LEARNING WITH RARE SUCCESS BUT RICH FEEDBACK VIA REFLECTION-ENHANCED SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Enabling Large Language Models (LLMs) to continuously improve from environmental interactions is a central challenge in post-training. While on-policy self-distillation offers a promising paradigm, existing methods predominantly treat environmental feedback as a passive conditioning signal. Consequently, they heavily rely on successful demonstrations and struggle to learn in rare-success regimes. To bridge this gap, we introduce Reflection-Enhanced Self-Distillation (RESD), a framework that transforms raw failure feedback into an active source of corrective supervision. Instead of passively appending feedback, RESD interprets failed trajectories by generating retrospective reflections to diagnose local errors, and curates a persistent global playbook to preserve reusable lessons across training steps. The enriched context enables the self-teacher to provide actionable token-level supervision even in the absence of successful rollouts. Empirical evaluations on multiple continual learning tasks demonstrate that RESD substantially outperforms standard self-distillation baselines. Furthermore, RESD achieves faster early-stage improvement than GRPO with $8\times$ more samples while using only a single rollout per prompt, highlighting its interaction efficiency.

1 INTRODUCTION

A fundamental challenge in the post-training of Large Language Models (LLMs) is enabling continuous improvement through environmental interactions. Traditionally, Reinforcement Learning (RL) algorithms, such as PPO or GRPO, have been the standard paradigm for aligning models with desired outcomes. However, a critical bottleneck in deploying RL for complex, multi-step tasks is the reliance on sparse reward signals. In scenarios where successful trajectories are exceedingly rare, standard RL often struggles to effectively bootstrap policies because it lacks dense supervision to guide the model through a vast exploration space.

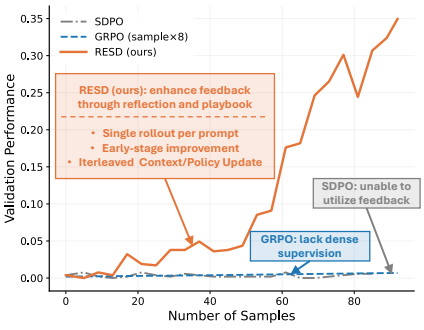


Figure 1: RESD improves interaction efficiency during training. The x-axis is the number of samples.

To mitigate the sparse reward problem, On-Policy Distillation (OPD) has emerged as a promising alternative (Agarwal et al., 2023; Lu & Lab, 2025; Xiao et al., 2026). Rather than relying on a single scalar reward at the end of a rollout, OPD leverages a stronger or privileged teacher model to compute target probabilities for every single token in the generated trajectory. This mechanism translates sparse trajectory-level outcomes into dense, token-level learning signals. However, standard OPD requires maintaining a separate expert model, which incurs high computational costs and risks distribution mismatch between the teacher and the student. To address this, self-distillation variants like SDPO instantiate the teacher from the model itself (Hübotter et al., 2026; Zhao et al., 2026). By conditioning the self-teacher on environmental feedback, SDPO eliminates the need for an external oracle while aligning the supervisory signal with the model's own generation distribution.

While self-distillation addresses both reward sparsity and distribution mismatch, its success hinges on the quality of the self-teacher's supervision. Current methods predominantly treat environmental

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.

feedback as a static, passive conditioning variable. *We argue that this overlooks a key design choice: the structural representation of the feedback itself.* In sparse-reward regimes where successful demonstrations are absent, the teacher is forced to derive supervision almost entirely from failed rollouts. We demonstrate in Figure 1 that simply feeding raw failure outcomes or unstructured environment errors into the teacher context yields only marginal student improvements.

Motivated by this observation, we propose Reflection-Enhanced Self-Distillation (RESD), a framework that enriches the teacher context with actively interpreted feedback. For each failed rollout, RESD first generates a retrospective reflection that identifies the likely cause of failure and the correction that would have avoided it. It then organizes recurring lessons into a persistent playbook, allowing future teacher prompts to reuse feedback-derived knowledge across training steps. The resulting enriched teacher context transforms raw failure feedback from a passive diagnostic signal into an actionable and reusable source of corrective supervision, enabling self-distillation to improve even when successful demonstrations are unavailable.

Empirically, we evaluate RESD on tasks designed to reflect continuous learning beyond common post-training distributions. These tasks are novel to the model, often begin in rare-success regimes, and provide rich execution feedback even though their rewards are sparse and binary. Across these settings, RESD substantially improves over standard self-distillation baselines and achieves faster early-stage improvement than GRPO while using only a single rollout per prompt. These results suggest that structured feedback enables sample-efficient bootstrapping from failures, while reward-based optimization remains a complementary tool once sufficient successful rollouts are available. Our contributions are as follows:

- We identify *feedback formulation* as a critical design axis for on-policy self-distillation, especially in sparse-reward regimes.
- We propose *Reflection-Enhanced Self-Distillation* (RESD), which enriches the teacher context with retrospective reflections and a persistent playbook. By interpreting failures and preserving recurring lessons across training steps, RESD turns raw feedback from a passive diagnostic signal into reusable corrective supervision. Empirically, we characterize the sample efficiency of feedback-enhanced self-distillation under an online streaming protocol.
- More broadly, by revealing that self-improving models are highly sensitive to how feedback is structured, we position our reflection and playbook mechanisms as a plug-and-play module that can readily enhance other self-distillation objectives.

2 PRELIMINARIES

Recent advancements such as Self-Distillation Policy Optimization (SDPO) (Hübotter et al., 2026) and On-Policy Self-Distillation (OPSD) (Zhao et al., 2026) introduce a "self-teacher" paradigm that replaces sparse outcomes in Reinforcement Learning with dense, token-level supervision without the need for a massive separate teacher model. In these frameworks, a "self-teacher" policy is typically instantiated as the model's current or moving-average weights θ_{old}, operating in a privileged context. The teacher is conditioned on both the intermediate state $s_t = (x, y_{<t})$ and the retrospective feedback $c(x, y)$, producing a corrected, feedback-informed distribution $\pi_{\theta_{\text{old}}}(\cdot|x, y_{<t}, c)$. The base policy π_θ is then optimized to match this privileged distribution at every decoding step. We formulate this matching objective using *f-divergences*. For a given time step t, let the likelihood ratio for any token $v \in \mathcal{V}$ be defined as $\tau_v^t = \frac{\pi_{\theta_{\text{old}}}(v|x,y_{<t},c)}{\pi_\theta(v|x,y_{<t})}$. To prevent extreme ratios from destabilizing training, we clip the likelihood ratio to a bounded range:

$$\tilde{\tau}_v^t = \text{clip}\big(\tau_v^t,\ \epsilon_{\min},\ \epsilon_{\max}\big)$$

The unified self-distillation objective then minimizes the expected f-divergence across all tokens $t \in [1, T]$:

$$\mathcal{L}_{\text{SD}}(\theta) = \mathbb{E}_{x,y\sim\pi_\theta,f}\left[\sum_{t=1}^{T}\sum_{v\in\mathcal{V}} \pi_\theta(v|x, y_{<t}) \cdot f(\tilde{\tau}_v^t)\right] \tag{1}$$

where $f(\tau)$ is a convex function. By selecting different f, we can recover various divergence measures, such as forward KL, reverse KL and Jensen-Shannon Divergence (JSD), each offering unique optimization properties. We provide a detailed discussion on the choice of f and its implications

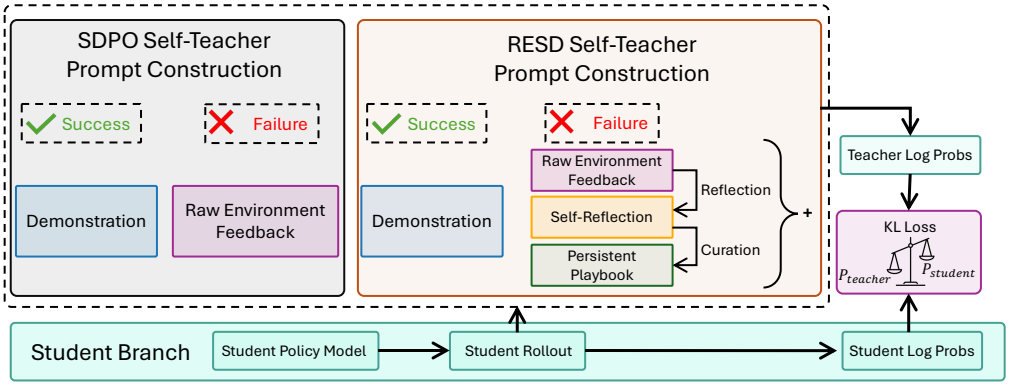


Figure 2: Overview of the RESD framework. The student generates a rollout and receives environment feedback. On failure, a local self-reflection diagnoses the error and a global playbook curation step distills reusable lessons into a persistent playbook. The enriched context (reflection, curated playbook, and any cached solutions) is fed to the teacher prompt, whose output distribution provides token-level supervision to the student via a KL loss.

in Appendix A. Since we sample trajectories from the base policy π_θ, the self-distillation objective is inherently on-policy. Such a formulation allows the model to recover forgotten behaviors after midtraining (Shenfeld et al., 2026; Chen et al., 2025; Shenfeld et al., 2025; Lai et al., 2026) and to avoid exposure bias (Lu & Lab, 2025; Agarwal et al., 2024).

