REWARDFLOW: PROPAGATING REWARD IN THE STATE GRAPHS OF AGENTIC LEARNING WITH LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) can operate as agents that interleave reasoning, action, and observation. Training them with reinforcement learning (RL) in multi-turn scenarios remains challenging due to long horizons and sparse terminal rewards, where this setting provides limited guidance for intermediate states, leading to unreliable credit assignment and diluted token-level updates. We address these limitations by proposing *RewardFlow*, a graph-based framework for reward modeling that represents agentic contexts as graphs, with states as nodes and actions as edges. RewardFlow constructs a state graph from multiple rollouts and propagates terminal rewards from successful states to all visited states using graph propagation methods such as Breadth-First Search and Personalized PageRank. This produces dense, state-wise, task-centric reward signals that indicate whether actions move the agent closer to or farther from success. Across text and visual domains on four challenging agent environments and three model sizes, RewardFlow consistently improves task success and training efficiency over strong group-based RL baselines. These results show that RewardFlow is a simple, scalable, and effective framework for mitigating sparse-reward credit assignment in agentic RL.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable generality in their reasoning capabilities, enabling multi-step thinking and decision-making. This pattern, also known as agentic reasoning, is essential for addressing complex real-world scenarios. For instance, GUI agents (Qin et al., 2025; Wang et al., 2025a) iteratively perform actions and observe GUI system transitions, requiring multi-step interaction with the external system to handle intricate operational tasks. Since general LLMs are not specifically designed for agentic reasoning, post-training methods, particularly Reinforcement Learning (RL), are widely employed to enhance these capabilities. Advanced RL algorithms such as GRPO (Shao et al., 2024), along with frameworks like Search-R1 (Jin et al., 2025), DeepResearcher (Zheng et al., 2025), and RAGEN (Wang et al., 2025b), elevate LLMs from passive chatbots to collaborative assistants, yielding significant benefits for both academia and industry.

Despite initial progress, current methods employing RL algorithms for optimizing agentic reasoning are constrained by sparse reward modeling. Although LLMs perform multiple generations within a single reasoning trajectory, they learn solely from the reward associated with the final outcome. While this RL optimization pattern, reliant on outcome rewards alone, has succeeded in single-turn reasoning tasks, such as mathematical problems, it lacks explicit supervision for the model's intermediate behaviors. This shortfall not only impairs optimization quality but also diminishes training efficiency. Nonetheless, modeling intermediate rewards is non-trivial, as there is a lack of clear and objective evaluation metrics to assess the quality of intermediate generations independent of the ultimate task-solving outcome. Prior approaches, such as constructing additional process evaluation metrics (Xia et al., 2025) and training process reward models (Lightman et al., 2023), are resource-intensive and prone to human bias, thereby reducing their efficacy and generalizability across a broad range of agentic tasks. This limitation raises a central question:

How can we objectively approximate the quality of intermediate generations in agentic learning?

To answer this question, we propose establishing an objective relationship between the final outcome and intermediate reasoning steps. Specifically, we adopt the standard formal representation for

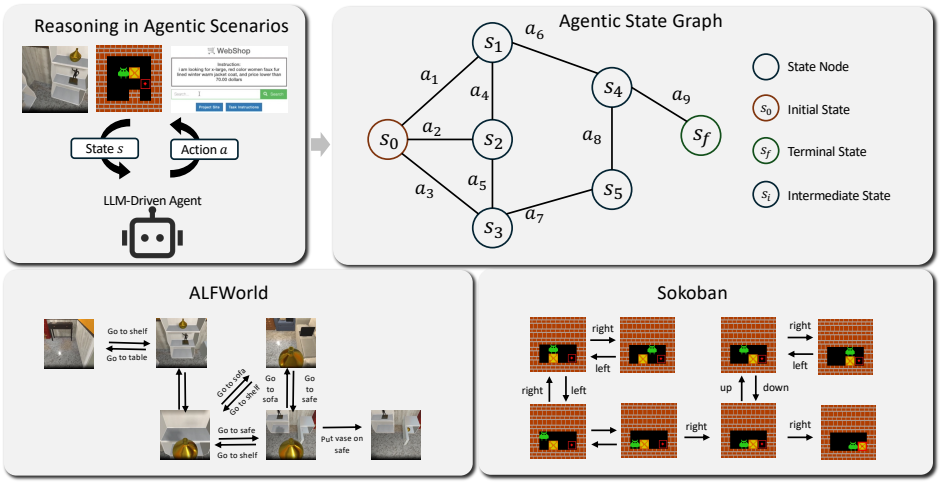

Figure 1: Illustration of projecting the rollout trajectories in agentic scenarios to an agentic state graph. Each node indicates a state, and each edge indicates an action. By constructing the agentic state graph, the connections across the states are straightforward for analysis.

agentic scenarios: the Markov Decision Process (MDP), in which agentic reasoning can be viewed as a chain alternating between states and actions. From this perspective, we observe that in general agentic scenarios, many given state can be reached from multiple prior states via diverse actions. Consequently, the MDP underlying these scenarios can be modeled as a state graph, with states as nodes and actions as edges, as shown in Fig. 1. Through such graph modeling, the topology of the state graph can reveal key informative attributes of the agentic scenario. For instance, the potential of a state to lead to successful task completion can be quantified by its distance (i.e., the minimum number of hops) to a success node associated with an outcome reward. Similarly, degrees of nodes reflect the importance, with higher degrees indicate greater importance.

Based on the constructed state graph, we propose RewardFlow, a reward modeling framework designed to address the sparse-reward issue in multi-turn agentic reasoning. Specifically, RewardFlow leverages the graph's topology to propagate the outcome reward from the success node to all intermediate nodes, employing classic graph propagation algorithms such as Breadth-First Search (Zhang & Yao, 2022) or Personalized PageRank (Page et al., 1999; Jeh & Widom, 2003). Since enumerating all global states and actions in general agentic scenarios is non-trivial, we approximate the state graph using rollouts sampled by RL algorithms, particularly group-based methods like GRPO, where multiple reasoning trajectories are generated to contribute to the graph modeling of a single scenario. However, this approximation strategy may introduce noise. For example, invalid actions generated by the policy can add extraneous edges that do not exist in the true state graph, while incomplete approximations may overlook crucial edges, rendering reward propagation unreliable. To mitigate invalid actions, RewardFlow filters them out and focuses solely on valid actions during graph construction. To address missing crucial edges, it converts the directed edges (defined by actions) into bi-directed edges, ensuring smoother and more comprehensive reward propagation. With these refined intermediate state rewards in place, we instantiate RewardFlow using GRPO to train LLMs' reasoning at the state level, enabling fine-grained strategy optimization.

We evaluate RewardFlow in both text and visual modalities across four challenging agentic environments: Sokoban (Schrader, 2018), ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022), and DeepResearch (Jin et al., 2025), using LLMs of three different scales. Empirical results show that RewardFlow substantially improves the optimization efficacy of reinforcement learning algorithms for LLM reasoning in agentic tasks, yielding a 7.4% gain on text-based benchmarks and a 25.3% average gain on visual benchmarks relative to the strongest baseline. Statistical analysis further reveals that RewardFlow produces the densest and most informative reward signal among trajectory-level and step-level process reward modeling approaches. Ablation studies on each technical component of the state-graph construction pipeline confirm the unique and indispensable contribution of every design choice. We additionally conduct targeted experiments demonstrating RewardFlow's robustness to poor rollout exploration and its consistent efficacy and efficiency over alternative process reward

modeling techniques. Collectively, these results establish RewardFlow as an effective, generalizable, and modality-agnostic framework for advancing reinforcement learning on complex agentic tasks.

In summary, our main contributions can be divided into three parts:

- **State Graph Modeling**: We formalize multi-turn agentic scenarios as structured state graphs, enabling principled analysis of the relationship across different states and actions (Sec. 3).
- **Process Reward Modeling**: We introduce RewardFlow, a graph-based framework that propagates outcome rewards to intermediate states, yielding reliable step-level reward estimates (Sec. 4).
- **Empirical validation**: We demonstrate efficacy of RewardFlow across multiple agentic scenarios and model sizes, substantially mitigating the challenges of sparse reward assignment. (Sec. 5).

2 PRELIMINARY

Notations. A graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$, where the node set $\mathcal{V}$, the relation set $\mathcal{R}$, and the edge set $\mathcal{E} = \{(x, r, v) : x, v \in \mathcal{V}, r \in \mathcal{R}\}$. We model agentic environments as a Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(\cdot \mid s, a)$ is the transition kernel over $\mathcal{S}$, and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function of the action that connect to new state, and $\gamma \in [0, 1]$ is the discount factor. In generalized agentic tasks, rewards are sparse: $r(s, a, s') = 0$ for nonterminal transitions and nonzero only when s' is terminal. Let $\mathcal{S}_{\mathrm{T}} \subseteq \mathcal{S}$ denote terminal states, namely the state when the task in the agentic environment is completed, *e.g.*, the web shopping agent submits the order to buy a good. Generally, $\mathcal{S}_{\mathrm{T}} = \mathcal{S}_{\mathrm{succ}} \cup \mathcal{S}_{\mathrm{fail}}$, where $\mathcal{S}_{\mathrm{succ}}$ denotes the state that the agent successfully fulfills the task in the agentic environments. The action to $\mathcal{S}_{\mathrm{succ}}$ is assigned a high reward, while the action to $\mathcal{S}_{\mathrm{fail}}$ is assigned a lower reward than the success actions.

RL in agentic environments. Given observation of the current state $s \in \mathcal{S}$, a policy $\pi_\theta(a \mid s)$ selects an action $a \in \mathcal{A}$, inducing transitions and rewards via $\mathcal{P}$ and r. The standard objective of RL in this agentic environment is to maximize discounted return as follows:

$$\max_{\theta} \; J(\theta) \;=\; \mathbb{E}_{\tau \sim \pi_\theta,\, \mathcal{P}} \left[\sum_{t=0}^{T-1} \gamma^t \, r(s_t, a_t, s_{t+1}) \right].$$

In sparse-reward settings where $r(s_j, a_t, s_{j+1}) = 0$ if $s_{j+1} \notin \mathcal{S}_{\mathrm{T}}$ and $r(s_t, a_t, s_{t+1}) \neq 0$ only at $s_{t+1} \in \mathcal{S}_{\mathrm{T}}$, the credit assignment and stable learning can be quite challenging.

3 CONSTRUCTING THE STATE GRAPH IN AGENTIC SCENARIOS

In this section, we construct the state graph for agentic MDPs. We begin by characterizing states as junctional and divergent, then formally define $\mathcal{G}_{\text{state}}$, approximate it through group-sampled trajectories, and refine it by state graph post-processing. An illustration is provided in Fig. 2.

Rethinking state in agentic reasoning. In typical agentic MDPs $\mathcal{M}$, many states $s \in \mathcal{S}$ are *junctional*, meaning they are reachable from multiple prior states via different actions. Formally, for a given state s, there exist distinct pairs $(s^{(i)}, a^{(i)})$ with $i = 1, \ldots, m$ ($m \geq 2$) such that $\mathcal{P}(s \mid s^{(i)}, a^{(i)}) > 0$. Conversely, many states $s' \in \mathcal{S}$ (excluding terminal states, indicating task success or failure) are *divergent*, meaning they admit multiple feasible actions. Formally, given a state s', there exist distinct actions $a'^{(j)} \in \mathcal{A}$ with $j = 1, \ldots, n$ ($n \geq 2$) such that $\mathcal{P}(s'' \mid s', a'^{(j)}) > 0$ for some successor state s'', i.e., these states connect to at least two distinct successors. From this perspective, agentic MDPs $\mathcal{M}$ can be viewed as state graphs $\mathcal{G}_{\text{state}}$ rather than linear chains.

