AgentRM: Enhancing Agent Generalization with Reward Modeling

Anonymous ACL submission

Abstract

Existing LLM-based agents have achieved strong performance on held-in tasks, but their generalizability to unseen tasks remains poor. Hence, some recent work focus on fine-tuning the policy model with more diverse tasks to improve the generalizability. In this work, we find that finetuning a reward model to guide the policy model is more robust than directly fine-tuning the policy model. Based on this finding, we propose AgentRM, a generalizable reward model, to guide the policy model for effective test-time search. We comprehensively investigate three approaches to construct the reward model, including explicit reward modeling, implicit reward modeling and LLM-as-a-judge. We then use AgentRM to guide the answer generation with Best-of-N sampling and step-level beam search. On nine agent tasks, AgentRM enhances the base policy model by 8.8 points on average, surpassing the top general agent by 4.0 points. As for the specializability, AgentRM can also boost a finetuned policy model and outperform the top specialized agent by 11.4 on held-in tasks. All the data and source codes will be released to facilitate the research in this area.

1 Introduction

Large language model (LLM)-based agents (Mialon et al., 2023; Sumers et al., 2023) have become a promising solution to complex interactive tasks (Xi et al., 2024) in recent years. While specialized agents (Wang et al., 2024b; Qin et al., 2023) achieve strong performance on held-in tasks, their generalizability to unseen tasks is poor. To address this challenge, existing works focus on integrating more diverse agent tasks including human-crafted (Zeng et al., 2023; Chen et al., 2024a; Xi et al., 2024; Zhang et al., 2024b; Acikgoz et al., 2025) and LLM synthesized (Hu et al., 2024; Fu et al., 2025), to perform multi-task fine-tuning on the base LLM.

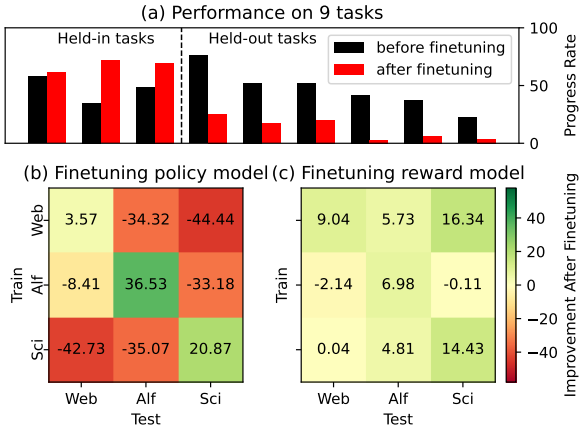

Figure 1: Finetuning the reward model is more robust than finetuning the policy model for agent tasks. (a) Finetuning the policy model leads to severe degradation on held-out tasks. (b)(c) show the performance of Best-of-5 with a reward model. Finetuning the policy model on one task degrades on others while finetuning the reward model mostly generalized to others.

Despite extensive efforts to scale task diversity for training the base LLM, we find finetuning the base LLM improves held-in task performance but degrades held-out task performance (Figure 1(a)). A potential explanation is that finetuning the base LLM, which is used as the policy model for token-by-token action generation, increases the likelihood of seen action tokens while decreasing that of unseen actions. Rather than finetuning the policy model directly, we hypothesize that finetuning a reward model to guide the policy model is more robust. Since the regression training objective of the reward function is inherently less sensitive to the specific distribution of action tokens. In our preliminary experiment, we perform Best-of-5, i.e. generating 5 candidate trajectories with the policy model and selecting one using the reward model. Figure 1(b)/(c) shows the improvement after finetuning the policy/reward model respectively on individual tasks. In Figure 1(b), only the diagonal values, i.e. performance of the held-in task which is

seen during training, are positive. Contrastly, Figure 1(c) reveals predominantly positive values, indicating that finetuning the reward model on a single task can enhance the performance on unseen tasks. Inspired by this, we introduce AgentRM, a generalizable reward model, to guide the policy model for effective test-time search. Since the effective construction of the reward model for agent tasks remains an open question, we investigate three representative reward modeling approaches including (1) explicit reward modeling (Zhang et al., 2024a) which learns the step-level rewards annotated by tree search, (2) implicit reward modeling (Yuan et al., 2024) which derives the inherent step-level rewards by training on outcome rewards, and (3) LLM-as-a-judge (Zheng et al., 2023) which directly prompts an LLM to assess the agent trajectory. We then use AgentRM to guide the answer generation in the Best-of-N sampling and step-level beam search.

Experimental results on nine agent tasks show that the explicit modeling consistently achieves the best performance. Concretely, it surpasses the top general agent by 4 points with a non-finetuned policy model, and surpasses the top task-specific agent by 11.4 points with a task-specific finetuned policy model. Further analysis shows our general reward model trained on states sampled by LLaMA-3-8B can be directly applied to enhance stronger policy models such as LLaMA-3-70B.

2 Task Formulation

The agent task with environment feedback can be formalized as a partially observable Markov decision process $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$ with instruction space $\mathcal{U}$, state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The initial state $s_1 = (u, o_0) \in \mathcal{S}$ consists of task instruction u and the initial observation o_0. At step t, conditioned on the current state s_t, the agent generates the next action $a_t \sim \pi(\cdot|s_t)$ based on its policy π. Then, the agent receives the environment observation $o_t \in \mathcal{O}$ and the state transforms to $s_{t+1} = (s_t, a_t, o_t) = (u, o_0, a_{<t+1}, o_{<t+1})$ according to transition function $\mathcal{T}$. The agent continues to interact with the environment until the task is finished or the maximum step is reached. The environment only provides the outcome reward at the final step $r_T(s_T, a_T) \in \mathcal{R}$, where T denotes the total step number. As illustrated in Section 3.2, we train a process reward model that produces rewards for intermediate steps $r_t(s_t, a_t), t < T$. We discuss the training details in Section 3.2.

