

---

# Progress Reward Model for Reinforcement Learning via Large Language Models

---

**Anonymous Author(s)**

Affiliation

Address

email

## Abstract

1 Traditional reinforcement learning (RL) algorithms face significant limitations in  
2 handling long-term tasks with sparse rewards. Recent advancements have leveraged  
3 large language models (LLMs) to enhance RL by utilizing their world knowledge  
4 for task planning and reward generation. However, planning-based approaches  
5 often depend on pre-defined skill libraries and fail to optimize low-level control  
6 policies, while reward-based methods require extensive human feedback or ex-  
7 haustive searching due to the complexity of tasks. In this paper, we propose the  
8 Progress Reward Model for RL (PRM4RL), a novel framework that integrates task  
9 planning and dense reward to enhance RL. For high-level planning, a complex task  
10 is decomposed into a series of simple manageable subtasks, with a subtask-oriented,  
11 fine-grained progress function designed to monitor task execution progress. For  
12 low-level reward generation, inspired by potential-based reward shaping, we use the  
13 progress function to construct a Progress Reward Model (PRM), providing theoreti-  
14 cally grounded optimality and convergence guarantees, thereby enabling effective  
15 policy optimization. Experimental results on robotics control tasks demonstrate  
16 that our approach outperforms both LLM-based planning and reward methods,  
17 achieving state-of-the-art performance.

18 

## 1 Introduction

19 Reinforcement learning (RL) has shown exceptional potential in solving sequential decision-making  
20 problems. However, in real-world scenarios involving complex, long-term tasks with sparse rewards,  
21 agents face significant challenges in planning and decision-making due to insufficient feedback  
22 signals[1, 2]. Traditional RL frameworks address these challenges primarily by focusing on two  
23 strategies: task decomposition and reward shaping. Hierarchical RL[3, 4] simplifies decision-making  
24 by autonomously decomposing complex tasks into hierarchical subtasks, while reward shaping  
25 techniques provide dense training signals through the reconstruction of reward functions, such as  
26 inverse RL[5, 6, 7] and intrinsic reward mechanisms[8]. Despite their advantages, both approaches  
27 heavily rely on expert data for priors[9] or supervision[10], and the large-scale acquisition of such  
28 data is often challenging, limiting the scalability and applicability of these methods.

29 Recent advancements in large language model (LLM) research have made significant progress in  
30 areas such as dialogue[11, 12], sequential decision-making[13, 14], and coding[15, 16, 17]. By  
31 leveraging the extensive and diverse human knowledge acquired during the pre-training phase, LLMs  
32 possess a broad range of world knowledge and emergent reasoning abilities[18, 19]. Consequently,  
33 LLM-Augmented RL has emerged as a promising approach to overcome the challenges faced by  
34 traditional RL methods[20]. To tackle the issues of long-term planning and sparse rewards, previous  
35 LLM-based approaches have evolved into two main paradigms: LLM as a planner[21, 22, 23] and  
36 LLM as a rewarder[24, 25, 26], as illustrated in Figure 1. As a planner, LLM decomposes complex,

37 long-term tasks into a series of short-term, goal-oriented subtasks, incorporating subtask priors into  
 38 the decision-making process. As a rewarder, LLM constructs a dense reward function for sparse-  
 39 reward tasks, providing guidance to the underlying policy based on the meaningful priors encoded  
 40 within the LLM.

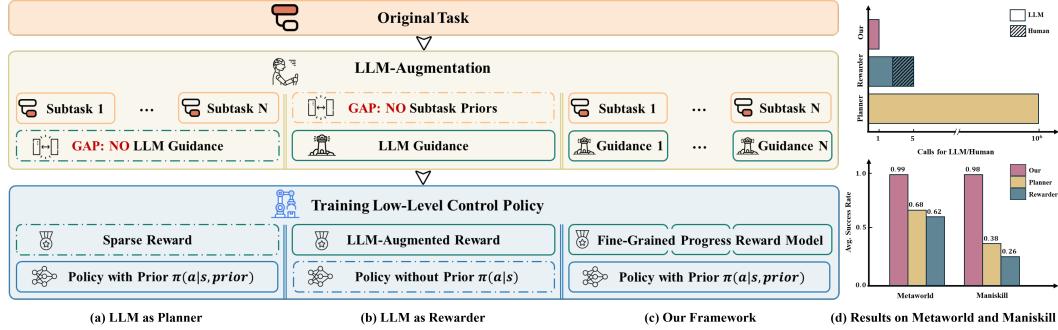


Figure 1: Quick view. (a), (b) and (c) shows the structural similarities and differences between LLM-Augmented methods. (d) is the average evaluation results on Metaworld[27] and Maniskill[28].

41 However, LLM as a planner primarily focuses on high-level planning. In the absence of LLM-  
 42 augmented dense reward guidance, task decomposition alone—relying solely on natural-language task  
 43 priors—is insufficient to support the effective learning of subtasks[29, 22]. Previous approaches have  
 44 alleviated this limitation by constructing pre-trained policy libraries[21] or incorporating traditional  
 45 motion planning to link subtasks[22], yet these methods lack flexibility and efficiency due to their  
 46 non-end-to-end nature. On the other hand, LLM as a rewarder provides low-level guidance. However,  
 47 without subtask decomposition, constructing a reward function for an entire long-term, complex  
 48 task remains a significant challenge for LLMs[30, 31]. Previous methods have enhanced the quality  
 49 of LLM-generated reward functions by incorporating human feedback[24] or evolutionary search  
 50 algorithms[25, 26], but these approaches still require extensive search and significant human expert  
 51 assistance. As shown in Figures 1a and 1b, both of these approaches focus on a single aspect of RL,  
 52 neglecting the importance of integrating both high-level planning and low-level reward shaping.

53 To address the challenges of long-term planning and sparse rewards in reinforcement learning more  
 54 efficiently, we propose a novel framework, Progress Reward Model for Reinforcement Learning  
 55 (PRM4RL), which integrates both high-level long-term planning and low-level dense rewards to  
 56 provide a comprehensive solution, as illustrated in Figure 1c. In high-level planning, the task is  
 57 decomposed into subtasks, each with termination conditions and fine-grained progress evaluations.  
 58 For low-level reward generation, we utilize the progress to construct an efficient dense reward  
 59 signal with theoretically grounded optimality and convergence guarantees. In practice, our pipeline  
 60 decomposes a complex task into a series of manageable, goal-oriented subtasks, each associated with  
 61 a natural language subtask prior. For each subtask, we build a fine-grained progress function that  
 62 dynamically tracks the execution of the task. Inspired by the potential field theorem, we regard the  
 63 progress function as a potential function and design a Progress Reward Model with optimality and  
 64 convergence guarantees. The evaluation results in Metaworld[27] and Maniskill[28] as shown in  
 65 Figure 1d demonstrate the effectiveness of our proposed framework, with significant improvements  
 66 in success rate and reduced LLM calls, outperforming existing methods in both performance and  
 67 efficiency.

## 68 2 Related Works

### 69 2.1 LLM for Planning

70 Large language models (LLMs), enriched with diverse world knowledge and human priors during  
 71 their pretraining phase, possess the ability to understanding and reasoning[18, 19, 17, 16, 11]. When  
 72 provided with relevant task information, LLMs are able to decompose complex, long-term tasks  
 73 into simpler, short-term tasks, each focused on specific objectives[32, 33]. Numerous approaches  
 74 have leveraged Task and Motion Planning (TAMP) to realize such task decomposition in robotics

75 scenes[34, 35]. However, when using RL for low-level control strategies, a significant gap arises  
 76 between high-level planning and low-level control—how to efficiently leverage the natural language  
 77 plans generated by LLMs to guide policy training? SayCan[21] addresses this challenge by utilizing  
 78 a set of pretrained policies, where the LLM simply selects the appropriate policy based on the plans.  
 79 Plan-Seq-Learn[22] uses traditional motion planning to link subtasks, alleviating the burden on RL  
 80 policies. In text-based environments, ELLM[23] trains RL policies by describing state transition in  
 81 natural language and computing the similarity between the executed actions and the plan, using this  
 82 similarity as a reward signal. However, this approach is computationally expensive and not suitable  
 83 for robotics tasks. In contrast, our method constructs an efficient and dense reward signal through  
 84 the Progress Reward Model (PRM), effectively bridging the gap between high-level planning and  
 85 low-level policy training.

## 86 2.2 LLM for Reward Generation

87 Reward functions are the primary mechanism to guide RL training[36, 37], leading to numerous  
 88 approaches that utilize LLMs to generate executable reward functions that provide dense rewards.  
 89 Text2Reward[24] uses LLMs to generate executable python functions based on a goal described  
 90 in natural language, with human involvement in the loop. Eureka[25] extends this paradigm by  
 91 introducing evolutionary search and automatic reward refinement through LLMs. REvolve[26]  
 92 translates implicit human knowledge into explicit reward functions for training RL with human  
 93 feedback. However, these methods treat the task as a whole and do not account for the potential  
 94 stage-wise features of complex tasks. As a result, the generation process of reward function tend to  
 95 be highly complex and often require human feedback[24, 26] or extensive search[25]. In contrast, our  
 96 approach simplifies the reward generation process by decomposing the task into short-term, simple  
 97 subtasks, providing more specific and clear guidance for each stage of the task.

## 98 3 Preliminary

99 In standard reinforcement learning (RL)[1] tasks, the problem is typically modeled as a Markov  
 100 Decision Process (MDP), denoted by  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ . Here,  $\mathcal{S}$  and  $\mathcal{A}$  represent the state  
 101 and action spaces, respectively,  $\gamma$  is the discount factor, and  $\rho_0$  is the distribution of initial states. The  
 102 dynamics of environment are captured by the transition function  $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ , which  
 103 defines the probability of transitioning from one state to another given a specific action. The reward  
 104 function  $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  quantifies the immediate reward received for taking an action in a particular  
 105 state. A trajectory  $\tau = (s_0, a_0, r_0, \dots, s_T, a_T, r_T)$  describes the sequence of states, actions, and  
 106 rewards encountered by an agent following a decision-making policy  $\pi$ . At each time step  $t$ , the agent  
 107 selects an action  $a_t$  according to  $a_t \sim \pi(\cdot|s_t)$ , and the environment responds by transitioning to the  
 108 next state  $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ . The state-action value function (Q-function)  $Q^\pi(s, a)$  represents the  
 109 expected cumulative return when taking action  $a$  in state  $s$  and subsequently following policy  $\pi$ :

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{k=0}^T \gamma^k R(s_{t+k}, a_{t+k}) \mid s_t = s, a_t = a \right] \quad (1)$$

110 The objective in RL is for the agent to learn a policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  that maximizes the expected  
 111 cumulative return defined as:

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{t=0}^T \gamma^t R(s_t, a_t) \right] \quad (2)$$

## 112 4 Method

113 Our framework, as illustrated in Figure 2, bridges the gap between high-level planning and low-level  
 114 training. We begin by decomposing a complex long-term task into a series of short-term subtasks,  
 115 with a determination function  $\Psi$  to classify the current subtask (Section 4.1). Next, we construct  
 116 a potential-based Progress Reward Model (PRM) derived from the progress function  $\Phi$ , which  
 117 generates effective reward signals that are proved to have convergence and optimality guarantees  
 118 (Section 4.2). Finally, the policy is augmented using both the PRM reward and the subtask priors  
 119 (Section 4.3).

