

Counterfactual Rollout Replay: Forkable Environments as Free Process Rewards for Software Engineering Agents

Yuanhao Li¹ Hongbo Wang^{1*} Xuhong Chen¹
Yiming Cao¹ Xunzhu Tang²

¹State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing 100876, China

²University of Luxembourg, Luxembourg

Abstract

Outcome-only reinforcement learning gives software engineering (SWE) agents a terminal success signal but little direct guidance about intermediate decisions. We introduce Counterfactual Rollout Replay (CRR), a training-time procedure that uses forkable executable environments to obtain step-level return contrasts. CRR selects a small set of decision points, restores each state, samples an alternative action, and rolls the branch forward under the policy. It retains the realised training trajectory and replaces the advantage at selected steps with the difference between its terminal return and the sampled counterfactual return. The method needs no human process labels or learned process reward model; *free* refers to those supervision costs, not replay compute. With a 14B policy, CRR improves pass@1 on SWE-bench Verified, SWE-bench Live, and SWE-rebench, and combines with process-reward and trajectory-search methods. On SWE-bench Verified, an equal-wall-clock comparison on the same hardware yields 41.7% versus 36.7% for extended outcome-only GRPO, a 5.0-point gain with fork overhead included. These results apply to environments with affordable, reliable state restoration; stochastic continuations and expensive or imperfect replay remain limitations.

1 Introduction

Software engineering agents solve repository-level issues through sequences of file inspection, code edits, shell commands, and tests. Benchmarks such as SWE-BENCH [Jimenez et al., 2024] and executable training environments such as SWE-GYM [Pan et al., 2025] and SWE-REBENCH [Badertdinov et al., 2025, 2026] make terminal-verifier reinforcement learning practical. PPO- and GRPO-style updates [Schulman et al., 2017, Shao et al., 2024] can improve task success, but a trajectory-level advantage does not distinguish a useful localisation decision from a later edit that breaks the patch. Learned process rewards and search provide alternative sources of guidance [Cui et al., 2025, Han et al., 2026, Peng et al., 2026, Djuhera et al., 2026]. We investigate whether executable alternatives can instead provide a training-time credit signal.

Forkability is shared with games and simulators; it is not itself our contribution. Its useful feature in SWE is access to a restorable repository and executable environment with auditable provenance. With sandbox infrastructure [Pan et al., 2025, Wang et al., 2025a, Yuan et al., 2026], an alternative action can be followed through the real task environment and evaluated by its verifier, without learning a transition model. This opportunity depends on reliable restoration and an affordable replay cost.

*Corresponding author: hbwang@bupt.edu.cn.

Counterfactual Rollout Replay (CRR) selects a small number of decision points from an on-policy trajectory, forks their states, and samples alternative actions and continuations. It retains the realised trajectory for training and uses the difference between realised and counterfactual terminal returns as the advantage at selected steps. Unlike selecting a verifier-best branch for training, the fork changes how the realised action is scored. *Free* means that CRR needs no human process labels, learned process reward model, or oracle hindsight annotation; every replay still consumes compute. Environment determinism also leaves future policy-sampling variance intact.

Our contributions are:

- **Executable contrast estimation.** CRR supplies an environment-grounded selected-step signal and integrates it into the PPO/GRPO update while retaining the realised trajectory.
- **Scoped analysis.** The estimator is unbiased for a proposal-dependent, action-conditional contrast at a fixed index. The implemented trajectory-level selector induces a selector-weighted objective, and the variance decomposition provides a diagnostic condition rather than a universal guarantee.
- **Controlled evaluation.** With a shared 14B backbone, CRR improves pass@1 across SWE-BENCH VERIFIED, SWE-BENCH LIVE, and SWE-REBENCH, and combines with PRM and trajectory-search training. A separate equal-wall-clock comparison on Verified retains a 5.0-point gain over extended vanilla GRPO with replay overhead charged. A matched Search-Select control tests the two uses of fork data under a reduced budget.

2 Related Work

Repository-level SWE agents and benchmarks. Tool-using agents such as ReAct established the reasoning-and-acting loop for language models [Yao et al., 2023], and SWE-BENCH reframed this loop as repository-level GitHub issue resolution [Jimenez et al., 2024]. Subsequent systems improved the agent-computer interface, search pipeline, and multi-agent scaffold: SWE-AGENT tailored file editing and test execution to autonomous agents [Yang et al., 2024]; AUTOCODEROVER, MAGIS, LINGMAAGENT, HYPERAGENT, and OPENHANDS developed increasingly capable repository-level scaffolds and platforms [Zhang et al., 2024, Tao et al., 2024, Ma et al., 2025a, Phan et al., 2024, Wang et al., 2025a]. Benchmarking has also moved toward cleaner and fresher evaluation through SWE-BENCH VERIFIED [OpenAI, 2024], SWE-BENCH LIVE [Zhang et al., 2025], SWE-BENCH+ [Aleithan et al., 2024], multilingual or re-collected SWE tasks [Badertdinov et al., 2025, 2026, Zan et al., 2024], and adversarial or audit-style reassessments [Wang et al., 2025b, Yu et al., 2026]. CRR is complementary to this line: it changes the training-time advantage estimator rather than the scaffold, benchmark, or verifier.

RLVR training for software engineering agents. Recent SWE-agent training systems move beyond prompting by optimising code policies against executable feedback. RLEF grounds code policies in execution feedback [Gehring et al., 2025], SWE-GYM provides thousands of executable Python tasks for training agents and verifiers [Pan et al., 2025], and recent pipelines such as SORFT, SWE-FIXER, SWE-LEGO, MURPHY, SKYRL-AGENT, SWE-DEV, and SWE-MINISANDBOX refine SFT, RL objectives, and infrastructure efficiency [Ma et al., 2025b, Xie et al., 2025, Tao et al., 2026, Ekbote et al., 2025, Cao et al., 2025, Du et al., 2025, Yuan et al., 2026]. The underlying optimisation machinery follows PPO-style clipped policy gradients [Schulman et al., 2017] and GRPO-style group-relative advantages introduced for mathematical reasoning and scaled in RLVR systems [Shao et al., 2024, Guo et al., 2025]. These pipelines make training possible, but their reward signal remains sparse and trajectory-level; CRR targets this credit-assignment bottleneck.

Process rewards and multi-turn credit assignment. One response to sparse terminal rewards is to train or infer dense process feedback. Process-supervised reward models [Lightman et al., 2024], implicit rewards such as PRIME [Cui et al., 2025], rubric-based SWE PRMs [Han et al., 2026], execution-free feedback models [Shum et al., 2025], and self-evolved reasoning rewards [Guan et al., 2025] all provide denser supervision than a terminal pass bit. A parallel line imposes structure on long-horizon trajectories through hierarchical agent RL [Peng et al., 2026, Zhou et al., 2024], hindsight or belief-update credit assignment [Tan et al., 2026, Auzina et al., 2026], and trajectory-search rollouts [Djuhera et al., 2026]. CRR differs mechanistically: it does not learn a process reward

model, infer credit from text, or select the best branch. It executes a counterfactual branch in the real SWE environment and uses the observed return differential as a training-time advantage signal.

Counterfactual and causal credit assignment. Counterfactual credit assignment has a pre-LLM history in RL. Future-conditional baselines separate skill from luck by learning hindsight-conditioned value functions [Mesnard et al., 2021], while counterfactual data augmentation and causal approaches manufacture alternative transitions under structural assumptions [Pitis et al., 2020, Lu et al., 2020]. Counterfactual reasoning has also been studied in multi-agent communication and information-theoretic credit assignment [Vanneste et al., 2020, Arumugam et al., 2021], and recent LLM-agent work adapts contextual counterfactual credit to multi-agent collaboration [Chen et al., 2026]. These works generally learn, model, or infer counterfactual outcomes; CRR exploits the special forkability of containerised SWE tasks to execute selected counterfactuals directly.

Forking, sandboxing, and inference-time search. The environmental property we use is not forkability in the abstract: games and simulators may also restore state, but SWE repositories can be restored cheaply, exactly, and auditably from a commit and executable environment. Containerised SWE infrastructures such as SWE-GYM, OPENHANDS, and SWE-MINISANDBOX make this practical during training [Pan et al., 2025, Wang et al., 2025a, Yuan et al., 2026], and CRR can be viewed as a Dyna-style use of the real environment as the model rather than a learned simulator [Sutton, 1991]. Inference-time tree search and test-time scaling for coding agents also branch environments, but they use branches to choose better completions or rollout continuations [Koh et al., 2025, Kim et al., 2026, Zhu et al., 2025, Barnes, 2026, Djuhera et al., 2026]. CRR uses forks differently: a branch is replayed to estimate what the realised action contributed relative to a plausible alternative, not to replace the behaviour policy’s chosen action.

Tree-based training and uncertainty-guided branching. Branching also supports training-time credit assignment: Tree-GRPO shares rollout prefixes and combines intra-tree and inter-tree group-relative advantages derived from outcome rewards [Ji et al., 2026]. This is distinct from inference-time best-first search for selecting action trajectories [Koh et al., 2025] and from Entropy-Guided Branching, which replays alternative actions at high-entropy steps to repair failed tool-use trajectories [Wei et al., 2026]. CRR instead retains the realised on-policy trajectory and uses environment-executed alternative-action contrasts to score selected decisions during training. Its contribution is therefore this credit-assignment construction in forkable SWE environments, rather than branching or entropy-based selection alone.

3 Preliminaries

We model issue resolution as a finite-horizon Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, H)$. State s_t includes the repository working tree, issue, conversation history, and tool observations. Actions are structured tool calls for file inspection, editing, shell execution, and testing [Yang et al., 2024]; executing a call in the sandbox induces $P(s_{t+1} \mid s_t, a_t)$ [Pan et al., 2025, Yuan et al., 2026]. We assume deterministic transitions for the admitted training tasks and describe the consistency filter and residual nondeterminism in Section F. The policy π_θ samples a trajectory $\tau = (s_0, a_0, \dots, s_T, a_T)$ with $T \leq H$ (default $H = 50$). Its terminal reward $R(\tau) \in \{0, 1\}$ records whether the final patch passes the held-out tests.

For expected return $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$ and an action-independent baseline b , the policy-gradient identity is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T (R(\tau) - b) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]. \quad (1)$$

For the GRPO comparison [Shao et al., 2024, Guo et al., 2025], we write the centred outcome advantage as $\hat{A}_t^{\text{GRPO}} = R(\tau) - \bar{R}^{(G)}$, where $\bar{R}^{(G)}$ is the mean return of G sibling trajectories from the same initial state. This scalar is shared across the realised trajectory. We do not assert that substituting a group mean that includes the realised return into (1) preserves an unbiased gradient estimator.

A sandbox is *forkable at horizon h* if it supports a snapshot $\sigma : \mathcal{S} \rightarrow \Sigma$ and restore $\rho : \Sigma \rightarrow \mathcal{S}$ such that executing any fixed sequence of h actions from $\rho(\sigma(s_t))$ has the same trajectory distribution as

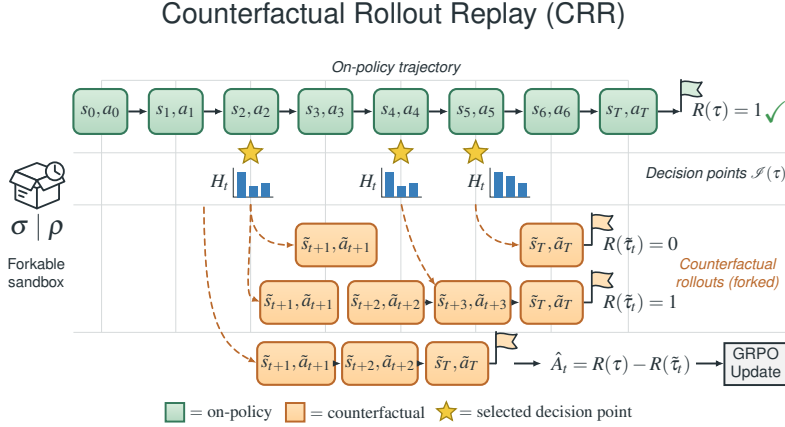


Figure 1: CRR retains the realised trajectory (green), forks selected decision points, and samples alternative continuations (orange). Their terminal-return differences replace outcome-only advantages at selected indices; the branches are not substituted for the realised training trajectory.

executing it from s_t . In the deterministic case, the resulting states are identical. Games and simulators can also satisfy this definition. The SWE setting provides a restorable executable repository state; whether snapshotting and replay are cheap enough to help training is an empirical resource question (Section 5.3).

