
SPA-RL: Reinforcing LLM Agents via Stepwise Progress Attribution

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang*, Wenjie Li

Department of Computing, The Hong Kong Polytechnic University

{hanlin-henry.wang, chak-tou.leong}@connect.polyu.hk

jian51.wang@polyu.edu.hk

{csjwang, cswjli}@comp.polyu.edu.hk

Abstract

Reinforcement learning (RL) holds significant promise for training LLM agents to handle complex, goal-oriented tasks that require multi-step interactions with external environments. However, a critical challenge when applying RL to these agentic tasks arises from delayed rewards: *feedback signals are typically available only after the entire task is completed*. This makes it non-trivial to assign delayed rewards to earlier actions, providing insufficient guidance regarding environmental constraints and hindering agent training. In this work, we draw on the insight that the ultimate completion of a task emerges from the cumulative progress an agent makes across individual steps. We propose **Stepwise Progress Attribution (SPA)**, a general reward redistribution framework that decomposes the final reward into stepwise contributions, each reflecting its incremental progress toward overall task completion. To achieve this, we train a progress estimator that accumulates stepwise contributions over a trajectory to match the task completion. During policy optimization, we combine the estimated per-step contribution with a grounding signal for actions executed in the environment as the fine-grained, intermediate reward for effective agent training. Extensive experiments on common agent benchmarks (including Webshop, ALFWorld, and VirtualHome) demonstrate that SPA consistently outperforms the state-of-the-art method in both success rate (+2.5% on average) and grounding accuracy (+1.9% on average). Further analyses demonstrate that our method remarkably provides more effective intermediate rewards for RL training. Our code is available at <https://github.com/WangHanLinHenry/SPA-RL-Agent>.

1 Introduction

Recent advances in large language models (LLMs) have demonstrated their impressive capability to serve as intelligent agents, yielding a wide range of application domains such as web agents [1, 2], search agents [3, 4, 5], code agents [6, 7], and embodied agents [8, 9, 10]. LLM-based agents often perform multi-step planning and interact with external environments to complete specific tasks. In such goal-reaching scenarios, reinforcement learning (RL) [11] is widely used, as it optimizes sequential decision-making and naturally endows agents with “explore-and-exploit” capabilities to maximize task performance in dynamic and interactive environments.

However, a significant challenge still remains for RL training in LLM agents: agentic tasks are often long-horizon with *sparse and delayed rewards*, where a reward indicating the task completion is provided only at the end of the trajectory. This nature of sparsity and delay makes it difficult to propagate the final reward signal back to the intermediate actions taken in earlier steps of the

*Corresponding author.

Table 1: Comparison among different RL-based training methods in the ALFWorld environment.

Method	Primary Optimization Objective	Process Rewarding	Success (%)
Policy Gradient [11]	Global (Maximize final reward)	✗	72.4
RLOO [14]	Global (Maximize final reward)	✗	71.6
GRPO [15]	Global (Maximize final reward)	✗	70.9
StepAgent [12]	Local (Optimize per-step action greedily)	Imitation Reward	75.4
PRM4A [13]	Local (Optimize per-step action greedily)	Stepwise Value	73.9
SPA (Ours)	Global (Maximize task completion)	Stepwise Progress	79.1

trajectory. Without such intermediate feedback, agents often struggle to distinguish which specific actions contribute positively or negatively to the final outcome. While recent studies have increasingly adopted process supervision to address the challenges posed by delayed rewards [12, 13], these approaches primarily emphasize local optimization (as summarized in Table 1). That is, they tend to greedily optimize for actions that appear optimal in the short term, often overlooking their alignment with long-term goal achievement. This limitation is critical, as short-sighted decisions can ultimately lead to globally suboptimal outcomes. To train effective LLM agents for long-horizon tasks, it is therefore essential to assign intermediate rewards that not only guide stepwise behavior but also provide guarantees of global optimality.

In this work, we introduce **Stepwise Progress Attribution (SPA)**, a general reward redistribution framework that effectively reinforces LLM agents by offering goal-oriented intermediate rewards. SPA is based on the observation that completing an agentic task requires accumulated progress from individual steps, with each step contributing toward the final goal. This *stepwise progress* serves as the basis for intermediate rewards. To achieve this, we train a progress estimator to assign a contribution score for each step. The sum of per-step scores represents the agent’s estimated task completion. The estimator is optimized to minimize the difference between the cumulative score and the actual task completion, i.e., the observed outcome reward. As the estimator becomes sufficiently accurate in predicting the task completion, its estimated per-step contribution becomes a valid indicator of incremental progress. At the same time, the summation form of stepwise contributions consistently aligns the intermediate attribution objective with the ultimate task completion goal (see Appendix A for theoretical analysis). During agent policy training, we combine the predicted stepwise progress with a *grounding signal* indicating whether each step is able to be executed in the environment as the final intermediate reward, to further enhance the learning effectiveness in RL. Extensive experiments on three common agentic benchmarks (WebShop [16], ALFWorld [17], and VirtualHome [18]) demonstrate that our SPA consistently outperforms previous approaches. On average, SPA achieves a 2.5% improvement in success rate and a 1.9% improvement in grounding accuracy compared to the state-of-the-art method. These results highlight the effectiveness of our method in both generating executable actions and completing tasks. Further analyses confirm that our method yields suitable intermediate rewards more effectively for developing LLM agents.

In summary, our key contributions are as follows:

- We propose Stepwise Progress Attribution (SPA), a novel framework that attributes incremental progress to each step within a multi-step trajectory. By decomposing delayed rewards into stepwise contributions, SPA enables effective reward redistribution and ensures a cohesive alignment between intermediate progress and final task completion.
- We seamlessly integrate SPA into reinforcement learning by leveraging its predicted stepwise progress as dense intermediate rewards. Combined with grounding signals for executable actions, this approach significantly enhances the training of LLM-based agents on complex agentic tasks.
- Our method achieves state-of-the-art performance across several widely used agent benchmarks (including Webshop, ALFWorld, and VirtualHome), consistently outperforming existing methods in success rate and grounding accuracy. Extensive analyses substantiate that SPA provides superior intermediate rewards, yielding more effective RL training outcomes.