3 METHOD

Our work builds on the finding that existing self-distillation methods rely heavily on successful demonstrations, and their supervision degrades when successful rollouts are scarce. We propose Reflection-Enhanced Self-Distillation (RESD), which enriches the teacher's context with retrospective feedback from failures and a persistent memory of past experiences, enabling corrective supervision even without successful demonstrations. We first motivate the problem by analyzing SDPO's failure mode under sparse rewards (§3.1), then detail the retrospective reflection and playbook curation mechanism (§3.2), and describe how these components are integrated into a memory-augmented self-distillation framework (§3.3).

3.1 SELF-DISTILLATION RELIES ON SUCCESSFUL PEER DEMONSTRATIONS

The privileged context of the SDPO self-teacher comprises two complementary signals: environment feedback on the current trajectory and successful peer demonstrations sampled within the same rollout batch (Hübotter et al., 2026). To disentangle their contributions, we conduct an ablation that varies the rollout batch size N while keeping all other hyperparameters fixed. When $N{=}1$, each prompt group samples a single rollout, eliminating the possibility of peer demonstrations entirely and leaving environment feedback as the sole supervisory signal. As shown in Figure 3, SDPO's performance degrades substantially in this regime, indicating that the teacher struggles to translate failure feedback alone into effective corrective distributions. This failure mode becomes particularly severe in the rare-success regime. If the per-rollout success rate is small, most rollout groups will contain only failures and provide no positive counterfactual demonstration, forcing the teacher to construct token-level supervision from raw failure feedback alone, which limits improvement during training.

3.2 FROM PASSIVE FEEDBACK EXPOSURE TO ACTIVE FEEDBACK UNDERSTANDING

The degradation observed in the $N{=}1$ setting suggests that environment feedback alone is not sufficient to support effective self-distillation. This is somewhat surprising, as the teacher is still given privileged information about the student's failed trajectory, and in principle should be able to use this information to produce a more corrective distribution. *We argue that the issue lies not in the absence of feedback but in how feedback is represented and used.* Existing self-distillation methods often

expose the teacher to environment feedback as additional context, but leave the interpretation of that feedback entirely implicit. In rare-success regimes, this passive use of feedback becomes a bottleneck: raw failure signals are often diagnostic rather than instructional, indicating that a trajectory failed without explaining which decision caused the failure.

This motivates a shift from passive feedback exposure to active feedback understanding. For feedback to become useful supervision, the teacher must perform two additional operations. **(a)** It must retrospectively interpret the failed trajectory, connecting the final feedback signal to the intermediate reasoning that caused the failure. **(b)** It must preserve useful lessons across trajectories, since many failures are repeated manifestations of the same hidden rule rather than independent mistakes. In this view, raw feedback is only a starting point, and *it must be transformed into causal explanations and persistent knowledge before it can reliably guide token-level distillation*. To test this idea, we first built SDPO+Ref, an initial version of RESD that adds only operation (a): the teacher receives a retrospective reflection on each failed rollout, with no playbook and no successful demonstrations. As shown by the yellow curve in Figure 3, reflection alone already recovers much of the performance lost in the $N{=}1$ regime. Further analysis on the distribution of per-prompt accuracy (i.e., we sample 4 times for each prompt and measure the proportion of correct trials) in Figure 4 shows that the performance improvement of SDPO ($N{=}8$) mostly comes from increasing the number of completely correct prompts, while SDPO+Ref ($N{=}1$) reduces the number of completely incorrect prompts, indicating that reflections help the model learn from failures.

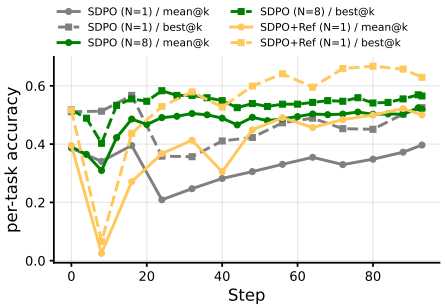
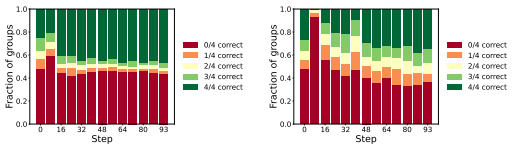


Figure 3: Per-task accuracy on FINER under varying rollout batch sizes. SDPO ($N{=}1$) degrades without peer demonstrations, while adding only reflection (SDPO+Ref, $N{=}1$), an initial version of RESD, recovers and surpasses the $N{=}8$ baseline.

3.3 REFLECTION-ENHANCED SELF-DISTILLATION (RESD)

Building on the above observation, we introduce Reflection-Enhanced Self-Distillation (RESD), a framework that operationalizes active feedback understanding within the self-distillation loop. RESD maintains two forms of persistent context: a playbook $\mathcal{P}_t$, following the broader idea of Zhang et al. (2025), that stores reusable lessons distilled from previous failures, and an optional solution buffer $\mathcal{B}$ that caches successful trajectories when they are available. At each training step, RESD first updates this context using the outcome of the current rollout, and then conditions the teacher on the enriched context to produce token-level supervision. This design allows failed trajectories to improve future teacher prompts, rather than being used only as one-shot raw feedback.



(a) SDPO (N=8). (b) SDPO+Ref (N=1).

Figure 4: Per-prompt mean accuracy distribution on FINER over training steps. SDPO+Ref reduces the fraction of all-wrong prompts more.

Retrospective Reflection on Failures. For a failed trajectory (x, y) with environment feedback $c(x, y)$, RESD invokes the student model in a diagnostic mode to generate a retrospective reflection. Formally, given the current playbook $\mathcal{P}_t$, the reflector outputs

$$r, \{\ell_{t,j}\}_{j=1}^{|\mathcal{P}_t|} = \textsc{Reflect}(x, y, c, \mathcal{P}_t; \theta),$$

where r is a natural-language explanation of the failure and $\ell_{t,j} \in \{\texttt{helpful}, \texttt{harmful}, \texttt{neutral}\}$ indicates whether the existing playbook entry $p_j \in \mathcal{P}_t$ helped or misled the current attempt. The reflection is designed to make the raw feedback more actionable by connecting it to specific reasoning steps. The entry-level tags provide a lightweight usage signal for maintaining the playbook over time.

Playbook Curation from Past Experiences. The playbook $\mathcal{P}_t = \{p_1, \ldots, p_M\}$ is a persistent set of natural-language entries that encode reusable lessons. Given a reflection r, the curator generates new

Algorithm 1 Reflection-Enhanced Self-Distillation (RESD)

Require: Student policy π_θ, teacher weights θ_{old}, playbook $\mathcal{P} \leftarrow \emptyset$, solution buffer $\mathcal{B} \leftarrow \emptyset$

for each training step **do**
  // Rollout & Reward
  Sample rollout $y \sim \pi_\theta(\cdot \mid x)$
  Obtain environment feedback $c(x, y)$ and reward $R(x, y)$
  // Context Update
  $\mathcal{P} \leftarrow \textsc{Concise}(\mathcal{P})$ ▷ prune stale/harmful entries
  if $R(x, y) <$ threshold **then**
    $r \leftarrow \textsc{Reflect}(x, y, c, \mathcal{P};\, \theta)$ ▷ generate reflection and tag playbook entries
    $\mathcal{P} \leftarrow \textsc{Curate}(\mathcal{P}, r;\, \theta)$ ▷ generate new playbook entries from reflection
  else
    $\mathcal{B}(x) \leftarrow y$ ▷ cache successful solution for future replay
  end if
  // Enriched Teacher Prompt Construction
  Retrieve buffered solution $\mathcal{B}(x)$ (if available)
  Construct enriched context: playbook $\mathcal{P}$, reflection r, previous trial y, feedback c, solution $\mathcal{B}(x)$
  // Policy Update
  Compute teacher distribution $\pi_{\theta_{\text{old}}}(\cdot \mid x, y_{<t}, c, r, \mathcal{P}, \mathcal{B}(x))$
  Update θ by minimizing $\mathcal{L}_{\text{SD}}(\theta)$
  $\theta_{\text{old}} \leftarrow \theta$ ▷ sync teacher weights
end for

entries to add:

$$\mathcal{P}_{t+1} = \mathcal{P}_t \cup \textsc{Curate}(\mathcal{P}_t, r;\, \theta)$$

where π_θ is prompted with the current playbook and the reflection, and produces a set of non-redundant entries that capture lessons not already present in $\mathcal{P}_t$. Each entry p_j maintains accumulated counts (h_j, d_j) of helpful and harmful tags received from the reflector across steps. Before each context update, a CONCISE operation prunes the playbook by removing entries where $d_j \geq h_j$ (net harmful) and, if $|\mathcal{P}_t|$ exceeds a budget $M_{\max}$, evicting the least recently tagged entries. This lifecycle ensures the playbook remains compact and converges toward entries that are repeatedly useful across trajectories. We detail the full CONCISE strategy, including trigger conditions and two removal variants, in Appendix G.

