GASP: GUIDED ASYMMETRIC SELF-PLAY FOR CODING LLMS

Swadesh Jana[*, 1] **Cansu Sancaktar**[*, 1, 2] **Tomáš Daniš**[1] **Georg Martius**[1, 2]
Antonio Orvieto[2, 3, 4] **Pavel Kolev**[1]

[1]University of Tübingen
[2]Max Planck Institute for Intelligent Systems
[3]ELLIS Institute Tübingen
[4]Tübingen AI Center

ABSTRACT

Asymmetric self-play has emerged as a promising paradigm for post-training large language models, where a teacher continually generates questions for a student to solve at the edge of the student's learnability. Although these methods promise open-ended data generation bootstrapped from no human data, they suffer from one major problem: not all problems that are hard to solve are interesting or informative to improve the overall capabilities of the model. Current asymmetric self-play methods are goal-agnostic with no real grounding. We propose **G**uided **A**symmetric **S**elf-**P**lay (GASP), where grounding is provided by real-data goalpost questions that are identified to pose a hard exploration challenge to the model. During self-play, the teacher first generates an easier variant of a hard question, and then a harder variant of that easier question, with the goal of gradually closing the gap to the goalpost throughout training. Doing so, we improve pass@20 on LiveCodeBench (LCB) by 2.5% over unguided asymmetric self-play, and through the curriculum constructed by the teacher, we manage to solve hard goalpost questions that remain out of reach for all baselines.

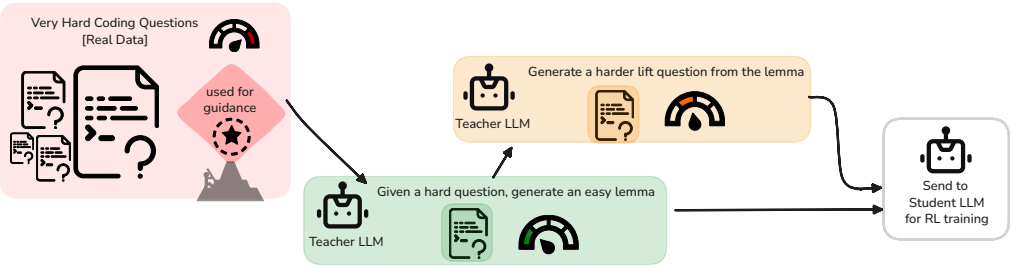


Figure 1: **Overview of GASP.** Self-play is guided by hard real-data coding questions that a standard RLVR run fails to solve. We refer to this subset as our **goalpost** questions. The teacher first generates an easy variant (**lemma**) of this goalpost and then a harder variant (**lift**), producing progressively more challenging questions that push the model's knowledge boundary.

1 INTRODUCTION

Asymmetric self-play aims to keep pushing the student model's abilities by generating problems at the frontier of learning. While such methods have recently shown promise for large language model (LLM) post-training (Zhao et al., 2025), they quickly run into a fundamental bottleneck: not

[*]Equal contribution. Correspondence to: swadeshjana@gmail.com and cansu.sancaktar@tue.mpg.de.

all hard problems at the agent's frontier are interesting or help unlock experiences that improve generalization on the downstream tasks we actually care about.

We argue that the teacher needs explicit guidance, i.e. some far-fetched goalpost, to better ground the *interestingness* of the problems it generates. In the prevailing reinforcement learning (RL) post-training paradigm, problems are typically sampled uniformly at random or from a fixed curriculum defined over a static dataset in a given environment (Moonshot AI, 2025). In practice, a non-trivial portion of this dataset remains consistently unsolved because it poses a hard exploration challenge. We hypothesize that these genuinely hard, real-data questions, which are known to be difficult yet relevant, form ideal goalposts for *guided* self-play.

In this work, we set out to investigate:

1. Can we design a guided self-play algorithm that makes meaningful progress on a designated set of hard questions during RL training? In other words, does steering self-play toward these very hard problems as goalposts actually work?
2. Does such guidance lead the teacher to propose better (i.e., more relevant and informative) problems overall, thereby improving performance on downstream tasks?

We propose **G**uided **A**symmetric **S**elf-**P**lay (**GASP**), where the teacher is guided by a set of hard real-data questions $\mathcal{H}$, namely our goalposts. Given such a target $h \in \mathcal{H}$, the teacher is prompted to generate an easier instance ℓ_0, which we refer to as the **lemma**, that aims to preserve the high-level motif of h. We accept a lemma only if it is *learnable but non-trivial* for the student, ensuring that it still provides a clear learning signal unlike the goalpost. We then prompt the teacher to generate a harder instance ℓ_1 using ℓ_0 as a scaffold; we refer to ℓ_1 as the **lift** question. In this way, ℓ_0 and ℓ_1 serve as stepping stones that build a curriculum. Our ultimate goal is to generate questions at the student's learnability frontier that remain relevant and useful for improving the model's coding capabilities, measured through performance on well-known benchmarks.

We evaluate GASP on `LiveCodeBench` (LCB) and show that goalpost guidance improves downstream coding performance. In particular, GASP consistently outperforms unguided asymmetric self-play (AZR) and is competitive with standard reinforcement learning with verifiable rewards (RLVR) trained on real-data, with improvements most pronounced at larger k. Beyond benchmark gains, GASP makes progress on the goalpost set itself: as training proceeds, some goalpost questions in $\mathcal{H}$ that remain unsolved by all baselines become solvable.

2 Related Work

Asymmetric self-play. Asymmetric self-play has been explored in games (Sukhbaatar et al., 2017) and robotics (OpenAI et al., 2021). More recently, this paradigm has also been applied to RL post-training with LLMs. Poesia et al. (2024) and Dong & Ma (2025) use teacher–student training for theorem proving. Ye et al. (2024) applies similar ideas for alignment, and Liu et al. (2025) and Zhang et al. (2025) apply self-play to math and general reasoning. Most notably and closest to our setting, Absolute Zero (AZR) employs asymmetric self-play in the coding domain. Kuba et al. (2025) also tackles asymmetric self-play in instruction-following, math and coding domains, and Teodorescu et al. (2023) is among early works proposing self-play for coding puzzles, though they only train the teacher. Recent work by Yu et al. (2025) also proposes guided self-play for mathematical and general reasoning. However, their grounding relies on few-shot anchoring to a small labeled dataset, rather than steering toward a designated hard set of real-data goalposts as in GASP. Finally, Wei et al. (2025) proposes self-play for software agents, where an LLM is trained to inject and repair software bugs of increasing complexity in real-world codebases, providing an alternative form of grounding.

Unsupervised environment design and automated curricula. A related line of work studies how to automatically generate and schedule training tasks given a parameterized task space. Parker-Holder et al. (2022) propose ACCEL, which mutates previously generated levels of the environment and prioritizes levels with high student regret, i.e. a gap between the current agent and an optimal agent, to push learning in grid-like game environments. Rutherford et al. (2024) show that commonly used regret approximations can correlate more with success rate than true regret, and propose

alternative signals based on student learnability. This perspective is closely related to the teacher objective used in GASP, which encourages generating questions that are challenging yet solvable.

Meta-learning. Sundaram et al. (2026) propose SOAR, a meta-learning approach that targets hard exploration in RLVR by grounding teacher training in a hard real-data subset in the math domain. At each iteration, the teacher generates synthetic questions for the student, and is rewarded based on the student's measured improvement on the hard subset (used only through this improvement signal). In contrast, GASP does not reward the teacher directly for improving on goalpost questions; solving goalposts is instead a byproduct of learning from lemma–lift stepping stones. A practical limitation of the meta-learning objective is that the improvement-based reward can be sparse or noisy when target questions are extremely hard, and it requires repeated inner-loop training and evaluation to obtain the teacher signal.