3 Methodology

The overview is depicted in Figure 2. Section 3.1 describes the behavior cloning through which we derive a policy model with basic task ability on held-in tasks. Section 3.2 elaborates on how we use the derived policy model to build our generalizable reward model. We systematically investigate three different reward modelings. Section 3.3 explains how we use our reward model to enhance the policy model's decision-making ability through test-time search.

3.1 Behavior Cloning

To obtain an initial policy π_{init} with basic task ability, crucial for collecting high-quality states, we split a portion of task instructions from the training set, annotate them by an expert agent and conduct supervised fine-tuning (SFT) on the expert trajectories $D_{expert} = \{(u^i, o_0^i, a_t^i, o_t^i)_{t=1}^{T_i}\}_{i=1}^{N}$ as follows:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N}\sum_{t=1}^{T_i} \log \pi_\theta(a_t^i \mid u^i, o_0^i, a_{<t}^i, o_{<t}^i) \tag{1}$$

where θ denotes the parameters of the policy model, N denotes the number of trajectories in D_{expert}, T_i denotes the total step of the i-th trajectory. Note that the data is formatted in ReAct-style (Yao et al., 2022), and we use a to denote the complete ReAct-style response (containing both thought and action tokens) generated by π for simplicity.

3.2 Reward Modeling

3.2.1 Explicit Reward Modeling

Given that agent tasks typically involve long-chain reasoning and vast search space, we organize the agent's search trajectories into tree structures and employ a Monte Carlo Tree Search (MCTS)-inspired approach to make the search process efficient. This approach aims to avoid redundant searches, encourage sampling diversity, and improve search efficiency.

The search tree consists of nodes representing states s_t and edges representing actions a_t. We consider the initial state s_1, which includes the task instruction u and the initial observation o_0, as the root node. A search trajectory starting from s_1 is formalized as a branch extending from the

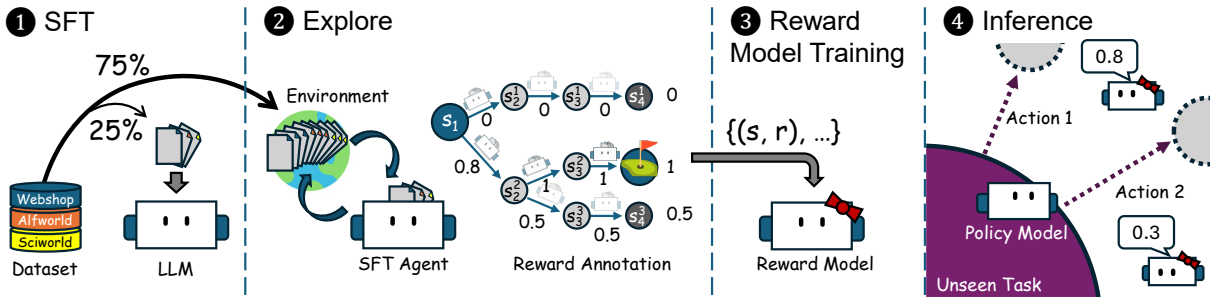

Figure 2: Overview. ❶ Deriving a supervised fine-tuned (SFT) agent on expert trajectories. ❷ Constructing search trees by exploring the environment using the SFT agent. ❸ Training a generalizable reward model, on state-reward pairs extracted from search trees. ❹ Enhancing the policy model, regardless of its initial strength, through test-time search guided by our reward model for unseen tasks.

root node. Each node records information such as the state content (action a_t and corresponding observation o_t), the number of visit $N(s_t)$, and the expected future reward $V(s_t)$ starting from state s_t. For each task instruction, we construct a search tree starting from the root node and expanding through repeating the following four stages for ω iterations:

Selection aims to identify the most promising node to be expanded in the next iteration. Starting from the root node, it traverses the tree by selecting child nodes according to the Upper Confidence Bound (UCB) value until a leaf is reached:

$$s_t = \underset{s_j \in \text{Children}(s_{t-1})}{\arg\max} \left(V(s_j) + c \cdot \sqrt{\frac{\log N(s_{t-1})}{1 + N(s_j)}} \right),$$

Expansion will be operated on the selected node s_t if it is not a terminal state exceeding the maximum step or finishing reasoning. The agent samples the next action $a_t \sim \pi(\cdot \mid s_t)$ for k times with temperature τ based on its policy. Actions with identical action tokens are merged to lower the cost of repetitive search, resulting in $\hat{k}$ next states $\{s_{t+1}^i\} = \{(s_t, a_t, o_t)^i\}, i = 1 \dots \hat{k}$.

Simulation is used to estimate the initial value of the above expanded node s_{t+1} by generating n complete trajectories from it to get the outcome reward returned by the environment and averaging their outcome rewards. To speed up the tree sear, we cache the rollout nodes for future expansion.

