COMPASSPLAY: REWARDING THE PROPOSER FOR WHERE IT MOVES THE SOLVER

Anonymous authors
Paper under double-blind review

ABSTRACT

In self-play, a proposer generates verifiable tasks to train a solver. Proposer rewards often depend on the solver's success rate, but equally difficult tasks can differ in their training value. We introduce CompassPlay, a self-play method that rewards the proposer through gradient alignment. The reward favors tasks whose solver loss gradients align with those of reference tasks representing the target capabilities. It draws on a first-order approximation to learning progress and scores each eligible task without additional solver training. Our experiments show gains in performance and training efficiency. In coding self-play with Qwen2.5-Coder-7B, a small external reference set guides task generation. CompassPlay improves average accuracy over AZR's difficulty reward by 1.5 percentage points on in-domain coding and 2.7 on out-of-domain mathematics. In Lean 4 theorem proving, CompassPlay matches the difficulty baseline's 150-iteration cumulative coverage with 40% fewer GPU-hours.

1 INTRODUCTION

Self-play turns task generation into part of reasoning-model training. A proposer creates problems, a solver attempts them, and a verifier checks the answers (Zhao et al., 2025; Bailey et al., 2026). This loop can expand training beyond a fixed collection of human-written problems. Its value depends on the tasks that the proposer learns to generate. The verifier checks whether a solution is correct, but does not establish how much training on generated tasks would help the solver. The proposer therefore needs a reward that connects task generation to progress on the capabilities we want to improve.

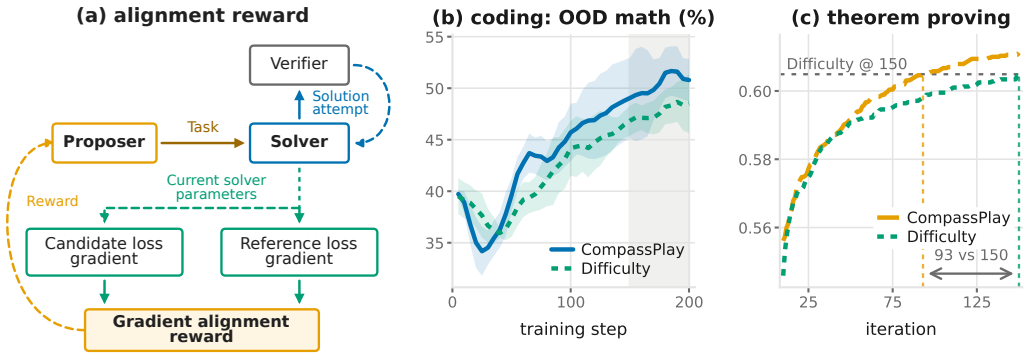


Figure 1: **(a)** CompassPlay rewards alignment between the solver's loss gradients on verified candidate solutions and on reference solutions. **(b)** Mean accuracy on five out-of-domain mathematics benchmarks during coding self-play. Curves show the mean and ± 1 s.d. across three seeds, averaged over three evaluation checkpoints. **(c)** Lean self-play with novelty weighting: cumulative coverage of 3,323 target theorems from iteration 10, with one run per method.

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

A common strategy favors tasks near the solver's current ability (Zhao et al., 2025; Huang et al., 2025). The solver's estimated success rate provides an inexpensive signal for finding such tasks. This encourages a curriculum that adapts as the solver improves. However, tasks with the same success rate can exercise different skills and contribute differently to target performance. A reward based only on success rate cannot distinguish these contributions.

One way to estimate training value is to train on candidate tasks and measure the resulting improvement. SOAR follows this principle (Sundaram et al., 2026). It trains a copy of the solver on a group of generated problems and evaluates the copy on reference problems. The accuracy gain becomes a shared reward for the group. This measures learning progress directly, but requires additional training and evaluation for each group. We seek a reward that estimates training value for individual tasks without these additional solver-training runs.

Gradient-based signals provide a natural proxy for this purpose. Prior work uses gradients to estimate training value for data selection (Xia et al., 2024; Yang et al., 2026) and influence-based feedback to guide synthetic-data generation (Li et al., 2025b; Fan et al., 2026). Building on these ideas, we introduce **CompassPlay**, a self-play method that rewards the proposer through gradient alignment. For a proposed task with a verified solution, CompassPlay compares the solver's loss gradient on that solution with the gradient of a reference loss for the target capabilities (Figure 1a). To first order, a small gradient step on the task lowers the reference loss in proportion to the inner product of the two gradients. The inner product also grows with the task's gradient norm, which can rise without better alignment, so CompassPlay rewards the cosine of the two gradients instead (§2.2). Both gradients are taken at the solver's current parameters, so the reference direction moves as the solver learns, and each eligible task receives its own score without additional solver training.

We evaluate CompassPlay on two different domains: coding and theorem proving. We construct the reference from the supervision available in each domain. In coding self-play, the reference is a fixed external set of problems with gold solutions, which is not included in proposer prompts. In our Lean setup, we use no external gold-proof reference set. We instead use the solver's verified proofs of solved seed theorems as local references. Here, the proposer also sees the seed statement. We observe that optimizing alignment in this setting encourages renamed copies of the seed. Weighting alignment by a novelty factor relative to the seed suppresses this behavior (§4.2). Whether reference tasks appear in proposer prompts therefore matters for reward design.

Our contributions are as follows.

- We introduce CompassPlay, a proposer reward based on the cosine between candidate and reference gradients (§2).
- We show that a proposer prompted with the seed behind its reference learns to copy it, and that a novelty factor suppresses this copying (§4.2).
- CompassPlay improves over AZR's difficulty reward (Zhao et al., 2025) on all eight coding and mathematics benchmarks, by 1.5 points on coding and 2.7 on out-of-domain mathematics (§3.1; Figure 1b). In Lean 4, it matches the difficulty baseline's coverage with 40% fewer GPU-hours (§4.3; Figure 1c).

2 METHOD

CompassPlay changes the proposer reward within the self-play loop. We first describe the self-play setting (§2.1), then present the reward design and reference choices (§2.2).

2.1 SELF-PLAY WITH A PROPOSER AND A SOLVER

In a classic self-play setting, a proposer policy π_ϕ generates tasks for a solver policy π_θ to attempt (Figure 1a). Their parameter vectors are ϕ and θ, respectively. The two policies may share weights, as in our coding experiments. We denote a task by x and a solver attempt by y. Given a conditioning input z (past tasks or a seed theorem), the proposer samples a candidate task $\tilde{x} \sim \pi_\phi(\cdot \mid z)$. The solver then samples k attempts $y_1, \ldots, y_k \sim \pi_\theta(\cdot \mid \tilde{x})$. A verifier v (a Python executor or the Lean 4 compiler) assigns each attempt a binary correctness reward $v(\tilde{x}, y_j) \in \{0, 1\}$. The resulting solve rate is $s(\tilde{x}) = \frac{1}{k} \sum_{j=1}^{k} v(\tilde{x}, y_j)$. The solver uses v as its reward for policy-gradient reinforcement

learning (RL). The proposer maximizes the expectation of its scalar reward $r(\tilde{x})$:

$$\max_{\phi} \mathbb{E}_{\tilde{x}\sim\pi_\phi(\cdot|z)}\big[r(\tilde{x})\big]. \tag{1}$$

A common difficulty reward is $r(\tilde{x}) = 1 - s(\tilde{x})$ for tasks with $0 < s(\tilde{x}) < 1$, optionally weighted by a judge score (Zhao et al., 2025; Bailey et al., 2026).