2 Preliminaries

2.1 Task Formulation

We formalize agentic problems as a partially observable Markov decision process (POMDP) defined by the tuple $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$, where $\mathcal{U}$ denotes the instruction space, $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, $\mathcal{O}$ the observation space, $\mathcal{T}: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function, and $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the reward function. Since our primary focus is on the high-level task planning capabilities of LLM agents, we restrict $\mathcal{U}$, $\mathcal{A}$, and $\mathcal{O}$ to subsets of natural-language form.

Given a task instruction $u \in \mathcal{U}$, the LLM agent π_θ at time step t takes an action $a_t \sim \pi_\theta(\cdot | u, e_{t-1})$ and receives the environmental feedback as the observation $o_t \in \mathcal{O}$. e_{t-1} denotes the historical interaction trajectory $(a_1, o_1, \dots, a_{t-1}, o_{t-1})$. Each action a_t incurs the environment state to $s_t \in \mathcal{S}$. The interaction loop terminates when either the agent completes the task or the maximum step is reached. The final trajectory is $e_n = (u, a_1, o_1, \dots, a_n, o_n)$, where n denotes the length of trajectory.

A key challenge in the above setting is that the reward function $\mathcal{R}$ typically provides a sparse signal, with meaningful rewards only at the end of a trajectory. Specifically, $r_t = 0$ for all $t < n$ and $r_n \in [0, 1]$ indicating task success or failure, respectively. This sparse structure makes it delayed and difficult to propagate final rewards to earlier steps, especially for long-horizon trajectories.

2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [19] is a popular reinforcement learning algorithm widely applied in various goal-reaching tasks. It directly maximizes the expected cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^n \gamma^{t-1} r_t \right], \quad (1)$$

where π_θ is the policy with parameters θ , γ the discount factor, and r_t the reward at step t .

Rather than taking unconstrained gradient steps, PPO uses a clipped surrogate objective to ensure that each policy update does not deviate too far from the previous one. Specifically, it optimizes

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right], \quad (2)$$

and the advantages $\hat{A}_t$ are typically computed via Generalized Advantage Estimation (GAE) [20], which blends temporal-difference errors $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$ with an exponentially decaying weight λ to trade off bias and variance. Meanwhile, a separate value network V_ϕ is trained by minimizing the mean-squared error between predicted values and empirical returns. PPO alternates between optimizing the clipped policy objective and updating the value function, yielding a robust and widely used algorithm for many goal-reaching problems, including language agentic tasks. In this work, our LLM agent training is built upon PPO.

2.3 Limitations of PPO in Long-Horizon Tasks with Sparse, Delayed Rewards

When rewards are observed only at episode termination, i.e., $r_t = 0$ for $t < n - 1$, $r_n \in [0, 1]$, the temporal difference (TD) error δ_t vanishes for all but the final step. Under GAE, we have

$$\hat{A}_t = \sum_{k=0}^{n-t-1} (\gamma\lambda)^k \delta_{t+k} \approx (\gamma\lambda)^{n-t-1} \delta_{n-1}, \quad (3)$$

where one-step TD error, except the last two steps, could be assumed as zero in the ideal situation. The detailed analyses are presented in Appendix B. Therefore, the policy gradient is written as:

$$\nabla_\theta J \approx \mathbb{E} \left[\sum_{t=1}^{n-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \hat{A}_t \right] = \mathbb{E} \left[\sum_{t=1}^{n-1} (\gamma\lambda)^{n-t-1} \delta_{n-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \right], \quad (4)$$

which carries an *exponentially vanishing* weight $(\gamma\lambda)^{n-1-t}$ on early actions whenever $\gamma\lambda < 1$. Consequently, PPO with GAE fails to propagate a sparse, delayed reward back to actions taken many steps before the end.

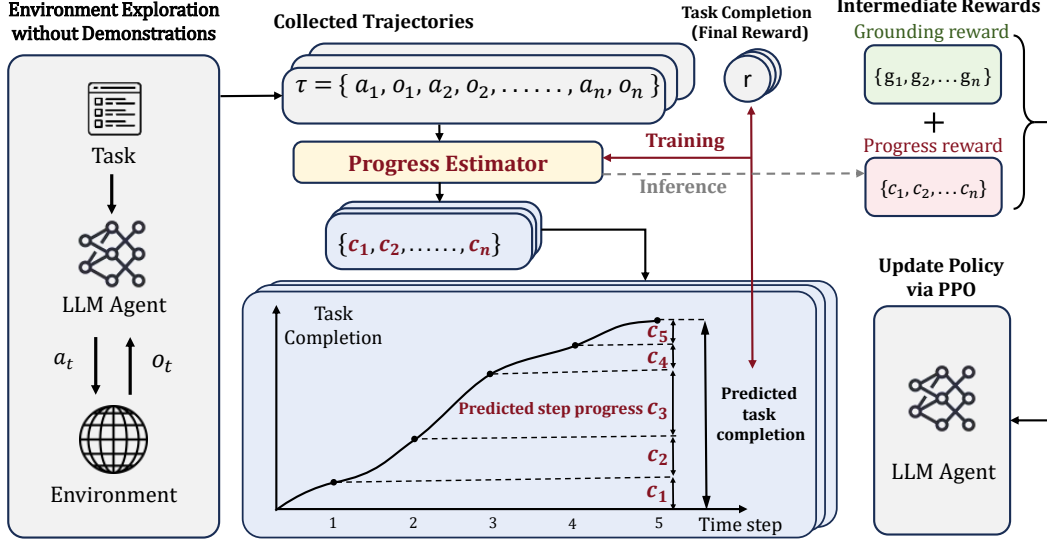


Figure 1: Overview of our proposed Stepwise Progress Attribution (SPA) framework for training LLM agents with RL.

In sum, the combination of exponentially vanishing advantages, severe discount attenuation, and degenerate value-function learning renders vanilla PPO ineffective for long-horizon tasks with sparse rewards. These foundational obstacles motivate our SPA framework, which injects intermediate supervisory signals to restore informative learning gradients throughout the trajectory.

3 Method

This section presents our proposed method, with the overview shown in Figure 1. First, we endow LLM agents with the basic task planning capability via behavior cloning (Section 3.1). Then, we construct a progress estimator to attribute the final reward to stepwise intermediate rewards (Section 3.2). Finally, we leverage these rewards for reinforcement learning of LLM agents (Section 3.3).

3.1 Behavior Cloning

To achieve basic task planning, we ask LLM agents to perform behavior cloning (BC) via supervised fine-tuning (SFT) on successful expert trajectories. Following previous studies [21, 10], We adopt the ReAct-style [22] trajectory format, where a Chain-of-Thought (CoT) rationale [23] is provided prior to each action. Since each rationale and its corresponding action are generated together, we denote the combined thought–action pair as a single unit a_t for simplicity.