Interleaving Context Update with Policy Optimization. Algorithm 1 summarizes the full RESD training loop. At each step t, the student π_θ generates a rollout $y \sim \pi_\theta(\cdot \mid x)$ and receives environment feedback $c(x, y)$ and reward $R(x, y)$ indicating correctness. Before the policy update, a context update phase modifies the persistent state: CONCISE first prunes $\mathcal{P}_t$, then, if the rollout failed, REFLECT and CURATE extend the playbook to $\mathcal{P}_{t+1}$; if it succeeded, the solution buffer is updated, $\mathcal{B}(x) \leftarrow y$, and $r = \emptyset$. The teacher is then synced with the student via an EMA update (Hübotter et al., 2026), and conditioned on the enriched context to produce token-level supervision:

$$\pi_{\theta_{\text{old}}}(\cdot \mid x, y_{<t}, c, r, \mathcal{P}_{t+1}, \mathcal{B}(x))$$

and the student minimizes the self-distillation objective $\mathcal{L}_{\text{SD}}(\theta)$ (Eq. 1). We additionally apply a per-sample weight to enhance the gradient contribution of successful samples, computed as a function of the batch success rate (details in Appendix B). Because $\mathcal{P}$ and $\mathcal{B}$ persist across steps, the supervision available to the teacher can improve over time even when the current batch contains no successful demonstration.

4 EXPERIMENTS

Our experiments are designed to address the following questions:

- Does RESD improve over self-distillation baselines, especially in rare-success regimes? (§4.2, §4.3)
- How does RESD compare to reward-based RL (GRPO) and other self-distillation methods (SRPO, RLSD)? (§4.4, §4.5)
- How does each component of RESD affect performance? (§4.6)

Table 1: Overview of evaluation tasks. "Train" and "Test" denote the number of unique problems in each split.

Task	Train	Test	Description
MANUFACTORIA-HAS (Sun et al., 2025)	742	132	Write DSL programs that check whether an input tape contains a target pattern
BOUNCINGSIM-EASY (Sun et al., 2025)	640	100	Write Python code to simulate 2-D multi-object bouncing dynamics (easy)
BOUNCINGSIM-MEDIUM (Sun et al., 2025)	320	100	Write Python code to simulate 2-D multi-object bouncing dynamics (medium)
FINER (Zhang et al., 2025)	1000	500	Tag financial named entities in SEC filings with XBRL labels

- How does RESD interpret and organize failure feedback at the instance level? (Appendix K)

We additionally evaluate instruction-following preservation and training latency in Appendices H and I.

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate on four tasks spanning program synthesis and physical reasoning. MANUFACTORIA-HAS and BOUNCINGSIM-EASY/MEDIUM are drawn from the RL-Grok benchmark (Sun et al., 2025), while FINER is from Zhang et al. (2025). The RL-Grok tasks are characterized by near-zero initial success rates, making them a natural testbed for learning from failure feedback. FINER, by contrast, exhibits a higher initial success rate, allowing us to assess whether RESD also benefits regimes where successful demonstrations are more accessible. While we mainly focus on tasks that require adapting to novel environments, RESD is also competitive on LiveCodeBench (Jain et al., 2024), a standard reasoning benchmark where success is common (Appendix F). Table 1 summarizes the datasets. We train `Qwen3-4B-Thinking-2507` for MANUFACTORIA-HAS, BOUNCINGSIM-EASY, and FINER, and `Qwen3-30B-A3B-Thinking-2507` for BOUNCINGSIM-MEDIUM.

Settings. We adopt an *online streaming* training protocol in which the model makes a single pass over the training data and each training example is seen at most once. For every incoming batch, the trainer executes an inner loop of up to $K = 4$ update iterations on the same set of prompts. This protocol is designed to mimic practical self-improvement scenarios, where deployed models continuously encounter new tasks, observe task outcomes, and must update under a limited interaction budget (N=1). Unless otherwise specified, we update the teacher weights by EMA with a rate of 0.0001. We optimize reverse KL as the self-distillation loss with a lower clip $\epsilon_{\text{min}} = 0.2$ and, by default, no upper clip, with a training batch size of 32. We provide further implementation details in Appendix J.

Metrics and Baselines. We report both *mean@4* (m@4), the average performance over four independently sampled responses per test problem, and *best@4* (b@4), the best performance among the four samples. We use two evaluation metrics: *Per-Task Accuracy*, which measures the fraction of problems whose response passes all test cases (equivalent to pass@1 for a single sample, also used as the reward), and *Per-Test-Case Accuracy*, which measures the average fraction of test cases passed across problems. Since different validation metrics may peak at different training steps, we select a single checkpoint per run using a rank-based rule: for each logged step, we rank its four validation metrics, sum the ranks, and report the checkpoint with the highest total rank. The same selection rule is applied uniformly to all methods. We additionally include **SDPO+ss**, a variant that also distills on the model's own successful rollouts, which SDPO skips by default, providing denser teacher supervision in higher-success regimes.

4.2 MAIN RESULTS

Table 2 reports the main results across all four tasks. RESD achieves the best performance across all tasks and metrics. The gains are largest in the rare-success regimes. On MANUFACTORIA-HAS, RESD improves per-task accuracy from $0.57/2.27$ under SDPO+ss to $35.80/65.91$ in m@4/b@4, while also improving per-test-case accuracy from $37.97/71.93$ to $76.95/96.53$. RESD also consistently improves over SDPO and SDPO+ss on both BOUNCINGSIM tasks, showing that failure-derived reflection and playbook memory are useful beyond the DSL setting. On FINER, where successful demonstrations are more accessible, RESD still obtains the strongest results, suggesting that structured

Table 2: Main results on four tasks. *m@4* and *b@4* denote mean@4 and best@4. Per-Task Accuracy is the fraction of problems whose response passes all test cases; Per-Test-Case Accuracy is the average fraction of test cases passed.

	MANUFACTORIA-HAS (Qwen3-4B)		BOUNCINGSIM-EASY (Qwen3-4B)		BOUNCINGSIM-MEDIUM (Qwen3-30B-A3B)		FINER (Qwen3-4B)	
Method	m@4	b@4	m@4	b@4	m@4	b@4	m@4	b@4
Per-Task Accuracy								
Base model	0.38	1.52	0.00	0.00	1.00	3.00	39.28	51.90
SDPO (Hübotter et al., 2026)	0.95	3.79	1.00	3.00	2.00	6.00	40.93	50.50
SDPO+ss	0.57	2.27	1.00	4.00	2.00	6.00	50.25	59.12
RESD (ours)	35.80	65.91	4.25	8.00	7.25	10.00	53.66	61.12
Per-Test-Case Accuracy								
Base model	25.31	60.97	27.85	33.53	23.72	34.67	69.65	79.21
SDPO (Hübotter et al., 2026)	37.66	73.55	28.98	46.20	24.38	34.47	71.27	78.21
SDPO+ss	37.97	71.93	29.68	47.73	24.93	35.80	76.15	82.21
RESD (ours)	76.95	96.53	38.87	55.80	34.72	41.07	78.98	84.32

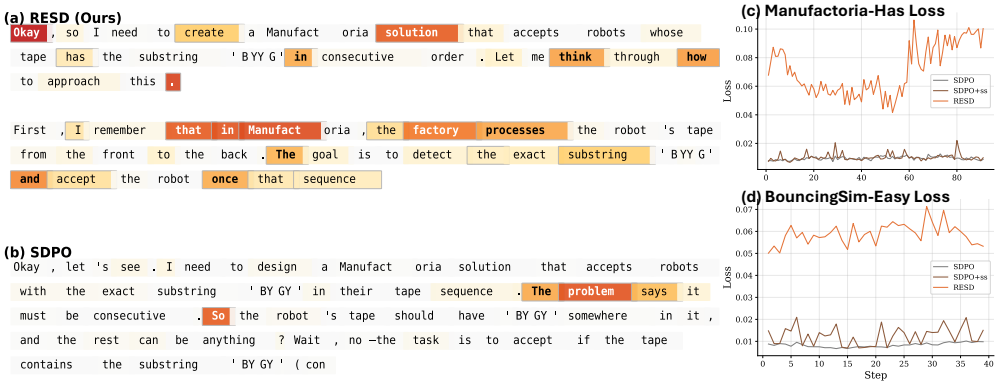


Figure 5: (a)(b) Token-level distillation loss for training step 20. Each token is shaded from white (zero loss) to dark red (high loss). (c)(d) Average per-token distillation loss across training steps for RESD and baselines.

