Technical Summary:
RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning

Kai-Yu Lu

1 Research Problem and Motivation

Large language models (LLMs) are increasingly deployed as autonomous agents in interactive environments that require multi-step decision making, memory across turns, and adaptation to stochastic feedback. Typical examples include web assistants that interact with complex user interfaces, symbolic planning in puzzle environments, and tutoring systems that must conduct multi-turn dialogues.

Existing reinforcement learning (RL) practices for LLMs primarily focus on static, single-turn tasks such as mathematical problem solving or code generation. In those settings, the model receives a prompt once and outputs a single response that is scored by a reward model or rule-based evaluator. The optimization objective is defined over individual prompt–response pairs rather than over entire interaction trajectories. This single-step view fails to capture the dynamics of long-horizon agent behavior.

When RL is applied to interactive agents, several difficulties arise. First, the agent must reason and plan over long horizons, where a decision taken at an early time step may only reveal its quality many steps later. Second, many real or simulated environments exhibit stochastic transitions and noisy rewards, so the same action taken in the same state can lead to different outcomes. Third, empirical evidence shows that naively adapting single-turn RL algorithms to multi-turn agents often leads to unstable training, gradient explosions, and mode collapse in behavior, combined with poor generalization across environment variations.

The central problem addressed in this work is therefore twofold. On the one hand, there is a need for a general RL framework that can handle reasoning-augmented multi-turn trajectories of LLM agents in stochastic environments. On the other hand, there is a need to understand which design choices in multi-turn RL pipelines lead to stable self-improvement of agents, and which choices lead to collapse or shallow shortcut strategies.

2 Related Work

2.1 Reinforcement learning for language models

Reinforcement learning has been used extensively to fine-tune language models using human or rule-based feedback. Proximal Policy Optimization (PPO) and actor–critic methods have become standard tools for optimizing single-turn prompt–response behavior. More recent work has introduced variants such as Group Relative Policy Optimization (GRPO) and other critic-free approaches that rely on relative ranking of responses within a batch to form policy gradients.

These approaches have demonstrated improvements in factuality, safety, and reasoning on static benchmarks. However, the majority of these methods treat each response independently, ignoring the temporal structure and state transitions present in interactive settings. As a consequence, they do not directly address the stability and generalization issues that arise when LLMs operate as agents over long trajectories.

2.2 Agent frameworks and multi-turn interaction

Parallel to RL for language models, a growing body of work has introduced agent frameworks that structure decision making into perception, planning, and action components. Early systems focus on tool calling, task decomposition, and modular pipelines, while recent frameworks explore multi-agent collaboration, embodied control, and web interaction.

Benchmarks such as Sokoban, Frozen Lake, and WebShop provide controlled environments for evaluating planning and interaction under different dynamics, including deterministic transitions, probabilistic transitions, and natural language grounding. These benchmarks are typically used with search-based planners or heuristic policies, rather than with RL-trained LLM policies.

The contribution of the present work lies at the intersection of these two lines of research. It introduces a unified trajectory-level RL framework for LLM agents and a modular system implementation that allows systematic analysis of training dynamics across symbolic and realistic environments.

3 Dataset Construction

In this study, the term "dataset" corresponds to collections of tasks and initial states in four environments used for training and evaluation. Three environments are symbolic and fully controllable, while one environment is a natural language web shopping environment.

3.1 Environments and task instances

3.1.1 Bandit

Bandit is a single-turn, stochastic environment that simulates a multi-armed bandit problem. Each task instance describes several symbolic arms (for example, professions such as "Teacher" or "Engineer") that differ in their underlying reward distributions. The agent receives a textual description and must select one arm. The reward is sampled from a stochastic distribution associated with the chosen arm, which introduces outcome variability even when the same action is repeated in the same context.

A variant called BanditRev inverts the reward associations between symbolic arms, creating counterintuitive mappings that require deeper reasoning rather than surface pattern matching.

