REINFORCEMENT LEARNING WITH DECOMPOSED SUBTASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Group Relative Policy Optimization (GRPO) and other reinforcement learning policy-gradient methods for training language model agents collapse an entire multi-turn rollout into a single scalar trajectory reward before that signal enters the policy update. When the underlying task is the composition of distinct skills, and especially when environmental feedback is sparse and delayed, this collapsing is lossy: the optimizer is left to implicitly infer which competency drove the outcome and how that should translate into behavioral change. We argue that the right primitive for these settings is not a better scalar but a decomposition: trajectory reward should be split along subtasks before it ever enters the policy update. We introduce **Reinforcement Learning with Decomposed Subtasks (RLDS)**, a recipe whose core is **Subtask-Decomposed Advantage Estimation (SDAE)**: a replacement for the scalar GRPO advantage that decomposes trajectory reward into per-subtask shares on a fixed subtask taxonomy, computes a group-relative advantage per subtask, and distributes per-token credit by weighting each subtask's advantage by its importance and concentrating it around the step where the reflection identifies that subtask's execution as being consequential to the outcome. We evaluate the resulting recipe on four agentic benchmarks spanning a range of complexity: FrozenLake (sparse grid navigation), HotpotQA (multi-hop QA with a single retrieval tool), ScienceWorld (long-horizon embodied science), and DeepResearch (long-form research with four tools and a composite rubric reward). Per-task heterogeneity diagnostics emitted during training show where decomposition pays off, and the empirical results match: gains scale with subtask heterogeneity, largest on the high-heterogeneity tasks ScienceWorld ($+6.3$ points, 95% CI $[-0.3, +12.8]$) and FrozenLake ($+5.9$ points, $[-0.7, +12.5]$), and within noise on the lower-heterogeneity HotpotQA and DeepResearch, which the diagnostics flagged as having little for decomposition to recover. ScienceWorld is additionally more compute-efficient under RLDS than under scalar GRPO (-10.9% wall-clock per training step), as long rollouts amortize the fixed reflect-and-grade overhead.

1 INTRODUCTION

Consider an agent operating in ScienceWorld (Wang et al., 2022), where the action space is deep and rollouts compose multiple competencies. Two rollouts in the same group both end with reward 0.4. One fails to gather all the items needed to run the experiment; the other gathers the correct items but misinterprets the results. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) subtracts the group mean, hands both rollouts the same advantage, and broadcasts that scalar uniformly across every token – so the second rollout's correctly-gathered tokens get punished for its unrelated interpretation failure. A grader watching either rollout can name the failure in one sentence; the optimizer treats the two trajectories as equivalent. This is the central pathology of scalar reward in compositional RL: a single number per trajectory cannot route credit to the competency that needs it, and a uniform per-token broadcast cannot route credit to the segment that earned it.

This work addresses both facets of the pathology by giving the optimizer a structured diagnosis in place of a scalar. Reflection-augmented training methods (§2) already have the model grade its own rollout in natural language, but compress that grade into a flat trajectory reward before it reaches the optimizer. **Reinforcement Learning with Decomposed Subtasks (RLDS)** keeps the diagnosis

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

structured end-to-end. The reflection phase is asked to grade each rollout along a fixed per-task taxonomy of competencies and to name, for each competency, the pivot step at which its outcome was determined; **Subtask-Decomposed Advantage Estimation (SDAE)** then routes the gradient through that judgment (§3). In place of GRPO's scalar advantage, SDAE computes a group-relative advantage along each of K subtask axes (addressing the across-competency facet, and recovering scalar GRPO when $K{=}1$) and shapes per-token credit around the reflector-identified pivot for each competency (addressing the across-position facet). The per-token gradient now varies both across competencies and across positions in the same rollout. In our motivating example, SDAE would assign a positive per-subtask advantage to the second rollout's object-gathering tokens (which succeeded relative to its group) while assigning a negative advantage to its scientific-reasoning tokens (which failed); scalar GRPO cannot make this distinction.

We evaluate RLDS on four agentic benchmarks (FrozenLake, HotpotQA, ScienceWorld, and DeepResearch) and treat the experiments as a test of a prediction our training-time diagnostics (§5) make: gains should scale with how much the per-subtask signal varies within a group, and with how widely the per-competency pivots spread across the trajectory. These are the two axes of variation that scalar GRPO with uniform per-token broadcast averages away. On the two tasks where RLDS shows a lift, we repeat training three times per condition and report the final checkpoint, ensuring gains do not stem from checkpoint selection alone (§5). On FrozenLake we also check the self-generated grades against ground-truth simulator events (Appendix C).

2 Related Work

RLDS sits at the intersection of reflection-based RL and token/step-level credit assignment. We briefly introduce both lines of work.

2.1 Reflection-Based RL

Self-reflection-based fine-tuning strategies allow the language model to reason using natural language in order to better correct potential mistakes (Shinn et al., 2023; Bensal et al., 2025; Renze & Guven, 2024). Our method draws on two recent reflection-based approaches, ERL (Shi et al., 2026a) and R^{3}L (Shi et al., 2026b).

ERL (Shi et al., 2026a) adds a reflect-then-retry phase inside each PPO (Schulman et al., 2017) step. After an initial rollout, the agent emits a replacement system prompt inside `<improved_prompt>` tags conditioned on the failed trajectory and a cross-episode memory of past rewrite attempts; a second rollout is drawn under that rewritten prompt; both attempts are optimized through the same group-relative objective; and a parallel SFT distillation loss trains the policy to reproduce the successful retry response given the *original* (unrewritten) prompt, internalizing retry-quality behavior into the base policy so the deployed model does not depend on reflection at inference time. We adopt this retry-distillation channel directly as part of our training recipe and do not claim novelty for it.

R^{3}L (Shi et al., 2026b) keeps the original system prompt fixed and replaces the rewrite with a structured JSON reflection report containing a root-cause analysis, an improvement suggestion, and a `retry_from_step` index. The retry resumes from that pivot conditioned on the reflection, and the gradient is shaped through three additional levers: a pivotal credit mask that zeros gradients on the shared prefix between the original attempt and the retry, an OPMD-style baseline that replaces the GRPO (Shao et al., 2024) group baseline with a retry-conditional one, and positive amplification that upweights the advantages of successful retries to prevent failure-dominated entropy collapse. R^{3}L also introduces an auxiliary SFT term that does NLL on the reflection text itself given its conditioning context, complementing ERL's response-side distillation; we adopt this term and discuss it alongside ERL's retry distillation in §3.2. We keep the report, the pivot mask, and the reflection-NLL term, and drop the OPMD baseline and positive amplification.

2.2 Token- and Step-Level Credit Assignment

A separate line of work redistributes credit inside the policy gradient without reflection. GTPO and GRPO-S (Tan et al., 2025) reshape the GRPO advantage by per-token policy entropy, concentrating credit on tokens likely to carry decision-relevant information; GRPO-λ (Parthasarathi et al., 2025)

uses eligibility traces and TD-style propagation to flow terminal reward backward through the trajectory. These methods *distribute* the scalar reward but treat entropy as a proxy for where credit should land, not a diagnosis of what should be reinforced.

A complementary direction trains a verifier for step-level supervision. Process Reward Models (Lightman et al., 2024) score each step of a reasoning trace; AgentPRM (Choudhury, 2025) extends this to agentic rollouts via Monte Carlo lookahead and a lightweight actor–critic update. PRMs factor reward cleanly but require human annotation or a trained verifier, and they factorize along the temporal step axis rather than competency dimensions: distinct failures on the same step, e.g., a wrong tool argument and a wrong reasoning premise, collapse into a single score.

3 Reinforcement Learning with Decomposed Subtasks

We begin by motivating our method in §3.1. We visualize the pipeline end-to-end in Figure 1 and Algorithm 1. We adopt the reflect-retry rollout from R^{3}L (Shi et al., 2026b); §3.2 describes the operational details we rely on, §3.3 defines the three-stage SDAE advantage at the core of RLDS, and §3.4 states the combined objective.

3.1 Motivation

Reflection-based methods generate trajectories that diagnose *what* failed in natural language but feed a scalar reward back to the optimizer. Token- and step-level credit-assignment methods redistribute reward inside a trajectory but treat the trajectory as opaque text, they decide *where* credit should land through entropy heuristics or learned step-level verifiers, not through an explicit decomposition of competencies. The reflection identifies which subtask each part of the trajectory was working on, the pivot localizes the decisive turn, and the retry exposes a counterfactual under which per subtask improvements can be measured. The contribution of this paper is RLDS, the recipe that consumes that signal: SDAE, the advantage estimator at its core, together with the mechanism we define to distribute those rewards across the trajectory.