Backpropagation is conducted once the values of the expanded nodes are determined. The value $V(s_{t+1}^i)$ is propagated back up the tree, updating each node's visit count N and state value V:

$$V(s_t) \leftarrow \frac{V(s_t) \cdot N(s_t) + \sum_{i=1}^{\hat{k}} V(s_{t+1}^i)}{N(s_t) + \hat{k}},$$

$$N(s_t) \leftarrow N(s_t) + \hat{k}$$

Reward Model Training For each task instruction in the held-in tasks i.e. Webshop, Alfworld, Sciworld, we construct a search tree and extract state values $V(s_t)$ to form the process reward model training dataset. To ensure the quality of the estimated value, we filter states whose visit count is smaller than threshold λ. We train a language model with a value head by minimizing the Mean Squared Error (MSE) loss between the predicted value $\hat{V}(s_t)$ and the provided value $V(s_t)$:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{t=1}^{N} (\hat{V}(s_t) - V(s_t))^2 \quad (2)$$

3.2.2 Implicit Reward Modeling

Inspired by (Rafailov et al., 2024; Yuan et al., 2024), which derives a process reward model inherently from training on complete trajectories, we also investigate implicit reward modeling without annotating process reward. Specifically, the outcome reward is parameterized as the log-likelihood ratios of the policy and reference models, i.e. $r_\theta(s_T, a_T) := \beta \log \frac{\pi_\theta(s_T, a_T)}{\pi_{ref}(s_T, a_T)}$. It is proved that the Q value $q_\theta^t(s_t, a_t)$ can be implicitly learned by θ (mathematical induction can be found in (Yuan et al., 2024)). The process reward r_θ^t can be derived as follows:

$$r_\theta^t := q_\theta^t - q_\theta^{t-1} = \beta \log \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{ref}}(a_t \mid s_t)} \quad (3)$$

where $\pi_\theta, \pi_{\text{ref}}$ represent the policy and reference model parameter respectively.

Reward Model Training For each task instruction in the held-in tasks, we sample 16 complete trajectories (s_T, a_T) with temperature τ to construct the process reward model training dataset. We train a language model θ with the MSE loss to integrate the scalar reward (progress rate) provided by the environment, unlike (Yuan et al., 2024) using the cross-entropy loss for binary reward.

3.2.3 LLM-as-a-judge

We do not prompt the LLM to output a discrete score for each trajectory since the score might be identical thus insufficient to select the best answer from a set of candidates (e.g., Best-of-N). Instead, we prompt the LLM to act as a selector with instructions in Appendix D.1.

3.3 Reward-Guided Search

We boost the policy model at test time via search methods guided by our general reward model.

Best-of-N samples N complete trajectories from the policy model and then selects the final answer according to the reward generated by the reward model.

Beam Search searches over the policy model's per-step prediction in the following steps:

- Initial Sampling: Sample $W_1 \times W_2$ initial actions for the first step.
- Scoring: Evaluate the new states using the reward model.
- Filtering: Retain only the top W_1 highest-scoring states.
- Action Expansion: For each of the remaining states, sample W_2 actions for the next step, generating a total of $W_1 \times W_2$ new states.
- Iteration: Repeat steps 2–4 until all maintained states terminate.

4 Experiments

4.1 Baselines

Apart from comparing with original greedy search, we compare our method with task-specific agents and general agents. Task-specific agents include SPIN (Chen et al., 2024b), NAT (Wang et al., 2024b), ETO (Song et al., 2024), StepAgent (Deng et al., 2024b), QLASS (Lin et al., 2025) and Agent-R (Yuan et al., 2025). General agents include Agent-FLAN (Chen et al., 2024a), AgentGym (Xi et al., 2024), AgentGen (Hu et al., 2024), AgentRefine (Fu et al., 2025). We also compare with close-sourced agent based on **gpt-4o** for reference. More details can be found in Appendix C.

4.2 Experimental Settings

Datasets We adopt the three agent tasks from ETO (Song et al., 2024) as our held-in tasks: Webshop for web navigation, Alfworld for embodied house holding, and Sciworld for embodied science experiments. We adopt agent tasks from AgentBoard (Ma et al., 2024) and AgentGym (Xi et al., 2024) as held-out tasks. Note that there are two sources of Alfworld and Sciworld. In order to align with the setting of previous works, we use he former to train the RM and evaluate in Section 4.3.2, while the latter is used for evaluation in Section 4.3.1. Details can be found in Appendix D.

Evaluation Metrics We use Success Rate which indicates whether a task is successfully completed, Progress Rate which is a scalar signal measuring the completion percentage of a task, and the average reward as the evaluation metrics.

Implementation Details We adopt the LLaMA3-8B-Instruct series model as our policy model. More details can be found in Appendix B. We divide 1/4 of the expert trajectories for SFT, i.e. 1938, 830, 370 for Webshop, Alfworld, Sciworld. The remaining 3/4 instruction is used to train reward model without expert annotation.

4.3 Results

4.3.1 Comparison with General Agents

In this setting, we compare our method with methods that aim to train a single unified agent for various tasks. To make a fair comparison, we use the original non-finetuned model as the policy model since fine-tuning leads to performance degradation on held-out tasks, and guide its generation with our AgentRM. From Table 1 we can observe that: (1) Existing general agents exhibit severe overfitting in held-in tasks, as their overall performance fail to substantially surpass those of the greedy search baseline. While AgentGym achieves a high score, it is primarily because most of the task environments are seen during training. This advantage, however, is offset by its notably weak performance on held-out tasks i.e. only 12.9 on Jericho and 16.6 on Pddl. (2) Three types of AgentRM bring varying degrees of improvement over the baseline. Among them, Explicit RM proves to be the most effective, enhancing the greedy search baseline by 8.8 on average. (3) On the Babyai task, which shares similarities with the held-in tasks Alfworld and Sciworld, the explicit RM exhibits significant positive transfer. Conversely, we observe that a policy model trained on Sciworld but not on Babyai tends to overfit to the action space of Sciworld. This phenomenon, generating actions not provided in the task instruction, is termed "action hallucination" in Chen et al. (2024a). (4) Best-of-5 with

