PASSING IS NOT LEARNING: RETHINKING GUIDANCE IN CODING AGENT TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Teacher guidance can turn failed coding attempts into verified successes, but these guided successes do not necessarily yield independent problem solving. We study terminal and repository-repair tasks that a 9B coding agent persistently fails after standard reinforcement learning, asking what guidance leaves behind once removed. Distilling autonomous, hint-assisted, and teacher trajectories improves unaided performance on intervention tasks but yields little gain on held-out tasks when privileged channels are blocked. Audits further show that success filtering also favors trajectories accessing privileged information, while open-channel evaluation can overstate unaided capability. We therefore propose Hint-Twin Coupled Reinforcement Learning (HTC-RL), a two-stage recipe that first distills mixed trajectories under hint-free contexts, then couples reinforcement learning on unhinted rollouts with hint-free online distillation of successful hinted rollouts. The complete recipe achieves 28.1 on Terminal-Bench 2.1 and 57.2 on SWE-bench Verified, 4.5 and 6.4 points above our Stage 0 standard-RL baseline. Against plain RL from the same distilled initialization, its gains extend beyond the initial demonstrations: on 197 terminal tasks absent from those demonstrations but included in continued training, it solves 16 versus 6. On 600 Stage 0-screened hard tasks from repositories excluded from our post-training recipe, it reaches 7.2% unaided pass@8, versus 5.2% for plain RL. Guidance should therefore be judged by the independent capability acquired from it, not simply by the passes it enables.

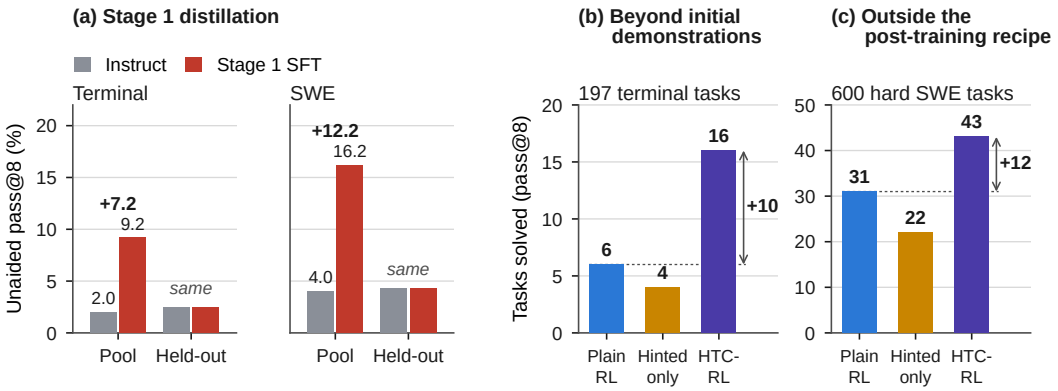

Figure 1: **What verified successes teach, and what continued training adds.** All numbers are unaided pass@8. (a) Stage 1 distillation raises performance on the intervention pools, while held-out performance stays at the Instruct level. (b) From the same Stage 1 checkpoint, guided supervision alone solves 4 of the 197 terminal tasks without Stage 1 demonstrations, and adding autonomous reward learning in HTC-RL solves 16, versus 6 for plain RL. These tasks are trained in Stage 2 and are not a held-out set. (c) On 600 Stage 0-screened hard tasks from two repositories excluded from post-training, HTC-RL solves 43, versus 31 for plain RL and 22 for hinted-only.

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.

1 INTRODUCTION

Reinforcement learning with verifiable rewards has become the standard way to train coding agents that work in terminals and code repositories (Luo et al., 2025; Ivison et al., 2026). Even after such training, some tasks remain unsolved across many attempts and several late checkpoints: no rollout passes verification. Every rollout then receives zero reward, all group-relative advantages vanish, and the tasks the agent most needs to learn contribute no gradient. Teacher guidance can supply the missing successes. A diagnostic hint can help the student solve a task, while a stronger model can provide a successful demonstration directly (Wang et al., 2026; Zelikman et al., 2022; Li et al., 2025; Pan et al., 2025; Yang et al., 2025; Yan et al., 2025; Ma et al., 2025).

Yet a success obtained with guidance does not show that the student can succeed on its own. We therefore separate three quantities: the *acquisition* of successful experience on a task, the student's *unaided execution* on that same task after training, and *transfer* to tasks that never entered training.

Successful experience can arise in three ways: the student retries until it passes, a teacher diagnoses a failed attempt and provides a hint, or the teacher solves the task directly. In every case, the verifier checks the final outcome rather than how it was reached. A trajectory may therefore pass using information unavailable at evaluation: a hint, the teacher's knowledge, or unintended environment information such as future fixes in the repository history, verifier files, or web results that reveal the target solution. We call the last category *privileged channels*. When data is selected only by whether it passes, both genuine skill and reliance on such help are rewarded (Skalse et al., 2022; Pan et al., 2022). We therefore measure *what verified successes actually teach a coding agent* once guidance is removed and known privileged channels are blocked, and ask how training can turn more of these successes into unaided capability.

Our first contribution is a systematic empirical characterization of the teaching value of successful experience (§3). We compare mixtures of autonomous successes, hint-assisted student trajectories, and teacher demonstrations by fine-tuning a fresh Instruct model on each and evaluating it without guidance and with known privileged channels blocked. Three findings emerge. First, the value of a source depends on the domain: hint-assisted trajectories add more on terminal tasks, and teacher demonstrations add more on repository repair. Second, gains on the intervention pools, the tasks on which guidance was collected, do not transfer detectably to our held-out sets. Distillation raises unaided pass@8 on the pools by 7.2 and 12.2 points, while held-out performance remains at the level of the original Instruct model (Figure 1(a)). Third, privileged channels inflate apparent transfer. Selecting teacher trajectories by success raises the share that use these channels from 57.6% to 76.4%, and students trained on any recipe lose most of their held-out successes once the channels are blocked. Together, these results separate acquiring verified successes from acquiring independent capability: intervention-pool gains do not necessarily transfer to held-out tasks, and open channels can overstate apparent transfer.