3 METHOD

We exclusively work in the coding domain, and focus on `LiveCodeBench` (LCB) (Jain et al., 2024). We aggregate queries between 2024.10 and 2025.02 as our evaluation set (216 questions; referred to as LCB^{v5}), following Yang et al. (2025). We treat all queries before 2024.08 as potential goalpost questions and as our training split, which consists of 601 questions. As goalpost questions are filtered exclusively from the training split (pre-2024.08), we ensure there is no contamination with the evaluation set (2024.10 – 2025.02).

In all our RL experiments and shown baselines, we start from the same base model QWEN2.5-CODER-7B.

3.1 BACKGROUND: STANDARD RLVR VS. ASYMMETRIC SELF-PLAY

In reinforcement learning with verifiable rewards (RLVR) in the coding domain, we start from a static dataset $\mathcal{D}$ of programming problems. In our case, each problem consists of a natural language description and a set of public example tests, with additional hidden test cases used for evaluation. At each iteration of RL training, we sample a problem $d \sim \mathcal{D}$ uniformly at random. We then sample a solution from the model $\pi_\theta(\cdot \mid d)$, verify it by running all public and private test cases, and assign a binary reward based on pass/fail.

Asymmetric self-play, in contrast, generates problems on-the-go. A single model takes on two roles: (1) the teacher proposing questions and (2) the student solving these questions. Unless stated otherwise, π_θ^T and π_θ^S share parameters θ and differ only by role prompting; all updates are applied to the same weights. The teacher is rewarded based on the student's performance on its proposals: questions should be challenging yet solvable for the student. The student receives rewards based on pass/fail on the questions proposed by the teacher. Absolute Zero (AZR) (Zhao et al., 2025) follows this paradigm in the coding domain.

In AZR, the teacher generates three types of coding tasks for the student to solve:

$$\textbf{Induction:} \quad \{(i_p, o_p)\}_{p=1}^{P} \;\Rightarrow\; \hat{f} \qquad \text{(infer } f \text{ from multiple input-output pairs)} \tag{1}$$

$$\textbf{Deduction:} \quad (f, i) \;\Rightarrow\; \hat{o} = f(i) \qquad \text{(predict output given program and input)} \tag{2}$$

$$\textbf{Abduction:} \quad (f, o) \;\Rightarrow\; f(\hat{i}) = o \qquad \text{(infer an input consistent with the output).} \tag{3}$$

Here, the induction task is the most aligned with typical code contest datasets, where a subset of the input-output pairs proposed by the teacher are used as private test cases (i.e. are not shown to the student).

AZR does not start from a human-curated real-world dataset. It makes use of a small dataset of seed questions (256), which are simple generic problems used for bootstrapping. Our method builds upon AZR, but instead of bootstrapping with seed questions and optimizing the teacher solely based on the student's learnability, GASP provides guidance with **goalpost** questions.

3.2 IDENTIFYING CANDIDATE GOALPOST PROBLEMS

Goalpost questions are intended to be hard questions that lie beyond the model's current knowledge boundary and remain unsolved through standard RL training. Our aim is to use these questions to help ground self-play in problems that are relevant for downstream improvement and guide teacher generations.

We identify these goalpost questions via a multi-stage filtering pipeline on the training split (LCB 2023.05 – 2024.08). Concretely, we keep only questions that satisfy pass@100$= 0$ under all of the following evaluations:

1. **Post-RL filter.** On the base model QWEN2.5-CODER-7B, we run three seeds of RL training and evaluate all checkpoints (saved every 50 iterations) from each run. We keep a question only if pass@100$= 0$ for every seed and every checkpoint. We also sample from the base model itself (equivalent to iteration 0 of RL) and discard any question with non-zero pass@100.

2. **AZR-checkpoint filter.** We further filter the remaining set using an AZR-trained checkpoint (initialized from QWEN2.5-CODER-7B), again retaining only questions with pass@100$= 0$.

3. **Final RL filter**. We perform an additional RL run on this hard set to remove any remaining solvable questions.

The final hard set $\mathcal{H}$ consists of 146 problems out of the whole training set (almost 25%), which we use as goalpost questions for guidance in GASP.

3.3 **GASP**: GUIDED ASYMMETRIC SELF-PLAY

GASP proposes an asymmetric self-play framework that is guided by goalpost questions, where we set out to remedy the goal-agnostic nature of AZR-like approaches and ground the teacher in a known notion of interestingness.

At a given iteration of self-play with GASP, the teacher is shown a hard goalpost question $h_i \in \mathcal{H}$ and is first prompted to generate an easier variant, which we call the **lemma** question ℓ_0. We accept ℓ_0 only if its estimated pass rate over N trials falls in a learnable band ($0.3 \leq p \leq 0.7$).

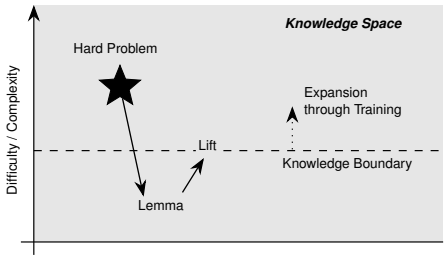


Figure 2: **Illustration of GASP.** Iterative training on generated lemma and lift questions expands the student's knowledge boundary, while the generated questions move closer to the goalpost h.

Conditioned only on an accepted lemma ℓ_0, we then prompt the teacher to generate a harder variant of the lemma, which we refer to as the **lift** question ℓ_1. Importantly, during lift generation the teacher is not shown the original hard question h_i that serves as the goalpost. This is a deliberate choice: conditioning the lift only on the lemma encourages the teacher to increase difficulty incrementally from the student's current frontier, rather than simply copying surface-level features of the goalpost.

Unlike in AZR, the GASP teacher only generates induction questions (Eq. 1); abduction and deduction variants are only introduced in the solver phase (Sec. 3.3.2). For RL updates, we use Task-Relative REINFORCE++ as proposed in Absolute Zero (AZR) (Zhao et al., 2025).

The overall training loop follows three phases (lemma, lift, solver) and is summarized in Alg. 1.

With more and more iterations of GASP, we repeatedly generate lemma-lift pairs (ℓ_0, ℓ_1) and train the student on them. This cycle expands the student's knowledge boundary, pushing it upward as shown in Fig. 2. As the student improves, the teacher can generate increasingly difficult lemma and lift questions that are closer to the original goalpost h, such that some of these goalpost questions initially out of reach become solvable.

Algorithm 1 One global iteration of GASP training

Require: Hard set $\mathcal{H}$; teacher/proposer π_θ^T (i.e. π_θ in teacher role); student/solver π_θ^S (i.e. π_θ in student role); trials N; valid-count target M; Global lemma and lift buffers G_{ℓ_0}, G_{ℓ_1} (empty at the start of GASP training)

Initialize lemma buffer $\mathcal{B}_0 \leftarrow \emptyset$ ▷ **Phase 1: Lemma generation**
while $|\mathcal{B}_0| < M$ **do**
  Sample a goalpost $h \sim \mathcal{H}$
  $\ell_0 \sim \pi_\theta^T(\cdot \mid h)$ ▷ generate lemma from goalpost
  Estimate $p \leftarrow \textsc{PassRate}(\pi_\theta^S, \ell_0, N)$
  Compute lemma reward r_{lemma}
  if $\textsc{IsValid}(\ell_0)$ **and** $0.3 \leq p \leq 0.7$ **then**
    $\mathcal{B}_0 \leftarrow \mathcal{B}_0 \cup \ell_0$
RLUpdate: Send lemma batch $\mathcal{B}_0$ to trainer and update π_θ^T
$G_{\ell_0} \leftarrow G_{\ell_0} \cup \mathcal{B}_0$ ▷ Add local lemma buffer to global buffer for future similarity checks