2.2 CompassPlay: Gradient alignment as the proposer reward

The difficulty term $1 - s(\tilde{x})$ gives the same score to tasks with the same solve rate. We propose to assess each task's training value through its effect on a reference loss. CompassPlay estimates this effect using first-order gradients.

Candidate utility. To formalize learning benefit, let $\mathcal{L}_{\text{ref}}(\theta)$ be a reference loss for the target problems. For a held-out reference set, we use the average loss on its gold solutions. When no such set is available, we use a local reference described below. Let $g(\tilde{x})$ denote the gradient of the solver's loss on a candidate paired with a verified solution. All gradients are taken with respect to the solver parameters θ. We define the candidate's *utility* as the decrease in $\mathcal{L}_{\text{ref}}$ under a hypothetical gradient step:

$$U(\tilde{x}) \;=\; \mathcal{L}_{\text{ref}}(\theta) \;-\; \mathcal{L}_{\text{ref}}\big(\theta - \eta\, g(\tilde{x})\big), \qquad g(\tilde{x}) = \nabla_\theta\, \ell\big(\tilde{x}, y^\star(\tilde{x}); \theta\big), \tag{2}$$

where $\ell(x, y; \theta)$ is the solver's cross-entropy (CE) loss, averaged over solution tokens. Here, $y^\star(\tilde{x})$ is a verified solution of the candidate, and η is the step size. This hypothetical loss-gradient step defines the scoring proxy. Actual solver training uses the policy-gradient objective described above.

First-order approximation. We write g_{ref} for the gradient of the reference loss. The symbols $\langle\cdot,\cdot\rangle$ and $\|\cdot\|$ denote the Euclidean inner product and norm, respectively. For a small step size η, equation 2 expands to

$$U(\tilde{x}) \;=\; \eta\,\big\langle g(\tilde{x}),\, g_{\text{ref}}\big\rangle \;+\; O(\eta^2), \qquad g_{\text{ref}} = \nabla_\theta \mathcal{L}_{\text{ref}}(\theta). \tag{3}$$

This first-order term corresponds to the single-step influence estimate underlying TracIn (Pruthi et al., 2020). GradAlign (Yang et al., 2026) uses a related gradient-alignment criterion to select RL training problems. Evaluating this approximation requires no optimizer step or second-order derivatives. Both gradients are taken at the solver's *current* parameters, so the reference direction evolves as the solver learns.

From inner product to cosine. The inner product in equation 3 depends on both gradient norms and directional alignment. For positive alignment, a larger candidate gradient norm can raise the score without improving that alignment. CompassPlay therefore normalizes both gradients and uses their cosine similarity as its base proposer reward:

$$r_{\cos}(\tilde{x}) \;=\; \cos\big(g(\tilde{x}),\, g_{\text{ref}}\big) \;=\; \frac{\langle g(\tilde{x}), g_{\text{ref}}\rangle}{\|g(\tilde{x})\|\,\|g_{\text{ref}}\|}. \tag{4}$$

The resulting reward is bounded in $[-1, 1]$ and invariant to positive rescaling of either gradient. We compare cosine and inner-product rewards in our coding ablation (§3.3; Appendix C.2).

The reference gradient. CompassPlay constructs $\mathcal{L}_{\text{ref}}$ from the reference data available in each domain. In coding self-play, we use a held-out *external* reference set $\mathcal{D}_{\text{ref}} = \{(x_j, y_j^\star)\}$ of problems with gold solutions, which is not included in proposer prompts (§3 describes the set). The reference loss is $\mathcal{L}_{\text{ref}}(\theta) = \frac{1}{|\mathcal{D}_{\text{ref}}|}\sum_j \ell(x_j, y_j^\star; \theta)$, and g_{ref} is its gradient at the current θ.

In our theorem-proving setup, we use no external gold-proof reference set. The solver's own verified proofs provide the reference supervision. Each candidate is generated from a seed statement that the solver has already proved. We use the seed's verified proof as a *local* reference, with $g_{\text{ref}} = \nabla_\theta \ell(x_{\text{seed}}, y^\star_{\text{seed}}; \theta)$. The reward favors conjectures whose proof gradients align with this reference gradient. The key distinction is whether reference tasks appear in proposer prompts. In the local setting, the proposer sees the seed statement whose verified proof defines g_{ref}. We leave the dependence of $g(\tilde{x})$ and g_{ref} on θ implicit. For local references, g_{ref} also depends on the candidate's seed.

The candidate gradient. CompassPlay computes $g(\tilde{x})$ from the solver's loss on a candidate paired with a *verified* solution. Unverified attempts are not used to compute this gradient. In coding self-play, we use the oracle answer in the solver's answer format. Depending on the task type, this answer is the executed output, the input supplied by the proposer, or the proposer's program. In theorem proving, we average the loss gradients over up to m verified proofs generated by the solver for each conjecture, with $m = 4$ in our experiments. We compute gradients with respect to all solver parameters and accumulate the cosine in single precision.

Which candidates are rewarded. CompassPlay applies the alignment reward to candidates solved in some but not all sampled attempts ($0 < s(\tilde{x}) < 1$). In theorem proving, the base loop also requires $s(\tilde{x})$ to be at most the 70th percentile of solve rates in the batch. Candidates outside these criteria receive no alignment reward. We handle them as in the base loop: a fixed penalty in coding and exclusion from the proposer batch in theorem proving. These rules govern proposer rewards; the solver retains its domain's training objective and task filters. The proposer update retains the base loop's advantage estimator, reward normalization within each batch, and KL and clipping terms.

Lower computational cost. CompassPlay is computationally cheaper than measuring learning benefit through repeated solver training and evaluation. It scores candidates individually using first-order gradients, without the additional training runs required by SOAR (Sundaram et al., 2026) (Appendix A). The added work consists of forward/backward passes on verified candidate and reference solutions. A shared reference gradient can be reused across candidates while the solver parameters remain fixed. Scoring does not update model parameters, and temporary gradients are discarded afterward. Relative to difficulty rewards, which reuse the solve-rate rollouts, the extra cost is the gradient computation. We report measured costs and training efficiency in §3.2, §4.3, and Appendix B.

