RL training for LLM agents using RAGEN

Andnet DeBoer Chenyu Zhu Praneeth Reddy Mallupalli

Northwestern University, CS 496 Pilot Project

Abstract

As large language models (LLMs) increasingly act as autonomous agents, optimizing their ability for multi-step, sequential reasoning becomes critical. In this pilot project, we setup a reinforcement learning (RL) training framework using RAGEN to train a text-based LLM agent to play the Sokoban puzzle game. We used Qwen 1.5B as our base model and conducted a comparative analysis of Proximal Policy Optimization (PPO) and Group-Relative Policy Optimization (GRPO) under matched compute conditions. Our findings reveal that both algorithms achieve high syntactic validity, but GRPO slightly outperforms PPO in overall success rate (13.9% vs. 12.9%) while offering significant memory advantages. Ultimately, this framework successfully demonstrates how RL can shape an LLM's internal reasoning traces to navigate complex, irreversible environments.

1 Introduction



Figure 1: A representative Sokoban board state. The agent must successfully navigate the grid to push all boxes onto the red targets without entering into irreversible deadlocks.

Teaching an LLM to take actions in a discrete environment requires moving beyond simple text generation to the realm of sequential decision-making. We chose Sokoban as our test environment because it perfectly stresses an LLM agent's ability to plan. While the action space is simple (Up, Down, Left, Right), the game requires multi-step reasoning, and wrong moves can easily result in irreversible deadlocks.

Our main objective was to setup a robust RL training framework using RAGEN to see if we could teach an LLM to play Sokoban. Once the pipeline was established, we sought to compare two distinct policy-gradient algorithms: Proximal Policy Optimization (PPO) and Group-Relative Policy Optimization (GRPO). By logging our training runs with Weights & Biases, we captured metrics on reward, success rates, and reasoning length.

In summary, our contributions are as follows:

- We successfully implemented and ran the RAGEN framework for an LLM agent in a Sokoban environment.
- We evaluated and compared PPO and GRPO on training efficiency, success rates, and action validity.
- We analyzed the model's generated `<think>` traces to better understand its step-by-step reasoning.

2 Related Work

Recent advances in agentic AI are heavily based on Reinforcement Learning from Human Feedback (RLHF) and related policy optimization methods to align model behavior with specific goals. PPO has long been the industry standard for these tasks due to its stability in complex, dense reward environments. However, PPO is highly memory-intensive because it requires maintaining Actor, Critic, Reference, and Reward models simultaneously (often consuming $\sim$4x the model size in VRAM).

More recently, GRPO has emerged as a lightweight alternative optimized for reasoning tasks and memory-constrained clusters. By eliminating the Critic model and instead using group sampling relative to the mean to estimate advantages, GRPO saves roughly 50% of VRAM. Our work builds upon these paradigms by applying them directly to a spatial reasoning task, providing a practical comparison of the two algorithms on a 1.5B parameter model.

3 Method and Theory

3.1 The RAGEN Framework and Agent Setup

At a high level, RAGEN is a specialized framework designed to turn a text-based LLM into an active, environment-grounded agent. Rather than passively answering static prompts, the agent interacts with a live environment in a continuous feedback loop.

We utilized a standard Sokoban board state containing grids, boxes, and targets. During each turn, the agent sees the current state of the board. The model (Qwen 1.5B) then outputs a `<think>...</think>` block where it reasons through its next move, followed by a final `<answer>` block containing one of the four discrete actions (Up, Down, Left, Right). Once the action is parsed and executed, the environment updates, and the new state is fed back to the model for the next turn.

3.2 Reward Design

The environment reward was structured to encourage efficient puzzle-solving while penalizing aimless wandering. We implemented a step penalty for every move taken, paired with a significant positive reward for successfully placing a box on a target.

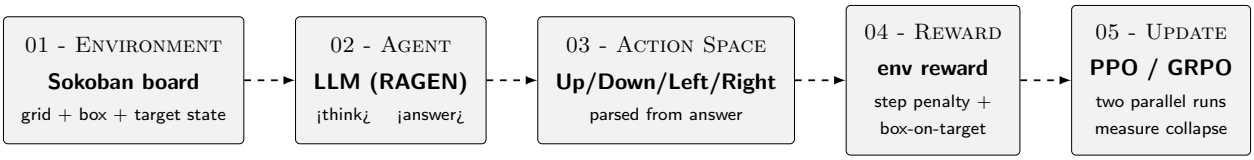


Figure 2: The reinforcement learning training pipeline. The environment state is fed to the agent, which generates an action that computes a reward, subsequently driving the policy update.