State graph $\mathcal{G}_{\text{state}}$. We construct the state graph from the above perspective. Formally, we define the projection of an agentic environment into a state graph $\mathcal{G}_{\text{state}} = \big(\mathcal{V}_{\text{state}}, \mathcal{R}_{\text{action}}, \mathcal{E}_{\text{transition}}\big)$, where

$$\mathcal{V}_{\text{state}} = \mathcal{S}, \quad \mathcal{R}_{\text{action}} = \mathcal{A}, \quad \mathcal{E}_{\text{transition}} = \big\{(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} : \mathcal{P}(s' \mid s, a) > 0\big\}. \tag{1}$$

Each edge (s, a, s') denotes a nontrivial transition from state s to state s' under action a, with edge labels drawn from the action space. This construction embeds the dynamics of the MDP into a

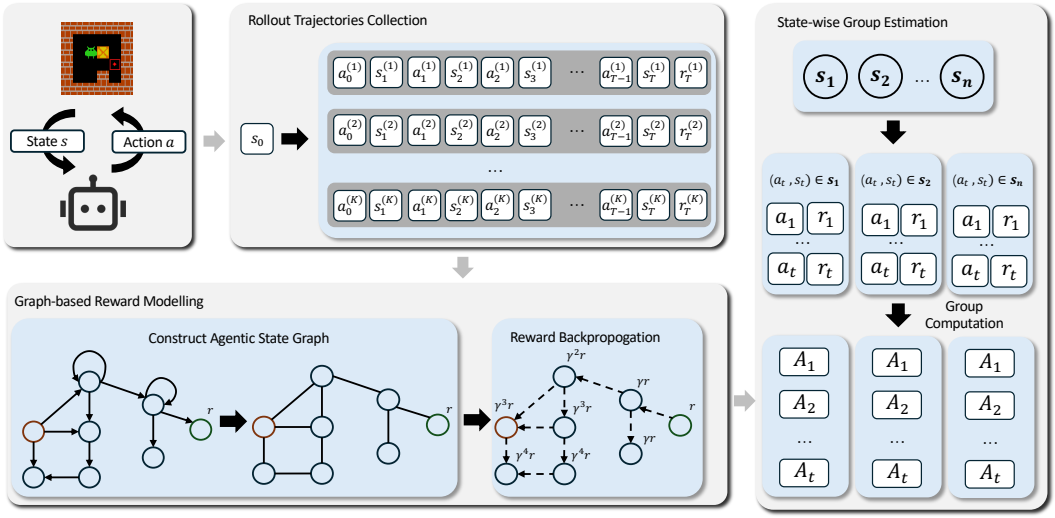

Figure 2: RewardFlow is a framework that aggregates states across multiple trajectories to construct an state graph. After eliminating self-loops, it transforms the directed graph into an undirected graph to ensure reliable reward backpropagation. For advantage estimation, RewardFlow employs state-wise group estimation and optimization, providing fine-grained guidance to optimize policy strategies in agentic scenarios.

structured graph representation, consistent with the general notation of graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$. By working with $\mathcal{G}_{\text{state}}$, we can explicitly leverage graph-structured methods to analyze reachability, connectivity, and pathways toward terminal states in agentic environments.

Constructing the "raw" state graph $\hat{\mathcal{G}}_{\text{state}}$ via group sampling. In realistic agentic settings, the full state graph is *not* directly observable: the agent only reveals states it actually visits. Any graph we build from experience is therefore an induced subgraph of the (unknown) true state graph. To improve coverage, we adopt *group sampling* (Guo et al., 2024) to generate diverse rollouts for a given task (initial state s_0), thereby exposing more states and transitions.

Let $\{\tau^{(i)}\}_{i=1}^{K}$ be K rollout trajectories collected by policy π_θ:

$$\tau^{(i)} = \big(s_0,\, a_0^{(i)},\, s_1^{(i)},\, a_1^{(i)},\, \ldots,\, a_{T_i-1}^{(i)},\, s_{T_i}^{(i)}\big), \tag{2}$$

where T_i is the length of the i-th trajectory. Together, these environment-specific preprocessing and aggregation techniques ensure high-quality state graphs, enabling RewardFlow to achieve reliable long-horizon credit assignment across both partially observable and highly stochastic domains. With these trajectories, we construct the *raw* state graph $\hat{\mathcal{G}}_{\text{state}} = \big(\hat{\mathcal{V}}_{\text{state}}, \hat{\mathcal{R}}_{\text{action}}, \hat{\mathcal{E}}_{\text{transition}}\big)$ by taking the union over observed states, actions, and transforms:

$$\hat{\mathcal{V}}_{\text{state}} = \bigcup_{i=1}^{K} \bigcup_{t=0}^{T_i} \{\phi(s_t^{(i)})\},\ \hat{\mathcal{R}}_{\text{action}} = \bigcup_{i=1}^{K} \bigcup_{t=0}^{T_i-1} \{a_t^{(i)}\},\ \hat{\mathcal{E}}_{\text{transition}} = \bigcup_{i=1}^{K} \bigcup_{t=0}^{T_i-1} \{(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)})\}. \tag{3}$$

Each triple $(s, a, s') \in \hat{\mathcal{E}}_{\text{transition}}$ encodes a directed, action-labeled edge from s to s'. Set unions deduplicate nodes and edges across trajectories. Notably, through construction, $\hat{\mathcal{G}}_{\text{state}}$ under-approximates the true state graph; increasing K and diversifying rollouts via group sampling enlarges coverage and sharpens the empirical estimates $\hat{\mathcal{P}}$ on observed (s, a) pairs.

Remark 3.1. In practical agentic environments, the observed state s is often stochastic or partially ambiguous, which can make different underlying states indistinguishable or prevent identical states from being reliably recognized. We address this challenge through state preprocessing strategies designed to ensure that states are distinguishable. A detailed description is provided in Appendix 10.1.

Refining the raw graph: pruning noisy edges and adding inverse edges. While $\hat{\mathcal{G}}_{\text{state}}$ usefully approximates the true state graph, two issues remain: (i) *invalid/no-op actions* that fail to change the

state produce spurious self-loops and noise; and (ii) many transitions are effectively one-way, creating cul-de-sacs that hinder the reliability of graph propogation. We address these issues as follows.

(1) Remove invalid edges. We discard transitions that either leave the state unchanged ($s' = s$), or correspond to actions that were executed but rejected/errored by the environment. Let $\textsc{valid}(s, a, s')$ be a predicate that is true iff the action completes successfully and results in a state change. Define

$$\hat{\mathcal{E}}^{\text{valid}}_{\text{transition}} := \big\{(s, a, s') \in \hat{\mathcal{E}}_{\text{transition}} : \; s' \neq s \;\wedge\; \textsc{valid}(s, a, s')\big\}. \tag{4}$$

(2) Add inverse edges to improve reachability. Many agentic environments admit natural inverses (*e.g.*, `open` $\leftrightarrow$ `close`, `forward` $\leftrightarrow$ `back`). Let $\text{inv} : \mathcal{A} \to \mathcal{A}$ be a *partial* inverse-action map (when an inverse is available). We augment the graph with inverse edges wherever $\text{inv}(a)$ exists:

$$\hat{\mathcal{E}}^{\text{double}}_{\text{transition}} := \hat{\mathcal{E}}^{\text{valid}}_{\text{transition}} \cup \big\{\,(s', \text{inv}(a), s) \;:\; (s, a, s') \in \hat{\mathcal{E}}^{\text{valid}}_{\text{transition}},\; \text{inv}(a) \text{ defined}\,\big\}. \tag{5}$$

Now, we obtain the refined state graph $\mathcal{G}^{\text{refined}}_{\text{state}} = \big(\hat{\mathcal{V}}_{\text{state}}, \hat{\mathcal{R}}^{\text{refined}}_{\text{action}}, \hat{\mathcal{E}}^{\text{double}}_{\text{transition}}\big)$, where

$$\hat{\mathcal{R}}^{\text{refined}}_{\text{action}} := \hat{\mathcal{R}}_{\text{action}} \cup \{\text{inv}(a) : a \in \hat{\mathcal{R}}_{\text{action}}, \text{inv}(a) \text{ defined}\}. \tag{6}$$

Pruning removes noisy transitions and actions, while inverse-edge augmentation guarantees reachability between any two states, thereby facilitating downstream RL optimization.

4 Learning with the State Graph

This section introduces RewardFlow, which converts sparse terminal feedback into dense, local guidance. Based on the constructed state graph, outcome rewards in terminal states are propagated backward using a multi-source reverse BFS to assign each state a potential reflecting its proximity to success; differences in this potential across consecutive states provide shaped, step-level feedback. Using node-local baselines, we compute state-wise group advantages and update the policy with a PPO-style objective, enabling fine-grained guidance. The full method is presented in Algorithm 1.

Reward propagation on the state graph (dense shaping). Sparse terminal rewards impede credit assignment. We convert them into *state-wise dense guidance* by propagating success backwards over the refined state graph $\mathcal{G}^{\text{refined}}_{\text{state}}$. The dense reward shaping includes the following three steps.

(1) Shortest-hop distance to success. Let $\mathcal{S}_{\text{succ}} \subseteq \hat{\mathcal{V}}_{\text{state}}$ be the set of success terminals. We run a multi-source *inverse* breadth-first-search (BFS) from all $s^\star \in \mathcal{S}_{\text{succ}}$ over the directed edges in $\hat{\mathcal{E}}^{\text{double}}_{\text{transform}}$ to compute the unweighted shortest-hop distance

$$d(s) := \min_{s^\star \in \mathcal{S}_{\text{succ}}} \text{dist}_{\text{hop}}(s \rightsquigarrow s^\star), \tag{7}$$

with the conventions $d(s) = 0$ for $s \in \mathcal{S}_{\text{succ}}$ and $d(s) = \infty$ if no success is reachable from s. Analytically, the complexity of this BFS is $O(|\hat{\mathcal{V}}_{\text{state}}| + |\hat{\mathcal{E}}^{\text{double}}_{\text{transform}}|)$.

(2) Reward propagation. Assign each state a geometric potential that decays with hop distance:

$$R(s) \;=\; \gamma^{\,d(s)}, \qquad \gamma \in (0, 1]. \tag{8}$$

Thus $R(s) = 1$ for success states, and R decreases monotonically as s gets farther from success.

(3) Action-level shaping. To score individual decisions, we define an action-shaped reward on any observed transition (s_t, a_t, s_{t+1}) (or any edge in the refined graph) by the potential difference

$$\tilde{r}(s_t, a_t) \;=\; R(s_{t+1}) - R(s_t). \tag{9}$$

Intuitively, $\tilde{r} > 0$ if a_t moves the agent closer to success, $\tilde{r} < 0$ if it moves it farther away, and $\tilde{r} = 0$ on equally-good plateaus. One may also use the potential-based shaping variant $\tilde{r}(s_t, a_t) = \gamma R(s_{t+1}) - R(s_t)$ to preserve optimal policies when *adding* $\tilde{r}$ to the environment reward.

This propagation produces dense, globally consistent feedback aligned with task completion: every state obtains a progress signal $R(s)$, and every action receives immediate credit $\tilde{r}$ reflecting its contribution toward (or away from) success, stabilizing and accelerating downstream RL optimization.

