination to receive the full reward.

ALFWorld. Our study leverages ALFWorld (Côté et al., 2019; Shridhar et al., 2020), a multi-turn platform tailored for simulating household tasks by converting the graphical representation of a house into descriptive language. A robot in is required to complete certain tasks based on the descriptions. This benchmark has gained recognition to evaluate LLM agents, with studies like Arabzadeh et al. (2024) demonstrating its efficacy. Our focus on the tomato picking task stems from its optimal mix of simplicity and representativeness.

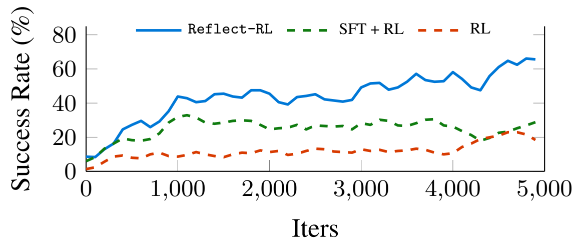

Figure 2: Training success rates of different training methods with GPT-2 XL in the pickup curriculum of the DangerousTaxi environment. We compared different RL methods for 5000 iterations during RLFT. SFT with 5000 iterations would only achieve 7% success rate, hence only RL methods are shown.

6 Experimental Results

To verify our approach, we apply `Reflect-RL` on GPT-2 XL (Radford et al., 2019). Table 2 presents a comprehensive evaluation of various models' performance across different environments. LMs still face challenges in multi-step decision-making in interactive environments, and `Reflect-RL` has significantly improved their decision-making capabilities in complex environments. This method not only utilizes the inherent strengths of LMs in *reflection* but also closely aligns with the multi-step decision-making process intrinsic to RL. Our findings highlight the potential of merging advanced prompting techniques with LMs to address complex RL tasks, establishing a new benchmark for future research in this field.

Open source models and commercial GPT models. We evaluated three open-source 7B models with necessary prompt engineering such as ReAct and memory mechanism included. These models all perform poorly on the three tasks, except for Mistral 7B on `AutoExplore` depth 1. We also examined GPT-3.5-turbo and GPT-4 (version 1106) through Azure OpenAI API. GPT-4 can achieve a success rate of 71% in `AutoExplore` depth 1, 81% in depth 2, and 84% in `ALFWorld`; meanwhile, GPT-3.5-turbo achieves a success rate of 31% in `AutoExplore` depth 1, 8% in depth 2, and 6% in `ALFWorld`. During the evaluation, we noticed potential data contamination of these two models: GPT-4 can sometimes identify near-optimal actions without extensive exploration of the space. In the `DangerousTaxi` environment, the success rates of the dropoff curriculum for GPT-4 and GPT-3.5-turbo are both 0%. Even though GPT-4 has 70% chance executing a valid action in each step, it is prone to failure upon committing minor errors along the long navigation path during multi-turn interactions. These observations suggest that even powerful LLMs may still need online RL training for multi-turn interactions.

SFT is not enough. Supervised fine-tuning (SFT) has been widely used offline to improve LMs' performance on specific tasks. However, our results (Table 2) indicate that SFT alone is not sufficient for complex RL tasks requiring multi-step decision-making. While SFT enhances task-specific knowledge, it fails to solve problems requiring deep reasoning, planning, and reflection.

Reflection helps learning. Incorporating reflective processes into LLMs significantly enhances decision-making and learning from past actions. Our comparative analysis between models with and without reflection capabilities highlights the importance of reflection for advanced understanding and adaptability in RL tasks. As shown in Figure 2, the curves representing online RL without reflection are constantly below the curve of `Reflect-RL`. Figure 4 shows a similar result.

Reflecting from mistakes is beneficial. The philosophy of "learning from mistakes" plays a meaningful role in `Reflect-RL`. Without negative reflection samples, the model's performance would be worse (in absolute difference) than the model trained with both positive and negative data. For `AutoExplore`, the test accuracies without negative examples are 33% and 12% for each curriculum, compared with 36% and 17% with negative examples. As shown in Figure 3, the solid curve represents the integration of negative examples into the SFT dataset, and we observed a faster convergence during RLFT.