3.3 Algorithmic Comparison

We ran two parallel training pipelines:

- **PPO (Actor-Critic):** Uses Generalized Advantage Estimation (GAE) via a Critic model to compute advantage.
- **GRPO (Group-Relative):** Computes advantages as a relative reward within a sampled group, bypassing the need for a separate value model.

4 Experiments and Evaluation

4.1 Setup and Key Metrics

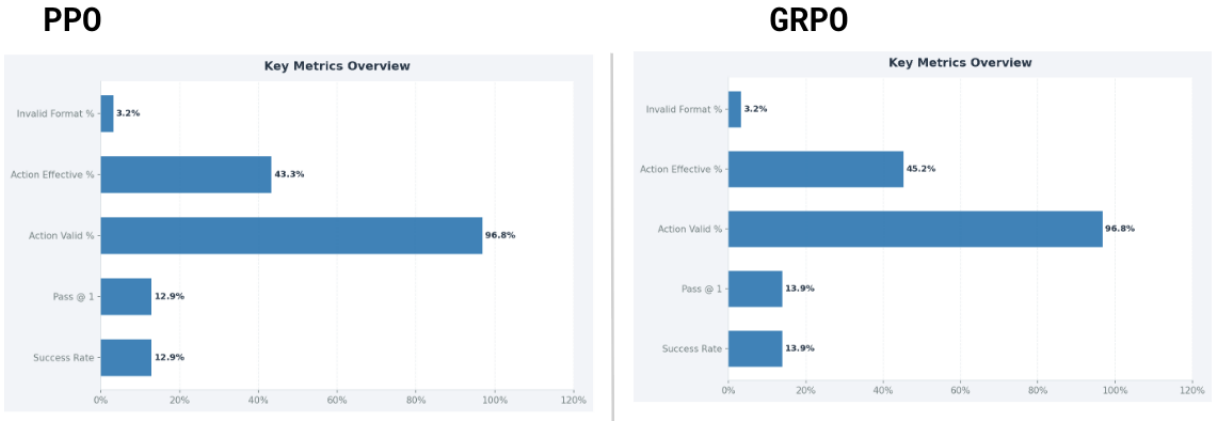


Figure 3: Overview of key performance metrics comparing PPO and GRPO

We evaluated our agent across 512 Sokoban episodes. Due to compute constraints, the agent was given a strict limit of up to 5 turns per episode. We tracked four primary metrics to gauge baseline capabilities:

1. **Invalid Format %:** How often the model's output was completely malformed. Both models excelled here, registering only 3.2%.
2. **Action Valid %:** Did the model output a syntactically correct move (e.g., "move left")? Both PPO and GRPO maintained a highly consistent 96.8% validity rate.
3. **Action Effective %:** Of the syntactically valid actions, how many actually changed the board state (i.e., not walking into a wall)? GRPO showed a slight edge here, reaching 45.2% compared to PPO's 43.3%.

4. **Pass@1 / Success Rate:** In this setup, these metrics are identical. PPO successfully solved 12.9% of the episodes, while GRPO performed slightly better at 13.9%.

4.2 Episode and Reward Analysis

Diving deeper into the episode data revealed several distinct behavioral patterns:



Figure 4: A comprehensive evaluation of PPO and GRPO across 512 evaluation episodes. Top left: Overall episode success and failure rates. Top right: Episode outcomes mapped against total turns taken. Bottom left: Frequency of total actions per episode. Bottom right: The bimodal distribution of cumulative RM scores.

- **Episode Outcomes:** Out of our 512 evaluation episodes, PPO recorded 66 successes and 446 failures (an 87.1% failure rate). GRPO managed 71 successes and 441 failures (an 86.1% failure rate).
- **Episode Length (Turns):** We observed a drastic split based on the length of the run. All episodes that concluded in under 5 turns were 100% successful. In contrast, when the agent exhausted the 5-turn limit, it almost never succeeded.
- **Actions per Episode:** Supporting the length data, our charts showed that successful runs peaked rapidly at just 2 actions. The agent usually solved the puzzle immediately if it could. Failures, however, were characterized by long, wasteful runs peaking at 5 and 6 actions.
- **RM Score Distribution:** Because of the step penalties, the cumulative reward scores were strictly bimodal. Successes clustered heavily around a score of +10.7 (10.71 for PPO, 10.68 for GRPO). Failures clustered tightly around -0.67 for PPO and -0.69 for GRPO, directly reflecting the accumulation of step penalties as the model fruitlessly ran out of turns.