Algorithm 1 RewardFlow

Require: Initial policy π_θ; behavior copy θ_{old}; task distribution $p(X)$; discount $\gamma \in (0, 1]$; clipping $\epsilon > 0$; group size (rollouts) K; horizon H; validity predicate $\text{VALID}(\cdot)$; partial inverse map $\text{inv} : \mathcal{A} \rightharpoonup \mathcal{A}$; normalizer F_{norm}
for each training iteration **do**
  Set behavior policy: $\theta_{\text{old}} \leftarrow \theta$
  // Group-sampled rollouts
  Sample task $x \sim p(X)$ and collect K trajectories $\{\tau^{(i)}\}_{i=1}^{K}$ from $\pi_{\theta_{\text{old}}}(\cdot \mid x)$ (Eq. (2))
  // State graph construction & refinement
  Build raw graph $\hat{\mathcal{G}}_{\text{state}} = (\hat{\mathcal{V}}_{\text{state}}, \hat{\mathcal{R}}_{\text{action}}, \hat{\mathcal{E}}_{\text{transition}})$ (Eq. (3))
  Prune invalid edges and obtain $\hat{\mathcal{E}}^{\text{valid}}_{\text{transition}}$ (Eq. (4))
  Add inverse edges and obtain: $\hat{\mathcal{E}}^{\text{double}}_{\text{transition}}$ (Eq. (5))
  $\mathcal{G}^{\text{refined}}_{\text{state}} \leftarrow (\hat{\mathcal{V}}_{\text{state}}, \hat{\mathcal{R}}^{\text{refined}}_{\text{action}}, \hat{\mathcal{E}}^{\text{double}}_{\text{transition}})$ (Eq. (6))
  // Reward propagation (dense shaping)
  Identify success terminals $\mathcal{S}_{\text{succ}}$ (from task spec or terminal labels)
  Compute hop distances $d(s)$ to nearest success via multi-source reverse BFS (Eq. (7))
  Compute state potentials $R(s) \leftarrow \gamma^{d(s)}$ (Eq. (8))
  For each observed $(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)})$, set $\tilde{r}_t^{(i)} \leftarrow R(s_{t+1}^{(i)}) - R(s_t^{(i)})$ (Eq. (9))
  // State-wise grouping and advantages
  For each state $\boldsymbol{s} \in \hat{\mathcal{V}}_{\text{state}}$, build $G(\boldsymbol{s})$ (Eq. (10))
  For each $\boldsymbol{s}$, compute baseline $\mu(\boldsymbol{s})$ and $\sigma(\boldsymbol{s})$ (Eq. (11))
  For each sample with $s_t^{(i)} = \boldsymbol{s}$, compute $A(\boldsymbol{s}, a_t^{(i)}) \leftarrow \big(\tilde{r}_t^{(i)} - \mu(\boldsymbol{s})\big)/\sigma(\boldsymbol{s})$ (Eq. (12))
  // Policy update
  Update θ by maximizing $\mathcal{J}_{\text{RewardFlow}}(\theta)$ (Eq. (13) and Eq. (14))
end for

Group-based, state-wise advantage estimation. We estimate advantages at the *state* level by treating the average shaped reward within a state as the baseline (expected reward). Unlike prior work that computes *trajectory-level* advantages (Feng et al., 2025b; Guo et al., 2024), we leverages the *propagated, state-wise* signal (Eq. 8) to score each decision locally, yielding low-variance, per-action credit assignment. This advantage estimation is achieved by the following three steps.

(1) Grouping by state. For any state-graph state $\boldsymbol{s} \in \mathcal{V}_{\text{state}}$, collect all action–reward observations taken when the agent visited $\boldsymbol{s}$ across the rollouts (recall $\tilde{r}(s_t, a_t) = R(s_{t+1}) - R(s_t)$ from Eq. (9)):

$$G(\boldsymbol{s}) \;:=\; \bigcup_{i=1}^{K} \bigcup_{t=0}^{T_i-1} \Big\{ \big(a_t^{(i)},\, \tilde{r}_t^{(i)}\big) \;:\; \big(\boldsymbol{s}, a_t^{(i)}, s_{t+1}^{(i)} \in \hat{\mathcal{E}}^{\text{double}}_{\text{transition}}\big)\Big\}. \tag{10}$$

This forms a set of samples drawn from the action choices available at $\boldsymbol{s}$ and their rewards.

(2) State-wise baseline and normalization. Let

$$\mu(\boldsymbol{s}) := \text{mean}\Big(\big\{\tilde{r}_t^{(j)} \mid (a_t^{(j)}, \tilde{r}_t^{(j)}) \in G(\boldsymbol{s})\big\}\Big), \sigma(\boldsymbol{s}) := F_{\text{norm}}\Big(\big\{\tilde{r}_t^{(j)} \mid (a_t^{(j)}, \tilde{r}_t^{(j)}) \in G(\boldsymbol{s})\big\}\Big) > 0, \tag{11}$$

where F_{norm} is any positive scale functional (e.g., sample standard deviation $+\varepsilon$, median absolute deviation, or simply 1 if no normalization is desired).

(3) Group-based advantage. For a sample $(\boldsymbol{s}, a_t^{(i)})$, we define the state-wise advantage as

$$A\big(\boldsymbol{s}, a_t^{(i)}\big) \;=\; \frac{\tilde{r}_t^{(i)} - \mu(\boldsymbol{s})}{\sigma(\boldsymbol{s})}. \tag{12}$$

Because the baseline $\mu(\boldsymbol{s})$ depends only on the state, $\mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{s})}[A(\boldsymbol{s}, a)] = 0$, preserving an unbiased policy-gradient while reducing variance. Intuitively, actions that outperform the state's average receive positive advantage, underperforming ones receive negative advantage, directly reflecting their relative contribution toward task completion from that state. Note that we set $\mu(\boldsymbol{s}) = 0$ and $\sigma(\boldsymbol{s}) = 1$ to prevent the unlearnable condition that the node has only one out degree.[1] Note that for rare states

[1]We explicitly discuss this condition in Appendix 10.4

with few samples, $\mu(\boldsymbol{s})$ and $\sigma(\boldsymbol{s})$ can be smoothed with shrinkage toward global statistics or updated online via exponential moving averages; F_{norm} is clipped by $\varepsilon > 0$ to ensure numerical stability.

Policy optimization with state-wise advantages. Given the state-wise groups and advantages $A(\boldsymbol{s}, a)$ from Eq. (12), we update the policy using a clipped surrogate objective (PPO-style) that upweights actions with high advantage and downweights those with low (or negative) advantage. Let $\{\tau^{(i)}\}_{i=1}^{K}$ be rollouts sampled from the behavior policy $\pi_{\theta_{\text{old}}}$, with lengths T_i, and let $N := \sum_{i=1}^{K} T_i$ be the total number of (state, action) samples. Define the per-step importance ratio

$$\rho_t^{(i)} := \frac{\pi_\theta\big(a_t^{(i)} \mid s_t^{(i)}, x\big)}{\pi_{\theta_{\text{old}}}\big(a_t^{(i)} \mid s_t^{(i)}, x\big)}, \tag{13}$$

where x denotes optional task/context information. The REWARDFLOW objective is

$$\mathcal{J}_{\text{RewardFlow}}(\theta) = \mathbb{E}_{\{\tau^{(i)}\}_{i=1}^{K} \sim \pi_{\theta_{\text{old}}}}\Big[\frac{1}{N}\sum_{i=1}^{K}\sum_{t=0}^{T_i-1} \min\Big(\rho_t^{(i)} A\big(s_t^{(i)}, a_t^{(i)}\big),\ \text{clip}\big(\rho_t^{(i)},\, 1-\epsilon,\, 1+\epsilon\big) A\big(s_t^{(i)}, a_t^{(i)}\big)\Big)\Big], \tag{14}$$

with $\epsilon > 0$ the clipping parameter and $\text{clip}(z, 1-\epsilon, 1+\epsilon) := \max\big(\min(z, 1+\epsilon), 1-\epsilon\big)$. The expectation is implemented by the empirical average over the collected rollouts. Here, the state-wise baseline makes $A(\boldsymbol{s}, a)$ zero-mean per state, yielding unbiased, reduced-variance gradients. The clipped ratio prevents overly large policy updates when $\rho_t^{(i)}$ deviates from 1, stabilizing training while steering probability mass toward actions that outperform the state's average.

5 EXPERIMENTS

5.1 EXPERIMENT SETUP

Datasets. We evaluate RewardFlow on the representative visual agent environment: **Sokoban** (Schrader, 2018), on the text-based environment **ALFWorld** (Shridhar et al., 2021) and **WebShop** (Yao et al., 2022), and on the highly stochastic and ambiguous environment of **DeepResearch** (Jin et al., 2025) tasks. We show brief descriptions as follows:

- **Sokoban** is a classic puzzle game in which agents must observe and analyze the grid layout and push boxes onto target locations, taxing spatial understanding and long-horizon planning. In this work, we evaluate models on 6×6 Sokoban boards.
- **ALFWorld** is a synthetic text-based simulator aligned with the embodied benchmark ALFRED (Shridhar et al., 2020), comprising 3,827 household tasks spanning six types: *Pick & Place* (Pick), *Examine in Light* (Look), *Clean & Place* (Clean), *Heat & Place* (Heat), *Cool & Place* (Cool), and *Pick Two & Place* (Pick2).
- **WebShop** is a challenging, large-scale web navigation benchmark where the agent must fulfill natural-language shopping instructions by issuing text commands such as "search" and "click" to web pages with over 1.18 million real Amazon products. The state space is effectively unbounded, with an extremely high branching factor (often numerous clickable elements or search results per page) and long horizons (average above 5 necessary actions per successful episode).
- **DeepResearch** is an agentic framework in which an agent uses a search engine to retrieve documents for answering knowledge-intensive questions. The environment state is defined as the list of documents returned by the search engine, and the agent's actions are the search queries it generates. A major challenge stems from the high sensitivity of retrieval results: semantically similar queries often yield substantially different document sets, which complicates accurate trajectory modeling, knowledge graph construction, and reward modeling. We evaluate on TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), HotpotQA (Yang et al., 2018), and 2WikiMultiHopQA (Ho et al., 2020). Among these, TriviaQA and PopQA primarily assess recall of single facts, whereas HotpotQA and 2WikiMultiHopQA require reasoning over multiple pieces of knowledge from diverse sources, typically necessitating multi-turn search interactions to reach the correct answer.

Table 1: Performance of RewardFlow compared with other RL methods. We report the average success rate (%) for each subtask as well as the overall result.

Type	Method	ALFWorld							WebShop	Sokoban	
		Pick	Look	Clean	Heat	Cool	Pick2	All	Succ.	Score	Succ.
Qwen2.5-1.5B-Instruct											
Prompting	Base	5.9	5.5	3.3	9.7	4.2	0	4.1	5.2	-	-
RL Training	RLOO	56.2	46.7	62.5	50.0	43.5	27.8	49.2	54.7	-	-
RL Training	GRPO	62.9	**53.3**	50.0	40.0	**45.8**	38.1	50.0	42.2	-	-
RL Training	GiGPO	59.4	46.7	62.5	44.4	43.5	**56.3**	53.1	55.5	-	-
RL Training	RewardFlow	**94.1**	40	**85.0**	**93.3**	21.4	**56.3**	**65.6**	**60.9**	-	-
Qwen2.5-(VL)-3B-Instruct											
Prompting	Base	36.4	42.9	9.1	7.1	5.3	4.5	16.4	1.6	0.5	14.1
RL Training	RLOO	78.1	46.7	75.0	31.3	43.5	33.3	55.5	59.4	1.0	22.7
RL Training	GRPO	79.8	57.1	75.8	21.4	31.6	36.4	56.2	53.9	1.3	26.6
RL Training	GiGPO	82.1	50.0	**76.9**	**53.3**	**60.9**	50.0	64.8	59.4	1.2	21.9
RL Training	RewardFlow	**94.6**	**70.0**	70.0	45.5	45.0	**60.0**	**69.5**	**60.9**	**2.2**	**49.2**
Qwen2.5-(VL)-7B-Instruct											
Prompting	Base	33.3	13.3	10.7	0	4.3	0	10.9	7.8	0.9	18.8
RL Training	RLOO	90.0	85.7	88.0	50.0	**85.7**	37.5	75.0	72.7	1.0	21.9
RL Training	GRPO	**92.3**	53.3	90.5	**77.8**	69.6	47.6	75.0	70.3	1.0	23.4
RL Training	GiGPO	84.6	80.0	96.0	71.4	73.9	**84.0**	82.8	72.7	1.4	34.4
RL Training	RewardFlow	84.6	**86.7**	**100.0**	71.4	69.6	**84.0**	**83.6**	**73.4**	**3.0**	**62.4**

Table 2: Performance of RewardFlow compared with GRPO on DeepResearch tasks. We report the average success rate (%) for each dataset.