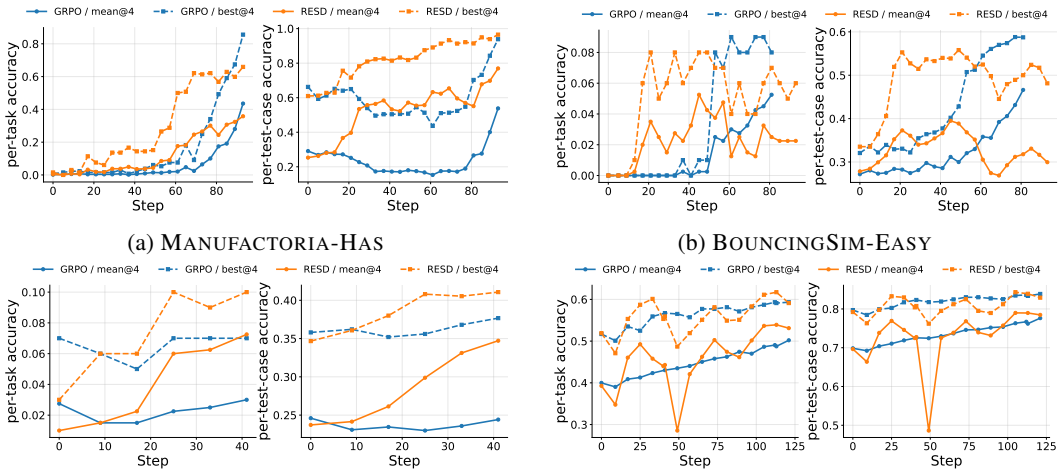
feedback remains beneficial even outside the sparsest regimes. Full validation training curves for every run in Table 2 are provided in Appendix C.

4.3 REFLECTION AND PLAYBOOK ELICIT A MORE CORRECTIVE TEACHER

We next examine how enriching the teacher prompt changes the self-distillation signal. Figure 5 plots both per-token and average per-token distillation losses over training steps. (1) Across tasks, RESD often yields a higher self-distillation loss than the baselines. Since this loss measures the discrepancy between the student distribution and the feedback-conditioned teacher distribution, a higher value suggests that the reflection- and playbook-augmented teacher provides a distribution that deviates more from the student's current behavior. (2) RESD places more loss mass on reasoning-decision tokens such as "create", "solution", "processes" and "The goal". These are points where the model chooses what to do next or how to frame the problem, suggesting that the teacher is correcting the reasoning strategy.

4.4 COMPARISON WITH GRPO

We compare RESD with GRPO (Shao et al., 2024), a strong on-policy reinforcement learning baseline, in Figure 6. The key difference is the interaction pattern: GRPO uses grouped rollouts with group size 8, requiring eight sampled trajectories and reward/feedback evaluations per prompt, whereas RESD uses a single rollout per prompt and updates its teacher context from that trajectory. At equal training steps, GRPO therefore receives $8\times$ more environment interactions. Under matched interaction budgets and wall-clock time (Appendix D), GRPO does not improve on MANUFACTORIA-HAS within the budget of a full RESD run, while RESD reaches 76.9 per-test-case m@4. Despite using



(a) MANUFACTORIA-HAS (b) BOUNCINGSIM-EASY

(c) BOUNCINGSIM-MEDIUM (d) FINER

Figure 6: Validation performance across training steps for RESD and GRPO. GRPO samples 8 rollouts per prompt, while RESD uses a single rollout.

fewer rollouts per prompt, RESD improves faster than GRPO in the early stage of training across the tasks in Figure 6. These results suggest that RESD is well suited for sample-efficient bootstrapping under limited interaction budgets. We note that RESD does not always improve monotonically and the validation curves exhibit occasional dips. On FINER, the sharp drop around step 50 is primarily caused by response truncation. On BOUNCINGSIM-EASY, the later-stage degradation after around step 60 suggests that once the model reaches a higher-success regime, dense self-distillation from feedback-derived contexts becomes less stable than reward-based optimization. A practical strategy is to use RESD to quickly reach a higher-success regime, then switch to stabler reward-based optimization such as GRPO once grouped rollouts provide reliable learning signals. The switching point can be detected online from training rollout accuracy, which closely tracks validation accuracy (Appendix E).

4.5 COMPARISON WITH OTHER SELF-DISTILLATION METHODS

Beyond vanilla self-distillation, recent methods modify how the objective treats failed trajectories. We compare against two of them: SRPO (Li et al., 2026) routes successful rollouts to GRPO and failed rollouts to entropy-weighted self-distillation, and RLSD (Yang et al., 2026) uses environment rewards to determine the update direction of the self-distillation signal. We run both on MANUFACTORIA-HAS, our sparsest task, with their canonical group size of 8, with reflection, playbook, and solution buffer disabled, and with all other settings identical to RESD. Table 3 shows that RESD achieves the highest m@4 and b@4 with a single rollout per prompt. RLSD is the strongest baseline, but still trails RESD while using $8\times$ more environment interactions per prompt. SRPO underperforms because its reward-aligned GRPO branch is almost never active in this regime: nearly all batches contain only failures, so about 98% of samples are routed to the distillation branch. These methods change the loss applied to failed trajectories, whereas RESD changes the context from which the teacher distribution is computed. The two are therefore complementary: RESD's reflection and playbook can be combined with other self-distillation objectives such as RLSD's.

Table 3: Other self-distillation methods on MANUF.-HAS (per-test-case acc.).

Method	N	m@4	b@4
SDPO (Hübotter et al., 2026)	1	37.66	73.55
SDPO+ss	1	37.97	71.93
SRPO (Li et al., 2026)	8	12.23	28.71
RLSD (Yang et al., 2026)	8	65.63	90.53
RESD (ours)	1	**76.95**	**96.53**

Table 4: Ablation of teacher prompt components. Per-test-case accuracy (m@4 / b@4).

	MANUF.-HAS		BSIM-EASY		FINER	
Method	m@4	b@4	m@4	b@4	m@4	b@4
RESD (full)	76.95	96.53	38.87	55.80	78.98	84.32
w/o reflection	51.41	79.26	36.85	55.33	77.78	82.26
w/o playbook	60.65	92.37	32.95	53.60	77.42	83.07
w/o solution buffer	60.24	80.10	31.97	52.40	78.36	83.42

4.6 Ablation Study on Teacher Prompt Components

We ablate RESD's enriched teacher context by removing local reflection, persistent playbook, or solution buffer. Table 4 shows that full RESD achieves the strongest performance across all three tasks. Removing reflection causes the largest degradation on MANUFACTORIA-HAS (76.95→51.41 m@4), while removing the playbook or solution buffer also lowers performance, suggesting that local reflection provides short-term corrections while the playbook preserves reusable lessons. The solution buffer further helps convert partial progress into fully correct solutions.

5 Related Work

On-policy self-distillation. Knowledge distillation is widely used as an alternative to supervised fine-tuning (SFT) when a stronger teacher model is available. Classical approaches train a student to match the output distribution or representations of a teacher (Kim & Rush, 2016; Sanh et al., 2019; Team et al., 2024; Ko et al., 2024). However, standard distillation relies on fixed data and suffers from student–teacher mismatch, since the student is evaluated on its own generated trajectories (Agarwal et al., 2023; Xu et al., 2024; Chen et al., 2024). To reduce this distribution shift, recent works study on-policy distillation (Lu & Lab, 2025; Xiao et al., 2026; Yang et al., 2025; DeepSeek-AI, 2026), where the student learns from teacher feedback on its own generations. This aligns the training distribution with the student policy and is closely related to online imitation learning. Building on this idea, self-distillation uses a self-teacher that has access to privileged information to guide the current policy. The teacher may condition on environment feedback or execution signals (Hübotter et al., 2026; Ye et al., 2026), or leverage reference solutions and hints (Zhao et al., 2026; Agarwal et al., 2024). Other works explicitly study privileged information distillation in this setting (Penaloza et al., 2026; Chen et al., 2025). While these methods are restricted to isolated instance-level corrections, our approach aggregates experience across samples into a richer training signal.