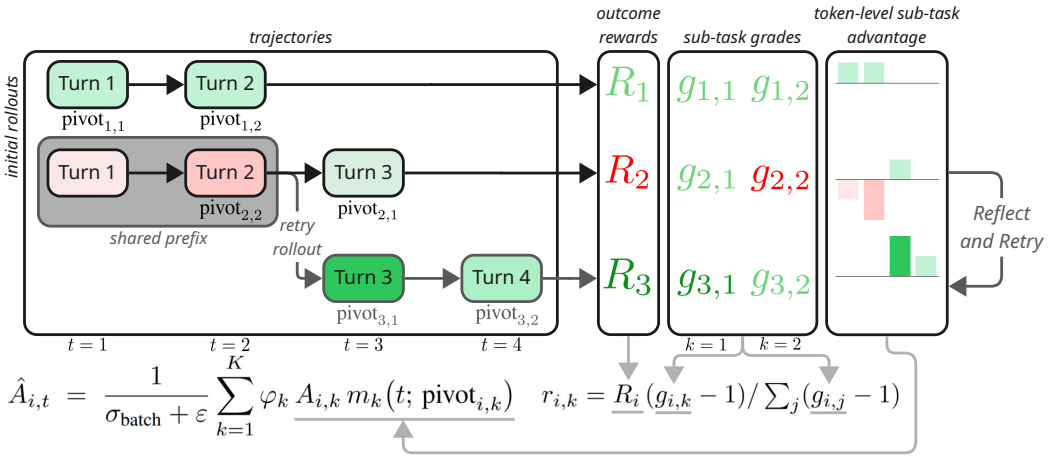

Figure 1: Visualization of SDAE for a single rollout with $G = 2$, $K = 2$ and $\mathcal{R} = 1$. We compute the final token-wise advantage $\hat{A}_{i,t}$ using reward values R from the environment, distributed among subtask grades $g_{i,k}$ from the policy model and credited to tokens using a gaussian kernel around pivot points $\text{pivot}_{i,k}$. Red and green hues indicate the relative values of rewards, subtask grades and token-wise advantage.

3.2 Preliminaries: Subtask Self-Grading on Reflect-Retry Rollouts

Each task is annotated with a fixed decomposition of K named subtasks $\{s_k\}_{k=1}^{K}$ with importance weights $\{\varphi_k\}$ summing to one, held constant throughout training and inference (per-task taxonomies in Appendix D).

Reflect-retry rollout. Each training step, π_θ draws G initial rollouts $\{y_i^{(1)}\}_{i=1}^{G}$ from a task and observes a scalar trajectory reward R_i for each. The same policy then self-grades each rollout under the taxonomy, emitting a per-subtask Likert grade $g_{i,k} \in \{1, \ldots, 5\}$ and a per-subtask pivot step $\text{pivot}_{i,k}$, the turn the grader identifies as the point at which subtask k's outcome was determined. The bottom-$\mathcal{R}$ failures are selected as reflection candidates; for each, π_θ emits a structured JSON reflection report and retries from `retry_from_step` to produce a corrected rollout $y_j^{(2)}$, which is itself self-graded the same way. Originals and retries are folded into a single GRPO group of size $G{+}\mathcal{R}$ and treated uniformly thereafter. The rollout follows R^{3}L (Shi et al., 2026b); the prompts π_θ are conditioned on as grader and reflector are reproduced verbatim in Appendix F.

π_θ is actor, grader, and reflector. The same policy plays all three generative roles here, namely actor (rollouts and retries), grader, and reflector, rather than offloading grading or reflection to an external judge. The auxiliary supervision pairs introduced below supervise π_θ in exactly the grader and reflector roles, so improvements on those skills compound with policy training; keeping every generative role on-policy also minimizes the distribution shift between sampled and trained tokens. The flip side is that the actor and grader share parameters, so we discipline the self-grader by requiring that subtask grades only *redistribute* the trajectory reward R_i across the K axes; they never scale it. By construction, $\sum_k r_{i,k} = R_i$ for every rollout: the environment owns reward magnitude and the grader owns its allocation across competencies, removing the moral hazard a self-grading judge would otherwise introduce.

Auxiliary supervision pairs. Beyond the GRPO group, the same rollout produces two off-policy distillation pairs per successful retry: a (reflection, conditioning context) pair for R^{3}L's reflection-NLL term (Shi et al., 2026b) and an (original prompt, retry response) pair for ERL's retry-distillation term (Shi et al., 2026a); both channels and their motivations are introduced in §2.1. We retain the top-K pairs per step ranked by retry reward delta, weight each pair by its delta so high-impact reflections dominate, and sum the two NLL terms into the auxiliary loss $\mathcal{L}_{\text{aux}}$ that enters the full objective in §3.4.

3.3 Subtask-Decomposed Advantage Estimation

Stage 1. Group-relative per-subtask advantages. We split each trajectory's reward into per-subtask shares using a zero-anchored proportion of the grades, $r_{i,k} = R_i\,(g_{i,k} - 1)/\sum_j (g_{i,j} - 1)$ (uniform $r_{i,k} = R_i/K$ when every $g_{i,j} = 1$); the -1 matches the rubric's grade-1-as-failure semantics so a critically-failed dimension contributes no share before baselining. Within each GRPO group of size $G{+}\mathcal{R}$ we then compute a subtask-specific mean and the per-subtask advantage:

$$\bar{r}_k \;=\; \frac{1}{G+\mathcal{R}} \sum_{i=1}^{G+\mathcal{R}} r_{i,k}, \qquad A_{i,k} \;=\; r_{i,k} - \bar{r}_k. \tag{1}$$

whereas in vanilla GRPO, the scalar baseline subtracts the group mean trajectory reward from each row, so two rollouts with different competency profiles but the same total reward look identical to the optimizer. Baselining each subtask axis separately exposes the dimensions on which a rollout was above or below the group along that axis, even when its trajectory reward is at the mean. The sign of $A_{i,k}$ now records, per subtask, whether the rollout was above or below average on that competency.

Stage 2. Importance-weighted sum and batch normalization. The per-subtask advantages are recombined into a row-level scalar under the static importance weights and normalized at the batch level:

$$A_i \;=\; \sum_{k=1}^{K} \varphi_k\, A_{i,k}, \qquad \hat{A}_i \;=\; \frac{A_i}{\sigma_{\text{batch}} + \varepsilon}, \tag{2}$$

where σ_{batch} is the standard deviation of $\{A_i\}$ across all decomposed rows in the batch. Batch-level normalization (rather than group-level) absorbs scale differences across tasks when training mixes them and keeps the optimizer well-conditioned. Groups for which no per-subtask grades were emitted (e.g., a grader-parse failure) fall back to vanilla GRPO advantages on the trajectory reward, so SDAE degrades gracefully on missing data. We log three diagnostics per step that characterize how much of the multi-dimensional signal SDAE actually recovers: sign-disagreement rate (the fraction of

rollouts whose per-subtask advantages disagree in sign within the group), GRPO correlation (Pearson correlation between the φ-weighted SDAE advantage and the scalar GRPO advantage on the same rollouts), and token-level advantage std (the per-row spread of per-token advantages once Stage 3's pivot kernel is applied). All three are analyzed in §5.1.

Stage 3. Token application. Stage 2 returns a row-level scalar; before it enters the PPO surrogate, we expand it back into a per-token advantage. The default GRPO move is to broadcast $\hat{A}_i$ uniformly over the response tokens of rollout i. SDAE supports that broadcast as a fallback, but when pivot steps are available we instead apply each subtask's advantage through its own per-token kernel centered on the turn the grader marked as decisive:

$$\hat{A}_{i,t} \;=\; \frac{1}{\sigma_{\text{batch}} + \varepsilon} \sum_{k=1}^{K} \varphi_k \, A_{i,k} \, m_k\big(t;\, \text{pivot}_{i,k}\big), \tag{3}$$

where σ_{batch} is the same denominator as in Stage 2 and $m_k(t;\, \text{pivot}_{i,k})$ is a smooth kernel in turn-units centered on $\text{pivot}_{i,k}$, normalized to mean one over the response tokens of rollout i so that the kernel *redistributes* credit across positions without changing the per-row total. The shared σ_{batch} keeps Stage 2's broadcast and Stage 3's per-token expansion on the same scale: averaging $\hat{A}_{i,t}$ over response tokens recovers $\hat{A}_i$. We use a Gaussian kernel in all reported experiments. Pivot masks compose multiplicatively with R^{3}L's global retry-step mask, which zeros gradients on the prefix shared between an original attempt and its retry, so on retry rows the gradient concentrates on the diverging suffix conditioned on the reflection.

3.4 Combined Objective

The full training loss combines a PPO surrogate evaluated against the SDAE per-token advantage with the auxiliary distillation loss $\mathcal{L}_{\text{aux}}$ defined in §3.2:

$$\mathcal{L}_{\text{total}} \;=\; \mathcal{L}_{\text{PPO}}\big(\hat{A}^{\text{SDAE}}_{i,t}\big) \;+\; \mathcal{L}_{\text{aux}}, \tag{4}$$

where $\mathcal{L}_{\text{PPO}}$ is the standard clipped surrogate with a per-token KL penalty against a frozen reference and $\mathcal{L}_{\text{aux}}$ is the auxiliary distillation loss introduced in §3.2, weighted per-pair internally by retry reward delta as described there.

Algorithm 1 RLDS training step.