Method	Single-hop QA		Multi-hop QA		Avg.
	TriviaQA	PopQA	HotpotQA	2Wiki	
Base	48.038	30.037	25.816	25.666	32.389
GRPO	57.373	41.426	33.108	27.802	39.927
RewardFlow	**58.498**	**42.233**	**33.636**	**33.704**	**42.018**

Baselines. We compare the performance of Rewardflow with three competitive RL training baselines on Sokoban, ALFWorld, and WebShop: RLOO (Ahmadian et al., 2024), GRPO (Shao et al., 2024), that have proven effective for language reasoning. We also include GiGPO (Feng et al., 2025b), the RL algorithm tailored to agent environments, which also enables state-wise advantage estimation by propagating the reward from the terminal state to prior states along the sampling trajectory. For DeepResearch environments, we compare RewardFlow with the GRPO algorithm.

Training settings. We employ Qwen2.5-VL-3B/7B-Instruct (Bai et al., 2025) to train on the Sokoban benchmark, and Qwen2.5-1.5/3/7B-Instruct on the ALFWorld and WebShop benchmark. For Sokoban, ALFWorld, and WebShop, we train models with 100 steps. For each training step, we randomly sample 16 distinct data (*i.e.*, interactive environments in agentic scenarios), with fixed 8 group-based sampling counts. For DeepResearch, we train Qwen2.5-3B-Instruct with 200 steps, 64 training batch sizes, and 5 rollout sampling numbers. For validation, we randomly sample 128 validation set and report the checkpoint's performance on the validation set every 10 steps. As for GiGPO, we fix the decay parameter γ as 0.95. For RewardFlow, we fix the decay parameter of BFS as 0.9. We conduct the experiments on Qwen2.5-7B with 4 NVIDIA A100 GPUs (80GB) or 4 NVIDIA H20 GPUs (90GB), while 2 A100s or 2 H20s are on Qwen2.5-1.5/3B. We leverage the Verl-Agent (Feng et al., 2025b) as infrastructure, which is an agent-friendly RL framework.

5.2 Main Results

We show our experimental results on Tabs. 1 and 2. As can be seen, while the Qwen models before training demonstrate poor performance in agentic scenarios, the RL training algorithm exhibits significant capability. Notably, our proposed RewardFlow significantly surpasses other RL methods, with 22.6% on Qwen2.5-3B-VL-Instruct and 28.1% on Qwen2.5-7B-VL-Instruct in success rate compared to the second-best algorithm in Sokoban benchmarks, and achieves 12.5% performance

Table 3: Performance of RewardFlow with or without each component on ALFWorld. We report the average success rate (%).

Method	ALFWorld
RewardFlow (w/o state preprocessing)	53.9
RewardFlow (w/o filtering)	60.2
RewardFlow (w/o reversed edges)	65.6
RewardFlow	**69.5**

Table 4: Comparison of RewardFlow with process reward modeling baselines on ALFWorld using Qwen2.5-1.5B-Instruct. We report success rate (%) and average training time per step.

Training Method	Success Rate	Time/Step (s)
GRPO + PRM	31.3	513.0
PPO	43.0	502.5
RewardFlow	**65.6**	**320.3**

improvement in ALFWorld on average. In WebShop, RewardFlow consistently outperforms prior RL methods that achieve the highest success rate at every model scale, with especially large gains on smaller models (+5.4% on Qwen2.5-1.5B-Instruct). For trajectory-level optimization methods like RLOO and GRPO, they lack state-wise reward assignment, which limits their performance in further improving the reasoning in agentic scenarios. For GiGPO, while it overcomes the limitation of assigning the state-wise rewards, the coarse reward estimation along the rollout trajectory may involve noise. For example, the invalid actions taken in the rollout trajectories will also be assigned a decayed reward. In contrast, RewardFlow removes these states caused by invalid actions and constructs a state graph for all valid states, which enables reliable reward modeling for each state and action, consistently surpassing the three baseline algorithms. In DeepResearch (Tab. 2), RewardFlow also demonstrates efficacy that outperforms the GRPO algorithms in the highly stochastic and ambiguous DeepResearch environment, achieving an average improvement of up to 2.09%. Compared with general QA tasks, the performance gain in multi-hop QA tasks is more significant, with a 5.902% improvement compared with the GRPO baseline in 2WikiMultiHopQA. These gains demonstrate RewardFlow's robustness: by propagating rewards across aggregated, semantically equivalent states, it delivers stable training signals despite severe retrieval noise. This validates the efficacy of our approach in real-world settings characterized by high stochasticity and observational ambiguity. These results show the remarkable efficacy of RewardFlow on agentic scenarios.

5.3 Understanding the Performance of RewardFlow

Beyond showing the reasoning capability, we show empirical results of ablations and analysis to further investigate the performance of RewardFlow.

Contributions of Each Technical Component in RewardFlow. We conduct ablation studies to evaluate the individual contributions of each component (state preprocessing, invalid action filtering, and edge reversal) to overall optimization performance. As shown in Tab. 3, all proposed components yield positive contributions, demonstrating their effectiveness in process reward modeling. Specifically, removing state preprocessing leads to a 15.6% performance drop, confirming its critical role in constructing high-quality state graphs in partially observable text environments. Eliminating invalid action filtering causes a 9.3% decline, indicating that unfiltered invalid or self-loop actions introduce noise into the graph, which in turn impairs accurate advantage estimation. Finally, although the edge reversal strategy may introduce some non-existent edges, it facilitates more comprehensive reward propagation without disrupting advantage estimation, resulting in an additional 3.9% improvement. These results collectively validate the positive impact of each component on reliable graph construction, effective reward propagation, and robust advantage estimation. Detailed discussion is in Appendices 10.1, 10.2, and 10.3.

Comparing with Process Reward Models. To compare process reward modeling accuracy against LLM-based alternatives, we evaluate RewardFlow against two baselines in Tab. 4: state-wise GRPO using Qwen2.5-7B-Instruct as a process reward model (PRM) for per-action quality scoring, and standard PPO with its implicit token-wise reward model. RewardFlow substantially outperforms both in efficacy and efficiency: it achieves +22.6% over PPO and +34.3% over GRPO+PRM. Computationally, PPO and GRPO+PRM incur heavy overhead from reward/critic model inference, whereas RewardFlow derives verifiable, task-aligned dense rewards directly via graph propagation from terminal signals and observed topology, enabling precise credit assignment at minimal cost. Detailed analysis is in Appendix 11.2.

Table 5: Ablation on exploration diversity (number of rollouts per training step). We report the success rate (%). For RewardFlow, we report average graph statistics in the induced graph.

Task	Method	Rollouts	Avg. Nodes	Avg. Edges	Success Rate (%)
Sokoban	GiGPO	4	-	-	21.1
		6	-	-	35.2
		8	-	-	33.6
	RewardFlow	4	10.1	13.8	**21.1**
		6	12.3	17.1	**40.6**
		8	14.0	21.0	**70.3**
ALFWorld	GiGPO	4	-	-	28.9
		6	-	-	51.6
		8	-	-	53.1
	RewardFlow	4	17.1	28.8	**42.2**
		6	22.9	41.3	**53.1**
		8	28.8	55.9	**65.5**

Table 6: Wall-clock time breakdown per training step (seconds) for Qwen2.5-(VL)-3B models.

Component	ALFWorld	WebShop	Sokoban
Rollout Sampling	205.01	202.17	188.72
Policy Update	175.09	205.99	60.05
Reference Probability Computation	46.43	51.31	26.32
Old Step Probability Computation	47.22	52.13	26.97
Graph Construction + Reward Propagation	**1.87**	**1.47**	**2.39**

Investigating the Relationship between Exploration and Performance. To evaluate robustness to limited exploration, we ablate the number of rollouts per training step (default: 8) on Sokoban and ALFWorld, training each variant for 200 and 100 steps, respectively. Fewer rollouts reduce trajectory diversity and search-graph quality, simulating constrained exploration. As shown in Tab. 5, RewardFlow consistently matches or outperforms GiGPO on both tasks, even with severely limited rollouts. Increasing rollouts brings larger gains for RewardFlow, widening its lead over baselines. These results highlight RewardFlow's superior data efficiency and robustness to suboptimal exploration. Detailed discussion is in Appendix 11.1.

Training Efficiency. As shown in Tab. 6, the time consumption of graph construction and reward propagation takes no more than 2.39 seconds across three environments, which is highly efficient compared with other necessary stages like rollout sampling and policy update in the RL process. We provide more empirical results in Appendix. 12.1.

6 Conclusion and Further Discussion

Conclusion. In this paper, we presented RewardFlow, a graph-based framework designed to tackle the sparse reward problem in reinforcement learning for LLM-based agents in multi-turn scenarios. By modeling agentic contexts as state graphs constructed from diverse rollouts, and propagating terminal rewards backward via propagating algorithms, RewardFlow generates dense, task-aligned reward signals for intermediate states. This approach enables more reliable process reward assignment, significantly improving LLMs reasoning performance in agentic scenarios. RewardFlow represents a simple yet scalable advancement for RL in agentic learning, unlocking the potential of LLMs in long-horizon, interactive domains.

Limitations. Despite its strong performance, RewardFlow has several limitations. First, it depends on the quality of the state graph, which can degrade in highly complex environments when limited rollouts fail to capture adequate state transitions. Second, in sparse environments where most states have only one incoming or outgoing action, the graph degenerates into near-linear chains, diminishing the benefits of graph-based reward propagation. Nevertheless, RewardFlow remains a robust and scalable approach that significantly advances RL optimization for LLM-based agents. Future work could improve graph construction robustness and develop adaptive denoising methods to extend its effectiveness to even more challenging scenarios.

REFERENCES

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.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Alexi Canesse, Mathieu Petitbois, Ludovic Denoyer, Sylvain Lamprier, and Rémy Portelas. Navigation with qphil: Quantizing planner for hierarchical implicit q-learning. *arXiv preprint arXiv:2411.07760*, 2024.

Xiaocong Chen, Siyu Wang, Julian McAuley, Dietmar Jannach, and Lina Yao. On the opportunities and challenges of offline reinforcement learning for recommender systems. *ACM Transactions on Information Systems*, 2024.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025a.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025b.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

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.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

Jian Hu. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*, 2025.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pp. 271–279, 2003.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free!, 2019.

Guanghe Li, Yixiang Shan, Zhengbang Zhu, Ting Long, and Weinan Zhang. Diffstitch: Boosting offline reinforcement learning with diffusion-based trajectory stitching. In *ICML*, 2024.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2023.

Yen-Ting Lin, Di Jin, Tengyu Xu, Tianhao Wu, Sainbayar Sukhbaatar, Chen Zhu, Yun He, Yun-Nung Chen, Jason Weston, Yuandong Tian, et al. Step-kto: Optimizing mathematical reasoning through stepwise binary feedback. *arXiv preprint arXiv:2501.10799*, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *ACL*, 2023.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford infolab, 1999.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, 2023.

Max-Philipp B. Schrader. Gym-Sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

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

Michal Smolik, Vaclav Skala, and Ondrej Nedved. A comparative study of lowess and rbf approximations for visualization. In *International Conference on Computational Science and Its Applications*, pp. 405–419. Springer, 2016.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 2012.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. Evaluating mathematical reasoning beyond accuracy. In *AAAI*, 2025.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*, 2025.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *NeurIPS*, 2018.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search. In *NeurIPS*, 2024.

Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM web conference 2022*, pp. 912–924, 2022.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, et al. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. In *ICLR*, 2024.

Appendix

7 Ethics Statement

This study does not involve human subjects, dataset releases, potentially harmful insights, applications, conflicts of interest, external sponsorship, discrimination, bias, fairness concerns, privacy or security issues, legal compliance issues, or research integrity concerns.

8 LLM Usage Disclosure

This manuscript was prepared with the assistance of LLMs, which were used for refining content and ensuring grammatical accuracy. The authors take full responsibility for the entire content of the manuscript and confirm that no LLM is listed as an author.