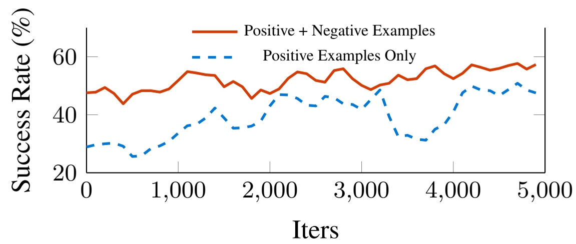

Figure 3: Training success rate with and without negative examples in the `AutoExplore` setting, each assessed in a single run. When negative examples are excluded, the training process exhibits decreased speed and lacks smoothness.

Curriculum learning (CL) accelerates learning. As shown in the top two curves in Figure 4, CL accelerates the learning curve for complex RL tasks by structuring the training process with challenging tasks. To ensure a fair evaluation, both learning approaches (`Reflect-RL` with and without CL) are pre-trained with the same reflection dataset during the SFT phase. The curriculum learning approach begins with an initial RL training phase focused on the pickup curriculum, followed by the dropoff curriculum. Without curriculum learning, the model is trained directly using the dropoff curriculum, resulting in slightly inferior performance.

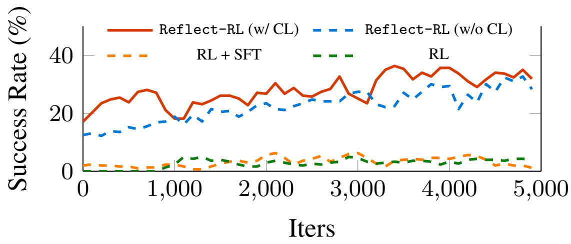

Figure 4: Comparison of training success rates in the drop-off curriculum in the `DangerousTaxi` environment. The top two curves represent `Reflect-RL`; "w/ CL" means the experiment incorporates curriculum learning (CL) and is trained with the pickup curriculum. The bottom two dashed curves represent online RL without reflection. All single run.

Sensitivity of the policy model with respect to the reflection model. For `DangerousTaxi` pickup subtask, using the same policy model after `Reflect-RL`, we switch the reflection model to GPT-2 Small 0.12B and Mistral 7B SFTed with the reflection data. The results for GPT-2 Small 0.12B, GPT-2 XL 1.56B, and Mistral 7B are 55%, 58% (as in Table 2), and 64%. This phenomenon indicates that the policy model is not extremely sensitive to the robustness/accuracy of the reflection model as the policy model can easily adapt. Additionally, using a more capable reflection model can improve the performance.

7 Discussion and Conclusion

Risk, impact, and responsible AI. In this study, we adhere to principles of Responsible AI by ensuring transparency, efficiency, and security in both the training and evaluation stages. An exemplar of our commitment is the development of `AutoExploreSandbox`, designed to reduce the risk of security issues in the file system. Recognizing the importance of ethical considerations and the social impact of our work, we pledge to engage in continuous evaluation of LMs's performance in multi-step environment.

Limitations. Our study, while comprehensive, acknowledges certain limitations. Although `ALFWorld` benchmark is multimodal, this study primarily focued on the text representation, leaving the examination of multimodal models and cross-attention encoding of other modalities (such as images and audio) for future work. Comparisons with commercial models is discussed in Section 6, but the proprietary nature and potential biases (e.g., unknown training data) limit a fair comparison with open-source models. Standardized benchmarks in the field are needed for further evaluation. Lastly, the reflection data utilized in our study is generated by GPT-4, which may not fully capture the distribution of real human data. This indicates the importance of integrating more authentic human-generated data in future evaluations.

Future direction. The primary goal of this study is to create an efficient online RL pipeline for LMs to perform multi-step problem solving. Building on this foundation, future research directions may explore the scalability of `Reflect-RL` to develop larger foundation models, enabling them to adapt to previously unseen environments with out-of-domain generalization capabilities. The two-player design in our framework may naturally be extended to other multi-agent settings where language models can show their strengths. Another future direction is to train the reflection model in RLFT stage as we freeze it because of the interference with the policy model (Appendix F.2), which will improve the reasoning ability of language models for decision-making tasks.