Initialize lift buffer $\mathcal{B}_1 \leftarrow \emptyset$ ▷ **Phase 2: Lift generation**
while $|\mathcal{B}_1| < M$ **do**
  Select $\ell_0 \sim \mathcal{B}_0$
  $\ell_1 \sim \pi_\theta^T(\cdot \mid \ell_0)$ ▷ generate lift conditioned only on lemma
  Estimate $p \leftarrow \textsc{PassRate}(\pi_\theta^S, \ell_1, N)$
  Compute lift reward r_{lift}
  if $\textsc{IsValid}(\ell_1)$ **and** $0.1 \leq p \leq 0.5$ **then**
    $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup \ell_1$
RLUpdate: Send lift batch $\mathcal{B}_1$ to trainer and update π_θ^T
$G_{\ell_1} \leftarrow G_{\ell_1} \cup \mathcal{B}_1$ ▷ Add local lift buffer to global buffer for future similarity checks

Construct solver training set $\mathcal{D}_S \leftarrow \mathcal{B}_0 \cup \mathcal{B}_1$ ▷ **Phase 3: Solver phase**
Compute solver rewards on $\mathcal{D}_S$ and update solver parameters

3.3.1 Teacher Training

Rewards Our teacher / proposer rewards are based on the learnability metric as proposed in (Rutherford et al., 2024). Let p denote the pass rate, i.e. the solve rate, of the student across N attempts. Learnability is defined as $p(1-p)^\alpha$ with $\alpha = 1$.

We use the standard learnability reward with $\alpha = 1$ for lemma proposals, which peaks at $p = 0.5$ and thus favors questions of intermediate difficulty:

$$r_{\text{lemma}} = \begin{cases} [4\,p\,(1-p)]^5, & \text{if } 0.3 \leq p \leq 0.7 \\ -0.5, & \text{otherwise.} \end{cases} \tag{4}$$

For lift proposals, we use a more skewed learnability reward that peaks at $p = 0.1$, favoring harder questions with lower student pass rates:

$$r_{\text{lift}} = \begin{cases} 10\,p\left(\frac{1-p}{0.9}\right)^9, & \text{if } 0.1 \leq p \leq 0.5 \\ -0.5, & \text{otherwise.} \end{cases} \tag{5}$$

The lemma and lift rewards are instances of a generalized learnability reward as further detailed in Appendix. A.1. The specific exponents and constants are chosen such that the reward curves peak within the desired pass-rate bands and are normalized to [0, 1], as shown in Fig. 3.

Additionally, we assign reward -1 to any lemma/lift candidate that has a format error.

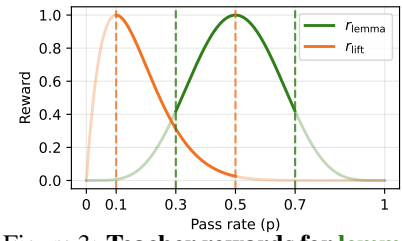


Figure 3: **Teacher rewards for lemma and lift proposals.** We show the learnability curves according to Eq. 4 and Eq. 5, where the vertical lines mark the corresponding valid bands.

Choosing the difficulty axis In GASP, we parameterize the axis along which we adjust the difficulty of a goalpost question h. An induction-style task is defined by (i) an underlying mapping f to be inferred and (ii) the observed input-output examples used to specify it.

In practice, we distinguish two modes of difficulty adjustment:
(1) The **I/O axis** increases (or decreases) instance or representation complexity while aiming to preserve the same underlying algorithmic motif, e.g. by changing the input schema (one list to multiple lists or nested lists) or by selecting examples that make the function f harder to infer.
(2) The **f axis** increases (or decreases) algorithmic complexity by modifying the mapping itself, e.g. by introducing additional constraints or composing new operations that require extra logic beyond the original rule.

For each lemma generation, we uniformly sample which axis to apply (I/O or f). The corresponding lift question then aims to increase the difficulty along the same axis.

Rejection sampling for lemma-lift proposals Each lemma/lift candidate returned by the teacher undergoes lightweight validity checks before it is passed to the solver and admitted to the training buffer (in addition to the pass-rate band checks in Algorithm 1). We follow prior work (Zhao et al., 2025) and reject proposals that are malformed, unsafe to execute, or exhibit non-deterministic behavior under repeated runs.

In addition, we enforce **novelty** with respect to the **global buffer** of previously accepted lemma/lift proposals. Concretely, we compute cosine similarity between embeddings of the proposed question text and generated code and those of previously accepted items. If the similarity exceeds 0.95 for any item in either buffer, the proposal is rejected (see Appendix. B for more details). This diversity filter prevents mode collapse and encourages the teacher to generate a broad set of distinct lemma/lift questions.

3.3.2 Student Training

For valid lemma and lift proposals produced in the teacher phase, we train the student using verifiable rewards based on pass/fail. For each training question, we choose uniformly at random whether to keep it in the induction format or to convert it into a deduction or abduction format. In the latter case, instead of presenting multiple input-output examples as in induction, we present the function f together with either a single input and ask the student to predict the corresponding output (deduction, Eq. 2), or a single output and ask the student to produce a corresponding input (abduction, Eq. 3). Since AZR reported gains from including deduction and abduction tasks, we adopt the same strategy to maintain diversity in the student training phase.

3.4 GASP with Real-data RL

We also test a variant of GASP that performs joint training on real-data and synthetic data generated through guided asymmetric self-play. After completing one GASP iteration as detailed in Algorithm 1, we additionally sample questions from the real-data training split $\mathcal{D}$, corresponding to the `LiveCodeBench` time window 2023.05 – 2024.08, and apply RLVR updates on these samples. Note that this is the same split used to construct our goalpost set, such that $\mathcal{H} \subset \mathcal{D}$.

4 Results

We evaluate GASP and report pass@k on our `LiveCodeBench` evaluation split (LCB^{v5}). We also evaluate GASP + Real-data RL, the joint-training variant described in Section 3.4, and a Real-data RL baseline that applies standard RLVR to the static LCB training split. We further compare against the base model Qwen2.5-Coder-7B and AZR, i.e. unguided self-play.

We find that GASP yields significant gains over the base model and consistently outperforms AZR, supporting the hypothesis that goalpost guidance improves the quality of self-play training signal. GASP is broadly competitive with Real-data RL across k, with slightly higher mean pass@k at larger k, though the difference is modest. Moreover, GASP + Real-data RL improves over GASP, indicating that guided self-play is effective on its own, while also being complementary to real-data training and able to benefit from additional real-world supervision when available. We also evaluate

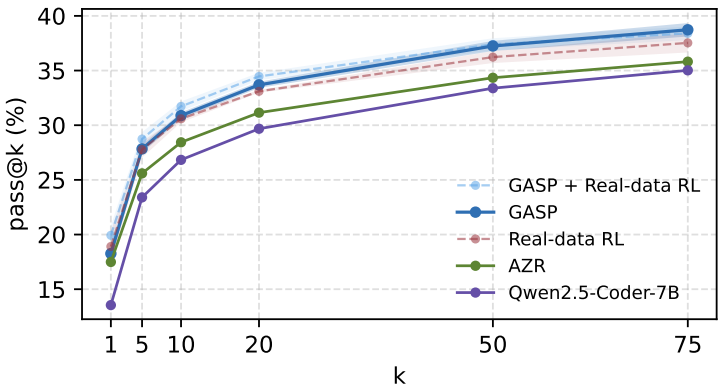


Figure 4: **Pass@k performance on the LCB eval benchmark (LCB^{v5}).** We compare GASP, GASP + Real-data RL, Real-data RL, AZR, and QWEN2.5-CODER-7B. All RL-based results are repeated over three seeds. For each seed of our RL runs, we perform single-checkpoint model selection by choosing the checkpoint that maximizes pass@20 on the LCB^{v5} evaluation split, and report mean $\pm$ std across seeds. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting their checkpoint based on best performance on their evaluation benchmarks (including LCB) over training from a single-seed run. We additionally report pass@1-based checkpoint selection in the appendix (Figure 8), which shows the same qualitative trends.