Method	Web	Embodied			Text Game			Tool		Overall
	Webshop	Alfworld†	Sciworld†	Babyai	Jericho	Pddl	Maze	ToolQuery	ToolOperation	
gpt-4o	57.7	79.9	76.9	64.1	34.0	69.8	76.0	61.8	37.6	65.9
Agent-FLAN	61.3*	79.7*	10.9	35.3	10.1	25.5	44.0	45.7	26.8	47.1
AgentGym	68.5*	76.9*	47.3*	61.4*	12.9	16.6	56.0*	69.7*	40.2*	59.3*
AgentGen	53.9	47.6	13.9	39.4	10.8	36.4	44.0	57.6	25.1	42.0
AgentRefine	-	63.8	42.6	50.4	32.3	37.8	-	-	-	-
Greedy Search	57.8	51.1	48.5	52.1	22.5	37.7	52.0	76.1	41.6	52.7
Best-of-5										
LLM-as-a-judge	55.6	59.0	29.3	58.3	20.3	22.9	72.0	**83.1**	41.9	52.1
Explicit RM	62.4	67.7	50.1	70.6	**30.0**	33.3	**80.0**	82.1	**43.9**	61.5
Implicit RM	60.5	61.8	35.4	58.2	23.3	26.0	68.0	81.2	38.8	54.7
Beam Search ($W_1 = 5, W_2 = 5$)										
Explicit RM	**64.4**	**72.4**	**51.7**	**71.2**	29.1	**41.4**	72.0	79.3	40.6	**63.3**

Table 1: Performance comparison with general agents.* indicates the task is seen during policy training and treated as held-in evaluation. †means the sources of Alfworld and Sciworld differ from those in Table 2 to align with previous works, detailed in Appendix D. Overall performance is averaged across tasks, weighted by test set sizes.

Method	Webshop	Alfworld	Sciworld
Greedy Search	61.4	71.6	66.6
SPIN	65.4	71.9	60.3
NAT	63.2	68.3	55.6
ETO	65.7	73.4	62.5
StepAgent	67.6	76.1	64.1
QLASS	70.3	82.8	66.4
Agent-R	63.9	-	70.2
gpt-4o	57.7	66.4	66.6
Best-of-5			
LLM-as-a-judge	60.5	64.9	62.3
Explicit RM	71.0	94.8	76.1
ImplicitPRM	66.4	94.8	70.6
Beam Search ($W_1 = 5, W_2 = 5$)			
Explicit RM	**75.3**	**96.3**	**82.6**

Table 2: Comparison with task-specific agents.

LLM-as-a-judge shows a 0.6 decline on overall performance compared to greedy search. Among all tasks, it performs relatively better on tool-related tasks, suggesting that LLM-as-a-judge is more effective on tasks with less complexity and smaller search space, while being less effective on complex tasks.

4.3.2 Comparison with Task-specific Agents

In this setting, we compare our method with methods that aim to train a specialized agent for each task. Instead of training task-specific policy models, we find a single policy model simultaneously trained on three tasks capable of mastering each task without compromising performance on any. Out of the same reason, we use the general RM same as Section 4.3.1 without task-specific fine-tuning. From the results in Table 2, Best-of-5 with Explicit RM enhances the policy model by 9.6, 23.2 and 9.5 on three held-in tasks respectively. It outperforms top specialized agents including Agent-R and QLASS across all tasks, showing potential in more practical scenarios where an agent is required to be proficient in more than one task (Acikgoz et al., 2025). Further improvements can be achieved through beam search.

5 Analysis

In the following analysis, unless otherwise stated, we report the results of explicit RM with Best-of-5 inference, as it outperforms the other two reward models notably.

5.1 Robustness against Perturbation

To test the extent of overfitting on the held-in tasks, we perform 5 types of perturbations on the held-in task. Specifically, we perturb available actions in the task instruction of Alfworld, which belongs to the held-in tasks for AgentGym and Agent-FLAN. See Appendix A for details of perturbation rules.

From Table 3 we can see that, simple data perturbation leads to a significant performance drop on the held-in task. In terms of the average score, AgentGym's success rate decreases by 25.6, whereas Agent-FLAN shows a more significant performance drop of 30.3. This suggests that they might simply be memorizing the correlations between instructions/observations and corresponding actions from the training data, rather than learning to respond to the given instructions and observations. Our method achieves the highest average

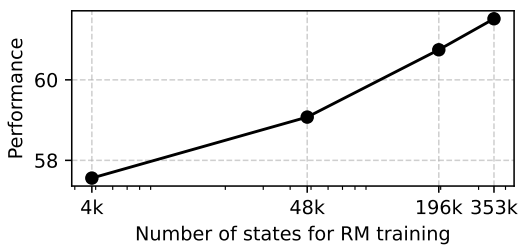

Figure 3: Scaling trend of training data.