Acknowledgement

We thank Adith Swaminathan, Julia Kiseleva, Ishai Menache, Chi Wang, Yin Tat Lee, and Yi Zhang from Microsoft Research for useful discussions and support.

We also want to thank Jieyu Zhang from University of Washington for their valuable feedback.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances.

Negar Arabzadeh, Julia Kiseleva, Qingyun Wu, Chi Wang, Ahmed Awadallah, Victor Dibia, Adam Fourney, and Charles Clarke. 2024. Towards better Human-Agent Alignment: Assessing Task Utility in LLM-Powered Applications.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2023. ChemCrow: Augmenting large-language models with chemistry tools.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code.

Yen-Chun Chen, Zhe Gan, Yu Cheng, Jingzhou Liu, and Jingjing Liu. 2020. Distilling Knowledge Learned in BERT for Text Generation.

Ching-An Cheng, Andrey Kolobov, Dipendra Misra, Allen Nie, and Adith Swaminathan. 2023. LLF-Bench: Benchmark for Interactive Learning from Language Feedback. *arXiv preprint arXiv:2312.06853*.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding Pretraining in Reinforcement Learning with Large Language Models.

Zane Durante, Bidipta Sarkar, Ran Gong, Rohan Taori, Yusuke Noda, Paul Tang, Ehsan Adeli, Shrinidhi Kowshika Lakshmikanth, Kevin Schulman, Arnold Milstein, et al. 2024. An Interactive Agent Foundation Model. *arXiv preprint arXiv:2402.05929*.

Jeffrey L. Elman. 1993. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99.

Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Joyce Cahoon, Rana Alotaibi, et al. 2024. NL2SQL Is a Solved Problem... Not! In *Proceedings of the CIDER 2024*.

Gemini Team. 2023. Gemini: A Family of Highly Capable Multimodal Models.

Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, and Yoshua Bengio. 2017. On integrating a language model into neural machine translation. *Computer Speech & Language*, 45:137–148.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models.

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. *arXiv preprint arXiv:2310.01798*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot Learning with Retrieval Augmented Language Models.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. 2022. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual Language Model Pretraining.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*.

Ximing Lu, Sean Welleck, Jack Hessel, Liwei Jiang, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. 2022. Quark: Controllable Text Generation with Reinforced Unlearning.

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. ReFT: Reasoning with Reinforced Fine-Tuning.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2023. Teaching Small Language Models to Reason.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. 2023. Orca 2: Teaching Small Language Models How to Reason.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive Learning from Complex Explanation Traces of GPT-4.

NVIDIA. 2024. Build a Custom LLM with Chat With RTX — NVIDIA. `https://www.nvidia.com/en-us/ai-on-rtx/chat-with-rtx-generative-ai/`. Accessed: 2024-02-13.

OpenAI. 2023. GPT-4 Technical Report.

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

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. 2023. Is Reinforcement Learning (Not) for Natural Language Processing: Benchmarks, Baselines, and Building Blocks for Natural Language Policy Optimization.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2022. Learning to summarize from human feedback.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

R.S. Sutton and A.G. Barto. 1998. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Walter Talbott, Katherine Metcalf, Natalie Mackraz, Devon Hjelm, and Alexander Toshev. 2023. Large Language Models as Generalizable Policies for Embodied Tasks.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. 2024. True Knowledge Comes from Practice: Aligning LLMs with Embodied Environments via Reinforcement Learning.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models.

Alexander Sasha Vezhnevets, John P Agapiou, Avia Aharon, Ron Ziv, Jayd Matyas, Edgar A Duéñez-Guzmán, William A Cunningham, Simon Osindero, Danny Karmon, and Joel Z Leibo. 2023. Generative agent-based modeling with actions grounded in physical, social, or digital space using Concordia. *arXiv preprint arXiv:2312.03664*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023a. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Zengzhi Wang, Qiming Xie, Zixiang Ding, Yi Feng, and Rui Xia. 2023b. Is ChatGPT a Good Sentiment Analyzer? A Preliminary Study.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback. *arXiv preprint arXiv:2306.14898*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of Thoughts: Deliberate Problem Solving with Large Language Models.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2023b. Retroformer: Retrospective Large Language Agents with Policy Gradient Optimization.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. 2023. Large Language Model Is Semi-Parametric Reinforcement Learning Agent. *arXiv preprint arXiv:2306.07929*.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2023. Can ChatGPT Understand Too? A Comparative Study on ChatGPT and Fine-tuned BERT.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-Tuning Language Models from Human Preferences.