Continual learning via self-reflection and experience accumulation. Recent work explores training-free continual learning by extracting reusable strategies from an LLM's own trajectories. Models can utilize verbal feedback as episodic memory (Shinn et al., 2023; Madaan et al., 2023), self-discover reasoning structures (Zhou et al., 2024; Wang et al., 2023), or maintain an evolving experience pool to dynamically inject refined strategies into future inference (Zhang et al., 2025; Yoran et al., 2023). Building on this trajectory-based memory paradigm, our method integrates experience accumulation directly into on-policy self-distillation.

6 Discussion and Conclusions

In this work, we address the challenge of continuous LLM self-improvement via environmental interactions, particularly in sparse-reward regimes. We introduce Reflection-Enhanced Self-Distillation (RESD), a framework that uses retrospective reflections and a persistent playbook to transform raw failure feedback into active, corrective supervision. Experiments on multiple tasks show that RESD improves interaction efficiency: with only a single rollout per prompt, it improves faster early in training than standard self-distillation methods and GRPO.

RESD as a feedback-enhancement module. The non-monotonic validation curves in Figure 6 highlight an important property of on-policy self-distillation: its effectiveness depends not only on the structure and reliability of the teacher context, but also on the optimization objective. Recent studies similarly observe that self-distillation can exhibit unstable dynamics (Kim et al., 2026; Li et al., 2026; Yang et al., 2026). Our results complement this line of work from a different angle. Rather than proposing a new self-distillation objective, RESD improves the information supplied to the self-teacher by converting raw failure feedback into local reflections and persistent playbook knowledge. This may explain why RESD is especially effective in the early rare-success regime, where reward-only methods obtain little signal and standard self-distillation lacks successful demonstrations. More broadly, RESD can be plugged into stronger self-distillation algorithms: sample routing can decide when to apply distillation (Li et al., 2026), reward-grounded objectives can determine update directions (Yang et al., 2026), and RESD can provide the structured feedback context that makes failed trajectories useful for token-level supervision.

AI USE STATEMENT

In this work, we used generative AI tools for "Implement methods". We have not used generative AI tools for "Help develop theoretical models or conceptual frameworks", "Formulate mathematical claims", "Propose or refine hypotheses", "Design or provide feedback on research methodology or experiments", "Support qualitative and thematic data analysis", "Interpret results" and "Generate synthetic data sets", "Provide critical ingredients for proving mathematical claims", "Assist in the writing of proofs", "Assist with translation", "Clean and reformat dataset" are not applicable to this work. Additionally, we used generative AI tools for "Edit a research paper to improve readability". We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will prepare source code, including the prompts used for REFLECT, CURATE, and teacher context construction, and will release it publicly upon publication. The RESD training loop is specified in Section 3 and Algorithm 1, with the playbook pruning strategy in Appendix G. The training protocol, evaluation metrics, and checkpoint selection rule are described in Section 4, and Appendix J lists the hardware, software, and per-task hyperparameters of RESD and the GRPO baseline. All base models (`Qwen3-4B-Thinking-2507` and `Qwen3-30B-A3B-Thinking-2507`) and datasets are publicly available; Table 1 gives the train/test split sizes, and Appendix F describes the LiveCodeBench split. Full validation curves for all runs in Table 2 are provided in Appendix C.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*, 12, 2023.

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*, 2024.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting, 2025. URL `https://arxiv.org/abs/2510.18874`.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL `https://arxiv.org/abs/2403.07974`.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of llms? *arXiv preprint arXiv:2603.24472*, 2026.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. Distillm: Towards streamlined distillation for large language models. *arXiv preprint arXiv:2402.03898*, 2024.

Song Lai, Haohan Zhao, Rong Feng, Changyi Ma, Wenzhuo Liu, Hongbo Zhao, Xi Lin, Dong Yi, Qingfu Zhang, Hongbin Liu, Gaofeng Meng, and Fei Zhu. Reinforcement fine-tuning naturally mitigates forgetting in continual post-training, 2026. URL https://arxiv.org/abs/2507.05386.

Gengsheng Li, Tianyu Yang, Junfeng Fang, Mingyang Song, Mao Zheng, Haiyun Guo, Dan Zhang, Jinqiao Wang, and Tat-Seng Chua. Unifying group-relative and self-distillation policy optimization via sample routing. *arXiv preprint arXiv:2604.02288*, 2026.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

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

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less, 2025. URL https://arxiv.org/abs/2509.04259.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Rl grokking recipe: How does rl unlock and transfer new algorithms in llms?, sep 2025. URL https://arxiv.org/abs/2509.21016.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is dpo superior to ppo for llm alignment? a comprehensive study. *arXiv preprint arXiv:2404.10719*, 2024.

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*, 2025.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. Answering questions by meta-reasoning over multiple chains of thought. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5942–5966, 2023.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Denny Zhou, Swaroop Mishra, Huaixiu S Zheng, et al. Self-discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems*, 37:126032–126058, 2024.

A Discussion on the Self-Distillation Objective

As established in Section 2, our unified self-distillation objective minimizes the expected f-divergence between the teacher distribution $\pi_{\theta_{\text{old}}}(\cdot|x, y_{<t}, c)$ and the student policy $\pi_\theta(\cdot|x, y_{<t})$. The choice of the generator function f yields qualitatively different optimization landscapes. Below we discuss three canonical instantiations and their practical implications.

Forward Kullback-Leibler (KL) Divergence. Setting $f(\tau) = \tau \log \tau$ recovers the standard distillation objective. Because the loss is an expectation weighted by the student policy π_θ yet penalizes deviations through $\tau \log \tau$, the resulting gradient is dominated by regions where the teacher assigns substantial probability mass. This produces *mean-seeking* behavior: the student is penalized heavily whenever it assigns low probability to tokens the teacher considers valid, encouraging it to cover all modes of the teacher distribution.

Reverse KL Divergence. Setting $f(\tau) = -\log \tau$ swaps the directional penalty, yielding *mode-seeking* behavior. Here the student incurs large loss when it generates tokens that the teacher deems highly improbable. In complex reasoning tasks, this property is particularly desirable because it actively suppresses logical hallucinations—tokens that are plausible under the student's distribution but inconsistent with the teacher's feedback-informed reasoning.

Jensen-Shannon Divergence (JSD). Setting $f(\tau) = \frac{\tau}{2} \log \tau - \frac{\tau+1}{2} \log\left(\frac{\tau+1}{2}\right)$ yields a smoothed, symmetric objective bounded between 0 and $\log 2$. By interpolating between the forward and reverse directions, JSD stabilizes training in regimes where the teacher and student distributions are nearly disjoint—precisely the situation that causes extreme gradient explosions under pure forward or reverse KL. This boundedness makes JSD a robust default when the quality gap between teacher and student is large or highly variable across training.

B Success-Rate Rebalancing

In the self-distillation loss, every sample in a rollout batch contributes equally by default. When the batch success rate is very low (or very high), the gradient signal is dominated by the majority

outcome, diluting the supervision from the minority. To counteract this imbalance, we weight each sample i by a scalar w_i that depends on the batch success rate $s = |\{i : R(x_i, y_i) \geq \tau\}|/B$:

$$w_i = \begin{cases} (1-s)^{\alpha} & \text{if } R(x_i, y_i) \geq \tau \quad \text{(success)}, \\ s^{\beta} & \text{otherwise} \quad \text{(failure)}, \end{cases} \tag{2}$$