3.1.2 Sokoban

Sokoban is a classical deterministic puzzle environment in which an agent pushes boxes on a grid to target locations. Each task instance defines a grid layout with walls, movable boxes, goals, and an initial agent position. Movements are irreversible in the sense that pushing a box into a corner may make the puzzle unsolvable. The environment is multi-turn and non-stochastic, so each action deterministically leads to a unique next state.

Additional evaluation sets include SokobanNewVocab, where action and object descriptions use alternative vocabulary, and LargeSokoban, where the grid is larger and puzzles are more complex.

3.1.3 Frozen Lake

Frozen Lake is a multi-turn, stochastic grid world. The agent must navigate from a start cell to a goal cell on a frozen lake with holes. At each step, the agent chooses a direction of movement, but due to slippery dynamics the actual transition is probabilistic and may deviate from the intended direction. The environment combines long-horizon planning with stochastic transitions.

3.1.4 WebShop

WebShop is a multi-turn, open-domain web shopping environment. Each task instance provides a natural language shopping goal, such as purchasing an item with specified attributes and constraints. The agent interacts with a simulated e-commerce website using text-based commands, including search queries, filters, navigation, and purchase actions. The environment requires grounding of natural language descriptions into actions on a semi-structured web interface.

3.2 Task collections and evaluation splits

For each environment, the study constructs fixed evaluation sets of 256 prompts or initial states. During evaluation, episodes are truncated after at most five turns. In training, each batch samples a fixed number of prompts and multiple rollouts per prompt, providing a structured collection of trajectories that serve as training data for policy optimization.

3.3 Preprocessing and representation

Environments are accessed through text interfaces. For symbolic environments, states are represented using textual descriptions of grid layouts, positions, and actions. For WebShop, the interface exposes page content, item descriptions, and available actions through text. Actions are represented as constrained text commands that the environment can execute, and success conditions are encoded as rule-based checks over environment states, such as reaching a goal cell, solving the puzzle, or purchasing an appropriate item.

4 Query Protocol and Task Definitions

4.1 Interaction protocol and horizon

All tasks are framed as Markov Decision Processes, where an LLM agent interacts with an environment over a finite horizon. An episode begins from an initial state and proceeds for at most a fixed number of turns, denoted by K. At each turn, the agent receives the current state description and interaction history and outputs a structured text response that contains both reasoning and executable actions.

In training, episodes are rolled out by sampling multiple trajectories from each initial state. In evaluation, trajectories are generated with fixed decoding temperature and deterministic truncation after the turn limit.

4.2 Success criteria and rewards

Each environment defines a task-specific notion of success and reward:

- In Bandit and BanditRev, success corresponds to choosing an arm with favorable expected return, and the reward is a scalar drawn from the reward distribution of the selected arm.
- In Sokoban and its variants, success corresponds to placing all boxes on target cells within the horizon. The environment returns intermediate rewards and a final success indicator.
- In Frozen Lake, success corresponds to reaching the goal cell without falling into a hole. Rewards reflect progress and terminal outcomes under stochastic transitions.
- In WebShop, success corresponds to purchasing an item that satisfies the shopping goal specification. Rewards account for successful completion within the interaction budget.

The study evaluates agents primarily using success rate, defined as the proportion of evaluation prompts for which the agent successfully completes the task.

4.3 Rollout reuse and Online-k protocol

The training pipeline adopts an Online-k rollout protocol. For a given policy snapshot, a batch of trajectories is collected and then reused for k consecutive parameter update steps. A smaller value of k corresponds to a higher rollout frequency and fresher data, while a larger value of k reduces rollout cost at the cost of stale data. Online-1 denotes the fully online setting where each update uses newly collected trajectories.

The effect of rollout reuse on convergence speed and generalization performance is examined by varying k and observing success rates on related tasks such as SokobanNewVocab and Frozen Lake.