Our second contribution is Hint-Twin Coupled Reinforcement Learning (HTC-RL), a training method that learns from guided successes while preserving the student's own unhinted reward signal (§4). The student samples each intervention task with and without a hint; we call these two prompt versions its *hint twins*. Unhinted rollouts are trained with a reward-centered policy gradient, while verified hinted successes are distilled online under the original hint-free prompt. Guidance thus supplies new solutions without replacing the learning signal from the student's unaided attempts.

We evaluate this design through a controlled comparison of three ways to use guided successes after distillation (§5). From the same distilled starting point, continuing with guided supervision alone never outperforms plain RL. Adding autonomous reward learning to the same supervision performs best on every set, and its additional pool gains come entirely from tasks absent from the initial demonstrations. On 197 such terminal tasks, HTC-RL solves 16, versus 6 for plain RL and 4 for hinted-only online distillation (Figure 1(b)); in both domains, HTC-RL and plain RL tie on the initially demonstrated tasks. The gains extend to 600 hard tasks from repositories outside our post-training recipe (Figure 1(c)) and to full benchmarks (Table 4).

2 PROBLEM SETUP

Tasks and models. We study two families of interactive coding tasks. *Terminal tasks*, drawn from the Tmax training set (Ivison et al., 2026), give the agent a natural-language instruction and a containerized shell. The agent issues commands, observes their output, and submits when done, and a hidden test script decides success. *Repository-repair tasks*, drawn from R2E-Gym (Jain et al., 2025), present an issue in a real Python repository. The agent edits the code, and hidden unit tests decide success. In both families the reward is binary and produced by a task verifier. The student is Qwen3.5-9B-Instruct (Qwen Team, 2026a). *Stage 0* is our reproduction of the Tmax RL recipe on this model. It produces the autonomous retries and executes the teacher hints used to build training data, and its three latest checkpoints screen for persistently failed tasks.

Experience sources. We collect three kinds of successful experience. **A**: autonomous successes, verified trajectories that the Stage 0 policy produces without help on tasks outside the intervention pool. The other two sources target tasks that Stage 0 cannot solve. A task enters the *intervention pool* when none of the three late Stage 0 checkpoints produces a verified success on it, which yields 251 terminal and 148 repository-repair tasks. **B**: hint-assisted successes on intervention tasks. A teacher model, Kimi K3 (Kimi Team, 2026), reads the task, a compressed failed Stage 0 trajectory, and the private verifier, and writes a short diagnostic hint that may not disclose test names, verifier-only values, code, or exact commands. Hints that fail a leakage screen are discarded. The Stage 0 policy then executes the task with the hint appended, and its verified successes are kept with the hint removed (Appendix C). The hint is used only during training and is never available at evaluation. **K**: teacher demonstrations on intervention tasks. The same teacher solves the task directly, and its verified solutions are converted into the student's message format. *Stage 1* fine-tunes the original Instruct model on A, B, and K at an approximate ratio of 8:1:1, with every example under its hint-free task context. It takes its A and B data from Stage 0, but its weights restart from Instruct.

Task exposure and unaided evaluation. Whether a task was demonstrated during training and whether the student is helped when evaluated are separate questions, and we keep them separate throughout (Appendix A). Among the 251 terminal intervention tasks, 54 have Stage 1 training samples (*initially demonstrated*) and 197 do not (*initially undemonstrated*). The 148 repository-repair intervention tasks split into 101 and 47 in the same way. All intervention tasks are trained in Stage 2. *Strictly held-out* sets of 161 terminal and 92 repository-repair tasks are drawn from the same persistently failed tasks and enter neither Stage 1 nor Stage 2. A *hard subset* of 600 repository-repair tasks (300 SymPy, 300 Matplotlib) comes from repositories excluded from Stage 0, Stage 1, and Stage 2 data and contains only tasks that Stage 0 fails persistently. *Unaided* evaluation removes teacher hints and blocks the known privileged channels, namely future repository history, verifier and expected-output files, and web access, while keeping normal tools and feedback (§3.2). On the intervention pools, the held-out sets, and the hard subset we report unaided pass@8 (Chen et al., 2021), counting a task as solved if any of eight attempts passes. Terminal-Bench 2.1 (The Terminal-Bench Team, 2025) and SWE-bench Verified (Chowdhury et al., 2024) are evaluated with pass@1 under the same protocol.

3 WHAT DO VERIFIED SUCCESSES TEACH CODING AGENTS?

3.1 LEARNING GAINS AND THEIR LIMITS

The three sources differ in who solves the task and with what help. A comes from tasks that Stage 0 already solves on its own, B keeps the student as the executor on intervention tasks but adds a teacher's diagnosis, and K replaces the student with the teacher. Table 1 reports what the original Instruct model learns from each combination under unaided evaluation. Comparing with A-only shows what B and K add, and comparing with Instruct shows the effect of the whole SFT stage.

Finding 1: the value of a source depends on the domain. Relative to A-only, hint-assisted trajectories B add 5.6 points on the terminal pool and teacher demonstrations K add 2.8. On the repository-repair pool the order reverses, with 0.7 points for B and 6.8 for K. Neither the student's guided execution nor the teacher's demonstration is the better source in every domain, and the mixture of all three achieves the highest point estimate on both pools.

Table 1: **Stage 1 recipes.** Unaided pass@8 (%) of a fresh Instruct model fine-tuned on each mixture of experience. Intervention pools contain the persistently failed tasks on which guided experience was collected. Held-out sets come from the same persistently failed tasks and enter neither Stage 1 nor Stage 2. Bold marks the best recipe on each intervention pool.

	Terminal		Repository repair (SWE)	
Recipe	Intervention (251)	Held-out (161)	Intervention (148)	Held-out (92)
Qwen3.5-9B-Instruct	2.0	2.5	4.0	4.3
A only	2.0	1.9	7.4	3.3
A + B	7.6	1.9	8.1	1.1
A + K	4.8	2.5	14.2	4.3
A + B + K (Stage 1)	**9.2**	2.5	**16.2**	4.3

Table 2: **Environment audit.** (a) Privileged-channel access in teacher trajectories, before and after selecting verified successes. The audited channels are future repository history, verifier and expected-output files, and web access. (b) Held-out successes (92 repository-repair tasks, pass@8) of Stage 1 students with these channels open versus closed, from a paired run. Normal tool use and local documentation lookups are not counted as privileged access.