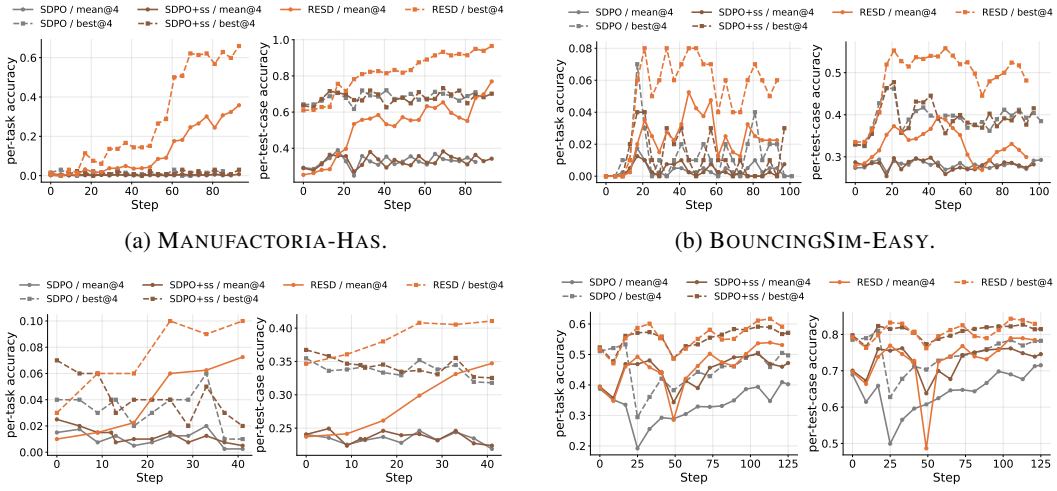
where $\alpha, \beta > 0$ are hyperparameters that control the strength of rebalancing, and τ is the success reward threshold. The weights are normalized so that their batch mean equals one, $\bar{w} = \frac{1}{B}\sum_i w_i = 1$, ensuring that the effective learning rate is unaffected. The weighted self-distillation loss becomes:

$$\mathcal{L}^{w}_{\mathrm{SD}}(\theta) = \frac{1}{B}\sum_{i=1}^{B} w_i \sum_{t=1}^{T_i}\sum_{v\in\mathcal{V}} \pi_\theta(v \mid x_i, y_{i,<t}) \cdot f\left(\tilde{\tau}^{t}_{v}\right). \tag{3}$$

When success is rare ($s \to 0$), successful samples receive high weights while failed samples receive low weights, amplifying the distillation signal from the few successful demonstrations. Conversely, when success is common, the corrective signal from failures is amplified. In all experiments we set $\alpha = \beta = 1$.

C Training Curves for Main Results

Figure 7 shows the validation training curves of the runs summarized in Table 2. Each panel plots validation accuracy over training steps for one task, comparing the baselines and RESD under the same rank-based checkpoint-selection rule described in Section 4.



(a) MANUFACTORIA-HAS. (b) BOUNCINGSIM-EASY.

(c) BOUNCINGSIM-MEDIUM. (d) FINER.

Figure 7: Validation accuracy over training steps for the runs reported in Table 2.

D Comparison under Matched Interaction and Wall-Clock Budgets

Figure 6 compares methods at equal training steps, which gives GRPO $8\times$ more environment interactions than RESD. GRPO cannot be run with a single rollout per prompt, since its group-relative advantage is zero when the group contains only one trajectory. We therefore compare methods at matched interaction budgets instead, reporting the best accuracy each method reaches within the budget. Table 5 shows that within the 2,976 interactions of a full $N{=}1$ run, GRPO does not improve over its initial accuracy, while RESD reaches 76.9. Even at the end of its own run, after 23,808 interactions ($8\times$ the budget of RESD), GRPO reaches 53.8. Among the $N{=}1$ methods, RESD is not ahead at the smallest budget: its gains begin after roughly 16 steps. From 1,024 interactions onward, RESD is ahead of SDPO and SDPO+ss, and the gap widens with the budget.

Table 5: Best per-test-case accuracy (m@4) reached on MANUFACTORIA-HAS (Qwen3-4B) within a given budget of environment interactions. N is the number of rollouts per prompt; each training step uses $32N$ interactions. 2,976 interactions correspond to the full 93-step run of the $N{=}1$ methods, or about 11 steps of GRPO.

		Environment interactions			
Method	N	≤ 512	$\leq 1{,}024$	$\leq 2{,}048$	$\leq 2{,}976$
GRPO (Shao et al., 2024)	8	29.1	29.1	29.1	29.1
SDPO (Hübotter et al., 2026)	1	34.5	39.0	39.0	39.0
SDPO+ss	1	**36.4**	36.4	37.9	38.2
RESD (ours)	1	28.3	**55.7**	**63.2**	**76.9**

Matched wall-clock time. Because RESD adds reflection and curation calls, fewer interactions do not automatically mean less training time. We therefore also compare methods at matched wall-clock time on BOUNCINGSIM-EASY, the task whose per-step latency is reported in Appendix I. All runs use the same single node with 8 H200 GPUs, so equal wall-clock time also means equal compute. We convert each evaluated checkpoint to elapsed training time by summing the per-step latencies in Figure 8 (evaluation time excluded); SDPO+ss is omitted because its latency was not measured.

Table 6: Best per-test-case accuracy (m@4) reached on BOUNCINGSIM-EASY (Qwen3-4B) within a given wall-clock training time.

		Wall-clock training time			
Method	N	$\leq$ 2h	$\leq$ 4h	$\leq$ 8h	$\leq$ 16h
GRPO (Shao et al., 2024)	8	28.0	28.0	29.8	35.8
SDPO (Hübotter et al., 2026)	1	29.3	29.6	29.6	29.6
RESD (ours)	1	**29.9**	**35.2**	**37.3**	**39.5**

RESD takes about 640s per step, compared with about 370s for SDPO and 990s for GRPO. Table 6 shows that RESD leads at every budget up to 16 hours, reaching 37.3 within 8 hours, when GRPO is at 29.8 and SDPO at 29.6. RESD's best accuracy is reached within about 8 hours and declines afterwards, while GRPO continues to improve later in its run (Figure 6(b)). This matches the regime RESD targets: it improves fastest early in training, and switching to reward-based optimization is preferable once successful rollouts become common.

E DETECTING LATE-STAGE DEGRADATION

As discussed in Section 4.4, RESD degrades on BOUNCINGSIM-EASY after around step 60, once the model reaches a higher-success regime. In deployment, a held-out validation set may not be available to detect this point. We find that the accuracy of the training rollouts, which is available online at no extra cost, tracks the validation trend closely.

Table 7: Training rollout accuracy and validation accuracy (per-test-case m@4) of RESD on BOUNCINGSIM-EASY around the late-stage degradation.

Step	45	49	57	65	69	73
Training accuracy	0.346	**0.429**	0.350	0.329	0.194	0.310
Validation accuracy	**0.395**	0.389	0.352	0.274	0.269	0.293

The two signals have a same-step correlation of 0.672 over the full run and 0.750 during the decline, and both reach their minimum at step 69 (Table 7). The validation values are used here only to verify the online signal. This suggests three practical ways to decide when to stop or switch away from RESD:

- **Monitor training accuracy.** Track a smoothed training-accuracy curve and react to a sustained decline relative to its own recent peak, rather than to a task-specific absolute threshold.
- **Monitor a small validation stream.** When a small deployment-representative validation set is available, keep the best checkpoint and stop, anneal, or switch objectives after a sustained decline.
- **Switch to a more stable objective.** RESD only changes how the teacher context is constructed, so its reflection and playbook can be kept while switching to a reward-grounded objective such as RLSD (Yang et al., 2026) or to GRPO once successful rollouts become common.

F RESULTS ON A STANDARD REASONING BENCHMARK

Our main experiments focus on tasks that are novel to the model and start in rare-success regimes. To test whether RESD remains useful when success is common, we additionally evaluate on LiveCodeBench (Jain et al., 2024), a standard competitive-programming benchmark.

Setup. We re-split the release-v6 problems into 924 training and 131 held-out problems with no overlap, and train `Qwen3-4B-Thinking-2507` with the same online streaming protocol for every method. We compare RESD and SDPO with a single rollout per prompt against GRPO with group sizes 8 and 2; 2 is the smallest group size for which GRPO's group-relative advantage is non-zero. We evaluate all 131 held-out problems with 4 rollouts each and report per-test-case accuracy (m@4). This task is not in a rare-success regime: the base model already passes 58% of test cases and solves 43% of problems.

Table 8: Results on LiveCodeBench (Qwen3-4B). We report the best held-out per-test-case accuracy (m@4) and its improvement over the base model, with 95% confidence intervals from a paired bootstrap over test problems (10,000 resamples). N is the number of rollouts per prompt.