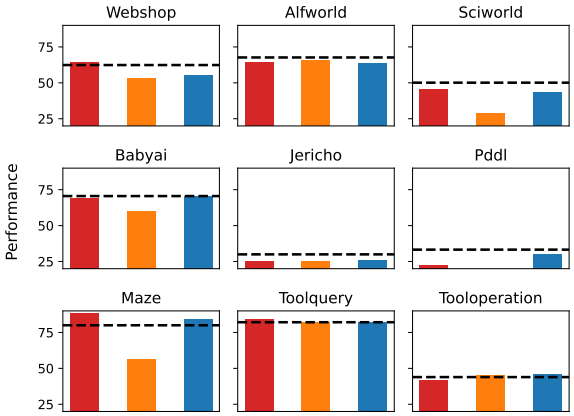

Figure 4: Performance of task-specific RM on 9 tasks. The red/orange/blue bar denotes RM trained on Webshop/Alfworld/Sciworld respectively. The dashed line denotes the performance of the general RM.

score with the lowest standard deviation, indicating that it develops the ability to make informed decisions, rather than memorizing patterns.

5.2 Scaling Trend of Generalization

We analyze the relationship between the training data size of the reward model and overall performance, with the results shown in Figure 3. The results demonstrate that even a relatively small dataset of 4k states is able to elicit significant reward modeling capabilities (57.6) for agent tasks, compared to the prompt-based training-free LLM-as-a-judge (52.1). This underscores the effectiveness of our approach in data-constrained scenarios. As the volume of training data increases, the performance exhibits a persistent log-linear growth without showing signs of saturation. The observed trend leaves room for continued performance optimization with expanded datasets.

5.3 Generalization of Task-specific RM

We examine the generalization of task-specific RM trained on each held-in task (Figure 4). The results reveal that, for most tasks, the general RM (dashed line) outperforms task-specific RMs, verifying the importance of task diversity in enhancing RM generalization. Besides, the task-specific RM trained on Alfworld exhibits comparatively weaker performance, which may be attributed to the use of success rate rather than the progress rate, which is a denser signal, as the outcome supervision when constructing RM training data.

5.4 Generalization to Other Policy Model

It is commonly thought that broad training data coverage is a requirement to ensure a good balance between adaptability to different policy distribution (Cui et al., 2025). We find that our RM can be effectively applied to states sampled by other LLM agents and enhance their performance. To verify it, we directly apply our RM, which is trained on states sampled by the LLaMA-3-8B agent, to a stronger one (LLaMA-3-70B) and a weaker one (AgentGen). From Table 4 we can see that our RM adapts well to different policy models and consistently improves the performance. Specifically, it improves the LLaMA-3-70B-based agent by 12.6 and AgentGen by 5.9, demonstrating more pronounced advantages for models that possess greater scale and potential. These encouraging results indicate that the trial-and-error task experience derived from a weaker yet more efficient agent can enhance the performance of stronger and more costly agents.

5.5 State Representation of Reward Modeling

As stated in Section 3.2, the input of our RM consists of thought tokens, action tokens, and observation tokens (except those of the last action). This section examines their respective contributions to the overall performance. Results are shown in Table 5. Explicit RM w/ last_observation means adding the observation of the last action to the state representation during both training and inference. It can be seen that the determination of state rewards for different tasks has varying degrees of reliance on the outcomes of actions. Overall, augmenting the action with its outcome does not bring significant improvement, suggesting that the RM might possess the ability to infer the outcome autonomously. Results w/o observation and w/o thought show that the individual removal of thought and observation has a negligible impact on the modeling. Results w/o thought & observation show that removing them simultaneously results in a drop of 3.2 points, indicating that thought and observation tokens provide complementary information to each other. In conclusion, the modeling primarily

Method	Original		Rule 1		Rule 2		Rule 3		Rule 4		Rule 5		Average(↑)		Std(↓)	
	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.	Succ.	Prog.
AgentGym	61.9*	76.9*	29.1	59.2	49.2	65.3	32.8	53.9	38.8	48.2	5.9	28.7	36.3	55.4	20.0	16.7
Agent-FLAN	67.2*	79.7*	21.6	58.8	51.4	71.3	27.6	53.5	52.2	67.9	1.5	19.7	36.9	58.5	22.0	22.5
AgentRefine	44.8	63.8	50.0	66.5	51.5	66.7	54.5	70.0	45.5	60.6	44.8	63.8	48.5	65.2	4.1	3.2
Ours	54.5	67.7	54.5	68.6	53.0	70.2	48.5	63.6	49.3	63.9	54.5	67.7	**52.4**	**66.9**	**2.7**	**2.6**

Table 3: Performance of Alfworld under different perturbation rules. Succ./Prog. denote Success/Progress Rate respectively. * indicates the task is seen during training and treated as held-in evaluation.

Method	Webshop	Alfworld	Sciworld	Babyai	Jericho	Pddl	Maze	Toolquery	Tooloperation	Overall
LLaMA-3-70B										
Greedy Search (w/o RM)	63.4	63.4	51.1	62.6	31.7	64.1	76.0	81.9	44.9	62.4
BestofN@5 (w/ RM)	69.5	86.9	78.8	72.0	43.0	67.9	96.0	84.6	45.9	74.9
Δ	6.1	23.5	27.7	9.4	11.3	3.9	20.0	2.7	1.0	12.6
AgentGen										
Greedy Search (w/o RM)	53.9	29.1	13.9	39.4	10.8	36.4	44.0	57.6	25.1	38.6
BestofN@5 (w/ RM)	58.7	45.0	10.6	44.6	14.7	42.9	44.0	62.8	30.2	44.4
Δ	4.7	15.9	-3.3	5.1	4.0	6.5	0.0	5.2	5.1	5.9

Table 4: Enhancement of our AgentRM to other policy models.

relies on action tokens. Utilizing only action tokens for modeling does not significantly impact the effectiveness and can accelerate the training and inference of the reward model.