Concretely, given an expert trajectory dataset $\mathcal{D} = \{(u, e)^{(i)}\}_{i=1}^{|\mathcal{D}|}$, we train the policy model by minimizing the negative log-likelihood of the expert thought–action pairs, excluding observations from the loss computation. After SFT, the LLM agent acquires fundamental task-planning capabilities, therefore resulting in a base agent π_{base} . This process significantly accelerates downstream policy optimization and reduces the sample complexity in the following RL training.

3.2 Reward Redistribution via Stepwise Progress Attribution

Trajectory Collection. Accurately estimating each action’s contribution to task completion requires observing that action across a wide range of interaction histories. To this end, we employ the base agent π_{base} to perform large-scale exploration in the environment, thereby broadly covering the space of possible state–action sequences. Instead of relying on manually designed complicated exploration schemes [24, 25, 10, 26], which constrain the action space and limit exploratory diversity, we adopt a simpler approach. Specifically, we employ our base agent π_{base} to conduct M rollouts for each task in the expert trajectory dataset $\mathcal{D}$ *without any demonstrations*. We then aggregate the

resulting trajectories to obtain the exploration dataset $\mathcal{D}_{\text{explore}} = \left\{ \left\{ (u_i, \hat{\tau}_{i,j}, R_{i,j}) \right\}_{j=1}^M \right\}_{i=1}^{|\mathcal{D}|}$, where $\hat{\tau}_{i,j}$ represents j -th trajectory of i -th task, $R_{i,j}$ denotes its corresponding final reward.

Progress Estimator Training. Unlike previous work that evaluates step-level rewards locally, we aim to construct stepwise rewards from a global task completion perspective. Specifically, we aim to measure the contribution of per-step action to the overall task completion, which we term *stepwise progress*. We treat the delayed reward as an indicator of the final task completion, then leverage the powerful reasoning capability of LLMs to construct a **progress estimator** $\mathcal{E}_\gamma$. It systematically decomposes the delayed reward into stepwise contributions, capturing how each action incrementally advances the agent towards task completion. Specifically, we append a lightweight multilayer perceptron (MLP) [27] to the last hidden layer of a pre-trained LLM π_γ , enabling it to output a scalar contribution score for each state–action pair:

$$\hat{c}_t = \text{MLP}(h_t), \quad h_t = f_{\pi_\gamma}(s_t, a_t), \quad (5)$$

where s_t represents the observed trajectory state $(u, \hat{a}_1, \hat{o}_1, \dots, \hat{a}_{t-1}, \hat{o}_{t-1})$ at step t , f_{π_γ} denotes the encoding operation of π_γ , and $\hat{c}_t$ quantifies the estimated contribution of action a_t at t -th step toward the overall task completion.

To achieve progress estimation, we assign a scalar contribution score to each action in the trajectory and constrain that the sum of these contributions matches the ground truth task completion. Concretely, we define the predicted task completion for a trajectory τ as:

$$\hat{R} = \sum_{t=1}^N \hat{c}_t, \quad (6)$$

where $\hat{c}_t$ and N denote the predicted t -th step contribution and the total number of steps in the trajectory τ , respectively. To train the progress estimator $\mathcal{E}_\gamma$, we minimize the Mean Squared Error (MSE) loss between predictions and observations as follows:

$$\mathcal{L}_{\text{PE}} = \frac{1}{|\mathcal{D}| * M} \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^M (\hat{R}_{i,j} - R_{i,j})^2, \quad (7)$$

where $\hat{R}_{i,j}$ and $R_{i,j} \in [0, 1]$ denote the predicted task completion score and the observed final reward in the j -th rollout of the i -th task, respectively. This objective encourages the model to learn to redistribute the final reward across each step in proportion to its actual contribution to task completion.

Stepwise Progress Prediction. Once the progress estimator is trained, we apply it to a given state–action pair (s_t, a_t) in each trajectory $\tau \in \mathcal{D}_{\text{explore}}$, yielding per-step contribution scores $\{c_t\}_{t=1}^n$. Each c_t quantifies how much action a_t contributes to the eventual task completion. Crucially, since $\sum_{t=1}^n c_t = \hat{R} \approx R$, the sum of per-step contributions is expected to recover the delayed reward R , ensuring that the redistributed rewards preserve the overall outcome while providing dense, fine-grained feedback. More detailed discussions are provided in Appendix A.

3.3 Reinforcement Learning with Intermediate Rewards

While obtaining the per-step contribution score c_t from the progress estimator $\mathcal{E}_\gamma$, we note that this term only captures the contribution of a step action toward eventual task completion—it does not guarantee that the action can be successfully executed in the environment. To ensure that each intermediate reward also reflects action executability, we introduce a *grounding signal* g_t that takes the value 1 if a_t can be successfully executed in the environment and 0 otherwise. Finally, we define the fused immediate reward as:

$$r_t^{\text{fused}} = \alpha c_t + \beta g_t, \quad (8)$$

where $\alpha, \beta > 0$ are the hyperparameters to balance contribution versus grounding. Injecting r_t^{fused} into the RL training objective yields both task-success and environment-grounded learning signals.

Building on the intermediate fused reward r_t^{fused} , we adapt the standard PPO [19] algorithm to effectively leverage our dense, progress-driven, and environment-grounded feedback. Concretely,

Table 2: Evaluation results on ALFWorld’s unseen tasks. These results are shown for each of the six task categories as well as the overall average. “Succ.” denotes the task success rate, and “Gro.” denotes the grounding accuracy (see Section 4.1). Higher values indicate better performance.