Table 1: **LCB^{v5} pass@1 and pass@20 results.** We use bold for **global best**, blue underline for best self-play and red for best with real-data training.

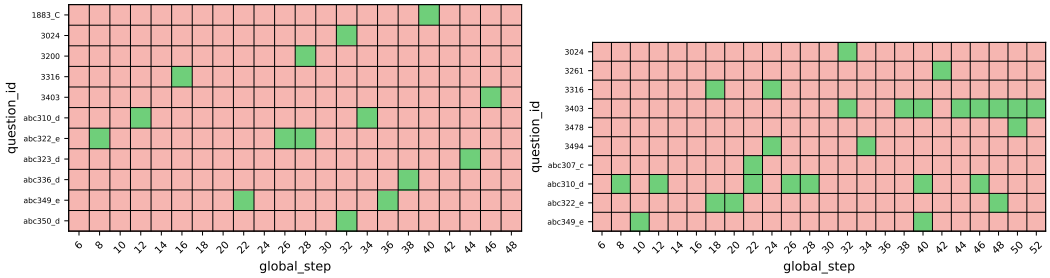
	Real-data Training	Real-data Guidance	LCB^{v5}	
			pass@1	pass@20
GASP (ours)	✗	✓	$18.26_{\pm 0.68}$	$33.69_{\pm 0.28}$
AZR	✗	✗	17.49	31.15
Real-data RL	✓	✗	$18.91_{\pm 1.40}$	$33.10_{\pm 0.12}$
GASP + Real-data RL (ours)	✓	✓	$\mathbf{19.93}_{\pm 0.88}$	$\mathbf{34.46}_{\pm 0.34}$
Qwen2.5-Coder-7B	—	—	13.55	29.68

greedy pass@1 (temperature 0) on LCB^{v5}, HumanEval^{+} (Liu et al., 2023), and MBPP^{+} (Liu et al., 2023) (see Table 3). On the target distribution (LCB^{v5}), GASP and GASP + Real-data RL maintain their advantage. On HumanEval^{+}, GASP closely follows AZR, while GASP + Real-data RL and GASP lead on MBPP^{+}.

Do we need rejection sampling? As described in Sec. 3.3.1, we perform a rejection sampling step in GASP to ensure that the generated lemma and lift proposals maintain sufficient diversity. We run an ablation of GASP without rejection sampling and report pass@k eval performance on LCB^{v5} in Figure 9. Although GASP without rejection sampling still outperforms AZR and shows comparable performance with real-data RL, we observe high variance across seeds.

Upon inspection of generated lemma and lift proposals, we can attribute this high variance to diversity collapse in some of the runs. By rejecting proposals that are overly similar to previously accepted ones, rejection sampling mitigates this failure mode and yields more consistent gains.

Can we solve the goalpost hard questions with GASP? As lemma and lift questions serve as stepping stones toward each goalpost, GASP can provide a curriculum that makes progress on questions in $\mathcal{H}$. By construction, these goalposts remain unsolved throughout the filtering pipeline in Sec. 3.2, and a standard RLVR run trained only on $\mathcal{H}$ also fails to solve any goalpost question over the course of training. Notably, the publicly released AZR checkpoint also solves none of the 146 goalpost questions.



(a) GASP

(b) GASP + Real-data RL

Figure 5: **Goalpost questions solved with GASP and GASP + Real-data RL.** Visualization over training checkpoints (x-axis: global step) and goalpost question IDs (y-axis). We include only goalpost questions that are solved at least once during training. A cell is marked as solved (green) if pass@100 > 0 (i.e., at least one of 100 samples passes); otherwise it is unsolved (red). Results are aggregated across three seeds, where a cell is marked solved if *any* seed solves the question at that checkpoint (union across seeds).

Evaluating checkpoints throughout RL training, GASP solves 11 unique goalpost questions out of 146 (6/3/3 per seed, union of 11 across three seeds), with all seeds making non-trivial progress on questions unsolvable by any baseline. In Figure 5, we visualize when each goalpost becomes solved across training, marking a goalpost as solved at a given checkpoint if it is solved by *any* seed. We report the corresponding per-seed counts for the GASP variants (without rejection sampling (Fig. 14) and GASP + Real-data RL) in Suppl. C.4 and Table 4.

We observe that goalpost questions are solved intermittently rather than persistently across evaluated checkpoints: a question solved at one checkpoint can be unsolved at the next. This is consistent with these questions lying at the frontier of the model's capability, where the correct reasoning path and answer lie in the tail of the sampling distribution. As GASP + Real-data RL showcases more consistent solving across checkpoints, one could argue that additional RL training on real data, which also includes the goalpost questions themselves in the training split, helps consolidate the capabilities unlocked by asymmetric self-play.

We ablate two further design choices: the input-output difficulty axis and the two-stage curriculum in GASP.

Difficulty axis ablation for the teacher. As described in Sec. 3.3.1, GASP uniformly samples between two modes of difficulty adjustment: the I/O axis (input–output complexity) and the f axis (algorithmic complexity). Removing the I/O axis in GASP, such that all difficulty adjustments operate only along the f axis, leads to comparable pass@k performance on LCB^{v5}, but with substantially increased variance among different seeds. We also observe worse goalpost performance (4 unique questions solved vs. 11 for GASP), suggesting that varying input–output complexity helps inject extra diversity into self-play, which is important for generating an effective curriculum. Full ablation results can be found in Appendix. C.2.

Two-stage curriculum ablations. Replacing the lemma–lift curriculum with a single-stage reward targeting intermediate difficulty (one-step medium, reward peak at student pass rate p=0.3) leads to worse pass@k on LCB^{v5} compared to GASP, while targeting only hard questions (one-step hard, peak at student pass rate p=0.1) performs slightly better but still lags behind GASP. On goalpost questions, one-step hard solves only 2 unique questions across 3 runs, suggesting that medium-difficulty stepping stones are important for an effective curriculum. Still, both one-step ablations outperform AZR at higher k, highlighting the value of guided synthetic data generation even without the full two-stage curriculum. See Appendix. C.3 for more details.

We also provide some qualitative examples of the generated lemma and lift questions as shown in Fig. 6 (more examples, including i/o difficulty axis, shown in Fig. 15 and Fig. 16).

Goalpost (Hard) `hard_id: abc351_d` `axis: f`

There is a grid of H rows and W columns. Some cells (possibly zero) contain magnets. The state of the grid is represented by H strings S_1, S_2, ..., S_H of length W. If the j-th character of S_i is #, it indicates that there is a magnet in the cell at the i-th row from the top and j-th column from the left; if it is ., it indicates that the cell is empty. Takahashi, wearing an iron armor, can move in the grid as follows: - If any of the cells vertically or horizontally adjacent to the current cell contains a magnet, he cannot move at all. - Otherwise, he can move to any one of the vertically or horizontally adjacent cells. However, he cannot exit the grid. For each cell without a magnet, define its degree of freedom as the number of cells he can reach by repeatedly moving from that cell. Find the maximum degree of freedom among all cells without magnets in the grid.

↓

Lemma `axis: f`

You will be given a grid representing a maze with walls and open spaces. The grid is filled with '.' for open spaces and '#' for walls. Your task is to find the largest contiguous group of open spaces in the grid. Your code should return the size of this largest contiguous group.

↓

Lift `axis: f`

Your task is to find the largest contiguous group of open spaces in a grid. The grid consists of open spaces represented by '.' and walls by '#'. However, this time, the grid may have diagonal connections between cells, and there will be additional complex patterns of walls. Write a function that solves this problem given such a grid.