5.6 Scaling Trend of Best-of-N

We select Pddl task to explore the potential gains from further increasing the number of candidates in the Best-of-N sampling using different reward modelings. The oracle result is obtained by selecting the best candidate based on the ground-truth label, which is not feasible in practice. We report it as an upper bound of performance. As shown in Figure 5, explicit RM yields consistent performance gains as the test-time compute increases. When the number of candidates increases to a certain extent, the implicit RM may become confused by the excessive number of candidates, leading to a degradation in performance. The effectiveness of using LLM-as-a-judge for scaling is limited. One reason is that as N increases, a growing number of tokens exceeding the maximum token limit of the model will be truncated. The findings indicate that additional research is necessary to establish robust test-time scaling laws with Implicit RM and LLM-as-a-judge, which we leave for future work.

5.7 Generalization to General Reasoning Task

The relationship between agent tasks and general reasoning tasks remains unclear. In this section, we explore the impact of our RM, merely trained on agent tasks, on the general reasoning tasks. We

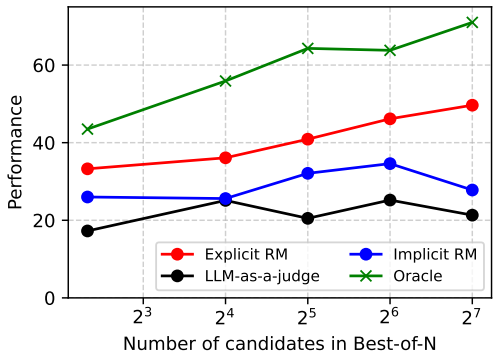

Figure 5: Scaling trend of Best-of-N.

directly apply our RM on several general reasoning benchmarks including GSM8k (Cobbe et al., 2021), MATH (Hendrycks et al., 2021) and codecontests (Li et al., 2022). We prompt the policy model to solve mathematical problems using a Python interpreter. Table 6 shows that, our RM trained on agent tasks has a negligible impact on general reasoning tasks, indicating the RM has acquired reasoning abilities common to general reasoning tasks, rather than merely fitting the patterns of agent tasks. Nevertheless, the results show the potential of our RM to serve as a general-purpose RM, which can be deployed across a wide range of applications without significant performance degradation.

6 Related Work

6.1 LLM-based Agent

Language agents have shown initial success in handling complex interactive tasks. Early works focus on building frameworks around prompt-based

Method	Webshop	Alfworld	Sciworld	Babyai	Jericho	Pddl	Maze	Toolquery	Tooloperation	Overall
Explicit RM	62.4	67.7	50.1	70.6	30.0	33.3	80.0	82.1	43.9	61.5
w/ last_observation	62.4	66.7	52.2	73.3	30.6	32.2	80.0	82.2	43.9	62.0
w/o observation	63.7	68.0	43.4	71.3	23.4	31.0	88.0	83.0	43.9	61.2
w/o thought	62.0	66.5	48.7	71.1	32.1	30.2	84.0	82.9	44.9	61.1
w/o thought & observation	62.4	66.0	45.7	69.1	22.1	25.4	44.0	83.2	39.4	58.3

Table 5: Ablation on state representation of Explicit RM.

Method	GSM8k	MATH500	Codecontests
Greedy Search	81.1	48.4	13.3
BestofN@5	79.1	49.2	13.9

Table 6: Performance on general reasoning tasks.

learning (Yao et al., 2022; Shinn et al., 2024). Recently, great efforts have been made to enhance the agent capability of open-sourced LLMs via finetuning (Chen et al., 2023; Yin et al., 2024). Qin et al. (2023); Deng et al. (2024a) imitate trajectories from expert agents (e.g., GPT-4 (Achiam et al., 2023)) for specialized ability such as tool-using or web navigation. Beyond imitation, self-improvement emerges as a promising solution to enhance performance without extensive expert annotation (Huang et al., 2023). Most works finetune models on self-generated trajectories following the self-training paradigm (Wang et al., 2024b; Chen et al., 2024b; Song et al., 2024; Xiong et al., 2024). Lately, increasing attention has been devoted to test-time self-improvement via scaling computation, e.g., generating multiple candidates and selecting the optimal one using techniques like reward models (Wang et al., 2024a; Zhai et al., 2024; Lin et al., 2025). We provide a comparison between their approach and our method in Section 6.2.

While effective for tasks seen during training, the above methods inherently compromise the agent's generalization capabilities for unseen tasks. To enhance agent generalizability, existing works integrate more diverse agent tasks for multi-task training either by human-crafted (Zeng et al., 2023; Chen et al., 2024a; Xi et al., 2024; Zhang et al., 2024b) or by LLM-sythesized (Hu et al., 2024; Fu et al., 2025). Although they alleviate overfitting to some extent, it can be observed in Table 1 that their performance on respective held-out tasks is either similar or inferior to that of the original backbone model. We are the first to propose a generalizable reward model and enhance the agent generalizability from the aspect of test-time search. Also, our method is orthogonal to theirs and can be applied to enhance their performance seamlessly, as shown in Section 5.4.