A counterfactual trajectory $\tilde{\tau}_t$ shares τ 's prefix through s_t , samples $\tilde{a}_t \sim q_{\neg a_t}(\cdot | s_t)$ with $\tilde{a}_t \neq a_t$, and then continues with independently sampled actions from π_θ . When the policy assigns near-unit probability to one action, we skip that index rather than force an out-of-support alternative. Even with deterministic transitions, the sampled continuations and terminal returns remain random.

4 Counterfactual Rollout Replay

4.1 The CRR Advantage Estimator

Given an on-policy trajectory τ , the selector chooses decision indices $\mathcal{I}(\tau) \subseteq \{0, \dots, T\}$. For each $t \in \mathcal{I}(\tau)$, CRR restores s_t , samples $\tilde{a}_t \sim q_{\neg a_t}(\cdot | s_t)$, and completes an independently sampled on-policy continuation. The selected-step advantage is

$$\hat{A}_t^{\text{CRR}} = R(\tau) - R(\tilde{\tau}_t). \quad (2)$$

CRR retains the realised trajectory rather than replacing it with a better branch. At unselected steps it uses the outcome-only advantage, giving

$$\hat{A}_t^* = \begin{cases} R(\tau) - R(\tilde{\tau}_t) & t \in \mathcal{I}(\tau), \\ R(\tau) - \bar{R}^{(G)} & \text{otherwise.} \end{cases} \quad (3)$$

We insert this hybrid advantage into the clipped surrogate:

$$\mathcal{L}_{\text{CRR}}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \min \left(r_t(\theta) \hat{A}_t^*, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^* \right) \right] - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \quad (4)$$

where $r_t(\theta) = \pi_\theta(a_t | s_t) / \pi_{\theta_{\text{old}}}(a_t | s_t)$ and β is the KL coefficient. This retains the PPO/GRPO update structure [Schulman et al., 2017, Shao et al., 2024], without implying a new stability or convergence guarantee. The fixed-index contrast below is unbiased, but selecting indices from an entire realised trajectory induces data-dependent weighting. The implemented objective is selector-weighted, not an unbiased estimator of the unweighted per-step GRPO objective.

4.2 Properties

Proofs appear in Section A.

Proposition 1 (Fixed-index action-conditional contrast). *Assume a deterministic sandbox forkable at horizon $H - t$ and a proposal $q_{\neg a_t}(\cdot \mid s_t)$ supported on alternative actions $\mathcal{A} \setminus \{a_t\}$. For a fixed index t and action-augmented history $h_t^+ = (s_0, a_0, \dots, s_t, a_t)$,*

$$\mathbb{E}[\hat{A}_t^{CRR} \mid h_t^+] = Q^{\pi_\theta}(s_t, a_t) - \mathbb{E}_{\tilde{a}_t \sim q_{\neg a_t}}[Q^{\pi_\theta}(s_t, \tilde{a}_t)].$$

The expectation includes the counterfactual action and future policy-sampling randomness in both continuations. This is an action-conditional contrast for the specified proposal. Only the idealised variant allowing $\tilde{a}_t = a_t$ with $q = \pi_\theta$ recovers the classical advantage by reducing the second term to $V^{\pi_\theta}(s_t)$.

Proposition 2 (Variance decomposition against an outcome baseline). *Write $X = R(\tau)$, $Y = R(\tilde{\tau}_t)$, and let $\bar{Y} = \bar{R}^{(G)}$ be the sibling-group baseline. For any conditioning variable or bucket B with finite second moments,*

$$\begin{aligned} \text{Var}[X - Y \mid B] &= \text{Var}[X - \bar{Y} \mid B] \\ &= \text{Var}[Y \mid B] - \text{Var}[\bar{Y} \mid B] - 2\text{Cov}(X, Y \mid B) + 2\text{Cov}(X, \bar{Y} \mid B). \end{aligned}$$

CRR therefore has lower conditional variance than the outcome baseline in buckets satisfying

$$\text{Cov}(X, Y \mid B) > \frac{1}{2}(\text{Var}[Y \mid B] - \text{Var}[\bar{Y} \mid B]) + \text{Cov}(X, \bar{Y} \mid B).$$

The decomposition gives a diagnostic condition, not universal variance reduction. Positive return covariance alone is insufficient. The task/prefix-bucket measurements and advantage-SNR diagnostic in Section C do not independently establish every term in this inequality.

4.3 Decision-Point Selector

We select at most k non-terminal decisions per trajectory (default $k = 4$). We approximate H_t by mean grammar-masked next-token entropy over semantic tokens of the emitted tool call. The tool-type score is 0 for passive read/search, 0.25 for focused tests, 0.75 for shell commands, and 1.0 for file edits; terminal submit actions are excluded. Define

$$\text{score}(t) = H_t(1 + \lambda_{\text{type}} \text{type}(a_t)). \quad (5)$$

The selector takes the top- k scores among steps with $H_t > \eta$, using fewer forks if fewer steps qualify. It combines uncertainty with a preference for state-changing actions. For ablations, Random samples up to k indices uniformly without replacement from the same eligible set; Stride selects approximately evenly spaced time-ordered quantiles of that set. All use the same proposal and fork cap.

4.4 Counterfactual Proposal

We use a tempered policy proposal excluding the realised action:

$$q_{\neg a_t}(a \mid s_t) \propto \pi_\theta(a \mid s_t)^{1/T_{\text{cf}}} \mathbf{1}[a \neq a_t], \quad T_{\text{cf}} > 0.$$

The default $T_{\text{cf}} = 0.7$ balances alternative-action diversity against plausible continuations. The reported sweep is broadly insensitive over $[0.5, 1.0]$. The zero-contrast control displayed at $T_{\text{cf}} = 0$ is a distinct control outside this positive-temperature proposal family; it is not its zero-temperature limit.

4.5 Implementation

The implementation uses SKYRL-AGENT’s asynchronous rollout framework [Cao et al., 2025] and SWE-MINISANDBOX [Yuan et al., 2026]. Algorithm 1 gives the training loop. On-policy generation and counterfactual replay run in logical worker pools on the same node, synchronising before the update. The default uses $G = 8$ sibling trajectories and up to $k = 4$ selected steps per trajectory, with one counterfactual continuation per selected step. Section 5.3 separates trajectory counts from total device-time and reports the equal-wall-clock comparison with all fork overhead charged.

Table 1: Full-budget resource totals for the three central training comparisons. Device-time includes on-policy rollout, counterfactual replay or PRM scoring, and learner updates. These settings do not have equal total compute. Wall-clock values are rounded to one decimal place.

	GRPO	SWE-Trace	CRR
Device-time (H100-hours)	188	233	321
Node wall-clock (hours)	23.5	29.1	40.1
Verified pass@1 (%)	36.4 ± 0.8	39.1 ± 0.7	41.7 ± 0.6

5 Experiments

5.1 Experimental Setup

Model and infrastructure. We train the open-weight QWEN3-14B backbone [Yang et al., 2025] with rank-128 LoRA adapters, using SKYRL-AGENT [Cao et al., 2025] for asynchronous dispatch and SWE-MINISANDBOX [Yuan et al., 2026] for execution. Training uses one node with eight H100 80GB GPUs; fork workers are logical roles whose device-time is charged to this node. The default setting uses $k = 4$ selected steps, one counterfactual per step, $T_{cf} = 0.7$, and learning rate 5×10^{-6} . Full-budget runs collect 24,000 on-policy trajectories and perform 4,000 optimizer updates. Sections D and E provide hyperparameters, prompts, and tools.

Data and evaluation. The post-filter training corpus contains 3,638 deterministic instances: 2,438 admitted SWE-GYM tasks [Pan et al., 2025] and 1,200 admitted SWE-REBENCH tasks [Badertdinov et al., 2025]. These counts follow the three-rerun filter (Section F). The SWE-REBENCH training/evaluation split precedes filtering and excludes overlap by task identity, issue/PR URL, collection month, repository commit window, and exact or near-duplicate issue/patch content. We evaluate greedy pass@1 with a 50-turn cap on SWE-BENCH VERIFIED [OpenAI, 2024], SWE-BENCH LIVE [Zhang et al., 2025], and held-out SWE-REBENCH Python tasks. Main results are means and standard deviations across three random seeds.

Baselines and resource control. Our same-backbone, same-scaffold reproductions cover zero-shot agents [Yang et al., 2024, Wang et al., 2025a], SFT [Pan et al., 2025, Xie et al., 2025], outcome-only and multi-turn RL [Gehring et al., 2025, Ekbote et al., 2025, Cao et al., 2025], process supervision [Han et al., 2026], and trajectory search [Djuhera et al., 2026]. Matching on-policy trajectories does not match total compute. For the central 24,000-trajectory comparison, GRPO, SWE-TRACE, and CRR consume 188, 233, and 321 H100-hours, respectively (Table 1). Table 2 therefore reports endpoint performance, while the separate matched-wall-clock experiment measures resource efficiency. We do not infer equal-compute gains on all three benchmarks from the endpoint table.

5.2 Main Results

CRR reaches 41.7%, 35.9%, and 32.6% on Verified, Live, and SWE-rebench (Table 2), exceeding vanilla GRPO by 5.3, 4.8, and 4.7 percentage points. The corresponding differences from SWE-TRACE are 2.6, 2.2, and 2.4 points. CRR also improves over TSR+GRPO, while combining CRR with TSR or a PRM raises performance further. The 44.9% Verified result uses CRR plus SWE-TRACE; the full CRR+TSR+PRM combination reaches 45.6% (Table 4).

5.3 Sample Efficiency and Fully Charged Training Time

Figure 2 separates two resource axes. On the left, CRR matches or exceeds the full-budget GRPO pass-rate within 6,000 on-policy trajectories, compared with 24,000 for GRPO. This is a conservative fourfold reduction in the counted on-policy trajectories; counterfactual suffixes are excluded, so this is not a fourfold compute saving.

The full accounting charges 164 H100-hours for on-policy rollout, 133 for the counterfactual pool, and 24 for learner updates: 321 H100-hours, or 40.125 hours on eight H100 GPUs (Table 14). Overlap between fork workers and generation is not deducted from device-time. The right panel compares continuing runs on identical hardware at equal elapsed training time. At 40.125 hours, CRR

Table 2: Pass@1 (%) with the same 14B backbone and scaffold. Values are means $\pm$ seed standard deviations over three seeds. Bold denotes the best non-stacked result; underlining denotes the best result within this table. The final column is the Verified difference from vanilla GRPO. These endpoint runs are not uniformly compute-matched.

Method	SWE-bench Verified	SWE-bench Live	SWE-rebench	Δ vs. GRPO
SWE-AGENT (zero-shot)	22.6 $\pm$ 1.4	18.9 $\pm$ 1.5	16.4 $\pm$ 1.6	−13.8
OPENHANDS (zero-shot)	23.4 $\pm$ 1.4	19.7 $\pm$ 1.5	17.0 $\pm$ 1.6	−13.0
SWE-GYM SFT	28.1 $\pm$ 1.2	24.0 $\pm$ 1.3	21.5 $\pm$ 1.4	−8.3
SWE-FIXER	30.7 $\pm$ 1.1	25.8 $\pm$ 1.3	22.8 $\pm$ 1.4	−5.7
RLEF (outcome-only PPO)	33.5 $\pm$ 1.0	28.6 $\pm$ 1.2	25.4 $\pm$ 1.3	−2.9
MURPHY	35.2 $\pm$ 0.9	30.0 $\pm$ 1.1	26.7 $\pm$ 1.2	−1.2
SKYRL-AGENT (vanilla GRPO)	36.4 $\pm$ 0.8	31.1 $\pm$ 1.1	27.9 $\pm$ 1.2	0.0
SWE-TRACE (rubric PRM)	39.1 $\pm$ 0.7	33.7 $\pm$ 1.0	30.2 $\pm$ 1.1	+2.7
TSR + GRPO	38.0 $\pm$ 0.8	32.5 $\pm$ 1.0	29.1 $\pm$ 1.1	+1.6
CRR (ours)	41.7 $\pm$ 0.6	35.9 $\pm$ 0.9	32.6 $\pm$ 1.0	+5.3
CRR + TSR	43.6 $\pm$ 0.5	37.6 $\pm$ 0.8	34.1 $\pm$ 0.9	+7.2
CRR + SWE-TRACE	<u>44.9 $\pm$ 0.4</u>	<u>38.8 $\pm$ 0.8</u>	<u>35.4 $\pm$ 0.9</u>	+8.5

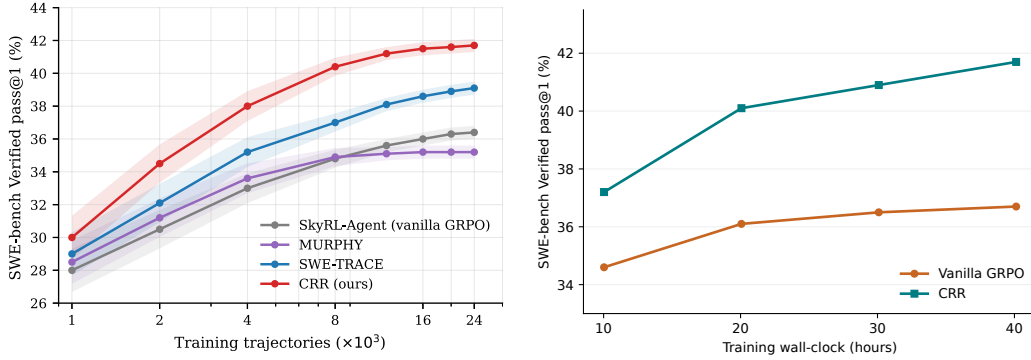


Figure 2: Verified performance against two distinct budgets. Left: newly collected on-policy trajectories, excluding counterfactual suffixes. Right: elapsed training time on the same $8 \times H100$ node with replay overhead fully charged. Right-panel checkpoints are observed three-seed means; per-checkpoint uncertainty bands are not available.

reaches 41.7% versus extended GRPO’s 36.7%, a 5.0-point advantage. GRPO has then completed 37,824 trajectories and 6,304 updates, compared with CRR’s 24,000 and 4,000. Timing includes forks, tests, waits, retries, optimization, and checkpoints; the complete ledger and timing exclusions appear in Table 15.