Figure 6: **Goalpost→lemma→lift examples from GASP run.** Functional-axis (f) selected for difficulty adjustment on hard question with id `abc351_d`. (shared motif: connectivity/reachability)

5 Discussion

We propose guided self-play with GASP and showcase that goalpost guidance helps generate questions that are not just progressively more difficult but also more relevant for downstream improvement. GASP consistently outperforms goal-agnostic self-play with AZR.

As we take a known hard problem, first making it easier through the *lemma* question and then harder again with the *lift*, we provide a curriculum through these stepping stones. As a result, we are able to solve some of these difficult questions that originally posed a hard-exploration challenge in RL, providing no learning signal on their own.

We find that GASP performs strongly without additional real-data RL updates, while the joint variant GASP + Real-data RL yields further improvements on LCB^{v5}. This indicates that guided self-play can be used for standalone training, but also complements standard RLVR pipelines when real-data supervision is available. The rejection sampling step improves stability by maintaining diversity among lemma/lift proposals, helping prevent overfitting and mode collapse.

Limitations We provide a proof of concept of guided asymmetric self-play in the coding domain, showing promising results, but there are several clear avenues for improvement. First, beyond the diversity-based similarity checks, we do not explicitly validate that the teacher's proposed lemma/lift questions provide *correct* guidance toward the goalpost. As a result, the generated stepping stones may not always align with the target hard question in an obvious way. Similarly, when selecting the difficulty axis, we observe that the input-output axis is often harder for the teacher, and the teacher can default back to increasing functional difficulty, which we do not penalize. Lightweight checks that measure alignment to the goalpost, or introducing a judge/reward model, could address both issues.

Second, we observe a more fundamental failure mode: current LLMs are not always good at abstracting the underlying concept of a task. They often borrow surface-level metaphors from the original problem rather than producing clean conceptual stepping stones. Relatedly, when asked to increase difficulty, they frequently do so by adding extra constraints, which is not always the most

informative form of complexity. Since models can often *judge* stepping-stone quality better than they can generate it, incorporating judge-based rewards can be a future direction.

Additionally, we use LCB pre-2024.08 as our training split, which is our sole real-data source. Scaling GASP with larger code training corpora is a natural next step.

Finally, our framework uses a fixed set of goalposts. An open question is what should happen once a goalpost is reached: ideally, the goalpost set should be updated over time so that guidance remains meaningful as the model improves. We leave dynamic goalpost updating to future work.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the computing resources provided by Max Planck Institute for Intelligent Systems (MPI-IS) and thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Cansu Sancaktar and Tomáš Daniš. Antonio Orvieto acknowledges financial support from the Hector Foundation. Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. This work was supported by the ERC - 101045454 REAL-RL and the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL https://arxiv.org/abs/2502.00212.

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

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

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, 2025. URL https://arxiv.org/abs/2510.24684.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Moonshot AI. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

OpenAI, Alex Paino, Hyeonwoo Noh, Lilian Weng, Qucheng Yuan, Casey Chu, and Wojciech Zaremba. Asymmetric self-play for automatic goal discovery in robotic manipulation, 2021. URL https://arxiv.org/abs/2101.04882.

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*, pp. 17473–17498. PMLR, 2022.

Gabriel Poesia, David Broman, Nick Haber, and Noah D. Goodman. Learning formal mathematics from intrinsic motivation, 2024. URL https://arxiv.org/abs/2407.00695. NeurIPS 2024 (Oral).

Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery, 2024. URL https://arxiv.org/abs/2408.15099.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play, 2017. URL https://arxiv.org/abs/1703.05407.

Shobhita Sundaram, John Quan, Ariel Kwiatkowski, Kartik Ahuja, Yann Ollivier, and Julia Kempe. Teaching models to teach themselves: Reasoning at the edge of learnability, 2026. URL https://arxiv.org/abs/2601.18778.

Laetitia Teodorescu, Cédric Colas, Matthew Bowers, Thomas Carta, and Pierre-Yves Oudeyer. Codeplay: Autotelic learning through collaborative self-play in programming environments. In *IMOL Workshop at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=GgzfIBxa18.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL https://arxiv.org/abs/2502.18449.

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

Ziyu Ye, Rishabh Agarwal, Tianqi Liu, Rishabh Joshi, Sarmishta Velury, Quoc V. Le, Qijun Tan, and Yuan Liu. Scalable reinforcement post-training beyond static human prompts: Evolving alignment via asymmetric self-play, 2024. URL https://arxiv.org/abs/2411.00062.

Wenhao Yu, Zhenwen Liang, Chengsong Huang, Kishan Panaganti, Tianqing Fang, Haitao Mi, and Dong Yu. Guided self-evolving llms with minimal human supervision. *arXiv preprint arXiv:2512.02472*, 2025.

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. *arXiv preprint arXiv:2505.03335*, 2025.

A Appendix

A.1 Generalized Learnability Reward

The lemma and lift rewards (Eq. 4, Eq. 5) are instances of a generalized family of learnability rewards:

$$\mathcal{R}(p;\, a, b) = \left[\frac{p}{a}\left(\frac{1-p}{1-a}\right)^{\frac{1-a}{a}}\right]^{b}, \qquad p \in [0,1],\ a \in (0,1),\ b \in \mathbb{R}. \tag{6}$$

This function attains its maximum at $p = a$, where a controls the target difficulty and b controls the sharpness of the peak. The lemma reward corresponds to $(a{=}0.5, b{=}5)$ and the lift reward to $(a{=}0.1, b{=}1)$.

By construction, the reward is normalized such that its maximum value satisfies $\mathcal{R}^*(p;\cdot) = 1$. This normalization ensures that different reward configurations remain comparable in magnitude while differing only in the region of the difficulty space that they prioritize.

B Buffer Dissimilarity

We monitor *buffer dissimilarity*, which measures how newly generated samples differ from previously generated ones stored in a buffer.

Let $\mathcal{B}_t$ denote the batch at iteration t and $\mathcal{M}_{t-1}$ denote the accumulated buffer from earlier iterations. Buffer dissimilarity is defined as

$$D_{\text{buffer}}(\mathcal{B}_t, \mathcal{M}_{t-1}) = \frac{1}{|\mathcal{B}_t|} \sum_{x \in \mathcal{B}_t} \left(\frac{1}{|\mathcal{M}_{t-1}|} \sum_{x' \in \mathcal{M}_{t-1}} \big(1 - \text{sim}(x, x')\big) \right). \tag{7}$$

This metric provides a notion of temporal novelty. High buffer dissimilarity indicates continued expansion into previously unexplored regions of the task space, whereas decreasing values suggest increasing reuse of rewarded patterns.

C Extended Results

We show a detailed view of the evaluation results on LCB^{v5} for pass@1 and pass@20 in Table 2. All pass@k evaluations use temperature 0.6 with $N = 100$ samples for the pass@k estimator. We also provide pass@k curves for RL runs, where we select each run's checkpoint by maximizing pass@1 in Figure 8. Under pass@1-based checkpoint selection, GASP exhibits a clearer separation from Real-data RL.

Note that all Real-data RL runs were trained until convergence. As reported by Zhao et al. (2025), AZR (with Qwen2.5-Coder-7B) was run for 350 global steps, after which the best checkpoint was chosen according to overall performance on the math and code evaluation benchmarks (including `LiveCodeBench`). GASP with rejection sampling was run for approximately 50 global iterations due to compute budget limitations. As shown in Fig. 7, performance on LCB^{v5} was still improving at the end of training, suggesting further gains are possible with additional compute. Each iteration is more costly than without rejection sampling, as the teacher must generate multiple proposals until a sufficiently novel one is accepted. These gains are achieved despite substantially fewer training iterations than the baselines.