(a) Trajectory audit: privileged-channel access

Trajectory set	Count	Denom.	Rate
Teacher batch, all trajectories	288	500	57.6%
Teacher batch, verified successes	230	301	76.4%
K (successful demonstrations)	172	229	75.1%
K, command names the target commit	137	229	59.8%

(b) Held-out SWE (92), open→closed

Student	Open	Closed	Δ
SFT(A)	12	3	−9
SFT(A+B)	10	1	−9
SFT(A+B+K)	11	4	−7

Finding 2: gains on the intervention pools do not carry over to held-out tasks. Distillation does teach. With the full mixture, unaided pass@8 rises from 2.0% to 9.2% on the terminal pool and from 4.0% to 16.2% on the repository-repair pool. Its held-out performance stays at the Instruct level, with 4 of 161 terminal tasks (2.5%) and 4 of 92 repository-repair tasks (4.3%), and no recipe exceeds Instruct on either held-out set by more than 0.1 points. The pools also contain tasks without Stage 1 samples, so pool performance is not the same as performance on demonstrated tasks. Section 5.3 separates the two.

3.2 Auditing apparent transfer

Finding 3: privileged channels inflate apparent transfer. These channels were open while the experience was produced, and Table 2 reports two audits of their effect. Panel (a) inspects teacher trajectories for accesses to these channels. Within one batch of 500 teacher trajectories, 57.6% access such a channel, and among its 301 verified successes the share rises to 76.4%. Of the 229 successful demonstrations in K, 75.1% access a privileged channel and 59.8% contain a command that names the target commit. Selecting by success therefore concentrates the trajectories one would least want to imitate. Panel (b) evaluates the same Stage 1 students on the 92 held-out repository-repair tasks with the channels open and then closed. Every recipe, including A-only, loses most of its held-out successes when the channels close, so the behavior is not limited to students trained on teacher data.

Success measured with the channels open therefore overstates both the student's independent capability and the teaching quality of a data source. All unaided results in this paper, including Table 1, are obtained with these channels blocked.

Summary. How much successful experience adds depends on the source and the domain. After distillation it raises the intervention pools by 7.2 and 12.2 points but leaves held-out performance at the Instruct level. Evaluating with the channels open adds 7 to 9 apparent successes on the 92 held-

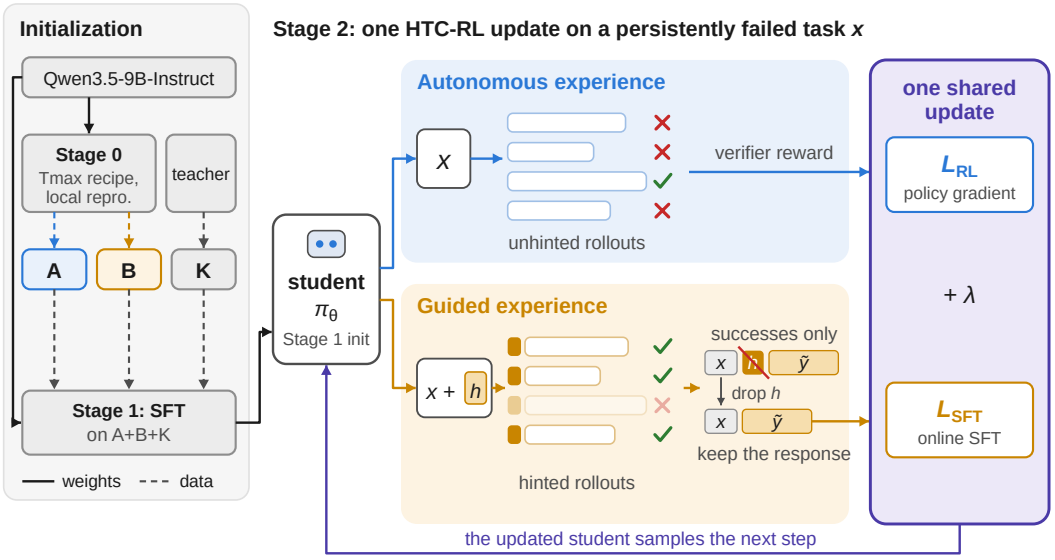

Figure 2: **HTC-RL: combines autonomous reward learning and hint-free online distillation.** Left: Stage 0 supplies data (dashed), and Stage 1 restarts from the original Instruct weights (solid). A, B, and K are autonomous successes, hint-assisted successes, and teacher demonstrations. Right: the student samples rollouts on the same task x with and without a teacher hint h. Unhinted rollouts, scored by the verifier, provide reinforcement-learning signals. Successful hinted trajectories provide online supervision under the original task context after the hint is removed from the prompt. Hinted rollouts receive zero policy-gradient advantage, and the two losses are combined in one student update.

out repository-repair tasks. Having established this gap, we examine whether continued autonomous reward learning can improve the use of guided successes.

4 Method: HTC-RL

HTC-RL continues training from the distilled Stage 1 student and combines supervision on guided successes with reward learning from unhinted rollouts. The two objectives are optimized in one update (Figure 2). We write x for the hint-free task prompt, h for its teacher hint, $x \oplus h$ for the prompt with the hint appended to its last user message, and π_θ for the student policy.

We use three units throughout. A *task* x is an underlying problem together with its environment and hidden verifier. A *prompt instance* is a task paired with a sampling condition, either $(x, \text{unhinted})$ or (x, hinted). A *rollout* is one complete environment execution from a prompt instance. The two prompt instances $(x, \text{unhinted})$ and (x, hinted) of the same task are its *hint twins*, and coupling their two objectives in one update gives HTC-RL its name.

4.1 Distilled initialization

Stage 1 is the mixture recipe of §3.1, fine-tuned from the original Instruct model with every example under its hint-free task context. Its checkpoint is the common starting point of all Stage 2 variants, so differences between variants come from Stage 2 training.

4.2 Two streams, two objectives

Each Stage 2 step samples 16 prompt instances and $G = 16$ rollouts from each, 256 rollouts in total. An ordinary task, one outside the intervention pool, contributes one unhinted prompt instance and receives the standard RL objective in every variant. An intervention task contributes two prompt

instances in every variant, and the variants differ in how these instances are sampled and trained. HTC-RL uses one unhinted instance and one hinted instance.