Reward variants. In coding self-play, CompassPlay uses the cosine reward $r_{\cos}(\tilde{x})$ from equation 4. In theorem proving, alignment with a local reference can reward copies of the seed (§4.1). We therefore multiply the cosine reward by a novelty factor. Let $J(\tilde{x}, x_{\text{seed}}) \in [0, 1]$ denote the token-Jaccard similarity between the candidate statement $\tilde{x}$ and its seed x_{seed}. The resulting reward is

$$r_{\cos \times \text{nov}}(\tilde{x}) = r_{\cos}(\tilde{x}) \left(1 - J(\tilde{x}, x_{\text{seed}})\right). \tag{5}$$

In the experiments, CompassPlay (COS) uses $r_{\cos}$, while CompassPlay (COS × NOVELTY) uses $r_{\cos \times \text{nov}}$. Appendix F summarizes both instantiations and gives the self-play loop in Algorithm 1, using the same similarity function J.

3 General reasoning with coding self-play

Setup. We evaluate CompassPlay in the coding self-play setting of AZR (Zhao et al., 2025). The proposer and solver share a single Qwen2.5-Coder-7B model (Hui et al., 2024). Tasks are built from Python programs and verified by execution. Task types, the solver objective, and the training configuration follow AZR. We compare three configurations:

- **SOLVE-ONLY**: only the solver objective is optimized. Proposals still come from the shared weights, which change as the solver learns.
- **AZR (DIFFICULTY)**: the difficulty reward of AZR, $1 - s$ for $0 < s < 1$.
- **CompassPlay (COS)**: the cosine gradient-alignment reward in equation 4.

AZR and CompassPlay differ only in the proposer reward. For CompassPlay, g_{ref} is the cross-entropy gradient of gold solutions to 32 problems from the MATH training split. This reference set is fixed throughout training, disjoint from evaluation, and not included in proposer prompts. Mathematics problems are used only for the reference gradient; solver-training tasks remain coding problems.

Each configuration runs for 200 steps. We evaluate with greedy decoding every 5 steps on three coding benchmarks: LiveCodeBench (Jain et al., 2025), CRUXEval-I, and CRUXEval-O (Gu et al.,

Table 1: In-domain coding accuracy (%). Trained methods are averaged over steps 150–200. CompassPlay and AZR report mean ± sample s.d. across three seeds; SOLVE-ONLY has one run. Initial model is Qwen2.5-Coder-7B before self-play. Mean averages benchmarks equally. Gain over AZR is the CompassPlay improvement in percentage points (pp).

Method	LiveCodeBench	CRUXEval-I	CRUXEval-O	Mean
Initial model	40.02	48.98	50.89	46.63
SOLVE-ONLY	46.78	58.05	60.20	55.01
AZR	49.87 ± 1.62	59.63 ± 0.12	61.23 ± 0.47	56.91 ± 0.72
CompassPlay (COS)	52.60 ± 0.17	60.08 ± 0.34	62.56 ± 0.44	**58.41 ± 0.18**
Gain over AZR (pp)	+2.73	+0.45	+1.33	**+1.50**

Table 2: Out-of-domain mathematics accuracy (%), using the same averaging as Table 1. Initial model reports accuracy before self-play. Gain over AZR is the CompassPlay improvement in percentage points (pp).

Method	MATH500	AMC23	GSM8K	Minerva	OlympiadBench	Mean
Initial model	53.40	45.00	67.00	18.00	19.30	40.54
SOLVE-ONLY	66.11	37.73	78.25	26.84	31.22	48.03
AZR	60.73 ± 2.63	43.94 ± 0.92	76.98 ± 1.83	26.54 ± 1.40	30.40 ± 2.04	47.72 ± 1.07
CompassPlay (COS)	63.34 ± 3.77	47.12 ± 6.56	78.72 ± 1.95	29.14 ± 2.34	33.53 ± 1.58	**50.37 ± 2.87**
Gain over AZR (pp)	+2.61	+3.18	+1.74	+2.60	+3.14	**+2.65**

2024). We also evaluate on five mathematics benchmarks: MATH500 (Lightman et al., 2024), AMC23, GSM8K (Cobbe et al., 2021), Minerva Math (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). Mathematics is out of domain with respect to the generated training tasks. CompassPlay and AZR use the same three random seeds; SOLVE-ONLY has one run. Appendix E gives benchmark and evaluation details.

3.1 MAIN RESULTS

Tables 1 and 2 average the 11 evaluations at steps 150–200 within each run, then report the mean and sample standard deviation across runs. The Mean column weights benchmarks equally.

CompassPlay improves mean accuracy over AZR on all eight benchmarks. The average gains are 1.50 percentage points on coding and 2.65 on mathematics. Against the single SOLVE-ONLY run, CompassPlay gains 3.40 and 2.34 percentage points on coding and mathematics, respectively. Figure 2 shows a sustained lead for CompassPlay over AZR late in training on LiveCodeBench.

3.2 COMPARISON WITH SOAR

CompassPlay estimates each candidate's training value from gradient alignment. SOAR (Sundaram et al., 2026) instead rewards candidate groups by the reference-accuracy gain after training a solver copy on each group. We compare the downstream accuracy and computational cost of these approaches. No public implementation was available, so we reimplemented SOAR on our training stack. It shares the base model, proposer, and outer optimizer with the other methods. Appendix A describes the inner loop, reference set, and deviations from the original recipe. We ran this baseline once for 20 outer steps.

Figure 3 shows that CompassPlay achieves higher final accuracy at lower total training cost than our SOAR reimplementation. A 200-step CompassPlay run uses 95 GPU-hours on average across three seeds, while the single 20-step SOAR run uses 109 GPU-hours.

The cost difference reflects how the rewards are computed. In our SOAR reimplementation, each outer step requires 16 inner RL runs; the observed rewards take 2–4 distinct values shared across 256 candidates. CompassPlay instead scores each eligible candidate using candidate and reference

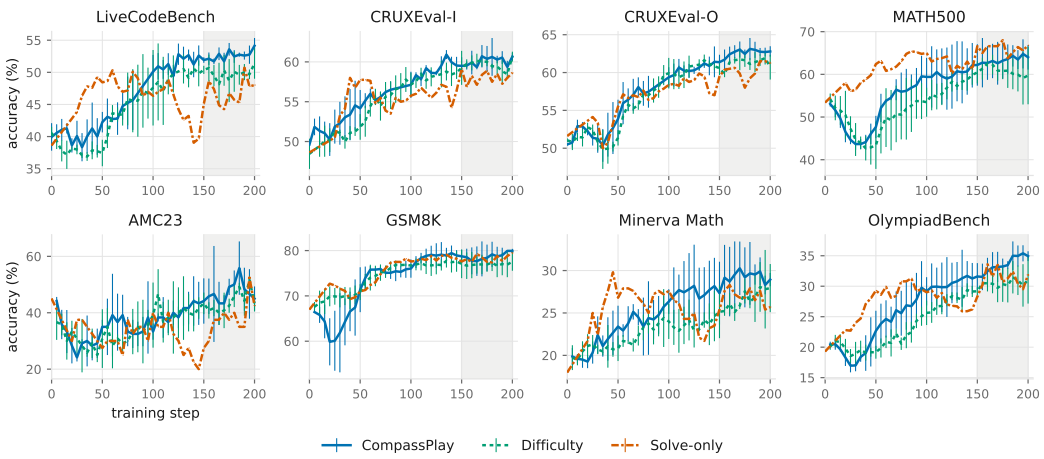


Figure 2: Accuracy over training on three coding and five mathematics benchmarks. The figure labels AZR as DIFFICULTY. These two methods report means over three seeds with ± 1 sample s.d. error bars; SOLVE-ONLY is a single run. The shaded band marks steps 150–200, the averaging window in Tables 1 and 2.