Require: task x, subtask taxonomy $\{s_k\}_{k=1}^{K}$ with importance weights $\{\varphi_k\}$, group size G, retry budget $\mathcal{R}$, policy π_θ

1: Sample initial rollouts $\{y_i^{(1)}\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid x)$ and observe trajectory rewards $\{R_i\}$
2: Self-grade each $y_i^{(1)}$ via π_θ: per-subtask Likert grade $g_{i,k} \in \{1, \ldots, 5\}$ and pivot step $\text{pivot}_{i,k}$
3: Select bottom-$\mathcal{R}$ failures by reward as reflection candidates
4: For each candidate, emit a structured reflection report and retry from `retry_from_step` to produce $\{y_j^{(2)}\}_{j=1}^{\mathcal{R}}$
5: Self-grade retries; fold originals and retries into one GRPO group of size $G{+}\mathcal{R}$

SDAE advantage:

6: Decompose: $r_{i,k} \leftarrow p_{i,k} R_i$, where $p_{i,k} = (g_{i,k} - 1) / \sum_j (g_{i,j} - 1)$ (uniform $1/K$ if all $g_{i,j} = 1$)
7: **Stage 1.** Group baseline: $A_{i,k} \leftarrow r_{i,k} - \bar{r}_k$, where $\bar{r}_k = \frac{1}{G+\mathcal{R}} \sum_{i=1}^{G+\mathcal{R}} r_{i,k}$
8: **Stage 2.** Importance-weighted sum + batch norm: $A_i \leftarrow \sum_k \varphi_k A_{i,k}$, $\hat{A}_i \leftarrow A_i / (\sigma_{\text{batch}} + \varepsilon)$
9: **Stage 3.** Token application: $\hat{A}_{i,t} \leftarrow \frac{1}{\sigma_{\text{batch}} + \varepsilon} \sum_k \varphi_k A_{i,k} \cdot m_{i,k,t}$, where $m_{i,k,t}$ is a kernel concentrated around $\text{pivot}_{i,k}$

10: PPO/GRPO policy update with $\hat{A}_{i,t}$ and the auxiliary distillation loss $\mathcal{L}_{\text{aux}}$ (single backward pass)

4 Experiments

We evaluate RLDS against scalar GRPO and the reflect-retry baseline (Algorithm 1, lines 1–5, with vanilla GRPO advantages) on four agentic benchmarks chosen to span the subtask heterogeneity

axis. RLVR uses group size G=10; reflection-enabled conditions use G=6 originals plus up to $\mathcal{R}$=3 retries, folded into a combined GRPO group of size $G+\mathcal{R}$ to match the per-step token budget. Subtask taxonomies, importance weights φ_k, reward formulas, and tool specifications are deferred to Appendices D–E.

FrozenLake. Grid-world navigation, no tools, 10 turns, sparse outcome reward. **HotpotQA.** Multi-hop QA with a single `search` tool over Wikipedia, 5 turns, F1-based reward; retrieval follows Search-R1 (Jin et al., 2025). **ScienceWorld** (Wang et al., 2022). Text-based interactive science, 30 procedural tasks, no tools, 30 turns, simulator partial credit. **DeepResearch.** Long-form grounded research with four tools, 10 turns, LLM-judged rubric plus deterministic formatting scores; setup follows DR Tulu (Shao et al., 2025).

5 Analysis

RLDS's gain over GRPO scales with how *process-heavy* and *long-horizon* a task is: the more subtask structure spreads across many turns, the more credit decomposition recovers along the within-group variance and pivot-spread axes §1 predicted gains would track. ScienceWorld coordinates navigation, manipulation, and procedural reasoning over 30-turn rollouts; FrozenLake interleaves hazard avoidance and dynamics inference turn by turn. The per-subtask decomposition recovers heterogeneous signal that GRPO averages flat. HotpotQA and DeepResearch carry minimal process content over short horizons; RLDS figures this out quickly, per-subtask variance reduces, pivots and retries land low-impact, and the SDAE advantage regresses to scalar RLVR. Figure 2 reports validation reward on all four tasks, with the largest gain on ScienceWorld.

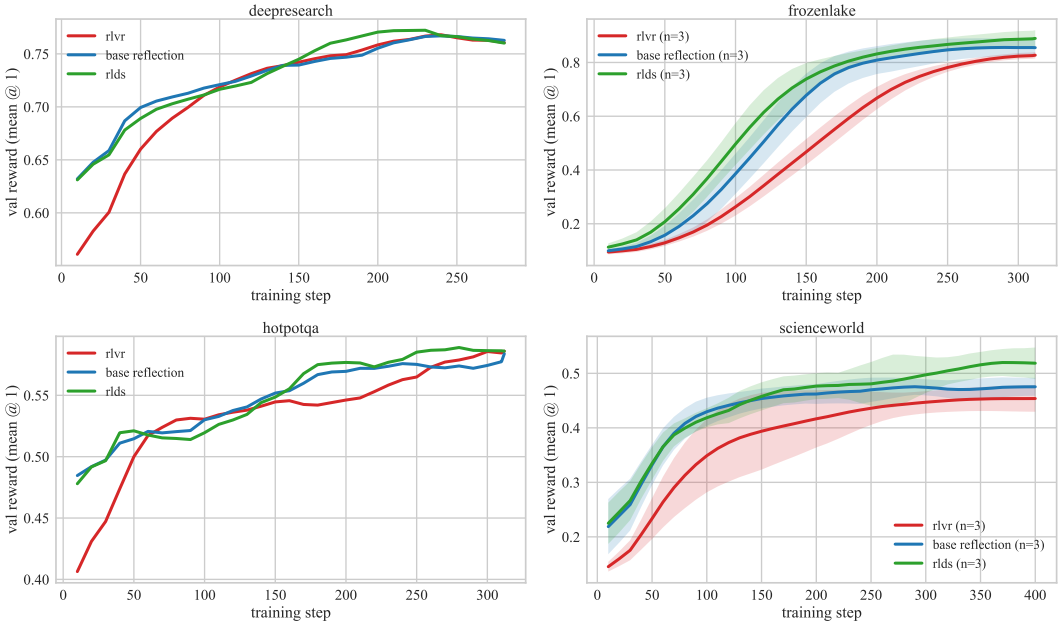

Figure 2: Validation reward over training for scalar GRPO (RLVR), the reflect-retry baseline (BASE REFLECTION), and RLDS. Curves marked n=3 (FrozenLake, ScienceWorld) are the mean of three training runs per condition, with ±1 SD bands across runs. DeepResearch and HotpotQA were run once per condition because they showed no lift.

5.1 SDAE Metrics

SDAE turns the scalar advantage into a per-token tensor through a three-stage pipeline: the rollout has to carry heterogeneous per-subtask signal at all, the φ-weighted decomposition has to then change which rollouts get the strongest gradient, and pivot masking finally has to redistribute that credit across tokens within a rollout. We emit one diagnostic per stage; a task that fails any of them collapses back to scalar broadcast at that point and SDAE has nothing to add downstream.

Stage 1: Is there heterogeneous signal? Sign-disagreement rate. Fraction of GRPO groups whose per-subtask advantages mix signs within a trajectory. A disagreeing row is precisely where scalar GRPO averages flat: the policy did well on competency A and badly on B, the trajectory reward folds them into one number, while RLDS preserves the signs through SDAE so each token gets gradient pushing toward the competency it served. Tasks with persistently high disagreement (Figure 3, ScienceWorld) clear this first gate and leave the most for the later stages to act on; tasks where subtasks move together fail it and SDAE has nothing to recover beyond GRPO's verdict regardless of how the rest of the pipeline is configured. Disagreement also serves as an early warning during training: if it collapses mid-run while reward keeps improving, the policy has converged onto a single-competency strategy and the decomposition is silently degenerating to a scalar.

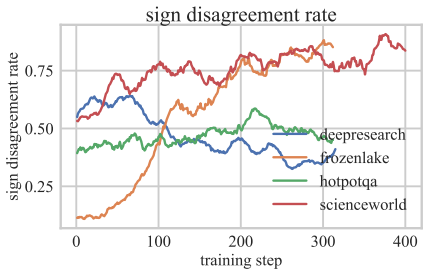

Figure 3: Sign-disagreement rate.

Stage 2: Does decomposition reweight rollouts? GRPO correlation. Pearson correlation between the per-row φ-weighted SDAE advantage (before pivot masking) and the scalar GRPO advantage on the same rollouts. Given that stage 1 has confirmed heterogeneous signal exists, this asks whether the decomposition step survives φ-weighting or cancels back to the scalar. A correlation near 1.0 means the per-subtask signal cancels and the decomposition is informationally redundant at the rollout level; a correlation closer to 0.6 (Figure 4) means RLDS is reweighting which rollouts get the strongest gradient on a non-trivial fraction of the batch.

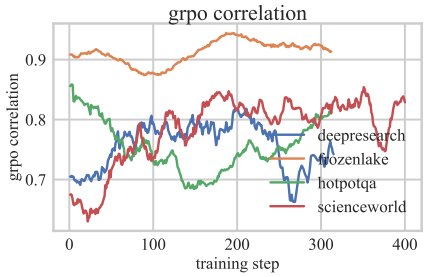

Figure 4: GRPO correlation.

Stage 3: Does pivot masking redistribute credit within a rollout? Token-level advantage std. For each row with pivot data, the standard deviation of per-token advantage values across response-mask-active positions, averaged across the batch. Conditional on stages 1 and 2 having fired, this isolates the spatial-shaping path: scalar broadcast produces zero by construction, and a non-zero value arises only when the per-subtask pivot masks land on different turn windows *and* the per-subtask advantages differ in sign or magnitude. Figure 5 confirms the path is active on the heterogeneous tasks; on HotpotQA, tokens near the late-trajectory `answer_grounding` pivot get visibly different credit than tokens near the early-trajectory `search_planning` pivot.

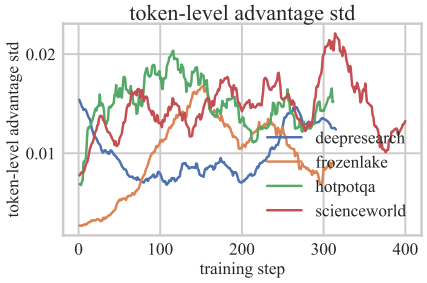

Figure 5: Token-level advantage std.