Appendix

Table of Contents

A Discussion on PPO

Proximal policy optimization (PPO) is an advanced policy gradient method, which aims to take the largest possible improvement step on a policy while ensuring the deviation from the previous policy is reasonably small. The update step is

$$\theta_{t+1} = \arg\max_{\theta} \mathbb{E}_{s,a}\Bigg[\min\Bigg\{\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}A^{\pi_{\theta_t}}(s,a), \\ \mathsf{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_t}}(s,a)\Bigg\}\Bigg],$$

where ϵ usually takes small values such as 0.1 or 0.2.

In practice, we found that PPO did not work well. For tasks with a large state space, action space, and a long horizon, the training processes were constantly unstable, with sudden drops of the expected total reward. Such tasks pose high difficulty for the value function estimator to learn the value functions when the policy network changes. Most importantly, inherent randomness (including dropout, padding length in different batches, `top_p` and `top_k`) of LMs results in high sensitivity for the terms of $\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}$. Though we want the policy to sample actions with small possibilities (e.g., $\pi_{\theta_t}(a|s) < \varepsilon$) to encourage exploration, high sensitivity will result in such values becoming to 0 in almost all the future re-evaluations.

B Illustrations of Pipeline

In this section we present detailed versions of Figure 1. Figure 5 is the illustration for data generation. Figure 6 is the illustration for SFT stage. Figure 7 is the illustration for RLFT stage.

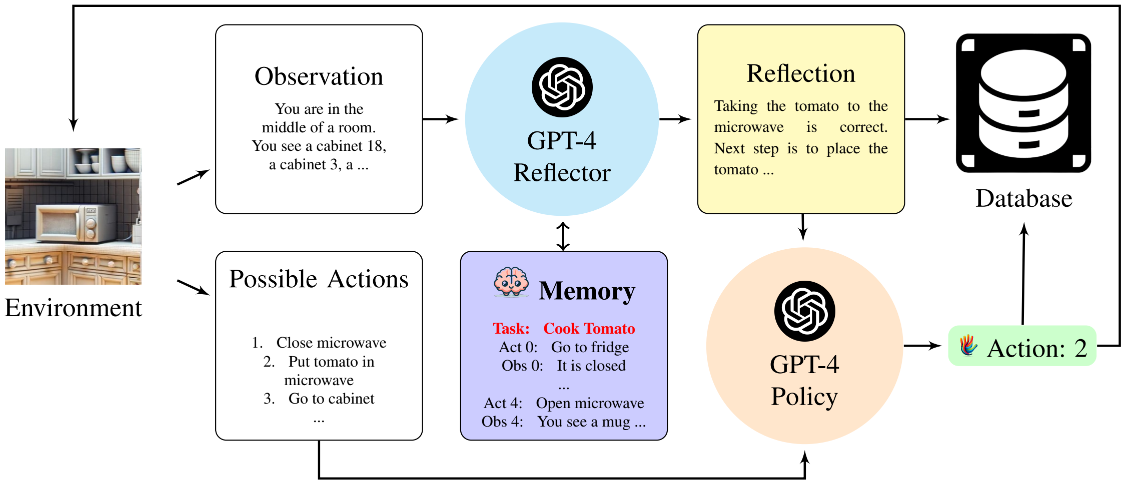

Figure 5: Pipeline of `Reflect-RL` data generation.