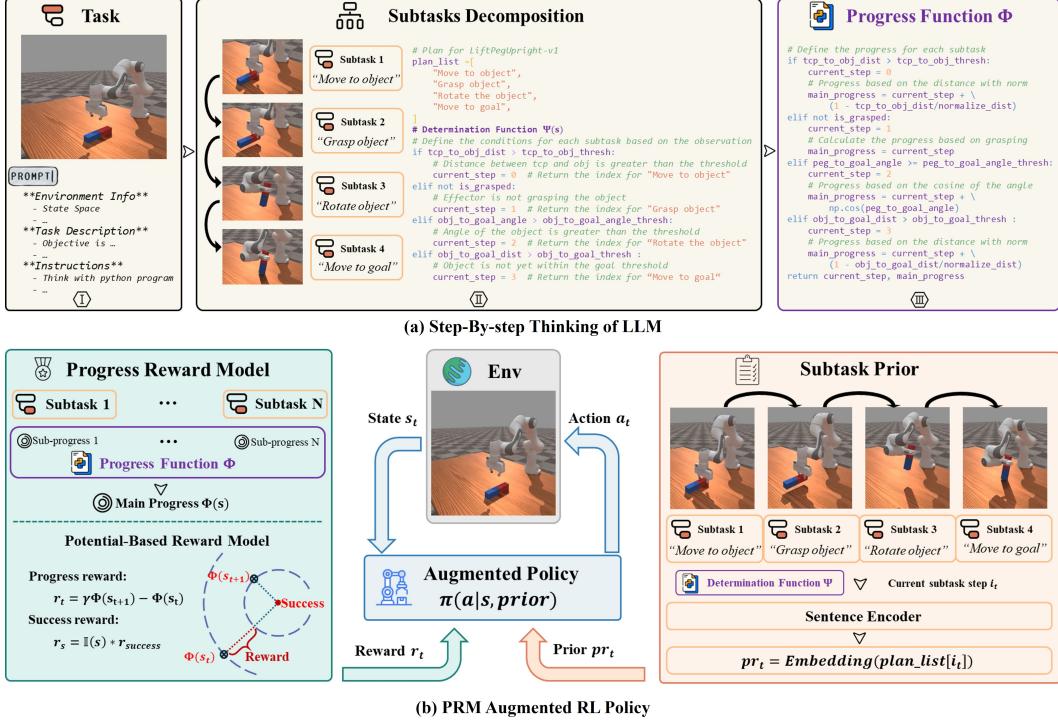


Figure 2: An overview of our framework. (a) We prompt the LLM using a Chain-of-Thought approach in Python code format. The reasoning process is provided as comments (e.g., using `#`), while the intermediate steps and final response are represented as executable Python statements. (b) Our framework augments RL in two ways: subtask prior and the PRM rewards.

120 **LLM usage pattern.** In our framework, we prompt an LLM to decompose the tasks and generate the  
 121 determination function  $\Psi$  and progress function  $\Phi$ . Considering that the intermediate inference steps  
 122 and the required final response are primarily represented as Python code (as shown later in Section  
 123 4.1, 4.2), inspired from PAL[38], we require the LLM to generate programs as intermediate steps, as  
 124 shown in Figure 2a. Specifically, we instruct the LLM to generate natural language reasoning steps  
 125 using comment syntax (e.g., `#` in Python), while the code itself is generated normally. Through  
 126 this program-aided Chain-of-Thought reasoning, the LLM can generate both the plan and progress  
 127 function in a single interaction.

128 **4.1 Subtask Decomposition**

129 Decomposing a long-term task into a series of short-term, manageable tasks provides the policy  
 130 with subtask priors and alleviates the complexity of constructing the progress function. Existing  
 131 methods[21, 22] have demonstrated that, given sufficient environment-related information, LLMs  
 132 are capable of task decomposition. In our framework, we leverage an LLM to break down the  
 133 task into a series of subtasks, each contributing to the success of the overall task. We construct  
 134 a Pythonic representation prompt that includes semantically rich contextual information such as  
 135 state space, environmental descriptions, and task descriptions (detailed in Appendix E), providing a  
 136 high-dimensional abstraction of environment-specific information.

137 While decomposing the subtasks, the LLM is also required to generate a natural language form  
 138 *subtask prior*, which is a description of the corresponding subtask. To ensure consistency and  
 139 generalization across the robotics control domain, we designed a fixed pattern for the subtask priors.  
 140 Specifically, we instruct the LLM to output each subtask prior in a "verb + noun" format (e.g., 'Move  
 141 to object' or 'Grasp object'), significantly enhancing the generalization ability of the trained policy.  
 142 The expected output from the LLM is a Python list, called *plan\_list*, as shown in Figure 2a(II), where  
 143 each element corresponds to a subtask prior.

---

**Algorithm 1** PRM4RL

---

```
Require: pythonic LLM prompt  $prompt$ , sentence encoder  $\mathbf{E}$ , policy  $\pi_\theta$ , MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ , RL optimization algorithm  $Optimize\_Algo$ 
// Request LLM
 $plan\_list, \Psi(s), \Phi(s) = LLM(prompt)$ 
// Augment State Space
 $\mathcal{P} : prior = \mathbf{E}(plan\_list[\Psi(s)])$ 
// Augment Reward Function
 $R^{PRM} = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t) + \mathbb{I}(s_{t+1}) \cdot r_{bonus}$ 
// Optimizing policy under augmented MDP
 $\mathcal{M}' = \langle \mathcal{S} + \mathcal{P}, \mathcal{A}, \mathcal{T}, R^{PRM}, \gamma, \rho_0 \rangle$ 
 $\theta^* = Optimize\_Algo(\mathcal{M}')$ 
```

---

144 However, guiding the policy with subtask priors presents another challenge: real-time tracking of  
145 which subtask the policy is currently executing. Previous approaches require calling the LLM at each  
146 timestep, either to regenerate a real-time plan[23] or to check the completion of ongoing subtasks[22],  
147 leading to significant time and computational overhead.

148 Our key insight is that, with a thorough understanding of the state space, we can determine which  
149 subtask the policy is in based solely on current state. For the previously generated  $plan\_list$ , we  
150 instruct the LLM to identify the completion condition for each subtask and locate the relevant features  
151 in the state space. We then construct a subtask determination function  $\Psi(s)$ , where given state  $s$ , it  
152 will return the index of current subtask in  $plan\_list$ . As shown in Figure 2a(II), the LLM ultimately  
153 generates an efficient, low-cost logical chain that ensures rigorous subtask determination.

154 **4.2 Potential-Based Progress Reward Model**

155 Our framework aims to construct a dense and efficient reward signal to guide the training of low-level  
156 policies. While previous approaches[24, 25, 26] directly design a reward function, we choose to  
157 model the progress of task execution instead. This approach is more task-relevant and provides  
158 stronger theoretical guarantees on optimality and convergence as proved later.

159 With the decomposition of subtasks, the focus of our progress function shifts from a long-term,  
160 complex task to short-term, simpler subtasks. This transformation simplifies task comprehension and  
161 the design of usable metrics. Based on the established subtask determination logic  $\Psi$ , we instruct the  
162 LLM to further define a progress function for each subtask, and then integrating these fine-grained  
163 sub-progress indicators into an overall evaluation of the task progress  $\Phi(s)$ , as shown in Figure 2a(III).  
164 The progress function offers a precise and efficient method for monitoring overall task execution.

165 Inspired by potential-based framework, we treat the progress function  $\Phi(s)$  as a form of potential and  
166 construct the Progress Reward Model (PRM) using the temporal difference form:

$$R_t^{PRM} = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t) + \mathbb{I}(s_{t+1}) \cdot r_{bonus} \quad (3)$$

167 where  $R_t^{PRM}$  is the dense reward given by PRM at timestep  $t$ ,  $\gamma$  is the discount factor, and the  
168 indicator function  $\mathbb{I}$  equals 1 when  $s_{t+1}$  meets the success conditions and 0 otherwise.  $\Phi(s_{t+1})$   
169 and  $\Phi(s_t)$  represent the progress at timesteps  $t+1$  and  $t$ , respectively. The PRM reward can be  
170 regarded as consisting of two parts: a sparse success reward and a potential-based reward shaping  
171 term. Following [39], we present the following theorem:

172 **Theorem 4.1** (Policy Invariance). *Define an MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ , where  $R_t = \mathbb{I}(s_{t+1}) \cdot r_{bonus}$  represents the sparse reward when the task is considered successful. Giving a potential-based  
173 reward shaping function as follows:*

$$F = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t)$$

175 *Let another MDP  $M' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R + F, \gamma, \rho_0 \rangle$ , then every optimal policy in  $M'$  will also be an  
176 optimal policy in  $M$  and vice versa.*

177 The proof of Theorem 4.1 is provided in Appendix A.1. Theorem 4.1 indicates that  $R^{PRM}$  does  
178 not distort the learning of the policy by introducing "deceptive" or "short-sighted" behaviors due to  
179 reward shaping. The optimal policy in  $M'$  remains focused on maximizing the original task objective,  
180 i.e. maximize success rate. Based on Theorem 4.1, we have the following corollary:

181 **Corollary 4.2** (Convergence Guarantee). *The introduction of potential-based reward shaping will  
182 not affect the convergence of reinforcement learning algorithms.*

183 Since the PRM reward has a theoretical convergence guarantee, we proceed to demonstrate its  
184 efficiency in convergence, following the work of [40] (Detailed proof provided in Appendix A.2).