Method	Overall Tasks		PICK	CLEAN	HEAT	COOL	LOOK	PICK2
	Succ.(%)	Gro.(%)	Succ.(%)	Succ.(%)	Succ.(%)	Succ.(%)	Succ.(%)	Succ.(%)
SFT [28]	73.1	89.9	79.2	77.4	73.9	61.9	83.3	58.8
RFT [29]	74.6	91.2	83.3	83.9	78.3	66.7	77.8	47.1
PPO [19]	73.9	90.4	83.3	87.1	73.9	61.9	77.8	47.1
ArCHer [1]	75.4	88.6	83.3	83.9	65.2	85.7	66.7	58.8
StepAgent [12]	75.4	91.5	83.3	87.1	78.3	71.4	77.8	41.2
RAGEN [8]	75.4	90.1	91.7	77.4	78.3	57.1	88.9	52.9
PRM4A [13]	73.9	90.8	58.3	80.6	73.9	71.4	100.0	58.8
SPA (Ours)	<u>79.1</u>	93.7	<u>95.8</u>	83.9	<u>87.0</u>	61.9	77.8	58.8
w/o grounding signal	77.6	91.1	83.3	<u>90.3</u>	65.2	66.7	88.9	<u>64.7</u>
w/o stepwise progress	77.6	<u>94.2</u>	87.5	<u>80.6</u>	78.3	66.7	88.9	58.8

we replace the sparse terminal reward r_t by r_t^{fused} throughout the policy optimization and advantage estimation. The modified expected return is formulated as:

$$J_{\text{fuse}}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^n \gamma^{t-1} r_t^{\text{fused}} \right], \quad (9)$$

and the clipped surrogate objective is unchanged in form:

$$\mathcal{L}_{\text{fused}}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, \hat{A}_t^{\text{fused}}, \text{clip} \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t^{\text{fused}} \right) \right], \quad (10)$$

where the advantages $\hat{A}_t^{\text{fused}}$ is computed via GAE using temporal-difference errors $\delta_t^{\text{fused}} = r_t^{\text{fused}} + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t)$, following Eq. (3). By injecting r_t^{fused} instead of the sparse r_t , we obtain a dense reward stream that both respects stepwise progress and action executability. This integration enhances credit assignment and policy optimization, distinguishing our approach from vanilla PPO, which relies solely on sparse end-of-episode feedback.

4 Experiments

In this section, we conduct extensive experiments to validate the effectiveness of our method. The results demonstrate superior performance compared to baseline methods across various environments. Additionally, we perform ablation studies to highlight the effectiveness of each component in our intermediate reward design.

4.1 Experimental Setup

Benchmarks. We evaluate our approach on three benchmark environments: **WebShop** [16] for web navigation, and two embodied household suites: **ALFWorld** [17] and **VirtualHome** [18]. In all of these environments, the interaction between the agent and the environment can be formulated as partially observable Markov decision processes. For each task in both environments, the agent need to conduct multiple actions while interacting with the environment and at the end of the trajectory, these environments would provide a final reward to indicate whether the task is completed. For more details of our evaluated environments and corresponding datasets, please refer to Appendix C.

Training Setup. We primarily use Llama-3.2-3B-Instruct as the base model for constructing both the LLM agents and the progress estimator. During the behavior cloning stage, the policy model is trained for 3 epochs, while the progress estimator is trained for 1 epoch using all the exploration data, D_{explore} . For exploration, the number of rollouts M for the LLM agent is set to 10, with a decoding temperature of 0.7 to ensure diverse explored trajectories. The hyperparameters α and β , which balance contribution and grounding in the shaped fused reward, are set to 1 and 0.5, respectively. In the reinforcement training stage, we employ LoRA [30] for efficient training. For additional implementation details, please refer to Appendix D.

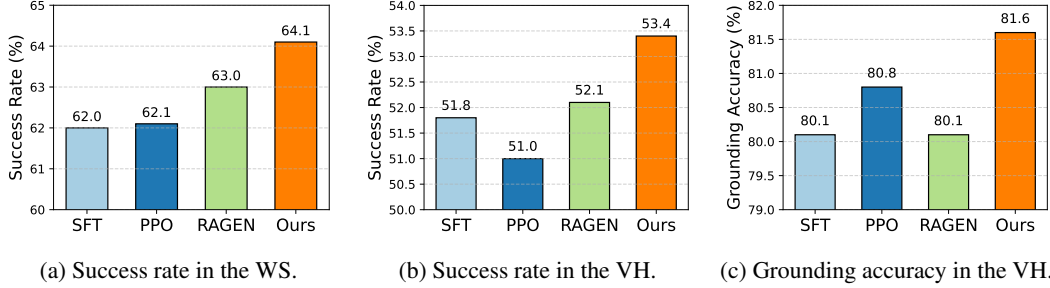


Figure 2: Performance comparison of various methods across different evaluation metrics and environments. “WS” and “VH” denote the Webshop and VirtualHome environments, respectively. In the WebShop environment, where the action set is predefined for each step and provided by the environment, the grounding accuracy consistently reaches 100%, owing to the powerful instruction-following capabilities of LLMs, and thus we do not report it here.

Evaluation Protocol. When evaluating the agent, we apply ReAct-style interaction format [22] with thought generated before the action. Additionally, we add a one-shot in-context example in the instruction prompt for each task. Relevant prompt templates are provided in Appendix F. The decoding temperature of the LLMs is set to 0 for deterministic generation. Following previous practices [21, 26], we mainly adopt **success rate** as the metric to evaluate the planning capability of LLM agents. We also report the **grounding accuracy**, which is defined as the average percentage of executable actions per trajectory, to evaluate the agent’s environment grounding capability.

Baselines. We evaluate our method against the following baseline methods: (1) **SFT** [28, 31] conducts behavior cloning on expert trajectories. (2) **RFT** [29] adds the success trajectories from rejection sampling to the expert trajectories and trains the agent on the augmented data with fine-tuning. (3) **PPO** [19] is an RL method directly optimizing the SFT agents to maximize the final reward. (4) **ArCHer** [1] is a hierarchical multi-turn RL framework designed for training LLM agents. (5) **StepAgent** [12] optimizes agent policy by incorporating stepwise supervision into reinforcement learning, thereby approaching the expert policy. (6) **RAGEN** [8] introduces a bi-level GAE approach, which enables more fine-grained rewards for RL training between different interaction turns. (7) **PRM4A** [13] utilizes rollout trajectories from exploration to train a process reward model, which offers a process-level signal for policy training.