Autonomous stream (reinforcement learning). From $(x, \text{unhinted})$ the student generates $G = 16$ rollouts $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$, each scored by the verifier with $R_i \in \{0, 1\}$. The advantage is centered within the group and is not divided by the group standard deviation:

$$A_i \;=\; R_i \;-\; \frac{1}{G}\sum_{j=1}^{G} R_j. \tag{1}$$

These rollouts are trained with a clipped, token-level policy-gradient objective (Schulman et al., 2017; Shao et al., 2024; Yu et al., 2025), which we denote $\mathcal{L}_{\mathrm{RL}}(\theta)$. This stream is identical to autonomous RL, and the hint plays no part in it.

Guided stream (hint-free online distillation). From (x, hinted) the same student generates $G = 16$ rollouts $\tilde{y}_1, \ldots, \tilde{y}_G \sim \pi_\theta(\cdot \mid x \oplus h)$. Only verified successes $\mathcal{S}(x) = \{\tilde{y}_i : R(\tilde{y}_i) = 1\}$ are kept. Each kept response is detached from the hinted context and re-attached to the original prompt, and the assistant-token mask is rebuilt so that the loss covers exactly the student's turns under the hint-free context. The objective is the negative log-likelihood of these responses,

$$\mathcal{L}_{\mathrm{SFT}}(\theta) \;=\; -\frac{1}{\sum_{(x,\tilde{y})} |m_{\tilde{y}}|} \sum_{(x,\tilde{y})} \sum_{t \in m_{\tilde{y}}} \log \pi_\theta(\tilde{y}_t \mid x, \tilde{y}_{<t})\,, \tag{2}$$

where $m_{\tilde{y}}$ is the set of valid assistant tokens and the sum runs over the global online batch of 16 retained trajectories. The average is taken over all valid assistant tokens in the batch rather than per trajectory, so a long successful trajectory contributes in proportion to its length. This term has no behavior log-probability and no importance ratio. The hint is used only to obtain the trajectory, and the student is then taught to produce that trajectory from the task alone.

4.3 Joint update and controls

The two objectives are combined in one optimizer step,

$$\mathcal{L}(\theta) \;=\; \mathcal{L}_{\mathrm{RL}}(\theta) \;+\; \lambda\, \mathcal{L}_{\mathrm{SFT}}(\theta), \qquad \lambda = 1, \tag{3}$$

and both terms are evaluated on the original hint-free context. Hinted rollouts receive a policy-gradient advantage of zero. They occupy their share of the sampling quota and of the RL token denominator, and their successful responses reach the parameters only through the supervised path. A hinted group therefore contributes a gradient even when all sixteen rollouts succeed, and a stream's sampling share differs from its effective gradient strength. Algorithm 1 in Appendix B summarizes one update.

Two controls isolate the two streams as complete training choices while keeping the number of prompt instances and rollouts per task unchanged. **Plain RL** replaces the hinted twin by a second unhinted prompt instance, so both instances of an intervention task are trained with $\mathcal{L}_{\mathrm{RL}}$. **Hinted-only** replaces the unhinted twin by a second hinted prompt instance, so verified successes are trained by equation 2 under the hint-free context and no policy-gradient term is applied to these tasks. All three variants start from the identical Stage 1 checkpoint, share the same task pool and the same ordinary-task RL term, and retain 38,400 rollouts each. Because the hinted stream produces additional supervised tokens, HTC-RL processes roughly 10% more training tokens than Plain RL. We match retained rollouts and report this token difference.

5 Experiments and Analysis

5.1 Protocol and baselines

All numbers are unaided. The three variants of §4.3 start from the same measured Stage 1 checkpoint, which solves 23 of the 251 terminal and 24 of the 148 repository-repair intervention tasks under pass@8, and train for 150 steps. The primary comparison is Plain RL, the same-initialization autonomous continuation.

Table 3: **Controlled continuation from a shared Stage 1 initialization.** Entries are solved tasks / unaided pass@8. Top: the four evaluation sets. Bottom: the 251 terminal intervention tasks stratified by whether a task had Stage 1 training samples. All three variants start from the same checkpoint and retain 38,400 rollouts.

	Terminal		Repository repair (SWE)	
Model	Intervention (251)	Held-out (161)	Intervention (148)	Held-out (92)
Stage 1: SFT(A+B+K)	23 / 9.2%	4 / 2.5%	24 / 16.2%	4 / 4.3%
Plain RL	30 / 12.0%	6 / 3.7%	26 / 17.6%	5 / 5.4%
Hinted-only	21 / 8.4%	4 / 2.5%	22 / 14.9%	5 / 5.4%
HTC-RL	**40 / 15.9%**	**9 / 5.6%**	**29 / 19.6%**	**9 / 9.8%**
	Initially demonstrated (54)		Initially undemonstrated, Stage 2-trained (197)	
Stage 1: SFT(A+B+K)	20		3	
Plain RL	24		6	
Hinted-only	17		4	
HTC-RL	**24**		**16**	

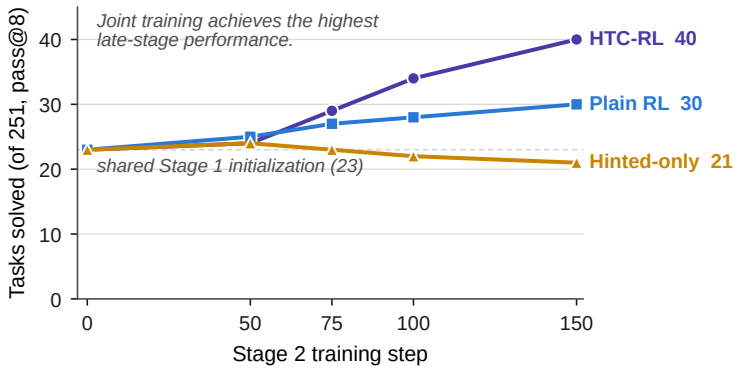

Figure 3: **Learning dynamics from a shared distilled initialization.** Solved terminal intervention tasks (of 251) at each measured checkpoint, under the same unaided pass@8 protocol. All variants begin with 23; at step 150, HTC-RL solves 40, plain RL 30, and hinted-only 21. The dynamics are consistent with guided supervision and autonomous reward learning reinforcing each other.