185 **Theorem 4.3** (Convergence Efficiency). *Consider two MDPs:  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R + F, \gamma, \rho_0 \rangle$  with  
186  $Q$ -function  $Q$  and  $M' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ . If we initialize the  $Q$ -function in  $M'$  as follows:*

$$Q'_0(s, a) = Q_0(s, a) + \Phi(s)$$

187 *Then for all timesteps  $t \geq 0$  and  $(s, a) \in \mathcal{S} \times \mathcal{A}$ :*

$$Q'_t(s, a) = Q_t(s, a) + \Phi(s)$$

188 *where  $Q_t$  and  $Q'_t$  are the  $Q$ -functions of MDP  $M$  and  $M'$  respectively.*

189 The mathematical equivalence between PRM and  $Q$ -value initialization, as established in Theorem  
190 4.3, provides two fundamental insights into why PRM accelerates convergence. First, it demonstrates  
191 that PRM effectively provides prior knowledge-based initialization for value functions, directly  
192 reducing the initial approximation error  $|Q_0 - Q^*|$ . Second, the proof formally justifies that any  
193 well-designed potential function  $\Phi(s)$  serves as an actionable curriculum, where the shaping term  
194  $\gamma\Phi(s') - \Phi(s)$  progressively guides exploration toward high-value regions without altering optimal  
195 policies.

196 These theoretical results establish PRM not only as a heuristic reward design but also as a mathemati-  
197 cally grounded approach to guide the policy with optimality and convergence.

### 198 4.3 Reinforcement Learning Augmentation

199 Giving an MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ , our framework augment it through both high-level planning  
200 and low-level reward shaping.

201 At the high level, the policy cannot directly process the natural language form of the *plan\_list*. To  
202 address this, we employ SimCSE[41], a sentence encoder  $\mathbf{E}$ , to embed the subtask prior. Specifically,  
203 the subtask prior is expressed as:

$$\mathcal{P} : \text{prior} = \mathbf{E}(\text{plan\_list}[\Psi(s)])$$

204 where subtask prior is the embedding of current subtask corresponding to state  $s$ . The state space is  
205 then augmented to  $\mathcal{S} + \mathcal{P}$ . At the low level, we use the reward  $R^{PRM}$  from Section 4.2 to efficiently  
206 guide the learning of policy. Finally, the augmented MDP is expressed as:

$$\mathcal{M}' = \langle \mathcal{S} + \mathcal{P}, \mathcal{A}, \mathcal{T}, R^{PRM}, \gamma, \rho_0 \rangle$$

207 This formulation integrates both high-level planning through the subtask prior and low-level control  
208 via the PRM reward to guide policy learning. Consequently, we can utilize existing RL optimization  
209 algorithms to optimize the policy under the augmented MDP  $M'$ . In this paper, we use Soft Actor-  
210 Critic (SAC)[42] and Proximal Policy Optimization (PPO)[43].

## 211 5 Experiment

212 In this section, we evaluate PRM4RL on a diverse suite of long-term, sparse reward RL tasks, testing  
213 its ability to provide high-level plan and guide low-level training.

### 214 5.1 Experiment Settings

215 The framework is implemented based on the Stable-Baselines3 framework[44], ensuring consistency  
216 across all methods. We employ the same RL algorithm with identical training parameters for all  
217 methods. For all LLM-related requirements, we utilize OpenAI’s GPT-4o[45]. Details of the hyper-  
218 parameters are provided in Appendix C. To ensure robustness and reliability, all experiments are  
219 conducted with five different random seeds.

220 **Baselines.** ELLM(ICML23) [23] utilizes an LLM as a high-level planner, embedding subtask priors  
221 into the policy. It serves as an example of an LLM-planner approach. T2R(ICLR24) [24] combines

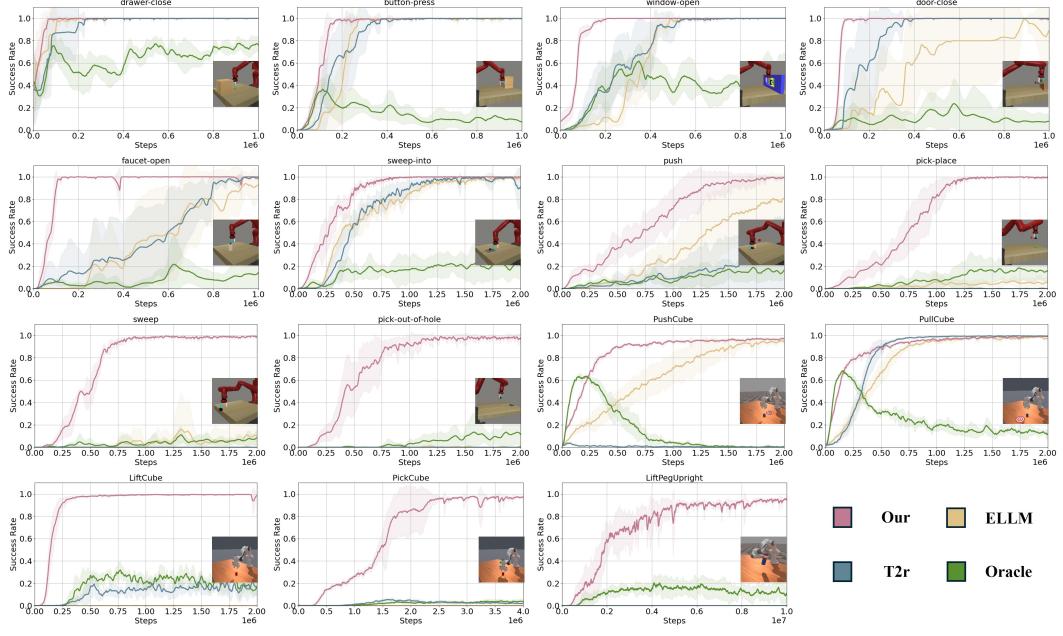


Figure 3: The learning curves of comparison methods on Metaworld and Maniskill measured by success rate. All experiments are conducted with 5 random seeds.

222 LLMs and human feedback to generate dense reward functions for policy training. We use it as an  
 223 example of an LLM-rewarder approach. Oracle is the expert-written dense reward function provided  
 224 by the environment. The details of the baseline settings are provided in Appendix B.2.

225 **Environments** MetaWorld[27] is an open-source simulated benchmark features a Sawyer robot  
 226 interacting with a tabletop setup that includes drawer, window, ball, faucet, door and many objects.  
 227 Maniskill[28] is an advanced robotics simulation platform designed for high-fidelity manipulation  
 228 tasks. Further details about the environment and the tasks are provided in Appendix D.

## 229 5.2 Evaluation Analysis

### 230 5.2.1 Outperform previous LLM approaches

231 As shown in Figures 3 and 4, our method consistently outperforms the comparison methods across  
 232 nearly all environments, demonstrating faster convergence and superior performance. We observe  
 233 the following advantages: (1) In complex long-term tasks such as MW-pick-out-of-hole and MS-  
 234 LiftPegUpright, the baseline methods achieve less than 15% success, while our method achieves  
 235 nearly 100% success, highlighting our ability to effectively solve challenging tasks. (2) When  
 236 compared to ELLM, our approach demonstrates a significantly improved convergence speed, which  
 237 underscores the efficiency of the Progress Reward Model in providing dense, theoretically guaranteed  
 238 rewards for the low-level control policy. This results in more stable and faster learning compared to  
 239 traditional LLM-based planning approaches. (3) In comparison with T2R, the subtask decomposition  
 240 in our framework plays a critical role in simplifying the coding process. By breaking down complex  
 241 tasks into more manageable subtasks, our method reduces the difficulty of understanding environment  
 242 dynamics and designing useful metrics, enabling us to solve complex tasks that T2R fails to address.  
 243 This advantage is particularly evident in tasks where the environment is more intricate and requires a  
 244 clear separation of the planning and reward stages.

### 245 5.2.2 Ablation Study

246 To further validate the effectiveness of the two augmentations in our framework, we conduct an  
 247 ablation study by constructing three additional baselines:

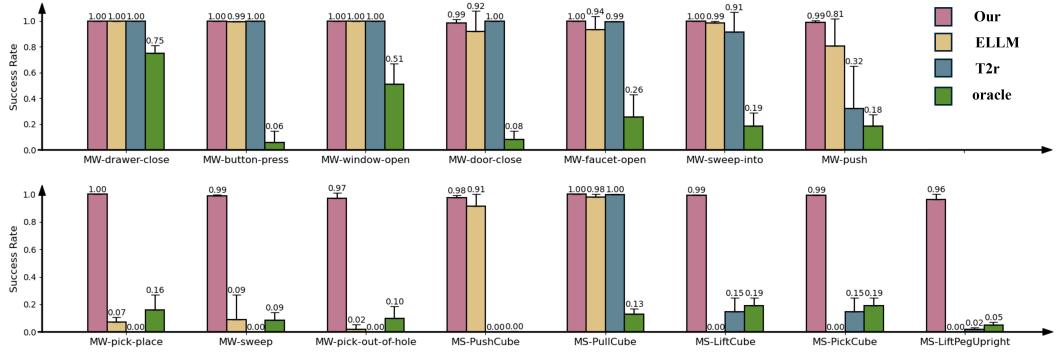


Figure 4: The evaluation results of comparison methods on Metaworld(With prefix ‘MW’) and Maniskill(with prefix ‘MS’) measured by success rate. All experiments are conducted with 5 random seeds.

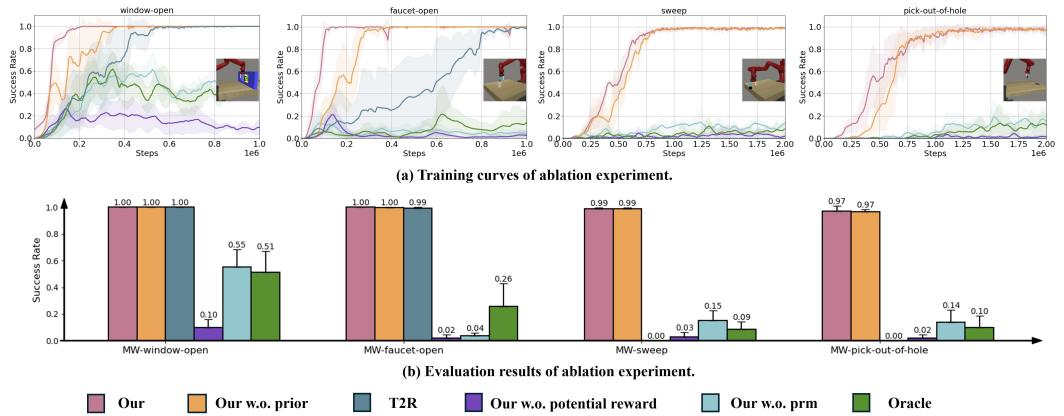


Figure 5: The evaluation results of ablation study on Metaworld measured by success rate. All experiment is conducted with 5 random seeds.

