# Expanding LLM Agent Boundaries with Strategy-Guided Exploration

**Andrew Szot** Apple
**Michael Kirchhof** Apple
**Omar Attia** Apple
**Alexander Toshev** Apple

## Abstract

Reinforcement learning (RL) has demonstrated success in post-training large language models (LLMs) as agents for tasks such as computer use, tool calling, and coding. However, exploration remains a central challenge in RL for LLM agents, especially as they operate in language-action spaces with complex observations and sparse rewards. In this work, we address exploration for LLM agents by leveraging the ability of LLMs to plan and reason in language about the environment to shift exploration from low-level actions to higher-level language strategies. We thus propose Strategy-Guided Exploration (SGE), which first generates a concise natural-language strategy that describes what to do to make progress toward the goal, and then generates environment actions conditioned on that strategy. By exploring in the space of strategies rather than the space of actions, SGE induces structured and diverse exploration that targets different environment outcomes. To increase strategy diversity during RL, SGE introduces mixed-temperature sampling, which explores diverse strategies in parallel, along with a strategy reflection process that grounds strategy generation on the outcomes of previous strategies in the environment. Across UI interaction, tool-calling, coding, and embodied agent environments, SGE consistently outperforms exploration-focused RL baselines, improving both learning efficiency and final performance. We show that SGE enables the agent to learn to solve tasks too difficult for the base model.

## 1 Introduction

Large language models (LLMs) are a promising foundation for building agents across a wide range of downstream tasks such as computer use [OpenAI, 2025], coding [Anthropic, 2025], tool usage [OpenAI, 2025], and robotics [Kim et al., 2024]. A key driver of progress in LLM agents is reinforcement learning (RL), which trains LLMs to autonomously act to solve tasks in external environments. RL has shown the ability to teach LLMs to reason, self-correct, and interact with complex environments using simple, rewards without human feedback [Guo et al., 2025].

Exploration is a central challenge in training LLM agents with RL. This challenge is amplified by sparse outcome rewards, which reward the agent only when it achieves the desired goal. In domains such as video game playing [Silver et al., 2017] or navigation [Wijmans et al., 2019], RL can train superhuman policies starting from a base policy that performs poorly. In contrast, RL for LLM agents operates in complex language action spaces such as code outputs or tool calls, and is initialized from a pretrained model that induces a concentrated policy over likely outputs. As a result, RL for LLM agents predominantly samples from and refines behaviors already supported by the base model, limiting the agent's ability to discover new successful trajectories. Consistent with this, empirical evidence from prior work in non-agentic reasoning domains demonstrates that RL post-training struggles to learn new tasks and instead refines outputs to high-reward trajectories the model is already capable of generating [Wu et al., 2025a, Yue et al., 2025].

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

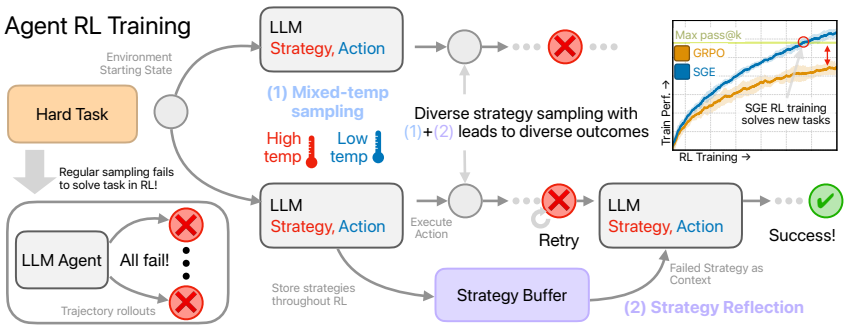



Figure 1: Overview of Strategy-Guided Exploration (SGE). SGE improves reinforcement learning (RL) training on hard agentic tasks for which the base model fails (left of figure). SGE addresses this by having the LLM policy output a language "strategy" and conditioning action generation on this strategy. The reasoning capabilities of the LLM enable diverse strategy generation through: (1) mixed-temperature sampling, where strategy tokens are sampled with higher temperature, and (2) strategy reflection, where strategies are generated to be distinct from earlier strategies in RL training. This enables SGE to explore to solve tasks that the base model is not capable solving (right of figure).

To improve exploration in RL training for LLM agents, we introduce Strategy-Guided Exploration (SGE), which expands agent capabilities by exploring new tasks not solvable by the base model. SGE leverages the ability of LLMs to plan and reason in natural language about the agent's environment. Instead of treating exploration as sampling environment actions, SGE first generates a language "strategy" and then produces the environment action conditioned on that strategy. The strategy is a concise and specific natural language description of what to do to make progress toward the goal. We show that with SGE, exploring in the space of strategies is easier than exploring in the space of actions, and that diverse strategies lead to better exploration in the environment, which increases the likelihood of success. Since strategies are high-level language descriptions of intended outcomes, the LLM can use its reasoning abilities to generate many distinct strategies. Conditioning on these strategies then guides the LLM to produce action sequences consistent with the strategy that explore different environment outcomes.

SGE introduces several techniques to generate diverse strategies during RL training to improve exploration. SGE does more than simply prompt the LLM agent to first produce a strategy, and introduces techniques to maximize strategy diversity throughout training. The first component of SGE is mixed-temperature sampling, which samples the strategy at a higher temperature than the rest of the token sequence. Higher temperature for action sampling often produces different actions that achieve the same environment outcome, such as clicking the same UI element at different positions. In contrast, increasing temperature in the higher-level strategy language space generates strategies that correspond to different outcomes. The second component of SGE is strategy reflection, where the model reflects on successful and failed strategies from earlier in RL training. This reflection helps ground strategy generation in the details of the environment dynamics and further diversify strategies from previous attempts. See Figure 1 for an overview of the SGE method.

Our experiments show that SGE solves challenging tasks during RL training that the base model cannot solve even after many attempts across the multi-step agentic environments of coding [Xia et al., 2025a], UI control [Rawles et al., 2024], tool-calling [Trivedi et al., 2024], and embodied agents [Szot et al., 2023]. SGE also outperforms exploration-focused RL baselines, achieving higher final performance and better learning efficiency. For example, in the multi-turn coding environment, the starting LLM solves 69% of the problems *at least once* out of 2048 attempts (pass@2048=0.69). The best performing RL baseline with a 64% final success rate (pass@1=0.64) is unable to surpass this ceiling in task pass rate as it attains no learning signal for the remaining difficult problems. SGE, capable of exploring to solve harder problems, achieves a 73% success rate. These training gains then translate to better generalization on unseen tasks. We demonstrate the broad applicability of SGE by showing consistent improvements with SGE across four diverse environments involving different observation and action spaces. We also ablate the SGE components of mixed-temperature sampling and strategy reflection, along with analyzing the impact of SGE across LLM size. We qualitatively analyze how the LLM is able to generate and use diverse strategies to solve new tasks

that are unsolvable by the base model. Overall, our experiments show how SGE uses exploration to unlock scaling RL training to expand agentic capabilities on difficult tasks.

## 2 Related Work

Exploration is a long-standing central challenge of RL. Simple forms of exploration based on action sampling struggle in sparse-reward or long-horizon problems that require complex exploration behaviors [Mnih et al., 2013, Sutton and Barto, 2018]. To address these shortcomings, some works model the uncertainty around the dynamics or state of the environment and use this as a learning signal to explore the environment [Burda et al., 2018, Pathak et al., 2017]. Other methods introduce temporally extended exploration without explicit uncertainty estimation [Osband et al., 2016, Plappert et al., 2017, Bellemare et al., 2016, Martin et al., 2017, Tang et al., 2017]. Hierarchical RL [Sutton et al., 1999, Nachum et al., 2018] and options learning [Bacon et al., 2017] seek to better handle exploration over long-horizon tasks. Like these works, SGE also focuses on complex exploration strategies but leverages the strengths of LLMs to do so. Other works use language for hierarchical control, but do so for communication between planner and control policies [Belkhale et al., 2024], not to improve exploration for solving harder tasks as in SGE.

Exploration in RL is especially important for LLM-based policies as they typically operate in complex language action spaces, such as open-ended code generation, and are rewarded for achieving sparse outcomes, such as calling a sequence of tools that accomplishes a goal. Prior work focusing on non-agentic reasoning tasks suggests that RL for LLM post-training primarily refines existing abilities of the base LLM instead of teaching the LLM to solve new problems that are unsolvable by the base model [Yue et al., 2025, Wu et al., 2025a]. We show that the improved exploration from SGE helps exceed these limits in agentic tasks. Works attempt to maintain high output entropy throughout training; however, these techniques are more designed to help training stability and less to promote exploration [Cui et al., 2025, Wang et al., 2025]. Other works directly train for policy diversity using a "pass@$k$" reward during training, where the model is rewarded if any attempt out of $k$ attempts succeeds [Chen et al., 2025a, Tang et al., 2025]. These methods also try to prevent policy collapse, but still require at least one of the $k$ attempts to succeed under the base model to obtain a learning signal. Unlike these works, our method incentivizes exploration in problems where the base model achieves minimal starting pass@$k$.