The cascade lines up with the reward lifts §5.2 reports: ScienceWorld clears all three gates and posts the largest gain, while HotpotQA and DeepResearch wash out at stage 1 over their short, process-light horizons and the SDAE advantage regresses toward scalar RLVR — which is the behavior we want when the rollout is not producing competency-tagged signal worth decomposing.

5.2 Compute Profile

Each RLDS step samples G originals, prompts the policy to grade and reflect on the bottom-reward ones, samples up to $\mathcal{R}$ retries, and grades each retry per-subtask — a fixed per-step overhead on top of the rollout. The interesting question is how this overhead scales with horizon, because the same machinery that costs the most on short tasks pays for itself on long ones.

Compute inverts at long horizons. Short-horizon step time roughly doubles ($2.4\times$/$2.3\times$/$1.6\times$ on FrozenLake/HotpotQA/DeepResearch), driven by reflection and grading. At long horizons retries resample only the suffix from the pivot (retry-to-original gen ratio 0.76/0.48/0.22 on FrozenLake/HotpotQA/ScienceWorld) and the reflection-plus-grading overhead shrinks once each rollout spans $\sim$4 K tokens over $\sim$33 turns; ScienceWorld lands at $0.89\times$ step-time despite strictly more work per trajectory. Reward gains are reported at fixed step count; a budget-matched comparison is left to future work.

Table 1: Per-task compute and reward lift. *Calls/ep*: measured model calls per RLVR episode. *Gen*: rollout time (G originals for RLVR; plus $\mathcal{R}$ retries for RLDS). *Judge*: per-subtask grading time (RLDS only). *Step*: training-step wall-clock. Δ *Step* $= (t_{\text{RLDS}} - t_{\text{RLVR}})/t_{\text{RLVR}}$, negative if RLDS is faster.

		Gen (s)		Judge (s)	Step (s)			
Task	Calls/ep	RLVR	RLDS	RLDS	RLVR	RLDS	Δ Step	Lift
HotpotQA	3.9	28.1	64.5	25.8	69.5	157.0	$+125.9\%$	–
DeepResearch	4.6	118.0	170.3	24.3	163.4	267.3	$+63.6\%$	–
FrozenLake	3.4	27.9	41.8	22.1	49.8	119.0	$+139.0\%$	$+5.9$ $[-0.7, +12.5]$
ScienceWorld	16.1	221.5	170.3	25.3	294.4	262.2	$\mathbf{-10.9\%}$	$+6.3$ $[-0.3, +12.8]$

5.3 Subtask Metrics

The step-level diagnostics in §5.1 compress every subtask into one number per step. They tell us *whether* RLDS recovers multi-competency signal but not *which* subtask is responsible. We expose two per-subtask readouts: **retry grade deltas** (mean change in grader score from the failed original to its reflect-retry) and **per-subtask pivot step** (average turn where the pivot mask lands). They answer the questions: did reflection help *this* competency and where does it sit in the trajectory. DeepResearch and HotpotQA analyses are available in the appendix.

FrozenLake. Figure 6 shows retry grade deltas are led by `hazard_identification` (near $+1.8$) and `path_planning`. `dead_end_detection` picks up sharply at the end of training as the remaining failure cases are tricky maps where the agent could get stuck. The pivot step means intuitively place the `dead_end_detection` deeper in the trajectory, while `hazard_identification` and `path_planning` are activities that happen towards the beginning.

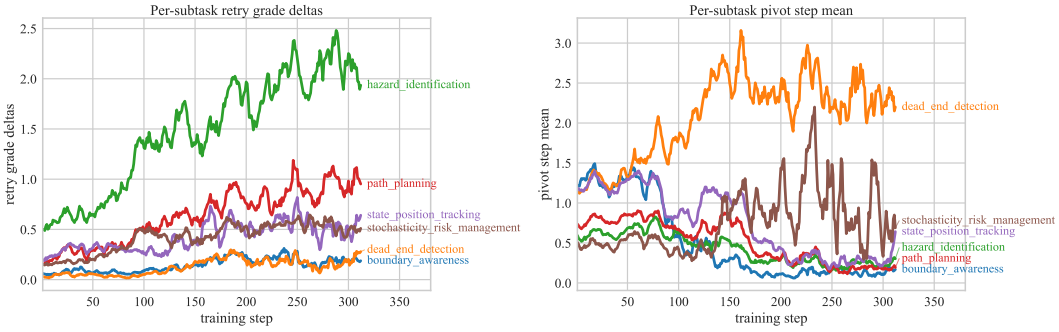

Figure 6: FrozenLake per-subtask metrics over training: retry grade deltas and pivot step.

ScienceWorld. ScienceWorld is the long-horizon stress test (30 turns, six subtasks) that demonstrates how subtask importance can shift over time. Figure 7 shows that around step 200 the model has figured out how to interact with the environment and what becomes more important is the `scientific_reasoning` competency. Once objects are collected this is the subtask that allows the model to compose experiments to answer the scientific questions at hand. Pivot steps fan across the 30-turn budget: `goal_decomposition`

at $\approx$ 2, `exploration_efficiency` and `object_identification` mid ($\approx$ 5), and `action_sequencing`, `scientific_reasoning`, `state_tracking` late ($\approx$ 11–13). The temporal range is the widest of any task.

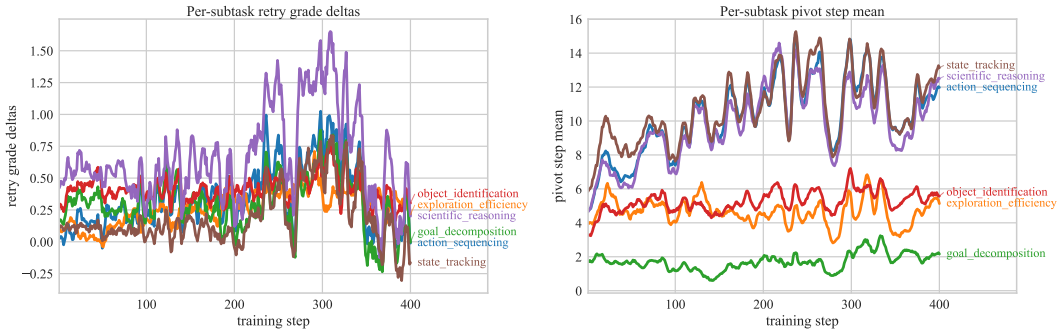

Figure 7: ScienceWorld per-subtask metrics over training: retry grade deltas and pivot step.

6 Limitations

Defining subtasks. The taxonomy is fixed per task and must be defined for each new environment. Aligning subtasks with *temporal stages* of the trajectory (e.g., HotpotQA's `search_planning` $\rightarrow$ `search_iteration` $\rightarrow$ `evidence_extraction` $\rightarrow$ `answer_grounding`) rather than orthogonal competencies ("reasoning" vs. "tool use" vs. "self-correction") proved useful; the latter produces graders whose pivot steps collapse onto the same turn, leaving the SDAE advantage doing scalar broadcast in disguise.

Zero-reward failures. Because subtask shares redistribute the trajectory reward, a rollout with $R_i{=}0$ gets $r_{i,k}{=}0$ on every subtask whatever its grades, and all failed rollouts in a group receive the same row-level advantage. This is deliberate: in early experiments where grades also set reward magnitude, the policy learned to inflate its own grades. The cost is that the direct update cannot credit a correctly executed early subtask in a failed rollout. This limitation could be overcome by using a separate model as the grader, but we leave this for future work.

7 Conclusion

Reflect-retry rollouts already produce a rich diagnosis: a competency-tagged judgment of what failed and which turn determined each failure. Scalar GRPO collapses that diagnosis into a single number per trajectory and forces the optimizer to average credit over competencies it cannot tell apart and across positions where they never fired. RLDS consumes it through the SDAE advantage: group-relative per-subtask baselines preserve the directional disagreements scalar GRPO would average flat, and per-subtask pivot masks reshape credit across response tokens around the turn the grader marked decisive. The prediction that fell out of this construction, that gains should track within-group per-subtask variance and pivot spread, is what the four benchmarks were chosen to probe. ScienceWorld carries within-group sign disagreement throughout training and fans its per-competency pivots from turn ≈ 2 to ≈ 13 across a 30-turn budget; it posts the largest gain ($+6.3$ points of final-checkpoint reward over scalar GRPO, averaged over three runs) and costs $0.89\times$ a scalar GRPO step because retries resample only the suffix from the pivot. FrozenLake carries the same heterogeneity at a shorter horizon and gains $+5.9$ points. HotpotQA and DeepResearch fail the within-group variance gate over their short, process-light horizons; SDAE regresses toward scalar RLVR by construction, and their single-run lifts sit in noise. The SDAE diagnostics (§5.1) locate where each task lives on this gradient, and the per-subtask retry deltas and pivot steps (§5.3) give the practitioner a continuous read on training. Subtask decomposition is the right primitive for multi-competency RL because the rollout produces more information than the scalar can carry. When the rollout already produces competency-tagged signal, the advantage estimator should consume it.

AI USE STATEMENT

LLM-based assistants were used in the research to write and debug training, evaluation and analysis code, and to analyze training logs and trajectories for the statistics we report. In preparing the paper, they were used to search for and check related work, to edit and polish author-written text for readability, and to write plotting code. We did not use generative AI tools to draft sections of the paper, to generate synthetic datasets, or to formulate or prove mathematical claims. The authors reviewed every claim, equation, citation and number in the paper and take responsibility for its content.