4.2 Overall Results

In this section, we present the overall results of our method compared to baseline approaches across various environments and evaluation metrics. As shown in Table 2, our method achieves the highest overall performance in unseen ALFWorld tasks, with a task success rate of 79.1%, which is 3.6% higher than the state-of-the-art method, StepAgent. Additionally, our approach reaches a grounding accuracy of 91.7%, surpassing all baseline methods. Notably, it outperforms across five out of six task categories, with significant improvements in the PICK and CLEAN categories, where success rates increased to 95.8% and 90.3%, respectively. These results underscore the effectiveness of our reward redistribution strategy in enhancing both task success and grounding accuracy. Ablation studies further confirm the effectiveness of our reward components. The contribution reward improves performance by accurately capturing step-wise contributions, while the grounding reward enhances execution accuracy through robust guidance. When combined, these rewards work synergistically to significantly boost overall performance, highlighting the importance of their integration. Beyond the ALFWorld environment, we evaluated our method in WebShop and VirtualHome, as illustrated in Figure 2. In WebShop, our approach achieved a success rate of 64.1%, outperforming the closest baseline, RAGEN, by 1.1%. In the more challenging VirtualHome environment, characterized by tasks with longer horizons, our method achieved a success rate of 53.4%, surpassing RAGEN by 1.3%, while also attaining the highest grounding accuracy of 81.6%. These results demonstrate the robustness of our method across diverse environments and task complexities, driven by effective reward decomposition.

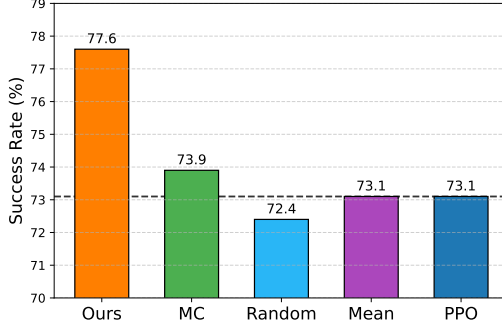


Figure 3: Performance with different intermediate rewards in the ALFWorld environment.

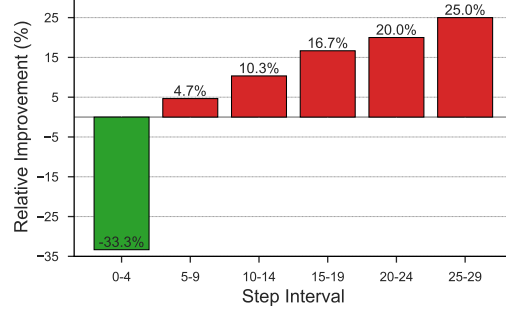


Figure 4: Relative performance improvement between our SPA and PPO at different step intervals.

5 Analyses and Discussions

5.1 Effectiveness Validation of the Progress Estimator

To verify that our trained progress estimator generates meaningful intermediate rewards rather than random noise, we evaluate five reward variants based on PPO in the ALFWorld environment. In our proposed SPA, the reward at each time step is derived from the stepwise progress reward and grounding reward. To assess the effectiveness of our method, we compare it against several alternative reward mechanisms. In the “MC” condition, the learned signals are replaced with a Monte Carlo rollout estimate of the current step reward. The “Random” condition introduces uniform random values within the range $[0, 1]$ as in-time rewards, effectively injecting noise into the system. The “Mean” condition distributes the terminal reward equally across all steps, ignoring step-specific contributions. Lastly, the standard PPO baseline relies solely on the delayed terminal reward without incorporating any intermediate rewards.

Figure 3 illustrates the success rates of the five reward variants. Our method outperforms all alternatives, achieving the highest success rate and surpassing the strongest heuristic, MC, by a notable margin. In contrast, both the Random and Mean conditions perform similarly to or worse than the standard PPO baseline, showing that arbitrary or uniformly distributed intermediate rewards fail to improve performance and can even be detrimental. These results highlight the effectiveness of the progress estimator in providing meaningful, well-calibrated intermediate rewards. By offering denser and more informative reward signals, our approach significantly improves credit assignment in long-horizon planning tasks, whereas naïve alternatives contribute little to no benefit.

5.2 Analyses of Long-horizon Task Completion

To evaluate the impact of SPA on long-horizon tasks, we compare its performance with PPO across varying step intervals in the ALFWorld test set. Specifically, the number of steps required by the LLM agent to complete different tasks varies, and we group tasks into different step intervals, counting the number of tasks completed in each interval and calculating the relative improvement of SPA over PPO in terms of task completion. As shown in Figure 4, SPA consistently outperforms PPO as the step interval increases, with a notable 25% improvement in the 25 ~ 29 step range. This trend highlights SPA’s enhanced ability to handle long-horizon scenarios by effectively attributing rewards to intermediate actions, thereby improving credit assignment over extended trajectories. Interestingly, SPA exhibits a relative performance drop in the 0 ~ 4 step interval, likely due to the simplicity of short-horizon tasks where exploration is unnecessary, and PPO’s reliance on terminal rewards suffices. This suggests that SPA’s strength lies in its ability to support complex, multi-step planning, particularly when long-term exploration and exploitation are critical.

5.3 Discussions on Different Methods in Terms of Credit Assignment

Addressing delayed rewards in agentic tasks is closely related to a typical problem in RL: *credit assignment*. It refers to the problem of determining which actions or decisions in a sequence

Table 3: Comparison of different LLM agent training methods based on credit assignment types, granularities, enhancement strategies, and their success rates in the ALFWorld environment.

Method	Type	Granularity	Enhancement Strategy	Success (%)
SPA (Ours)	GAE	Token	Reward Redistribution	79.1
RAGEN [8]	GAE	Token	Hierarchical Decomposition	75.4
PRM4A [13]	GAE	Token	Reward Shaping	73.9
StepAgent [12]	GAE	Token	Reward Shaping	75.4
PPO [19]	GAE	Token	None	73.9
RLOO [14]	RLOO	Trajectory	None	71.6
GRPO [15]	GRPO	Trajectory	None	70.9
Policy Gradient [11]	None	Trajectory	None	72.4

were responsible for a specific outcome, especially when the reward is received much later than the action that caused it. Table 3 summarizes different methods based on their credit assignment types, granularities, and enhancement strategies. We conclude that **granularity** plays a key role in performance. Trajectory-level methods, such as GRPO and RLOO, assign credits to an entire trajectory but perform worse than Policy Gradient, which lacks explicit credit assignment. This highlights the limitations of coarse-grained credit assignment. In contrast, token-level methods, including RAGEN, PRM4A, and ours, consistently achieve higher success rates, showcasing the necessity of fine-grained credit assignment. However, token-level approaches face optimization inefficiencies due to their high complexity. This suggests that step-level credit assignment, which balances granularity and efficiency, is a promising direction. **Enhancement strategies** also impact training process. Methods like RAGEN and PRM4A introduce intermediate rewards or hierarchical decomposition to address delayed rewards, which is effective. Our method further ensures consistency between intermediate and delayed rewards through reward redistribution, achieving the highest success rate. However, relying on additional models for intermediate rewards introduces potential bias. Future research could explore model-free reward redistribution and step-level granularity to improve scalability and robustness.