248 **w.o. PRM reward** (denoted as ‘our w.o. prm’): The PRM reward is  
249 replaced with the oracle reward, while retaining the subtask prior.

250 **w.o. subtask prior** (denoted as ‘our w.o. prior’): No subtask prior is  
251 provided, while retaining the PRM reward.

252 **w.o. potential-based reward shaping** (denoted as ‘our w.o. potential  
253 reward’): The progress function is used directly as the reward, while  
254 retaining the subtask prior.

255 The results are shown in Figure 5. We observe the following findings:

256 (1) Compared with ‘our w.o. prm’, replacing the PRM reward with the  
257 oracle reward leads to a significant drop in performance. This demonstrates  
258 that, without an efficient and effective reward signal, directly feeding the  
259 subtask prior to the policy does not improve its performance. In most  
260 cases, the training curves and evaluation results of ‘our w.o. prm’ are  
261 similar to those of the oracle, with some instances where adding the  
262 subtask prior even causes a slight performance decrease.

263 (2) The performance of ‘our w.o. prior’ shows that abandoning the subtask  
264 prior results in slower convergence in the training curves. This highlights the role of the subtask

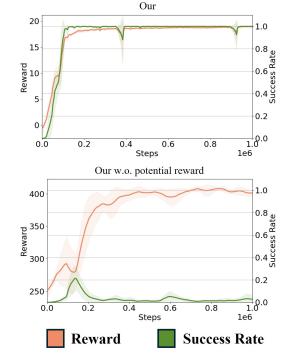


Figure 6: Training details of our and our w.o. potential reward.

265 prior in enabling efficient convergence for complex, long-term tasks. Moreover, ‘our w.o. prior’ still  
 266 outperforms T2R, further demonstrating the effectiveness of the PRM reward in guiding the policy.

267 (3)The performance of ‘our w.o. potential reward’ further demonstrate the effectiveness of our  
 268 potential-based reward model, where directly using progress as reward signal lead to a significant  
 269 drop in performance. We also analyzed the anomalous behavior of ‘our w.o. potential reward’, where  
 270 the success rate initially rises but then declines as training progresses. As shown in Figure 6, the  
 271 episode reward increases, while the success rate paradoxically decreases. This suggests that the using  
 272 progress function directly as reward function suffers from inefficiencies, as there are instances where  
 273 the task fails despite receiving a high reward, leading the policy to learn in an undesirable direction.  
 274 In contrast, our method ensures stable convergence, with the PRM providing well-structured and  
 275 task-relevant rewards with optimality and convergence guarantees.

276 In summary, the two enhancements in our framework—the subtask prior and the PRM reward—work  
 277 synergistically to produce a "1+1 > 2" effect. Without the subtask prior, convergence speed is hindered,  
 278 and without the reward signal, providing effective and efficient guidance becomes challenging.

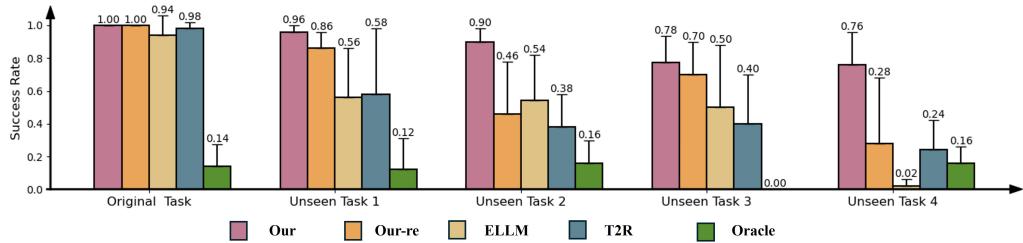


Figure 7: The evaluation results of generalization experiment. The results are averaged on 5 random seeds and 100 evaluations per seed.

### 279 5.2.3 Generalization Results

280 To evaluate the generalization capabilities of the trained policy, we test the trained policy’s perfor-  
 281 mance on unseen tasks in MetaWorld. Additional results and detailed information regarding the  
 282 tasks are provided in Appendix B.1. As shown in Figure 7, while most methods perform well on the  
 283 original task, we observe a performance decline on the unseen tasks. In comparison to other methods,  
 284 our framework experiences the least performance loss, demonstrating relatively strong generalization  
 285 abilities. Notably, when comparing to our ‘w.o. prior’ (our framework without the subtask prior),  
 286 we observe a significant drop in performance on unseen tasks. This highlights the critical role of the  
 287 subtask prior in enhancing generalization. The “verb + noun” pattern we designed effectively captures  
 288 commonalities across different tasks, acting as a powerful prior that guides the policy to perform well  
 289 on unseen tasks. Thus, the subtask prior serves as a robust tool for improving the generalization of  
 290 reinforcement learning policies across a wide range of tasks.

## 291 6 Conclusion

292 In this paper, we introduced the Progress Reward Model for Reinforcement Learning (PRM4RL),  
 293 a novel framework that integrates high-level planning and low-level reward shaping to address the  
 294 challenges of long-term planning and sparse rewards in reinforcement learning. At the high level, we  
 295 decompose a complex task into manageable subtasks with subtask prior. At the low level, we design  
 296 a Progress Reward Model with optimality and convergence guarantee. The experimental results  
 297 demonstrate the effectiveness and efficiency of our method, significantly improving both convergence  
 298 speed and the ability to solve complex, multi-stage tasks.

299 **Limitations.** Similar to T2R[24] and other methods[23, 21, 22] that depend on LLMs for planning  
 300 or coding, our approach could be affected by LLM hallucinations during reasoning[46]. Besides,  
 301 while our method demonstrates strong performance across various tasks, its applicability to domains  
 302 outside of robotics remains unexplored. We anticipate that further optimization and adaptation of the  
 303 framework will be necessary to address diverse environments and tasks.

304 **References**

- 305 [1] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1.  
306 MIT press Cambridge, 1998.
- 307 [2] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal,  
308 and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and  
309 analysis. *Machine Learning*, 110(9):2419–2468, 2021.
- 310 [3] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical  
311 reinforcement learning. *Advances in neural information processing systems*, 31, 2018.
- 312 [4] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical rein-  
313 forcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35,  
314 2021.
- 315 [5] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges,  
316 methods and progress. *Artificial Intelligence*, 297:103500, 2021.
- 317 [6] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy  
318 inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.
- 319 [7] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal  
320 control via policy optimization. In *International conference on machine learning*, pages 49–58.  
321 PMLR, 2016.
- 322 [8] Shanchuan Wan, Yujin Tang, Yingtao Tian, and Tomoyuki Kaneko. Deir: efficient and robust  
323 exploration through discriminative-model-based episodic intrinsic rewards. *arXiv preprint*  
324 *arXiv:2304.10770*, 2023.
- 325 [9] Anna Penzkofer, Simon Schaefer, Florian Strohm, Mihai Bâce, Stefan Leutenegger, and Andreas  
326 Bulling. Int-hrl: towards intention-based hierarchical reinforcement learning. *Neural Computing*  
327 and Applications
- 328 [10] Xinlei Pan, Eshed Ohn-Bar, Nicholas Rhinehart, Yan Xu, Yilin Shen, and Kris M Kitani.  
329 Human-interactive subgoal supervision for efficient inverse reinforcement learning. *arXiv*  
330 *preprint arXiv:1806.08479*, 2018.
- 331 [11] Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. A survey on recent  
332 advances in llm-based multi-turn dialogue systems. *arXiv preprint arXiv:2402.18013*, 2024.
- 333 [12] Ethan Brooks, Logan Walls, Richard L Lewis, and Satinder Singh. Large language models can  
334 implement policy iteration. *Advances in Neural Information Processing Systems*, 36:30349–  
335 30366, 2023.
- 336 [13] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel:  
337 Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial*  
338 *Intelligence*, volume 38, pages 19632–19642, 2024.
- 339 [14] Yuanzhao Zhai, Tingkai Yang, Kele Xu, Dawei Feng, Cheng Yang, Bo Ding, and Huaimin  
340 Wang. Enhancing decision-making for llm agents via step-level q-value models. In *Proceedings*  
341 *of the AAAI Conference on Artificial Intelligence*, volume 39, pages 27161–27169, 2025.
- 342 [15] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan,  
343 Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation  
344 models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- 345 [16] Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin,  
346 Weishan Zhao, Li Zhu, Hari Sundaram, et al. Codescope: An execution-based multilingual mul-  
347 titask multidimensional benchmark for evaluating llms on code understanding and generation.  
348 *arXiv preprint arXiv:2311.08588*, 2023.
- 349 [17] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan  
350 Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint*  
351 *arXiv:2412.15115*, 2024.

- 352 [18] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan  
 353 Cao. React: Synergizing reasoning and acting in language models. In *International Conference  
 354 on Learning Representations (ICLR)*, 2023.
- 355 [19] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,  
 356 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in  
 357 llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- 358 [20] Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Yue Chen, Guolong Liu, Gaoqi Liang, Junhua  
 359 Zhao, Jinyue Yan, and Yun Li. Survey on large language model-enhanced reinforcement  
 360 learning: Concept, taxonomy, and methods, 2024.
- 361 [21] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel  
 362 Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding  
 363 language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.
- 364 [22] Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. Plan-seq-  
 365 learn: Language model guided rl for solving long horizon robotics tasks. *arXiv preprint  
 366 arXiv:2405.01534*, 2024.
- 367 [23] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek  
 368 Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language  
 369 models. In *International Conference on Machine Learning*, pages 8657–8677. PMLR, 2023.
- 370 [24] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang,  
 371 and Tao Yu. Text2reward: Automated dense reward function generation for reinforcement  
 372 learning. In *International Conference on Learning Representations (ICLR), 2024 (07/05/2024-  
 373 11/05/2024, Vienna, Austria)*, 2024.
- 374 [25] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh  
 375 Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design  
 376 via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
- 377 [26] Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires.  
 378 Revolve: Reward evolution with large language models using human feedback. *arXiv preprint  
 379 arXiv:2406.01309*, 2024.
- 380 [27] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and  
 381 Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement  
 382 learning. In *Conference on Robot Learning (CoRL)*, 2019.
- 383 [28] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao,  
 384 Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav  
 385 Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto  
 386 Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation  
 387 and rendering for generalizable embodied ai. *Robotics: Science and Systems*, 2025.
- 388 [29] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit  
 389 Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models.  
 390 *arXiv preprint arXiv:2311.05772*, 2023.
- 391 [30] Vishnu Sarukkai, Brennan Shacklett, Zander Majercik, Kush Bhatia, Christopher Ré, and  
 392 Kayvon Fatahalian. Automated rewards via llm-generated progress functions, 2024.
- 393 [31] Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wicher, Yinxiao Liu, and Lei Meng. Drlc:  
 394 Reinforcement learning with dense rewards from llm critic. *arXiv e-prints*, pages arXiv-2401,  
 395 2024.
- 396 [32] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and  
 397 Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks,  
 398 2023.
- 399 [33] Quan Yuan, Mehran Kazemi, Xin Xu, Isaac Noble, Vaiva Imbrasaite, and Deepak Ramachandran.  
 400 Tasklama: Probing the complex task understanding of language models, 2023.