5 Modeling Approach

5.1 MDP formulation and single-step baseline objective

The starting point is the standard single-step RL objective on prompt–response pairs. Let D denote a dataset of prompts, s a prompt sampled from D, and a a response sampled from the policy $\pi_\theta(\cdot \mid s)$ parameterized by θ. The single-step objective is

$$J_{\text{step}}(\theta) = \mathbb{E}_{s\sim D,\, a\sim\pi_\theta(\cdot|s)}\big[R(s,a)\big]. \tag{1}$$

In Equation (1), $J_{\text{step}}(\theta)$ denotes the expected reward for individual prompt–response pairs. The function $R(s,a)$ is a scalar reward that measures the quality of response a to prompt s, for instance reflecting correctness or preference alignment. The expectation is taken over prompts drawn from the empirical data distribution and responses sampled from the current policy. This objective treats each response independently and does not model state transitions.

To handle multi-turn interactions, the problem is reformulated as a Markov Decision Process (MDP). An MDP is defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P\}$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and P denotes the transition dynamics and reward generation process. A state $s_t \in \mathcal{S}$ encodes the current observation and history at time step t. An action $a_t \in \mathcal{A}$ represents an environment-executable command, which is produced as part of the LLM output. The transition kernel $P(\cdot \mid s_t, a_t)$ specifies the distribution over the next state s_{t+1} and reward r_t.

At each time step t, the policy $\pi_\theta(a_t \mid s_t, \tau_{<t})$ maps the current state and past trajectory $\tau_{<t}$ to a distribution over actions. The environment then samples the reward and next state from the transition dynamics.

5.2 Trajectory-level objective in StarPO

To incorporate full trajectories into the optimization, the study introduces StarPO (State–Thinking–Actions–Reward Policy Optimization). In StarPO, a trajectory τ is a sequence of states, reasoning-augmented actions, and rewards collected over a horizon of length at most K. The objective is to maximize the expected cumulative reward over entire trajectories:

$$J_{\text{StarPO}}(\theta) = \mathbb{E}_{\mathcal{M},\, \tau\sim\pi_\theta}\big[R(\tau)\big]. \tag{2}$$

In Equation (2), $J_{\text{StarPO}}(\theta)$ denotes the expected trajectory-level reward under policy π_θ in MDP $\mathcal{M}$. The outer expectation is taken over the distribution of environments or tasks and over trajectories τ sampled from the current policy. The scalar value $R(\tau)$ is the cumulative reward along the trajectory, for example a sum of per-step rewards or a success indicator at the end of the episode. This objective encourages behaviors that yield high long-horizon performance rather than optimizing isolated steps.

5.3 Reasoning-augmented action format

At each time step t, the LLM produces a structured text output that separates intermediate reasoning from the final executable action. The output has the following template:

$$a_t^T = \texttt{<think>} \ldots \texttt{</think>}\,\texttt{<answer>}\, a_t \,\texttt{</answer>}. \tag{3}$$

In Equation (3), a_t^T is the full textual output token sequence at time step t, including the content enclosed in `<think>` and `</think>` tags that describes the agent's reasoning, and the content enclosed in `<answer>` and `</answer>` tags that encodes the environment-executable action. The symbol a_t denotes the action sequence that the environment will execute. This design allows the policy to maintain explicit reasoning traces while also producing concrete actions.

The environment receives a_t, applies the corresponding transition, and returns the next state s_{t+1} and reward r_t. Repeating this process from an initial state s_0 yields a trajectory $\tau = \{s_0, a_0^T, r_0, s_1, \ldots, a_{K-1}^T, r_{K-1}, s_K\}$.

5.4 Optimization strategies within StarPO

StarPO supports different policy optimization algorithms at the token level by decomposing trajectory probabilities into products of token probabilities. For each trajectory τ_i containing $|\tau_i|$ tokens, the framework applies one of the following objectives.