6 Related Work

LLM Agent Learning. To address agentic tasks, existing LLM agent learning approaches can be broadly divided into two categories: supervised fine-tuning methods and reinforcement learning (RL)-based methods. Supervised fine-tuning typically relies on collecting large-scale expert trajectories to enhance the planning capabilities of agents [32, 28, 31]. Additionally, some studies designed manual exploration strategies to construct reflection trajectories, enabling agents to develop self-reflection and better adapt to robust and dynamic environments [9, 33]. Reinforcement learning (RL)-based methods, such as Proximal Policy Optimization [19], enable LLM agents to learn directly from interaction experiences [21, 26, 12]. Unlike supervised fine-tuning, RL naturally aligns with the objective of maximizing cumulative rewards through agent-environment interactions. This makes RL particularly well-suited for agentic tasks. Therefore, in this work, we focus on RL-based approaches to further enhance the capabilities of LLM agents. The key lies in learning through interactions and optimizing for long-term rewards.

Process Supervision in RL. LLM agents must interact with the environment across multiple steps before receiving a final reward at the end of the trajectory, making it challenging to assess the quality of intermediate actions [26, 34]. This lack of intermediate feedback hinders RL training and often requires extensive interactions with the environment to evaluate the trajectory. Incorporating process supervision or intermediate signals has been verified as effective in greatly enhancing the effectiveness and efficiency of LLM agent training. Existing methods have extensively explored rule-based approaches [35, 36] to shape intermediate rewards, Monte Carlo sampling to estimate the value of the current step [13, 37, 38, 26], and using imitation of expert policies as rewards [12]. However, these methods primarily focus on optimizing local actions and lack a holistic perspective on task completion. To address this, we propose measuring the progress of stepwise actions toward overall task completion as intermediate rewards, offering a more goal-oriented signal for RL.

7 Conclusion

In this work, we propose Stepwise Progress Attribution (SPA), a novel framework designed to tackle the challenge of sparse and delayed rewards in reinforcement learning. SPA redistributes the final reward into fine-grained, stepwise signals, enabling more precise credit assignment by aligning each incremental action with overall task progress. We validate the effectiveness of SPA through extensive experiments on representative agent benchmarks. Across all tasks, SPA consistently achieves state-of-the-art performance, significantly improving both success rates and grounding accuracy. These results demonstrate that by offering more informative intermediate feedback, SPA provides a practical and scalable solution for training LLM agents on complex, long-horizon tasks.

References

- [1] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.
- [2] Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*, 2025.
- [3] Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-rl: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.
- [4] Hongru Wang, Cheng Qian, Wanjuan Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Otc: Optimal tool calls via reinforcement learning. *arXiv preprint arXiv:2504.14870*, 2025.
- [5] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.
- [6] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854, 2023.
- [7] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658, 2024.
- [8] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.
- [9] Hanlin Wang, Chak Tou Leong, Jian Wang, and Wenjie Li. E2cl: Exploration-based error correction learning for embodied agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7626–7639, 2024.
- [10] Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. Steca: Step-level trajectory calibration for llm agent learning. *arXiv preprint arXiv:2502.14276*, 2025.
- [11] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [12] Zhirui Deng, Zhicheng Dou, Yutao Zhu, Ji-Rong Wen, Ruibin Xiong, Mang Wang, and Weipeng Chen. From novice to expert: Llm agent policy optimization via step-wise reinforcement learning. *arXiv preprint arXiv:2411.03817*, 2024.
- [13] Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

- [14] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.
- [15] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [16] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.
- [17] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.
- [18] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.
- [19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [20] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- [21] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, 2024.
- [22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [24] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36:31967–31987, 2023.
- [25] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, 2023.
- [26] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! Llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1556–1572, 2024.
- [27] Hind Taud and Jean-Francois Mas. Multilayer perceptron (mlp). In *Geomatic approaches for modeling land change scenarios*, pages 451–455. Springer, 2017.
- [28] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.
- [29] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

- [30] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [31] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.
- [32] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*, 2024.
- [33] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.
- [34] Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. Atlas: Agent tuning via learning critical steps. *arXiv preprint arXiv:2503.02197*, 2025.
- [35] Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*, 2024.
- [36] Shihan Dou, Yan Liu, Haoxiang Jia, Limao Xiong, Enyu Zhou, Wei Shen, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, et al. Stepcode: Improve code generation with reinforcement learning from compiler feedback. *arXiv preprint arXiv:2402.01391*, 2024.
- [37] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.
- [38] Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.
- [39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [40] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

A Theoretical Analyses of the Progress Estimation

In this section, we analyze the proposed reward decomposition approach, where the **progress estimator** $\mathcal{E}_\gamma$ is trained to predict stepwise contributions. We demonstrate that this approach preserves the original policy optimization objective and does not alter the policy gradient, ensuring theoretical consistency.

Consider an episodic MDP (or POMDP over interaction histories) with a state (or history) sequence $e_0 \rightarrow e_1 \rightarrow \dots \rightarrow e_n$, actions $a_t \in \mathcal{A}$, transition dynamics $P(e_t | e_{t-1}, a_t)$, and a sparse reward function defined as:

$$r_t = \begin{cases} 0, & t < n, \\ r \in [0, 1], & t = n. \end{cases} \quad (11)$$

To replace the sparse terminal reward r with dense, stepwise feedback, we introduce a **Progress Estimator** $\mathcal{E}_\gamma$. The estimator predicts the per-step contribution $\hat{c}(s_t, a_t)$, which quantifies the contribution of each action a_t in state s_t to the overall task completion. Specifically, we define:

$$\hat{c}(s_t, a_t) = \text{MLP}(\mathbf{h}_t), \quad \mathbf{h}_t = f_{\pi_\gamma}(s_t, a_t), \quad (12)$$

where f_{π_γ} extracts the contextual embedding of the state-action pair (s_t, a_t) from the pre-trained policy model π_γ .