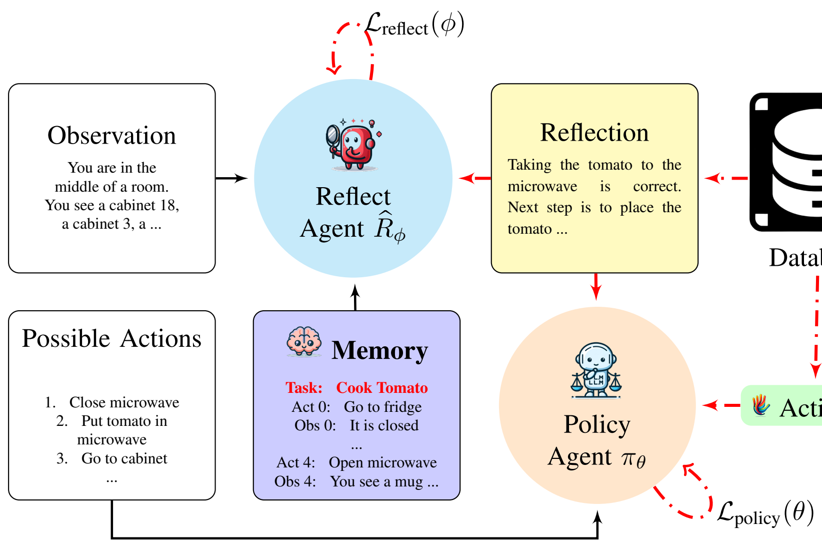

Figure 6: Pipeline of `Reflect-RL` SFT stage.

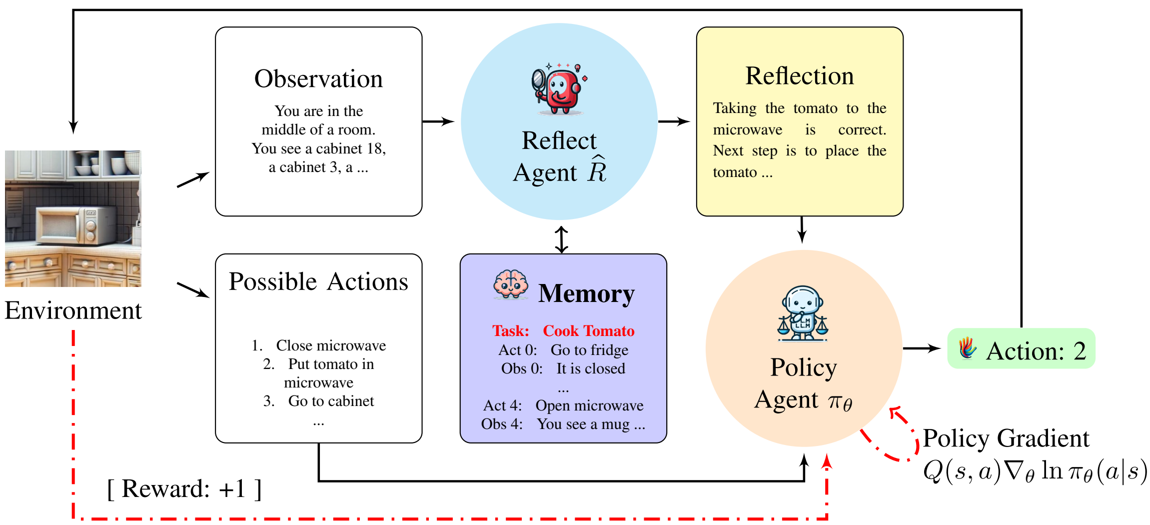

Figure 7: Pipeline of `Reflect-RL` RLFT stage.

C Autonomous Exploration Details

Autonomous exploration in a well-organized repository can reduce the number of reads of files to a large extent. Ideally, if the repository has n files and is organized as a k-ary tree, the best language model only takes $O(k + \log_k n)$ (compared to $O(n)$ using exhaustive enumeration) commands to identify the correct file, then proceed with the specific needs of reading, editing, and executing. This serves as an motivation of autonomous exploration benchmark.

C.1 Autonomous Exploration Sandbox

`AutoExploreSandbox` is a sandbox protecting the original repository from modification. An instance of `AutoExploreSandbox` could be initialized with the path to the original repository, then this instance will create a temporary directory in a specified location (could possibly be a ram disk) and make a duplication of the original repository. `AutoExploreSandbox` supports two main functions:

1. **Executing system commands:** For the purpose of our work, commands such as "`cd`", "`ls`", "`cat`", "`head`", "`tail`", "`echo`", "`python`" and "`pip`" are supported to enable document retrieval and coding.

2. **Tracking changed files:** The user of `AutoExploreSandbox` could call a function to get the list of the changed files and their contents compared to the original status when creating the sandbox.