5.4.1 PPO with a learned critic

Proximal Policy Optimization (PPO) introduces a surrogate objective with importance sampling ratios and clipping to constrain policy updates. Let $\tau_{i,(t)}$ denote the t-th token in trajectory τ_i and $\tau_{i,<t}$ its prefix. The policy under old parameters is denoted by π_{old}. A value function critic estimates token-level advantages $A_{i,t}$. The PPO objective in StarPO is

$$J_{\text{PPO}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\tau_i|}\sum_{t=1}^{|\tau_i|}\min\left[\frac{\pi_\theta(\tau_{i,(t)} \mid \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} \mid \tau_{i,<t})}A_{i,t},\ \text{clip}\left(\frac{\pi_\theta(\tau_{i,(t)} \mid \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} \mid \tau_{i,<t})}, 1-\epsilon, 1+\epsilon\right)A_{i,t}\right]. \tag{4}$$

In Equation (4), G is the number of trajectories in the batch. The term $\pi_\theta(\tau_{i,(t)} \mid \tau_{i,<t})$ is the probability assigned by the current policy to token $\tau_{i,(t)}$ given its prefix, and $\pi_{\text{old}}(\tau_{i,(t)} \mid \tau_{i,<t})$ is the same probability under the previous policy before the current update. The ratio between these probabilities forms an importance sampling weight that measures how much the new policy deviates from the old one at each token. The advantage $A_{i,t}$ captures how much better or worse this token choice is compared with the critic's baseline estimate. The clipping function with threshold ϵ truncates large deviations in the importance ratio, which prevents excessively large policy updates and stabilizes training. The overall objective averages the clipped and unclipped terms to encourage updates that improve expected advantage while respecting the proximity constraint.

5.4.2 GRPO with trajectory-level normalization

Group Relative Policy Optimization (GRPO) is a critic-free method that normalizes rewards within a batch. Each trajectory τ_i is assigned a scalar reward $R(\tau_i)$. The method computes a normalized advantage shared across all tokens in that trajectory:

$$\hat{A}_{i,t} = \frac{R(\tau_i) - \text{mean}\left(\{R(\tau_1), \ldots, R(\tau_G)\}\right)}{\text{std}\left(\{R(\tau_1), \ldots, R(\tau_G)\}\right)}. \tag{5}$$

In Equation (5), the numerator subtracts the mean reward of the batch from the reward of trajectory τ_i, and the denominator divides by the sample standard deviation of rewards in the batch. This standardization yields a zero-mean, unit-variance advantage that assigns positive values to trajectories that outperform the batch average and negative values to those that underperform.

The GRPO objective becomes

$$J_{\text{GRPO}}(\theta) = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \min \left[\frac{\pi_\theta(\tau_{i,(t)} \mid \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} \mid \tau_{i,<t})} \hat{A}_{i,t},\ \text{clip}\left(\frac{\pi_\theta(\tau_{i,(t)} \mid \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} \mid \tau_{i,<t})}, 1-\epsilon, 1+\epsilon \right) \hat{A}_{i,t} \right]. \tag{6}$$

In Equation (6), the structure mirrors that of PPO, except that the advantage is formed directly from normalized trajectory rewards rather than from a learned critic. This yields a simpler pipeline without auxiliary value networks, at the cost of potentially higher variance in advantage estimates.

5.5 RAGEN system design

RAGEN is a modular system that implements StarPO for LLM agents. It provides components for rollout generation, reward computation, and trajectory-level optimization. The system interfaces with various environments through a unified text-based API and supports both PPO and GRPO as optimization backends.

The rollout module handles multi-turn generation with structured reasoning and actions. The reward module allows for custom reward functions tailored to each environment, including success indicators and intermediate shaping signals. The optimization module supports batching over multiple initial states and multiple rollouts per state, together with generalized advantage estimation, entropy regularization, and format penalties that encourage valid reasoning–action structures.

The system is designed to be extensible so that new environments, reward schemes, and rollout strategies can be integrated without modifying the core training loop.

5.6 StarPO-S: stabilized variant with uncertainty filtering

Empirical observations show that naive multi-turn RL often leads to an instability pattern called Echo Trap, in which agents overfit to locally rewarded reasoning templates, diversity collapses, and gradient norms spike. To mitigate this, the study introduces StarPO-S, a stabilized variant of StarPO.