Method	N	Best accuracy	Improvement over base [95% CI]
GRPO (Shao et al., 2024)	8	65.3	+7.4 [+3.5, +11.7]
GRPO (Shao et al., 2024)	2	66.3	+8.4 [+5.1, +12.0]
SDPO (Hübotter et al., 2026)	1	64.6	+6.5 [+2.1, +10.7]
RESD (ours)	1	**67.2**	+8.5 [+4.7, +12.3]

Results. Table 8 shows that all methods improve significantly over the base model, and RESD reaches the highest accuracy while using a single rollout per prompt. The differences between methods are within evaluation noise on this 131-problem test set: the paired differences are +0.9 [−1.7, +3.6] against GRPO (N=2), +1.9 [−1.5, +5.4] against GRPO (N=8), and +2.5 [−0.5, +5.5] against SDPO. We therefore read these results as RESD being competitive with GRPO and SDPO when success is common, rather than as a significant improvement; its large gains remain specific to rare-success regimes.

Rare-success problems. We further isolate the problems on which the base model fails all 8 sampled rollouts. We count such a problem as repaired if a method goes from no passing rollout at the first inner-loop iteration of its batch to all rollouts passing at a later iteration of the same batch. RESD repairs 4 of 32 such problems (12.5%), compared with 3 of 29 for SDPO (10.3%), 1 of 33 for GRPO with N=2 (3.0%), and 1 of 34 for GRPO with N=8 (2.9%). These counts are small, but they are consistent with the main experiments: the methods that obtain dense supervision from a single failed rollout repair rare failures more often than group-relative optimization.

Reliability of the playbook. Playbook entries are written and tagged by the model itself, so incorrect or over-generalized lessons can enter the playbook. We measure how often this happens and how quickly such entries are removed, using the first 49 training steps of the LiveCodeBench run. Reflections are used once and never stored, so only playbook entries can persist.

- **Harmful entries are pruned.** 81 entries became net-harmful ($d_j \geq h_j$); 76 of them were removed, with a median of 3 steps after turning harmful, and the remaining 5 recovered to net-helpful.

- **Stale entries are pruned.** Of the 246 entries created, 98 were removed as unused. The playbook stayed between 72 and 80 entries, below its cap of 120, so none of the 174 removals was forced by the size limit.
- **Useful lessons are concentrated and stable.** The top 10% of entries hold 95.9% of all helpful tags.
- **Lessons transfer across problems.** At least 87.2% of helpful tags were earned on problems outside the batch in which the entry was created, and outside that batch helpful tags outnumber harmful tags 7.5:1. Cross-problem interference also occurs: 24 of the 60 entries tagged by other problems became net-harmful, and 20 of these 24 were removed within the observed window.

Consistent with this, the fraction of previously solved problems that fail again at a later inner-loop iteration is 9.0% for RESD, compared with 11.1% for SDPO under the same protocol without a playbook.

Table 9: Global versus per-problem playbook on LiveCodeBench. The per-problem variant keeps a separate playbook for each problem, with no sharing across problems; everything else is identical. The per-problem run is partial (59 of 112 steps), so both variants are compared over the same 12 batches.

Metric	Global (default)	Per-problem
Best held-out per-test-case accuracy (m@4)	**67.2**	66.0
Regression rate (solved → failed) ↓	**9.0%**	13.7%
Repair rate on rare-success problems ↑	**12.5%**	6.3%

Table 9 compares the default global playbook with a per-problem playbook. Accuracy is similar, but sharing lessons across problems lowers the regression rate and doubles the repair rate on rare-success problems, where a problem has no useful history of its own.

G PLAYBOOK CONCISE STRATEGY

The CONCISE operation in Section 3.3 maintains the playbook $\mathcal{P}$ as a compact set of reusable lessons over the course of training. We describe two concrete instantiations used in our experiments and the trigger conditions under which they are applied.

Bullet categorization. At the time of each CONCISE call, every entry $p_j \in \mathcal{P}$ is classified based on its tag history. Let h_j and d_j denote the accumulated helpful and harmful counts assigned by the reflector across training steps (Section 3.3). We partition the playbook as

- **Unused:** $h_j + d_j = 0$ (never tagged),
- **Harmful:** $d_j \geq h_j$ and $d_j > 0$ (net harmful),
- **Helpful:** otherwise.

In addition, we record the most recent step τ_j at which each entry was tagged and use step $- \tau_j$ as a staleness measure.

Concise methods. Given a budget $M_{\max}$ on the playbook size, we consider two removal strategies:

- *Prioritized.* Remove all unused and harmful entries. If $|\mathcal{P}|$ still exceeds $M_{\max}$, randomly drop helpful entries until the cap is met.
- *Staleness.* Remove all harmful entries and all *stale unused* entries (unused for more than one context-update step). If $|\mathcal{P}|$ still exceeds $M_{\max}$, additionally evict the longest-unused remaining entries in order of increasing τ_j until the cap is met.

Both strategies guarantee that persistently harmful entries are removed; they differ in how they treat unused entries. The *staleness* variant is more patient in that unused entries are given one additional step to be tagged before becoming eligible for removal, which helps preserve newly curated lessons that may take a few reflections before receiving a usage signal.

Trigger conditions. CONCISE is invoked under two conditions: (i) a *frequency trigger*, which fires every F context-update steps (default $F = 4$), and (ii) a *limit trigger*, which fires whenever $|\mathcal{P}| > M_{\max}$ between scheduled invocations. In the former case, the playbook is pruned down to its harmful/stale subset without enforcing the budget; in the latter, the budget $M_{\max}$ is enforced.

Default configuration. We use the *staleness* strategy for all tasks; the $M_{\max}$ used for each task is listed in Appendix J.

H PRESERVATION OF INSTRUCTION FOLLOWING CAPABILITY

Table 10: IFEval accuracy after training on each task.

	Per-Test-Case Acc.		Per-Task Acc.	
Checkpoint	m@4	b@4	m@4	b@4
Base model	87.62	91.33	82.50	87.00
After MANUF.-HAS	87.96	91.33	83.75	87.00
After BSIM-EASY	87.83	92.33	83.25	88.00
After FINER	86.88	90.33	80.50	86.00

A common challenge in continual learning is catastrophic forgetting, where improvements on specific reasoning tasks may degrade the model's foundational instruction-following capabilities. To verify that RESD avoids this trade-off, we evaluate the models on the IFEval benchmark (Zhou et al., 2023) before and after training. As shown in Table 10, RESD successfully preserves the model's general capabilities across all evaluation settings. For instance, after training on the MANUFACTORIA-HAS task, the per-task mean@4 accuracy actually increases slightly from 82.50% to 83.75%, while the per-test-case mean@4 accuracy remains stable at around 87–88%.

I ANALYSIS ON TRAINING LATENCY

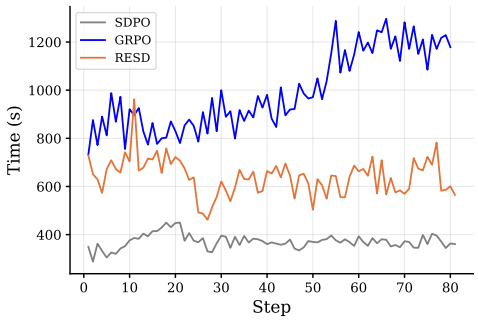


Figure 8: Per-step training latency for BOUNCINGSIM-EASY.

While RESD significantly improves interaction efficiency by requiring only a single rollout per prompt, the generation of retrospective reflections and playbook curation introduces additional inference steps. To understand the latency implications of these components, we analyze the per-step training latency of RESD compared to SDPO and GRPO baselines. As illustrated in Figure 8, standard SDPO exhibits the lowest latency about 400s per-step, because it only computes the teacher distribution and KL loss for a single trajectory. RESD introduces a moderate and stable computational overhead due to the diagnostic inference required for the reflection and curation operations. However, RESD remains substantially faster than GRPO. Because GRPO relies on group-relative optimization, it must repeatedly sample, execute, and evaluate a large batch of trajectories, causing its latency to climb above 1200s per step.

Sources of overhead. On BOUNCINGSIM-EASY, RESD takes about 640s per step, compared with about 370s for SDPO; the difference comes from the REFLECT and CURATE calls. Three design

choices keep this overhead bounded. First, failure diagnosis and curation of new entries run only on failed rollouts; for successful rollouts, the model only tags which playbook entries were helpful. Second, the playbook size is capped at $M_{\max}$ entries and pruned by CONCISE (Appendix G), and the teacher prompt length is capped (Appendix J), so the cost of each call does not grow with training length. Third, both operations are inference calls served by the same vLLM engine used for rollout generation, so they add no gradients, optimizer state, or separate model to GPU memory. Because the per-step overhead is stable, RESD still leads SDPO and GRPO at every matched wall-clock budget up to 16 hours (Appendix D).

J IMPLEMENTATION DETAILS

Infrastructure. All experiments are conducted on a single node with 8 NVIDIA H200 GPUs. Each experiment completes in approximately one day of wall-clock time. We use vLLM for rollout generation with tensor parallelism across 4 GPUs, and FSDP for distributed training. Rollout generation uses temperature 1.0 with top-p sampling at 0.95. All models have thinking mode enabled.