REPRODUCIBILITY STATEMENT

§3 and Algorithm 1 specify the SDAE advantage and the full training step. Appendix B gives the shared training recipe, per-task hyperparameters, and run statistics (Table 2), including the pivot-kernel settings; Appendix D lists every subtask taxonomy with its verbatim grader descriptions and importance weights; Appendix E describes each benchmark's action space, observations, reward, termination, and data sources; and Appendix F reproduces the grading and reflection prompts verbatim. Code and configuration files for all methods and benchmarks are included in the supplementary material.

REFERENCES

Shelly Bensal, Umar Jamil, Christopher Bryant, Melisa Russak, Kiran Kamble, Dmytro Mozolevskyi, Muayad Ali, and Waseem AlShikh. Reflect, retry, reward: Self-improving LLMs via reinforcement learning. *arXiv preprint arXiv:2505.24726*, 2025.

Sanjiban Choudhury. Process reward models for LLM agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Conference on Language Modeling*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Prasanna Parthasarathi, Mathieu Reymond, Boxing Chen, Yufei Cui, and Sarath Chandar. GRPO-λ: Credit assignment improves LLM reasoning. *arXiv preprint arXiv:2510.00194*, 2025.

Matthew Renze and Erhan Guven. Self-reflection in large language model agents: Effects on problem-solving performance. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 516–525. IEEE, 2024.

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

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen-tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. DR Tulu: Reinforcement learning with evolving rubrics for deep research. *arXiv preprint arXiv:2511.19399*, 2025.

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

Taiwei Shi, Sihao Chen, Bowen Jiang, Linxin Song, Longqi Yang, and Jieyu Zhao. Experiential reinforcement learning. *arXiv preprint arXiv:2602.13949*, 2026a.

Weijie Shi, Yanxi Chen, Zexi Li, Xuchen Pan, Yuchang Sun, Jiajie Xu, Xiaofang Zhou, and Yaliang Li. R^{3}L: Reflect-then-retry reinforcement learning with language-guided exploration, pivotal credit, and positive amplification. *arXiv preprint arXiv:2601.03715*, 2026b.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Hongze Tan, Zihan Wang, Jianfei Pan, Jinghao Lin, Hao Wang, Yifan Wu, Tao Chen, Zhihang Zheng, Zhihao Tang, and Haihua Yang. GTPO and GRPO-S: Token and sequence-level reward shaping with policy entropy. *arXiv preprint arXiv:2508.04349*, 2025.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

A 5.3 Subtask Metrics Continued

DeepResearch. DeepResearch's five-skill taxonomy (`retrieval_strategy`, `analytical_reasoning`, `factual_accuracy`, `completeness_coverage`, `citation_practice`) maps onto the natural temporal stages of a research trajectory, and Figure 8 shows the pivot steps separating accordingly within the 10-turn budget: `retrieval_strategy` sits earliest at $\approx$ 1.5, `analytical_reasoning` and `factual_accuracy` in the middle at $\approx$ 2.5–3, and `completeness_coverage` and `citation_practice` latest at $\approx$ 3.5–4 — search first, synthesize and state facts in the middle, cover and cite at the end. Retry grade deltas compress after the first $\sim$50 steps and then climb sharply past step 200, led by `factual_accuracy` (peaks $\approx$ 2.0); `citation_practice` hovers near zero throughout, consistent with the deterministic formatting score already disciplining that channel and leaving little for reflection to add.

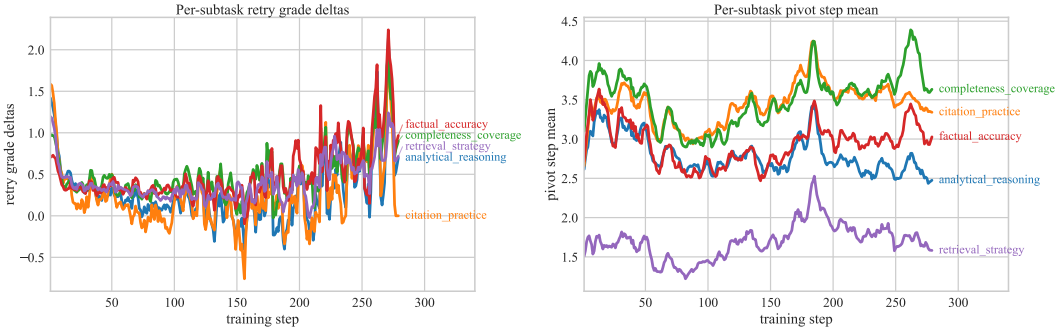

Figure 8: DeepResearch per-subtask metrics over training: retry grade deltas and pivot step.

HotpotQA. Figure 9 visualizes HotpotQA's four-skill taxonomy (`search_planning`, `search_iteration`, `evidence_extraction`, `answer_grounding`) and gives the cleanest per subtask picture in the suite from a temporal perspective. The retry grade deltas fade quickly and settle into a small positive gain. This also maps to initial over-performance on the validation reward as shown in Figure 2 that fades by step 50.

B Training Runs

Shared training recipe. All runs fine-tune Qwen3-4B-Instruct-2507 with GRPO via verl + SGLang. The optimizer is AdamW with a constant learning rate of 1×10^{-6}, fixed KL coefficient 0.001 against a frozen reference, PPO clip ratio 0.28, one PPO epoch per step, and entropy coefficient 0. Each run does one epoch over the task's train split with validation every 10 update steps. Lifts in Table 1 use

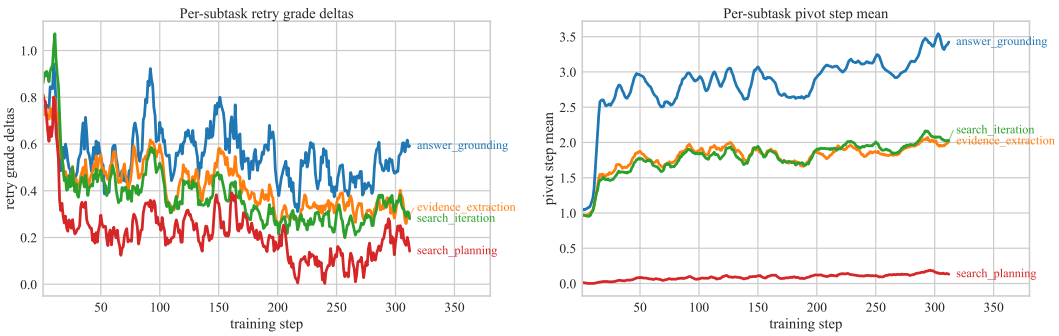

Figure 9: HotpotQA per-subtask metrics over training: retry grade deltas and pivot step.

the final checkpoint, averaged over three runs. The actor uses FSDP2 with the SGLang rollout engine and the verl multi-turn agent loop in Hermes tool-call format. Each run uses one node with 8×H100 80 GB.

Per-task hyperparameters and run statistics. Table 2 reports, for each task, the dataset sizes, the runtime knobs that differ from the shared recipe (max prompt/response, max turns, temperature, group size, batch size), the total number of update steps, the median wall-clock per step and per run, the hardware, and the volume of external-API traffic that the rollout loop generates. Step times and step counts are pulled directly from MLflow (`perf/time_per_step`); all other rows are pinned configuration. Tool-call and judge-call volumes are reported per rollout (mean) and totalled per RLDS run; only DeepResearch has paid external dependencies.

Table 2: Per-task training run statistics. Slash-separated cells report *RLVR / RLDS* where the two conditions diverge. *Tool calls / rollout* are MLflow-logged per-rollout means at training-time temperature; *Total tool calls / run* aggregate originals + retries across the full RLDS run. Cache hit rates on Serper-backed tools sit at 1–5% during training (the cache is most useful across seeds, not within a single run). Dollar costs are deliberately omitted because they depend on Serper / OpenAI contract pricing tier; raw call volumes are reported instead so cost can be computed against the reader's own pricing.

	FrozenLake	HotpotQA	ScienceWorld	DeepResearch
Data				
Train examples	20,000	20,000	1,796	6,480
Validation examples	500	500	887	341
Optimization				
Train batch size	64	64	64	32
Group size G (RLVR / RLDS)	10 / 6	10 / 6	10 / 6	8 / 6
Max episode turns	10	5	30	10
Max assistant / total (tokens)	1k / 4k	1k / 4k	8k / 8k	2k / 16k
Sampling temperature	0.7	0.7	1.0	1.0
Train epochs	1	1	20	2
Compute				
Total update steps	312	312	400	315
Wall-clock / step (RLVR / RLDS, s)	49.8 / 119.0	69.5 / 157.0	294.4 / 262.2	163.4 / 267.3
Total wall-clock (RLVR / RLDS, h)	4.3 / 10.3	6.0 / 13.6	32.7 / 29.1	12.9 / 23.4
Hardware		8×H100 80 GB		
External APIs (RLDS run)				
Tool calls / rollout (mean)	0 (no tools)	∼5 (local)	0 (no tools)	∼3.9
Total tool calls / run	0	∼300k (local FAISS)	0	∼260k (Serper)
Total judge calls / run	0	0	0	∼50–100k (`gpt-5.4-nano`)

ScienceWorld's train split is small (1,796 examples), so the run is configured for 20 epochs to give the policy enough update steps on the long-horizon rollouts; the high parametric-variation count per task (every example draws a fresh JVM seed) keeps later epochs from being pure replay. FrozenLake and HotpotQA see a single pass over 20k examples at batch 64 (= 312 steps); DeepResearch runs

2 epochs over 6,480 examples at batch 32 and stops at the 315-step wall-clock cap. The compute-profile reported in Table 1 uses these same MLflow numbers; the wall-clock totals here are step time multiplied by step count and exclude data-prep, validation-only steps, and warmup.