Other methods introduce exploration-specific techniques to LLM RL training. Gao et al. [2025] uses the random network distillation framework [Burda et al., 2018] to reward novel token sequences while Cheng et al. [2025] directly rewards the policy entropy. Our method also incentivizes exploration, but does so through language reasoning, and we demonstrate that it empirically outperforms these added exploration objectives. Furthermore, these works focus on LLM reasoning for math or reasoning problems with a question-and-answer format, while we focus on agentic settings where an LLM agent takes a series of actions to achieve a goal. Song et al. [2025] adds a UCB exploration reward over the predicted final answer, but we focus on action spaces with an unbounded number of action possibilities. Li et al. [2025] uses a pretrained diversity classifier score as a reward, which we do not have in our multi-step agentic setting. Other works use the pretrained knowledge of LLMs to provide policy feedback [Klissarov et al., 2023], which is orthogonal to how SGE uses the LLM to reason over strategies for better exploration in sparse reward settings. Works also use LLMs to guide exploration in other non-LLM policies [Hao et al., 2025, Tam et al., 2022], whereas we focus on exploration in a single end-to-end LLM agent.

Other works on post-training LLMs for reasoning also introduce language abstractions or plans of reasoning steps or solutions. Qu et al. [2025] uses distillation to first output a "reasoning abstraction", similar to a strategy in SGE. However, SGE does not use a teacher model and introduces diversity-maximizing techniques in RL such as mixed-temperature sampling and strategy reflection. Wu et al. [2025b] also requires multiple expert models to generate different modes and focuses on distillation rather than online RL, like in SGE. Chen et al. [2025b] introduces a "hint" to guide policy responses, but this hint is generated from the ground truth answer. Unlike these works, SGE operates in multi-step agentic settings where only the desired final outcome is specified, such as passing a set of tests, and the ground truth correct action response is unknown.

# 3 Approach

## 3.1 Preliminaries

We formalize the agentic tasks we focus on as Partially Observable Markov Decision Processes (POMDPs) with goal space $\mathcal{G}$, observation space $\mathcal{O}$, action space $\mathcal{A}$, environment transition function $\mathcal{T}$, and a reward model $R$. For brevity, we omit other elements of the MDP. In the settings we study, $\mathcal{G}$ is a natural language description of the goal, $\mathcal{O}$ is a textual or visual input, and $\mathcal{Y}$ is the space of any textual outputs by the LLM. The environments we consider involve actions like tool-calling and coding, thus the action space $\mathcal{A} \subset \mathcal{Y}$ is expressed as natural language. The reward model $R$ is a sparse reward assigned at the end of the episode for successfully completing the task.

We consider an LLM policy $\pi : \mathcal{G} \times \mathcal{O} \to \mathcal{Y} \times \mathcal{A}$ with the goal of maximizing the POMDP cumulative reward. At the episode start, the LLM receives a goal $g \sim \mathcal{G}$ and starting observation $o_1 \in \mathcal{O}$. First, the LLM policy samples an intermediate reasoning trace (chain of thought) $y_1 \sim \pi(\cdot \mid g, o_1)$ and then produces the action $a_1 \sim \pi(\cdot \mid g, o_1, y_1)$ which is executed in the environment and returns the next observation $o_2 = \mathcal{T}(o_1, a_1)$. This processes repeats to sample a trajectory $o_1, y_1, a_1, o_2, y_2, a_2, \ldots, a_T, o_T, r_T$ where $r_T$ is the outcome reward. The objective is to train the LLM policy with RL to maximize the expectation of rewards over $\mathcal{G}$.

## 3.2 Strategy-Guided Exploration

A key challenge in RL is learning from sparse reward functions that only score the binary success of achieving the goal. LLM agents operating in complex environments and language action spaces are unlikely to stumble onto the goal by chance, thus the starting LLM must already have the capability to intelligently search to find the goal. Prior work shows that this reliance on the base model's sampling prevents discovering truly new solutions [Yue et al., 2025, Wu et al., 2025a, Sun et al., 2025].

We thus introduce Strategy-Guided Exploration (SGE) to incentivize the LLM policy to explore and discover solutions to tasks unsolvable by the starting policy. The key idea of SGE is for the policy to produce high-level language strategies before actions, and then explicitly encourage diversity in strategy sampling to drive exploration. SGE does not require ground truth solutions, privileged information or access to a stronger LLM, and instead modifies the LLM sampling process during RL training. Specifically, SGE makes three modifications over standard RL training: (1) strategy prompting, (2) mixed-temperature sampling, and (3) strategy reflection.

**Strategy Prompting**: For goal $g \in \mathcal{G}$ and observation $o_t$ at episode step $t$, instead of sampling an intermediate reasoning trace $y_t \sim \pi(\cdot \mid g, o_t)$, we first sample a strategy $s_t \sim S_\pi(\cdot|g, o_t)$ from "strategy sampling distribution" $S_\pi$. For clarity, we subsume the goal and observation $(g, o_t)$ into just the observation $o_t$. The intermediate reasoning trace and then action is generated with this strategy, meaning the full output distribution is $\pi(a_t|y_t, s_t, o_t)\pi(y_t|s_t, o_t)S_\pi(s_t|o_t)$. The strategy is a concise language description that maps to a specific and non-overlapping distribution of actions. As we will empirically demonstrate, $S_\pi$ induces actions that explore the environment more efficiently than general intermediate reasoning or direct action sampling. The strategy is different from the other intermediate chain-of-thought text or action tokens in that it is sampled from the different distribution $S_\pi$, which uses a specific prompt and two additional techniques for generating diverse strategies described later in this section.

Specifically, for a given observation, SGE modifies the prompt to include text like: *First, give a strategy of what action to take after 'Strategy:' then generate the action after 'Action:'*. Producing a strategy before an action is established in prior work [Qu et al., 2025, Chen et al., 2025b], as are reflection and the reuse of past experience in memory-augmented agents [Wang and Jiang, 2026, Wu et al., 2025c, Xia et al., 2026, 2025b, Zhang et al., 2026]. Our contribution is the combination of mixed-temperature sampling and outcome-conditioned strategy reflection to encourage strategy diversity and improve exploration during end-to-end RL training, as we demonstrate empirically.

**Mixed-Temperature Sampling**: This generates strategies at a higher token sampling temperature than typical for the model inference. This higher temperature only affects the tokens sampled from the strategy distribution $S_\pi$ and not the remaining LLM outputs from $\pi$. This means that the model response is decoded with a mixed-temperature, where the initial strategy is decoded with a high temperature and the remaining tokens are decoded with a comparatively lower temperature. The insight is that diverse strategies will more effectively explore the environment than directly sampling

more diverse actions. Sampling actions with a high temperature can result in actions that are only superficially distinct. For example, in UI control, high-temperature action sampling can result in the agent tapping at different coordinates on the same button. This added noise could hinder RL if, for example, the UI agent's noisy tap locations sometimes click off the intended button. We empirically find it is easier for the model to sample different strategies, and then condition the lower-temperature action generation on these strategies. To explore different strategies, SGE produces $K$ parallel strategies per task, which has no added generation overhead over the group-based RL algorithms typically used in LLM RL training, such as GRPO [Shao et al., 2024] or RLOO [Ahmadian et al., 2024], which require $K$ parallel responses per task for advantage estimation.

**Strategy Reflection**: During RL training, SGE uses outcomes from earlier attempts at the same task to encourage alternative strategies. With some probability, SGE does "Negative Strategy Reflection". If a strategy fails (meaning the outcome reward $r_T = 0$), then we condition a subsequent rollout in the same task on the failed strategy sequence and instruct the policy to critique it and generate an alternative that addresses its flaws. The prompt explicitly encourages diversity: *"Your new strategy must be different from this failed strategy sequence and try something new"*. The policy receives only the failed strategy sequence, not the associated actions or intermediate observations. In Coding, we additionally include the verifier feedback for the failed strategy (the program output with failed tests or runtime errors), which all methods also receive as an observation after the first attempt. Likewise, SGE also incorporates "Positive Strategy Reflection" where, with some probability, strategy generation is conditioned on a strategy from the same task that ended in success. The prompt instructs the agent to produce a new strategy inspired by the successful example in the same task, encouraging exploration of variations on successful strategies. We find that the negative reflection enables the policy to solve new episodes beyond parallel strategy sampling. Positive reflection results in better learning efficiency as the agent is able to better leverage successful examples and maintain higher output entropy with multiple successful strategies solving the same task. Strategies are stored in task-specific buffers and used to condition future rollouts for the same task, so the reflected strategies are from old versions of the policy. This further boosts strategy diversity over the strategies purely generated from a single policy instance.

We detail all the prompts, including the strategy and reflection prompts in Section A.1. All prompts are short and consistent between environments with slightly different language in how actions are referred to depending on the environment and LLM. Environments are multi-step, so we generate a per-action strategy at every step. In strategy reflection, we condition the reflection process on the entire sequence of strategies from the previous episode. Past strategy sequences thus guide exploration during training; we evaluate the trained policy without mixed-temperature sampling, strategy reflection, or retrieval of stored experience. SGE and all baselines are evaluated using the same strategy prompt and decoding settings. Our evaluation domains have on the order of 10's of steps per episode, which keeps this extra context manageable. Future work extending SGE to environments with more steps could summarize long sequences of per-step strategies into a more compact meta-strategy to save the cost of the extra context.