A central idea in StarPO-S is trajectory-level uncertainty-based filtering. For a given policy π_θ and MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P\}$, consider a specific task instance represented by an initial state s_0. Multiple trajectories can be sampled from this initial state. The outcome uncertainty is defined as the standard deviation of rewards over these trajectories:

$$U(\pi_\theta, \mathcal{M}, s_0) = \text{Std}_{\tau \sim \pi_\theta(\cdot|s_0)} \left[R(\tau) \right]. \tag{7}$$

In Equation (7), $U(\pi_\theta, \mathcal{M}, s_0)$ is a scalar that measures how variable the trajectory rewards are when starting from s_0. A high value indicates that sometimes the agent succeeds and sometimes fails on this instance, implying that the task is neither trivial nor hopeless. Such instances are particularly informative for learning.

StarPO-S sorts task instances by this uncertainty and retains only the top percentage with highest reward variability for gradient updates. This discards low-variance instances where the agent is consistently correct or consistently incorrect, focusing the optimization on examples that provide informative gradients.

In addition to uncertainty filtering, StarPO-S adopts gradient shaping techniques inspired by previous work. First, Kullback–Leibler divergence penalties between the current policy and an initial supervised model are weakened or removed to allow more flexible policy updates in multi-turn settings. Second, asymmetric clipping strategies such as Clip-Higher apply different clipping thresholds for positive and negative advantages, allowing beneficial updates more leeway while constraining harmful ones. Together, these modifications improve gradient stability, delay or avoid collapse, and enhance final performance across tasks.

6 Empirical Results

6.1 Training configuration and hyperparameters

The main experiments train Qwen-2.5 Instruct models of different sizes. A 0.5B-parameter variant is used for the three symbolic tasks (Bandit, Sokoban, Frozen Lake), and a 3B-parameter variant is used for WebShop. Training is conducted on H100 GPUs for 100 to 200 iterations of alternating rollout and update steps.

Each batch samples $P = 8$ prompts and $N = 16$ rollouts per prompt, with up to 5 turns and up to 10 actions per episode. Policy updates use PPO or GRPO together with Generalized Advantage Estimation (GAE) with discount factor $\gamma = 1.0$ and trace parameter $\lambda = 1.0$. The Adam optimizer is employed, with an entropy bonus coefficient $\beta = 0.001$ to encourage exploration and a small response-format penalty of -0.1 to discourage malformed outputs that do not follow the required `<think>`–`<answer>` structure.

Evaluation uses 256 fixed prompts per environment, decoding with temperature $T = 0.5$ and truncating episodes after 5 turns. The metrics monitored include success rate, rollout entropy, in-group reward standard deviation, response length, and gradient norm.

6.2 Baseline StarPO performance and instability patterns

The baseline StarPO framework with PPO or GRPO optimization is first evaluated across all environments. Symbolic tasks such as Bandit and Sokoban exhibit an initial increase in success rate followed by collapse as training progresses. PPO tends to be more stable than GRPO on these tasks, delaying collapse and achieving higher peak performance, consistent with the expectation that a learned critic can provide smoother value estimates in deterministic or low-noise environments.

In contrast, on the stochastic Frozen Lake environment, GRPO shows superior stability compared with PPO, likely because critic-based value estimation is more challenging when transitions are probabilistic. On WebShop, both PPO and GRPO achieve high success rates, with performance starting from a strong base and improving rapidly, attributable to rich pretrained language priors and meaningful reward signals.

To study collapse in more detail, early and late trajectories are compared. In Bandit, early-stage trajectories contain diverse reasoning sequences that reference symbolic semantics and trade-offs between arms, whereas late-stage trajectories degenerate into repetitive templates, indicating over-amplification of locally rewarded patterns.

Several indicators reveal the onset of collapse. Average training reward eventually plateaus or decreases after an initial improvement. In-group reward standard deviation often declines substantially before average reward degrades, signaling loss of behavioral diversity. Gradient norm exhibits sharp spikes, after which recovery is rarely observed. Output entropy, which reflects the randomness of token predictions, ideally follows a smooth decay but frequently shows erratic changes or abrupt drops in collapsing runs. These observations collectively reveal a characteristic Echo Trap pattern in multi-turn agent training.