5.2 Controlled continuation from a shared distilled initialization

Table 3 (top) compares three ways of continuing from the same Stage 1 checkpoint. Hinted-only continues with guided supervision alone on intervention tasks and ends below its Stage 1 starting point on both intervention pools, at 21 versus 23 terminal tasks and 22 versus 24 repository-repair tasks. It never outperforms Plain RL and ties it only on the repository-repair held-out set. Plain RL improves both pools on its own. HTC-RL combines autonomous reward learning with guided supervision and attains the highest point estimate in every column, solving 10, 3, 3, and 4 more tasks than Plain RL on the four sets.

We attribute the Hinted-only result to its training signal. Hinted-only trains the student on trajectories that succeeded because the hint was present, and nothing in its update tells the student which of those steps it cannot yet reproduce alone. Its brief rise and later decline in Figure 3 fit this reading. The unhinted twin in HTC-RL supplies the missing signal, because its rewards measure what the student can do without the hint.

5.3 Gains come from tasks beyond the initial demonstrations

The 197 initially undemonstrated terminal tasks test whether continued training only re-learns what Stage 1 demonstrated or widens the range of tasks the student can solve. Table 3 (bottom) supports

Table 4: **Overall performance and transfer.** (a) Full-benchmark pass@1 across pipeline stages; the three Stage 2 variants share the Stage 1 initialization. (b) Unaided pass@8 on 600 hard repository-repair tasks, 300 each from SymPy and Matplotlib, two repositories excluded from the post-training recipe; tasks are screened by Stage 0 persistent failure, and entries count tasks solved at least once in eight attempts. †Our local reproduction of the Tmax recipe.

(a) Full benchmarks, pass@1

Model	TB 2.1	SWE-V
Qwen3.5-9B-Instruct	20.2	49.8
Stage 0: Tmax recipe†	23.6	50.8
Stage 1: SFT(A+B+K)	24.7	54.6
Stage 2: Plain RL	25.8	55.6
Stage 2: Hinted-only	23.6	54.8
Stage 2: HTC-RL	**28.1**	**57.2**

(b) Hard subset, unaided pass@8

Model	SymPy	Matplotlib	Total	Rate
Stage 1 init.	9	15	24	4.0%
Plain RL	12	19	31	5.2%
Hinted-only	8	14	22	3.7%
HTC-RL	**17**	**26**	**43**	**7.2%**

the second. On the 54 initially demonstrated tasks, HTC-RL and Plain RL both solve 24. On the 197 initially undemonstrated tasks, HTC-RL solves 16, Plain RL solves 6, and Hinted-only solves 4. The entire net gain of ten solved tasks over Plain RL therefore comes from the initially undemonstrated subset, on which HTC-RL also solves four times as many tasks as hinted-only online distillation. Because these tasks are trained in Stage 2, where the guided stream can create new online demonstrations, the result shows capability extending beyond the initial demonstration coverage, and the held-out sets remain the test of transfer. Repository repair shows the same pattern. On its 101 initially demonstrated tasks, HTC-RL and Plain RL again both solve 24, and the net gain of three solved tasks comes entirely from the 47 initially undemonstrated tasks, where HTC-RL solves 5 and Plain RL solves 2 (Appendix E). In both domains the two variants perform identically on the tasks that Stage 1 demonstrated, and the whole advantage of HTC-RL lies beyond that coverage.

5.4 Learning dynamics

Figure 3 tracks solved terminal intervention tasks at fixed checkpoints of each variant. All variants start at 23. The early checkpoints differ by one or two tasks (24 vs. 25 at step 50 and 29 vs. 27 at step 75 for HTC-RL and Plain RL), and the variants separate later: 34 vs. 28 at step 100 and 40 vs. 30 at step 150. Hinted-only rises briefly and then declines to 21. These dynamics are consistent with bootstrapping. Supervision from guided successes changes the shared parameters, more autonomous rollouts then succeed, and tasks whose autonomous groups begin to mix successes and failures start to produce a policy-gradient signal. The guided stream only needs successful trajectories, while the autonomous stream needs reward variance, which the guided stream helps create.

5.5 Overall performance and transfer to hard tasks outside the recipe

Table 4 tests whether the training response reaches beyond the diagnostic pool, with different metrics in its two panels. Panel (a) reports full-benchmark pass@1. HTC-RL reaches 28.1 on Terminal-Bench 2.1 and 57.2 on SWE-bench Verified, which is +2.3 and +1.6 points over the same-initialization Plain RL and +4.5 and +6.4 points over our Stage 0 reproduction. Every stage of the pipeline moves both benchmarks, and HTC-RL is the best Stage 2 choice on both.

Panel (b) reports unaided pass@8 on the 600 hard tasks from SymPy and Matplotlib, two repositories that appear in none of the post-training data and whose tasks were screened by Stage 0 persistent failure. HTC-RL solves 43 tasks (17 SymPy, 26 Matplotlib) against 31 for Plain RL (12 and 19), an increase from 5.2% to 7.2% that is present in both repositories.

6 Related Work

What imitation and distillation teach. Imitating a stronger model can reproduce its style while closing little of the capability gap on tasks outside the imitation data (Gudibande et al., 2023), supervised fine-tuning tends to memorize where outcome-rewarded RL generalizes (Chu et al., 2025),

and pass@k analyses question how far RL widens what a model can solve (Yue et al., 2025). Our findings add the agentic-coding case, with verified successes from three sources and an explicit information boundary that separates pool gains from gains on audited held-out tasks.

Reinforcement learning for coding agents. Reinforcement learning has been applied to repository repair and terminal use (Wei et al., 2025; Luo et al., 2025; Ivison et al., 2026), alongside executable training environments and data pipelines (Pan et al., 2025; Jain et al., 2025; Yang et al., 2025), building on agent scaffolds and benchmarks (Yang et al., 2024; Wang et al., 2025; Jimenez et al., 2024; The Terminal-Bench Team, 2025). With binary rewards and group-relative advantages (Shao et al., 2024; DeepSeek-AI et al., 2025; Yu et al., 2025), a task provides no policy-gradient signal when all its rollouts fail, and HTC-RL adds a stream that supplies successes on such tasks.