- 401 [34] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. Nl2tl: Transforming natural  
402 languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766*,  
403 2023.
- 404 [35] Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan.  
405 Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In  
406 *2024 IEEE International conference on robotics and automation (ICRA)*, pages 6695–6702.  
407 IEEE, 2024.
- 408 [36] Rati Devidze, Parameswaran Kamalaruban, and Adish Singla. Informativeness of reward  
409 functions in reinforcement learning, 2024.
- 410 [37] Jonas Eschmann. Reward function design in reinforcement learning. *Reinforcement learning  
411 algorithms: Analysis and Applications*, pages 25–33, 2021.
- 412 [38] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan,  
413 and Graham Neubig. Pal: Program-aided language models. In *International Conference on  
414 Machine Learning*, pages 10764–10799. PMLR, 2023.
- 415 [39] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transforma-  
416 tions: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer,  
417 1999.
- 418 [40] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of  
419 Artificial Intelligence Research*, 19:205–208, September 2003.
- 420 [41] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence  
421 embeddings, 2022.
- 422 [42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy  
423 maximum entropy deep reinforcement learning with a stochastic actor, 2018.
- 424 [43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal  
425 policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- 426 [44] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah  
427 Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of  
428 Machine Learning Research*, 22(268):1–8, 2021.
- 429 [45] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark,  
430 AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv  
431 preprint arXiv:2410.21276*, 2024.
- 432 [46] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi  
433 Li, and Yuchi Ma. Exploring and evaluating hallucinations in llm-powered code generation.  
434 *arXiv preprint arXiv:2404.00971*, 2024.

435 **A Theoretical Proof**

436 **Theorem A.1** (Policy Invariance). Define an MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ , where  $R_t = \mathbb{I}(s_{t+1}) \cdot$   
 437  $r_{\text{bonus}}$  represents the sparse reward when the task is considered successful. Giving a potential-based  
 438 reward shaping function as follows:

$$F = \gamma \cdot \Phi(s_{t+1}) - \Phi(s_t)$$

439 Let another MDP  $M' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R + F, \gamma, \rho_0 \rangle$ , then every optimal policy in  $M'$  will also be an  
 440 optimal policy in  $M$  and vice versa.

441 *Proof.* For the original MDP  $M$ , we know that its optimal  $Q$ -function  $Q_M^*$  satisfies the Bellman  
 442 Equations

$$Q_M^*(s, a) = \mathbb{E}_{s' \sim P_{s,a}(\cdot)} \left[ R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_M^*(s', a') \right]$$

443 Some simple algebraic manipulation then gives us

$$Q_M^*(s, a) - \Phi(s) = \mathbb{E}_{s'} \left[ R(s, a, s') + \gamma \Phi(s') - \Phi(s) + \gamma \max_{a' \in \mathcal{A}} (Q_M^*(s', a') - \Phi(s')) \right]$$

444 If we now define  $\hat{Q}_{M'}(s, a) \triangleq Q_M^*(s, a) - \Phi(s)$  and substitute that and  $F(s, a, s') = \gamma \Phi(s') - \Phi(s)$   
 445 back into the previous equation, we get

$$\begin{aligned} \hat{Q}_{M'}(s, a) &= \mathbb{E}_{s'} \left[ R(s, a, s') + F(s, a, s') + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_{M'}(s', a') \right] \\ &= \mathbb{E}_{s'} \left[ R'(s, a, s') + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_{M'}(s', a') \right] \end{aligned}$$

446 But this is exactly the Bellman equation for  $M'$ . For the undiscounted case, we moreover have  
 447  $\hat{Q}_{M'}(s_0, a) = Q_M^*(s_0, a) - \Phi(s_0) = 0 - 0 = 0$ . So,  $\hat{Q}_{M'}(s, a)$  satisfies the Bellman equations  
 448 for  $M'$ , and must in fact be the unique optimal  $Q$ -function. Thus,  $Q_{M'}^*(s, a) = \hat{Q}_{M'}(s, a) =$   
 449  $Q_M^*(s, a) - \Phi(s)$  and the optimal policy for  $M'$  therefore satisfies.

$$*pi_{M'}^*(s) \in \arg \max_{a \in \mathcal{A}} Q_{M'}^*(s, a) \quad (4)$$

$$= \arg \max_{a \in \mathcal{A}} Q_M^*(s, a) - \Phi(s) \quad (5)$$

$$= \arg \max_{a \in \mathcal{A}} Q_M^*(s, a) \quad (6)$$

450 \* and is therefore also optimal in  $M$ . To show every optimal policy in  $M$  is also optimal in  $M'$ , simply  
 451 apply the same proof with the roles of  $M$  and  $M'$  interchanged (and using the shaping function  $-F$ ).  
 452 This completes the proof.  $\square$

453 **Theorem A.2** (Convergence Efficiency). Consider two MDPs:  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R + F, \gamma, \rho_0 \rangle$  with  
 454  $Q$ -function  $Q$  and  $M' = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0 \rangle$ . If we initialize the  $Q$ -function in  $M'$  as follows:

$$Q'_0(s, a) = Q_0(s, a) + \Phi(s)$$

455 Then for all timesteps  $t \geq 0$  and  $(s, a) \in \mathcal{S} \times \mathcal{A}$ :

$$Q'_t(s, a) = Q_t(s, a) + \Phi(s) \quad (7)$$

456 where  $Q_t$  and  $Q'_t$  are the  $Q$ -functions of MDP  $M$  and  $M'$  respectively.

457 *Proof.* We prove the equivalence by induction on the update steps.

458 **Base case ( $t = 0$ ):** By initialization definition:

$$Q'_0(s, a) = Q_0(s, a) + \Phi(s) \quad \forall (s, a) \quad (8)$$

459 **Inductive step:** Assume  $Q'_k(s, a) = Q_k(s, a) + \Phi(s)$  holds for all  $(s, a)$  at step  $k$ . Consider the  
 460 Q-learning update for both agents when transitioning from  $(s, a)$  to  $(s', a')$  with reward  $r$ :

461 In MDP  $M$ :

$$\begin{aligned} Q_{k+1}(s, a) &= Q_k(s, a) + \alpha_k \left[ r + F(s, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right] \\ &= Q_k(s, a) + \alpha_k \left[ r + (\gamma \Phi(s') - \Phi(s)) + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right] \end{aligned}$$

462 In MDP  $M'$ :

$$\begin{aligned} Q'_{k+1}(s, a) &= Q'_k(s, a) + \alpha_k \left[ r + \gamma \max_{a'} Q'_k(s', a') - Q'_k(s, a) \right] \\ &= [Q_k(s, a) + \Phi(s)] + \alpha_k \left[ r + \gamma \max_{a'} [Q_k(s', a') + \Phi(s')] - [Q_k(s, a) + \Phi(s)] \right] \\ &= Q_k(s, a) + \Phi(s) + \alpha_k \left[ r + \gamma \max_{a'} Q_k(s', a') + \gamma \Phi(s') - Q_k(s, a) - \Phi(s) \right] \end{aligned}$$

463 Simplify the difference:

$$\begin{aligned} Q'_{k+1}(s, a) - Q_{k+1}(s, a) &= \Phi(s) + \alpha_k [\gamma \Phi(s') - \Phi(s)] - \alpha_k [\gamma \Phi(s') - \Phi(s)] \\ &= \Phi(s) \end{aligned}$$

464 Thus  $Q'_{k+1}(s, a) = Q_{k+1}(s, a) + \Phi(s)$  maintains the invariant. By induction, the equivalence holds  
 465 for all  $t \geq 0$ .  $\square$

## 466 B Experiment Details

### 467 B.1 Additional Generalization Results

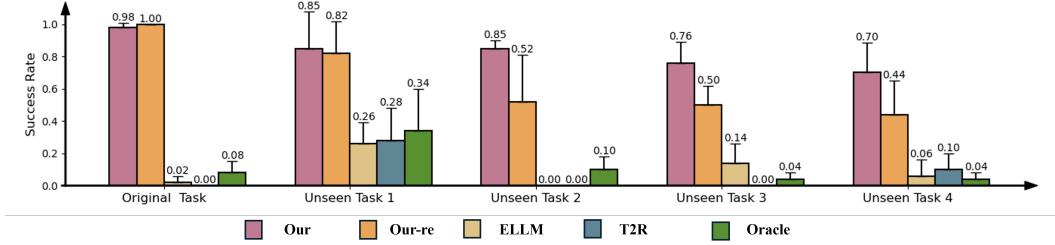


Figure 8: The evaluation results of generalization experiment.

468 We conduct additional generalization evaluations. Specifically, for the evaluation results shown  
 469 in Section 5.2.3, the original task is ‘Metaworld-faucet-open’, the unseen tasks are ‘Metaworld-  
 470 drawer-close’, ‘Metaworld-button-press-wall’, ‘Metaworld-button-press’, ‘Metaworld-coffee-button’,  
 471 respectively. For the results shown in Figure B.1, the original task is ‘Metaworld-pick-place’, the  
 472 unseen tasks are ‘Metaworld-drawer-close’, ‘Metaworld-pick-place-wall’, ‘Metaworld-sweep-into’,  
 473 ‘Metaworld-handle-press’, respectively. We test the trained policy in evaluation tasks, and the results  
 474 are averaged over 5 seeds and 100 test per seed.

### 475 B.2 Baseline Settings

476 **ELLM**(ICML23)[23] utilizes an LLM as a high-level planner, embedding subtask priors into the  
 477 policy. Since ELLM has not been previously tested on Metaworld and Maniskill, we re-implement it  
 478 the experiments. For goal embedding, we gave the same subtask prior as in our framework to the  
 479 policy; For reward, we then gave it the subtask success reward, which is also goal-related as in its  
 480 original application. We also use our  $\Psi$  for determining current subtask to avoid enormous LLM  
 481 calls.