Operating range and signal diagnostics. A reduced-budget sweep allocates 48 H100-hours per arm and evaluates all 500 Verified instances with one seed. Multiplying snapshot/restore/replay cost by $m = 1, 2, 4$ gives CRR 36.2, 34.8, and 33.2%, versus GRPO’s 33.4%. The observed advantage disappears between $m = 2$ and $m = 4$; this is a measured interval, not a general threshold. Advantage-SNR and task/bucket covariance diagnostics are reported in Section C and Figure 4. They characterize the measured signals without proving universal variance reduction or per-step causality.

5.4 Credit Assignment versus Search-Select

To distinguish credit assignment from selecting better branches, we compare CRR with a Search-Select-GRPO control. Both use the same selector, $k = 4$ forks per realized trajectory, verifier, horizon, group size, and clipped learner. Search-Select replaces a realized training sequence only when a fork has strictly higher verifier return; ties retain the realized sequence. It then applies ordinary GRPO, discarding unused branches. CRR retains the realized sequence and uses the return differential for credit assignment. Under equal device-time, CRR is higher in all three paired seeds (Table 3),

Table 3: Two uses of matched fork executions. Every arm receives 48 H100-hours per seed; greedy evaluation covers all 500 Verified instances. Values are means $\pm$ seed standard deviations over three seeds.

Method	Fork use	Pass@1 (%)
Vanilla GRPO	None	33.3 ± 0.7
Search-Select-GRPO	Sequence selection	34.4 ± 0.8
CRR	Credit assignment	36.2 ± 0.6

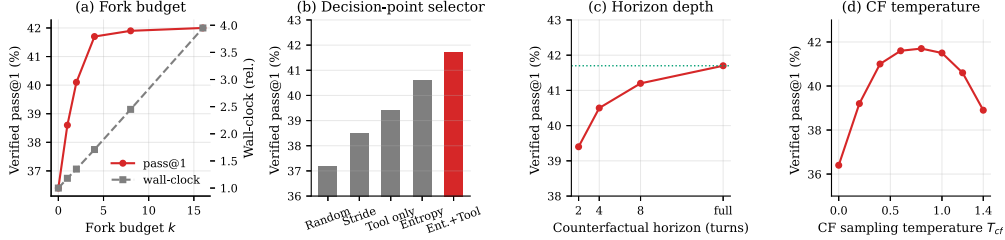


Figure 3: Verified ablations: (a) fork budget and its relative wall-clock cost, not a matched-time comparison; (b) selector; (c) counterfactual horizon; (d) proposal temperature, with a separate zero-contrast control plotted at zero.

supporting this mechanism distinction within the tested protocol. Section P gives the branch-selection and log-probability accounting.

5.5 Ablations and Composition

Figure 3 examines four design choices. Increasing k from zero to one gains 2.2 points; $k = 4$ recovers 5.3 of the 5.6-point gain at $k = 16$, with additional wall-clock cost. The entropy-plus-tool-type selector improves over either component alone. For truncated horizons $h \in \{2, 4, 8\}$, the verifier evaluates the fork’s current working tree; full-horizon rollouts continue to submission or the 50-turn cap. Performance is relatively stable for $T_{cf} \in [0.5, 1.0]$ and degrades at higher temperatures. The point plotted at zero is a separate zero-contrast control outside the positive-temperature proposal family. Section G covers concurrency and the PPO outer loop.

Table 4 shows gains when CRR is combined with TSR and PRM supervision. Adding CRR raises the four configurations by 5.3, 5.6, 5.8, and 5.6 points, respectively. The full stack reaches 45.6%. These endpoint comparisons demonstrate compatible mechanisms; they do not establish a significance claim or equal-compute advantage for the stacked systems.

5.6 Sampling Consistency and Counterfactual Quality

The gain persists beyond greedy evaluation. Table 5 evaluates eight sampled completions per instance at temperature 0.6 using the three full-budget checkpoints. CRR improves pass@8 by 3.9 points over GRPO and all-8 success by 5.2 points. All-8 is the fraction of instances solved by every sampled completion: it measures consistency across samples, not the causal quality of individual actions. The full pass@1/2/4/8 results are in Section P.

The appendix further tests how the fork budget is allocated and whether counterfactual quality explains the gain. Among 20,000 logged forks, 12% use an immediately invalid alternative; this class contributes 8% of a class-mean plug-in contrast diagnostic. This diagnostic is not a decomposition of the total training gradient. A validity-filtered proposal changes pass@1 from 36.2% to 36.4% in the reduced-budget, single-seed comparison, which does not establish equivalence. Selector-alignment probes find enrichment for patch-relevant edits and large sampled contrasts, without identifying per-step causality. Controlled replay-noise experiments show gradual degradation in the tested setting; averaging multiple counterfactuals does not generally remove systematic replay bias. Severe trajectory-length heterogeneity remains unresolved. Section P gives the full protocols and diagnostics.

Table 4: Composition on Verified, measured by pass@1 (%). TSR, PRM, and their combination are added to the base GRPO configuration. The 44.9% result uses CRR+PRM; 45.6% uses CRR+TSR+PRM.

Configuration	none	TSR	PRM	TSR+PRM
Base GRPO	36.4	38.0	39.1	40.0
Base GRPO + CRR	41.7	43.6	44.9	<u>45.6</u>

Table 5: Sampling evaluation on all 500 Verified instances using eight completions per instance ($T = 0.6$). Reported percentages are averaged over three full-budget training seeds. Greedy pass@1 is included for context.

Method	Greedy pass@1	Sampled pass@1	pass@8	all-8
Vanilla GRPO	36.4	35.9	53.8	18.7
SWE-Trace	39.1	38.6	55.6	21.2
CRR	41.7	41.2	57.7	23.9

6 Discussion

Limitations. CRR requires affordable, reliable forks and a trustworthy terminal verifier. Expensive restoration can erase its resource-efficiency advantage, and the reported operating points do not isolate severe trajectory-length skew. The reduced-budget noise experiments test a controlled injection protocol; averaging multiple continuations does not remove systematic replay bias. Deterministic transitions also leave future policy-sampling variance intact. Our unbiasedness result concerns a fixed-index contrast, whereas the implemented selector induces a weighted training objective without an established convergence guarantee. The selector-alignment diagnostics provide decision-relevance proxies, not proof of per-step causality. Additional boundary cases and cost diagnostics appear in Section M.

Broader impact. Better SWE agents can support software maintenance, while executable counterfactual traces make their training signals more inspectable. The same capabilities can accelerate malicious software development, and verifier-coupled training can exploit incomplete tests or harnesses. CRR does not resolve these risks. We recommend adversarial test-suite hardening [Yu et al., 2026], execution-free corroboration such as SWE-RM [Shum et al., 2025], and documentation of replay traces and verifier limitations when releasing training artefacts.

Conclusion. CRR uses executable counterfactual continuations to score selected actions while retaining the realised training trajectory. It requires neither process labels nor a learned reward model, but charges the additional replay compute. The reported experiments show improved pass@1 across three SWE benchmarks and an equal-wall-clock gain over vanilla GRPO on Verified. Its practical value depends on affordable, reliable forkability; understanding replay noise, selector weighting, and heterogeneous trajectory costs remains an important direction.

Acknowledgments and Disclosure of Funding

This work received no third-party funding or support. The authors declare no competing interests.

References

- Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024.
- Dilip Arumugam, Peter Henderson, and Pierre-Luc Bacon. An information-theoretic perspective on credit assignment in reinforcement learning. *arXiv preprint arXiv:2103.06224*, 2021.
- Ilze Amanda Auzina, Joschka Strüder, Sergio Hernández-Gutiérrez, et al. Intrinsic credit assignment for long horizon interaction. *arXiv preprint arXiv:2602.12342*, 2026.
- Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, et al. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.
- Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, et al. SWE-rebench V2: Language-agnostic SWE task collection at scale. *arXiv preprint arXiv:2602.23866*, 2026.
- Jarrold Barnes. Surprisal-guided selection: Compute-optimal test-time strategies for execution-grounded code generation. *arXiv preprint arXiv:2602.07670*, 2026.
- Shiyi Cao, Dacheng Li, Fangzhou Zhao, et al. SkyRL-Agent: Efficient RL training for multi-turn LLM agent. *arXiv preprint arXiv:2511.16108*, 2025.
- YanJun Chen, Yirong Sun, Hanlin Wang, et al. Contextual counterfactual credit assignment for multi-agent reinforcement learning in LLM collaboration. *arXiv preprint arXiv:2603.06859*, 2026.
- Ganqu Cui, Lifan Yuan, Zefan Wang, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.
- Aladin Djuhera, Swanand Ravindra Kadhe, Farhan Ahmed, Heiko Ludwig, and Holger Boche. TSR: Trajectory-search rollouts for multi-turn RL of LLM agents. *arXiv preprint arXiv:2602.11767*, 2026.
- Yaxin Du, Yuzhu Cai, Yifan Zhou, et al. SWE-Dev: Evaluating and training autonomous feature-driven software development. *arXiv preprint arXiv:2505.16975*, 2025.
- Chanakya Ekbote, Vijay Lingam, Sujay Sanghavi, et al. MURPHY: Multi-turn GRPO for self-correcting code generation. *arXiv preprint arXiv:2511.07833*, 2025.
- Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 19034–19055. PMLR, 2025. URL <https://proceedings.mlr.press/v267/gehring25a.html>.
- Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL <https://openreview.net/forum?id=5zwF1GizFa>.
- Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025. doi: 10.1038/s41586-025-09422-z.
- Hao Han, Jin Xie, Xuehao Ma, et al. SWE-TRACE: Optimizing long-horizon SWE agents through rubric process reward models and heuristic test-time scaling. *arXiv preprint arXiv:2604.14820*, 2026.
- Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree Search for LLM Agent Reinforcement Learning, 2026. URL <https://arxiv.org/abs/2509.21240>.

- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://openreview.net/forum?id=VTF8yNQm66>.
- Joongwon Kim, Wannan Yang, Kelvin Niu, et al. Scaling test-time compute for agentic coding. *arXiv preprint arXiv:2604.16529*, 2026.
- Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. URL <https://openreview.net/forum?id=QF0N3x2XVm>.
- 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 (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.
- Chaochao Lu, Biwei Huang, Ke Wang, José Miguel Hernández-Lobato, Kun Zhang, and Bernhard Schölkopf. Sample-efficient reinforcement learning via counterfactual-based data augmentation. *arXiv preprint arXiv:2012.09092*, 2020.
- Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. Alibaba LingmaAgent: Improving automated issue resolution via comprehensive repository exploration. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE)*, pages 238–249, 2025a. doi: 10.1145/3696630.3728549.
- Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. SoRFT: Issue resolving with subtask-oriented reinforced fine-tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11427–11441. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.acl-long.559. URL <https://aclanthology.org/2025.acl-long.559/>.
- Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Éric Moulines, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 7654–7664. PMLR, 2021. URL <https://proceedings.mlr.press/v139/mesnard21a.html>.
- OpenAI. Introducing SWE-bench Verified. <https://openai.com/index/introducing-swe-bench-verified/>, 2024. Updated February 24, 2025.
- Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 47717–47737. PMLR, 2025. URL <https://proceedings.mlr.press/v267/pan25g.html>.
- Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, et al. HiPER: Hierarchical reinforcement learning with explicit credit assignment for large language model agents. *arXiv preprint arXiv:2602.16165*, 2026.
- Huy N. Phan, Tien N. Nguyen, Phong X. Nguyen, and Nghi D. Q. Bui. HyperAgent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.
- Silviu Pitit, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/294e09f267683c7ddc6cc5134a7e68a8-Abstract.html>.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

- 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.
- KaShun Shum, Binyuan Hui, Jiawei Chen, et al. SWE-RM: Execution-free feedback for software engineering agents. *arXiv preprint arXiv:2512.21919*, 2025.
- Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991. doi: 10.1145/122344.122377.
- Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, et al. Hindsight credit assignment for long-horizon LLM agents. *arXiv preprint arXiv:2603.08754*, 2026.
- Chaofan Tao, Jierun Chen, Yuxin Jiang, et al. SWE-Lego: Pushing the limits of supervised fine-tuning for software issue resolving. *arXiv preprint arXiv:2601.01426*, 2026.
- Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. MAGIS: LLM-based multi-agent framework for GitHub issue resolution. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1647.
- Simon Vanneste, Astrid Vanneste, Kevin Mets, Ali Anwar, Siegfried Mercelis, Steven Latré, and Peter Hellinckx. Learning to communicate using counterfactual reasoning. *arXiv preprint arXiv:2006.07200*, 2020.
- Ruiyi Wang and Prithviraj Ammanabrolu. A practitioner’s guide to multi-turn agentic reinforcement learning. *arXiv preprint arXiv:2510.01132*, 2025.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025a. URL <https://openreview.net/forum?id=OJd3ayDDoF>.
- You Wang, Michael Pradel, and Zhongxin Liu. Are “solved issues” in SWE-bench really solved correctly? an empirical study. *arXiv preprint arXiv:2503.15223*, 2025b.
- Rongzhe Wei, Ge Shi, Min Cheng, Na Zhang, Pan Li, Sarthak Ghosh, Vaibhav Gorde, and Leman Akoglu. Long-Horizon Plan Execution in Large Tool Spaces through Entropy-Guided Branching, 2026. URL <https://arxiv.org/abs/2604.12126>.
- Chengxing Xie, Bowen Li, Chang Gao, et al. SWE-Fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. *arXiv preprint arXiv:2501.05040*, 2025.
- An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- Boxi Yu, Yang Cao, Yuzhong Zhang, et al. SWE-ABS: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark. *arXiv preprint arXiv:2603.00520*, 2026.
- Danlong Yuan, Wei Wu, Zhengren Wang, et al. SWE-MiniSandbox: Container-free reinforcement learning for building software engineering agents. *arXiv preprint arXiv:2602.11210*, 2026.
- Daoguang Zan, Zhirong Huang, Ailun Yu, et al. SWE-bench-java: A GitHub issue resolving benchmark for Java. *arXiv preprint arXiv:2408.14354*, 2024.

- Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! *arXiv preprint arXiv:2505.23419*, 2025. NeurIPS 2025 Datasets and Benchmarks Track.
- Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 1592–1604, 2024. doi: 10.1145/3650212.3680384.
- Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 62178–62209. PMLR, 2024. URL <https://proceedings.mlr.press/v235/zhou24t.html>.
- King Zhu, Hanhao Li, Siwei Wu, et al. Scaling test-time compute for LLM agents. *arXiv preprint arXiv:2506.12928*, 2025.

A Proofs

Proof of Proposition 1. Fix an action-augmented history $h_t^+ = (s_0, a_0, \dots, s_t, a_t)$. Determinism of the sandbox fixes the environment transitions for any realised future action sequence, but the policy continuation after a_t remains stochastic. Let

$$G(s, a, \pi_\theta) = \mathbb{E}_{a_{t+1:T} \sim \pi_\theta} [R(\tau) \mid s_t = s, a_t = a]$$

denote the expected terminal return after taking action a at state s and then continuing with π_θ . By definition, $G(s_t, a_t, \pi_\theta) = Q^{\pi_\theta}(s_t, a_t)$, and similarly $G(s_t, \tilde{a}_t, \pi_\theta) = Q^{\pi_\theta}(s_t, \tilde{a}_t)$ for a realised counterfactual action. Taking expectations over both the counterfactual action $\tilde{a}_t \sim q_{-a_t}$ and the future policy continuation randomness gives

$$\mathbb{E}[\hat{A}_t^{\text{CRR}} \mid h_t^+] = G(s_t, a_t, \pi_\theta) - \mathbb{E}_{\tilde{a}_t \sim q_{-a_t}} [G(s_t, \tilde{a}_t, \pi_\theta)] = Q^{\pi_\theta}(s_t, a_t) - \mathbb{E}_{\tilde{a}_t \sim q_{-a_t}} [Q^{\pi_\theta}(s_t, \tilde{a}_t)].$$

If one uses the idealised proposal $q = \pi_\theta$ without excluding the realised action, the second term is $V^{\pi_\theta}(s_t)$ and the contrast recovers the classical advantage. The implemented proposal conditions on $\tilde{a}_t \neq a_t$, so the exact estimator targets the alternative-action contrast stated in Proposition 1. $\square$

Proof of Proposition 2. Let $X = R(\tau)$, $Y = R(\tilde{\tau}_t)$, and $\bar{Y}$ a sibling-group baseline drawn from the same group. For any conditioning variable or bucket B ,

$$\begin{aligned} \text{Var}[X - Y \mid B] &= \text{Var}[X \mid B] + \text{Var}[Y \mid B] - 2 \text{Cov}(X, Y \mid B), \\ \text{Var}[X - \bar{Y} \mid B] &= \text{Var}[X \mid B] + \text{Var}[\bar{Y} \mid B] - 2 \text{Cov}(X, \bar{Y} \mid B). \end{aligned}$$

Subtracting the two identities gives the decomposition in Proposition 2; rearranging terms gives the stated sufficient condition. Section C reports prefix-bucket covariance and SNR diagnostics supporting the empirical regime in which this condition is plausible, but the decomposition itself does not assert universal variance reduction. $\square$

B Algorithm Pseudo-Code

Algorithm 1 reproduces the per-iteration training loop of CRR in pseudo-code. It is implemented atop the asynchronous multi-turn rollout dispatcher of SKYRL-AGENT [Cao et al., 2025] with the container-free sandboxing of SWE-MINISANDBOX [Yuan et al., 2026]. The fork pool runs on dedicated workers in parallel with the main rollout pool; in our profile, fork/materialisation and counterfactual rollout are the two largest overhead components (Section K).

Algorithm 1 Counterfactual Rollout Replay (per training iteration)

Require: Policy π_θ , task batch $\{i_1, \dots, i_B\}$, budget k , entropy threshold η , proposal q_{-a}

Ensure: Updated policy parameters θ'

```

1: for each task  $i_b$  in parallel do
2:   Sample  $G$  on-policy trajectories  $\{\tau^{(g)}\}_{g=1}^G$  from  $\pi_\theta$  in the sandbox
3:   for  $g = 1, \dots, G$  do
4:     Compute step scores via Eq. (5); let  $\mathcal{I}_g$  be up to  $k$  non-terminal indices with  $H_t > \eta$ 
5:     for  $t \in \mathcal{I}_g$  do
6:       Snapshot sandbox state  $\sigma(s_t)$ 
7:       Sample  $\tilde{a}_t \sim q_{-a_t}(\cdot \mid s_t)$  with  $\tilde{a}_t \neq a_t$ 
8:       Restore  $\rho(\sigma(s_t))$  and roll out  $\tilde{\tau}_t$  on-policy from  $\tilde{a}_t$ 
9:       Record  $R(\tilde{\tau}_t)$ 
10:    end for
11:  end for
12:  Form hybrid advantages  $\hat{A}_t^*$  via Eq. (3)
13: end for
14: Update  $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_{\text{CRR}}(\theta)$  using all collected  $(s_t, a_t, \hat{A}_t^*)$ 

```

C Empirical Correlation Diagnostics

Proposition 2 gives a variance decomposition and a sufficient covariance-and-baseline condition for CRR to reduce conditional variance relative to an outcome baseline. As a diagnostic for this regime, we measure whether on-policy and counterfactual returns covary positively within task/prefix buckets. For every training step at which counterfactual rollouts are collected, we bin paired returns by task id, selected-step decile, and a prefix signature derived from the first ten semantic tokens of the realised action history, then compute the within-bucket covariance and average across buckets weighted by bucket size.

Table 6: Empirical task/prefix-bucket covariance $\hat{Cov}(R(\tau), R(\tilde{\tau}_t) \mid B)$ at five checkpoints during training. The covariance remains strictly positive throughout, with the average value increasing as the policy becomes more confident in its prefix decisions.

Step	200	1,000	2,000	3,000	4,000
$\hat{Cov}$ (mean)	0.044	0.071	0.103	0.119	0.128
$\hat{Cov}$ (5th pct.)	0.018	0.029	0.046	0.059	0.068
$\hat{Cov}$ (95th pct.)	0.094	0.131	0.171	0.183	0.198

The 5th-percentile covariance is strictly positive at every checkpoint. This does not by itself prove the sufficient condition in Proposition 2, because the condition also depends on counterfactual variance and sibling-baseline covariance. It supports the intended regime diagnostically: as the policy improves, task/prefix buckets become more predictive of eventual outcome and counterfactual continuations from nearby prefixes end in increasingly correlated terminal states.

The plotted per-step advantage-SNR diagnostic uses the magnitude $|\hat{A}_t^*|$ of the hybrid advantage relative to its standard deviation across the rollout pool, recorded every 200 optimizer updates. It is an advantage-signal diagnostic; it does not directly measure the variance of the complete policy-gradient update.

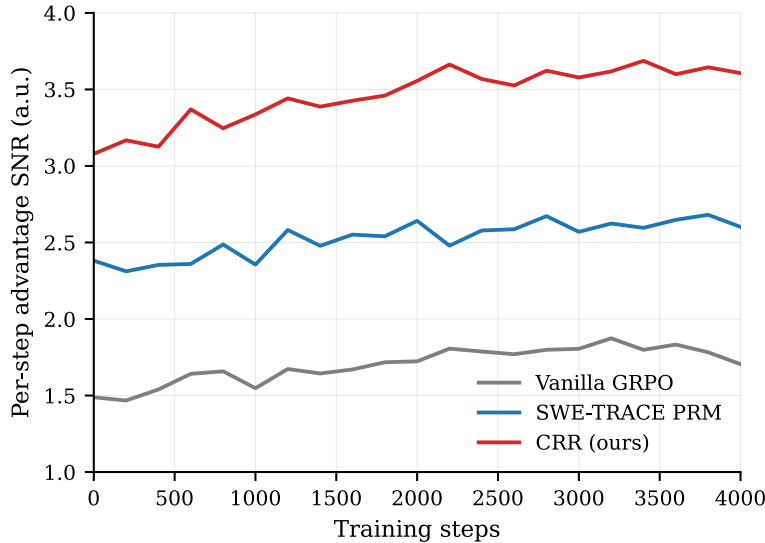


Figure 4: Per-step advantage SNR across training. CRR achieves approximately twice the advantage SNR of vanilla GRPO and about 1.4 times that of the SWE-TRACE PRM in the reported diagnostic. This advantage-SNR comparison does not establish the sufficient variance condition in Proposition 2.

D Hyperparameters

Table 7 lists the full-budget CRR configuration. We distinguish newly collected on-policy environment trajectories from optimizer minibatch updates over the collected rollout buffer. The reduced-budget controls use a fixed 48 H100-hour budget and may complete different numbers of trajectories and updates. We did not tune CRR-specific hyperparameters separately from the GRPO baseline; the only parameters introduced by CRR are the fork budget k , the counterfactual temperature T_{cf} , the entropy threshold η , and the type weight λ_{type} .

Table 7: Default hyperparameters for the full-budget CRR runs. Ablations vary their stated factor, and equal-device-time controls may complete fewer trajectories or updates.