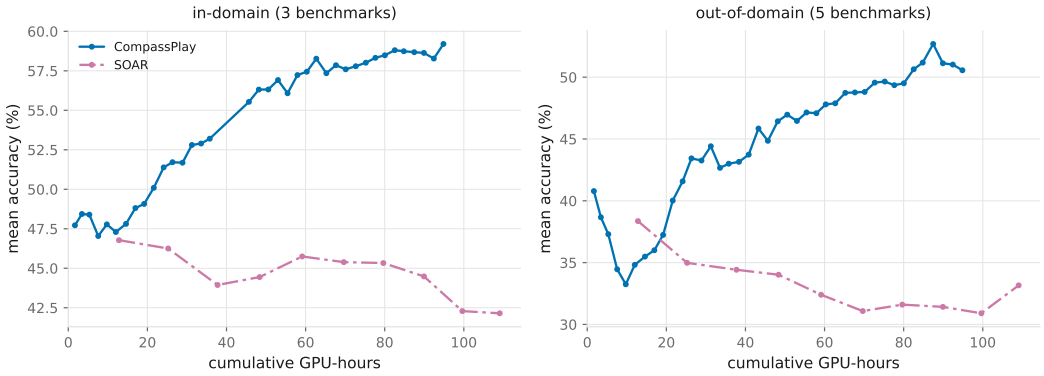


Figure 3: Mean coding and mathematics accuracy against cumulative GPU-hours. CompassPlay (COS) reports mean accuracy and cost across three seeds; SOAR is our single-run reimplementation. Each point is an evaluation checkpoint. GPU-hours are training wall-clock time multiplied by the number of GPUs, excluding benchmark evaluation. Appendix B gives the comparison with AZR at matched GPU-hours.

gradients, without training a solver copy. Appendix B reports the per-phase costs and the comparison with AZR at matched GPU-hours.

3.3 ABLATIONS

We test the two design choices in §2.2: which gradients represent the candidate and the reference, and how the two gradients are compared. Each configuration is run once.

CE and REINFORCE gradients. The solver is trained with policy-gradient RL, whereas CompassPlay computes both gradients from the CE loss on verified or gold solutions (§2.2). We therefore test whether gradients of the RL objective give a better reward. We replace CE gradients with REINFORCE estimates of the gradient of the expected verifier reward on the reference side alone or on both sides (Williams, 1992). The CE configuration outperforms both variants at the CE run's final compute budget. With a shared seed, the CE run uses 93 GPU-hours for 200 training steps, versus 161 with REINFORCE gradients on both sides (Appendix C.1).

Cosine and dot-product rewards. The first-order utility in equation 3 is the inner product, which also grows with the candidate gradient norm. We test whether normalizing it to the cosine improves training. Cosine achieves higher mean accuracy than the dot product over the final quarter of training: 58.31% vs. 55.99% on coding and 48.13% vs. 45.16% on mathematics. Appendix C.2 gives the settings and learning curves.

4 THEOREM PROVING WITH LEAN SELF-PLAY

Setup. We evaluate CompassPlay on the 3,323 target theorems of SGS (Bailey et al., 2026) using separate DeepSeek-Prover-V2-7B models for the proposer and solver (Ren et al., 2025). We use full-parameter updates and check proofs with Lean 4 (de Moura & Ullrich, 2021) against mathlib4 (The mathlib Community, 2020). Our implementation adapts the SGS training framework to condition conjectures on solved target theorems (seeds); the original SGS recipe conditions on unsolved targets. For each candidate, g_{ref} is computed from its seed's verified proof at the current solver parameters. We use no external gold-proof reference set.

In each iteration, the solver attempts every unsolved target theorem. We report cumulative coverage on this target set: the fraction of target theorems solved at least once. Each method is run once.

Methods. We compare proposer-training strategies within the same adapted setup:

- **SOLVE-ONLY**: the proposer is frozen, and only the solver is trained.
- **SGS reward**: the SGS guide score times the difficulty term $1 - s$.
- **DIFFICULTY**: the solve-rate reward $1 - s$.
- **CompassPlay (COS)**: the cosine reward in equation 4, $\cos(g(\tilde{x}), g_{\text{ref}})$, with the local reference.
- **CompassPlay (COS × NOVELTY)**: alignment weighted by novelty relative to the seed (§4.2).

4.1 COPYING WITH A LOCAL REFERENCE

The proposer sees the seed whose proof defines its local reference. Repeating that seed can therefore offer a shortcut to high alignment without introducing a new task. We measure copying by the fraction of eligible conjectures with token-Jaccard similarity above 0.8 to their seed. In Figure 4, the copy rate of CompassPlay (COS) reaches 96% at iteration 75, while DIFFICULTY falls below 1% by iteration 40. A separate name-normalization audit confirms that many conjectures are pure renamings of their seeds (Appendix D).

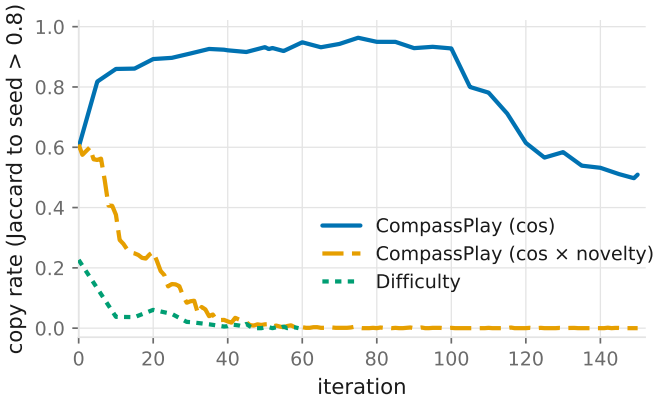
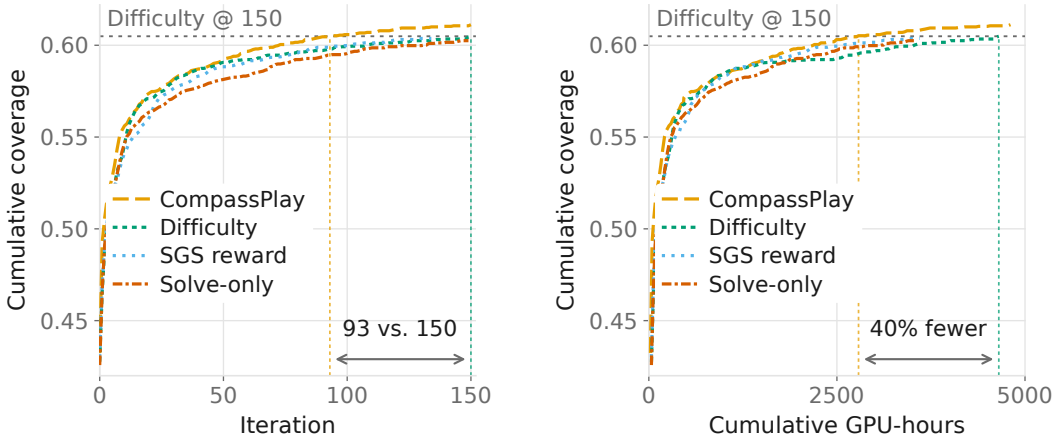


Figure 4: Copy rate: the fraction of eligible conjectures with token-Jaccard similarity to the seed above 0.8. The COS and COS × NOVELTY curves denote the two CompassPlay variants.