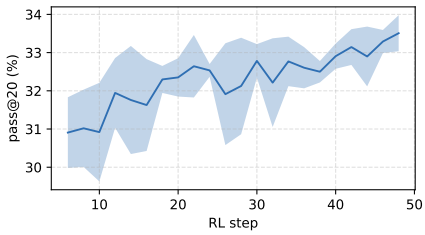


Figure 7: Pass@20 performance on the **LCB^{v5} benchmark throughout RL training for GASP.** Evaluation performance continues to improve throughout training and shows no plateau by the final iteration, where the runs were truncated due to compute constraints.

In Table 3, we report greedy pass@1 (temperature 0) on LCB^{v5}, HumanEval^{+}, and MBPP^{+}. HumanEval^{+}[1] and MBPP^{+}[2] are substantially more saturated benchmarks. On HumanEval^{+}, the smallest benchmark with 164 problems, AZR leads, though all methods remain within a narrow margin. On MBPP^{+}, all methods perform comparably, confirming that self-play training does not degrade general coding ability.

Table 2: **Performance on the LCB^{v5} benchmark.** We compare GASP pass@1 and pass@20, as well as ablations of our method, with standard RLVR training on all LCB questions in our training time split (Real-data RL), as well as performance of the AZR checkpoint open-sourced by Zhao et al. (2025), and the base model QWEN2.5-CODER-7B. We show performance across 3 seeds of RL training respectively for two selection criteria: in each seed run we choose the model checkpoint with the best pass@1 or pass@20 performance. GASP outperforms AZR (unguided self-play) across all selection criteria. We use bold for **global best**, blue underline for best self-play and red for best with real-data training.

		Training on Real-data	Real-data Guidance	*selected by pass@1*		*selected by pass@20*	
				pass@1	pass@20	pass@1	pass@20
	GASP	✗	✓	$19.10_{\pm 0.92}$	$33.63_{\pm 0.35}$	$18.26_{\pm 0.68}$	$33.69_{\pm 0.28}$
Ablations	GASP no i/o axis	✗	✓	$19.36_{\pm 1.35}$	$33.31_{\pm 1.34}$	$18.94_{\pm 1.69}$	$33.47_{\pm 1.14}$
Ablations	GASP w/o rejection sampling	✗	✓	$17.97_{\pm 0.79}$	$32.72_{\pm 1.81}$	$17.64_{\pm 1.33}$	$33.03_{\pm 1.50}$
Ablations	GASP one-step medium	✗	✓	$17.10_{\pm 0.33}$	$31.41_{\pm 0.03}$	$15.73_{\pm 1.61}$	$32.36_{\pm 0.03}$
Ablations	GASP one-step hard	✗	✓	$17.76_{\pm 1.14}$	$32.20_{\pm 0.91}$	$17.13_{\pm 0.68}$	$33.37_{\pm 0.12}$
	AZR	✗	✗	17.49	31.15	17.49	31.15
	Real-data RL	✓	✗	$20.35_{\pm 0.75}$	$30.84_{\pm 1.83}$	$18.91_{\pm 1.40}$	$33.10_{\pm 0.12}$
	GASP + Real-data RL	✓	✓	$\mathbf{20.55}_{\pm 0.26}$	$\mathbf{34.25}_{\pm 0.45}$	$\mathbf{19.93}_{\pm 0.88}$	$\mathbf{34.46}_{\pm 0.34}$
Base	Qwen2.5-Coder-7B	—	—	13.55	29.68	13.55	29.68

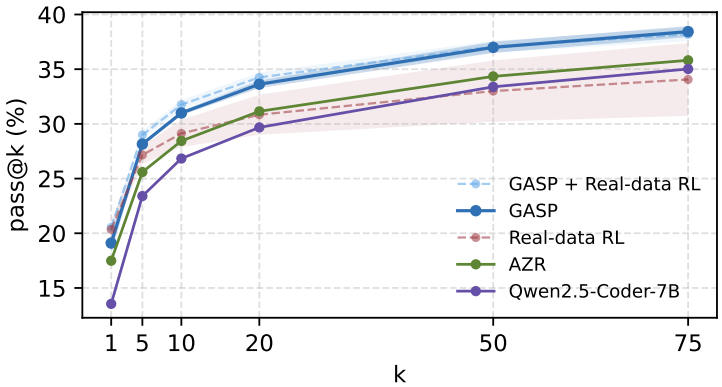


Figure 8: **Pass@k performance on the LCB eval benchmark (LCB^{v5}) with checkpoint selected for best pass@1.** We compare GASP, GASP + Real-data RL, Real-data RL, AZR, and QWEN2.5-CODER-7B. All RL-based results are repeated over three seeds. For each seed of our RL runs, we select the checkpoint with the best pass@1 within that run, and report mean ± std across seeds. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting the best checkpoint over training from a single-seed run.

C.1 ABLATION: REJECTION SAMPLING IN GASP

We showcase performance of GASP without rejection sampling in Fig. 9. Compared to GASP with rejection sampling, we observe high variance among seeds, while still seeing improved performance over unguided self-play with AZR. We attribute the high variance to the diversity collapse

[1]HumanEval^{+} extends HumanEval (Chen et al., 2021), a benchmark of 164 Python function completion tasks given a signature and docstring, with more comprehensive test cases.

[2]MBPP^{+} extends the Mostly Basic Python Problems (MBPP) benchmark (Austin et al., 2021) with more comprehensive test cases (378 questions).

Table 3: **Greedy pass@1 performance on LCB^{v5}, HumanEval^{+} and MBPP^{+}.** Checkpoint selected by best LCB pass@20. Mean over seeds; ± shows standard deviation. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting the best checkpoint based on performance on their evaluation benchmarks, including LCB$^{v1\text{-}5}$, HumanEval^{+}, and MBPP^{+} over training from a single-seed run. Shaded area indicates our target distribution, i.e. LCB^{v5}. We use bold for **global best**, blue underline for best self-play and red for best with real-data training.

Method	LCB^{v5}	HumanEval^{+}	MBPP^{+}
GASP	$19.75_{\pm 2.33}$	$79.67_{\pm 0.35}$	$70.63_{\pm 0.70}$
AZR	17.13	**83.54**	69.58
Real-data RL	$18.21_{\pm 1.41}$	$78.66_{\pm 1.61}$	$69.31_{\pm 0.95}$
GASP + Real-data RL	$\mathbf{20.52}_{\pm 3.60}$	$80.49_{\pm 1.61}$	$\mathbf{71.60}_{\pm 1.00}$
Qwen2.5-Coder-7B	13.89	79.88	69.58

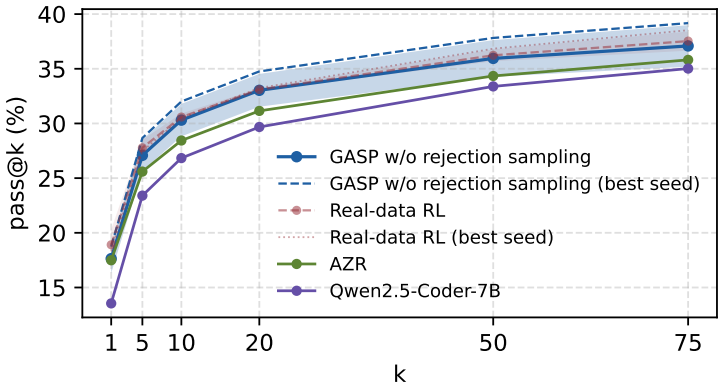


Figure 9: **Pass@k performance on the LCB eval benchmark (LCB^{v5}) for GASP without rejection sampling.** We compare GASP without rejection-sampling, Real-data RL, AZR, and QWEN2.5-CODER-7B. All RL-based results are repeated over three seeds. For each seed of our RL runs, we select the checkpoint with the best pass@20 within that run, and report mean ± std across seeds. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting the best checkpoint over training from a single-seed run.

shown in Fig. 10: the timing of this collapse varies across runs, with seeds that maintain diversity longer achieving stronger benchmark performance. Interestingly, GASP without rejection sampling solves more unique goalpost questions across seeds (14 vs. 11; see Table 4), suggesting that concentrated training on similar problem types can benefit specific hard tasks even as general benchmark consistency suffers.