R3L / RLDS knobs. For the reflection-augmented conditions (BASE REFLECTION, RLDS) we use the same R3L recipe across tasks (`rlef/config/rlef_grpo.yaml`). Reflection is triggered on the bottom 50% of failures per group; the policy itself acts as both grader and reflector (no external judge model), grading on a 1–5 Likert with an outcome-aware `pivot_step`. Auxiliary SFT is enabled with reflection coefficient 0.05 and retry-distillation coefficient 0.10, capped at 64 reflect pairs and 64 retry pairs per step. RLDS additionally enables the per-subtask pivot path with a Gaussian kernel (σ=1 turn on FrozenLake, HotpotQA, and DeepResearch; σ=3 turns on ScienceWorld), rescaled to mean one over the row's response tokens; the legacy OPMD baseline and positive amplification are disabled.

C Validating Subtask Grades Against Simulator Events

The subtask grades and pivots are generated by the policy itself, so we check them against ground truth where the environment provides it. FrozenLake is deterministic in our setup (`is_slippery=False`) and each logged trace stores the board, so every episode can be replayed exactly. We score all 4,563 reflected rollouts from the original FrozenLake RLDS run (31 checkpoints, 312 steps) against five event markers. Four are terminal conditions read from the environment's own messages: falling into a hole, hitting the step limit, hitting the grid boundary, and emitting an invalid action. The fifth, *backtrack region*, is computed by breadth-first search: the agent entered a cell whose only distance-reducing move leads back to the cell it just left, or moved strictly farther from the goal than its own best distance so far.

The run's taxonomy has six subtasks. We leave `stochasticity_risk_management` out of this analysis because it does not map cleanly to any environment signal (the grid is not slippery, so there is no event that marks a stochasticity failure), and report the other five. For each of these subtasks and each event, Table 3 reports the difference in mean grade between episodes where the event fired and episodes where it did not (grade points on the 1–5 scale; Welch's t-test, two-sided, Bonferroni-corrected over all 30 subtask–event tests, including the excluded subtask). A rank-based test agrees on 28 of those 30 tests. Every reflected rollout has reward exactly 0, so no difference in the table can come from reward.

Table 3: Change in mean subtask grade when a ground-truth event fires, FrozenLake. Bold: each subtask's own failure mode. $^{***}p<0.001$, $^{**}p<0.01$, $^{*}p<0.05$ after Bonferroni correction.

Subtask (φ_k)	fell in hole	max steps	hit boundary	invalid action	backtrack region
`path_planning` (0.30)	$+0.13^{***}$	-0.05	$+0.00$	-0.32^{***}	$+0.02$
`hazard_identification` (0.20)	$\mathbf{-0.73^{***}}$	$+0.28^{*}$	$+0.59^{***}$	$+0.54^{***}$	$+0.28^{***}$
`state_position_tracking` (0.15)	$+0.37^{***}$	$+0.44^{***}$	-0.84^{***}	$\mathbf{-1.28^{***}}$	-0.05
`dead_end_detection` (0.10)	$+0.49^{***}$	-0.09	-0.27^{***}	-0.59^{***}	$\mathbf{-0.33^{***}}$
`boundary_awareness` (0.10)	$+0.43^{***}$	$+0.46^{***}$	$\mathbf{-1.55^{***}}$	-1.15^{***}	-0.12^{**}
episodes with event	2,542	298	535	361	2,250

Four of the five subtasks grade markedly lower on the failure they are meant to detect, and in each case that is the largest drop in the event's column: `boundary_awareness` falls 1.55 points on boundary hits, `state_position_tracking` 1.28 on invalid actions, `hazard_identification` 0.73 on holes, and `dead_end_detection` 0.33 on backtrack regions. Grades also *rise* when a different failure fires, i.e., the grader marks the subtask as not implicated. `path_planning` is the exception: pooled over training it shows no relationship with backtrack regions ($+0.02$, n.s.), though it falls 0.32 points ($p<0.001$) in the second half of training, when planning failures dominate and holes have largely stopped occurring.

Retry steps. For each failed rollout we also compute an oracle decisive step by replay: the earliest action that provably changed the outcome (a move off the grid, a step into a hole, or a move that increased the breadth-first distance to the goal). The reflector's `retry_from_step` lands within one step of the oracle in 80% of failures (n=4,233; mean error 0.80 steps), against 72% for shuffled

retry steps (observed values permuted across episodes) and 72% for random ones (drawn uniformly from each episode's own steps). Always predicting step 0 reaches 86%, because the oracle step is step 0 in 66% of failures, so the reflector is not a precise localizer. This is one reason SDAE spreads credit with a Gaussian kernel rather than placing it on a single turn; the observed error is within the kernel's bandwidth. This check scores the reflector's retry step, not the per-subtask pivots that Stage 3 consumes.

D SUBTASK DECOMPOSITIONS

This section lists the subtask taxonomy used by the grader for each task: subtask name, the description shown to the grading model verbatim, and the static importance weight φ_k used by SDAE. Weights sum to 1.0 per task. Names are exactly the strings the grader returns in its JSON output and the strings under which we log per-subtask metrics. The taxonomies live in `rlef/tasks/<task>/constants.py` as `SUBTASKS`.

D.1 FROZENLAKE

Six subtasks over a 10-turn budget. Subtasks split spatial reasoning (`path_planning`, `boundary_awareness`) from hazard reasoning (`hazard_identification`, `dead_end_detection`, `stochasticity_risk_management`) and from execution-state tracking, so the grader can pin the decisive turn separately for each. Our boards are not slippery (`is_slippery=False`).

Table 4: FrozenLake subtask taxonomy.

Subtask	Description (shown to grader)	φ_k
`path_planning`	Finding a safe route to the goal.	0.30
`hazard_identification`	Recognizing which tiles are holes.	0.20
`state_position_tracking`	Maintaining accurate knowledge of current position across steps.	0.15
`stochasticity_risk_management`	Planning paths robust to unintended moves on slippery ice.	0.15
`dead_end_detection`	Identifying configurations where the chosen path will lead to no escape.	0.10
`boundary_awareness`	Avoiding invalid moves at grid edges.	0.10

D.2 HOTPOTQA

Four subtasks over a 5-turn budget, mapped to the canonical multi-hop pipeline (*plan* $\rightarrow$ *retrieve* $\rightarrow$ *extract* $\rightarrow$ *ground*). The temporal ordering of pivot steps in Figure 9 matches this pipeline: `search_planning` pins early, `answer_grounding` pins late.

D.3 SCIENCEWORLD

Six subtasks over a 30-turn budget — the longest horizon in the suite and the task with the widest spread of pivot-step means. Subtasks split early-trajectory planning (`goal_decomposition`, `exploration_efficiency`) from mid-trajectory enactment (`object_identification`, `action_sequencing`) from late-trajectory verification (`state_tracking`, `scientific_reasoning`).

D.4 DEEPRESEARCH

Five subtasks over a 10-turn budget. Subtasks separate the search loop (`retrieval_strategy`) from the content of the report (`factual_accuracy`, `analytical_reasoning`, `completeness_coverage`) and from its sourcing (`citation_practice`). The temporal stages emerge in Figure 8.

Table 5: HotpotQA subtask taxonomy.

Subtask	Description (shown to grader)	φ_k
`search_planning`	Decomposing the multi-hop question and choosing what to look up first; turning each sub-question into a focused initial query rather than passing the question verbatim. Load-bearing in the first tool-call turn.	0.25
`evidence_extraction`	Pulling the right facts out of retrieved passages; identifying which passage actually contains the bridge entity or answer span and not over-trusting the first plausible result.	0.30
`search_iteration`	Adapting the search plan in response to what came back; feeding bridge entities into the next query, broadening when too little is returned, narrowing when too much, knowing when to stop.	0.30
`answer_grounding`	Tying the boxed answer to specific retrieved evidence; not falling back on prior knowledge when retrieval came back empty or off-topic.	0.15

Table 6: ScienceWorld subtask taxonomy.

Subtask	Description (shown to grader)	φ_k
`goal_decomposition`	Breaking the experiment into sequential subgoals.	0.20
`object_identification`	Identifying which objects are needed and where to find them.	0.20
`action_sequencing`	Performing actions in the correct causal order.	0.20
`state_tracking`	Monitoring changes in object properties and environment state.	0.15
`scientific_reasoning`	Applying correct scientific principles to the experiment.	0.15
`exploration_efficiency`	Navigating and exploring the environment without wasted steps.	0.10

E BENCHMARK DETAILS

Per-task action spaces, observation formats, reward functions, and termination conditions referenced in §4. FrozenLake and ScienceWorld are described in full below; HotpotQA and DeepResearch are summarized briefly as they follow established setups.

E.1 FROZENLAKE

Action space. Free-form text. Each turn the agent emits a single `<action>D</action>` tag where `D` is one of `Up`, `Down`, `Left`, `Right`. There is no tool-calling layer; the interaction parses the tag and executes the corresponding grid move. Actions outside the four cardinal directions, missing tags, and malformed responses are recorded as invalid and consume the turn without changing state. Boundary moves leave the agent in place.

Observation. The initial observation contains the rendered grid using abstract symbols (`A`=agent, `B`=goal, `C`=hole, `D`=safe tile) together with a goal description. Symbols are deliberately abstract — without semantic priors — so the model must learn the mapping from interaction. Subsequent turns return the updated grid, contextual notes (boundary hits, invalid actions), and the remaining turn budget. Terminal observations name the cause of termination (*reached the goal*, *fell into the hole*, *hit the max step limit*).