C.2 Autonomous Exploration Copilot

`AutoExploreCopilot` is an agent mediating between language models, humans, and `AutoExploreSandbox`. An instance of `AutoExploreCopilot` could be initialized with a natural language question and the corresponding repository to work in. The main function of `AutoExploreCopilot` is to give natural language descriptions of the current autonomous exploration task for either human or language models to make decisions. The interaction proceeds in loops (k starts from 0):

- **Step** $3k+1$**: Prompting.** `AutoExploreCopilot` compiles a prompt p_k given the current status of `AutoExploreSandbox`, which includes the question, current working directory (cwd) in the repository, files and folders under cwd (optional, can be used to replace `ls` and reduce interaction), historical commands $c_0, c_1, \ldots, c_{k-1}$ from the human or language model, and execution result of the last command c_{k-1}.

- **Step** $3k+2$**: Querying.** `AutoExploreCopilot` sends the prompt p_k to human or LM and gets the response. This response may contain excessive information such as analysis of the current situation (which is a typical behavior of GPT-4), so `AutoExploreCopilot` needs to extract system command c_k from the response.

- **Step** $3k+3$**: Executing.** `AutoExploreCopilot` sends the system command c_k to `AutoExploreSandbox` and gets the execution results. The results contain standard output and standard error, such as the file content after "`cat`" and runtime error of "`python`".

The interaction ends when the response in step $3k + 2$ contains an exit signal stipulated in the prompt.

`AutoExploreCopilot` is capable of prompting GPT-4 to do the entire task, while for the smaller models in this work we only set the goal to be a subtask (file identification).

C.3 Labeled Dataset

The licenses are bounded by each open-source repository used in this dataset.

Using GPT-4 from Azure OpenAI service, we constructed a synthetic repository called "Coffee Company", which contains documents (in `.md` format), codes (in various programming languages), and database files (in `.csv` format). This repository contains around 12 million tokens. In addition, we downloaded 12 open-source repositories containing codes and documentations from GitHub.

After collecting the repositories, we built a labeled dataset regarding autonomous exploration. Each datum in the dataset contains the following fields: the name of the repository n, a natural language question q, an answer to this question a, the related file f, and the shortest system command path to reach this file $c^\star = (c_0^\star, c_1^\star, \ldots, c_{L-1}^\star)$. As a start point, this work focus on an important step in autonomous exploration: find the correct file f given the natural language question q, so the answer a is for future work.

This dataset is generated in a "reverse question generation" manner. We first enumerate the pair (n, f), then send the content of f to GPT-4 to let it generate several pairs of (q, a). We prompt GPT-4 to ask questions on the functionality of the file by requiring it to analyze the file's role in the whole repository.

This dataset contains 1764 training data (292 user queries), 505 validation data (86 user queries), and 252 test data (44 user queries).

Here is the prompt template for `AutoExplore` label generation:

```
Below is a text file {NAME} from a
repository. This repository is deployed
as a backend service, providing users
```

```
with certain services. Users want to use
specific functionality or ask questions
about the services, such as "tell me the
business philosophy of this company" or
"what is the high-level architecture of
the proposed model". These inquiries are
guaranteed to be answered by reading some
text files.

Your task is to first analyze its
content. Then, come up with some user
queries which involves this text file,
along with the answers to them. Use the
following format:

# ANALYSIS
...
- QUERY1: ...
- ANSWER1: ...
- QUERY2: ...
- ANSWER2: ...

----- Text -----
{CONTENT}
```

C.4 Reflection Generation

Here is the system message for `AutoExplore` reflection generation:

```
You are a helpful assistant to explore
a file system. Given a natural language
task, you need to generate a sequence
of system commands to identify the
correct file. During interaction, you
can only output a single choice number
as response, which comes from a list
of commands given to you. For example,
the possible commands are: ["A. cat
test.py", "c. cd progs", "9. cd .."].
Your answer should be "A", "c", or "9",
not the entire command.

A special command 'id X' is introduced
to this task, which means to identify
the file X as the final answer. Once
you are sure X is the answer, use 'id'
to explicitly identify it, then the
interaction terminates. Remember, simply
'cat' a file does not identify it.
```

Here is p_{reflect}: `Now analyze the current situation and plan what to do next using 50 words. Don't give the choice yet. If you have identified the correct file in previous steps, you should exit at this step.`

Here is p_{negative}: `The last command opens a wrong folder or file, which is a suboptimal move. Give reasons why the folder or file is incorrect and plan what to do next using 50 words. Don't give the choice yet.`

D Other Benchmark Details

D.1 Dangerous Taxi

OpenAI Gym uses MIT License.