C.2 Ablation: Removing I/O Difficulty Axis in GASP

We report the performance of GASP without the input-output (i/o) difficulty axis in Fig. 11. Compared to GASP where we adjust difficulty along both the functional and input-output axes, we observe comparable performance, but higher variance among seeds. We hypothesize that the i/o axis helps diversify the synthetic training distribution, resulting in more varied data and more consistent performance across seeds.

C.3 Ablation: One-step Guidance in GASP

We ablate the two-step lemma–lift curriculum by training with only one difficulty level: either medium or hard.

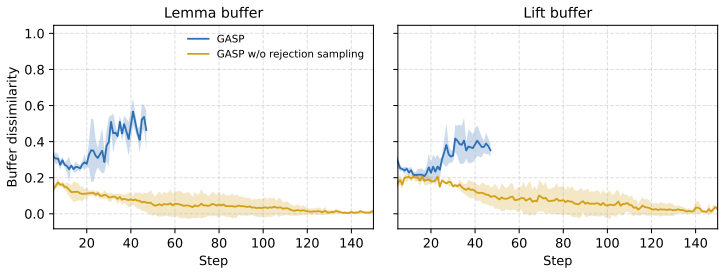


Figure 10: **Dissimilarity of proposed lemma and lift questions with respect to their corresponding buffers for GASP with and without rejection sampling.** We observe that GASP without rejection sampling suffers from diversity collapse as RL training progresses, with proposed questions becoming increasingly similar to those already in the buffer. Results shown for three seeds. (see Appendix. B for more on the buffer dissimilarity computation.)

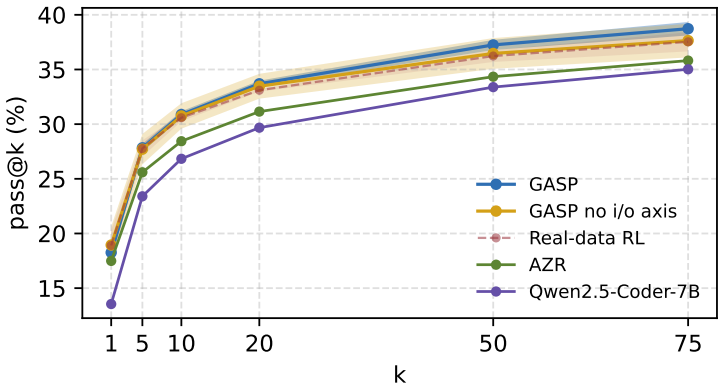


Figure 11: **Pass@k performance on the LCB eval benchmark (LCB^{v5}) for GASP without input-output (i/o) difficulty axis for the teacher proposals.** We compare GASP without the i/o difficulty axis (only relying on functional difficulty axis) with Real-data RL, AZR, and QWEN2.5-CODER-7B. All RL-based results are repeated over three seeds. For each seed of our RL runs, we select the checkpoint with the best pass@20 within that run, and report mean $\pm$ std across seeds. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting the best checkpoint over training from a single-seed run.

One-step ablation rewards. The one-step ablations replace the two-stage lemma–lift curriculum with a single reward using the generalized form $\mathcal{R}(p;\, a, b)$ from Eq. 6:

$$r_{\text{one-step med}} = \begin{cases} \mathcal{R}(p;\, 0.3,\, 5), & \text{if } 0.1 \le p \le 0.5 \\ -0.5, & \text{otherwise,} \end{cases} \tag{8}$$

$$r_{\text{one-step hard}} = \begin{cases} \mathcal{R}(p;\, 0.1,\, 1), & \text{if } 0.1 \le p \le 0.5 \\ -0.5, & \text{otherwise.} \end{cases} \tag{9}$$

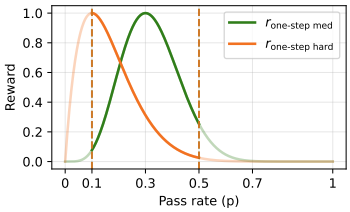


Figure 12: **Teacher rewards for one-step medium and one-step hard proposals.**

The one-step medium reward peaks at $p = 0.3$, targeting intermediate difficulty, while the one-step hard reward peaks at $p = 0.1$, matching the lift reward. Both use the same reward band $[0.1, 0.5]$, as shown in Fig. 12.

Results Both one-step variants underperform GASP on LCB (Table 2), with the gap most visible at pass@1. The one-step hard variant achieves slightly better benchmark performance than one-step medium, despite solving far fewer goalpost questions (2 vs. 11, Table 4). This is consistent with the hypothesis that harder training questions benefit downstream evaluation, while easier lemma questions serve as stepping stones toward the goalposts. The two-step curriculum appears to combine

both effects, though further investigation is needed to disentangle their individual contributions more precisely.

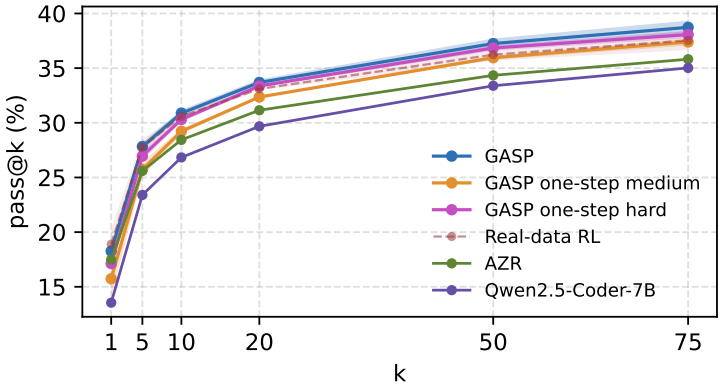


Figure 13: **Pass@k performance on the LCB eval benchmark (LCB^{v5}) for GASP with one-step instead of the two-step lemma and lift generation.** We compare GASP with a one-step curriculum (intermediate difficulty: one-step medium, hard: one-step hard), Real-data RL, AZR, and QWEN2.5-CODER-7B. All RL-based results are repeated over three seeds. For each seed of our RL runs, we select the checkpoint with the best pass@20 within that run, and report mean $\pm$ std across seeds. For AZR, we evaluate the authors' publicly released checkpoint (trained from QWEN2.5-CODER-7B); the authors report selecting the best checkpoint over training from a single-seed run.

C.4 SOLVED GOALPOST QUESTIONS

We report the number of unique goalpost questions in $\mathcal{H}$ solved across training checkpoints in Table 4. A goalpost is counted as solved if pass@100 > 0 at any checkpoint within a run, counted at most once per seed. Although GASP without rejection sampling solves 14 unique goalposts across seeds (5/8/5) versus 11 for GASP (6/3/3), it also trains for nearly 4$\times$ as many steps (190 vs. 48), giving it substantially more opportunities to encounter goalpost questions. We additionally provide the goalpost-solve plot for the GASP variant without rejection sampling in Fig. 14. As discussed in Appendix. C.1, without rejection sampling, more and more similar lemma and lift questions are proposed by the teacher and are then subsequently solved by the student. We hypothesize that these similar questions might result in a more targeted effort toward the goalpost questions in some instances; however, this comes at the cost of high seed variance on downstream benchmarks. When truncated to the same step range as GASP, GASP without rejection sampling solves only 3 unique goalposts compared to GASP's 11, suggesting that rejection sampling enables more sample-efficient discovery of hard solutions.

D HYPERPARAMETER SETTINGS

Shared settings. All experiments use a batch size of 64, a learning rate of 10^{-6}, and a temperature of 1.0 for training and 0.6 for evaluation. All training is performed on the `LiveCodeBench` training split (pre-2024.08), with evaluations on the LCB^{v5} split as well as HumanEval^{+}, and MBPP^{+}. All experiments use the vLLM (Kwon et al., 2023) and verl (Sheng et al., 2024) frameworks for distributed training and evaluation. Following AZR, we use 4$\times$ H100 GPUs for training and 1$\times$ H100 for evaluation, with code execution managed through the same process as AZR.