9 Related Work

RL for LLM reasoning. Reinforcement learning (RL) (Kaelbling et al., 1996) optimizes policies to maximize rewards and has recently enhanced LLM reasoning (Xu et al., 2025). Approaches fall into two paradigms: off-policy methods train from data collected by other policies—e.g., DPO (Rafailov et al., 2023), often combined with MCTS (Zhang et al., 2024; Xie et al., 2024), though paired preferences are costly; newer variants reduce this burden via single-instance or stepwise feedback (KTO (Zhou et al., 2024), Step-KTO (Lin et al., 2025)) and by removing the reference model (SimPO (Meng et al., 2024)). On-policy methods (e.g., PPO (Schulman et al., 2017), Reinforce++ (Hu, 2025)) optimize using data from the current policy, typically with an auxiliary reward model that increases compute and risks reward hacking (Guo et al., 2025); GRPO (Shao et al., 2024), and RLOO (Ahmadian et al., 2024; Kool et al., 2019) mitigates this via group-based sampling and relative advantage estimation, with extensions such as DAPO (Yu et al., 2025) and Dr. GRPO (Liu et al., 2025) advancing optimization for complex reasoning (Guo et al., 2025).

RL for agentic scenarios. Recent advancements in RL have extended its application to train LLMs in agentic scenarios, enabling agents to interact with tools and external environments to tackle complex problems. Frameworks such as Search-R1 (Jin et al., 2025), R1-Searcher (Song et al., 2025), and DeepResearcher (Zheng et al., 2025) integrate LLMs with retrieval tools, including document vector databases and search engines, using GRPO to enhance both reasoning and tool usage capabilities. For computationally intensive tasks like mathematical reasoning, frameworks such as ReTool (Feng et al., 2025a), ToRL (Li et al., 2025), and Verl-Tool (Jiang et al., 2025) leverage Python environments to optimize agents' abilities to solve problems through code-based complex calculations. Despite their promise, these frameworks struggle with long-horizon reasoning. To address this, ARPO (Dong et al., 2025) employs an entropy-guided strategy during the rollout process to encourage exploration in high-entropy states, leading to improved trajectory collection and optimization. Similarly, SimpleTIR (Xue et al., 2025) identifies and filters out trajectories containing "void turn", which can destabilize multi-turn agentic training, ensuring more stable optimization. Notably, GiGPO (Feng et al., 2025b) capitalizes on the observation that many states across different trajectories are identical, combining state-level and trajectory-level advantage computations to achieve fine-grained reward modeling.

10 Further Method Details

10.1 State Preprocessing

The state preprocessing in RewardFlow effectively handles stochasticity and ambiguity **by aggregating semantically equivalent states using embeddings or enriching the information of states.** Here, we outline the specific challenges posed by stochasticity and ambiguity in the ALFWorld and

DeepResearch environments, followed by the targeted state-preprocessing strategies we employed (described in Section 3) and applied consistently in both settings.

For ALFWorld:

- Challenge: Although raw text observations are usually distinct, ALFWorld is partially observable and often omits critical object property changes. For example, after the agent cleans an apple, the observation may still read “You are carrying an apple” without indicating that it is now cleaned, creating ambiguous states that can degrade the quality of the state graph.
- Solution: We enrich the raw observation by automatically detecting transformative actions (e.g., clean, heat, cool) and appending the resulting property changes to the text (e.g., “You are carrying an apple [cleaned]”). This ensures that states before and after such actions are distinguishable, yielding accurate node representations and reliable aggregation.

For DeepResearch:

- Challenge: DeepResearch is inherently highly stochastic and ambiguous. Semantically similar search queries frequently return substantially different document sets, causing (1) high stochasticity (from similar actions to divergent raw states) and (2) ambiguity (functionally equivalent research progress represented by distinct observations). This leads to severe graph fragmentation if raw states are used directly.
- Solution: We perform embedding-based node aggregation using a sentence transformer. States s and s' are merged into a single super-node if their cosine similarity exceeds a threshold (default 0.9). This eliminates near-duplicate nodes, mitigates fragmentation from query paraphrasing, preserves meaningful transitions, and produces a compact, semantically coherent graph that supports stable reward propagation via BFS.

10.2 Invalid/No-Op Actions Filtering

Invalid and no-op actions arise from the LLM policy’s outputs during interaction with the environment, but they are distinct in nature. Invalid actions include cases where (1) the policy model outputs responses from which no valid action can be extracted (e.g., malformed or nonsensical text that fails parsing), or (2) the policy outputs an action that is not among the admissible actions in the current environment state (e.g., attempting an unavailable command). In contrast, no-op actions are valid outputs that the environment accepts but result in no state transition, such as self-examination or redundant commands like "look" or “examine objects” in ALFWorld, which do not alter the state. These definitions align with the VALID predicate, which checks for successful execution and state change.

Filtering invalid and no-op actions ensures a cleaner state graph, improving the accuracy of reward propagation. In practice, invalid actions can lead to erroneous states that do not exist in the true agentic environment; for instance, in ALFWorld, they trigger a fallback state like "Nothing happens," which introduces noise and distorts the graph structure by creating inaccurate nodes or edges. For no-op actions, although they are valid, retaining them adds self-loop edges that increase graph complexity without contributing to propagation: BFS computes shortest distances to success states and ignores loops, but they unnecessarily inflate computation. Moreover, for degree-sensitive methods like Personalized PageRank, self-loops can mislead reward estimation by artificially increasing node degrees, potentially skewing potentials $R(s)$. Thus, under our propagation settings, eliminating these actions refines $\hat{G}_{\text{state}}$ for reliable backpropagation from success terminals S_{succ}. Our ablation study in Tab. 3 confirms that filtering invalid and no-op actions improves overall performance by reducing noise in credit assignment.

Given invalid actions, invalid states occur when the agent executes an invalid action, prompting the environment to return feedback that does not reflect a genuine state update. In many agentic environments, such as ALFWorld, the agent may attempt actions that are not feasible given the current context, resulting in responses like "Nothing happens." This feedback is erroneously interpreted as a new state during rollout collection, even though it carries no meaningful environmental information. For instance, consider the following exemplar trajectory from ALFWorld:

ALFWORLD EXAMPLE

```
{\color{blue}
"
STEP: 1
STATE: -= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room. Looking quickly around you, you see
    a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6,
    a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1,
    a coffeemachine 1, a countertop 1, a diningtable 1, a drawer 2, a
    drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin
    1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a
    stoveburner 1, and a toaster 1.

Your task is to: put a cool apple in microwave.
ACTION: go to cabinet 1
---
STEP: 2
STATE: You arrive at cabinet 1. The cabinet 1 is closed.
ACTION: go to cabinet 1 [INVALID ACTION]
---
STEP: 3
STATE: Nothing happens. [INVALID STATE]
ACTION: open cabinet 1
---
STEP: 4
STATE: You open the cabinet 1. The cabinet 1 is open. In it, you see
    nothing.
"
}
```

If left unfiltered, this string "Nothing happens" would be treated as an ordinary node in the state graph, creating noisy nodes and edges that distort the propagated rewards. We remove these states with this strategy to ensure accurate graph construction and reward propagation. The ablation study of Tab. 3 supports the efficacy of this strategy.

10.3 Reversing Edges

Bi-directed edges preserves the fidelity of the graph structure while addressing exploration sparsity. RewardFlow first constructs a faithful directed graph from collected rollouts, recording exactly what the policy experienced: no transitions are removed or altered. We then add reverse edges solely for the propagation phase, effectively treating the graph as undirected during BFS. This simple step solves a key practical issue: early in training, many visited states lack outgoing paths to success because of limited policy exploration. Without reverse edges, these states receive zero reward, making useful actions unlearnable. Bi-directionality ensures rewards flow broadly across the sampled subgraph using shortest-path distances, dramatically improving credit assignment without requiring more rollouts or environment assumptions.

Our ablation study confirms that filtering invalid and no-op actions improves overall performance by reducing noise in credit assignment. To empirically validate this, we conducted experiments on ALFWorld, comparing RewardFlow with and without the filtering step. Retaining invalid/self-loop actions yields robust but degraded performance, as the noisy graph hampers accurate advantage estimation and policy updates. The results are as follows:

Our benchmarks (ALFWorld, Sokoban, WebShop) cover wide scenarios, and bi-directionality performs appropriately in each:

- **Reversible actions (most common):** Actions like "go to kitchen" in ALFWorld or navigation in WebShop have natural reverses. Adding reverse edges exactly reflects real environment dynamics, increasing reward density and guidance strength.

- **Irreversible but approximable actions:** Some actions lack direct reverses but can be undone via short unobserved paths. Adding reverses slightly overestimates reachability, yet relative reward ordering remains correct-states closer to success always receive higher rewards, preventing any inversion where worse actions appear better.

- **Fully irreversible actions (e.g., dead-ends in Sokoban):** Leading actions receive low or zero propagated rewards, correctly teaching the policy to avoid such paths. No misleading positive signals are introduced. For environments with many true dead-ends, users can simply disable bi-directionality; the method falls back to uni-directional propagation or alternative algorithms (e.g., PageRank).

Edge reversing is an optional heuristic whose use should be guided by empirical performance in the target environment. We explicitly treat it as a configurable component rather than a hard requirement. Our ablation studies on ALFWorld (Tab. 3) show that the performance is improved by introducing edge reversing. However, we respectfully agree that this strategy may not be generalizable to several environments, especially those that have many irreversible actions. Thus, by simply running a quick hyperparameter sweep over (with reverse edges, without reverse edges), the most appropriate variant can be selected that performs best on their specific task. The rest of the RewardFlow pipeline (state graph construction, BFS distance computation, and state-wise advantage estimation) remains unchanged.

10.4 State-wise Advantage

The vanilla computation of advantage $A(\boldsymbol{s}, a_t^{(i)}) = \frac{\tilde{r}_t^{(i)} - \mu(\boldsymbol{s})}{\sigma(\boldsymbol{s})}$ may cause that the policy can not learn the action from nodes that only have one out degree in the state graph, since $r_t^{(i)} - \mu(\boldsymbol{s}) = 0$. To mitigate this problem, we specially set the state-wise baseline to $\mu(s) = 0$ and the scale to $\sigma(s) = 1$. This directly gives the single observed action an advantage of $A(s, a) = \tilde{r}(s, a) - 1$. Consequently, actions that move closer to success ($\tilde{r} > 0$) receive positive advantage, while actions that move away ($\tilde{r} < 0$) receive negative advantage, even when the state appears only once. This simple fallback eliminates the risk of zero advantages across large numbers of unique states and ensures meaningful learning signals for nearly all actions in the batch.

10.5 Prompt Templates

In this section, we provide the prompt used in RL training to guide the model to do basic agentic reasoning. The prompt with no history denotes the initial prompt, while the prompt with history denotes the prompt in intermediate states.