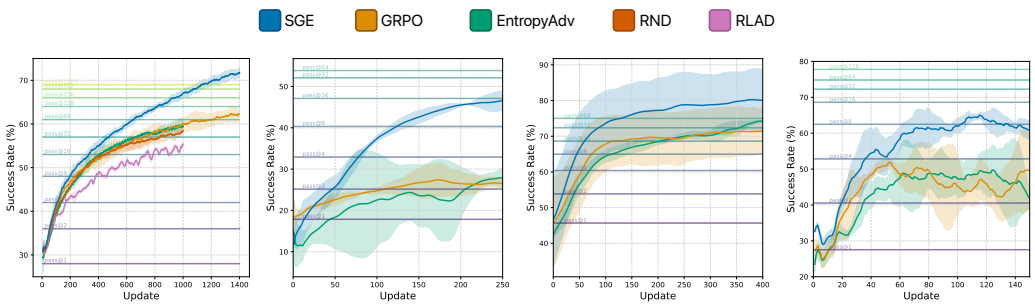
### 3.3 RL Details

We use SGE with GRPO [Shao et al., 2024]. However, since SGE only alters the sampling distribution via mixed-temperature sampling and policy conditioning via strategy reflection, it is compatible with any online RL algorithm. Strategy tokens, like all other generated tokens, are included in the GRPO objective with the same trajectory-level advantage. Log-probabilities are computed under the temperature-scaled behavior policy, using the same per-token temperature as in sampling, which varies between the strategy and remaining tokens. We also use the GRPO modifications proposed by DAPO [Yu et al., 2025], except without the clip higher probability ratio, which we found was broadly harmful across all our experimental domains and baselines. We detail all method details, prompts, and hyperparameters in Section A.

## 4 Experiments

### 4.1 Environments and Baselines

We show results across four diverse agentic domains to demonstrate the broad applicability of SGE.




(a) Coding (b) AndroidWorld (c) LangR (d) AppWorld

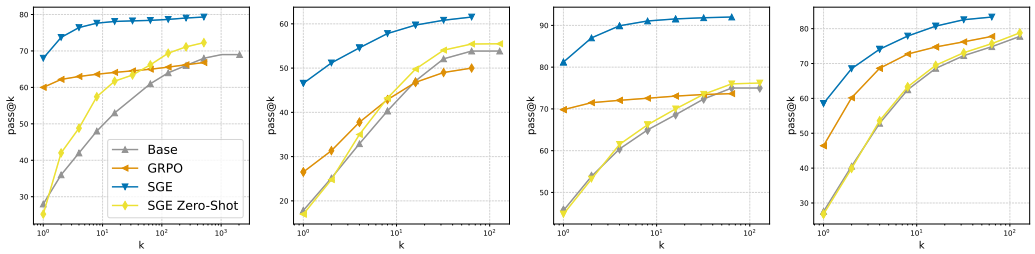
Figure 2: RL training curves for SGE and baselines in each environment. Training curves show the average pass@1 rate versus the number of RL updates for the training tasks across 3 random seeds in training runs, with the shaded area representing the standard deviation of the pass@1 rate across the seeds. The horizontal lines indicate the base model pass@$k$ rate with lighter shades meaning a higher $k$. In the Coding environment, SGE and GRPO are trained longer than the other approaches to show better converged performance. RND and RLAD baselines are also only shown for this environment since they underperform the GRPO and EntropyAdv.

- AndroidWorld [Rawles et al., 2024] is a phone UI control environment where the agent operates from RGB pixel inputs and outputs low-level UI interactions, including tapping pixel coordinates, swiping, or typing text. We use 26 of the tasks for training and evaluate on 30 test tasks consisting of non-visual question answering tasks in the overall AndroidWorld task set. Both the train and test sets span 15 apps, with the test set including tasks on 4 apps unseen in the training set.
- Language Rearrangement (LangR) [Szot et al., 2023]: An embodied household agent is instructed to rearrange objects in a home environment to complete a language instruction, such as "Find and put all the apples in the fridge". We use a textual environment observation, which is a partially observable textual representation of the robot's vicinity, including the object the robot is currently holding, and which objects and receptacles are in front of the robot. The action space includes high-level skills such as picking objects and navigating to receptacles. We use the standard train split and evaluate generalization to unseen houses.
- Coding [Xia et al., 2025a]: Given a coding problem description, the agent must output Python code to pass a set of unit tests. We use only the "hard" category of problems from the dataset, giving 606 tasks from the train split and 228 tasks from the test split. We make this environment multi-turn by providing the output of the executed program as an observation for the policy to produce updated Python code in another attempt.
- AppWorld [Trivedi et al., 2024]: A multi-step tool-calling benchmark in a simulated app ecosystem across 9 applications and 457 APIs. We only train and test on the "Easy" categorization of problems, which are still challenging for the LLMs we finetune. This filtering results in 36 training problems, and we evaluate generalization to unseen tasks with the 57 tasks from the "Test Normal" split.

We train separate models for each environment. See Section F for full details.

We compare SGE against the following approaches for LLM RL training, with a focus on baselines that are intended to improve exploration.

- **GRPO**: Regular GRPO with the standard policy gradient objective.
- **Entropy Advantage (EntropyAdv)** [Cheng et al., 2025]: Modifies GRPO to incentivize exploration by adding the policy entropy to the advantage for every token. This approach was demonstrated to be more effective for LLM RL exploration than standard entropy regularization that adds an entropy loss term to the RL objective [Schulman et al., 2017].
- **Random Network Distillation (RND)** [Gao et al., 2025]: Adds an exploration reward for producing novel actions using random network distillation (RND) [Burda et al., 2018]. RND trains a network to predict the output of a randomly initialized frozen target network based on the final LLM activation for each action. The prediction error is provided as an exploration reward when the agent fails to achieve the goal, encouraging the model to explore novel output sequences.




(a) Coding (b) AndroidWorld (c) LangR (d) AppWorld

Figure 3: Comparing the pass@$k$ of the final policies trained by SGE and GRPO and the starting base model across the environments on the training tasks. We compute the pass@$k$ for $k$ in powers of 2, and the x-axis is on a logarithmic scale. The "Base" line shows that the pass@$k$ of the base model eventually plateaus for large enough $k$. GRPO raises the "Base" curve, enabling the pass@1 to closely match the pass@$k$ for the highest $k$. On the other hand, SGE exceeds this pass@$k$ ceiling of the base model and enables solving new problems that are unsolvable by the base model. "SGE Zero-Shot" is the base model with SGE's mixed-temperature sampling and no RL training. We report averages across all tasks in the training set.

- **RL with Abstraction Discovery (RLAD)** [Qu et al., 2025]: This work first outputs an abstraction for the problem, akin to the strategy in SGE, and then conditions the response on the abstraction. RLAD trains the abstraction and solution generator by augmenting training with a KL divergence penalty and sometimes dropping out the strategy. We use the same LLM weights for both roles of abstraction and solution generator. RLAD serves as a baseline also addressing explicit strategy generation.
- **Base model pass@k**: This zero-shot evaluates the base model with $k$ independent attempts per task and is the percentage of the time that any of the attempts are successful as judged by the ground truth task verifier. This shows how methods improve on the existing capabilities of the base model.

We find prompting the policy to produce a strategy before the action to be marginally useful for even non-SGE baselines; therefore, all baselines use this improved prompt for a fair comparison (see Section B for more details). Thus, the difference between SGE and the standard GRPO baseline is that SGE uses the mixed-temperature sampling and strategy reflection. All methods, SGE included, are trained with the GRPO. For LangR and Coding, we finetune the Qwen3-4B-Instruct model [Yang et al., 2025a]. For AppWorld, we finetune the Qwen3-8B reasoning model, as we found the larger size was necessary for stable GRPO training. For AndroidWorld, which has visual observations, we finetune the Qwen2.5-VL-3B model [Bai et al., 2025]. We only run the RND and RLAD baselines on Coding since they underperform the GRPO and EntropyAdv baselines. All hyperparameters and full implementation details are provided in Section A.

Note that since we operate in an agentic setup where an agent needs to take actions to achieve a goal, we cannot apply some of the methods from prior works that are specific to single-step reasoning question answer domains. We cannot apply the UCB exploration method from Song et al. [2025] since there is an unbounded number of action trajectories that can lead to the goal. The agentic environments we evaluate in also do not contain reference solutions, so we cannot compare to the method from Chen et al. [2025b], which evaluates in the math domain and assumes access to the problem answers. Li et al. [2025] assumes access to a pretrained diversity classifier, which is difficult to apply to strategy and action sequences.

### 4.2 Empirical Comparison to Baselines

**RL training results:** We compare SGE RL training to baselines in Figure 2, which demonstrates SGE achieves higher RL training performance. Specifically, we plot the average pass@1 rate, which is the percentage of the time the agent succeeds in solving the problem on the train set, along with the standard error across 3 random training seeds in RL training. The overlaid horizontal lines indicate the pass@$k$ of the base model for $k$ values up to where the pass@$k$ does not change when doubling $k$. This means the base LLM cannot solve any more tasks, even with twice as many attempts. The plots demonstrate that SGE achieves higher final training performance than baselines across

all environments, with on average SGE achieving 27% higher relative final success than the next best baseline for that environment. Notably, baselines are unable to surpass the maximum pass@$k$ performance of the base model, confirming the findings from works such as Yue et al. [2025] for agentic settings. However, in Coding and LangR, SGE surpasses the maximum pass@$k$ by on average 11% relative performance, demonstrating that the exploration from SGE enables the model to learn new behaviors not demonstrated by the base model.

SGE also outperforms the EntropyAdv and RND baselines, which also focus on better exploration in LLM RL. We find that while EntropyAdv and RND increase policy token output entropy in RL training, unlike SGE, they cannot explore to exceed the maximum pass@$k$. These methods were developed in the context of question answering in math reasoning problems, while we focus on agentic exploration, where diversity over outcomes is more important than diversity over token outputs. Different output tokens may correspond to actions that produce similar effects, for example, the agent may make only minor variations to the syntax of the code in the Coding environment or tap the same button at different positions in AndroidWorld. SGE also outperforms RLAD, another method that leverages strategies. RLAD is hampered by slow training, potentially due to the KL divergence penalty required in the method. RLAD also only showed results for non-agentic question answering problems for the smaller Qwen3-1.7B model. SGE better leverages strategy generation for exploration in agentic RL. We provide additional comparisons to results reported in prior work, using standard benchmark settings where possible, in Section C. Since training settings differ across papers, these comparisons provide context rather than controlled evidence of superiority.