6.3 Effect of StarPO-S and uncertainty-based filtering

StarPO-S is evaluated by varying the proportion of high-uncertainty instances retained for training. Filtering out low-variance rollouts significantly improves stability. For example, on Frozen Lake with PPO, keeping

Setting	Retained rollouts	Stability in symbolic tasks	Relative training time
No filtering	100%	Early collapse in Bandit and Sokoban	Baseline
Moderate filtering	75%	Delayed collapse, higher success plateaus	Reduced
Aggressive filtering	50%	Collapse largely mitigated in many runs	Further reduced

Table 1: Qualitative effect of uncertainty-based trajectory filtering in StarPO-S. Retaining only high-variance instances improves stability and reduces training time.

Responses per prompt	SingleSokoban	SokobanNewVocab	Frozen Lake
32	21.09%	20.22%	17.97%
16	20.31%	21.48%	19.53%
8	20.31%	19.53%	17.19%
4	20.70%	25.39%	21.48%
2	19.92%	25.00%	12.50%
1	19.53%	22.27%	12.50%

Table 2: Effect of task diversity on generalization performance. Four responses per prompt provide a good balance between task diversity and per-instance comparison.

only the top 75% highest-uncertainty instances extends the stable training regime compared with using all rollouts, and keeping 50% can essentially avoid collapse within the training horizon. Similar trends are observed on Sokoban, where filtering reduces the dominance of repetitive trajectories and preserves exploration longer.

Table 1 summarises the qualitative effect of uncertainty-based filtering. At the same time, fewer trajectories per update reduce total training time, leading to computational savings without sacrificing performance.

When combined with gradient shaping techniques such as KL penalty removal and asymmetric clipping, StarPO-S consistently outperforms vanilla StarPO across Bandit, Sokoban, Frozen Lake, and WebShop. Success rates are higher and collapse is delayed or avoided, confirming that carefully selected training data and controlled gradient updates are key to stable agent learning.

6.4 Task diversity, action budget, and rollout frequency

The study further analyses how different rollout design choices affect generalization.

6.4.1 Task diversity and responses per prompt

With a fixed batch size, increasing the number of distinct prompts reduces the number of responses per prompt, and vice versa. Table 2 shows the effect of varying the number of responses per prompt when training on Sokoban and evaluating on related tasks.

The results indicate that four responses per prompt yield the best overall generalization, particularly on SokobanNewVocab and Frozen Lake. This configuration offers sufficient diversity across tasks while preserving multiple rollouts per task for within-instance comparison.

6.4.2 Per-turn action budget

Another factor is the maximum number of actions permitted per turn. Table 3 presents success rates for different action budgets.

Max actions per turn	Sokoban	SokobanNewVocab	LargeSokoban	Frozen Lake
1	12.11%	13.67%	1.17%	11.72%
2	16.41%	21.09%	3.52%	18.36%
3	19.53%	19.53%	1.95%	20.88%
4	26.95%	26.95%	5.08%	20.70%
5	28.13%	25.78%	6.25%	21.09%
6	33.59%	31.64%	6.64%	18.36%
7	22.27%	28.52%	3.91%	19.53%

Table 3: Performance across environments for different per-turn action budgets. Moderate budgets around five to six actions per turn support effective planning without excessive noise.

Training task	Train on Bandit			Train on Sokoban		
Method	Bandit	BanditRev	Frozen Lake	LargeSokoban	Sokoban	SokobanNewVocab
StarPO-S	100.00%	67.58%	19.92%	2.34%	21.48%	18.75%
NoThink	81.25%	56.25%	19.53%	2.73%	20.73%	26.17%

Table 4: Generalization performance with and without explicit reasoning traces. Reasoning improves generalization in single-turn Bandit tasks but exhibits mixed effects in multi-turn Sokoban.

Action budgets of five or six yield the best performance across Sokoban and its variants. Budgets that are too small severely restrict planning, while overly large budgets introduce long and noisy rollouts that hinder learning.