482 **Text2Reward**(ICLR24)[24] utilizes an LLM to write dense reward function. As T2R has evaluate  
 483 its performance on Metaworld and Maniskill, we directly reuse its generated reward functions. If  
 484 there are few-shot and zero-shot versions of reward functions, we always choose the few-shot version  
 485 for better performance. Specifically, for ‘Metaworld-drawer-close’, ‘Metaworld-window-open’,  
 486 ‘Metaworld-button-press’, ‘Metaworld-sweep-into’, ‘Metaworld-door-close’, ‘Maniskill-LiftCube’,  
 487 ‘Maniskill-PickCube’, we reuse its generated reward functions. For other tasks that are not orginally  
 488 evaluated in T2R, we generate a reward function following its code.

489 **C Hyper-Parameter Details**

490 In this section, we provide the hyper-parameter details used for our reward function generation and  
 491 reinforcement learning backbones. For progress function generation, we base on GPT-4o. In the  
 492 experiments of the main body, the temperature of sampling is set to 0.7 for each experiment.  
 493 For reinforcement learning training, we employed the open-source PPO[43] and SAC  
 494 implementation[42] from Stable-Baselines3[44]<sup>1</sup>, and list the hyper-parameters in Table C and C.

Table 1: Hyper-parameter of SAC algorithm applied to each task.

Hyper-parameter	Value
Discount factor $\gamma$	0.99
Target update frequency	2
Learning rate	$3 \times 10^{-4}$
Train frequency	1
Soft update $\tau$	$5 \times 10^{-3}$
Gradient steps	1
Learning starts	4000
Batch Size	512
# of layers	3
Hidden units per layer	256
Initial temperature	0.1
Rollout steps per episode	500

Table 2: Hyper-parameter of PPO algorithm applied to each task.

Hyper-parameter	Value
Discount factor $\gamma$	0.8
# of epochs per update	15
Learning rate	$1 \times 10^{-4}$
# of environments	8
Batch size	400
Target KL divergence	0.1
# of layers	3
Hidden units per layer	256
# of steps per update	3200

495 We utilize a 4 Nvidia Geforce RTX-3090 graphic cards, 128 core CPUs, and 256 GiB memory server  
 496 for RL training. The time required for training a policy is approximately 1 hours per task (5 seeds  
 497 running simultaneouly) for MetaWorld, and between 1 to 10 hours for ManiSkill, varying with the  
 498 task’s difficulty.

<sup>1</sup>stable-baselines3 v2.6.0 (MIT License), code available at <https://github.com/DLR-RM/stable-baselines3>

499 **D Tasks**

500 In this section, we provide a full list of tasks within each simulation environment, accompanied by  
 501 their corresponding language instructions. Across all tasks, we follow the default settings of their

502 **D.1 Metaworld**

503 In METAWORLD[27]<sup>2</sup> environment, we use a 7 DoF Sawyer robot arm with a fixed base to complete  
 504 tabletop tasks. For all tasks, the observation space is a combination of the 3D position of the robot  
 505 end-effector, a normalized measurement of gripper openness, the 3D position of the manipulated  
 506 object, the quaternion of the object, all of the previous measurements, and the goal position. The  
 507 environment adopts end-effector delta position control, which means the action space consists of the  
 508 change of the end effector's 3D position, as well as the normalized torque the gripper should apply.  
 509 For all tasks, the initial and target positions of the manipulated object and the initial joint positions of  
 510 the robot arm are variable.

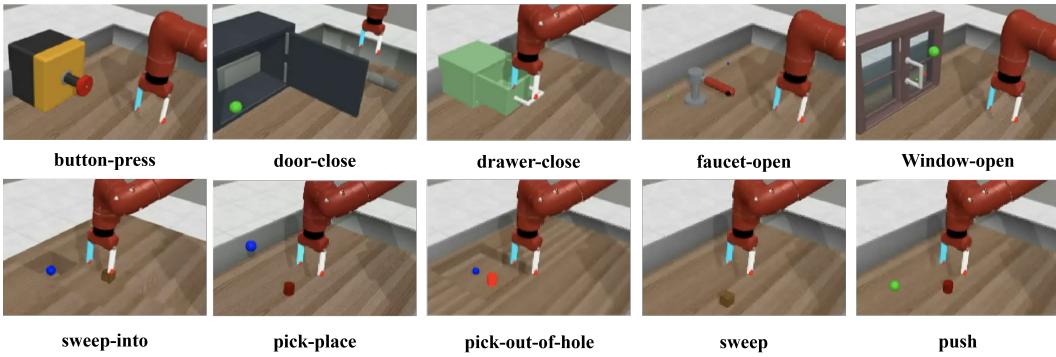


Figure 9: Tasks in Metaworld.

The task descriptions are provided in the following Table D.1.

Table 3: Task list of Metaworld

Task	Task Description
button-press	Press a button in y coordination.
door-close	Close a door with a revolving joint by pushing the door's handle.
drawer-close	Close a drawer by its handle.
faucet-open	Rotate the handle counter-clockwise.
window-open	Push and open a sliding window by its handle.
sweep-into	Sweep a puck from the initial position into a hole.
pick-place	Pick up an object and move it to the goal location.
pick-out-of-hole	Pick an object out of a hole and move it to the goal location.
sweep	Sweep a puck off the table.
push	Push an object to the goal location.

511

512 **D.2 Maniskill**

513 MANISKILL[28]<sup>3</sup> environment uses a 7 DoF Franka Panda as the default robot arm. For all tasks, the  
 514 observation space consists of robot proprioception information (e.g. current joint positions, current  
 515 joint velocities, robot base position and quaternion in the world frame) and task-specific information

<sup>2</sup>Metaworld v2.0.0 (MIT License), code available at <https://github.com/Farama-Foundation/Metaworld>

<sup>3</sup>Maniskill3 v3.0.0 (Apache-2.0 License), code available at <https://github.com/haosulab/ManiSkill>

516 (e.g. goal position, endeffector position). We use end-effector delta pose control mode for this  
 517 environment, which controls the change of 3D position and rotation (represented as an axis-angle in  
 518 the end-effector frame). For all tasks, the initial and target positions of the manipulated object, the  
 519 initial joint positions of the robot arm and physical parameters (e.g. friction and damping coefficient)  
 520 are variable.

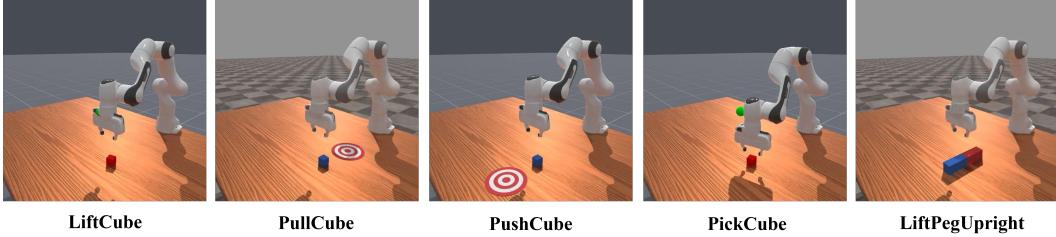


Figure 10: Tasks in Maniskill.

The task descriptions are provided in the following Table D.2.

Table 4: Task list of Maniskill

Task	Task Description
LiftCube	Pick up cube A and lift it up by 0.2 meters.
PullCube	Pull a cube to the goal position.
PushCube	Push a cube to the goal position.
PickCube	Pick up cube A and move it to the 3D goal position.
LiftPegUpright	Move a peg laying on the table to any upright position on the table.

521

## 522 E Prompt Details

523 To ground LLMs into robotics simulation environments, following Text2Reward[24] we propose  
 524 a Pythonic prompt, which can be abstracted by the experts who developed the environment. This  
 525 class-like prompt is a more compact representation than simply listing all environment attributes  
 526 linearly, which can delete redundant information and save more tokens, and this Pythonic prompt can  
 527 also better bootstrap Python code generation. More specifically, our prompt leverages python class,  
 528 class attribute, python typing and comments to recursively define the environment. For simplicity,  
 529 here we prompt the progress function to return the current subtask index together with the main  
 530 progress.

531 Here we provide an example of the zero-shot prompt for Metaworld manipulation tasks:

532 **Basic Prompt** provides the foundational context for the task. The prompt establishes the user’s role  
 533 as an expert in robotics, reinforcement learning, and code generation, focusing on the development of  
 534 a Python function called ‘progress function’ for RL. The objective is to help generate code that tracks  
 535 progress during task execution, guiding the learning of a low-level policy.

### Basic Prompt

You are an expert in robotics, reinforcement learning and code generation. We are going to use a robot arm to complete given tasks. Now I want you to help me write a python function named ‘progress function’ of reinforcement learning.

536

537 **Environment Information** defines the environment in which the robot operates. The BaseEnv class  
 538 describes the robot and its interaction with objects in the environment, including their positions and  
 539 states. The Robot and RigidObject classes further define the robot’s end effector, object positions,  
 540 and other relevant parameters. This context provides essential details for the task, enabling the LLM  
 541 to make informed decisions about task progress and control.

## Environment Information

```
class BaseEnv(gym.Env):
    self.robot : Robot # the robot in the environment
    self.obj : RigidObject # the first object in the environment
    self.goal_position : np.ndarray[(3,)] # indicate the 3D
        position of the goal
    near_object = float(tcp_to_obj <= 0.3)
    success = float(obj_to_target <= 0.07)

class Robot:
    self.ee_position : np.ndarray[(3,)] # indicate the 3D
        position of effector
    self.hand_init_pos: np.ndarray[(3,)] # indicate the initial
        3D position of effector
    self.tcp = self.ee_position - [0, 0, 0.045] # indicate the
        3D position of effector's tool center point

class RigidObject:
    self.position : np.ndarray[(3,)] # indicate the 3D position
        of the rigid object
    self.quaternion : np.ndarray[(4,)] # indicate the quaternion
        of the rigid object
    self.obj_init_pos : np.ndarray[(3,)] # indicate the initial
        3D position of the rigid object
```

542

543 **Task Information** provides a description of the task to be completed by the robot. The task  
544 description, which needs to be filled in by the user, is crucial for decomposing the task into manageable  
545 subtasks.

## Task Information

{ Fill in the task description here. } (e.g. In pick-place, the robot need to pick up the object  
and move it to the goal position.)

546

547 **Basic Instructions** provide guidelines for writing the code necessary for the task. The instructions  
548 emphasize the importance of using existing Python packages only when necessary, avoiding the  
549 invention of new variables or attributes, and incorporating step-by-step reasoning with clear comments.  
550 These instructions ensure that the code remains efficient, understandable, and aligned with the task's  
551 requirements.