Next, in Figure 3 we plot the pass@$k$ curve of the trained policy across SGE and baselines. Regular GRPO training "flattens" the pass@$k$ curve, raising the pass@1 value with RL to closer match the pass@k value. On the other hand, SGE benefits from increased test time scaling by increasing in performance with increased $k$ to consistently surpass the base model. This indicates that SGE enables the policy to solve new tasks that are not solvable by the base model, even for very large $k$, where the pass@$k$ has plateaued for the base LLM. This also occurs in AndroidWorld and AppWorld, where the RL pass@1 does not surpass the base model max pass@$k$. In Figure 3, mixed-temperature sampling without RL training ("SGE Zero-Shot") slightly lowers the pass@1 of the base model but increases its pass@$k$ for larger $k$, though it remains well below the RL-trained SGE policy.

**Generalization to unseen tasks**: Next, in Table 1, we evaluate the generalization of the RL-trained policies to unseen task instances using the test splits earlier described in Section 4.1. The results in Table 1 show that SGE trains policies that generalize to new problems better than baselines across the four considered environments. This shows that SGE not only explores the train environments better but also learns generalizable behaviors that transfer to unseen tasks, indicating that the agent is able to utilize its ability to solve harder problems during training time on the unseen test tasks.

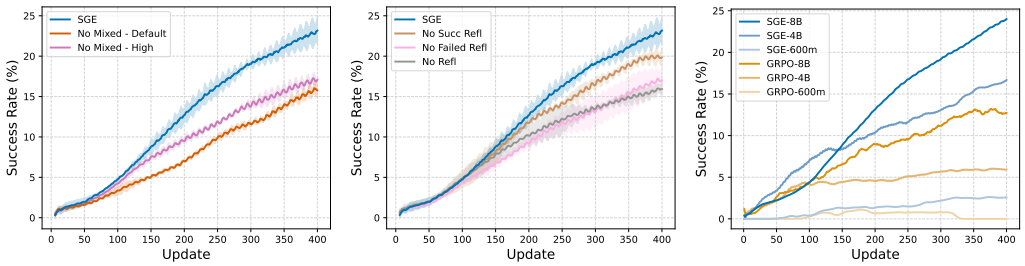
| | Zero-Shot | GRPO | SGE |
|---|---|---|---|
| Coding | 13.5 | 22.0 $\pm$ 0.3 | **29.2** $\pm$ 0.7 |
| AndroidWorld | 16.7 | 21.9 $\pm$ 0.3 | **36.7** $\pm$ 1.3 |
| LangR | 42.6 | 46.0 $\pm$ 0.4 | **60.8** $\pm$ 0.6 |
| AppWorld | 47.8 | 49.3 $\pm$ 5.0 | **66.6** $\pm$ 2.9 |

Table 1: Test evaluation on unseen tasks not in RL training with average and standard error pass@1 across 3 random seeds. All coding checkpoints are evaluated after 1k updates. Zero-Shot evaluates the base-LLM directly on the test set. RL improves generalization, and SGE further boosts it.

### 4.3 Analysis

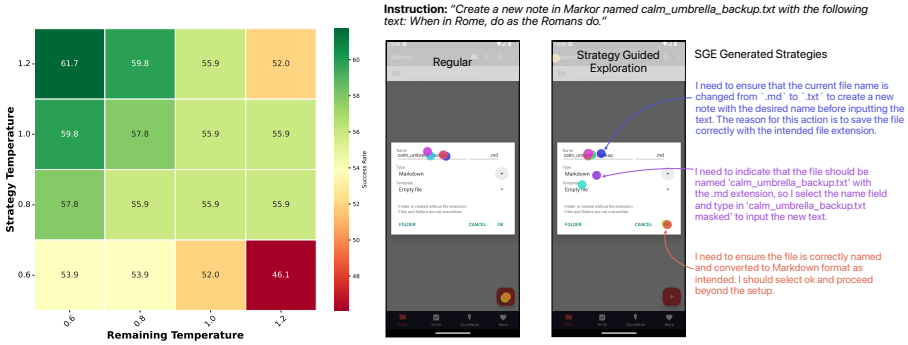
This section further analyzes SGE in a setup that focuses on difficult tasks where the base model struggles. Specifically, we filter the hard category of the coding environment from Section 4.2 to only the problems where pass@256 of the Qwen3-4B model is $0\%$, which leaves $144$ problems. These difficult problems test the limit of the model's ability to explore and find creative solutions. For this section, we finetune the Qwen3-8B model as it has more efficient RL training on these difficult problems for running analyses.

**SGE component ablation**: First, we analyze the contribution of the SGE components in Figure 4. In Figure 4a, we remove the mixed-temperature sampling from SGE and instead sample all tokens with the default action temperature ("No Mixed - Default") or the high temperature used to sample the strategy ("No Mixed - High"). Figure 4a shows that either using the default or higher temperature results in worse performance than the mixed-temperature sampling. Figure 4b shows that the failed strategies are more important in the reflection process than the successful strategies. Both types




(a) Mixed-Temp Ablation (b) Reflection Ablation (c) Scaling Results

Figure 4: Left: parallel strategy sampling ablation by comparing to sampling all tokens with high temperature or default LLM temperature. Middle: sequential strategy reflection ablation by removing SGE's ability to reflect on successful, negative and all strategies. Right: Effect of LLM size on RL training with and without SGE. Ablation results use 3 random seeds, and scaling results use 1 seed.




(a) Pass@16 vs. Temperature (b) Qualitative Example of SGE in AndroidWorld

Figure 5: Left: Comparing the effect of different mixed-temperature sampling settings on the zero-shot pass@16 performance in the Coding environment. The high-values in the top-left indicate that a mixed-temperature sampling produces the best results. Right: Visualizing actions predicted by SGE vs. regular sampling in the *MarkorCreateNote* task in AndroidWorld. 16 actions are sampled for the same observation screenshot with the tap locations visualized as colored circles overlaid on the screen (non-tap actions are not visualized). To complete the task, the agent must tap the file extension dropdown. Unlike with regular sampling, SGE is able to sample this correct action.

of reflection help the overall SGE performance. Mixed-temperature sampling and reflection also improve performance in AndroidWorld, LangR, and AppWorld (Table 3 in Section B).

**Effect of mixed-temperature sampling on the base model:** In Figure 5a, we further analyze the effect of mixed-temperature sampling by comparing different temperature values for the strategy and the remaining tokens when sampling from the base model. Figure 5a shows that a high strategy temperature and a relatively lower temperature on the remaining tokens produce the highest pass@16, and thus the strongest exploration results. For RL training, we primarily care about the exploration capabilities of the model; thus, we use this mixed-temperature setting in SGE.

**Effect of model scaling:** In Figure 4c, we analyze the effect of scaling the base LLM parameter count on RL training with or without SGE by training with 600M, 4B, and 8B Qwen3 model sizes. Expectedly, training greatly improves with larger models, with the positive effects of SGE being most prominent at larger model scales. SGE improves training at the 4B and 8B model scales, however, the smallest 600M model achieves close to $0\%$ success and SGE only barely helps. This indicates that SGE is limited by the reasoning and planning capability in the LLM, and if the base LLM cannot generate or leverage diverse strategies, SGE, like the GRPO baseline, struggles to explore.

**SGE qualitative exploration capabilities:** In Figure 5b, we visualize a qualitative example showing the SGE's exploration capabilities over standard GRPO training in the AndroidWorld *MarkorCreateNote* task. The visualized observation occurs in the middle of the episode where the agent has

already opened the new file dialogue and entered the filename and now must change the file extension from *.md* to *.txt*. The QwenVL-2.5-3B model struggles with this step as it frequently tries to change the extension by typing *.txt* into the filename, while only selecting the other file extension in the dropdown actually changes the extension. The left screenshot of Figure 5b shows that the tap locations of the regular policy sampling result in tapping slightly different locations on the filename extension. However, the right screenshot of Figure 5b shows that SGE policy sampling taps several locations around the file creation dialogue box, and correctly interacts with the file extension dropdown menu. Each of these different taps is driven by a different strategy. While many taps are incorrect, the correct tap is covered, thus increasing the task's pass rate as empirically demonstrated in Figure 5a.

See Section B for additional experimental results on the impact of the strategy prompt, qualitative examples of the strategy generation, and further analyzing exploration in SGE.

## 5 Conclusion and Limitations

This work introduces Strategy-Guided Exploration (SGE), a method for exploration in RL training for LLM agents. SGE uses the textual reasoning abilities of LLMs to produce diverse high-level action strategies and then conditions the action generation on these textual strategies. These strategies provide an effective space for agents to explore diverse outcomes in the environment. SGE uses mixed-temperature sampling and strategy reflection to improve the diversity of generated strategies and thus achieve better exploration to solve challenging tasks. We empirically demonstrate that SGE outperforms prior RL approaches across a variety of agentic domains in coding, UI control, tool-calling, and embodied AI.

A limitation of SGE is that it requires the starting LLM to have sufficient reasoning and planning capacity to generate and leverage the strategies. We design SGE for agentic scenarios where the agent interacts with an external environment, and leave extending SGE for other non-agentic RL scenarios, like math reasoning training, for future work. SGE also introduces the additional cost of having to produce a strategy before every step. This limits real-world deployment where response latency is important. Future work can investigate how to dynamically predict new strategies only when necessary.