ALFWORLD TEMPLATE NO HIS

```
"
You are an expert agent operating in the ALFRED Embodied Environment.
Your current observation is: {current_observation}
Your admissible actions of the current situation are:
    [{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation.
    This reasoning process MUST be enclosed within  <think>  </think>
     tags.
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within  <action>
    </action>  tags.
"
```

ALFWORLD TEMPLATE

```
"
You are an expert agent operating in the ALFRED Embodied Environment.
    Your task is to: {task_description}
Prior to this step, you have already taken {step_count} step(s).
    Below are the most recent {history_length} observations and the
    corresponding actions you took: {action_history}
You are now at step {current_step} and your current observation is:
    {current_observation}
Your admissible actions of the current situation are:
    [{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation.
    This reasoning process MUST be enclosed within  <think>  </think>
     tags.
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within  <action>
    </action>  tags.
"
```

SOKOBAN TEMPLATE NO HIS

```
"
You are an expert agent operating in the Sokoban environment.

# Symbols and Their Meaning
- Walls (`#`): These block movement. You can't move through or push
    anything into walls.
- Floor (`_`): Open spaces where you can walk and move boxes.
- Targets (`O`): The spots where boxes need to go.
- Boxes (`X`): These are what you need to push onto the targets.
- Player (`P`): That's you! You'll move around the grid to push boxes.
- Box on Target (``): A box successfully placed on a target.
- Player on Target (`S`): You standing on a target.

# Your Goal
Your goal is to push all the boxes (`X`) onto the target spots (`O`).
    Once all boxes are on the targets, you win!

# Rules
You can only push boxes. You can't pull them, so plan ahead to avoid
    getting stuck.
You can't walk through or push boxes into walls (`#`).
To avoid traps, do not push boxes into corners or against walls where
    they can't be moved again.

# Current Step
Your current observation is:
{current_observation}
Your admissible actions are ["up", "down", "left", "right"].

Now it's your turn to make a move (choose ONE action only for the
    current step).
You should first reason step-by-step about the current situation
    observe the positions of boxes and targets, plan a path to push a
    box toward a target, and avoid traps like corners or walls. This
    reasoning process MUST be enclosed within <think> </think> tags.
```

```
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within <action> </action>
    tags.
"
```

SOKOBAN VISUAL TEMPLATE

```
"
You are an expert agent operating in the Sokoban environment. Your
    goal is to push all the boxes onto the target spots. Once all
    boxes are on the targets, you win!

# Rules
You can only push boxes. You can't pull them, so plan ahead to avoid
    getting stuck.
You can't walk through or push boxes into walls.
To avoid traps, do not push boxes into corners or against walls where
    they can't be moved again.

# Visual Elements in the Image:
Character: A small, green alien-like figure with two antennae and
    black eyes. It represents you.
Box: A yellow crate marked with an orange "X" across its front. It is
    the box you need to push.
Target: A black tile outlined in red, with a small red diamond shape
    in the center. It marks the destination where a box should be
    pushed.

# Current Step
Your current observation is shown in the image: <image>
Your admissible actions are ["up", "down", "left", "right"].

Now it's your turn to make a move (choose ONE action only for the
    current step).
You should first reason step-by-step about the current situation
    observe the positions of boxes and targets, plan a path to push a
    box toward a target, and avoid traps like corners or walls. This
    reasoning process MUST be enclosed within <think> </think> tags.
Once you've finished your reasoning, you should choose an admissible
    action for current step and present it within <action> </action>
    tags.
"
```

11 Further Discussion

11.1 Does RewardFlow Rely on the High-quality Exploration?

RewardFlow does not perform exhaustive exploration and scales to massive state/action spaces. RewardFlow derives states, actions, and transitions directly from RL rollout sampling, and no exhaustive exploration is required. The induced graph is a compact subset of visited trajectories, and reward propagation operates solely on this subgraph. In large agentic environments, RewardFlow remains applicable as long as the LLM agent can sample at least one successful trajectory per batch. This avoids "exhaustive visitation" and ensures tractability, even in continuous or high-dimensional spaces, by clustering states via embeddings or hashing (as in our stochastic handling below) without assuming pure discreteness.

The accuracy of the state graph that is critical for reward modeling is not obviously affected by the poor exploration. RewardFlow ensures accurate state graph construction by building graphs

solely from actual environment interactions (states and actions), as induced from LLM-generated rollouts. This guarantees that the graph is a faithful topological representation of visited trajectories, without hallucinations or invalid edges. As long as at least one successful trajectory is sampled in a group, propagation methods like BFS reliably assign state-wise rewards reflecting task-centric metrics, such as the minimum actions needed to reach success from each state (shortest path in the graph). Poor exploration (e.g., low diversity) merely results in a sparser but still accurate graph. modeling a subset of the environment without introducing errors or misguiding propagation. This contrasts with trajectory-level methods, where sparse rewards can dilute credit assignment across entire paths. Our filtering of invalid/self-examination actions further enhances reliability by removing noise, ensuring propagation focuses on meaningful transitions. The ablation study in Tab. 5 supports this, where RewardFlow consistently outperforms GiGPO, even reducing rollouts number, indicating that RewardFlow is robust to poor exploration.

We further measure the entropy of the policy during different training steps and investigate how the entropy affects graph density. We measure policy entropy as $H(p) = -\sum_{i=1}^{|V|} p(x_i) \log p(x_i)$, where $|V|$ denotes the length of the policy's vocabulary and x_i denotes each token in the vocabulary. To study the interplay between entropy and graph structure, we evaluate three checkpoints of the Qwen-2.5-1.5B-Instruct model during RewardFlow training on ALFWorld: step 0 (initial supervised model), step 50, and step 100. For each checkpoint, we collect K = 8 rollouts per task using group sampling, compute the entropy, and record graph statistics along with validation performance.

Table 7: Training progress of Qwen-2.5-1.5B-Instruct on the target task. Entropy, invalid rates, graph size, and success rate across training steps.

Training Step	Entropy	Invalid State Rate (%)	Invalid Action Rate (%)	Avg. Node	Avg. Edge	Success Rate (%)
0	1.099	33.1	41.0	24.1	46.9	4.1
50	0.555	0.3	0	33.1	55.9	38.3
100	0.295	0.1	0	19.9	30.6	65.6

At step 0, the policy has very high entropy (1.099) and generates many invalid or no-op actions, which are subsequently pruned. As a result, even though the policy is highly exploratory, a large fraction of transitions do not contribute valid edges, yielding only moderately dense graphs and poor reward signal propagation. At step 50, entropy has decreased significantly (0.555), the model almost never produces invalid actions, and exploration remains sufficient to discover diverse valid paths. This produces the densest state graphs (33.1 nodes and 55.9 edges on average), maximizing the coverage of the refined graph and enabling the most effective BFS-based reward backpropagation, which explains the rapid performance improvement at this stage. At step 100, entropy is lowest (0.295), and the policy has converged toward near-deterministic optimal behavior. It now focuses almost exclusively on high-reward paths to success states, resulting in the sparsest graphs (19.9 nodes and 30.6 edges on average). Despite having the fewest nodes and edges, these compact graphs are highly efficient: almost every observed transition reduces the shortest-hop distance to a success state, making reward shaping extremely precise and leading to the highest validation success rate (65.6%). These results reveal a clear and insightful trend: excessively high entropy harms graph density due to invalid actions, moderate entropy maximizes graph coverage and reward densification during learning, and low entropy ultimately yields minimal yet highly informative graphs that support optimal performance.

Furthermore, the state-graph visualization case studies in ALFWorld (Appendix 12.4) demonstrate that, even with a single successful sampling rollout, the constructed state graph accurately captures the topological relationships among states, enabling precise reward assignment to intermediate states. With additional sampling rollouts, the graph modeling is further refined, yielding even more reliable rewards.

11.2 Comparing RewardFlow with LLM-based PRM Methods

Existing PRMs and preference learning approaches are largely confined to static, single-turn reasoning tasks (e.g., math problems (Lightman et al., 2023)), where paired preferences or step-level scores can be obtained relatively easily. In contrast, long-horizon agentic tasks involve dynamic state transitions and environment feedback, making human annotation of intermediate preferences prohibitively costly (20-40 steps per episode in ALFWorld). Recent agentic RL methods (e.g., ARPO (Dong

et al., 2025), SimpleTIR (Xue et al., 2025)) rely on trajectory- or group-level supervision and do not employ stepwise PRMs. GiGPO (Feng et al., 2025b), the closest related work, combines state- and trajectory-level advantages but ignores topological relationships across states. This gap underscores RewardFlow's novelty: it delivers dense, topology-aware supervision without requiring human annotations or auxiliary reward models.

11.3 The Potential to Use GNN for Reward Propagation

GNN-based reward propagation methods demonstrate challenges in on-policy RL training. In ALFWorld, Sokoban, WebShop, and DeepResearch, GNN-based propagation is less applicable due to practical constraints. GNNs require collecting and labeling additional graph data (e.g., node/action features and ground-truth rewards), followed by separate training phases, which introduce overhead in data quality dependence and generalization challenges on dynamically growing, large-scale graphs from online rollouts. Moreover, GNNs demand additional GPU resources and computation time for inference during each training step, reducing overall efficiency in on-policy RL. Heuristic methods like BFS/PageRank, by contrast, are lightweight, interpretable, and directly leverage the induced graph topology without external training, ensuring reliable propagation of verifiable terminal rewards. This aligns with RewardFlow's goal of simple, scalable credit assignment in sparse-reward, multi-turn agentic scenarios, where exploration is ongoing, and graphs evolve per batch.

GNN-based propagation methods have the potential to be applied in semantically rich environments. GNNs offer advantages in incorporating semantic information from node (state) and edge (action) embeddings, potentially yielding more nuanced reward propagation strategies that account for contextual similarities beyond pure topology. This could be particularly beneficial in fixed-task environments with abundant offline data, such as: (1) video game domains like StarCraft (Vinyals et al., 2019), where state graphs are predefined and GNNs can learn from massive replay buffers to predict value functions; (2) robotics tasks in simulated worlds (e.g., MuJoCo (Todorov et al., 2012)), where physical semantics (e.g., joint angles, object interactions) enable GNNs to generalize across similar trajectories; (3) molecular design (De Cao & Kipf, 2018) or drug discovery graphs (You et al., 2018), where node features (e.g., atom types) allow learned propagation to optimize sparse rewards like binding affinity; or (4) offline RL in recommendation systems (Chen et al., 2024), with static user-item graphs enabling pre-trained GNNs for reliable, semantics-aware credit assignment. In these cases, a trained GNN could provide task-specific reliability once deployed, and this is a promising extension for future work in RewardFlow variants tailored to such domains.

11.4 Does Reward Contradictions Occurs in RewardFlow?

RewardFlow's graph construction avoids such conflicts in deterministic environments due to the inherent consistency of transitions. In ALFWorld, WebShop, and Sokoban, the transition function $P(s'|s,a)$ is deterministic, and states are distinct (textual configurations in ALFWorld and WebShop; grid layouts in Sokoban). Thus, any two rollouts that visit the same state-action pair necessarily reach the identical next state. The induced exploration graph is therefore free of conflicting edges. For reward propagation, our BFS/PageRank algorithms conservatively take the maximum propagated reward per node (equivalent to the shortest-path distance to the nearest successful terminal in BFS), yielding a unique, consistent value for every reachable state regardless of how many or which trajectories discovered it.

The semantic similarity clustering strategy avoids the contradiction of reward assignment. In the highly stochastic DeepResearch environment, semantically equivalent queries can surface different retrieved passages, leading to superficially distinct but functionally identical states. To prevent conflicting reward assignments, we cluster states whose embeddings have high cosine similarity, merge them into a single super-node, and assign the max-pooled propagated reward (or averaged, as ablation shows negligible difference). This simple yet effective deduplication ensures that near-identical research states receive identical credit, eliminating contradictions while preserving the underlying topology. Empirical studies in DeepResearch (Tab. 2) demonstrate that this clustering contributes critically to about 2.09% average gain over GRPO baselines under high stochasticity.

11.5 Comparing RewardFlow with Off-Line Reward Densify Methods

Conceptually, RewardFlow densifies the reward derived from on-policy rollout sampling, which has relevance with other off-policy reward densify methods like DiffStitch Li et al. (2024) and QPHIL Canesse et al. (2024). Here, we discuss the key differences between RewardFlow and these methods. RewardFlow densifies sparse terminal rewards by constructing a state graph from on-policy LLM rollouts and propagating success signals backward via graph propagation algorithms such as BFS or Personalization PageRank, yielding dense state-wise rewards without generating synthetic data. In contrast, DiffStitch is an offline data-augmentation method that explicitly stitches low-reward and high-reward trajectories by sampling diffusion-generated sub-trajectories between any two states, while QPHIL achieves implicit stitching at test time through a VQ-VAE-quantized high-level transformer planner that autoregressively predicts discrete landmark sequences. Thus, RewardFlow is a pure reward-shaping approach on observed rollouts, whereas DiffStitch and QPHIL rely on either explicit trajectory synthesis or hierarchical discrete planning.