## Basic Instruction

1. You are allowed to use any existing python package if applicable. But only use these packages when it's really necessary.
2. Do not invent any variable or attribute that is not given.
3. Think step by step, add comment for reasoning and thought when you write code.

552

553 **Reasoning Instructions** outline a step-by-step procedure for task decomposition and the generation  
554 of key functions.

## Reasoning Instructions

Follow the steps below:

- a. Decompose the task and generate plan\_list
- a.1 Decompose the given task into a series of subtasks. Each subtask should be described in a

555

‘verb + noun’ form. tips: ‘move to object’, ‘grasp object’, ‘move to goal’...

a.2 Generate a python list named plan\_list where each subtask is an element of the plan\_list.

b. Write a ‘determination\_function’ for determine which subtask we are in based on Environment Information above.

b.1 For each subtask in the plan\_list, consider when each subtask is considered completed, express the condition in python code form. Typical completion conditions: -‘Move’ completion: distance under threshold -‘Grasp’ completion: object’s z-coordinate higher than its initialize z-coordinate.

b.2 Write a ‘determination\_function’ using ‘if’ and ‘elif’ to categorize the current subtask based on Environment Information - consider the logic of determining subtasks completion, using ‘if not’ ‘elif not’ if needed.

b.3 The ‘determination\_function’ return the current subtask’s index in plan\_list.

c. Write a ‘progress\_function’

c.1 For each subtask, construct a progress metric named ‘subprogress’ indicating the progress of the corresponding subtask. The ‘subprogress’ should be range from 0 (no progress) to 1 (completed) Typical subprogress metric: the distance between the robot’s gripper and target object normalized by its initial distance;

c.2 Based on the ‘determination\_function’, add the ‘subprogress’ computation codes in the corresponding ‘if’ or ‘elif’ block.

c.3 Add the ‘main\_progress’ computation codes in each subtask’s ‘if’ or ‘elif’ block. -‘main\_progress’ is the progress of the entire task. - typically, main\_progress can be calculated by adding the current subtask idx and current subprogress, regarding all completed subprogress as ‘1’. The completed progress can be set to ‘1’ without computing again.

c.4 ‘progress function’ return the main\_progress and the current subtask’s index in plan\_list.

556

## 557 F Example of Generated Progress Function

558 In this section, to help better understand the generation ability and results, we provide zero-shot  
 559 generated reward function on Metaworld. The response of the prompt defined in Appendix E is as  
 560 follows:

LLM response in Metaworld-pick-place

```
import numpy as np
# Plan the task into subtasks
def generate_plan_list():
    # Task decomposition: [verb + noun]
    plan_list = [
        'move_to_object',
        'grasp_object',
        'move_to_goal',
        'release_object'
    ]
    return plan_list

# Determining the current subtask based on environment
# information
def determination_function(env, plan_list):
    # Decompose the task completion into conditions for each
    # subtask

    # Subtask 1: 'move to object'
    if np.linalg.norm(env.robot.tcp - env.obj.position) > 0.3:
        return 0 # Still moving towards the object

    # Subtask 2: 'grasp object'
    elif env.obj.position[2] <= env.obj.obj_init_pos[2]:
```

561

```

        return 1 # Object not yet grasped

    # Subtask 3: 'move to goal'
    elif np.linalg.norm(env.obj.position - env.goal_position) >
        0.07:
        return 2 # Still moving towards the goal

    # Subtask 4: 'release object'
    else:
        return 3 # Task is completed

# Progress computation for each subtask
def progress_function(env):
    plan_list = generate_plan_list()
    subtask_idx = determination_function(env, plan_list)

    # Initialize variables
    main_progress = 0.0 # Overall task progress
    subprogress = 0.0 # Progress of the current subtask

    # Subtask 1: 'move to object'
    if subtask_idx == 0:
        # Progress of moving towards the object
        initial_distance = np.linalg.norm(env.robot.tcp - env.
            obj.obj_init_pos)
        current_distance = np.linalg.norm(env.robot.tcp - env.
            obj.position)
        subprogress = 1 - (current_distance / initial_distance)
        main_progress = subtask_idx + subprogress # Task
            progress is the sum of subtasks

    # Subtask 2: 'grasp object'
    elif subtask_idx == 1:
        # Progress of grasping the object (object lifted above
            initial position)
        subprogress = (env.robot.ee_position[2] - env.obj.
            obj_init_pos[2]) / 0.5 # Normalize based on height
        main_progress = subtask_idx + subprogress

    # Subtask 3: 'move to goal'
    elif subtask_idx == 2:
        # Progress of moving towards the goal position
        initial_distance_to_goal = np.linalg.norm(env.obj.
            position - env.obj_init_pos)
        current_distance_to_goal = np.linalg.norm(env.obj.
            position - env.goal_position)
        subprogress = 1 - (current_distance_to_goal /
            initial_distance_to_goal)
        main_progress = subtask_idx + subprogress

    # Subtask 4: 'release object'
    elif subtask_idx == 3:
        # Task is complete, no further progress is needed
        subprogress = 1.0
        main_progress = 1.0

    return main_progress, subtask_idx

```

563 **NeurIPS Paper Checklist**

564 The checklist is designed to encourage best practices for responsible machine learning research,  
565 addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove  
566 the checklist: **The papers not including the checklist will be desk rejected.** The checklist should  
567 follow the references and follow the (optional) supplemental material. The checklist does NOT count  
568 towards the page limit.

569 Please read the checklist guidelines carefully for information on how to answer these questions. For  
570 each question in the checklist:

- 571 • You should answer **[Yes]** , **[No]** , or **[NA]** .
- 572 • **[NA]** means either that the question is Not Applicable for that particular paper or the  
573 relevant information is Not Available.
- 574 • Please provide a short (1–2 sentence) justification right after your answer (even for NA).

575 **The checklist answers are an integral part of your paper submission.** They are visible to the  
576 reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it  
577 (after eventual revisions) with the final version of your paper, and its final version will be published  
578 with the paper.

579 The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation.  
580 While "**[Yes]**" is generally preferable to "**[No]**", it is perfectly acceptable to answer "**[No]**" provided a  
581 proper justification is given (e.g., "error bars are not reported because it would be too computationally  
582 expensive" or "we were unable to find the license for the dataset we used"). In general, answering  
583 "**[No]**" or "**[NA]**" is not grounds for rejection. While the questions are phrased in a binary way, we  
584 acknowledge that the true answer is often more nuanced, so please just use your best judgment and  
585 write a justification to elaborate. All supporting evidence can appear either in the main paper or the  
586 supplemental material, provided in appendix. If you answer **[Yes]** to a question, in the justification  
587 please point to the section(s) where related material for the question can be found.

588 **IMPORTANT**, please:

- 589 • **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist",**
- 590 • **Keep the checklist subsection headings, questions/answers and guidelines below.**
- 591 • **Do not modify the questions and only use the provided macros for your answers.**

592 **1. Claims**

593 Question: Do the main claims made in the abstract and introduction accurately reflect the  
594 paper's contributions and scope?

595 Answer: **[Yes]** .

596 Justification: The main claims made in the abstract and introduction accurately reflect the  
597 paper's contribution and scope.

598 Guidelines:

- 599 • The answer NA means that the abstract and introduction do not include the claims  
600 made in the paper.
- 601 • The abstract and/or introduction should clearly state the claims made, including the  
602 contributions made in the paper and important assumptions and limitations. A No or  
603 NA answer to this question will not be perceived well by the reviewers.
- 604 • The claims made should match theoretical and experimental results, and reflect how  
605 much the results can be expected to generalize to other settings.
- 606 • It is fine to include aspirational goals as motivation as long as it is clear that these goals  
607 are not attained by the paper.

608 **2. Limitations**

609 Question: Does the paper discuss the limitations of the work performed by the authors?

610 Answer: **[Yes]** .

611 Justification: Please refer to Section 6 for limitations.

612 Guidelines:

- 613 • The answer NA means that the paper has no limitation while the answer No means that  
614 the paper has limitations, but those are not discussed in the paper.
- 615 • The authors are encouraged to create a separate "Limitations" section in their paper.
- 616 • The paper should point out any strong assumptions and how robust the results are to  
617 violations of these assumptions (e.g., independence assumptions, noiseless settings,  
618 model well-specification, asymptotic approximations only holding locally). The authors  
619 should reflect on how these assumptions might be violated in practice and what the  
620 implications would be.
- 621 • The authors should reflect on the scope of the claims made, e.g., if the approach was  
622 only tested on a few datasets or with a few runs. In general, empirical results often  
623 depend on implicit assumptions, which should be articulated.
- 624 • The authors should reflect on the factors that influence the performance of the approach.  
625 For example, a facial recognition algorithm may perform poorly when image resolution  
626 is low or images are taken in low lighting. Or a speech-to-text system might not be  
627 used reliably to provide closed captions for online lectures because it fails to handle  
628 technical jargon.
- 629 • The authors should discuss the computational efficiency of the proposed algorithms  
630 and how they scale with dataset size.
- 631 • If applicable, the authors should discuss possible limitations of their approach to  
632 address problems of privacy and fairness.
- 633 • While the authors might fear that complete honesty about limitations might be used by  
634 reviewers as grounds for rejection, a worse outcome might be that reviewers discover  
635 limitations that aren't acknowledged in the paper. The authors should use their best  
636 judgment and recognize that individual actions in favor of transparency play an impor-  
637 tant role in developing norms that preserve the integrity of the community. Reviewers  
638 will be specifically instructed to not penalize honesty concerning limitations.

639 **3. Theory assumptions and proofs**

640 Question: For each theoretical result, does the paper provide the full set of assumptions and  
641 a complete (and correct) proof?

642 Answer: **[Yes]**.

643 Justification: Please refer to Appendix A for the proof of the theorems.

644 Guidelines:

- 645 • The answer NA means that the paper does not include theoretical results.
- 646 • All the theorems, formulas, and proofs in the paper should be numbered and cross-  
647 referenced.
- 648 • All assumptions should be clearly stated or referenced in the statement of any theorems.
- 649 • The proofs can either appear in the main paper or the supplemental material, but if  
650 they appear in the supplemental material, the authors are encouraged to provide a short  
651 proof sketch to provide intuition.
- 652 • Inversely, any informal proof provided in the core of the paper should be complemented  
653 by formal proofs provided in appendix or supplemental material.
- 654 • Theorems and Lemmas that the proof relies upon should be properly referenced.

655 **4. Experimental result reproducibility**

656 Question: Does the paper fully disclose all the information needed to reproduce the main ex-  
657 perimental results of the paper to the extent that it affects the main claims and/or conclusions  
658 of the paper (regardless of whether the code and data are provided or not)?