6.2 Reward Modeling for LLM

Recent advancements in reward modeling for LLMs mainly focus on general reasoning tasks such as maths and code (Uesato et al., 2022; Lightman et al., 2023; Wang et al., 2023; Zhang et al., 2024a). Different from those tasks, agent tasks typically possess a larger search space due to long-chain reasoning and environment dynamics. Data scarcity is also a challenge pronounced in agent tasks (Ma et al., 2024), making it impractical to develop task-specific reward models. Existing works (Wang et al., 2024a; Zhai et al., 2024; Putta et al., 2024; Lin et al., 2025) focus on training task-specific process reward models by Monte Carlo Tree Search based methods. We are the first to investigate the feasibility of a generalizable reward model, promoting the usage of reward models in agent tasks. Besides, we investigate 2 additional reward modelings and validate them on 6 additional complex agent tasks with larger search space.

7 Conclusion

we introduce AgentRM, a generalizable reward model, to enhance both the specializability and generalizability of language agents via test-time search. We comprehensively investigate three reward modelings with two inference methods, i.e. Best-of-N sampling and beam search. Among them, explicit reward modeling achieves consistently the best performance on all tasks. Guiding the base policy model with AgentRM surpasses top general agents on six held-out tasks. As for the performance on specific tasks, AgentRM can also boost a finetuned policy model and outperform specialized agents on three held-in tasks. Further analysis shows our general reward model trained on states sampled by LLaMA-3-8B can be directly transferred to stronger policy models. This work sheds lights on test-time scaling for agent systems.

Limitations

We conclude the limitations of this work as follows:

- Due to the significant efforts required to implement additional agent interactive environments, we only include three agent tasks as held-in tasks. According to the scaling trend of training data in Section 5.2, incorporating more tasks could further enhance the performance.

- Due to the resource constraints, we set the maximum iteration and number of simulations in MCTS as 40 and 1. Increasing these parameters could lead to more precise process reward estimations which we leave for further work.

- We do not explore the potential of equipping our policy model with prompt engineering designed for agent such as Reflexion (Shinn et al., 2024).

Acknowledgments

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Emre Can Acikgoz, Jeremiah Greer, Akul Datta, Ze Yang, William Zeng, Oussama Elachqar, Emmanouil Koukoumidis, Dilek Hakkani-Tür, and Gokhan Tur. 2025. Can a single model master both multi-turn conversations and tool use? calm: A unified conversational agentic language model. *Preprint*, arXiv:2502.08820.

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

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024a. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. 2025. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024a. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Zhirui Deng, Zhicheng Dou, Yutao Zhu, Ji-Rong Wen, Ruibin Xiong, Mang Wang, and Weipeng Chen. 2024b. From novice to expert: Llm agent policy optimization via step-wise reinforcement learning. *arXiv preprint arXiv:2411.03817*.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025. Agentrefine: Enhancing agent generalization through refinement tuning. *Preprint*, arXiv:2501.01702.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, Saravan Rajmohan, and Dongmei Zhang. 2024. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. *arXiv preprint arXiv:2408.00764*.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Zongyu Lin, Yao Tang, Xingcheng Yao, Da Yin, Ziniu Hu, Yizhou Sun, and Kai-Wei Chang. 2025. Qlass:

Boosting language agent inference via q-guided stepwise search. *Preprint*, arXiv:2502.02584.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Preprint*, arXiv:2305.18290.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. 2024a. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. 2023. Math-shepherd: A label-free step-by-step verifier for llms in mathematical reasoning. *arXiv preprint arXiv:2312.08935*.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024b. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, et al. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. *arXiv preprint arXiv:2406.11176*.

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

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. 2024. Free process rewards without process labels. *Preprint*, arXiv:2412.01981.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

Yuanzhao Zhai, Tingkai Yang, Kele Xu, Feng Dawei, Cheng Yang, Bo Ding, and Huaimin Wang. 2024. Enhancing decision-making for llm agents via step-level q-value models. *Preprint*, arXiv:2409.09345.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. 2024b. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023.

Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Perturbation Details

We modify the available actions in Alfworld to ensure that the changes consist of different tokens (or token order) while conveying the same semantic information. We revise the environment and the examples in the prompt accordingly.

- Perturbation 1: change clean $\{obj\}$ with $\{recep\}$, cool $\{obj\}$ with $\{recep\}$, heat $\{obj\}$ with $\{recep\}$ to clean $\{obj\}$ using $\{recep\}$, cool $\{obj\}$ using $\{recep\}$, heat $\{obj\}$ using $\{recep\}$ in the instruction
- Perturbation 2: change go to $\{recep\}$ to move to $\{recep\}$ in the instruction
- Perturbation 3: change take $\{obj\}$ from $\{recep\}$ to from $\{recep\}$ take $\{obj\}$ in the instruction
- Perturbation 4: delete all space between item name and item number in the instruction
- Perturbation 5: remove all alfworld data in the training set and retrain the model

B Implementation Details

Hyperparameters are listed in Table 7. The SFT data is obtained by randomly selecting 1/4 expert trajectories from the training set. The remaining 3/4 of the data is reserved for constructing RM training data. In the explicit reward data construction stage, we set the iteration number ω as 40, the exploration constant c in UCB as 0.5, the filtering threshold λ as 3, the number of the rollout in simulation n as 1, the rollout policy as greedy, the expansion width k as 5. We leverage the AdamW optimizer. All experiments are carried out on 8 NVIDIA A100 80G GPUs. We use vLLM (Kwon et al., 2023) to implement both the policy model and reward model during inference.

Stage	SFT	Explicit RM Training	Implicit RM Training
Learning Rate	2e-5	1e-5	5e-7
Cosine Scheduler Warm Up	0.1	0.03	5e-7
Batch Size	64	96	64
Weight Decay	0.0	0.0	0.0
Epoch	3	2	1
β	-	-	0.05

Table 7: Training hyper-parameters of different stages.