Training. We train for a single epoch over the training data (online streaming). Each batch undergoes $K = 4$ inner-loop update iterations. The batch size is 32 and we use a single rollout per prompt (N=1) for all RESD experiments. We apply a learning rate warmup of 10 steps. For validation, we sample 4 responses per test problem to compute mean@4 and best@4 metrics.

Self-distillation. The self-distillation loss is computed over the top-100 tokens in the student's vocabulary at each position. Success-rate rebalancing (Appendix B) is applied on BOUNCINGSIM-EASY and FINER.

Context updater (RESD-specific). The playbook is maintained globally (shared across all training examples). Playbook entries are tagged as helpful or harmful during reflection on both failed and successful samples. The solution buffer caches successful trajectories for future teacher prompt construction.

Per-task hyperparameters. All tasks use the staleness-based CONCISE strategy (Appendix G). Unless otherwise noted, we use EMA update rate 0.0001, lower clip $\epsilon_{\min}$=0.2, and no upper clip. Task-specific deviations are as follows:

- MANUFACTORIA-HAS: upper clip $\epsilon_{\max}$=5, max playbook entries $M_{\max}$=200, max prompt/response length 49152/20480 tokens.
- BOUNCINGSIM-EASY: max playbook entries $M_{\max}$=120, success-rate reweighting enabled, max prompt/response length 58368/25600 tokens.
- BOUNCINGSIM-MEDIUM: EMA update rate 0.01, max playbook entries $M_{\max}$=150, max prompt/response length 58368/25600 tokens.
- FINER: max playbook entries $M_{\max}$=120, success-rate reweighting enabled, max prompt/response length 49152/20480 tokens.

GRPO baseline. For GRPO, we use a rollout group size of 8 per prompt with the same learning rate (1e-6), batch size (32), and inner-loop steps (K=4). GRPO does not use self-distillation, context updating, or EMA.

K CASE STUDY: HOW RESD INTERPRETS FAILURE FEEDBACK

Table 11 traces RESD on a representative MANUFACTORIA-HAS task across four consecutive inner-loop updates, illustrating how local environment feedback is progressively organized into reusable task knowledge. This case highlights two mechanisms that are difficult to see from aggregate accuracy alone. *First,* RESD *exposes learning progress before sparse success appears*: the model fixes syntax, then transition handling, then a more subtle edge case involving empty-tape behavior. *Second, the playbook accumulates situation-specific rules*: different failure modes require different corrections (e.g., skipping non-pattern symbols vs. rejecting when the tape is empty), and the playbook records each rule alongside the situation it applies to. A cached successful exemplar ultimately demonstrates

how these rules coexist in a complete working program. This supports the view that failure feedback becomes useful not merely by being appended to the prompt, but by being organized into persistent, scoped knowledge.

Table 11: Case study on MANUFACTORIA-HAS: "Accept if the tape contains BRBR." In this task, the model must write a finite-state machine program in a domain-specific language that processes an input tape—a sequence of colored symbols (e.g., `GRBRBRY`, `BRGBY`)—one symbol at a time from left to right. Each state reads the next symbol and branches to a successor state depending on the symbol; a tape is accepted by reaching a designated accept state and rejected otherwise. Here, the target program must accept any tape containing the contiguous substring BRBR (e.g., accept `GRBRBRY`, reject `BRGBY`), requiring the model to handle arbitrary surrounding symbols, restart matching after partial mismatches, and correctly reject when the input ends mid-match. "acc." denotes per-test-case accuracy. Each row corresponds to one inner-loop update step, showing how RESD progressively diagnoses and corrects errors through reflection and playbook accumulation. Full generated programs for each step are provided in Appendix K.1.

Step	acc.	Failure reason	Reflection / playbook update
45	0.00	Program references a non-existent state, causing a parse error	Reflection extracts a syntax rule: all branch targets must reference states that are explicitly declared in the program.
46	0.84	Incorrectly rejects tapes whose leading symbols are not part of the pattern (e.g., rejects `GBRBR` because it starts with `G`, even though BRBR appears later)	Reflection identifies that state `s0` routes a leading `G` to reject instead of continuing the search. Playbook records: non-pattern symbols before or between matches should be skipped, not rejected.
47	0.96	Overcorrection: the fix for skipping non-pattern symbols introduces an infinite loop between two states when the tape becomes empty at a non-terminal state	Reflection identifies that non-pattern symbols should be skipped when the tape is non-empty, but the program should reject when the tape is empty at a non-terminal state. The playbook records both situation-specific rules.
48	1.00	—	A successful solution is cached.

K.1 FULL PROGRAM OUTPUTS

We provide the complete DSL programs generated at each inner-loop step for the case study above ("Accept if the tape contains BRBR"). Each listing corresponds to one row of Table 11. We highlight the buggy lines with inline comments.

Step 45 (acc. = 0.00): Parse error from undefined node reference. The program references `state0` in a `PULLER_YG` node, but `state0` is never declared as a node, causing a parse error.

```
START start:
    NEXT node_rb

PULLER_RB node_rb:
    [B] node_b1
    [R] node_rb
    [EMPTY] node_yg

PULLER_YG node_yg:
    [Y] node_rb
    [G] node_rb
    [EMPTY] state0       # BUG: 'state0' is never declared

PULLER_RB node_b1:
    [R] node_b1r1
    [B] node_b1
    [EMPTY] node_yg

...

END end
```

Step 46 (acc. = 0.84): Correct pattern matching logic, but rejects non-R/B characters. The state machine correctly implements substring matching transitions for B and R. However, all `[EMPTY]` branches route to `NONE`, which causes immediate rejection when a non-R/B character (Y or G) is encountered—e.g., input `GGYYYBBRBR` is rejected at state `s0` because the leading `G` triggers `[EMPTY] NONE`.

```
START start:
    NEXT s0

PULLER_RB s0:
    [B] s1
    [R] s0
    [EMPTY] NONE          # BUG: rejects Y/G characters

PULLER_RB s1:
    [R] s2
    [B] s1
    [EMPTY] NONE          # BUG: same issue

PULLER_RB s2:
    [B] s3
    [R] s0
    [EMPTY] NONE          # BUG: same issue

PULLER_RB s3:
    [R] end
    [B] s1
    [EMPTY] NONE          # BUG: same issue

END end
```

Step 47 (acc. = 0.96): Overcorrection introduces an infinite loop. The model adds a `PULLER_YG` node to handle non-R/B characters, fixing the step 46 bug. However, when the tape becomes empty at a non-terminal state, `PULLER_RB` routes to `state0_yg`, whose `[EMPTY]` routes to `NONE`—but if there are remaining characters that alternate between the two node types, they bounce back and forth infinitely.

```
START start:
    NEXT state0

PULLER_RB state0:
    [B] state1
    [R] state0
    [EMPTY] state0_yg     # Routes non-R/B to PULLER_YG

PULLER_RB state1:
    [R] state2
    [B] state0
    [EMPTY] state0_yg

PULLER_RB state2:
    [B] state3
    [R] state0
    [EMPTY] state0_yg

PULLER_RB state3:
    [R] end
    [B] state0
    [EMPTY] state0_yg

PULLER_YG state0_yg:
    [Y] state0
    [G] state0
    [EMPTY] NONE          # BUG: causes infinite loop when tape
                          # is empty between state0 and state0_yg

END end
```

Step 48 (acc. = 1.00): Correct solution. The final program correctly handles all cases: non-pattern symbols are skipped via dedicated `PULLER_YG` nodes, and empty-tape transitions at non-terminal states correctly reject. A cached successful solution from a prior rollout provided the complete exemplar that guided this output.

```
START start:
    NEXT s0

PULLER_RB s0:
    [B] s1
    [R] s0
    [EMPTY] s0_yg

PULLER_YG s0_yg:
    [Y] s0
    [G] s0
    [EMPTY] NONE

PULLER_RB s1:
    [R] s2
    [B] s1
    [EMPTY] s1_yg

PULLER_YG s1_yg:
    [Y] s1
    [G] s1
    [EMPTY] NONE

PULLER_RB s2:
    [B] s3
    [R] s0
    [EMPTY] s2_yg

PULLER_YG s2_yg:
    [Y] s2
    [G] s2
    [EMPTY] NONE

PULLER_RB s3:
    [R] end
    [B] s1
    [EMPTY] s3_yg

PULLER_YG s3_yg:
    [Y] s3
    [G] s3
    [EMPTY] NONE

END end
```