Learning with guidance and privileged information. Training with information unavailable at test time has a long history (Vapnik & Vashist, 2009; Chen et al., 2020; Ross et al., 2011), and our verifier-informed hints are privileged information in this sense. For language models, off-policy guidance mixes teacher trajectories into the policy-gradient group (Yan et al., 2025), question augmentation adds partial solutions to hard prompts (Li et al., 2025), and hindsight hint distillation self-distills hint-scaffolded successes of SWE agents (Wang et al., 2026). Hindsight self-distillation for long-horizon agents targets the credit-assignment side of the same problem, providing step-level supervision (Yeo et al., 2026; Li et al., 2026; Jin et al., 2026). HTC-RL differs in the objective. Its hinted trajectories receive zero policy-gradient advantage and contribute through hint-free supervision when successful, while the hint-free rollouts of the same task keep their reward-driven objective.

Success-filtered self-training and online distillation. Filtering a model's own successes and fine-tuning on them (Zelikman et al., 2022; Yuan et al., 2023; Gulcehre et al., 2023; Singh et al., 2024) and distilling from a teacher on student-generated contexts (Agarwal et al., 2024; Gu et al., 2024) share our supervised path. Closest in spirit are methods that combine supervised and reinforcement objectives on hard questions (Ma et al., 2025; Fu et al., 2025). In HTC-RL the supervised signal comes from the student's own hinted execution, verified by the environment, and is trained under the original task context, so the supervised path trains the same unaided behavior that is evaluated.

Evaluation integrity. Reward hacking and specification gaming are well documented (Skalse et al., 2022; Pan et al., 2022; Denison et al., 2024; Baker et al., 2025). Our audit adds a measurement-side observation for agentic coding. Success filtering concentrates privileged-channel use in teacher trajectories, and students trained on autonomous data alone also lose held-out successes once the channels close. We therefore treat the information boundary as part of the evaluation protocol.

7 Discussion and Conclusion

A verified trajectory is evidence that a solution was reached under the conditions that produced it; it is not, by itself, evidence that the student has acquired the ability to reach that solution unaided. This distinction matters whenever training introduces information or assistance that will disappear at deployment. Guidance can expose useful solutions that the student would not discover on its own, but success during data collection should not be conflated with the capability that remains after that guidance is removed.

HTC-RL turns this distinction into a training mechanism. For each hard task, hinted execution supplies successful behavior, while unhinted rollouts preserve the verifier-based RL signal. Successful hinted trajectories are then trained under the original hint-free context. This coupling lets guidance bootstrap tasks with little or no autonomous reward signal, while subsequent unhinted successes bring those tasks back into ordinary reward learning. In this sense, HTC-RL uses guidance to expand where RL can learn, rather than replacing RL with imitation.

Our study focuses on a 9B student, two coding-agent domains, and Stage 0-screened hard tasks. An important next question is how guided bootstrapping should scale as models become stronger and the boundary between solvable and persistently failed tasks shifts. The value of guidance lies not in the successes it produces, but in the capability that remains after the guidance is removed.

Reproducibility statement

Section 2 defines the task families, the intervention-pool screening rule, the three experience sources and their mixing ratio, the task-exposure sets, and the unaided evaluation protocol. Section 4 and Algorithm 1 specify the two objectives, the advantage normalization, the assistant-token masking, the global token-level averaging, and the loss weight. Appendix A lists the task-exposure sets, Appendix B the training and evaluation configuration, Appendix C how hints are generated, screened, inserted, and removed, and Appendix D the audit procedure. Task identifiers for all diagnostic sets and the per-checkpoint evaluation results behind Figure 3 will be released with the code.

Ethics statement

This work trains and evaluates coding agents in sandboxed containers on public open-source repositories and synthetic terminal tasks. The environment audit in §3.2 documents that agents can exploit privileged information channels when these are left open; we report this to improve evaluation practice and block the channels in all reported evaluations. Teacher models were used to generate hints and demonstrations for training; no human subjects were involved. We do not release trajectories that access privileged channels.

AI use statement

In this work, we used generative AI tools to draft and polish the manuscript; retrieve and identify relevant literature; discuss research ideas and provide feedback on experimental design and result interpretation; assist with software development, experiment diagnostics, and figure-generation code; and generate synthetic training data, including teacher diagnostic hints and demonstrations. We did not use generative AI tools to prove mathematical claims. The authors reviewed AI-assisted code, analyses, citations, and prose for correctness and attribution, and validated the empirical claims against the underlying experiment records. We take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, 2020.

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.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI blog, `https://openai.com/index/introducing-swe-bench-verified/`, 2024.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, et al. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *arXiv preprint arXiv:2406.10162*, 2024.

Yuqian Fu, Tinghong Chen, Jiajun Chai, Xihuai Wang, Songjun Tu, Guojun Yin, Wei Lin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. SRFT: A single-stage method with supervised and reinforcement fine-tuning for reasoning. *arXiv preprint arXiv:2506.19767*, 2025.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024.

Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. The false promise of imitating proprietary LLMs. *arXiv preprint arXiv:2305.15717*, 2023.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. Tmax: A simple recipe for terminal agents. *arXiv preprint arXiv:2606.23321*, 2026.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Xiaolong Jin, Dingmin Wang, Vijay Lingam, and Varun Kumar. AHEAD: Adaptive hindsight with environment-augmented distillation for agentic RL. *arXiv preprint arXiv:2608.24114*, 2026.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. QuestA: Expanding reasoning capacity in LLMs via question augmentation. *arXiv preprint arXiv:2507.13266*, 2025.

Xiaozhe Li, Tianyi Lyu, Yang Li, Yichuan Ma, Peiji Li, Linyang Li, Qipeng Guo, Dahua Lin, and Kai Chen. What and when to distill: Selective hindsight distillation for multi-turn agents. *arXiv preprint arXiv:2605.19447*, 2026.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, et al. DeepSWE: Training a state-of-the-art coding agent from scratch by scaling RL. Agentica / Together AI technical blog, 2025.

Lu Ma, Hao Liang, Meiyi Qiang, Lexiang Tang, Xiaochen Ma, Zhen Hao Wong, Junbo Niu, Chengyu Shen, Runming He, Bin Cui, and Wentao Zhang. Learning what reinforcement learning can't: Interleaved online fine-tuning for hardest questions. *arXiv preprint arXiv:2506.07527*, 2025.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*, 2022.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *International Conference on Machine Learning*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, February 2026a.