RLVR. RLVR training uses GRPO for 500 steps with 3 seeds. For each prompt, n=8 responses are sampled for advantage calculation. We observe saturation of pass@k within 200 steps and a loss of generalization beyond that.

GASP. GASP experiments run for approximately 50 global iteration steps, with evaluations every 2 steps. Each global step involves lemma generation with rejection sampling, lift generation with

Table 4: Unique goalpost questions solved across training checkpoints (out of 146 goalposts). A question is counted if it is solved at any checkpoint within a run (pass@100 > 0). Per-seed counts and union across three seeds are reported. **Bold**: global best. †Zero by construction (goalposts filtered to be unsolvable by these methods).

	Method	Per-seed	Unique (union)
	GASP	6 / 3 / 3	11
Ablations	GASP no i/o axis	2 / 2 / 2	4
Ablations	GASP w/o rejection sampling	5 / 8 / 5	**14**
Ablations	GASP one-step medium	6 / 4 / 4	11
Ablations	GASP one-step hard	0 / 1 / 1	2
	AZR	—	0†
	Real-data RL	—	0†
	GASP + Real-data RL	9 / 2 / 3	10
Base	Qwen2.5-Coder-7B	—	0†

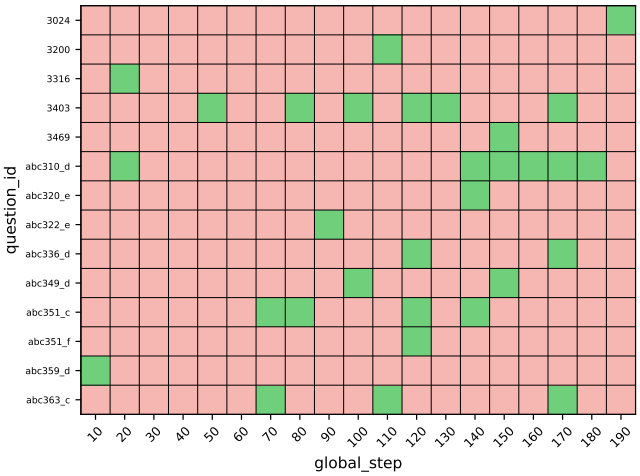


Figure 14: **Goalpost questions solved by GASP without rejection sampling.** Visualization over training checkpoints (x-axis: global step) and goalpost question IDs (y-axis). We include only goalpost questions that are solved at least once during training. A cell is marked as solved (green) if the corresponding question is solved at that checkpoint; otherwise it is unsolved (red). Results are aggregated across three seeds, where a cell is marked solved if *any* seed solves the question at that checkpoint (union across seeds).

rejection sampling, and a solver update. Each rejection sampling step can contain multiple generation steps until conditions are met or `max_attempts` is reached, with one model update per rejection sampling step. As described in Algorithm Alg. 1, there can be multiple lemma generation steps within one global iteration until the minimum count of total generated programs per step is met. GASP models are trained using Task-Relative REINFORCE++, following AZR (Zhao et al., 2025). For comparison, the authors of AZR report training for at least approximately 350 steps.

Ablations without rejection sampling run for approximately 200 global iteration steps, with evaluations every 10 steps. In both cases, evaluation performance had not yet saturated, suggesting room for further improvement.

Solver. We set N=10 for estimating the solver's success rate, based only on induction tasks (not abduction or deduction). For each task, we obtain 5 example input-output pairs: 2 are shown as public examples and 3 are held out for private testing to avoid overfitting.

Goalpost (Hard) `hard_id: abc364_e` `axis: io`

Takahashi has prepared N dishes for Snuke. The dishes are numbered from 1 to N, and dish i has a sweetness of A_i and a saltiness of B_i. Takahashi can arrange these dishes in any order he likes. Snuke will eat the dishes in the order they are arranged, but if at any point the total sweetness of the dishes he has eaten so far exceeds X or the total saltiness exceeds Y, he will not eat any further dishes. Takahashi wants Snuke to eat as many dishes as possible. Find the maximum number of dishes Snuke will eat if Takahashi arranges the dishes optimally.

↓

Lemma `axis: io`

Snuke has a limit for either the sweetness or saltiness, and he wants to know the maximum number of options he can choose from without exceeding the limit. Given a limit X and a list of options with sweetness and saltiness, determine the maximum number of options he can choose.

↓

Lift `axis: io`

Snuke now has two limits, one for the sweetness of food and another for the saltiness. He wants to know the maximum number of options he can choose without either of these limits being exceeded. Given two limits for sweetness and saltiness, and a list of options with their sweetness and saltiness, determine the maximum number of options he can choose without exceeding either limit.

Goalpost (Hard) `hard_id: abc358_e` `axis: io`

AtCoder Land sells tiles with English letters written on them. Takahashi is thinking of making a nameplate by arranging these tiles in a row. Find the number, modulo 998244353, of strings consisting of uppercase English letters with a length between 1 and K, inclusive, that satisfy the following conditions: - For every integer i satisfying $1 < i < 26$, the following holds: - Let a_i be the i-th uppercase English letter in lexicographical order. - The number of occurrences of a_i in the string is between 0 and C_i, inclusive.

↓

Lemma `axis: io`

Design a Python function that takes two arguments: a string of characters and an integer representing the length of the strings to be generated. The function should return the number of unique strings of the given length that can be formed using the characters in the input string. Note that the order of characters matters, and each character can be used any number of times.

↓

Lift `axis: io`

Design a Python function that takes two arguments: a string of characters and an integer representing the length of the strings to be generated. The function should return the number of unique palindromic strings of the given length that can be formed using the characters in the input string. Note that each character can be used any number of times, and the order of characters matters.

Figure 15: **Goalpost→lemma→lift examples from GASP run (with rejection sampling)** Input–output-axis (io) selected for difficulty adjustment on hard questions with id `abc364_e` (top) and `abc358_e` (bottom). Shared motifs: for `abc364_e` maximizing count under cumulative resource constraints (1D→2D feasibility); for `abc358_e` counting strings under per-letter usage caps (mod / DP).

Goalpost (Hard) `hard_id: 3414` `axis: f`

You are given a non-negative integer k. There exists a staircase with an infinite number of stairs, with the lowest stair numbered 0. Alice has an integer jump, with an initial value of 0. She starts on stair 1 and wants to reach stair k using any number of operations. If she is on stair i, in one operation she can: Go down to stair i - 1. This operation cannot be used consecutively or on stair 0. Go up to stair i + 2^{jump}. And then, jump becomes jump + 1. Return the total number of ways Alice can reach stair k.

↓

Lemma `axis: f`

You are given a non-negative integer k. You start at stair 1 and want to reach stair k using any number of operations. In one operation, you can either: 1. Move down to the previous stair (i - 1). 2. Move up to the next two positions (i + 1). Your task is to write a function that returns the total number of ways you can reach stair k starting from stair 1.

↓

Lift `axis: f`

You are given a non-negative integer k and a positive integer moves_limit. You start at stair 0 and want to reach stair k using any number of operations, but you are limited by the moves_limit. In one operation, you can either: 1. Move down to the previous stair (i - 1). 2. Move up to the next two positions (i + 1). Your task is to write a function that returns the total number of ways you can reach stair k starting from stair 0, given the moves_limit.

Figure 16: **Goalpost→lemma→lift examples from GASP run (with rejection sampling)** Function-axis (f) selected for difficulty adjustment on hard question with id `3414`. Shared motif: counting paths / number of ways in a stateful process with constrained moves; DP on position + auxiliary state.