Here is the system message for `DangerousTaxi` reflection generation:

```
Given a problem state, the actions you
have taken, and the observations you
have.

You need to give reflection on your
actions, such as:
- What is the consequence of your
previous action?
- How is your previous action? Good or
bad? Why?
- What is the next action you want to
take if possible? Why?

I might give you some spoiler information
and optimal action for cheating, but you
should not mention that you have seen any
spoilers, optimal actions, or any other
information that you should not know.
Pretend you are smart and just know these
information.

Don't use any words related to "optimal"
in your reflection.

Keep your reflection concise within 100
words.

For instance,
Because ..., so I ...
The task is to,... I
I found ... So...
etc.
```

Here is p_{negative}: `The previous actions might contain some mistakes.` p_{negative} is directly appended to the observation prompt $p(s)$.

D.2 ALFWorld

ALFWorld and TextWorld use MIT License, Fast Downward uses GNU General Public License (GPL) v3.0.

`ALFWorld` shares the same system message and p_{negative} with `DangerousTaxi`.

Hyperparameter	Value
Train batch size	1 on 4090 2 on A6000 and A40
Evaluate / Sample trajectory batch size	4
Gradient accumulation steps	1
Learning rate	2×10^{-4}
Gradient clipping norm	0.3
Weight decay	1×10^{-3}
Precision	bf16
LoRA quantization	8bit
LoRA α	16
LoRA rank	64
Maximum token length	1024
Temperature	1
Top p	1
Top k	99999

Table 3: Hyperparameters of experiments

E Experiment Details

(a) `AutoExplore` (b) `DangerousTaxi` (c) `ALFWorld`

Figure 8: Illustration of our environment

We use NVIDIA RTX 4090, RTX A6000, and Tesla A40 for the training and evaluation of our proposed `Reflect-RL` method. Python, PyTorch, HuggingFace PEFT, and AutoGen are used throughout the project.

All the experiments share the set of hyperparameters in Table 3.

For `AutoExplore`, it takes around 30 sec per iteration under the setting of horizon $H = 15$. For `DangerousTaxi`, it takes 30 sec per iteration under the setting of horizon $H = 15$ in pickup, and 60 sec per iteration under the the setting of horizon $H = 30$ in dropoff. For `ALFWorld`, it takes 90 sec per iteration under the setting of horizon $H = 15$.

F Qualitative Observations

F.1 Illustration of Reflection-Aided Decision-Making

Figures 9 to 14 show an entire trajectory of our reflection-aided decision-making method. Irrelevant observations and actions are omitted by "...".

F.2 Same Model for Reflection and Policy

Figures 15 and 16 are two log scripts in an experiment. In this experiment, the reflection model $\widehat{R}$ and policy model π_{θ_t} are the same. Thus, the gradients computed on the action part will be influence the reflection part. The reflect quality degraded drastically after 1194 iterations of updates.

Observation:
You are in the middle of a room. Looking quickly around you, you see a cabinet 20, ..., a fridge 1, ...
Your task is to: put a tomato in microwave.
Reflection:
Begin by searching the cabinets or the fridge to find a tomato, then place it into the microwave. // Check cabinets: The most common place to find a tomato is in the cabinet or the fridge, but sometimes a cabinet isn't relevant.
Action List:
Action 0: go to cabinet 1
Action 1: go to cabinet 10
...
Action 27: go to fridge 1
...
Action:
27

Figure 9: Step 1 of a reflection-aided decision-making example.

Observation:
You are in the middle of a room. Looking quickly around you, you see a cabinet 20, ...
Your task is to: put a tomato in microwave.
Here is the previous path:
--- Step: 0 ---
Action: go to fridge 1
Observation: You arrive at loc 24. The fridge 1 is closed.
Reflection:
The initial action of going to the fridge aligns well with the task, as tomatoes are typically stored in a fridge. To progress, the next logical action would be to open the fridge to search for a tomato.
Action List:
Action 0: examine fridge 1
Action 1: go to cabinet 1
...
Action 41: open fridge 1
Action:
41

Figure 10: Step 2 of a reflection-aided decision-making example.