6.4.3 Rollout frequency and Online-k strategies

Rollout freshness is studied by varying the Online-k parameter. Agents trained with Online-1, which collects new rollouts for every update, achieve the fastest convergence and strongest generalization. Higher values such as Online-5 or Online-10, where the same batch is reused multiple times, lead to slower improvement and lower final performance. This supports the principle that multi-turn RL is most effective when optimization is closely aligned with the current policy behavior, reducing policy–data mismatch.

6.5 Reasoning traces and their evolution

The effect of explicit reasoning traces is evaluated by comparing StarPO-S with its NoThink variant, which removes the `<think>` segment and only outputs actions. Table 4 summarizes generalization performance.

For Bandit and BanditRev, explicit reasoning substantially improves performance, particularly in the counterintuitive BanditRev setting where symbolic–reward associations are inverted. This suggests that reasoning traces help internalize symbolic structure beyond simple pattern memorization.

In multi-turn Sokoban tasks, the effect is mixed. The NoThink variant sometimes matches or exceeds StarPO-S on certain generalization benchmarks. Analysis of reasoning length, measured as the number of tokens inside the `<think>` block, reveals that reasoning traces shrink over time for many tasks.

Table 5 reports average reasoning lengths at different training steps.

The results show that reasoning verbosity declines as training proceeds. In the more challenging ReverseBandit environment, reasoning length remains relatively higher, consistent with the need for deeper processing. In multi-turn tasks with sparse or delayed rewards, final success rewards alone do not distinguish between coherent reasoning and trial-and-error strategies. As a result, the agent may learn to suppress

Task	Bandit			Sokoban		
Step	Original	NoThink	ReverseBandit	ReverseNoThink	Original	NoThink
0	66.0	12.7	68.8	12.7	307.1	68.6
100	25.6	12.4	33.7	13.0	104.6	55.3
200	17.6	12.4	30.7	13.0	89.5	60.0

Table 5: Average `<think>` token length at different training steps. Reasoning length decreases over time for many tasks, and remains higher in harder settings such as ReverseBandit.

explicit reasoning text and rely on shallow action patterns that still achieve success, at least under certain conditions.

This phenomenon underscores the importance of fine-grained, reasoning-aware reward design. Without explicit incentives for high-quality intermediate reasoning, multi-turn RL may encourage shortcut behaviors even in the presence of structured output formats.

7 Summary

This technical summary reviewed a study on training LLM agents in multi-turn, stochastic environments using trajectory-level reinforcement learning. The StarPO framework formulates agent training as an optimization over reasoning-augmented trajectories, unifying state representations, intermediate thoughts, actions, and rewards into a single objective. The RAGEN system implements this framework across symbolic and web environments, enabling systematic analysis of agent learning dynamics.

Empirical results reveal that naively applying single-turn RL methods such as PPO and GRPO to multi-turn settings often leads to an Echo Trap failure mode, characterized by collapse in reward diversity, repetitive reasoning templates, and gradient spikes. To address these challenges, the stabilized StarPO-S variant introduces uncertainty-based trajectory filtering and gradient shaping methods, which together improve stability, delay or prevent collapse, and yield higher success rates across environments.

The study further demonstrates that rollout design choices, including task diversity, per-turn action budget, and rollout frequency, significantly influence generalization and convergence. Moderate action budgets and high rollout freshness are particularly beneficial. Finally, the analysis of reasoning traces shows that explicit reasoning improves generalization in simple single-turn tasks, but tends to fade in multi-turn settings when rewards only reflect final outcomes. This highlights the need for more granular, reasoning-aware reward signals in the design of long-horizon RL for LLM agents.

Limitations of the current work include the focus on relatively small-scale environments, the absence of established RL components such as replay buffers, and the restriction to text-based tasks without multimodal inputs. Future directions include extending the framework to larger and more complex environments, incorporating multimodal observations and actions, and designing reward models that explicitly evaluate and reinforce the quality of intermediate reasoning steps.