659 Answer: **[Yes]**.

660 Justification: Please refer to Appendix C for experiment information.

661 Guidelines:

- 662 • The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

## 5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: **[Yes]**.

Justification: We will provide the code in the supplementary materials.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- 717           • Providing as much information as possible in supplemental material (appended to the  
718            paper) is recommended, but including URLs to data and code is permitted.

719           **6. Experimental setting/details**

720           Question: Does the paper specify all the training and test details (e.g., data splits, hyper-  
721            parameters, how they were chosen, type of optimizer, etc.) necessary to understand the  
722            results?

723           Answer: **[Yes]** .

724           Justification: Please refer to Appendix C and D for training and test details.

725           Guidelines:

- 726           • The answer NA means that the paper does not include experiments.  
727           • The experimental setting should be presented in the core of the paper to a level of detail  
728            that is necessary to appreciate the results and make sense of them.  
729           • The full details can be provided either with the code, in appendix, or as supplemental  
730            material.

731           **7. Experiment statistical significance**

732           Question: Does the paper report error bars suitably and correctly defined or other appropriate  
733            information about the statistical significance of the experiments?

734           Answer: **[Yes]** .

735           Justification: Please refer to the results in Section 5 for error bars.

736           Guidelines:

- 737           • The answer NA means that the paper does not include experiments.  
738           • The authors should answer "Yes" if the results are accompanied by error bars, confi-  
739            dence intervals, or statistical significance tests, at least for the experiments that support  
740            the main claims of the paper.  
741           • The factors of variability that the error bars are capturing should be clearly stated (for  
742            example, train/test split, initialization, random drawing of some parameter, or overall  
743            run with given experimental conditions).  
744           • The method for calculating the error bars should be explained (closed form formula,  
745            call to a library function, bootstrap, etc.)  
746           • The assumptions made should be given (e.g., Normally distributed errors).  
747           • It should be clear whether the error bar is the standard deviation or the standard error  
748            of the mean.  
749           • It is OK to report 1-sigma error bars, but one should state it. The authors should  
750            preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis  
751            of Normality of errors is not verified.  
752           • For asymmetric distributions, the authors should be careful not to show in tables or  
753            figures symmetric error bars that would yield results that are out of range (e.g. negative  
754            error rates).  
755           • If error bars are reported in tables or plots, The authors should explain in the text how  
756            they were calculated and reference the corresponding figures or tables in the text.

757           **8. Experiments compute resources**

758           Question: For each experiment, does the paper provide sufficient information on the com-  
759            puter resources (type of compute workers, memory, time of execution) needed to reproduce  
760            the experiments?

761           Answer: **[No]** .

762           Justification: Partially. We report the type of compute workers, memory, time of execution  
763            in Section C.

764           Guidelines:

- 765           • The answer NA means that the paper does not include experiments.  
766           • The paper should indicate the type of compute workers CPU or GPU, internal cluster,  
767            or cloud provider, including relevant memory and storage.

- 768           • The paper should provide the amount of compute required for each of the individual  
769           experimental runs as well as estimate the total compute.  
770           • The paper should disclose whether the full research project required more compute  
771           than the experiments reported in the paper (e.g., preliminary or failed experiments that  
772           didn't make it into the paper).

773           **9. Code of ethics**

774           Question: Does the research conducted in the paper conform, in every respect, with the  
775           NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

776           Answer: **[Yes]** .

777           Justification: The research conforms with the NeurIPS Code of Ethics.

778           Guidelines:

- 779           • The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.  
780           • If the authors answer No, they should explain the special circumstances that require a  
781           deviation from the Code of Ethics.  
782           • The authors should make sure to preserve anonymity (e.g., if there is a special consid-  
783           eration due to laws or regulations in their jurisdiction).

784           **10. Broader impacts**

785           Question: Does the paper discuss both potential positive societal impacts and negative  
786           societal impacts of the work performed?

787           Answer: **[NA]** .

788           Justification: There is no societal impact of the work performed.

789           Guidelines:

- 790           • The answer NA means that there is no societal impact of the work performed.  
791           • If the authors answer NA or No, they should explain why their work has no societal  
792           impact or why the paper does not address societal impact.  
793           • Examples of negative societal impacts include potential malicious or unintended uses  
794           (e.g., disinformation, generating fake profiles, surveillance), fairness considerations  
795           (e.g., deployment of technologies that could make decisions that unfairly impact specific  
796           groups), privacy considerations, and security considerations.  
797           • The conference expects that many papers will be foundational research and not tied  
798           to particular applications, let alone deployments. However, if there is a direct path to  
799           any negative applications, the authors should point it out. For example, it is legitimate  
800           to point out that an improvement in the quality of generative models could be used to  
801           generate deepfakes for disinformation. On the other hand, it is not needed to point out  
802           that a generic algorithm for optimizing neural networks could enable people to train  
803           models that generate Deepfakes faster.  
804           • The authors should consider possible harms that could arise when the technology is  
805           being used as intended and functioning correctly, harms that could arise when the  
806           technology is being used as intended but gives incorrect results, and harms following  
807           from (intentional or unintentional) misuse of the technology.  
808           • If there are negative societal impacts, the authors could also discuss possible mitigation  
809           strategies (e.g., gated release of models, providing defenses in addition to attacks,  
810           mechanisms for monitoring misuse, mechanisms to monitor how a system learns from  
811           feedback over time, improving the efficiency and accessibility of ML).

812           **11. Safeguards**

813           Question: Does the paper describe safeguards that have been put in place for responsible  
814           release of data or models that have a high risk for misuse (e.g., pretrained language models,  
815           image generators, or scraped datasets)?

816           Answer: **[NA]** .

817           Justification: This paper pose no such risks.

818           Guidelines:

- 819           • The answer NA means that the paper poses no such risks.

- 820           • Released models that have a high risk for misuse or dual-use should be released with  
 821            necessary safeguards to allow for controlled use of the model, for example by requiring  
 822            that users adhere to usage guidelines or restrictions to access the model or implementing  
 823            safety filters.
- 824           • Datasets that have been scraped from the Internet could pose safety risks. The authors  
 825            should describe how they avoided releasing unsafe images.
- 826           • We recognize that providing effective safeguards is challenging, and many papers do  
 827            not require this, but we encourage authors to take this into account and make a best  
 828            faith effort.

829           **12. Licenses for existing assets**

830           Question: Are the creators or original owners of assets (e.g., code, data, models), used in  
 831            the paper, properly credited and are the license and terms of use explicitly mentioned and  
 832            properly respected?

833           Answer: **[Yes]** .

834           Justification: Please refer to Appendix C and D for detailed information of used assets.

835           Guidelines:

- 836           • The answer NA means that the paper does not use existing assets.
- 837           • The authors should cite the original paper that produced the code package or dataset.
- 838           • The authors should state which version of the asset is used and, if possible, include a  
 839            URL.
- 840           • The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- 841           • For scraped data from a particular source (e.g., website), the copyright and terms of  
 842            service of that source should be provided.
- 843           • If assets are released, the license, copyright information, and terms of use in the  
 844            package should be provided. For popular datasets, [paperswithcode.com/datasets](http://paperswithcode.com/datasets)  
 845            has curated licenses for some datasets. Their licensing guide can help determine the  
 846            license of a dataset.
- 847           • For existing datasets that are re-packaged, both the original license and the license of  
 848            the derived asset (if it has changed) should be provided.
- 849           • If this information is not available online, the authors are encouraged to reach out to  
 850            the asset's creators.

851           **13. New assets**

852           Question: Are new assets introduced in the paper well documented and is the documentation  
 853            provided alongside the assets?

854           Answer: **[Yes]** .

855           Justification: Included in the supplementary materials.

856           Guidelines:

- 857           • The answer NA means that the paper does not release new assets.
- 858           • Researchers should communicate the details of the dataset/code/model as part of their  
 859            submissions via structured templates. This includes details about training, license,  
 860            limitations, etc.
- 861           • The paper should discuss whether and how consent was obtained from people whose  
 862            asset is used.
- 863           • At submission time, remember to anonymize your assets (if applicable). You can either  
 864            create an anonymized URL or include an anonymized zip file.

865           **14. Crowdsourcing and research with human subjects**

866           Question: For crowdsourcing experiments and research with human subjects, does the paper  
 867            include the full text of instructions given to participants and screenshots, if applicable, as  
 868            well as details about compensation (if any)?

869           Answer: **[NA]** .

870           Justification: This paper does not involve crowdsourcing nor research with human subjects.

871 Guidelines:

- 872     • The answer NA means that the paper does not involve crowdsourcing nor research with  
873        human subjects.
- 874     • Including this information in the supplemental material is fine, but if the main contribu-  
875        tion of the paper involves human subjects, then as much detail as possible should be  
876        included in the main paper.
- 877     • According to the NeurIPS Code of Ethics, workers involved in data collection, curation,  
878        or other labor should be paid at least the minimum wage in the country of the data  
879        collector.

880     **15. Institutional review board (IRB) approvals or equivalent for research with human  
881        subjects**

882     Question: Does the paper describe potential risks incurred by study participants, whether  
883        such risks were disclosed to the subjects, and whether Institutional Review Board (IRB)  
884        approvals (or an equivalent approval/review based on the requirements of your country or  
885        institution) were obtained?

886     Answer: [NA].

887     Justification: This paper does not involve crowdsourcing nor research with human subjects.

888     Guidelines:

- 889     • The answer NA means that the paper does not involve crowdsourcing nor research with  
890        human subjects.
- 891     • Depending on the country in which research is conducted, IRB approval (or equivalent)  
892        may be required for any human subjects research. If you obtained IRB approval, you  
893        should clearly state this in the paper.
- 894     • We recognize that the procedures for this may vary significantly between institutions  
895        and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the  
896        guidelines for their institution.
- 897     • For initial submissions, do not include any information that would break anonymity (if  
898        applicable), such as the institution conducting the review.

899     **16. Declaration of LLM usage**

900     Question: Does the paper describe the usage of LLMs if it is an important, original, or  
901        non-standard component of the core methods in this research? Note that if the LLM is used  
902        only for writing, editing, or formatting purposes and does not impact the core methodology,  
903        scientific rigorousness, or originality of the research, declaration is not required.

904     Answer: [Yes].

905     Justification: Please refer to Section 4 for the usage of LLM.

906     Guidelines:

- 907     • The answer NA means that the core method development in this research does not  
908        involve LLMs as any important, original, or non-standard components.
- 909     • Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>)  
910        for what should or should not be described.