## Funding Statement

This work was done while the authors were at Apple and was fully funded by Apple, with no third-party funding.

## References

OpenAI. Computer-using agent: Introducing a universal interface for ai to interact with the digital world. 2025. URL `https://openai.com/index/computer-using-agent`.

Anthropic. Claude code, 2025. URL `https://claude.com/product/claude-code`.

OpenAI. Openai o3 and o4-mini system card. Technical report, OpenAI, 2025. URL `https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf`. Technical report. System Card.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 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.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

Fang Wu, Weihao Xuan, Ximing Lu, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025a.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms. *arXiv preprint arXiv:2504.14655*, 2025a.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Walter Talbott, Katherine Metcalf, Natalie Mackraz, Devon Hjelm, and Alexander Toshev. Large language models as generalizable policies for embodied tasks. *arXiv preprint arXiv:2310.17722*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *NeurIPS*, 2016.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. *arXiv preprint arXiv:1706.08090*, 2017.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. Rt-h: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*, 2025a.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. Optimizing language models for inference time objectives using reinforcement learning. *arXiv preprint arXiv:2503.19595*, 2025.

Jingtong Gao, Ling Pan, Yejing Wang, Rui Zhong, Chi Lu, Qingpeng Cai, Peng Jiang, and Xiangyu Zhao. Navigate the unknown: Enhancing llm reasoning with intrinsic motivation guided exploration. *arXiv preprint arXiv:2505.17621*, 2025.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*, 2025.

Yuda Song, Julia Kempe, and Remi Munos. Outcome-based exploration for llm reasoning. *arXiv preprint arXiv:2509.06941*, 2025.

Tianjian Li, Yiming Zhang, Ping Yu, Swarnadeep Saha, Daniel Khashabi, Jason Weston, Jack Lanchantin, and Tianlu Wang. Jointly reinforcing diversity and quality in language model generations. *arXiv preprint arXiv:2509.02534*, 2025.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

Qianyue Hao, Yiwen Song, Qingmin Liao, Jian Yuan, and Yong Li. Llm-explorer: A plug-in reinforcement learning policy exploration enhancement driven by large language models. *arXiv preprint arXiv:2505.15293*, 2025.

Allison Tam, Neil Rabinowitz, Andrew Lampinen, Nicholas A Roy, Stephanie Chan, DJ Strouse, Jane Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *Advances in neural information processing systems*, 35:25377–25389, 2022.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. Rlad: Training llms to discover abstractions for solving reasoning problems. *arXiv preprint arXiv:2510.02263*, 2025.

Chen Henry Wu, Sachin Goyal, and Aditi Raghunathan. Mode-conditioning unlocks superior test-time scaling. *arXiv preprint arXiv:2512.01127*, 2025b.

Justin Chih-Yao Chen, Becky Xiangyu Peng, Prafulla Kumar Choubey, Kung-Hsiang Huang, Jiaxin Zhang, Mohit Bansal, and Chien-Sheng Wu. Nudging the boundaries of llm reasoning. *arXiv preprint arXiv:2509.25666*, 2025b.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Delta-code: How does rl unlock and transfer new programming algorithms in llms? *arXiv preprint arXiv:2509.21016*, 2025.

Prince Zizhuang Wang and Shuli Jiang. SLEA-RL: Step-level experience augmented reinforcement learning for multi-turn agentic training, 2026. URL `https://arxiv.org/abs/2603.18079`.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle, 2025c. URL `https://arxiv.org/abs/2510.16079`.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL `https://arxiv.org/abs/2602.08234`.

Siyu Xia, Zekun Xu, Jiajun Chai, Wentian Fan, Yan Song, Xiaohan Wang, Guojun Yin, Wei Lin, Haifeng Zhang, and Jun Wang. From experience to strategy: Empowering LLM agents with trainable graph memory, 2025b. URL `https://arxiv.org/abs/2511.07800`.

Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. RetroAgent: From solving to evolving via retrospective dual intrinsic feedback, 2026. URL `https://arxiv.org/abs/2603.08561`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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.

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

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL https://arxiv.org/abs/2502.13923.

Xin-Ye Li, Ren-Biao Liu, Yun-Ji Zhang, Hui Sun, Zheng Xie, and Ming Li. Exploring pass-rate reward in reinforcement learning for code generation. *arXiv preprint arXiv:2605.02944*, 2026.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Jiayu Huang, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Yuxiao Dong, et al. Mobilerl: Online agentic reinforcement learning for mobile gui agents. In *International Conference on Learning Representations*, volume 2026, pages 35282–35315, 2026.

Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents. *arXiv preprint arXiv:2602.05832*, 2026.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. *arXiv preprint arXiv:2509.25047*, 2025.

Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, et al. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2025b.

Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34, 2021.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

Andrew Szot, Bogdan Mazoure, Harsh Agrawal, R Devon Hjelm, Zsolt Kira, and Alexander Toshev. Grounding multimodal large language models in actions. *Advances in Neural Information Processing Systems*, 37: 20198–20224, 2024.

Andrew Szot, Bogdan Mazoure, Omar Attia, Aleksei Timofeev, Harsh Agrawal, Devon Hjelm, Zhe Gan, Zsolt Kira, and Alexander Toshev. From multimodal llms to generalist embodied agents: Methods and lessons. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 10644–10655, 2025.

**Algorithm 1** STRATEGY-GUIDED EXPLORATION pseudocode.

Hyperparameters: Max buffer size $B$, token sampling temp $\tau$, strategy sampling temp $\tau_s$, good strategy sample probability $p_G$, bad strategy sample probability $p_B$, starting state and goal distribution $\rho$
Initialize: policy $\pi_\theta$, success and failure strategy buffers $\mathcal{B}_G^g, \mathcal{B}_B^g = \{\}$ for each training task instance $g$
**for** RL update $i = 1, \ldots, N$ **do**
    Sample starting State $o_1, g_i \sim \rho$
    **for** Parallel environment rollout $k = 1, \ldots, K$ to generate trajectory batch $\{\tau_k\}_{k=1}^K$ **do**
        Set $C_S = \emptyset$, if Unif$(0,1) < p_B$ then $C_S \sim \mathcal{B}_B^{g_i}$, else if Unif$(0,1) < p_G$ then $C_S \sim \mathcal{B}_G^{g_i}$
        **for** Environment steps $t = 1, \ldots, M$ **do**
            Sample strategy: $s_t \sim \pi_\theta(\cdot|g_i, o_t, C_S; \tau_s)$
            Sample remaining tokens: $y_t \sim \pi_\theta(\cdot|s_t, g_i, o_t; \tau)$
            Execute action in environment: $o_t \leftarrow \mathcal{T}(o_t, \text{ExtractAction}(y_t))$
        **end for**
    **end for**
    After all $K$ rollouts finish, add the strategies of each $\tau_k$ to $\mathcal{B}_G^{g_i}$ if the goal was achieved or $\mathcal{B}_B^{g_i}$ otherwise for use in later updates, evicting the oldest entry if the buffer is full.
    Do standard GRPO update with collected $\{\tau_k\}_{k=1}^K$.
**end for**

# A Implementation Details

## A.1 Prompts

SGE requires setting three prompts: (1) the *strategy prompt* which guides the LLM to first generate a strategy before the remaining tokens, (2) the *positive reflection prompt* which conditions the strategy generation on previously executed successful strategies, and (3) the *negative reflection prompt* which conditions the strategy generation on previously executed failed strategies. The prompts we use for the different environments are largely the same. Differences are that the action is referred to differently in the environments as "code", "tool call", or "action". Additionally, the AndroidWorld prompt required formatting the strategy as XML tags to follow the Qwen UI prompt format [Bai et al., 2025], whereas the other environments use markdown formatting. In the reflection prompts, the strategy to reflect on is inserted in `{strats}`.

Coding environment:

- Strategy prompt: `First give a strategy of how to solve the question after ''### Strategy''. Then write the code to solve the question based on the strategy and question in ''### Code''.`
- Positive reflection prompt: `First give a strategy of how to solve the question after ''### Strategy'' inspired by this successful strategy.{strats} Then write the code to solve the question based on the strategy and question in ''### Code''.`
- Negative reflection prompt: `First, after ''### Strategy'' critique the failed strategy and how it can be fixed. Be precise. Then address this critique by writing a better strategy. Make sure the strategy is detailed, and the code is easy to implement from the strategy. {strats} Then write the code to solve the question based on the strategy and question in ''### Code''.`

AndroidWorld environment:

- Strategy prompt: `For each function call, return a strategy in the <strategy></strategy> XML tags and a json object with function name and arguments within <tool_call></tool_call> XML tags.`
- Positive reflection prompt: `Here is a previous strategy that was successful at each step: {strats}`

- Negative reflection prompt: `Here are the previous sequence of strategies that failed: {strats} Your new strategy must be different from this failed strategy sequence and try something new.`

LangR environment:

- Strategy prompt: `First give a strategy of how to solve the question after ‘‘### Strategy’’. Then output the tool call action after ‘‘### Action’’.`
- Positive reflection prompt: `Here is a previous strategy that was successful at each step: {strats} First give a strategy of how to solve the question after ‘‘### Strategy’’ inspired by this successful strategy. Then generate the tool call in ‘‘### Action’’.`
- Negative reflection prompt: `Here is a previous strategy that failed: {strats} First, after ‘‘### Strategy’’ critique the failed strategy and how it can be fixed. Be precise. Then address this critique by writing a better strategy. Make sure the strategy is detailed and the code is easy to implement from the strategy. Then generate the tool call in ‘‘### Action’’.`

AppWorld environment:

- Strategy prompt: `Solve this step by step. First think about what your next step should be. Then write the code to execute that step.`
- Positive reflection prompt: `Here is a previous approach that was successful at each step:\n{strats}\n Inspired by this successful approach, first think about what your next step should be. Then write the code to execute that step.`
- Negative reflection prompt: `Here is a previous approach that failed:\n{strats}\nFirst, think and critique the failed approaches and propose how it can be fixed. Be precise. Then address this critique by writing a better approach. Make sure the approach is detailed and the code is easy to implement from the approach. Once you are done thinking, write the code to execute your next step.`

In AppWorld, we use the reasoning model and take the entire content in `<think>...</think>` as the strategy. We use the non-reasoning mode for LangR and coding, meaning the model does not first output the `<think>...</think>` block. Instead, we extract the strategy from the “Strategy" markdown block part of the prompt described in Section A.1. Note that even without the explicit “think" block, the model still produces textual reasoning and a strategy outside of the action. While we demonstrate SGE works both with and without reasoning mode enabled, we use the non-reasoning mode on LangR and Coding as it is more efficient to train since it requires fewer tokens per action, yet it is still sufficient to reason over strategy generation. Qwen2.5-VL does not have any reasoning model, so we extract the strategy from the prompted `<strategy>...</strategy>` XML tag. Note that all methods and baselines use the strategy prompt for all environments, while SGE with strategy reflection uses the positive and negative reflection prompts.