Hyperparameter	Value
Base model	QWEN3-14B
Training data	Post-filter SWE-GYM (2,438) + disjoint SWE-REBENCH train split (1,200)
RL algorithm	GRPO with CRR advantage
Rollout dispatch microbatch size	64
Group size G	8
Trajectories per rollout collection	256
Rollout collection rounds	about 94
Total on-policy rollout trajectories	24,000
Max turns per trajectory H	50
Fork budget k	4
Counterfactual samples per fork K	1 in the full-budget main comparison
Counterfactual temperature T_{cf}	0.7
Selector entropy threshold η	0.4
Selector type weight λ_{type}	0.6
Learning rate	5×10^{-6}
Warmup steps	100
Optimizer minibatch/update steps	4,000
KL penalty β	0.04
PPO clip ϵ	0.2
Discount γ	1.0
LoRA rank / alpha	128 / 256
Optimiser	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$)
Hardware	8×NVIDIA H100 80GB
Framework	SKYRL-AGENT + SWE-MINISANDBOX
Main CRR training compute	321 H100-hours per seed; see Table 14
Random seeds	{11, 23, 47}

E Prompt and Tool Catalogue

All experiments share the same prompting scaffold to isolate the contribution of the training procedure. The system prompt instructs the agent to act as a software engineer resolving a single GitHub issue through a fixed tool catalogue; the user prompt contains the issue title, body, and an indication of the working directory. The system prompt is reproduced below for reference.

System prompt (verbatim):

You are an autonomous software engineer working in a containerised development environment. Your goal is to resolve the GitHub issue described by the user by editing the relevant source files and validating your changes against the project’s test suite. You have access to the following tools: `view_file`, `edit_file`, `run_shell`, `run_tests`, `submit_patch`. Each turn you must emit a single tool call as a JSON object. The episode terminates when you call `submit_patch` or after 50 turns. Read carefully before editing, edit minimally, and verify with the test suite before submission.

Tool catalogue. The agent has five tools: `view_file(path, start, end)` returns up to 200 lines of a file with line numbers; `edit_file(path, start, end, replacement)` replaces a contiguous range with new content and returns a diff; `run_shell(command)`

`timeout()` executes a shell command in the repository root, with a default 60-second timeout; `run_tests(test_filter)` invokes the project’s canonical test runner, filtered by an optional pytest-style selector; and `submit_patch()` terminates the episode and triggers the held-out test suite. The tool schemas used by the harness follow these semantics, and `stdout` is truncated deterministically to keep observations bounded.

Decoding settings. Greedy pass@1 evaluation uses temperature 0.0, while training rollouts use temperature 0.6; the counterfactual proposal $q_{\neg a_t}$ overrides the latter with $T_{cf} = 0.7$ and rejection-conditions on an alternative action as described in Section 4.4. The multi-sample evaluation in Table 5 instead uses evaluation temperature 0.6 and eight completions per instance. Tool calls are constrained by a JSON-grammar decoder that rejects malformed structured outputs at the token level; rejection statistics during training are reported in Section G.

F Determinism Filter and Network Replay

CRR’s unbiasedness guarantee (Proposition 1) hinges on the assumption that the sandbox is deterministically forkable. Real SWE task instances violate this assumption in three ways: (i) tests that are flaky in time, ordering, or random seed; (ii) tests that depend on network state (package indexes, time-of-day APIs, mutable third-party services); and (iii) tests whose environment introduces non-determinism through file-system races or thread scheduling. We describe the three filters and replay mechanisms used to enforce determinism on the training corpus.

Three-rerun consistency filter. For every upstream candidate task instance we execute the canonical solution patch three times in independent sandboxes, with different random seeds and shuffled test orderings. A task is admitted to the training set only if the held-out test suite produces identical pass/fail outcomes on all three runs. This filter discards 6.8% of the upstream SWE-GYM candidate pool and 9.4% of the upstream SWE-REBENCH candidate pool. After filtering, the admitted corpus used by the main training runs contains 2,438 SWE-GYM tasks and 1,200 SWE-REBENCH training tasks, for 3,638 deterministic instances total. The discarded instances are dominated by tests that depend on dictionary iteration order, integer hashing, or implicit time-zone conversion; we report per-repository discard rates for the largest repositories in Table 8.

Table 8: Discard rate of the three-rerun determinism filter for the largest repositories in the upstream SWE-GYM and SWE-REBENCH candidate pools. Repositories with networked test fixtures dominate the discards.

Repository	Tasks evaluated	Discard rate (%)
sympy	482	4.1
django	591	11.7
scikit-learn	277	5.4
pytest	162	7.4
matplotlib	191	6.8
sphinx	219	9.6
flask	138	12.3
requests	96	18.8
astropy	240	5.0
xarray	166	6.0

Deterministic package mirror. Network operations during patch installation are routed through a deterministic mirror that pins every package version, hash, and metadata blob at the time the task instance is admitted. Subsequent counterfactual rollouts hit the same frozen mirror, so `pip install` and equivalent operations return bit-identical artefacts even if upstream packages mutate during training.

HTTP replay layer. For tasks whose test suite issues outbound HTTP requests (a small minority dominated by `requests`, `flask`, and `sphinx` extension hooks), we capture every request-response pair during canonical solution execution and replay them deterministically during training. The replay layer matches on request method, URL, and a normalised body digest; mismatches are surfaced as task-level errors and the instance is excluded.

SWE-REBENCH split hygiene. The 1,200 admitted SWE-REBENCH training tasks are disjoint from the held-out SWE-REBENCH Python evaluation split. We split the candidate stream by collection month before filtering and then remove overlaps by task identifier, issue URL, pull-request URL, repository/commit window, exact patch hunk, near-duplicate patch hunk, and near-duplicate issue text. Held-out evaluation tasks are never used for rollout collection, counterfactual replay, selector tuning, or hyperparameter selection.

Residual non-determinism. Even after all three filters, a small residual non-determinism survives at the level of file-system iteration order on certain glob-heavy tests; we treat such cases as additional noise on top of the binary terminal reward. The full-budget main comparison uses $K = 1$ counterfactual sample per selected fork. In a separate residual-nondeterminism robustness setting, $K = 2$ averaging reduces stochastic terminal-noise variance at a reported $1.15\times$ throughput penalty; this setting is excluded from the full-budget ledger in Table 14. Averaging is not a correction for systematic replay bias, and the additional single-seed injection experiments establish only the behaviour observed under their stated reduced-budget protocol.

G Additional Ablations

Fork-pool concurrency. The asynchronous fork pool runs counterfactual rollouts on a separate logical worker pool while the main rollout pool produces on-policy trajectories. We profile concurrent counterfactual workers $W \in \{1, 2, 4, 8, 16\}$. Table 9 reports relative throughput inside this fork-pool profiling harness, not the end-to-end on-policy-trajectory per H100-hour accounting used in Table 14. Throughput grows sublinearly: increasing W from 8 to 16 changes the relative throughput from 4.25 to 4.46, about 4.9%. The reported pass-rates range from 41.6 to 41.7 %; this local profile is distinct from the reduced-budget equal-wall-clock $W = 1$ control.

Table 9: Fork-pool profiling throughput and pass-rate as a function of counterfactual-worker concurrency W . Throughput is normalised to $W = 1$ inside the profiling harness. These profiling results are not an end-to-end device-time ledger or evidence of invariance under arbitrary changes in concurrency.

Concurrency W	1	2	4	8	16
Relative fork-pool throughput	1.00	1.75	2.89	4.25	4.46
Verified pass@1 (%)	41.6	41.7	41.7	41.7	41.6

GRPO vs. PPO outer loop. CRR can be inserted into either policy-gradient outer loop. Replacing GRPO with PPO yields $41.4 \pm 0.7\%$ for CRR-PPO on SWE-BENCH VERIFIED, compared with $41.7 \pm 0.6\%$ for CRR-GRPO and $34.1 \pm 1.0\%$ for the outcome-only PPO baseline. The reported PPO improvement is +7.3 points. These endpoint summaries are descriptive; they do not establish statistical equivalence between CRR-PPO and CRR-GRPO or a corrected equal-compute PPO comparison. Equation (3) changes the selected-step advantages while retaining the outer-loop update structure.

Selector baselines. The Random selector samples up to $k = 4$ eligible non-terminal steps uniformly without replacement from the same entropy-thresholded eligibility set used by CRR. The Stride selector sorts eligible non-terminal steps by time and chooses approximately evenly spaced quantiles of the trajectory. Both baselines use the same fork budget and counterfactual proposal as CRR; only the decision-index selection rule changes.

Tool-grammar rejection statistics. The JSON-grammar decoder used during training rejects 1.4% of tokens at the start of training and 0.3% at convergence, with malformed tool calls falling fastest. We see no evidence that grammar enforcement biases the counterfactual proposal: the rejection rate at counterfactual-sampling temperature $T_{\text{cf}}=0.7$ is within 0.05 percentage points of the on-policy rate at every checkpoint.

Counterfactual sample size K . We compare the main setting $K=1$ (one counterfactual per fork) against $K=2$ and $K=4$. The averaging effect materialises as expected: per-step advantage SNR

scales as approximately $\sqrt{K}$. Pass-rate improvements diminish quickly: $K=2$ gains +0.4 points over $K=1$ at $1.6\times$ counterfactual cost, while $K=4$ gains only +0.6 points at $3.2\times$ cost. We adopt $K=1$ in all main-text numbers as the Pareto-optimal choice; higher- K runs are ablations and residual-nondeterminism robustness checks rather than part of the main accounting.

Curriculum effects. We checked whether the relative gain from CRR is sensitive to a difficulty-sorted training curriculum. Sorting tasks from easy to hard by canonical-trajectory length increases vanilla GRPO pass-rate by +0.7 points but only changes CRR pass-rate by +0.2 points; CRR is partially robust to curriculum choice because its per-step advantages already discount easy steps with low entropy. We use the unsorted random ordering for all main-text experiments to remove this confound.

H Additional Per-Repository Evaluation

Figure 5 reports an additional per-repository comparison on SWE-BENCH VERIFIED, using a separate set of experimental runs from the main comparison in Table 2. CRR achieves higher pass@1 than vanilla GRPO on nine of the ten displayed repositories and matches GRPO on `flask`. These comparisons do not isolate trajectory length from other repository characteristics.

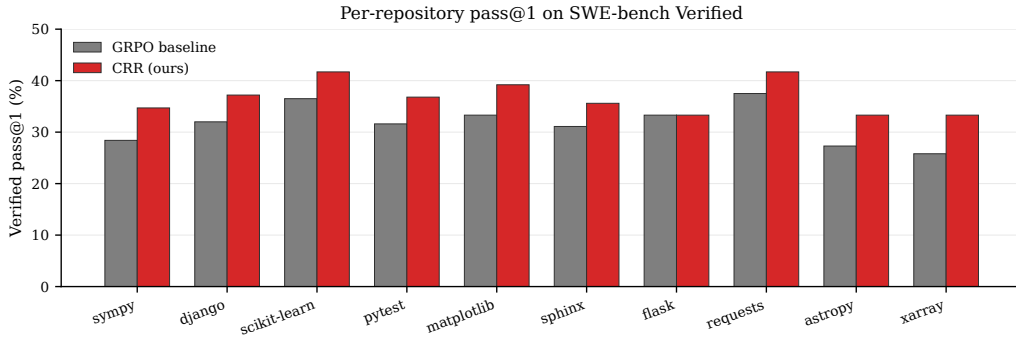


Figure 5: Per-repository pass@1 on SWE-BENCH VERIFIED from a separate experiment, averaged over three seeds.

I Error-Mode Taxonomy and Per-Mode Breakdown

We adopt the four-class taxonomy of Wang and Ammanabrolu [2025] for analysing failed trajectories and add an “Other” bucket for failures that the deterministic classifier cannot assign cleanly to one of the four modes. Each failed instance is labelled by a single dominant error mode through a deterministic post-hoc classifier that inspects the agent’s final tool-call sequence and the diagnostic output of the held-out test suite.

Wrong localisation. The agent edits a file or function that is not on the canonical fix path. The held-out test suite fails because the actual defect remains untouched. This mode is dominant on repositories with deep abstraction layers (`sympy`, `astropy`).

Partial fix. The agent edits the correct location but misses a co-required change elsewhere (e.g. updates the implementation but not its caller, or fixes the reported test but leaves a regression elsewhere). The held-out test suite passes the originally-failing test but fails at least one previously-passing test.

Regression. The agent’s edit introduces a new failure unrelated to the issue, typically by an aggressive global rewrite or by a misfired `sed`-style transformation.

Harness subversion. The agent modifies the test harness, conf test fixtures, or assertion expectations rather than the implementation under test. Such a patch may produce an apparent verifier

pass by changing the verifier target itself, but we count it as a failure after an audit check flags the modified harness files. We detect this class through a static check on the patched files against a canonical exclusion list.