Qwen Team. Alibaba unveils Qwen3.8-Max: Its largest and most capable flagship model to date. Alibaba Cloud press release, https://www.alibabacloud.com/en/press-room/alibaba-unveils-qwen3-8-max, 2026b.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

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

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

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

The Terminal-Bench Team. Terminal-Bench: Benchmarking agents on hard, realistic tasks in terminal environments. https://www.tbench.ai, 2025.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009.

Shengjie Wang, Guanghe Li, Zonghan Yang, and Yang Gao. Hindsight hint distillation: Scaffolded reasoning for SWE agents from CoT-free answers. *arXiv preprint arXiv:2605.11556*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

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. *arXiv preprint arXiv:2502.18449*, 2025.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

Woongyeong Yeo, Yumin Choi, Taekyung Ki, and Sung Ju Hwang. HINT-SD: Targeted hindsight self-distillation for long-horizon agents. *arXiv preprint arXiv:2605.17873*, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

APPENDIX

A TASK EXPOSURE SETS

Table 5 lists the diagnostic evaluation sets together with their exposure to each training stage. The two dimensions that matter for interpretation are whether a set contributed samples to Stage 1 and whether it was practiced in Stage 2. The held-out guarantee covers Stages 1 and 2; we make no claim about Stage 0 exposure of the held-out sets. Stage 0 transfers data, not weights, to later stages, and the intervention pools and the hard subset are defined by Stage 0 persistent-failure screening. Terminal-Bench 2.1 and SWE-bench Verified are public benchmarks and are not part of this table.

Table 5: Task exposure of the diagnostic evaluation sets. "Stage 1 samples" means that successful trajectories of the task entered the Stage 1 SFT corpus. "Stage 2 practice" means that the task was in the Stage 2 training pool.

Set	Size	Stage 1 samples	Stage 2 practice	Reading
Terminal, initially demonstrated	54	yes	yes	continued performance on demonstrated tasks
Terminal, initially undemonstrated	197	no	yes	learning beyond initial demonstrations
Terminal, strictly held-out	161	no	no	transfer relative to Stages 1–2
SWE, intervention pool	148	partly (101)	yes	intervention-pool learning
SWE, initially undemonstrated	47	no	yes	learning beyond initial demonstrations
SWE, strictly held-out	92	no	no	transfer relative to Stages 1–2
Hard subset (SymPy 300, Matplotlib 300)	600	no	no	repositories excluded from Stages 0–2; Stage 0 screened

The intervention pools are defined by Stage 0 persistent failure: a task is included when none of the three latest Stage 0 checkpoints produces a verified success on it. The hard subset applies the same screening rule to tasks from SymPy and Matplotlib; these two repositories are excluded from the Stage 0, Stage 1, and Stage 2 training data. The exclusion concerns our post-training recipe, not the pretraining corpus of the base model.

B TRAINING AND EVALUATION CONFIGURATION

Table 6 collects the configuration that determines the comparisons in the paper. Values fixed by the recipe are given explicitly; optimizer, context-length, and sampling settings are identical across the three Stage 2 variants and are listed in the last rows.

C HINT GENERATION, INSERTION, AND REMOVAL

Which tasks receive hints. Hints are written for persistently failed tasks, and only intervention-pool tasks use them in training; held-out tasks never receive a hint in training or evaluation. A task whose hint fails the leakage screen below receives no hint.

Algorithm 1 One HTC-RL update on an intervention task x through its hint twins ($G = 16$ rollouts each)

1: Sample $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$ ▷ autonomous stream, hint-free
2: $R_i \leftarrow \mathrm{Verify}(x, y_i)$; $A_i \leftarrow R_i - \frac{1}{G}\sum_j R_j$ ▷ no std. normalization
3: Sample $\tilde{y}_1, \ldots, \tilde{y}_G \sim \pi_\theta(\cdot \mid x \oplus h)$ ▷ guided stream
4: $\mathcal{S} \leftarrow \{\tilde{y}_i : \mathrm{Verify}(x, \tilde{y}_i) = 1\}$; set PG advantage of every $\tilde{y}_i$ to 0
5: **for** $\tilde{y} \in \mathcal{S}$ **do** re-attach $\tilde{y}$ to x; rebuild the assistant-token mask $m_{\tilde{y}}$
6: **end for**
7: $\mathcal{L}_{\mathrm{RL}} \leftarrow$ clipped policy gradient on $\{(x, y_i, A_i)\}$; $\mathcal{L}_{\mathrm{SFT}} \leftarrow$ equation 2 on $\{(x, \tilde{y}) : \tilde{y} \in \mathcal{S}\}$
8: $\theta \leftarrow \theta - \eta \nabla_\theta \big(\mathcal{L}_{\mathrm{RL}} + \lambda \mathcal{L}_{\mathrm{SFT}}\big)$, $\lambda = 1$ ▷ one shared update

Table 6: Configuration shared by the three Stage 2 variants and the settings in which they differ.

Item	Setting
Student / initialization	Qwen3.5-9B-Instruct → Stage 1 SFT(A+B+K) checkpoint, identical for all variants
Stage 1 corpus	A : B : K ≈ 8 : 1 : 1, every example under its hint-free task context
Units	task x; prompt instance $(x, \text{unhinted})$ or (x, hinted); rollout = one complete environment execution from a prompt instance
Per step	16 prompt instances × $G = 16$ rollouts = 256 rollouts
Prompt instances per task	ordinary task: one, unhinted; intervention task: two in every variant (HTC-RL: its hint twins; Plain RL: two unhinted; Hinted-only: two hinted)
Stage 2 length	150 steps; checkpoints every 25 steps, evaluated at steps 0, 50, 75, 100, 150
Retained rollouts per variant	$150 \times 256 = 38{,}400$
Advantage	$A_i = R_i - \frac{1}{G}\sum_j R_j$; no division by the group standard deviation
Policy-gradient term	clipped token-level objective on unhinted rollouts; hinted rows have advantage 0
Online SFT term	NLL over valid assistant tokens, averaged over all such tokens in a global batch of 16 retained trajectories; $\lambda = 1$
Behavior ratio in SFT term	none (no behavior log-probability, no importance weight)
Ordinary tasks	standard RL objective, identical in all variants
Training tokens	HTC-RL ≈ 10% more than Plain RL (additional supervised tokens)
Evaluation, diagnostic sets	unaided pass@8: no hints, designated privileged channels blocked, normal tools kept
Evaluation, benchmarks	Terminal-Bench 2.1 and SWE-bench Verified, pass@1, same unaided protocol
Hardware	4 nodes × 8 A100 GPUs per variant; 24 vLLM inference engines and 8 learners; asynchronous rollout depth 2
Environment limits	300 s per command, 1200 s per test run, 75 min per sandbox; groups lost to infrastructure failures are dropped
Optimizer / learning rate	fused AdamW; learning rate 1×10^{-6} with a constant schedule; no warmup; gradient clipping at 1.0; weight decay 0; one optimizer update per step (one epoch, one mini-batch)
Context length / sampling	2,048 prompt tokens; 16,384 tokens per turn; 65,536 tokens per trajectory; packed length 67,584; at most 64 tool calls per rollout; temperature 1.0; top-p 1.0