(a) (b)

Figure 5: Cumulative coverage of CompassPlay (COS × NOVELTY) and baselines at available checkpoints through iteration 150. The dotted line marks DIFFICULTY's coverage at iteration 150. GPU-hours are computed as training wall-clock time times 8 GPUs.

Table 3: Ablation of CompassPlay (COS × NOVELTY). Copy rate is measured at iteration 115. Gain is the mean number of additional targets solved over DIFFICULTY, using available paired checkpoints in iterations 15–115.

Method	Alignment	Novelty	Copy rate ↓	Gain (targets) ↑
CompassPlay (COS)	✓	×	0.712	+3.1
NOVELTY-ONLY	×	✓	0.000	−7.9
CompassPlay (COS × NOVELTY)	✓	✓	**0.000**	**+13.6**

4.2 NOVELTY-WEIGHTED ALIGNMENT

To reduce copying in this setting, we multiply the COS reward by novelty relative to the seed, giving CompassPlay (COS × NOVELTY) in equation 5. The novelty factor is zero at token-Jaccard similarity 1 and shrinks the alignment score's magnitude as overlap increases. We retain eligible candidates in the proposer batch and apply the base loop's reward normalization.

With CompassPlay (COS × NOVELTY), the copy rate stays below 0.5% over iterations 60–150 (Figure 4). At the audited checkpoints, the pure-rename rates are 1.0% for COS × NOVELTY and 77.5% for COS (Appendix D).

4.3 COVERAGE AND COMPUTATIONAL COST

CompassPlay (COS × NOVELTY) reaches DIFFICULTY's 150-iteration cumulative coverage in 93 iterations (Figure 5a). Reaching that coverage costs 2,787 versus 4,652 GPU-hours, a 40% reduction (Figure 5b). At iteration 150, the two methods have solved 2,031 vs. 2,010 targets.

4.4 ABLATION

We remove each factor of the COS × NOVELTY reward in turn (Table 3). The novelty-only variant uses the novelty factor $1 - J(\tilde{x}, x_{\text{seed}})$ alone as the proposer reward. CompassPlay (COS × NOVELTY) and the novelty-only variant both have zero copy rate at iteration 115, yet their mean gains over DIFFICULTY across iterations 15–115 are +13.6 and −7.9 solved targets, respectively. The combined reward therefore achieves better coverage than novelty alone in these runs.

5 Related work

Proposer rewards in self-play. Many self-play methods use solver feedback to guide task generation (Zhao et al., 2025; Bailey et al., 2026; Dong & Ma, 2025; Poesia et al., 2024). Some favor tasks near the solver's current ability using estimates of success or uncertainty (Huang et al., 2025; Liu et al., 2025). Others reward tasks that cause solver failure (Zhang et al., 2025; Kuba et al., 2025). SGS supplements difficulty with an LLM guide score (Bailey et al., 2026). GASP uses hard goalpost problems to guide generation (Jana et al., 2026), while Pu et al. (2026) study data gating and reward grounding. SOAR rewards task groups by the gain in reference accuracy after training a solver copy on them (Sundaram et al., 2026). CompassPlay uses gradient alignment as a proxy for learning progress. Each eligible candidate receives a score without these additional solver-training runs.

Learnability signals. Automatic curricula prioritize tasks using intermediate success rates (Florensa et al., 2018), learning progress (Matiisen et al., 2017), or regret (Dennis et al., 2020; Parker-Holder et al., 2022). In LLM reinforcement learning, SEC uses a learnability-based curriculum (Chen et al., 2025), and LearnAlign uses gradient alignment for data selection (Li et al., 2025a). CompassPlay uses a related alignment signal to train a proposer to generate tasks.

Influence-based data selection and generation. TracIn estimates data influence by accumulating gradient inner products across training checkpoints (Pruthi et al., 2020). LESS selects instruction data using gradient similarity to target examples (Xia et al., 2024). MATES learns an influence model for pretraining data selection (Yu et al., 2024). GradAlign uses alignment between candidate and validation gradients for RL data selection (Yang et al., 2026). Influence feedback can also guide generation. Montessori-Instruct trains a teacher using changes in reference loss for individual training examples and updates the student between rounds (Li et al., 2025b). OptimSyn uses optimizer-aware gradient influence to reward a rubric generator (Fan et al., 2026). CompassPlay uses gradient alignment in self-play with executable or formal verification. Both proposer and solver learn throughout training, and the reward uses gradients at the current solver parameters. In Lean, we observe substantial copying when the proposer sees the seed defining its local reference. Novelty weighting reduces this behavior (§4.2). In our coding setup, reference tasks are not included in proposer prompts and no novelty factor is used.

6 Conclusion

CompassPlay rewards proposed tasks through gradient alignment with a reference objective. It scores each eligible candidate without training a separate solver copy. In coding self-play, CompassPlay uses an external reference and improves average accuracy over AZR on both coding and mathematics. In Lean, the solver's verified seed proofs provide local references, but the proposer can learn to copy the seeds it sees. With novelty weighting, CompassPlay reduces copying and matches the difficulty baseline's 150-iteration cumulative coverage with 40% fewer GPU-hours. These results support gradient alignment as a practical proposer reward and highlight copying as a concern when reference tasks are also used to prompt the proposer.

Reproducibility statement

Section 2 defines the proposer reward (Equations 4 and 5) and the self-play loop it plugs into. Sections 3 and 4 describe each domain's training setup and its adaptations to the cited framework. Appendix E lists the hardware, benchmarks and evaluation protocol, Appendix A describes our reimplementation of the SOAR baseline, and Appendix B breaks down the compute cost of every method.

AI use statement

We used generative AI tools to assist with translation and to edit the text of this paper for readability. We did not use them to generate data, develop theoretical models, formulate or prove mathematical claims, propose hypotheses, design experiments, implement methods, process data, or interpret results. The authors reviewed all AI-edited text for accuracy. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Luke Bailey, Kaiyue Wen, Kefan Dong, Tatsunori Hashimoto, and Tengyu Ma. Scaling self-play with self-guidance. *arXiv preprint arXiv:2604.20209*, 2026.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for LLM reasoning. *arXiv preprint arXiv:2505.14970*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction (CADE)*, 2021.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025.