Table 10: Error-mode distribution among failed instances on SWE-BENCH VERIFIED, normalised to percentages of all failures of the corresponding method. The table reports conditional shares among failures; Table 11 converts the main comparison to approximate absolute rates over all evaluated instances.

Method	Wrong loc.	Partial fix	Regression	Harness subv.	Other
Vanilla GRPO	34.2	39.7	19.4	6.7	0.0
SWE-TRACE	33.6	35.1	24.5	6.8	0.0
CRR (ours)	39.5	24.2	11.3	6.6	18.4
CRR + SWE-TRACE	40.8	21.9	9.1	6.8	21.4

Table 11: Approximate absolute error-mode rates over all SWE-BENCH VERIFIED instances, obtained by multiplying Table 10’ conditional failure shares by the corresponding failure rates from Table 2.

Method	Wrong loc.	Partial fix	Regression	Harness subv.	Other
Vanilla GRPO	21.75	25.25	12.34	4.26	0.00
CRR (ours)	23.03	14.11	6.59	3.85	10.73

Relative to vanilla GRPO, CRR has lower conditional shares and lower approximate absolute rates of *partial fix* and *regression* failures, alongside a larger residual “Other” bucket. This pattern does not hold uniformly across methods: SWE-TRACE’s conditional regression share is 24.5%, compared with 19.4% for vanilla GRPO. The error taxonomy is a descriptive post-hoc analysis and does not establish that an individual mode changed because of improved credit assignment. *Harness subversion* remains present in all methods, so these results do not show that counterfactual credit alone resolves verifier exploitation. We discuss the implications for future closed-loop training pipelines in Section 6.

J Qualitative Case Studies

This appendix presents three illustrative trajectories from the training run. For each case we report the on-policy trajectory, a sampled counterfactual rollout from a selected decision point, and the resulting per-step advantage. These examples illustrate how the sampled return contrast enters training; a single pair of stochastic continuations does not establish the causal contribution of an individual action.

Case study 1: a localisation-driven win

The agent is asked to fix a numerical-precision bug in a popular scientific library. On the on-policy trajectory the agent reads the test file at turn 3, recognises the relevant tolerance constant, edits a single line at turn 7, runs the focused test at turn 8, and submits at turn 11; the held-out test suite passes. CRR selects turn 3 as a high-leverage decision point: at this turn the policy entropy is high (the agent could equally have read the implementation file or the documentation). The counterfactual fork samples “view_file(documentation.md)” instead and continues on-policy; without the test file in context the agent fails to identify the correct tolerance and the held-out test suite fails. The resulting advantage is $\hat{A}_3 = 1 - 0 = 1$, sharply credited to the early-localisation decision rather than to the eventual edit.

Case study 2: an avoided regression

The agent is asked to fix a serialisation bug. On the on-policy trajectory the agent edits the serialiser correctly at turn 12 and verifies with a focused test at turn 14, then submits at turn 16; the held-out test suite passes. CRR selects turn 12 as a high-leverage decision point. The counterfactual fork samples a more aggressive global rewrite of the serialiser interface; on-policy continuation in the

fork produces a patch that passes the focused test but introduces a regression in a previously-passing serialisation roundtrip test. The sampled advantage is $\hat{A}_{12} = 1 - 0 = 1$, assigning positive credit to the realised edit relative to this sampled alternative.

Case study 3: a beneficial counterfactual

The agent is asked to fix a parameter-validation bug. On the on-policy trajectory the agent edits the validator at turn 6 but introduces a typo in an exception message, causing a “raises_with_message” assertion in the held-out test suite to fail; reward is 0. CRR selects turn 6. The counterfactual fork samples a slightly different edit at turn 6 that happens to use the correct exception message; on-policy continuation produces a passing patch with reward 1. Advantage: $\hat{A}_6 = 0 - 1 = -1$, sharply punishing the typo-introducing edit even though the bulk of the trajectory was correct—a credit assignment that outcome-only RL with a single bit at the end of fifty turns cannot reproduce.

These cases illustrate early localisation, a contrast between edits with different regression outcomes, and a beneficial alternative to a failing edit. They are examples rather than an estimate of how much of the total training signal each pattern explains.

K Rollout-Dispatch Cost Breakdown

Table 12 reports the critical-path wall-clock latency of one rollout-dispatch microbatch of 64 trajectories on the 8×H100 node, referenced in the additional diagnostics of Section M. CRR multiplies the microbatch latency of vanilla GRPO by $1.71\times$, dominated by counterfactual fork/materialisation, counterfactual rollout, and additional test execution; the policy update step itself is essentially unchanged. The largest single overhead component in our stack is counterfactual fork/materialisation (36.2 seconds), followed by counterfactual rollout (28.1 seconds), suggesting that future infrastructure improvements should target both snapshot/restore materialisation and asynchronous rollout scheduling.

Table 12: Critical-path wall-clock latency per 64-trajectory rollout-dispatch microbatch on 8×H100 (seconds). The reported CRR microbatch latency is $1.71\times$ that of vanilla GRPO. This local profile does not replace the end-to-end device-time accounting in Table 14 or the training-window comparison in Table 15.

Component	Vanilla GRPO	CRR (ours)	Δ
Forward rollout (32 turns)	78.0	78.0	+0.0
Test execution	9.4	9.4	+0.0
Counterfactual fork ($k = 4$)	0.0	36.2	+36.2
Counterfactual rollout	0.0	28.1	+28.1
Counterfactual test execution	0.0	6.3	+6.3
Policy update (GRPO step)	12.7	12.9	+0.2
Total	100.1	170.9	+70.8

L Compute Accounting and Equal-Wall-Clock Comparison

Figure 2 measures sample efficiency in newly collected on-policy trajectories; it does not charge counterfactual replay. The consolidated device-time accounting in Table 14 charges 188 H100-hours to vanilla GRPO, 233 to SWE-TRACE, and 321 to CRR per full-budget run and seed. CRR’s total includes 133 H100-hours of fork-pool device-time in addition to 164 for on-policy rollout and 24 for learner updates. These scheduler-attributed components are mutually exclusive; overlapping their execution does not remove their device-time cost.

Table 15 provides the equal-wall-clock comparison on the same 8×H100 node. At 40.125 hours, CRR reaches 41.7 % on SWE-BENCH VERIFIED, versus 36.7 % for the extended vanilla-GRPO run, a 5.0-point difference in three-seed means. This comparison includes dispatch, tests, forks, waits, retries, optimisation, and checkpoints; one-time loading, final evaluation, and separate ablations are excluded. The comparison establishes the observed margin over vanilla GRPO at the reported operating points, not equal-compute dominance over every baseline or on every benchmark.

Table 13: Configuration and accounting conventions for the full-budget CRR run. Method-specific device-time components are reported in Table 14; equal-wall-clock checkpoints are reported in Table 15.

Quantity	Main value	Meaning
Training corpus SWE-REBENCH split	3,638 tasks disjoint	post-filter admitted deterministic tasks train/eval separated before filtering and decontaminated
On-policy trajectories	24,000	newly collected policy rollouts per full-budget CRR run/seed
Fork budget	$k = 4$	maximum selected decision points per trajectory
Counterfactual samples	$K = 1$	main setting; $K = 2, 4$ only in ablations
Main counterfactual tuples	about 96,000	nominal maximum $24,000 \times k \times K$ per seed
Main CRR training compute	321 H100-hours	rollout, fork-pool, and learner device-time per seed
CRR node wall-clock	40.125 hours	321/8 on the $8 \times \text{H100}$ node
Cost profile unit	64-trajectory microbatch	Table 12, not a single trajectory

M Additional Diagnostics and Discussion

Limitations. CRR rests on deterministic forkability, which breaks down in three regimes. Flaky tests bias the counterfactual differential; a three-rerun consistency filter discards $\sim 7\%$ of the upstream SWE-GYM candidate pool to enforce determinism (Section F). Network-dependent operations are mitigated by a deterministic package mirror and an HTTP replay layer. Residual file-system or scheduling nondeterminism is handled in robustness checks with K -shot averaging, while the main results use $K = 1$. Fork/materialisation and counterfactual rollout remain the dominant training-time overheads, motivating throughput-aware scheduling that more aggressively interleaves on-policy and counterfactual rollouts. Finally, the analysis assumes binary terminal rewards and an alternative-action proposal q_{-a_t} with support on plausible policy actions; extending to weighted or multi-objective rewards requires generalising Eq. (2) to weighted return functionals.

Repository comparisons, error modes, and cost. Section H reports an additional per-repository comparison from a separate experiment. These results do not isolate the effect of severe trajectory-length skew on training throughput. Using the error taxonomy of Wang and Ammanabrolu [2025] (Section I), CRR has lower conditional shares of *partial fix* and *regression* failures than vanilla GRPO, but also a larger residual “Other” category. The approximate rates over all evaluated instances are reported separately in Table 11; conditional shares should not be read as absolute failure rates or causal evidence about credit assignment. CRR multiplies rollout-dispatch microbatch latency by $1.71 \times$ (Section K). With all fork overhead charged, the equal-wall-clock endpoint in Table 15 is 41.7 % for CRR versus 36.7 % for vanilla GRPO on SWE-BENCH VERIFIED.

Model-based framing and broader impact. CRR can be read as a degenerate model-based RL method [Sutton, 1991] whose “model” is the actual environment rather than a learned approximator: it inherits access to the true transition function but pays the full per-fork cost. Because CRR depends on a verifier, it is bounded by that verifier’s integrity. In Section I, *harness subversion* accounts for 6.6% of CRR failures and 6.7% of vanilla-GRPO failures. Their approximate rates over all evaluated instances are 3.85% and 4.26%, respectively; these descriptive results do not establish a specific effect on verifier exploitation. We recommend pairing CRR-trained agents with adversarial test-suite hardening [Yu et al., 2026] and execution-free corroboration such as SWE-RM [Shum et al., 2025]. Every counterfactual rollout produces an auditable execution trace that can support examination of the resulting process signal.

N Counterfactual Trajectory Dataset

A side artefact of training is a dataset of counterfactual tuples that records, for every selected decision point, the on-policy and counterfactual returns together with the replay metadata needed to audit the fork. The main run uses 24,000 on-policy rollout trajectories, fork budget $k = 4$, and $K = 1$,

yielding about 96,000 counterfactual tuples per seed and 288,000 tuples across the three main seeds. Across all seeds, ablations, and diagnostic runs, the dataset to be released contains approximately 1.4 million tuples spanning 3,638 task instances. Each tuple contains the task id, policy seed/version, selected index t , sandbox snapshot identifier, realised action a_t , counterfactual action $\tilde{a}_t$, on-policy and counterfactual action suffixes sufficient for replay, terminal returns $R(\tau)$ and $R(\tilde{\tau}_t)$, proposal log-probability, and selector score. We expect this dataset to be useful as direct supervision for downstream PRM training, for verifier evaluation, and for studying the structure of credit assignment in long-horizon code agent trajectories. The dataset will be released under the same licence as SWE-GYM alongside the camera-ready version of the paper.

O Reproducibility Checklist

Code. The reference implementation is built on top of SKYRL-AGENT [Cao et al., 2025] and SWE-MINISANDBOX [Yuan et al., 2026] with no proprietary dependencies. The CRR-specific additions amount to fewer than 700 lines of Python.

Data. All training tasks are drawn from the public SWE-GYM and SWE-REBENCH releases. The main training corpus contains 3,638 post-filter deterministic tasks, and the SWE-REBENCH training split is disjoint from the held-out SWE-REBENCH Python evaluation split by the rules in Section F. The flaky-test filter, the package mirror configuration, and the recorded HTTP responses will be released with the code alongside the camera-ready paper.

Compute. The corrected full-budget device-time totals are 188 H100-hours for vanilla GRPO, 233 for SWE-TRACE, and 321 for CRR per seed (Table 14). CRR’s 321 H100-hours correspond to 40.125 hours on the 8×H100 node. Table 15 reports separate equal-wall-clock checkpoints, including all fork overhead. The reduced-budget controls use 48 H100-hours per arm. These training-window costs exclude one-time loading, final evaluation, and separate ablations; they are not an aggregate cost for reproducing every experiment.

Random seeds and uncertainty. The full-budget main experiments use seeds $\{11, 23, 47\}$ at the level of policy initialisation, the GRPO group sampler, and the counterfactual proposal $q_{\neg a_t}$. Reported main-table standard deviations describe variation across these three seeds. The equal-wall-clock curves and matched Search-Select control also report three-seed means; the latter includes seed standard deviations. The reduced-budget fork-overhead, approximate-forkability, validity-filtering, and fixed-budget K sweeps are single-seed diagnostics and do not estimate seed variability. Small differences in these sweeps do not establish significance or equivalence.