Adapting DiffStitch or QPHIL to the on-policy agentic RL setting of RewardFlow is challenging: DiffStitch would require continuously retraining a diffusion model on shifting on-policy data, introducing severe instability and being prohibitive for LLM fine-tuning loops; QPHIL's offline-pretrained VQ-VAE and transformer planner would quickly become misaligned as the LLM explores new regions of the text/visual state space, breaking landmark consistency and high-level planning reliability. These distributional-shift issues make both methods hard to adapt to on-policy LLM reinforcement learning, which is why RewardFlow's non-parametric graph propagation on current rollouts remains simpler and more stable.

12 Further Experiments

12.1 Training Efficiency

We show average numbers of nodes and edges in the induced graph of rollout sampling (Tab. 8), where we collect the data from ALFWorld and WebShop via the sampling from Qwen2.5-1.5B-Instruct with 8 rollouts, where ALFWorld takes 25 interactive steps, and WebShop takes 15 interactive steps. For Sokoban, we use Qwen2.5-VL-3B-Instruct with 8 rollouts and 15 interactive steps. The results show that for all three agentic environments, the nodes and edges are no more than 42 and 77, respectively, indicating that the induced graph through rollout does not involve numerous nodes and edges, where the graph construction and reward propagation process would not take much time. We also show the average time consumption of each training stage per step to demonstrate that RewardFlow is a lightweight and efficient reward modeling method.

Table 8: Average and maximum size of induced state graphs from 8 on-policy rollouts (Qwen2.5-1.5B/VL-3B-Instruct). Even in the worst case, graphs remain extremely compact.

Environment	Avg. Nodes	Avg. Edges	Max Nodes	Max Edges
ALFWorld	28.8	55.9	42	77
WebShop	36.8	48.2	54	65
Sokoban	14.0	21.0	27	39

12.2 Training Until Convergence

Table 9: Extended training to 200 steps on Sokoban (Qwen2.5-VL-3B-Instruct). All methods fully converge well before 200 steps, and RewardFlow's advantage further widens after convergence.

Method	Success @100 steps (%)	Success @200 steps (%)	Δ (200-100)
RLOO	22.7	41.4	+18.7
GRPO	26.6	39.1	+12.5
GiGPO	21.9	33.6	+11.7
RewardFlow (ours)	**49.2**	**70.3**	**+22.1**

We continued training the exact same checkpoints of Sokoban from the main experiments for an additional 100 steps (total 200 steps) under identical hyperparameters. All methods plateau between

150 and 200 steps, confirming full convergence. As shown in Tab 9, RewardFlow's lead over the strongest baseline grows from +22.6% at 100 steps to +28.9% at 200 steps. Notably, while baselines improve by at most +18.7% with the extra budget, RewardFlow gains +22.1%, showing that its denser credit assignment continues to extract meaningful signal even in later training phases. These results confirm that our reported advantages are not artifacts of early stopping and remain robust (and even strengthen) under fully converged conditions.

12.3 Propagation Strategy

In this section, we try another strategy in BFS: the average distance $d(s) = \frac{1}{|S_{\text{succ}}|}\sum_{s^\star \in S_{\text{succ}}} \text{dist}_{\text{hop}}(s \rightsquigarrow s^\star)$ to compare with the original propagation strategy $d(s) = \min_{s^\star \in S_{\text{succ}}} \text{dist}_{\text{hop}}(s \rightsquigarrow s^\star)$. As shown in Tab. 10, using minimum hop distance for reward propagation demonstrates consistently higher performance than the mean distance. Especially in Sokoban, using minimum distance shows a 15.6% improvement in success rate compared with mean distance. Intuitively, farther routes would dilute the signal, assigning a lower value $R(s)$ to a state with one short path but many long ones, even if the short path is viable. This might yield $\tilde{r}(s_t, a_t) < 0$ for actions leading to such states, discouraging promising transitions and destabilizing the state-wise advantages.

However, while simply integrate mean distance performs worse, alternatives like weighted mean distances could mitigate these issues but would require additional design and empirical support. For example, assigning higher weights to shorter paths (e.g., exponential decay based on hop length) might better aggregate topological information, but this introduces hyperparameters and risks overfitting to rollout noise, complicating the multi-source reverse BFS. Our choice of min distance keeps the approach simple and topology-aware, ensuring reliable propagation, which improves reachability without such overhead. Our results in Table 1 indicate that the min-based method already provides robust, task-centric signals across model sizes.

In addition, incorporating uncertainty into reward propagation is an insightful suggestion, as it could refine the signal by accounting for policy variability, though it introduces subjective elements distinct from objective graph topology. Uncertainty, such as entropy over $\pi_\theta(a|s)$, reflects the LLM policy's confidence rather than inherent environment structure, potentially enhancing exploration in high-uncertainty states. However, integrating it (e.g., via entropy-regularized distances) would require careful adaptation to avoid biasing the objective distance metric.

Table 10: Ablation on propagation strategies.

Model	Environment	Propagation	Success Rate (%)
Qwen-2.5-1.5B-Instruct	ALFWorld	Mean Distance	62.5
		Min Distance	**65.6**
Qwen-2.5-VL-3B-Instruct	Sokoban	Mean Distance	33.6
		Min Distance	**49.2**

12.4 State Graph Cases

This section showcases a constructed state graph given different numbers of rollout trajectories. The state graph of one sampling rollout can be found in Figs. 3, 4. The state graph of two sampling rollouts can be found in Figs. 5, 6. The state graph of three sampling rollouts can be found in Figs. 7, 8.

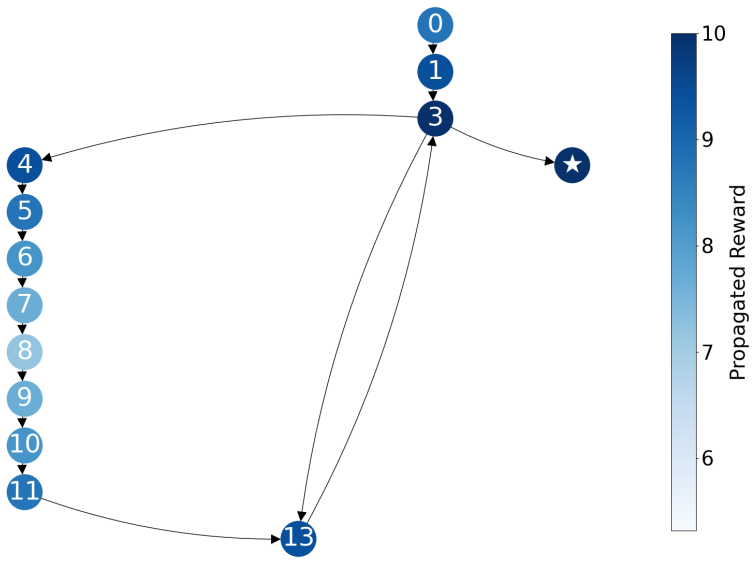

Figure 3: The constructed state graph with 1 sampling rollout. The deeper color indicates a higher assigned reward. The initial state is in Node 0, and the success terminal state is in Node ★. The corresponding text representation of nodes and edges can be found in Fig. 4.

Node List

Node 0: -= Welcome to TextWorld, ALFRED! =-
You are in the middle of a room. Looking quickly around you, you see a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 2, a countertop 1, a diningtable 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: clean some mug and put it in coffeemachine
Node 1: You arrive at countertop 1. On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 2: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 3: You pick up the mug 1 from the countertop 1
Node 4: You move the mug 1 to the coffeemachine 1
Node 5: Nothing happens
Node 6: You pick up the mug 1 from the coffeemachine 1
Node 7: You arrive at countertop 2. On the countertop 2, you see a egg 2, a plate 2, a plate 1, a spatula 2, a spatula 1, and a winebottle 1
Node 8: Nothing happens
Node 9: You arrive at sinkbasin 1. On the sinkbasin 1, you see a potato 2, a spoon 1, and a tomato 1
Node 10: You clean the mug 1 using the sinkbasin 1
Node 11: You arrive at coffeemachine 1. On the coffeemachine 1, you see nothing
Node 12: On the coffeemachine 1, you see nothing
Node 13: You move the mug 1 to the countertop 1
Node ★: You Won!

Edge List

Node 0 -> Node 1: go to countertop 1
Node 1 -> Node 3: take mug 1 from countertop 1
Node 3 -> Node 4: move mug 1 to coffeemachine 1
Node 3 -> Node 13: move mug 1 to countertop 1
Node 3 -> Node ★: move mug 1 to coffeemachine 1
Node 4 -> Node 5: clean mug 1 from countertop 1
Node 5 -> Node 6: take mug 1 from coffeemachine 1
Node 6 -> Node 7: go to countertop 2
Node 7 -> Node 8: go to countertop 2
Node 8 -> Node 9: go to sinkbasin 1
Node 9 -> Node 10: clean mug 1 with sinkbasin 1
Node 10 -> Node 11: go to coffeemachine 1
Node 11 -> Node 13: move mug 1 to countertop 1
Node 13 -> Node 3: take mug 1 from countertop 1

Figure 4: The text description of nodes and edges of state graph in Fig. 3

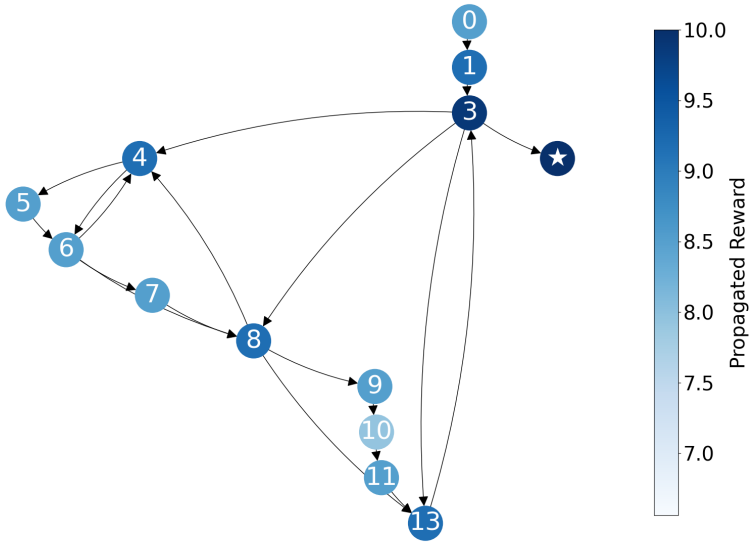

Figure 5: The constructed state graph with 2 sampling rollouts. The deeper color indicates a higher assigned reward. The initial state is in Node 0, and the success terminal state is in Node ★. The corresponding text representation of nodes and edges can be found in Fig. 6.