Zhiting Fan, Ruizhe Chen, Tianxiang Hu, Ru Peng, Zenan Huang, Haokai Xu, Yixin Chen, Jian Wu, Junbo Zhao, and Zuozhu Liu. OptimSyn: Influence-guided rubrics optimization for synthetic data generation. In *International Conference on Learning Representations (ICLR)*, 2026.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*, 2018.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *International Conference on Machine Learning (ICML)*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization. *arXiv preprint arXiv:2501.03262v9*, 2025. URL https://arxiv.org/abs/2501.03262v9.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

Swadesh Jana, Cansu Sancaktar, Tomáš Daniš, Georg Martius, Antonio Orvieto, and Pavel Kolev. GASP: Guided asymmetric self-play for coding LLMs. *arXiv preprint arXiv:2603.15957*, 2026.

Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training. *arXiv preprint arXiv:2509.07414*, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shipeng Li, Zhiqin Yang, Shikun Li, Xiaobo Xia, Hengyu Liu, Xinghua Zhang, Gaode Chen, Dong Fang, Ying Tai, and Zhe Peng. LearnAlign: Data selection for LLM reinforcement learning with improved gradient alignment. *arXiv preprint arXiv:2506.11480*, 2025a.

Xiaochuan Li, Zichun Yu, and Chenyan Xiong. Montessori-Instruct: Generate influential training data tailored for student learning. In *International Conference on Learning Representations (ICLR)*, 2025b.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. SPICE: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*, 2017.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning (ICML)*, 2022.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Sophia Xiao Pu, Zhaotian Weng, Chengzhi Liu, Jayanth Srinivasa, Gaowen Liu, William Yang Wang, and Xin Eric Wang. Survive or collapse: The asymmetric roles of data gating and reward grounding in self-play RL. *arXiv preprint arXiv:2605.22217*, 2026.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

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

Shobhita Sundaram, John Quan, Ariel Kwiatkowski, Kartik Ahuja, Yann Ollivier, and Julia Kempe. Teaching models to teach themselves: Reasoning at the edge of learnability. *arXiv preprint arXiv:2601.18778*, 2026.

The mathlib Community. The Lean mathematical library. In *ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2020.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning (ICML)*, 2024.

Ningyuan Yang, Weihua Du, Weiwei Sun, Sean Welleck, and Yiming Yang. GradAlign: Gradient-aligned data selection for LLM reinforcement learning. *arXiv preprint arXiv:2602.21492*, 2026.

Zichun Yu, Spandan Das, and Chenyan Xiong. MATES: Model-aware data selection for efficient pretraining with data influence models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Zhengxin Zhang, Chengyu Huang, Aochong Oliver Li, and Claire Cardie. Better LLM reasoning via dual-play. *arXiv preprint arXiv:2511.11881*, 2025.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute Zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

A THE SOAR BASELINE

SOAR (Sundaram et al., 2026) is an asymmetric self-play method whose proposer reward measures solver learning progress. Our reimplementation samples $g = 4$ groups of $n = 64$ candidate problems per outer step. It trains a solver copy on each group for $r = 4$ repeated inner RL runs of 10 steps, evaluates the copy on a held-out reference set of hard problems, and rolls it back. Every candidate in a group receives the group's accuracy gain:

$$R(X_k) = \mathrm{Acc}(\theta'_k, Q_R) - \mathrm{Acc}(\theta, Q_R).$$

Here X_k is the k-th group, Q_R is the reference set, θ denotes the current solver parameters, and $\mathrm{Acc}(\theta'_k, Q_R)$ is the reference accuracy after training on X_k, averaged over the r repeated runs.

Each outer step of our reimplementation requires $g \times r = 16$ inner RL runs and 17 reference-set evaluations. It produces at most g distinct reward values per outer step (2–4 observed in our run), with each group reward shared by all 64 candidates in that group. CompassPlay (COS) assigns a score to each eligible candidate individually (Figure 3).

No public implementation of SOAR was available, so we reimplemented it from the paper on our training stack, keeping the base model, the proposer and the outer optimizer the same as for the other methods. Following the paper, the reference set is drawn from held-out problems the base model cannot solve (a pool of $\sim$370 MATH-train problems with 0/32 successes, 92 resampled every outer step); the reference pool of COS is drawn from the same held-out MATH-train split but without the difficulty filter. The recipe could not be followed exactly on a single-model self-play stack; the five deviations are: the inner loop uses the outer REINFORCE++ (Hu et al., 2025) objective instead of RLOO (Ahmadian et al., 2024) because the stack samples one rollout per problem; one problem type per inner step; the group reward replaces the whole proposer reward rather than only its difficulty term; promotion updates weights but not the teacher on that step, and not the optimizer state. SOAR was run once, for 20 outer steps. Its outer step contains 16 inner RL runs and 17 reference-set evaluations (measured on a smoke run: about 10.8 s per inner step and 20 s per evaluation); its 20 outer steps cost 109 GPU-hours in total.

B PER-STEP COST BREAKDOWN

GPU-hours are training wall-clock hours multiplied by the number of GPUs. Coding and Lean runs use different GPUs (Appendix E), so GPU-hours are comparable within a domain but not across domains. Periodic benchmark evaluations are excluded; SOAR's internal reference evaluations are included. To locate the overhead of CompassPlay (COS), Table 4 compares it with AZR (DIFFICULTY). The configurations share the training loop and differ in the proposer reward. Per-phase timings come from the training framework's timers on one run of each method (8 GPUs, mean over the run). A step is split into three phases: *proposer reward*, the solve-rate estimation rollouts and reward computation, including gradient alignment for COS; *optimizer update*, the forward/backward pass and parameter update on the training batch; and *remainder*, candidate and solution generation, log-probability passes, weight synchronization to the sampler, and bookkeeping. Cells are seconds per step, with the phase's share of the step in parentheses.

Table 4: Per-phase wall-clock of one training step (seconds; share of the step in parentheses), one run of each method on 8 GPUs, mean over the run.

reward	proposer reward (s)	optimizer update (s)	remainder (s)	step total (s)
DIFFICULTY	72.1 (44%)	13.7 (8%)	77.1 (47%)	162.9
COS (ours)	128.8 (58%)	13.4 (6%)	80.2 (36%)	222.4

Most of the COS overhead is in the proposer-reward phase (+56.7 s of about 60 s per step), which includes candidate and reference gradient computation. Optimizer-update time remains similar (13.7 vs. 13.4 s).

The optimizer update accounts for under 10% of a step. SOLVE-ONLY has a similar total training cost to DIFFICULTY (70 vs. 71 GPU-hours per 200 steps). It still samples 192 proposals per step from the shared weights, has the solver attempt them, and runs the same solve-rate estimation. It trains

about a third of the tokens, with the PPO (Schulman et al., 2017) mini-batch halved automatically, but optimizer updates account for only a small fraction of the total cost. DIFFICULTY computes its reward from the existing solve-rate rollouts without additional gradient computation.