C Baselines

C.0.1 General Agents

Agent-FLAN (Chen et al., 2024a) is an improvement of AgentTunning focusing on training "thought" in ReAct. **AgentGym** (Xi et al., 2024) uses various environments to ensure generalization and conducts both SFT and DPO. **AgentGen** (Hu et al., 2024) uses LIMA to synthesize diversified agent-tuning data. **AgentRefine** (Fu et al., 2025) propose an agent synthesis framework to encompass diverse environments and construct a dataset by prompting a strong LLM to refine its error action according to the environment feedback. For a fair comparison, all general agents receive the task instruction and one successful trajectory as input and respond in ReAct-style. For a fair comparison, we reproduce Agent-FLAN, AgentGym and AgentGen based on LLaMA-3-8B-Instruct. Since AgentRefine has not open sourced, we only report the results on five tasks in (Fu et al., 2025) with LLaMA-3-8B-Instruct backbone.

C.0.2 Task-specific Agents

SPIN (Chen et al., 2024b) augments the expert trajectory dataset with the agent's successful trajectories. **NAT** (Wang et al., 2024b) and **ETO** (Song et al., 2024) incorporate failed trajectories into the training process, allowing the agent to learn from its failure experiences. **StepAgent** (Deng et al., 2024b) utilizes step-wise reward to optimize the agent's reinforcement learning process. **QLASS** (Lin et al., 2025) guides stepwise search with trained task-specific Q-value models. **Agent-R** (Yuan et al., 2025) leverages MCTS to construct training samples that recover correct trajectories from erroneous ones. Results of SPIN, NAT, ETO, StepAgent are taken from (Deng et al., 2024b) with LLaMA-3-8B-Instruct backbone. Since QLASS has not open sourced, we report the results in (Lin et al., 2025) with LLaMA-2-chat backbone.

D Task Statistics

Table 8 presents the statistics of both held-in and held-out tasks. We adopt agent tasks from AgentBoard (Ma et al., 2024) and AgentGym (Xi et al., 2024) as held-out tasks: **Alfworld**, **Sciworld**, **Babyai** for embodied house holding, **Jericho** and

Pddl and **Maze** for text game, **ToolQuery** and **ToolOperation** for tool using. Results of Section 4.3.1/Section 4.3.2 are conducted on Alfworld and Sciworld implemented by AgentBoard (Ma et al., 2024)/ETO (Song et al., 2024) respectively to align with previous works. They have slight differences in action space and test set number. Only Alfworld and Sciworld from ETO (Song et al., 2024) provide training data, hence we collect RM training data from the ETO environment.

D.1 LLM-as-a-judge prompt

We list the prompt of the LLM-as-a-judge method as follows:

```
You are trajectory reward model, an expert in defining which trajectory is better and closer to solving the task. Here is the task description:
******************************
task description: {task_description}
task goal: {task_goal}
******************************
Here are several candidates. They are all trying to solve the task. Their trajectories are as follows.
******************************
CANDIDATE1:
{candidate_1}
******************************
CANDIDATE2:
{candidate_2}
******************************
CANIDATE3:
{candidate_3}
******************************
CANIDATE4:
{candidate_4}
******************************
CANIDATE5:
{candidate_5}
******************************
```

We force the LLM to call the following function to give the answer:

```
[{
    "type": "function",
    "function": {
"name": "choose_preferred_answer",
"description": "Choose the preferred answer for the task within all given answers.",
"parameters": {
"type": "object",
"properties": {
"preference": {
"type": "number",
"enum": [1, 2, 3, 4, 5],
"description": "The index of the preferred answer in all given answers (ranging from 1 to 5)."
},
},
}
    }
}]
```

D.2 Preference Accuracy of RM

We evaluate the quality of our RM estimated step reward by assessing its ability to determine preferences between state pairs. AgentBoard (Ma et al., 2024) offers a method to compute the progress rate for each state by annotating subgoals for every task. We create state pairs with a progress rate difference exceeding a threshold of 0.3. Then, we calculate the accuracy of our RM in predicting preferences (Table 9). Despite predicting reward for each state independently, Explicit RM still demonstrates better preference judgment accuracy on most tasks compared to LLM-as-a-judge which sees pairwise states during inference.

task	Webshop	Alfworld	Sciworld	Babyai	Jericho	PDDL	Maze	Toolquery	Tooloperation
# Train	10426	3321	1483	-	-	-	-	-	-
# SFT	1938	830	370	-	-	-	-	-	-
# RM Training	8488	2491	1113	-	-	-	-	-	-
# Test	200	134(ETO)/134(AgentBoard)	211(ETO)/90(AgentBoard)	112	20	60	25	60	40
Reward Type	scalar	binary	scalar	scalar	scalar	scalar	binary	scalar	scalar
Avg. Turn	3	6	15	10	20	20	4.3	5	6
Max. Turn	10	20/30	$[15, 120]$/30	30	30	30	30	30	30
Action Space	2	10/13	19/21	8	150	8	4	15	16

Table 8: Statistics of held-in and held-out tasks.

Method	Babyai	Jericho	Pddl	Maze	Toolquery	Tooloperation
LLM-as-a-judge	65.7	46.0	70.7	65.8	81.4	42.1
Explicit RM	77.0	64.9	65.4	94.7	72.9	57.9

Table 9: The accuracy of judging relative step reward.