The predicted contribution $\hat{c}(s_t, a_t)$ decomposes the reward into dense, stepwise signals using a potential function Φ_ψ , such that:

$$\hat{c}(s_t, a_t) = \Phi_\psi(e_t) - \Phi_\psi(e_{t-1}), \quad (13)$$

where Φ_ψ assigns a scalar potential value to each interaction history e_t . By directly predicting the difference $\Phi_\psi(e_t) - \Phi_\psi(e_{t-1})$ using $\mathcal{E}_\gamma$, we bypass the need to explicitly compute $\Phi_\psi(e_t)$, making the training efficient and focused on the stepwise contributions.

The cumulative reward under this decomposition is:

$$\hat{R} = \sum_{t=1}^n \hat{c}(s_t, a_t) = \Phi_\psi(e_n) - \Phi_\psi(e_0). \quad (14)$$

By enforcing $\Phi_\psi(e_0) = 0$, we ensure $\hat{R} = \Phi_\psi(e_n)$. To align $\hat{R}$ with the ground truth reward r , we train $\mathcal{E}_\gamma$ to minimize the following objective.

The Progress Estimator $\mathcal{E}_\gamma$ is trained to predict contributions $\hat{c}(s_t, a_t)$ such that the cumulative reward $\hat{R}$ matches the observed ground truth reward r . This is achieved by minimizing the **sum-of-deltas loss** over trajectories $\tau \sim D_{\text{expl}}$:

$$\mathcal{L}_{\text{tel}}(\gamma) = \mathbb{E}_{\tau \sim D_{\text{expl}}} [(\hat{R} - r)^2], \quad (15)$$

where $\hat{R} = \sum_{t=1}^n \hat{c}(s_t, a_t)$. Under standard assumptions (e.g., sufficient model capacity, i.i.d. sampling, and convexity of the expected risk), the minimizer of $\mathcal{L}_{\text{tel}}$ satisfies:

$$\hat{R} = \sum_{t=1}^n \hat{c}(s_t, a_t) \rightarrow \mathbb{E}[r | e_n]. \quad (16)$$

This ensures that the cumulative contributions predicted by $\mathcal{E}_\gamma$ align with the observed task completion score r .

Using the dense, stepwise contributions $\hat{c}(s_t, a_t)$ in place of the sparse terminal reward r , we compute the policy gradient under the REINFORCE objective [11]:

$$\nabla_\theta J(\theta) = \mathbb{E} \left[\sum_{t=1}^n \nabla_\theta \log \pi_\theta(a_t | e_{t-1}) G_t \right], \quad (17)$$

where $G_t = \sum_{k=t}^n r_k$ is the return from step t onward. Substituting the constructed reward $\hat{c}(s_t, a_t)$, the return becomes:

$$G_t = \sum_{k=t}^n \hat{c}(s_k, a_k) = \Phi_\psi(e_n) - \Phi_\psi(e_{t-1}). \quad (18)$$

The policy gradient then becomes:

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_{t=1}^n \nabla_{\theta} \log \pi_{\theta}(a_t | e_{t-1}) (\Phi_{\psi}(e_n) - \Phi_{\psi}(e_{t-1})) \right]. \quad (19)$$

Since $\Phi_{\psi}(e_n)$ depends only on the terminal state e_n , terms involving $\Phi_{\psi}(e_{t-1})$ telescope to zero in expectation. Therefore, the gradient simplifies to:

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_{t=1}^n \nabla_{\theta} \log \pi_{\theta}(a_t | e_{t-1}) G_t \right], \quad (20)$$

which is identical to the gradient under the original sparse reward. This demonstrates that training $\mathcal{E}_{\gamma}$ and using the predicted contributions $\hat{c}(s_t, a_t)$ does not alter the policy optimization objective.

In summary, the progress estimator $\mathcal{E}_{\gamma}$ directly predicts stepwise contributions $\hat{c}(s_t, a_t)$, which decompose the terminal reward r into dense, meaningful feedback. By predicting $\hat{c}(s_t, a_t) = \Phi_{\psi}(e_t) - \Phi_{\psi}(e_{t-1})$, the estimator avoids explicitly computing the potential values Φ_{ψ} while ensuring that the cumulative reward $\hat{R}$ matches r . Crucially, this reward decomposition does not alter the policy optimization objective, preserving the equivalence of the policy gradient updates while enabling dense credit assignment.

B PPO's Limitations in Addressing Delayed Rewards

Assume that the value network V_{ϕ} has converged to the true discounted return

$$V_{\phi}(s_t) = \mathbb{E} \left[\sum_{k=t+1}^n \gamma^{k-t-1} r_k \right]. \quad (21)$$

In the sparse-reward setting, where $r_k = 0$ for all $k < n$ and only $r_n \in \{0, 1\}$ may be nonzero, this expectation collapses to

$$V_{\phi}(s_t) = \gamma^{n-t-1} \mathbb{E}[r_n], \quad V_{\phi}(s_n) = 0. \quad (22)$$

By definition, the one-step TD-error at time t is

$$\delta_t = r_{t+1} + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t). \quad (23)$$

For each intermediate step $t = 1, \dots, n-2$, since $r_{t+1} = 0$ and

$$\gamma V_{\phi}(s_{t+1}) = \gamma [\gamma^{n-(t+1)-1} \mathbb{E}[r_n]] = \gamma^{n-t-1} \mathbb{E}[r_n] = V_{\phi}(s_t), \quad (24)$$

it follows that

$$\delta_t = 0, \quad t = 1, \dots, n-2. \quad (25)$$

At the penultimate step $t = n-1$, using $V_{\phi}(s_n) = 0$ one obtains

$$\delta_{n-1} = r_n + \gamma V_{\phi}(s_n) - V_{\phi}(s_{n-1}) = r_n - \mathbb{E}[r_n], \quad (26)$$

which in general does not vanish.

Under Generalized Advantage Estimation (GAE) [20], the advantage estimate becomes

$$\hat{A}_t = \sum_{k=0}^{n-1-t} (\gamma\lambda)^k \delta_{t+k} = (\gamma\lambda)^{n-1-t} \delta_{n-1}, \quad (27)$$

and the resulting PPO policy gradient is

$$\nabla_{\phi} J \approx \mathbb{E} \left[\sum_{t=1}^{n-1} \nabla_{\phi} \log \pi_{\phi}(a_t | s_t) \hat{A}_t \right] = \mathbb{E} \left[\sum_{t=1}^{n-1} (\gamma\lambda)^{n-1-t} \delta_{n-1} \nabla_{\phi} \log \pi_{\phi}(a_t | s_t) \right]. \quad (28)$$

Whenever $\gamma\lambda < 1$, the factor $(\gamma\lambda)^{n-1-t}$ decays exponentially in the temporal gap $n-1-t$. Early actions thus receive vanishing weight in the gradient, and a single terminal reward cannot effectively propagate back to decisions made many steps before the end. Consequently, PPO with GAE proves ineffective for long-horizon, sparse-reward, multi-turn tasks.