### A.2 Further SGE Details

See Algorithm 1 for SGE pseudocode. Lines 5-12 show how the policy inference during data collection is augmented with the mixed-temperature sampling and strategy reflection. Otherwise, SGE follows typical GRPO training. We maintain a separate success and failure buffer for each task instance, e.g., for each of the 606 training problems in Coding. The $K$ rollouts are conditioned only on strategies from earlier episodes of the same task, not on strategies from the current $K$ rollouts, since the buffers are updated only after the current rollouts finish. Each buffer is first-in-first-out, so adding a strategy to a full buffer evicts the oldest entry, keeping the most on-policy data.

### A.3 Hyperparameters

Table 2 breaks down the hyperparameters for RL training for each environment. As Table 2 demonstrates, the values are largely the same between the domains. Shared RL hyperparameters were first tuned by running regular GRPO in the environment. Some of the settings relating to the batch size changed in each environment due to the different required context lengths. The number of output

| | Code | AndroidWorld | LangR | AppWorld |
|---|---|---|---|---|
| Learning Rate | 1e-6 | 1e-6 | 1e-6 | 1e-5 |
| Group Size | 16 | 6 | 16 | 8 |
| Global Update Batch Size | 128 | 64 | 128 | 128 |
| # Output Tokens / Step | 2048 | 564 | 520 | 768 |
| Clip Param | 0.2 | 0.2 | 0.2 | 0.2 |
| Trajectories Collected Per Update | 1024 | 192 | 512 | 128 |
| Per-Update Epochs | 2 | 2 | 1 | 2 |
| Token Temperature | 0.7 | 0.7 | 0.6 | 0.7 |
| SGE Strategy Temperature | 1.2 | 1.2 | 1.0 | 1.2 |
| SGE Strategy Buffer Size | 32 | 32 | 32 | 32 |
| SGE Failure Strategy Reflection Prob | 0.25 | 0.25 | 0.25 | 0.25 |
| SGE Success Strategy Reflection Prob | 0.1 | 0.1 | 0.1 | 0.5 |

Table 2: RL hyperparameters across domains. Unless otherwise specified, these values are shared between all approaches. SGE prefix means the value only applies to SGE, and not the baselines. *# Output Tokens / Step* refers to the total token budget per LLM output for each step in the environment. This multiplies the number of steps in an episode by the total number of tokens produced in an episode. The *Global Update Batch Size* is in terms of the number of steps, not the number of trajectories, which consist of multiple steps. *Group Size* is the number of trajectories used to estimate the advantage in GRPO. *Trajectories per update* is the number of samples collected in the environment for each update. *Token Temperature* is the temperature used to generate the non-strategy tokens.

tokens for AndroidWorld and LangR was selected by taking the maximum action length for that environment and adding 512 additional tokens for non-action intermediate outputs for chain-of-thought or strategy generation. We use a constant learning rate throughout training, and do not employ any learning rate scheduler. The RL parameters in the top section of Table 2 are shared between all RL methods. Each RL training job across all environments was run over 16 H100s. The number of updates used to train each method in each environment is displayed in the x-axis of Figure 2. For all the experiments, we train all LLM parameters, including the LLM vocabulary embedding and output layers. For the Qwen2.5-VL experiments, we freeze the visual encoder module and only train the LLM components.

### A.4 Baseline Details

**Entropy Advantage**: This method shapes the per-token advantage term with the token entropy. Specifically, for advantage estimate $A_t$ at output token $o_t$ with output distribution entropy $\mathcal{H}_t$, the advantage term is modified as:

$$A'_t = A_t + \min\left(\alpha \cdot \mathcal{H}_t, \frac{|A_t|}{\kappa}\right)$$

Where $\alpha$ is the scaling coefficient and $\kappa$ controls the clipping threshold. As in Cheng et al. [2025], we set these to $\kappa = 2$ and $\alpha = 0.4$. Note that the entropy $\mathcal{H}_t$ is detached from the computational graph. This shaping ensures that the entropy does not dominate the advantage term or reverse the sign of the advantage.

**RL with Abstraction Discovery (RLAD)**: Unlike the other baselines and SGE, this approach uses a KL divergence term to the starting policy. We set the KL divergence loss coefficient to be $0.001$. We skip conditioning the solution generation on the strategy $25\%$ of the time in RL training. Note that, unlike Qu et al. [2025], we implement RLAD in a single model where the abstraction and solution generation are performed by a single model.

**Random Network Distillation (RND)**: RND adapted for LLM post-training is referred to as "i-MENTOR" in Gao et al. [2025]. To adapt this method to a multi-step decision-making formulation, we use the final token activation for each action in the output sequence for the RND prediction task. Specifically, let $y_{t,1}, \ldots, y_{t,L}$ be the sequence of output final activations, meaning the final hidden state before the LLM token logit projection head, for step $t$ in the environment, and which are then decoded into action $a_t$. This hidden state is passed through a randomly initialized MLP, $\overline{f}$, to form a target $\overline{z} = \overline{f}(y_{t,L})$. This baseline learns an MLP $f$ and is trained to predict the target latent:

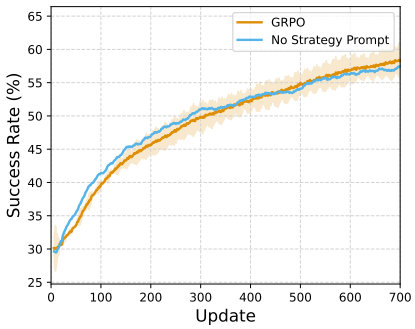



Figure 6: Impact of prompting on RL training in Coding environment. Regular GRPO training is with 3 random seeds. Training without the strategy prompt is only with a single random seed.

$\|f(y_{t,L}) - \overline{f}(y_{t,L})\|_2^2$. The RND reward is calculated based on

$$R^{\text{RND}} = \frac{\|f(y_{t,L}) - \overline{f}(y_{t,L})\|_2^2}{\text{std}\left(\|f(y_{t,L}) - \overline{f}(y_{t,L})\|_2^2\right)}$$

The RND reward is then assigned to actions from trajectories that receive zero reward.

## B Additional Results

**Impact of strategy prompt**: The strategy prompt is a short text asking the LLM to generate a strategy before generating an action. The exact prompts for each environment are detailed in Section A.1. Figure 6 compares standard GRPO training with and without the strategy prompt, demonstrating that the agent performs mostly the same with the strategy prompt and ends with slightly higher performance. In this result, both approaches were allocated the same token budget for the per-action response. From these results, all methods in Section 4 use the strategy prompt for a consistent comparison.

**Impact of SGE on zero-shot pass@$k$**: Figure 10 breaks down the zero-shot performance across a variety of temperature settings in terms of the pass@1 performance. The figure demonstrates that the mixed-temperature negatively impacts the pass@1. However, this lower pass@1 comes with improved pass@16, as earlier shown in Figure 5a, which ultimately leads to better RL.

**Component ablation across environments**: Table 3 extends the SGE component ablation from Figure 4 to AndroidWorld, LangR, and AppWorld. Mixed-temperature sampling and strategy reflection are important across all four environments, although their relative importance varies by environment. In AndroidWorld, mixed-temperature sampling is especially important, which we hypothesize is due to the low-level, tap-based action space. In AppWorld, strategy reflection is more important, which we hypothesize is due to its more expressive code-block action space.

| | SGE No Reflection | SGE No Mixed Sampling | SGE |
|---|---|---|---|
| Coding (Hardest) | 16.0 | 15.7 | **23.2** |
| AndroidWorld | 40.8 | 34.9 | **46.6** |
| LangR | 75.0 | 73.1 | **81.2** |
| AppWorld | 48.8 | 55.0 | **58.2** |

Table 3: Final success rates when removing strategy reflection or mixed-temperature sampling from SGE. Coding (Hardest) results are from Figure 4. AndroidWorld, LangR, and AppWorld models are trained with the same settings as Figure 2 using one random seed.

**Qualitative strategy analysis:** We provide qualitative examples of the strategies SGE produces during RL training in Figure 7 and Figure 8 for the coding environment. Figure 7 shows an example of how SGE generates diverse strategies for the same problem. Figure 8 shows an example of how SGE uses negative strategy reflection to improve the diversity of generated strategies and achieve successful outcomes in the environment.