Node List

Node 0: -= Welcome to TextWorld, ALFRED! =-
You are in the middle of a room. Looking quickly around you, you see a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 2, a countertop 1, a diningtable 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: clean some mug and put it in coffeemachine
Node 1: You arrive at countertop 1. On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 2: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 3: You pick up the mug 1 from the countertop 1
Node 4: You move the mug 1 to the coffeemachine 1
Node 5: Nothing happens
Node 6: You pick up the mug 1 from the coffeemachine 1
Node 7: You arrive at countertop 2. On the countertop 2, you see a egg 2, a plate 2, a plate 1, a spatula 2, a spatula 1, and a winebottle 1
Node 8: Nothing happens
Node 9: You arrive at sinkbasin 1. On the sinkbasin 1, you see a potato 2, a spoon 1, and a tomato 1
Node 10: You clean the mug 1 using the sinkbasin 1
Node 11: You arrive at coffeemachine 1. On the coffeemachine 1, you see nothing
Node 12: On the coffeemachine 1, you see nothing
Node 13: You move the mug 1 to the countertop 1
Node ★: You Won!
Node 15: You are not carrying anything
Node 16: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a pan 1, and a spatula 3
Node 17: You are carrying: a mug 1
Node 18: This is a normal mug 1. In it, you see nothing

Edge List

Node 0 -> Node 1: go to countertop 1
Node 0 -> Node 1: go to countertop 1
Node 1 -> Node 3: take mug 1 from countertop 1
Node 1 -> Node 3: take mug 1 from countertop 1
Node 3 -> Node 4: move mug 1 to coffeemachine 1
Node 3 -> Node 8: move mug 1 to sinkbasin 1
Node 3 -> Node 4: move mug 1 to coffeemachine 1
Node 3 -> Node 13: move mug 1 to countertop 1
Node 3 -> Node ★: move mug 1 to coffeemachine 1
Node 4 -> Node 5: clean mug 1
Node 4 -> Node 6: take mug 1 from coffeemachine 1
Node 4 -> Node 5: move mug 1 to sinkbasin 1
Node 4 -> Node 5: clean mug 1 from countertop 1
Node 5 -> Node 6: take mug 1 from coffeemachine 1
Node 5 -> Node 6: take mug 1 from coffeemachine 1
Node 6 -> Node 4: move mug 1 to coffeemachine 1
Node 6 -> Node 8: go to coffeemachine 1
Node 6 -> Node 8: lean on coffeemachine 1
Node 6 -> Node 7: go to countertop 2
Node 7 -> Node 8: go to countertop 2
Node 8 -> Node 4: move mug 1 to coffeemachine 1
Node 8 -> Node 13: move mug 1 to countertop 1
Node 8 -> Node 9: go to sinkbasin 1
Node 9 -> Node 10: clean mug 1 with sinkbasin 1
Node 10 -> Node 11: go to coffeemachine 1
Node 11 -> Node 13: move mug 1 to countertop 1
Node 13 -> Node 3: take mug 1 from countertop 1
Node 13 -> Node 3: take mug 1 from countertop 1

Figure 6: The text description of nodes and edges of state graph in Fig. 5

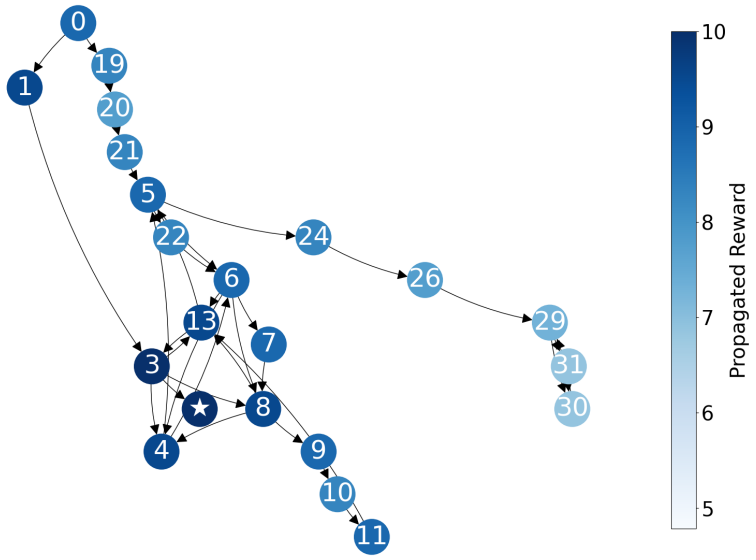

Figure 7: The constructed state graph with 3 sampling rollouts. The deeper color indicates a higher assigned reward. The initial state is in Node 0, and the success terminal state is in Node ★. The corresponding text representation of nodes and edges can be found in Fig. 8.

Node List

Node 0: -= Welcome to TextWorld, ALFRED! =-
You are in the middle of a room. Looking quickly around you, you see a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 2, a countertop 1, a diningtable 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: clean some mug and put it in coffeemachine
Node 1: You arrive at countertop 1. On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 2: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, a pan 1, and a spatula 3
Node 3: You pick up the mug 1 from the countertop 1
Node 4: You move the mug 1 to the coffeemachine 1
Node 5: Nothing happens
Node 6: You pick up the mug 1 from the coffeemachine 1
Node 7: You arrive at countertop 2. On the countertop 2, you see a egg 2, a plate 2, a plate 1, a spatula 2, a spatula 1, and a winebottle 1
Node 8: Nothing happens
Node 9: You arrive at sinkbasin 1. On the sinkbasin 1, you see a potato 2, a spoon 1, and a tomato 1
Node 10: You clean the mug 1 using the sinkbasin 1
Node 11: You arrive at coffeemachine 1. On the coffeemachine 1, you see nothing
Node 12: On the coffeemachine 1, you see nothing
Node 13: You move the mug 1 to the countertop 1
Node ★: You Won!
Node 15: You are not carrying anything
Node 16: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a pan 1, and a spatula 3
Node 17: You are carrying: a mug 1
Node 18: This is a normal mug 1. In it, you see nothing
Node 19: You arrive at cabinet 2. The cabinet 2 is closed
Node 20: You open the cabinet 2. The cabinet 2 is open. In it, you see a winebottle 2
Node 21: You close the cabinet 2
Node 22: You arrive at coffeemachine 1. On the coffeemachine 1, you see a mug 1
Node 23: On the coffeemachine 1, you see nothing
Node 24: You pick up the spatula 3 from the countertop 1
Node 25: On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, and a pan 1
Node 26: You arrive at sinkbasin 1. On the sinkbasin 1, you see a potato 2, a spoon 1, and a tomato 1
Node 27: You are carrying: a spatula 3
Node 28: There's nothing special about spatula 3
Node 29: You clean the spatula 3 using the sinkbasin 1
Node 30: Nothing happens
Node 31: You arrive at countertop 1. On the countertop 1, you see a butterknife 3, a butterknife 1, a lettuce 1, a mug 1, and a pan 1

Edge List

Node 0 -> Node 19: go to cabinet 2
Node 0 -> Node 1: go to countertop 1
Node 0 -> Node 1: go to countertop 1
Node 1 -> Node 3: take mug 1 from countertop 1
Node 1 -> Node 3: take mug 1 from countertop 1
Node 3 -> Node 4: move mug 1 to coffeemachine 1
Node 3 -> Node 8: move mug 1 to sinkbasin 1
Node 3 -> Node 4: move mug 1 to coffeemachine 1
Node 3 -> Node 13: move mug 1 to countertop 1
Node 3 -> Node ★: move mug 1 to coffeemachine 1
Node 4 -> Node 5: clean mug 1
Node 4 -> Node 6: take mug 1 from coffeemachine 1
Node 4 -> Node 5: move mug 1 to sinkbasin 1
Node 4 -> Node 5: clean mug 1 from countertop 1
Node 5 -> Node 22: go to coffeemachine 1
Node 5 -> Node 24: take spatula 3 from countertop 1
Node 5 -> Node 6: take mug 1 from coffeemachine 1
Node 5 -> Node 6: take mug 1 from coffeemachine 1
Node 6 -> Node 13: move mug 1 to countertop 1
Node 6 -> Node 4: move mug 1 to coffeemachine 1
Node 6 -> Node 8: go to coffeemachine 1
Node 6 -> Node 8: lean on coffeemachine 1
Node 6 -> Node 7: go to countertop 2
Node 7 -> Node 8: go to countertop 2
Node 8 -> Node 4: move mug 1 to coffeemachine 1
Node 8 -> Node 13: move mug 1 to countertop 1
Node 8 -> Node 9: go to sinkbasin 1
Node 9 -> Node 10: clean mug 1 with sinkbasin 1
Node 10 -> Node 11: go to coffeemachine 1
Node 11 -> Node 13: move mug 1 to countertop 1
Node 13 -> Node 5: move mug 1 to coffeemachine 1
Node 13 -> Node 3: take mug 1 from countertop 1
Node 13 -> Node 3: take mug 1 from countertop 1
Node 19 -> Node 20: open cabinet 2
Node 20 -> Node 21: close cabinet 2
Node 21 -> Node 5: go to cabinet 2
Node 22 -> Node 6: take mug 1 from coffeemachine 1
Node 24 -> Node 26: go to sinkbasin 1
Node 26 -> Node 29: clean spatula 3 with sinkbasin 1
Node 29 -> Node 30: clean mug with sinkbasin 2
Node 29 -> Node 31: go to countertop 1
Node 30 -> Node 29: clean spatula 3 with sinkbasin 1
Node 31 -> Node 30: go to countertop 1

Figure 8: The text description of nodes and edges of state graph in Fig. 7

12.5 TRAINING DYNAMIC

In this section, we present the performance curves for the training and validation sets to analyze the training dynamics of RewardFlow. The Sokoban task illustration is shown in Fig. 9. After 100 training steps, RewardFlow demonstrates superior performance on both the 3B and 7B models compared to other baselines, highlighting the exceptional effectiveness of our framework. While other training algorithms exhibit flat performance on both training and validation sets, RewardFlow achieves significant improvement between 70 and 80 training steps on the Qwen-VL-2.5-3B model. We attribute this to the initially weaker performance of the Qwen-VL-2.5-3B model, where early training steps yield fewer successful trajectories due to sparse rewarded states. As training progresses, the model's capabilities improve, generating more successful trajectories and enabling the policy to learn more effectively in later stages, resulting in substantial gains after 70 steps. Conversely, the Qwen-VL-2.5-7B model, which benefits from greater pretraining capacity due to its larger size, shows distinct performance improvements on both training and validation sets within the first 30 steps. The performance of ALFWorld can be found in Fig. 10, while other training algorithms are similar in curve trend; the trend of RewardFlow is distinct and higher than other training methods. Compared with the visual benchmark Sokoban, the text-based ALFWorld is easy to be understand by LLMs, thus the policy in early training stages can collect sufficient successful trajectories for RewardFlow to model the state-wise reward and conduct effective optimization.

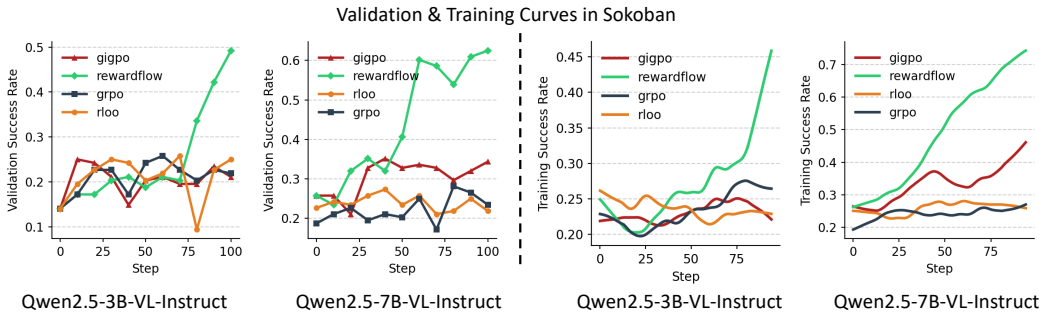

Figure 9: The Validation and training curves in Sokoban on both Qwen-2.5-VL-3B-Instruct and Qwen-2.5-VL-7B-Instruct, we report the performance on the validation set every 10 training steps, the curves of the training set are smoothed using LOWESS (Smolik et al., 2016)

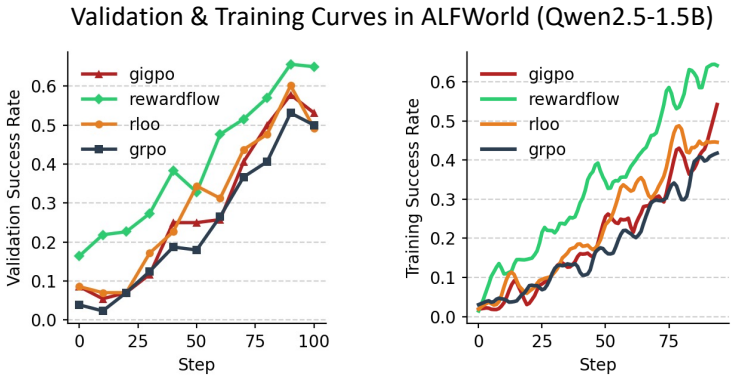

Figure 10: The Validation and training curves in ALFWorld on both Qwen-2.5-1.5B-Instruct, we report the performance on the validation set every 10 training steps, the curves of the training set are smoothed using (Smolik et al., 2016)