Observation:
You are in the middle of a room. Looking quickly around you, you see a cabinet 20, ...
Your task is to: put a tomato in microwave.
Here is the previous path:
--- Step: 0 ---
...
--- Step: 1 ---
Action: open fridge 1
Observation: You open the fridge 1. The fridge 1 is open. In it, you see a egg 1, a pot 2, a pot 1, a tomato 2, and a tomato 1.
Reflection:
My initial actions were efficient, as I quickly located and opened the fridge to find the tomato necessary for my task. The next logical step is to take a tomato from the fridge and then proceed to put it in the microwave to complete the task.
Action List:
Action 0: close fridge 1
...
Action 46: take tomato 2 from fridge 1
Action:
46

Figure 11: Step 3 of a reflection-aided decision-making example.

Observation:
You are in the middle of a room. Looking quickly around you, you see a cabinet 20, ..., a microwave 1, ...
Your task is to: put a tomato in microwave.
Here is the previous path:
--- Step: 0 ---
...
--- Step: 2 ---
Action: take tomato 2 from fridge 1
Observation: You pick up the tomato 2 from the fridge 1.
Reflection:
The actions taken so far have effectively brought the tomato into my hands, which is in alignment with the first part of the given task. The next step is to navigate to the microwave and complete the operation to place the tomato inside it to fulfill the objective of placing a tomato in the microwave.
Action List:
Action 0: close fridge 1
...
Action 32: go to microwave 1
...
Action:
32

Figure 12: Step 4 of a reflection-aided decision-making example.

Observation:
You are in the middle of a room. Looking quickly around you, you see a cabinet 20, ...
Your task is to: put a tomato in microwave.
Here is the previous path:
--- Step: 0 ---
...
--- Step: 3 ---
Action: go to microwave 1
Observation: You arrive at loc 22. The microwave 1 is closed.
Reflection:
The actions taken thus far have successfully identified and retrieved the tomato from the fridge, progressing towards the task goal. The next step should be to open the microwave and put the tomato inside to complete the task effectively.
Action List:
Action 0: examine microwave 1
...
Action 43: open microwave 1
Action:
43

Figure 13: Step 5 of a reflection-aided decision-making example.

Observation:

```
You are in the middle of a room.
Looking quickly around you, you see a
cabinet 20, ...
Your task is to: put a tomato in
microwave.
Here is the previous path:
--- Step: 0 ---
...
--- Step: 4 ---
Action: open microwave 1
Observation: You open the microwave
1. The microwave 1 is open. In it,
you see a mug 1.
```

Reflection:

```
Successfully procuring the tomato
from the fridge was a correct step
towards completing the task; however,
an oversight occurred in not removing
the mug from the microwave before
attempting to place the tomato
inside. The next action should be
to put the tomato in the microwave,
which completes the task at hand.
```

Action List:

```
Action 0: close microwave 1
...
Action 44: put tomato 2 in/on
microwave 1
```

Action:

```
44
```

Figure 14: Step 6 of a reflection-aided decision-making example.

Observation:

```
# Task
What is the reason for the planned
domain name change for the website?
# Current working directory
.
# Files under current working
directory
README.md
# Command history

# Execution result of your last
command
```

Reflection:

```
The README.md file typically contains
information about the project,
including reasons for the planned
domain name change.
```

Action List:

```
t. exit
103. cat README.md
110. id README.md
```

Action:

```
103
```

Figure 15: A log script from iteration #10.

Observation:

```
# Task
What is the reason for the planned
domain name change for the website?
# Current working directory
.
# Files under current working
directory
README.md
# Command history

# Execution result of your last
command
```

Reflection:

```
cat README.md id README.md #
Explanation:  cat README.md id
README.md # Explanation:  cat
README.MD id README.md
```

Action List:

```
166.  id README.md
T. exit
g.  cat README.md
```

Action:

```
g
```

Figure 16: A log script from iteration #1204.