Matched-GPU-hours readout for the methods whose per-step cost differs. Table 5 gives, for each budget, the last evaluation point with cumulative cost within the budget.

Table 5: Mean accuracy (%) at matched GPU-hours. For CompassPlay (COS) and AZR (DIFFICULTY), accuracy and cumulative cost are averaged across three seeds at each evaluation step; SOAR is our single-run reimplementation. Each budget uses the last step whose mean cumulative cost is within budget. The endpoint row gives final-checkpoint accuracy and total GPU-hours in parentheses; these are distinct from the final-quarter averages in Tables 1 and 2.

	in-domain			OOD		
budget (GPU-h)	COS	DIFFICULTY	SOAR	COS	DIFFICULTY	SOAR
20	49.07	48.94	46.78	37.24	39.33	38.36
40	53.20	54.98	43.94	43.15	43.65	34.42
60	57.23	57.38	45.75	47.09	46.85	32.40
69	57.86	56.90	45.75	48.77	48.10	32.40
endpoint	59.20 (95)	57.60 (71)	42.15 (109)	50.56 (95)	47.93 (71)	33.16 (109)

C ABLATIONS ON THE CODING TASK

The two ablations summarized in §3.3, in full.

C.1 THE REFERENCE GRADIENT

Question. The reward $\cos(g(\tilde{x}), g_{\text{ref}})$ needs a reference gradient g_{ref} that stands for "progress on the held-out problems". The main method uses the cross-entropy (CE) loss gradient on gold solutions. This is cheap and low-variance, but the solver itself is trained with RL. Does a REINFORCE loss-gradient estimate give a better signal, and at what cost?

Methods. We compare three gradient configurations. The reference-sampling settings also differ; Appendix E gives those used by the REINFORCE variants.

- **COS (CE reference)**: both sides use CE loss gradients of gold solutions. The CE loss is averaged over solution tokens. This is the main method.
- **REINFORCE reference only**: we use a REINFORCE (Williams, 1992) estimate of $\nabla_\theta \mathbb{E}[v(x, y)]$ on held-out problems (Appendix E), where $v(x, y)$ is the solver's verifier reward. For alignment, the reference gradient is the negative of this estimate. The candidate side keeps the CE loss gradient.
- **REINFORCE both sides**: both sides use the negative of their respective REINFORCE reward-gradient estimates, so both are loss gradients.

Each configuration is a single run with the same seed. Figure 6 has the form of Figure 3.

Results. At the CE run's final compute budget (93 GPU-hours), COS has higher in-domain and OOD accuracy than both REINFORCE configurations. The configuration with REINFORCE gradients on both sides uses 161 GPU-hours for 200 steps, compared with 93 for COS with the same seed, a factor of $1.7\times$. The reference-only variant combines a REINFORCE loss-gradient estimate on the reference with CE loss gradients on candidates. Its OOD accuracy drops below 30% in the first 30 steps and ends below COS.

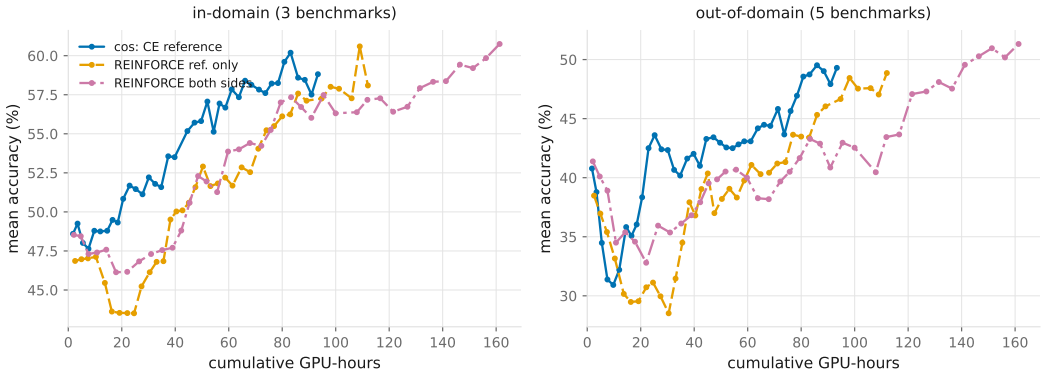


Figure 6: Reference-gradient ablation: CE reference (the main method) against REINFORCE reference gradients, accuracy against cumulative GPU-hours. Single runs with a shared seed.

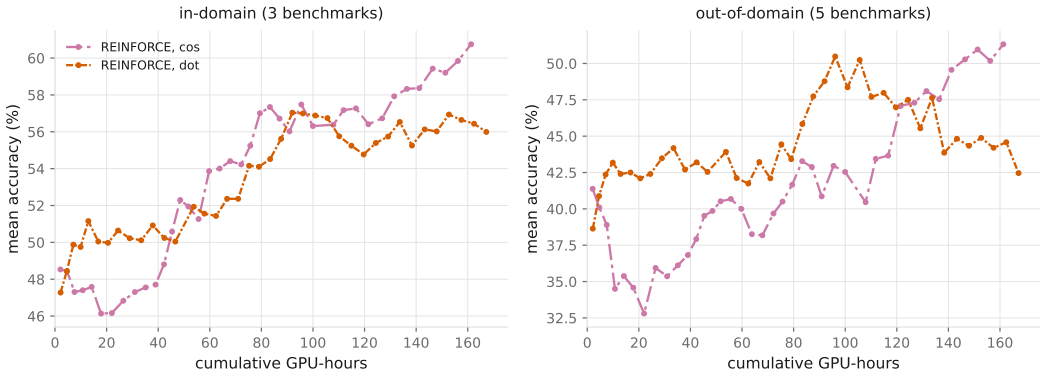


Figure 7: Cosine against dot product as the alignment score, accuracy against cumulative GPU-hours, with REINFORCE gradients on both sides (Appendix C.1). The two runs share the seed and differ in one flag.

C.2 Cosine vs dot product

Question. The dot product $\langle g(\tilde{x}), g_{\text{ref}} \rangle$ in the first-order approximation depends on both gradient norms and directional alignment. For positive alignment, a larger candidate gradient norm $\|g(\tilde{x})\|$ can raise the score without changing the gradient direction. Cosine is invariant to positive rescaling of either gradient. Does this normalization improve training?

Methods. Two runs that share the seed and differ in one flag, cosine vs dot, both with REINFORCE gradients on both sides (the REINFORCE-both-sides configuration of Appendix C.1 is the cosine member). Figure 7 has the form of Figure 3.

Results. Dot leads OOD early and mid-run, the curves cross around step 160, and **cosine pulls away from there**: over the final quarter of the run it is ahead by 2.3 points in-domain and 3.0 OOD. Dot peaks mid-run on both axes and then declines, whereas cosine peaks at the last step. For cosine versus dot product, mean accuracy over steps 150–200 is 58.31% versus 55.99% in-domain and 48.13% versus 45.16% OOD. This is the averaging window of Tables 1 and 2, applied here unchanged.