Limitations of reproduction. Because CRR depends on environment determinism, exact bit-level reproduction of pass-rates may differ across container engines (podman vs. docker) and across operating-system kernel versions. The controlled nondeterminism diagnostics characterise only the stated injection protocol; they do not guarantee robustness to arbitrary systematic changes in environment behaviour.

P Extended Evaluation and Resource Accounting

This section provides the resource ledger, matched-budget controls, and diagnostic experiments that complement the main evaluation. We distinguish full-budget training runs from reduced-budget comparisons. All device-time is expressed in H100-hours; wall-clock time is elapsed time on the specified hardware. Equal device-time does not imply equal numbers of completed trajectories, because replay and proposal costs differ between configurations.

Evaluation protocols. The full-budget experiments use the final checkpoints from the main comparison. The reduced-budget protocol allocates 48 H100-hours to each arm, holds the scaffold and hyperparameters fixed except for the tested factor, and evaluates the final checkpoint greedily on all 500 SWE-BENCH VERIFIED instances. Reduced-budget sensitivity experiments use one training seed unless stated otherwise. The matched Search-Select comparison instead uses three seeds per arm. Means and seed standard deviations are reported only where available; small differences in single-seed experiments are descriptive and do not establish statistical significance or equivalence.

P.1 Complete device-time accounting

Table 14 separates on-policy generation, counterfactual workers, auxiliary scoring, and learner updates. The categories are mutually exclusive scheduler-attributed device-time. Fork workers are logical roles on the same 8-H100 node, rather than additional uncharged hardware. The counterfactual pool contributes 133 H100-hours, bringing CRR’s full-budget total to 321 H100-hours. This complete total is the reference for interpreting the full-budget comparison.

Table 14: Full-budget resource accounting and SWE-BENCH VERIFIED pass@1. Performance is the mean $\pm$ seed standard deviation over three training seeds. Device-time includes all listed worker roles. Wall-clock durations in this table are rounded to one decimal place; the exact CRR endpoint used in the matched-time comparison is 40.125 hours.

Accounting item	Vanilla GRPO	SWE-TRACE	CRR
On-policy rollout (H100-hours)	164	164	164
Counterfactual fork pool (H100-hours)	0	0	133
PRM scoring / auxiliary work (H100-hours)	0	45	0
Learner updates (H100-hours)	24	24	24
Total device-time (H100-hours)	188	233	321
Wall-clock on one 8×H100 node (hours)	23.5	29.1	40.1
Verified pass@1 (%)	36.4 $\pm$ 0.8	39.1 $\pm$ 0.7	41.7 $\pm$ 0.6

The device-time and wall-clock totals agree: $188 = 8 \times 23.5$ and $321 = 8 \times 40.125$. Fork-slot utilization was 78%. In elapsed time, 58% of fork time overlapped on-policy generation, but this overlap is not subtracted from device-time. Consequently, the sample-efficiency comparison and the resource-efficiency comparison answer different questions: the former counts trajectories, while the latter must charge the additional counterfactual work.

P.2 Matched wall-clock learning curves and cumulative ledger

The matched-time comparison trains both methods on the same 8×H100 node and extends GRPO to CRR’s full training-window duration. Each column in Table 15 corresponds to an observed checkpoint from one continuing run per seed, rather than an interpolation between independently trained endpoints. Both arms use 100 optimizer updates of warmup to a learning rate of 5×10^{-6} , followed by a constant learning rate.

Table 15: Full-budget matched-wall-clock comparison on the same 8×H100 node. Pass@1 is the three-seed mean on SWE-BENCH VERIFIED; trajectory, update, and token rows give the cumulative ledger at the stated checkpoints. Token counts are in millions. All four checkpoints are observed.

Fraction of CRR training window	25%	50%	75%	100%
Wall-clock (hours)	10.031	20.063	30.094	40.125
GRPO pass@1 (%)	34.6	36.1	36.5	36.7
GRPO completed trajectories	11,648	20,992	29,632	37,824
GRPO optimizer updates	1,941	3,499	4,939	6,304
GRPO generated tokens (M)	28.70	57.40	86.11	114.76
GRPO effective tokens (M)	23.54	47.07	70.61	94.10
CRR pass@1 (%)	37.2	40.1	40.9	41.7
CRR completed on-policy trajectories	8,704	14,464	19,456	24,000
CRR optimizer updates	1,451	2,411	3,243	4,000
CRR on-policy generated tokens (M)	20.89	39.78	58.37	76.80
CRR all-generated tokens (M)	32.4	61.7	90.5	119.0
CRR effective tokens (M)	17.1	32.6	47.9	63.0

Generated tokens include counterfactual suffixes for CRR. Effective tokens are unique, loss-masked on-policy tokens. The training window begins at dispatch and ends at the last committed update. It includes tests, forks, waits, retries, optimization, and checkpointing; one-time loading, final evaluation, and separate ablation runs are excluded from both arms. At 40.125 hours, CRR reaches 41.7% and extended GRPO reaches 36.7%, an observed difference of 5.0 percentage points with the

fork overhead fully charged. The larger number of GRPO trajectories and optimizer updates at this endpoint is part of the matched-time comparison, not a quantity held fixed.

P.3 Sensitivity to fork overhead and concurrency

Table 16 varies the cost multiplier m for the measured snapshot, restore, and replay component. Rollout and learner costs remain fixed, and $m = 1$ denotes the native cost. Each arm receives the same 48 H100-hour budget, so increasing m reduces the number of on-policy trajectories completed before the cutoff.

Table 16: Fork-overhead sensitivity under the reduced-budget protocol: 48 H100-hours per arm, one training seed, and greedy pass@1 on all 500 Verified instances. The GRPO reference is the same 33.4% run for each cost multiplier.

Fork-cost multiplier	$m = 1$	$m = 2$	$m = 4$
CRR pass@1 (%)	36.2	34.8	33.2
Vanilla GRPO pass@1 (%)	33.4	33.4	33.4
CRR minus GRPO (percentage points)	+2.8	+1.4	−0.2
CRR completed on-policy trajectories	$\approx 6,000$	4,312	$\approx 2,760$

The reported completion counts follow the accounting relation

$$n_m = \frac{48}{0.004868 + 0.003132m}, \quad (6)$$

where the numerator is the device-time budget in H100-hours and the denominator is the corresponding cost per completed on-policy trajectory. Counts reflect asynchronous completions at the time cutoff. The observed CRR–GRPO margin changes sign between $m = 2$ and $m = 4$. This interval characterizes the measured single-seed operating points; it is not a universal break-even guarantee for other workloads or hardware.

Serial fork-worker control. An additional equal-wall-clock control with one fork worker ($W = 1$) reaches 34.8% compared with the 33.4% GRPO reference. Thus a gain is observed even without a large concurrent fork pool at this operating point. This result and the full-budget timing comparison do not isolate severe variation in trajectory lengths. Long suffixes, expensive restoration, or highly skewed trajectory lengths may reduce throughput and alter the useful operating range. Concurrency, scheduler behavior, rollout caps, and the costs charged to each worker role must therefore accompany any transfer of these results to another system.

P.4 Credit assignment versus selecting a branch

The Search-Select control isolates two uses of the same executable forks. Search-Select uses branch returns to select a training sequence, whereas CRR retains the realized trajectory and uses the return differential to assign credit to its actions. Both fork-using arms share the selector, number of fork executions per realized trajectory, branch-scoring verifier, rollout horizon, group size, device-time budget, and clipped learner.

Table 17: Matched Search-Select comparison. All arms use 48 H100-hours per seed and greedy evaluation on all 500 Verified instances. Values are the mean $\pm$ seed standard deviation over three training seeds. CRR and Search-Select use the same fork count per realized trajectory; GRPO has no forks.

Method	Use of forks	Pass@1 (%)
Vanilla GRPO	None	33.3 ± 0.7
Search-Select-GRPO	Select the training sequence	34.4 ± 0.8
CRR	Credit assignment on realized actions	36.2 ± 0.6

Search-Select protocol. Each realized trajectory uses the same selector and $k = 4$ fork executions as CRR, with identical rollout caps and verifier endpoints. Branches are scored with the terminal

verifier return $R(\cdot)$; no process reward model or separate judge is used. The verifier-best fork-derived sequence replaces the realized training sequence only if its return strictly exceeds that of the realized sequence. A tie retains the realized sequence. The resulting $G = 8$ sequences form the GRPO group, with ordinary group-relative advantages computed from their returns. Unselected branches are discarded and enter neither the group nor its advantage calculation. In particular, Search-Select does not reuse CRR’s counterfactual return differential.

The behavior-policy log-probabilities of each generated sequence are retained, and current-policy log-probabilities are recomputed during optimization to form the clipped current-to-behavior ratio. This ratio accounts for policy drift; it does not remove the selection of the training distribution. The control deliberately implements an expert-iteration-style distribution in which a better branch may replace the realized sequence. The replacement rate is 12%, and PPO clipping is activated for 26% of tokens in the replacement sequences.

CRR exceeds Search-Select in all three paired seeds, with a mean difference of 1.8 percentage points. These results support the benefit of the credit-assignment use of fork data under the matched protocols. They do not imply that selective branching or entropy-guided rollout allocation is itself new, nor do they establish dominance over all forms of inference-time search.

P.5 Multiple-sample evaluation

Table 18 evaluates the full-budget final checkpoints using eight temperature-sampled trajectories per instance. Besides greedy and sampled pass@1, it reports pass@2, pass@4, pass@8, and all-8. The last metric is the fraction of instances for which all eight samples succeed, and measures consistency across sampled trajectories.

Table 18: Multiple-sample evaluation on SWE-BENCH VERIFIED using full-budget final checkpoints and three training seeds. Each instance receives eight independent samples at temperature 0.6. All values are percentages; the greedy column is evaluated separately from the sampled columns.

Method	Pass@1 greedy	Pass@1 sampled	Pass@2	Pass@4	Pass@8	All-8
Vanilla GRPO	36.4	35.9	43.4	48.8	53.8	18.7
SWE-TRACE	39.1	38.6	45.7	50.8	55.6	21.2
CRR	41.7	41.2	48.1	53.1	57.7	23.9

The CRR advantage over GRPO persists at pass@8, with a difference of 3.9 percentage points. All-8 improves by 5.2 points, indicating greater consistency across samples. These are trajectory-level evaluation metrics and do not directly measure the quality or causal effect of individual decisions.

P.6 Counterfactual sample count and allocation across steps

Let K denote the number of counterfactual samples per selected step and k the number of selected decision points. Table 19 compares allocating four forks to more decision points with allocating them to more alternatives at fewer points. All arms retain the 48 H100-hour device-time budget, so completed trajectory counts need not match. The matched-fork rows hold kK fixed per realized trajectory, rather than asserting that the total number of forks completed during training is identical.

Table 19: Counterfactual-sample sensitivity under the reduced-budget protocol: 48 H100-hours per arm, one training seed, and greedy evaluation on all 500 Verified instances. Advantage SNR uses the main evaluation’s diagnostic and is expressed relative to GRPO.

Configuration	Forks per trajectory	Pass@1 (%)	Relative advantage SNR
$K = 1, k = 4$ (default)	4	36.2	$2.0\times$
$K = 2, k = 4$ (twice as many forks)	8	36.6	$2.5\times$
$K = 2, k = 2$ (matched forks)	4	35.8	$2.3\times$
$K = 4, k = 1$ (matched forks)	4	35.0	$2.6\times$

Within this sweep, allocating forks across more decision points performs at least as well as concentrating them on more alternatives at fewer points. Increasing K from one to two at fixed $k = 4$ raises the measured SNR by 25%, but doubles the fork count per trajectory. The observed 0.4-point pass@1 difference is not resolved by this single-seed comparison. For context, the full-budget anchors in Table 14 have seed standard deviations of 0.6–0.8 points; those standard deviations are not uncertainty estimates for the reduced-budget sweep itself.

P.7 Controlled departures from deterministic replay

The executable substrate and three-rerun consistency filter are shared by the main-comparison methods. To probe imperfect replay, a fraction ρ of replayed steps receives an output resampled from the recorded rerun distribution. Table 20 compares a single counterfactual with an averaged contrast using $K = 2$ counterfactual samples. These perturbations model the specified output-resampling mechanism, rather than every possible source of environment nondeterminism.