**State visitation exploration analysis:** We further quantify the exploration capabilities of SGE by analyzing state-visitation of the method and regular GRPO training throughout the course of RL in the coding environment. We do so by measuring the number of distinct outcomes encountered per task during RL training. For the coding environment, we count each distinct set of tests that fails as a distinct outcome. This means we measure the functionality of the program rather than the code itself. Figure 11 shows that SGE encounters more diverse outcomes throughout RL training than regular GRPO. This means that the programs produced by SGE better explore different functionalities in the Python execution environment. This exploratory behavior helps SGE reach the sparse positive outcome more frequently during RL training.

## C Empirical Comparison to Prior Work

We compare SGE with results reported in prior work, using standard benchmark settings where possible for each benchmark. Training settings (data, hyperparameters, base models) differ across papers, so these comparisons provide additional context rather than controlled evidence of superiority. The controlled comparisons in Section 4.2 remain our primary evidence.

**Coding (LeetCode):** We compare to Li et al. [2026] by evaluating on the entire test dataset using the same single-turn evaluation setting. Our main results report only the Hard split, whereas Li et al. [2026] reports the entire test dataset. The average pass@1 on the full test set is 61.1 for SGE, 55.7 for our GRPO baseline, and 53.2 for GRPO as reported in Li et al. [2026]. SGE outperforms the GRPO model from Li et al. [2026], and our GRPO baseline also outperforms theirs. However, the training settings are not identical, as our training uses different hyperparameters and only trains on the Hard training split.

**AndroidWorld:** Direct comparison is difficult because prior work uses substantially more training data and newer models. MobileRL [Xu et al., 2026] uses 100K steps of SFT data and performs RL on all of AndroidWorld and AndroidLab. SGE uses no SFT data and performs RL on only 26 AndroidWorld tasks. UI-Mem [Xiao et al., 2026] uses 256 training queries across AMEX, AndroidLab, and UI-Genie and trains Qwen3-VL, whereas SGE uses Qwen2.5-VL. Despite these differences, we evaluate SGE on the entire AndroidWorld benchmark and obtain an overall success rate of 45.6. The training settings are not directly comparable.

**LangR:** We use the language-only version of the benchmark, whereas prior work reports performance only on the visual version. We therefore do not make a direct comparison with those results.

Since cross-paper training settings differ, we do not claim superiority over the broader agentic RL literature. Our conclusion is limited to the evidence presented: SGE is effective across four diverse domains and outperforms the controlled exploration baselines evaluated under our settings.

## D Response Length Comparison

SGE and GRPO are trained and evaluated with the same token budget per response (Table 2), and GRPO also uses the strategy prompt (Section 4.1). Both methods are also trained with the same iteration over tasks, so each task is trained on equally often. Table 4 reports the average number of generated tokens per response for the SGE and GRPO trained policies, showing that both generate similar numbers of output tokens.

| | **SGE** | **GRPO** |
|---|---|---|
| Coding | 495 | 485 |
| AndroidWorld | 390 | 399 |
| LangR | 387 | 365 |
| AppWorld | 625 | 616 |

Table 4: Average number of generated tokens per response for the SGE and GRPO trained policies.

```
### Question
You are given an integer array nums. The uniqueness array of nums is the sorted array that
      contains the number of distinct elements of all the subarrays of nums. In other words, it is
      a sorted array consisting of distinct(nums[i..j]), for all 0 <= i <= j < nums.length. Return
      the median of the uniqueness array of nums.

<Remainder of problem setup...>
First give a strategy of how to solve the question after "### Strategy". Then write the code to
      solve the question based on the strategy and question in "### Code".

<Strategy 1>
### Strategy

The problem requires finding the median of the "uniqueness array" of an array nums, where the
      uniqueness array contains the number of distinct elements in all possible subarrays.

Let's break down the approach:

1. **Key Insight**:
   - The total number of subarrays is n*(n+1)/2 for an array of length n.
   - The uniqueness array will contain values from 1 (single elements) up to at most n (entire
     array).
   - Since we only need the median (not the full array), we can binary search the answer.
2. **Binary Search on Answer**:
   - Candidate uniqueness values range from 1 to n.
   - For a candidate mid, count how many subarrays have number of distinct elements <= mid.
   - If that count is at least the median position, then the median <= mid; otherwise it is larger.

3. **Counting Subarrays with <= k distinct elements**:
   - Use a sliding window to count subarrays with at most k distinct elements.
   - Maintain a frequency map for the current window.
   - If distinct elements exceed k, move the left pointer until distinct <= k again.
   - Add the number of valid subarrays ending at each right index.
4. **Median Derivation**:
   - total_subarrays = n*(n+1)/2
   - target_rank = (total_subarrays + 1) / 2 (1-indexed)
   - Find the smallest x such that count(distinct <= x) >= target_rank.

### Code
<Implementation reasoning and code following strategy 1...>

<Strategy 2>
### Strategy
1. **Understanding the Problem**:
   - Compute the multiset of distinct-element counts over all subarrays of nums.
   - Return the median of that multiset after sorting.
2. **Key Observations**:
   - There are n*(n+1)/2 subarrays, so enumerating all is infeasible.
   - Instead, count how many subarrays have distinct count <= k for a given k, and use that for
     selection.
3. **Solution Strategy**:
   - Use a sliding window to compute count_at_most(k).
   - Binary search the smallest k such that count_at_most(k) reaches the median rank.
   - Return that k.

### Code
<Implementation reasoning and code following strategy 2...>
```

Figure 7: Two strategies generated by SGE for the same task for the first step in the Coding environment. The text in "<...>" is for visualization and not part of the actual response. Where specified, some of the response is omitted for clarity. The strategies are under `<Strategy 1>`, `<Strategy 2>` and are independently generated and are only sequentially shown for visualization. The two strategies follow different approaches to solving the same task, with strategy 1 directly computing a frequency histogram via enumeration and strategy 2 using binary search and a sliding window. Each strategy is followed by by additional textual reasoning and Python code actually implementing the strategy.

```
### Question
You are given a 0-indexed array heights of positive integers, where heights[i]
     represents the height of the ith building. If a person is in building i, they can
      move to any other building j if and only if i < j and heights[i] < heights[j].
     You are also given another array queries where queries[i] = [ai, bi]. On the ith
     query, Alice is in building ai while Bob is in building bi. Return an array ans
     where ans[i] is the index of the leftmost building where Alice and Bob can meet
     on the ith query. If Alice and Bob cannot move to a common building on query i,
     set ans[i] to -1.

<Remainder of problem setup...>

Here is my previous FAILED strategy:
‘‘‘
<Start of strategy omitted...>
Optimization strategy:
- For any query [ai, bi], we need to determine if there’s a building j where: - j >
     max(ai, bi) (since both need to be able to reach j, j must be after both) -
     height[j] > max(height[ai], height[bi])

We can precompute the next greater element using a stack (monotonic stack) concept to
     quickly find the next element that is greater than a given value, but since we
     need the leftmost such building that is greater than both, we can instead iterate
      from max(ai, bi) + 1 to the end and track the first building with height > max(
     height[ai], height[bi]).
‘‘‘

First, after "### Strategy" critique the failed strategy and how it can be fixed. Be
     precise. Then address this critique by writing a better strategy. Make sure the
     strategy is detailed and the code is easy to implement from the strategy. Then
     write the code to solve the question based on the strategy and question in "###
     Code".

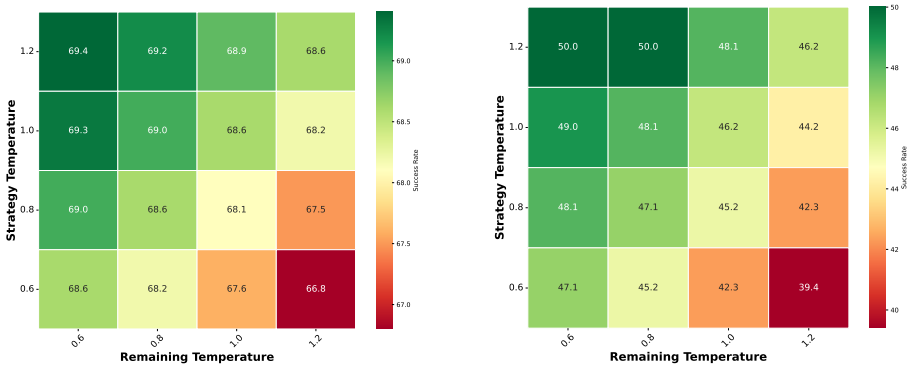
### Strategy
The previous failed approach incorrectly assumes that for a query "[ai, bi]", the
     meeting building must be to the right of both ai and bi and have height greater
     than max(heights[ai], heights[bi]). While this condition is necessary, it is not
     sufficient due to a misunderstanding of the movement constraints.

Critical Insight: The movement rule says that a person from building i can only move
     to building j if i < j and heights[i] < heights[j]. So Bob is at building 4, and
     there’s no building to his right.

<Remainder of strategy omitted>

### Code
<Implementation reasoning and code following corrected strategy>
```

Figure 8: Qualitative example in the Coding environment of how SGE uses negative strategy reflection to improve the diversity of generated strategies. The text in “<...>" is for visualization and not part of the actual response. Where specified, some of the response is omitted for clarity. The agent reflects on a previously executed strategy that resulted in a failed outcome from earlier in RL training with the negative strategy reflection prompt. The original strategy incorrectly missed a corner case by assuming the meeting point must be strictly after both positions. But this misses the case where the rightmost person’s building is itself a valid meeting point. The agent reflects on this strategy, sees the error, and then addresses the shortcomings of the old strategy with a new strategy.