D PURE-RENAME AUDIT

For each method, the eligible conjectures of one iteration are compared with their seed after normalizing theorem names, bound variable names and whitespace. *pure-rename* is exact equality after normalization; *near-copy* is a token-Jaccard of at least 0.9 after normalization; *Jaccard* > 0.8 is the raw-token criterion used in Figure 4. All values in percent.

The raw-token rates here are recomputed for the audit. The COS and COS $\times$ NOVELTY curves use online measurements with slightly different tokenization, so the audited and plotted Jaccard rates at the same iteration can differ.

Table 6: Pure-rename and token-overlap rates at each method's audited iteration (percent).

Method	Iteration	Pure-rename %	Near-copy %	Jaccard > 0.8 %
CompassPlay (COS)	39	**77.50**	92.20	91.06
CompassPlay (COS $\times$ NOVELTY)	40	**1.01**	4.48	1.46
DIFFICULTY	39	**0.27**	2.45	0.54

E EXPERIMENTAL DETAILS

Hardware. Coding runs use 8 AMD MI355X GPUs; Lean runs use 8 AMD MI250X GPUs. GPU-hours are therefore comparable within a domain but not across domains.

Benchmarks (Section 3). In-domain: LiveCodeBench (Jain et al., 2025), competitive-programming problems collected after the base model's training cutoff, scored by hidden tests; CRUXEval-I and CRUXEval-O (Gu et al., 2024), predicting the input, respectively the output, of a short Python function, the two directions of the code-reasoning task that self-play trains on. Out-of-domain: MATH500 (Lightman et al., 2024), a 500-problem subset of the MATH competition benchmark (Hendrycks et al., 2021); AMC23, the 40 problems of the 2023 American Mathematics Competitions; GSM8K (Cobbe et al., 2021), grade-school word problems; Minerva Math (Lewkowycz et al., 2022), undergraduate STEM problems; and OlympiadBench (He et al., 2024), olympiad-level mathematics. Evaluation is greedy (temperature 0, $k = 1$).

Evaluation window. Tables 1 and 2 average over the final quarter of training, steps 150–200, to reduce sensitivity to any single checkpoint. Greedy evaluations fluctuate from one checkpoint to the next (Figure 2).

REINFORCE reference gradient (Appendix C). The REINFORCE estimator targets $\nabla_\theta \mathbb{E}[v(x, y)]$. The expectation averages over reference problems x and solver responses $y \sim \pi_\theta(\cdot \mid x)$; $v(x, y)$ is the binary correctness reward defined in §2.1. For alignment, we use the corresponding loss gradient: the negative of this reward-gradient estimate. We use 8 samples per problem on a pool of 128 held-out problems, refreshed every 25 steps.

F INSTANTIATION DETAILS AND PSEUDOCODE

The coding instantiation uses the AZR training loop (Zhao et al., 2025). The Lean instantiation adapts SGS (Bailey et al., 2026) to condition conjectures on solved seed theorems (Section 4). Within each domain's setup, the methods share the training loop and solver objective. They use different proposer rewards, or omit proposer optimization for SOLVE-ONLY. The coding baseline uses AZR's DIFFICULTY reward. The Lean baselines include the SGS guide-score-times-difficulty reward and DIFFICULTY, which omits the guide. Table 7 summarizes the two instantiations; Algorithm 1 gives one iteration. In both domains, eligibility for the alignment reward requires $0 < s < 1$.

Table 7: The two instantiations of the reward.

	coding self-play	theorem-proving self-play
base loop	AZR (Zhao et al., 2025)	adapted SGS (Bailey et al., 2026)
proposer / solver weights	shared, one model	separate copies of one base model
environment v	Python execution	Lean 4 + mathlib4
tasks	program-based deduction, abduction, induction	Lean statements
proposer conditioning z	past self-generated tasks	a seed statement the solver has proved
verified solution $y^\star(\tilde{x})$	oracle answer in the solver's format	up to $m = 4$ verified proofs, averaged
reference g_{ref}	external: mean gradient on 32 held-out MATH problems with gold solutions, fixed set, recomputed at current θ	local: gradient on the seed's own proof
proposer reward	$r_{\cos}$, replacing the difficulty reward for eligible candidates	$r_{\cos \times \text{nov}}$, replacing the SGS judge $\times$ difficulty reward; min-max normalized within the batch as in SGS
eligible candidates	$0 < s < 1$ ($k = 8$)	$0 < s \leq$ the batch's 70th solve-rate percentile ($k = 8$)
proposer update	REINFORCE++ with KL, jointly with the solver	reward-weighted REINFORCE
solver update	REINFORCE++ on binary correctness	REINFORCE on verified proofs of tasks with $s \leq 1/2$
where the reward is computed	inside the reward function, on the training workers	after verification, before the trainer starts, on one device

Algorithm 1 Self-play with a gradient-aligned proposer reward (one iteration).

Require: solver π_θ, proposer π_ϕ, environment v, reference (external $\mathcal{D}_{\text{ref}}$ or local seeds), samples per task k, proofs per candidate m (local only)

candidates $\{\tilde{x}_i\} \leftarrow \pi_\phi(\cdot \mid z_i)$ ▷ propose
for each candidate and each target task x **do**
  sample $y_1, \ldots, y_k \sim \pi_\theta(\cdot \mid x)$; verify $v(x, y_j)$; $s(x) \leftarrow \frac{1}{k} \sum_j v(x, y_j)$ ▷ solve, verify
end for
if external reference **then**
  $g_{\text{ref}} \leftarrow \nabla_\theta \frac{1}{|\mathcal{D}_{\text{ref}}|} \sum_j \ell(x_j, y_j^\star; \theta)$ ▷ once per iteration
end if
for each eligible candidate $\tilde{x}_i$ ($0 < s(\tilde{x}_i) < 1$; Lean: $s(\tilde{x}_i) \leq$ 70th percentile) **do**
  $y^\star(\tilde{x}_i) \leftarrow$ verified solution (oracle answer, or verified proofs)
  $g_i \leftarrow \nabla_\theta \ell(\tilde{x}_i, y^\star(\tilde{x}_i); \theta)$ ▷ no optimizer step
  if local reference **then**
    $g_{\text{ref}} \leftarrow \nabla_\theta \ell(x_{\text{seed},i}, y^\star_{\text{seed},i}; \theta)$
  end if
  $r_i \leftarrow \cos(g_i, g_{\text{ref}})$
  if local reference **then**
    $r_i \leftarrow r_i \cdot \big(1 - J(\tilde{x}_i, x_{\text{seed},i})\big)$ ▷ novelty factor
  end if
end for
ineligible candidates keep the base loop's handling (penalty or exclusion)
update π_θ on verified attempts with reward v ▷ base solver objective
update π_ϕ on candidates with reward r ▷ base proposer objective