Table 20: Controlled replay-noise sensitivity under the reduced-budget protocol: 48 H100-hours per arm, one training seed, and greedy pass@1 on all 500 Verified instances. The mean-advantage-shift row is a diagnostic of the injected perturbation, not a pass-rate. A dash denotes an unreported configuration.

Injected nondeterminism ρ	0%	5%	10%
CRR, $K = 1$: pass@1 (%)	36.2	35.6	34.8
CRR, $K = 2$: pass@1 (%)	—	36.0	35.4
Vanilla GRPO reference: pass@1 (%)	33.4	33.4	33.4
Mean advantage shift	0	0.01	0.02

Performance declines gradually at the measured noise levels, and $K = 2$ recovers 0.4–0.6 percentage points relative to $K = 1$ under the same perturbation level. The zero-noise advantage shift is zero by construction; the other observed shifts are 0.01–0.02. These observations are consistent with the specified approximately symmetric injection mainly adding variance, but do not establish unbiasedness under arbitrary replay noise. Averaging samples cannot generally remove a systematic shift in the counterfactual return distribution. Reliable state restoration and a clearly characterized replay-noise model remain part of CRR’s operating conditions.

P.8 Counterfactual quality and validity filtering

An immediately invalid action can produce an easy contrast, but need not force its branch’s terminal return to zero: subsequent actions may recover. We classify 20,000 logged forks into immediately invalid actions, valid actions whose branches fail, and valid actions whose branches succeed. Table 21 reports each class frequency p_c and class-mean contrast $\bar{\Delta}_c$, where $\Delta = R(\tau) - R(\tilde{\tau})$.

Table 21: Composition of 20,000 logged counterfactual forks. This is a log diagnostic, not a separate 500-instance evaluation or a multi-seed performance comparison. The final column is the normalized class-mean plug-in quantity $p_c|\bar{\Delta}_c|/\sum_j p_j|\bar{\Delta}_j|$, expressed as a percentage.

Alternative class	Fork frequency (%)	Mean contrast	Plug-in share (%)
Immediately invalid	12	+0.15	8
Valid, branch fails	46	+0.40	83
Valid, branch succeeds	42	−0.05	9

Valid but failing alternatives account for 83% of this class-mean plug-in quantity, compared with 8% for immediately invalid alternatives. The quantity summarizes class frequencies and class means. It is not the fraction of total gradient magnitude, nor the fraction of the sum of absolute per-sample contrasts, because taking the absolute value after averaging within a class can cancel contrasts of different signs.

For a more direct check, the validity-filtered proposal resamples an alternative up to three times until a syntax/schema check passes; if none passes, the step is skipped. Table 22 reports the resulting performance.

Table 22: Validity-filtered proposal under the reduced-budget protocol: 48 H100-hours per arm, one training seed, and greedy evaluation on all 500 Verified instances.

Proposal	Pass@1 (%)
Unfiltered counterfactual proposal	36.2
Validity-filtered counterfactual proposal	36.4

The 0.2-point difference does not establish equivalence. It provides no evidence in this experiment that CRR’s measured gain requires immediately invalid alternatives. A decision to change the default proposal would require development-set evaluation rather than choosing the default from this test-set comparison.

Proposal freshness and multiple samples. The proposal uses the current policy with temperature adjustment and is sampled in the same rollout batch as the realized trajectory. It therefore lags the learner by at most one optimizer round in this implementation, rather than relying on a separately trained proposal model. This describes the measured pipeline and is not a general guarantee for more asynchronous deployments. When multiple counterfactual samples disagree in their terminal returns, the contrast uses their average; the resource and SNR trade-off is examined in Table 19.

P.9 Decision-relevance diagnostics for the selector

We evaluate selector alignment using two complementary data collections. The gold-patch-file first-touch statistic uses 2,000 logged training trajectories. The two advantage-magnitude statistics use a separate 300-trajectory probe in which counterfactuals are additionally evaluated at random, stride, and entropy-only step sets. This probe allows the selectors to be compared against the same counterfactual-advantage distribution.

Table 23: Selector alignment with decision-relevance proxies. The first numeric column uses 2,000 logged training trajectories; the remaining columns use the shared 300-trajectory probe. These are diagnostics of training trajectories, not pass@1 measurements on the 500-instance test set.

Selector	Hits gold-patch file on first touch (%)	Selected steps in top-decile $ \hat{A} $ (%)	Mean $ \hat{A} $ at selected steps
Random- k	13	10	0.11
Stride- k	15	11	0.11
Entropy only	30	24	0.18
Entropy + tool type	41	28	0.21

Relative to random selection, entropy plus tool type has approximately $3.2\times$ the gold-patch-file first-touch hit rate and $2.8\times$ the top-decile advantage enrichment. These results indicate that the selector allocates evaluations to steps enriched for patch-relevant edits and larger executable return contrasts. Gold-patch overlap and sampled contrast magnitude are proxies; neither establishes an individual step’s causal role in the final outcome.

Interpretation boundaries. The diagnostic evidence is consistent with useful allocation of a limited fork budget, but does not extend the fixed-index action-conditional analysis to a convergence guarantee for the selector-weighted training objective. The reduced-budget, single-seed experiments characterize measured sensitivity rather than a universal ordering of configurations. The evidence also leaves the effects of severe trajectory-length heterogeneity unresolved. These distinctions remain relevant even when the final endpoint improves under complete resource accounting.

NeurIPS Paper Checklist

1. Claims

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

Answer: [\[Yes\]](#)

Justification: The abstract and Section 1 describe executable counterfactual credit assignment, the fixed-index CRR contrast estimator, empirical comparisons, and the operating limits of counterfactual replay. The estimator and its assumptions are stated in Section 4 (Propositions 1 and 2), the empirical comparisons appear in Table 2, Table 4, and Figures 2–3, and the dataset is described in Section N. The equal-wall-clock comparison in Table 15 is restricted to vanilla GRPO on SWE-BENCH VERIFIED; the broader main-table comparison is not described as uniformly compute-matched. We do not claim universality across RL domains.

2. Limitations

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

Answer: [\[Yes\]](#)

Justification: Section 6 discusses limitations, including flaky tests, network-dependent operations, residual policy-sampling variance after the fork, selector-induced weighting, and verifier dependence. Section F reports the empirical discard rate of the upstream determinism filter (6.8% on SWE-GYM, 9.4% on SWE-REBENCH). Section I reports that harness subversion remains present in the evaluated methods; these observations do not show that counterfactual credit resolves verifier exploitation.

3. Theory assumptions and proofs

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

Answer: [\[Yes\]](#)

Justification: Proposition 1 is stated for fixed selected indices with explicit assumptions (sandbox determinism and forkability at horizon $H - t$, an action-augmented history, and an alternative-action proposal q_{-a_t}). Proposition 2 is a variance decomposition with an explicit sufficient condition, not a universal variance-reduction guarantee. Complete proofs are given in Section A. Section C reports prefix-bucket covariance diagnostics that support the empirical regime in which the condition is plausible.

4. Experimental result reproducibility

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

Answer: [\[Yes\]](#)

Justification: The full-budget CRR configuration is listed in Table 7, the prompt and tool catalogue are reproduced verbatim in Section E, the determinism filter, SWE-REBENCH split hygiene, HTTP replay layer, selector entropy approximation, and tool-type mapping are described in Sections 4.3 and F, and Algorithm 1 specifies the per-iteration training loop. The main experiments use seeds {11, 23, 47}. The reduced-budget controls state their 48 H100-hour limit and distinguish three-seed comparisons from single-seed diagnostics; Section O summarises these protocols.

5. Open access to data and code

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

Answer: [\[No\]](#)

Justification: We plan to release the CRR implementation (fewer than 700 lines of Python on top of SKYRL-AGENT and SWE-MINISANDBOX), the deterministic package mirror configuration, the recorded HTTP responses, and the counterfactual trajectory dataset of approximately 1.4M tuples (Section N) alongside the camera-ready paper. This version does not bundle these artefacts. Training data is drawn from public SWE-GYM and SWE-REBENCH releases. The planned resources will be linked from the camera-ready version under the stated dataset licence.

6. Experimental setting/details

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

Answer: [Yes]

Justification: Section 5.1 describes the base model (QWEN3-14B), post-filter training corpus (3,638 deterministic instances), disjoint SWE-REBENCH train/evaluation split, evaluation protocol (pass@1 with greedy decoding, 50-turn budget, three random seeds), and controlled baseline reproductions. The full hyperparameter list including optimiser settings appears in Table 7. CRR-specific hyperparameters (k , K , T_{cf} , η , λ_{type}) were not tuned separately from the GRPO baseline; the values reported are the result of a single sweep on a held-out 200-instance subset of SWE-GYM and are reused across all benchmarks.

7. Experiment statistical significance

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

Answer: [Yes]

Justification: Table 2 reports means and seed standard deviations over three seeds. The consolidated full-budget accounting and matched Search-Select control also report three-seed means and standard deviations; Table 15 reports three-seed means. The composition table presents point summaries without displayed seed standard deviations. The reduced-budget fork-overhead, approximate-forkability, validity-filtering, and fixed-budget K sweeps are single-seed diagnostics. We do not interpret their small differences as statistically significant or as evidence of equivalence, and we do not treat pooled seed standard deviations as a hypothesis test.

8. Experiments compute resources

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

Answer: [No]

Justification: Section 5.1 reports the hardware (one node, $8 \times$ H100 80GB GPUs). Table 14 separates the full-budget device-time components, totalling 188 H100-hours for vanilla GRPO, 233 for SWE-TRACE, and 321 for CRR per seed. Table 15 reports the equal-wall-clock training windows, and the reduced-budget controls state their 48 H100-hour limit. Section K reports a separate rollout-dispatch microbatch profile. These scoped costs do not provide a complete per-experiment ledger for every baseline, composition run, and auxiliary diagnostic; we therefore answer No to the universal form of this question.

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics?

Answer: [Yes]

Justification: The work uses public training data (SWE-GYM, SWE-REBENCH) and public evaluation benchmarks (SWE-BENCH VERIFIED, SWE-BENCH LIVE, SWE-REBENCH). No human subjects are involved. No personally identifying information is collected, processed, or released. We comply with the dual-use considerations discussed in the broader-impact paragraph of Section 6.

10. Broader impacts

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

Answer: [Yes]

Justification: Section 6 explicitly discusses both directions. Positive impacts include accelerating legitimate software development and improving the auditability of process-supervised RL through transparent fork-and-replay traces. Negative impacts include the dual-use risk of stronger code agents accelerating the production of malicious software and the risk that CRR-trained agents amplify verifier exploitation if the test suite is itself unsafe; we recommend pairing CRR-trained agents with adversarial test-suite hardening [Yu et al., 2026] and execution-free corroboration signals [Shum et al., 2025].

11. Safeguards

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

Answer: [\[Yes\]](#)

Justification: We do not release a pretrained model checkpoint that is novel relative to publicly available QWEN3-14B. We will release training code, configuration files, and the counterfactual trajectory dataset alongside the camera-ready paper; the dataset contains only metadata (task identifiers, action sequences, terminal returns) and does not include scraped private code. The released artefacts will be subject to the same licence as the underlying SWE-GYM corpus.

12. Licenses for existing assets

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

Answer: [\[Yes\]](#)

Justification: SWE-GYM [\[Pan et al., 2025\]](#) and SWE-REBENCH [\[Badertdinov et al., 2025\]](#) are used under their respective public licences and cited at every use. The official model card for QWEN3-14B [\[Yang et al., 2025\]](#) lists the checkpoint under the Apache 2.0 licence. SKYRL-AGENT [\[Cao et al., 2025\]](#) and SWE-MINISANDBOX [\[Yuan et al., 2026\]](#) are used under their open-source licences. Every cited baseline and benchmark is credited in Sections 2 and 5.1.

13. New assets

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

Answer: [\[No\]](#)

Justification: This version does not bundle the new counterfactual trajectory dataset. The dataset is described in Section N, including task identifiers, policy seed/version, selected index, sandbox snapshot identifiers, realised and counterfactual actions, replayable action suffixes, terminal returns, proposal log-probability, and selector score; a README and JSON schema are planned alongside the dataset at the camera-ready stage.

14. Crowdsourcing and research with human subjects

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

Answer: [\[N/A\]](#)

Justification: This work does not involve crowdsourcing or research with human subjects. All evaluations are automatic against deterministic test suites.

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

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

Answer: [\[N/A\]](#)

Justification: This work does not involve human subjects and therefore no IRB review is applicable.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research?

Answer: [\[Yes\]](#)

Justification: The agent policy itself is an LLM (QWEN3-14B), which is a core component of the method and is described in Section 5.1.