(a) AppWorld (b) AndroidWorld

Figure 9: Zero-shot pass@16 of the base model for different sampling temperatures on the strategy and remaining tokens in AppWorld and AndroidWorld. Green indicates higher performance.

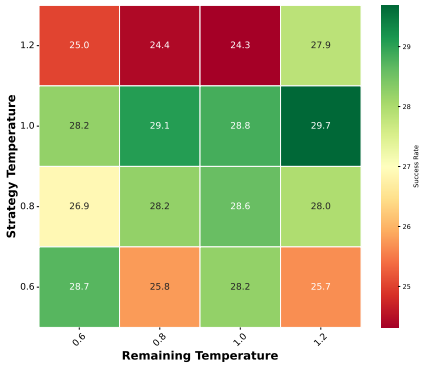



Figure 10: Comparing the effect of the sampling temperature on the strategy and remaining tokens in terms of pass@1. Green indicates higher performance.

# E Mixed-Temperature Sampling Across Environments

Figure 9 extends the zero-shot pass@16 temperature analysis from Figure 5a to AppWorld and AndroidWorld. AppWorld results are computed over 128 evaluations per episode and AndroidWorld results over 64 evaluations per episode. In both environments, a high strategy temperature combined with a lower temperature on the remaining tokens gives the highest pass@16, consistent with the Coding results. The effect is larger in AndroidWorld, where pass@16 ranges from 39.4 to 50.0, than in AppWorld, where it ranges from 66.8 to 69.4.

# F Environment Details

## F.1 Coding

We turn the dataset of coding problems from Xia et al. [2025a] into a multi-turn coding environment. This dataset consists of Python coding problems categorized into “Easy”, “Medium”, and “Hard” difficulties. For all of our training, we only work with the 606 problems from the “Hard” category. We report test performance on the 228 hard category tasks from the test set with 8 independent evaluations per episode.

Each problem consists of a problem description, example input and outputs, constraints on input values, and the starter code. The action space is a Python program, so the agent outputs self-contained Python code to solve the coding problem. This code is run in a new Python instance with the test conditions provided by the problem episode and must finish within 2 seconds before timing out. The reward is $+1$ if all of the tests pass within the time limit and $0$ otherwise. The observation for the

next step is the stack trace of any runtime errors that occurred or any of the unit tests that failed. In a multi-turn fashion, the agent generates a new program based on this observation. For our experiments, all episodes last for 2 steps, meaning the agent has one action to correct errors from the first action.

### F.2 Android World

We use the same action and observation space from the Qwen2.5-VL UI agent results reported in Bai et al. [2025]. The observation space is a $2400 \times 1080$ image. The action space consists of low-level UI control interactions like tapping an $x, y$ coordinate, scrolling, inputting text, a long tap and more. We also use the same Qwen2.5-VL UI agent prompt from Bai et al. [2025]. We insert the strategy prompts immediately after the instruction.

We first remove the visual question answering tasks from the original 115 AndroidWorld tasks Rawles et al. [2024], since we focus on multi-step interactive tasks to achieve a goal rather than training agents to answer questions. We then removed some of the tasks where the Qwen2.5-VL-72B model achieved no success rate. We split the remaining 56 tasks into a train and test set for evaluating SGE and baselines. We use the following 26 tasks for training spanning 11 apps:

- AudioRecorderRecordAudioWithFileName
- SimpleCalendarAddOneEvent
- SimpleCalendarAddOneEventTomorrow
- SimpleCalendarAddRepeatingEvent
- SimpleCalendarDeleteOneEvent
- ContactsAddContact
- FilesDeleteFile
- MarkorAddNoteHeader
- MarkorCreateFolder
- MarkorCreateNote
- MarkorDeleteNote
- MarkorEditNote
- SimpleSmsReply
- SimpleSmsReplyMostRecent
- SimpleSmsSend
- RecipeAddMultipleRecipes
- RecipeAddSingleRecipe
- RecipeDeleteDuplicateRecipes
- RecipeDeleteMultipleRecipes
- RecipeDeleteSingleRecipe
- ExpenseDeleteMultiple
- ExpenseDeleteSingle
- ExpenseAddSingle
- RetroCreatePlaylist
- TurnOffWifiAndTurnOnBluetooth
- SimpleCalendarAddOneEventRelativeDay

We then report the test performance as the average success rate across the following tasks with 4 evaluations per episode.

- AudioRecorderRecordAudio
- SystemBluetoothTurnOff
- SystemBluetoothTurnOn
- SystemWifiTurnOff
- SystemWifiTurnOn
- CameraTakePhoto
- CameraTakeVideo
- ClockStopWatchRunning
- ClockTimerEntry
- ExpenseAddMultiple
- ExpenseDeleteDuplicates2
- ExpenseDeleteMultiple2
- FilesMoveFile

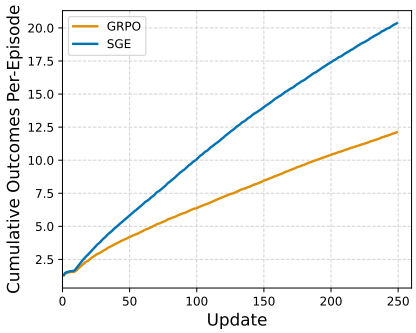



Figure 11: The cumulative unique program outcomes per task in the coding environment during RL training, where each distinct set of tests that passes is considered a distinct outcome. SGE achieves more outcomes faster during RL training compared to standard GRPO, demonstrating its effectiveness in exploration.

- MarkorChangeNoteContent
- MarkorDeleteAllNotes
- MarkorDeleteNewestNote
- MarkorMoveNote
- RecipeDeleteDuplicateRecipes2
- RecipeDeleteDuplicateRecipes3
- RecipeDeleteMultipleRecipesWithNoise
- RecipeDeleteSingleWithRecipeWithNoise
- RetroSavePlaylist
- SimpleCalendarAddOneEventInTwoWeeks
- SimpleCalendarDeleteEvents
- SimpleCalendarDeleteEventsOnRelativeDay
- SimpleCalendarEventsInTimeRange
- SimpleCalendarNextEvent
- SimpleCalendarNextMeetingWithPerson
- SystemBrightnessMax
- SystemBrightnessMin

Notably, the test set includes the following unseen apps: System, Retro, Clock, and Camera. The test set thus includes new tasks in seen apps and new tasks in unseen apps.

To achieve a scalable deployment of AndroidWorld simulations for training and evaluation, we separate the environment from the training or evaluation loops in a dedicated Kubernetes cluster. Each Kubernetes pod is equipped with a multi-threaded HTTP server that defines APIs matching those of the Gym environment interface. A light client-environment wrapper is used by the training or evaluation loop to communicate with the environment server to send actions and retrieve environment state (e.g. screenshots). The network latency is minimal compared to the CPU time taken to render the states and run the Android emulator for every step. Future work can investigate how SGE scales to synthetically generated tasks or hybrid action spaces that include tool-calling in AndroidWorld [Ramrakhya et al., 2025, Yang et al., 2025b].

### F.3 Language Rearrangement (LangR)

Language Rearrangement Szot et al. [2023] is an embodied AI environment using the Habitat simulator Szot et al. [2021], Puig et al. [2023]. We evaluate the test performance on the "New Scenes" split, consisting of 100 testing episodes. We report numbers with 8 independent evaluations per episode. The textual observations take the form "Robot is holding a `<hold-object>` and is at the `<match-receptacle>` with objects `<object1>`, `<object2>`, ...". Where "`<hold-object>`" is the name of the object the robot is holding (such as an "apple"). A receptacle is a static entity that objects can be placed on or within (such as "fridge" or "kitchen table"). The "`<match-receptacle>`" is the receptacle that the robot is within 1.5 meters of and is facing. Finally, "`<object1>`" denotes the names of the objects on the receptacle.

We use the same prompt followed by prior works Szot et al. [2023, 2024, 2025] and add the following text to the end of that prompt to support the textual observations `Action history:{history}\n The user query: {instruction}\n Current robot state: {state}\n`

### F.4 AppWorld

AppWorld [Trivedi et al., 2024] is a simulated app ecosystem with multiple applications and populated synthetic user databases. The agent must output API calls in the form of Python code blocks to fulfill a user instruction, such as using a payments app to send money to friends. The agent is allowed to take multiple steps in the environment, observing the output of the previous code block and executing new code blocks. All code blocks are part of one Python session, meaning that previous variables and definitions can be reused. The action space is thus a Python code block. The observation space is the stdout of the previous code block, which includes any print statements or errors. For our setting, we limit each episode to 16 steps, which we found was sufficient for the Easy split we consider. We use the same prompt from Trivedi et al. [2024], which includes one in-context example and then describes the rules of the AppWorld APIs.

We train on 36 tasks, which are the “Easy" category of tasks from the original training set. We then test on the 57 “Easy" category tasks from the original “Test Normal" dataset with 8 evaluations per episode.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the contributions and scope of SGE from Section 3 and the experimental results in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We include all method and baseline implementation details in Section A and include all evaluation details in Section F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: While we are not to include code with this submission, all our results are on open source benchmarks, we fully describe all implementation details (Section A), and full evaluation details (Section F).

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, we provide these in Section 4.1, Section A, and Section F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our results in Section 4 report standard error in the RL training curves and evaluation results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide these details in Section A.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work deal with fundamental research in exploration in RL and is not tied to immediate societal impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not pose such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses several models and datasets that are properly credited and used according to their license.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Finetuning LLMs with RL is central to the proposed method and overall paper. We describe the usage in detail throughout the paper.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.