Figure 5: W&B Training Curves displaying response length, action effectiveness, and validity.

4.3 Training Curves

Tracking the internal logs provided excellent visibility into the learning progression across four distinct graphs:

1. `train/response_length`: The length of the generated tokens steadily dropped over the training steps. As the model learned the environment, its internal reasoning traces became more concise and targeted.

2. `train/CoordSokoban/num_actions`: The average number of actions per step remained relatively high, oscillating generally between 4.0 and 5.0 throughout the run, aligning with our strict turn limits.

3. `train/CoordSokoban/action_is_valid`: The validity of the model's syntax stabilized very quickly and stayed consistently near 1.0 for the duration of the training, showing it learned the game's grammar immediately.

4. `train/CoordSokoban/action_is_effective`: This curve provided the clearest view of actual reinforcement learning. It started incredibly low (around 0.2) at the beginning of training, but climbed steadily and sharply, eventually hovering near 0.7 to 0.8. The agent actively learned to push boxes efficiently instead of making useless moves against boundaries.

4.4 Case Study: Reasoning Traces

Saving the `<think>` traces allowed us to inspect the model's logic rather than just its final reward. Early in training, the model's reasoning was often disjointed. By the end of training, we observed structured spatial reasoning. A representative success trace looks like this:

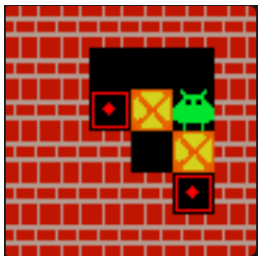
> *Model Input State:* Box at (2,3), Target at (2,4). Player at (2,2).
> `<think>` The player is directly to the left of the box. The target is directly to the right of the box. If the player moves Right, they will push the box into the target space.

There are no walls blocking this movement. </think>
<answer> Right </answer>

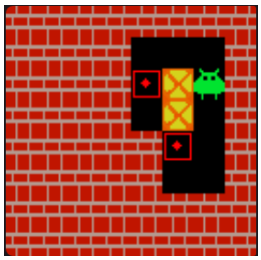
This explicit step-by-step breakdown prevents the model from taking impulsive, irreversible actions, directly contributing to the high valid action rate.

4.5 Qualitative Result on Solved Maps

To better understand what the trained agent could actually solve, we inspected successful rollouts after trying 1000 random seeds. The strongest cases were not very complicated Sokoban maps, but relatively small two-box layouts, as shown in Figure 6. These examples suggest that the current policy learned some local Sokoban mechanics, especially how to recognize short pushing opportunities when the player, boxes, and targets are already close to each other.



(a) Successful 6x6 two-box case.



(b) Successful 8x8 two-box case.

Figure 6: Qualitative examples of the strongest successful cases found after trying 1000 random seeds. The current agent succeeds mainly on small two-box maps where the solution can be completed through short-horizon local planning.

However, we did not observe reliable success on very complicated maps that require long setup sequences, multi-box coordination, or recovery from difficult intermediate states. Therefore, the current result should be interpreted as a proof of concept on simpler Sokoban layouts rather than a fully trained policy that can solve complex Sokoban environments.

5 Conclusion and Discussion

In this project, we established an RL framework for training an LLM agent with RAGEN in a discrete Sokoban environment. The agent learned to produce mostly valid actions and showed some ability to solve short-horizon boards. However, the results should be interpreted carefully: we did not manage to train an agent that can reliably solve very complicated maps. The successful examples were mainly simple two-box configurations where the player, boxes, and targets were already arranged close enough for a small number of local moves.

Within this limited setting, GRPO showed a small advantage over PPO in success rate and action effectiveness, while also offering a memory advantage by avoiding a separate critic model. This suggests that GRPO is promising for our RAGEN setup, but the current result is not strong enough to claim that the agent has learned robust long-range spatial planning.

The main limitation of the current setup is still the short episode horizon. Since Sokoban often requires setup moves before a box can be scored, the 5-turn limit makes deeper puzzles especially difficult. Future work should increase the turn limit, train with a broader curriculum of map difficulty, and evaluate whether the model can solve puzzles that require detours, coordinated box movement, and prevent deadlocks.