Reward. Sparse binary outcome: $+1$ on reaching the goal tile, 0 otherwise. There is no shaping, step cost, or partial credit. Goal detection is read from the interaction's authoritative `turn_scores` signal; text-based detection is used only as a last-resort offline fallback because the model can otherwise game the reward by emitting the terminal phrase.

Termination. An episode ends as soon as any of: (i) the agent steps on the goal, (ii) the agent steps on a hole, (iii) the turn budget (10) is exhausted, or (iv) the response token budget is insufficient for the next observation.

Table 7: DeepResearch subtask taxonomy.

Subtask	Description (shown to grader)	φ_k
`factual_accuracy`	Getting the facts right: correct identifications, diagnoses, definitions, and numerical answers with the right values, units, and significant figures the rubric requires.	0.15
`analytical_reasoning`	Reasoning analytically about the evidence rather than echoing it: explaining the mechanism, derivation, or causal chain behind each claim, and producing prose that interprets, compares, and connects evidence across sources rather than stitching quotes together. Citations decorate genuine analysis; they do not substitute for it.	0.25
`completeness_coverage`	Answering the full question the user asked — covering every sub-question, comparison, condition, or step the question raises rather than stopping after the most salient slice or skipping the less obvious aspects.	0.15
`retrieval_strategy`	Choosing among `search_papers` (Scholar breadth), `read_paper` (PDF body via `pdf_url`), `google_search` (open web), and `browse_webpage` (single URL); picking literature vs. web tools per claim; issuing focused queries; recognising when to broaden, narrow, or stop.	0.20
`citation_practice`	Citing rigorously: every substantive claim is tied to specific retrieved evidence rather than padded with repeated cite tags; citations draw on authoritative, well-established sources over snippet-quality material when both are available; the answer triangulates across independent sources rather than over-relying on any single piece of evidence.	0.25

Dataset. Procedurally generated. Maps vary in size between 5×5 and 8×8 and in frozen-tile probability $p \in [0.65, 0.85]$; each generated map is validated by depth-first search to admit a path from start to goal of length ≤ 12. The default split is $20{,}000$ training maps and 500 validation maps. Random-board generation is the default, but the dataset format also supports explicit board serialization for fixed-eval scenarios.

Runtime. `max_user_turns` = 10, sampling temperature = 0.7, max prompt length = 1024 tokens, max response length = 4096 tokens.

E.2 HotpotQA

Action space. OpenAI-style function calling in Hermes format. A single tool is exposed:

- `search(query: str, top_k?: int ∈ [1,50])` — queries an E5 dense retriever backed by a FAISS index over the Search-R1 / ERL Wikipedia corpus. Default `top_k` $= 5$.

The terminal action is to emit the final answer in `\boxed{...}` anywhere in a free-form response. Tool calls and the boxed-answer terminal are mutually exclusive per turn.

Observation. Hermes-formatted tool returns enumerate retrieved passages as `[1] <title> <snippet>`, `[2] ...`, with a truncation notice when results exceed the per-result cap (1,200 chars) or the per-call total cap (8,000 chars). Non-tool, non-terminal observations are concise prompts to either continue searching or submit an answer. Terminal observations report the outcome explicitly: *"Your answer is correct"* (EM $= 1$), *"partially correct (F1 score: x)"* (F1 ≥ 0.3), or *"incorrect"*.

Reward. Computed against the normalized gold answer (lowercase, articles and punctuation stripped) at the boxed-answer turn:

- Exact-match reward 1.0 when the normalized prediction equals the normalized gold.
- Otherwise token-level F1 against the gold; if F1 ≥ 0.3 the reward equals F1, else reward is 0.

Episodes that never emit a boxed answer receive 0. There are no process or penalty terms; tool calls are free.

Termination. Episodes end when (i) the agent submits a boxed answer, (ii) the 5-turn budget is exhausted, or (iii) the response token budget cannot fit the next observation (estimated at 3 chars/token plus a 20-token overhead).

Dataset. The HotpotQA distractor split from Hugging Face (`hotpotqa/hotpot_qa`). The default training pool is $\sim$90 k examples; we evaluate on a fixed 500-example validation slice. The retrieval setup (E5 encoder, FAISS index over the Search-R1 Wikipedia dump) follows Search-R1 (Jin et al., 2025).

Runtime. `max_user_turns` $= 5$, sampling temperature $= 0.7$, max prompt length $= 1{,}024$ tokens, max response length $= 4{,}096$ tokens.

E.3 SCIENCEWORLD

Action space. Free-form text wrapped in `<action>...</action>` tags, dispatched to the ScienceWorld JVM via py4j. The simulator accepts 18 templated commands: `open`/`close OBJ`, `activate`/`deactivate OBJ`, `connect OBJ to OBJ`, `disconnect OBJ`, `use OBJ [on OBJ]`, `look around`, `examine OBJ`, `look at OBJ`, `read OBJ`, `move OBJ to OBJ`, `pick up OBJ`, `pour OBJ into OBJ`, `mix OBJ`, `teleport to LOCATION`, `focus on OBJ`, and `wait`. Two meta-commands — `help` and `objects` — are free and return the simulator's enumeration of currently valid actions and referenceable entities respectively. Unrecognized commands return rejection markers (*No known action matches*, *Unknown action*, *Ambiguous request*) and the interaction surfaces matching valid verbs to aid self-correction.

Observation. Natural-language descriptions emitted by the JVM, covering the current location, inventory, and immediately observable objects. The first observation concatenates the task instruction and reset state. Each subsequent observation is appended with the standard formatting reminder (`<think>...</think><action>...</action>`). Terminal observations include a steps-used count, the final score on a 0–100 scale, per-step reward attribution, and a qualitative outcome label.

Reward. The base signal is the simulator's monotonic-max partial-credit score, normalized to $[0, 1]$. Two multiplicative discounts and two small additive shaping terms are applied:

- *Stagnant-score discount* ($\times 0.5$): triggered when the score reaches a positive value but never improves thereafter and ends below 1.0. Targets binary-guess and prior-knowledge exploitation where the agent collects an early sub-goal and then idles.
- *No-experiment discount* ($\times 0.25$): triggered when a procedure-required task scores ≥ 0.5 but the agent never executes any experimental verb (`connect`, `activate`, `deactivate`, `use`, `mix`, `pour`, `wait`). Mitigates the *focus-on-X then move-to-box* reward hack on tasks like `test-conductivity` where the simulator's partial-credit scorer awards classification sub-goals even when the required experiment was not performed. Classification task families (`find-`$\star$, `lifespan-`$\star$, `identify-life-stages`) are exempt because `focus` is the terminal action by design.
- *Reasoning-absence penalty* (-0.1 when `<think>` tags appear in fewer than half of the turns) and *action-diversity bonus* (up to $+0.05$ for varied action verbs). These are reusable components shared across tasks.

Termination. An episode ends when the JVM signals task completion, the turn budget (30) is exhausted, the same action is emitted three turns in a row (early-termination guard against degenerate loops), or an internal JVM error occurs (the episode returns score 0.0 gracefully).

Dataset. ScienceWorld (Wang et al., 2022) provides 30 procedural tasks across 10 science topics with parametric variations. We use simplification mode `easy`. We support two splits: (i) the per-task

train/dev split distributed with the benchmark, and (ii) an R3L-style task-generalization split with 17 train tasks and 13 held-out test tasks. Per-task variations are sampled at 50% by default.

Runtime. `max_user_turns` = 30 (matching the JVM step limit), sampling temperature = 1.0, max prompt length = 8192 tokens, max response length = 8192 tokens.

E.4 DEEPRESEARCH

Action space. OpenAI-style function calling in Hermes format with four tools, all backed by the Serper.dev infrastructure plus local PDF/web fetchers:

- `search_papers(query: str, limit?: int ∈ [1, 10])` — Google Scholar via Serper. Internally Serper is always called with `max=10` to maximize cache hits and the model-visible result list is truncated to `limit` (default 5).
- `read_paper(pdf_url: str)` — downloads the PDF locally with a browser-like User-Agent, validates content type and magic bytes (paywall detection), and converts to markdown. Hard cap 15 MB.
- `google_search(query: str, gl?: str, hl?: str)` — open-web search via Serper, defaulting to 10 organic results.
- `browse_webpage(url: str)` — Serper scrape returning markdown for a single URL. Default per-page char cap 20,000.

All tools share a 120 s wall-clock cap and exponential-backoff retries. Outputs are cached in a Ray actor (LRU, max 16,384 entries) shared across rollout workers. The Serper tools share a per-process semaphore (default 12/worker, ∼96 global); `read_paper` runs locally with a separate, larger semaphore. The terminal action is closing an `<answer>...</answer>` block; substantive claims inside the answer must be wrapped in `<cite id="...">` with ids returned by retrieval tools.

Observation. Tool results are emitted as `[id=<id>] <title> ... [cites=N]` headers followed by snippet/content; paper ids use the prefix `p_<sha1>` and web ids are URL hashes. Truncation notices are appended when char caps fire. Terminal observations report the rubric score on a $[0, 1]$ scale plus verdict-bucket counts (*"F fully met, P partial, Z not met"*) over the N rubric criteria. Criterion titles and grader hints are deliberately *omitted* from the terminal observation to prevent answer leakage into reflection / aux-SFT.