Table 4: Statistics of the benchmarks used in our experiments.

Benchmark	Train	Test	#Actions	#Avg./Max. Turns
Webshop	1938	200	-	3.6 / 9
ALFWorld	2,851	134	13	8.0 / 20
VirtualHome	4,920	247	40	11.5 / 20

C Additional Details of Agent Benchmarks

WebShop² [22] is a simulated online shopping platform designed as an environment for agents to navigate and make purchases in response to user-provided instructions. Within this environment, agents interact with the platform to evaluate and select products, ultimately executing a "buy" action when a purchase decision is made. The environment assigns a final reward upon completing the purchase, which is determined based on a set of matching heuristics that evaluate the alignment of the selected product’s attributes and price with the given instructions. This setup provides a controlled setting for studying decision-making and optimization in goal-oriented agent behavior.

ALFWorld³ [17] offers interactive TextWorld environments that are meticulously aligned with the embodied environments introduced in ALFRED [17]. This framework challenges agents to navigate complex household settings and execute high-level instructions, thereby testing their ability to perform practical tasks. The dataset is structured into two distinct evaluation sets: a seen set, designed to assess in-distribution generalization, and an unseen set, which comprises novel task instances to evaluate out-of-distribution generalization capabilities. At the conclusion of each trajectory, the environment provides a binary reward, indicating whether the agent has successfully completed the assigned task. This setup facilitates a clear and measurable assessment of agent performance in both familiar and novel scenarios.

VirtualHome⁴ [18] is a comprehensive dataset that encompasses 292 high-level household tasks and 1,374 unique action plans across 6,201 diverse virtual environments. The dataset was meticulously curated through detailed manual annotations provided by Amazon Mechanical Turk workers, who thoroughly labeled both the tasks and their corresponding action plans. Each entry in the dataset comprises three key components: a high-level task, a descriptive explanation, and executable action programs specifically designed to be compatible with the VirtualHome environment. To evaluate task completion, all tasks were executed within the environment, and the final state of the environment was recorded upon task completion. A task was deemed successfully completed if the state of the environment, after exploration by the LLM agent, matched the predefined target state. To ensure the dataset’s high quality, only trajectories that achieved successful final outcome rewards were retained. Additionally, it was verified that every action in the planning sequence could be executed within the environment. Following prior work [10], we utilized their high-quality dataset for our experiments and analyses.

D Implementation Details

During the construction of the base agent, we trained the model for 3 epochs. We use the AdamW optimizer [39] and cosine annealing learning rate scheduler to optimize the model parameters. For the warm-up stage, the batch size is 16 and the learning rate is set to $2e-5$. For the training of the progress estimator, we utilized all exploration data and trained the model for 1 epoch. The batch size was set to 8, and we used the Adam optimizer with a learning rate of $3e-6$, along with the WarmupLR scheduler. In the reinforcement learning phase, we also trained the model for only 1 epoch, setting the learning rate to $1e-5$. To enable more efficient training, we adopted LoRA (Low-Rank Adaptation) for parameter-efficient fine-tuning [30]. Specifically, only the LoRA parameters were updated during training, while the base model parameters remained frozen. During the inference phase, we evaluated

²<https://github.com/princeton-nlp/WebShop>

³<https://github.com/alfworld/alfworld>

⁴<https://github.com/xavierpuigf/virtualhome>

the agent’s performance using greedy decoding with the temperature set to 0. To accelerate inference, we leveraged the vLLM library [40] to optimize the generation process of the LLMs.

All experiments were conducted on a computational cluster equipped with 8 NVIDIA A6000 GPUs, each with 48GB of memory. Throughout our experiments, we used an open-source model, LLaMA-3.2-3B-Instruct, and strictly adhered to the licensing terms for academic use associated with this model. The licensing terms can be found at the following link: <https://huggingface.co/meta-llama/Llama-3.2-1B/blob/main/LICENSE.txt>.

E Limitations

While our approach demonstrates superior performance compared to baseline methods, it is important to acknowledge the following limitations of our current work:

- (1) **Dependency on Task-Specific Tuning:** Despite the strong performance of SPA across various agentic tasks, the progress estimator in our framework requires domain-specific tuning. This reliance on task-specific customization could limit the generalizability of SPA to unseen or highly diverse agentic environments, where task characteristics may differ significantly from those in our evaluated benchmarks.
- (2) **Scalability to Real-World Complexities:** While SPA is designed to handle long-horizon tasks effectively, its performance in real-world environments with extremely large action spaces or trajectories spanning thousands of steps remains unexplored. In such scenarios, the accuracy of the progress estimator may degrade, which could negatively impact the overall performance of the framework.

F Prompt Templates

We provide the inference prompt for each task, which includes a general instruction, a one-shot example, the specific task instruction, and history trajectory.

Inference Prompt

General Instruction:
Human: Interact with a household to solve a task. Imagine you are an intelligent agent in a household environment and your target is to perform actions to complete the task goal... Your response should use the following format:
Thought: <your thoughts>
Action: <your next action>

In-Context Example:
Human: The task is Drink (Drink water).
...
Task Instruction:
Human: The task is xxx.
(History trajectory)
...

Figure 5: Inference prompt template for ALFWorld and VirtualHome environment.

Inference Prompt

General Instruction:

Human: You are web shopping. I will give you instructions about what to do. You have to follow the instructions. Every round I will give you an observation and a list of available actions, you have to respond an action based on the state and instruction. You can use search action if search is available. You can click one of the buttons in clickables. An action should be of the following structure: search[keywords] click[value] If the action is not valid, perform nothing. Keywords in search are up to you, but the value in click must be a value in the list of available actions. Remember that your keywords in search should be carefully designed.
Thought: I think ...
Action: click[something]

In-Context Example:

Human: i need a long lasting 6.76 fl oz bottle of l'eau d'issey, and price lower than 100.00 dollars [SEP] Search.
...

Task Instruction:

Human: i needd xxx.
(History trajectory)
...

Figure 6: Inference prompt template for WebShop environment.