Teacher and inputs. Hints are written by Kimi K3 (Kimi Team, 2026) in up to three rounds; a task still without a valid hint falls back to Qwen3.8-Max (Qwen Team, 2026b). Sampling uses temperature 0.3, top-p 0.95, and at most 2,048 output tokens. The teacher sees three inputs: (i) the student-visible task instruction, truncated to 2,500 characters; (ii) one failed Stage 0 trajectory on the task, preferring failures that ended naturally, compressed to its last eight actions with reasoning removed, each action truncated to 800 characters, error-focused observation excerpts of up to 700 characters, and the final message, within 9,000 characters in total; and (iii) the task's private verifier script, truncated to 6,000 characters. The teacher does not see a reference solution or gold patch.

Instructions. The teacher is asked for one short hint that helps the student succeed on a fresh attempt from scratch. It may state a concept-level root cause, point to evidence the student overlooked in its own trajectory, suggest what to investigate next, and describe a self-check before submission. It must not name test paths or test functions, must not state values that appear only in the verifier, and must not give ready-made code or exact commands; a requirement known only from the verifier may be referred to by its category. The answer is a JSON object whose four rendered fields carry word budgets: likely root cause (40 words), overlooked evidence (30), suggested direction (40), and self-check (25). Every hinted prompt is checked to fit within 2,048 prompt tokens.

Leakage screen. A lexical scan extracts entities from the verifier script: file paths, quoted strings of 4 to 80 characters, test-function names, and numbers with four or more digits; the encoded expected outputs of repository-repair verifiers are decoded first. An entity is secret when it appears neither in the task instruction nor in the compressed trajectory. A hint that contains a hard secret entity, one that includes a path separator, underscore, colon, or space, starts with `test_`, is a number of four or more digits, has twelve or more characters, or is a dotted identifier of six or more characters, is quarantined and not regenerated. Matches on short plain words are reported but not blocked. The screen blocks literal disclosure of verifier content; conveying a verifier-only requirement by its category is permitted by design, consistent with the privileged role of the hint.

Insertion. The hint is rendered under the header "Hint from a prior debugging session" as a note from a senior engineer who reviewed a previous failed attempt, with the four fields as bullets, and appended to the end of the task prompt. The environment and its files are unchanged.

Removal for B. An exact-match rule removes the rendered hint block from a hint-assisted Stage 0 trajectory. A trajectory enters B only if exactly one hint block was removed, its verifier reward is at least 0.5, the episode ended naturally, no tool call failed, and no test file was modified; responses that echo the hint header or its wording are rejected, and at most four trajectories are kept per task.

Removal in the guided stream. To convert a verified hinted success $\tilde{y}$ into an online training example, we (i) replace the hinted prompt by the original hint-free prompt x used by the autonomous stream; (ii) keep the assistant turns and tool-result turns of $\tilde{y}$ unchanged; and (iii) re-tokenize the resulting conversation and rebuild the assistant-token mask $m_{\tilde{y}}$ so that exactly the student's tokens receive loss, while tool results and user turns do not. The hint is therefore present only when the trajectory is sampled and absent from every training context.

The online SFT term averages over all valid assistant tokens in the global batch of 16 retained trajectories (equation 2), with $\lambda = 1$. This normalization weights a long successful trajectory in proportion to its length rather than treating every trajectory equally; it fixes how the supervised term is averaged and does not by itself equalize the magnitude of the two gradient terms.

D AUDIT PROCEDURE

A trajectory is counted as accessing a privileged channel when any of its commands reads information that is designated as unavailable to an honest solver: future commits or reflogs in the repository history, files that contain the expected output of hidden tests, the verifier's own scripts, or external sources that reveal the target fix or another task-specific answer. Ordinary tool use, general documentation lookups, and inspection of the working tree at the task's starting commit are not counted. Panel (a) of Table 2 applies this rule to one batch of 500 teacher trajectories and to its 301 verified successes, and to the 229 successful demonstrations in K; "command names the target commit" counts trajectories that contain at least one command referencing the commit hash of the fix, not the fraction of all commands. Panel (b) evaluates each Stage 1 student on the 92 held-out repository-repair tasks with the channels open and with the channels closed, in a paired run with the same attempt budget. The audit reports observed accesses; it does not assert that every accessed trajectory depended on the access to succeed.

E SUPPLEMENTARY RESULTS

Repository-repair pool by initial demonstration coverage. Of the 148 repository-repair intervention tasks, 101 had Stage 1 samples and 47 did not. On the 47, HTC-RL solves 5 and Plain RL solves 2 under unaided pass@8. With pool totals of 29 and 26 (Table 3), both variants solve 24 of the 101 demonstrated tasks, matching the terminal pattern of 24 and 24.

Per-repository counts on the hard subset. Relative to Plain RL, HTC-RL solves 5 more SymPy tasks (17 vs. 12) and 7 more Matplotlib tasks (26 vs. 19), for 12 more tasks in total (43 vs. 31).

Measured checkpoints behind Figure 3. Solved terminal intervention tasks (of 251) at steps 0 / 50 / 75 / 100 / 150: Plain RL 23 / 25 / 27 / 28 / 30; Hinted-only 23 / 24 / 23 / 22 / 21; HTC-RL 23 / 24 / 29 / 34 / 40. The end points coincide with Table 3.