Reward. A weighted sum of one outcome and three auxiliary signals, each in $[0, 1]$:

$$r = 0.7\, r_{\text{rubric}} + 0.1\, r_{\text{citation}} + 0.1\, r_{\text{format}} + 0.1\, r_{\text{search}},$$
$$r_{\text{citation}} = 0.6\, F_1^{\text{cite}} + 0.4\, r_{\text{citation-format}}.$$

- r_{rubric}: an LLM judge (default `gpt-5.4-nano` with low reasoning effort) grades the final answer against the per-instance static rubric on a 0–5 scale per criterion, normalized by the sum of criterion weights.
- F_1^{cite}: per-claim F1 between each `<cite id>`'d span and the cited snippet, evaluated by an LLM judge and averaged. Toggleable via env var (`DEEPRESEARCH_CITATION_F1_ENABLED`); when off, r_{citation} falls back to $r_{\text{citation-format}}$ alone.
- $r_{\text{citation-format}}$: fraction of cited ids that were actually retrieved by a tool call (anti-hallucination check on ids).
- $r_{\text{format}} = 0.5 \cdot \mathbf{1}[\text{has answer block}] + 0.3 \cdot \mathbf{1}[\text{has cite}] + 0.2 \cdot \mathbf{1}[\text{has tool call}]$.
- $r_{\text{search}} = \min(n_{\text{tool calls}}/3, 1)$.

A length penalty (soft target 6,000 chars, hard cap 15,000, max penalty configurable) is available but disabled by default.

Termination. Episodes end when (i) the agent closes an `<answer>` block, (ii) the response token budget cannot fit the next feedback (3 chars/token estimate, 20-token overhead), (iii) the prompt buffer fills, or (iv) the 10-turn budget is exhausted. All terminal paths invoke the rubric judge subject to a process-wide 90 s wall-clock cap.

Dataset. A merged pool of five Hugging Face sources, all distributed with static per-instance rubrics: `rl-research/dr-tulu-rl-data` (OpenScholar, AstaBench-ScholarQA-CS2, SearchArena slices, capped at 2 k / 2 k / 5 k), `anisha2102/RaR-Science-20k-o3-mini` (1.5 k), and `anisha2102/RaR-Medicine-20k-o3-mini` (0.5 k). Heterogeneous rubric schemas are normalized to {`id`, `title`, `description`, `weight`, `leak_safe_hint`}; leak-safe hints are LLM-generated to support auxiliary distillation without exposing the gold answer. Setup otherwise follows DR Tulu (Shao et al., 2025); the evolving-rubric mechanism from the original paper is intentionally omitted (rubrics are static).

Runtime. `max_user_turns` $= 10$, sampling temperature $= 1.0$, max prompt length $= 2{,}048$ tokens, max response length $= 16{,}384$ tokens.

F Reflection and Grading Prompts

This section reproduces the prompts used by π_θ in the reflect–retry loop on HotpotQA. The same template structure (system prompt, rollout-time conditioning, R3L two-pass grading, R3L graded reflection) is used on every task; only the rubric strings (Appendix D) and the system prompt are task-specific. All prompts live verbatim in `rlef/tasks/<task>/constants.py` and substitution placeholders (e.g. `{trajectory_summary}`, `{sub_tasks_section}`) are filled by `rlef/r3l/credit.py` at rollout time.

System prompt (HotpotQA).

```
You are a helpful assistant who answers questions directly and efficiently.
When you have enough evidence, provide your final answer inside \boxed{}.

Use the search tool when you need evidence.
You may optionally pass top_k to the search tool when you need broader retrieval,
but prefer smaller top_k values unless more coverage is necessary because long
search responses may be truncated.
When calling tools, pass valid JSON arguments with double-quoted keys and values,
for example: {"query":"who wrote the iliad","top_k":3}.
Do not output tool schemas, pseudo-JSON, or commentary in tool arguments.
```

Grading prompt (R3L two-pass, all tasks). The grader sees the full trajectory plus the per-task subtask list (rendered from `SUBTASKS` as a numbered `{sub_tasks_section}`) and emits one Likert grade and one outcome-aware `pivot_step` per subtask. The pivot semantics are explicitly outcome-aware: on a successful trajectory the pivot is the earliest *load-bearing* step for the subtask; on a failure it is the earliest *degradation* step.

```
You are an evaluator that grades an agent's trajectory across competency
dimensions and identifies, for each, the single step where that competency
was most *pivotal* for the outcome that occurred. Score each dimension on a
1-5 scale where 1=critical failure, 2=poor, 3=adequate, 4=good, 5=excellent.

A "pivot_step" is outcome-aware:
- If the trajectory succeeded ('success' or 'success_but_inefficient'),
  the pivot_step is the earliest step where the agent's handling of this
  competency was load-bearing for success - a decision that, if reversed
  or substituted with a plausible alternative, would have meaningfully
  reduced the chance of reaching the correct answer.
- If the trajectory failed ('failure'), the pivot_step is the earliest
  step where this competency first degraded - the point at which the
  root cause of the failure manifested along this dimension.

Use null when no specific step is pivotal for this competency.
```

```
{sub_tasks_section}

## Trajectory
{trajectory_summary}

Please output ONLY a single valid JSON object in the following format:
{
  "trajectory_outcome": "One of: 'success', 'success_but_inefficient', 'failure'",
  "root_cause_analysis": "If the trajectory failed, explain why. If it
    succeeded, explain what made it succeed. Trace observable symptoms back
    to a fundamental cause using iterative 'why' questioning.",
  "sub_task_grades": [
    {
      "sub_task": "Name of the competency dimension from the list above.",
      "score": "Integer 1-5 (1=critical failure, 5=excellent).",
      "assessment": "One-sentence justification for the score.",
      "pivot_step": "Integer step index where this competency was pivotal,
        with outcome-aware semantics described above, or null."
    }
  ]
}

Include one entry in sub_task_grades for EVERY competency dimension above.
```

Graded-reflection prompt (R3L two-pass, all tasks). After grading, the same policy is asked to write a reflection conditioned on the grades. The output is a structured JSON object with a `retry_from_step` field that the trainer feeds into the per-subtask pivot mask.

```
You are a Reflector analyzing a trajectory that has already been graded
across competency dimensions. Use the grades to inform your analysis.

{grading_section}

{memory_section}{contrastive_section}
## Failed trajectory
{trajectory_summary}

Analyze the trajectory in light of the grades above. Identify which
subtask(s) to address and produce a reflection that will most improve
performance on a retry.

Please output ONLY a single valid JSON object in the following format:
{
  "trajectory_summary": "Concise overview covering: (1) strategy employed,
    (2) final result, (3) key observations about execution quality.",
  "root_cause_analysis": "Deep causal analysis grounded in specific
    trajectory steps. Walk forward and name the earliest step where the
    agent had a real choice and picked wrong; quote that step's action
    or observation. Chain reasoning explicitly from that decisive moment
    to the final outcome.",
  "trajectory_outcome": "One of: 'success', 'success_but_inefficient',
    'failure'",
  "improvement_suggestion": "Actionable principle for the retry attempt.",
  "retry_from_step": "Non-negative integer naming the earliest step where
    a different action would have materially changed the outcome. Quote
    that step's action or observation in root_cause_analysis. Use 0 when
    the first action itself was the decisive wrong choice."
}
```

The corresponding ScienceWorld, FrozenLake, and DeepResearch prompts share this structure verbatim; the only differences are the system prompt text and the contents of `{sub_tasks_section}` (Appendix D). The single-pass variant `R3L_REFLECTION_PROMPT` (used when the grading pass is disabled) drops the `{grading_section}` block but preserves the same JSON output schema sans `sub_task_grades`.

System prompt (excerpt).

```
You are an agent playing a game on a grid, acting as a reasoning engine.

# Game Rules
*(To be discovered through play - consult any additions below.)*

# Strategic Playbook
*(To be discovered through play - consult any additions below.)*

# Required response structure
<think>
**1. Rules & Playbook Consulted:** ...
**2. Analysis of the Current State:** ...
**3. Prediction of the Value of Current States:** ...
**4. Prediction of Immediate Consequences:** ...
</think>
<action>Up | Down | Left | Right</action>
```

User turn 1 (initial observation).

```
Current Observation:
D D D D D D C
D D C D D D D
D D B D D C D
C D D C D D D
D C C C C A C
D D D C D C C
C C D C D D C
You have not achieved the goal yet. Please give the next action.
```

Assistant turn 1 (response, abridged).

```
**1. Rules & Playbook Consulted:** None available. The presence of 'C',
'D', 'B', and 'A' suggests possible progression rules ... however,
without explicit game rules, the interpretation is speculative.

**4. Prediction of Immediate Consequences:**
1. Action: Right - leads to (3,3) ('D'), opens path toward center.
2. Action: Down - leads to (4,2) ('C'). 'C' is adjacent to 'A' in
   row 5, column 5. A path C->C->C->C->A suggests a direct route to 'A'.

Both actions are plausible, but down leads more directly toward a
contiguous path to 'A' via 'C' tiles.

<action>Down</action>
```

User turn 2 (terminal observation).

```
The agent moved onto a 'C' tile and fell into the hole.
```

Figure 10: Anecdotal FrozenLake rollout at eval step 30. The empty `Game Rules` block leaves the policy without the `C`-is-a-hole prior, and the abstract-symbol vocabulary (§E) prevents pre-training from filling the gap. The grader pins `hazard_identification` (highest weight after `path_planning`) as the failed competency and the reflector emits an addition that fixes the rule for the retry.