HITCHRL: NON-BLOCKING TEACHER SUPERVISION ON IDLE GPUS IN ASYNCHRONOUS RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) with verifiable rewards improves the reasoning of large language models, and group-based methods such as GRPO learn from reward differences among responses to the same prompt. A group whose responses all fail therefore yields no signal, and such groups make up 36.6% of those used for updates in a teacher-free asynchronous RL run on a 1.7B model. A stronger, frozen teacher can still guide them, since it evaluates the student's own responses token by token and gives a direction even when every response fails. Large-scale RL, however, increasingly runs asynchronously on separate GPU pools for generation and optimization, and a teacher does not fit easily into this structure. Given dedicated GPUs, it takes them from the student, and run in step with generation or updates, it makes them wait. We introduce HitchRL, which runs the teacher with no GPUs of its own while no student update waits for an unfinished teacher target. Since a frozen teacher gives a response the same targets whenever it runs, HitchRL defers teacher inference and, with the teacher's weights offloaded to CPU memory, lets it hitchhike on the idle time of the existing GPUs, returning them when generation or optimization needs them, while updates proceed on ready data. With the same eight GPUs, HitchRL raises the eight-benchmark mean accuracy of a 1.7B student from 14.20% with the teacher-free AReaL to 17.04%, the best of all methods compared. This gain over AReaL extends to a larger student, a student from another model family, and code generation with execution feedback. During training, it reaches the same OlympiadBench accuracy as AReaL and as on-policy distillation followed by AReaL in 63% and 76% less wall-clock time, while keeping 0.95 of AReaL's generation throughput with a 30B-A3B teacher and 0.90 even with a 235B teacher.

1 INTRODUCTION

Reinforcement learning (RL) is widely used to improve the reasoning capabilities of large language models (LLMs). A common formulation is group-based RL, which turns the relative rewards of responses sampled for the same prompt into a learning signal, and Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is a representative instance. Because the signal is relative, a group whose responses all receive the same reward has zero advantages and supplies no reward-based policy-gradient signal. With verifiable, binary rewards such groups are common, and the missing signal matters when every response is incorrect, as in 36.6% of the groups used for updates in our teacher-free asynchronous RL run on a 1.7B model. Supplying a learning direction for them requires supervision beyond their own rewards. Successful demonstrations can provide it but must be collected or generated separately, while auxiliary objectives refine the signal that outcome rewards already provide, using the model's own outputs or the rewards. In on-policy distillation, a teacher instead evaluates the student's own response tokens, and teacher–student log-probability differences give token-level supervision even without a reward contrast. The price is teacher computation.

We study how to pay this price within the GPUs of an asynchronous RL pipeline. Response lengths vary widely, so in synchronous RL each update waits for the longest response in its batch while GPUs that finished shorter ones sit idle. Asynchronous RL, used for recent large models such as GLM-5 (GLM-5-Team, 2026), Olmo 3.1 (Team Olmo, 2025), and MiniMax M2.5 (MiniMax, 2026), reduces this wait by running generation and optimization concurrently on separate rollout and training GPUs and periodically synchronizing the updated weights. A teacher does not fit easily into

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.

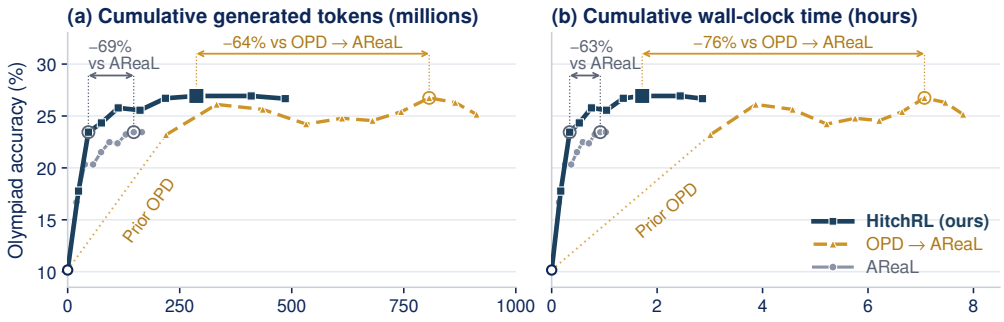

Figure 1: Olympiad accuracy of the Qwen3-1.7B student recorded during training against (a) generated response tokens and (b) wall-clock time, with a 30B-A3B teacher for HitchRL and for the on-policy distillation (OPD) stage of OPD → AReaL. The dotted line is the prior OPD stage, and callouts give HitchRL's reduction in the cost of first reaching each baseline's highest plotted score.

this structure. For example, running the teacher asynchronously as a third concurrent stage requires GPUs of its own, and since every GPU is already assigned to the rollout or training pool, they must be taken from one of these pools, as in existing asynchronous systems with a teacher. Serializing the teacher with generation and the update instead brings the waiting back, since a response can be distilled only after the teacher has computed log-probabilities for its tokens, and generation or the whole update then waits for a result that only the distilled responses need. Avoiding both leaves the teacher only the idle time of the existing GPUs, where its targets can finish long after their responses were generated. This idle time appears at moments that are hard to predict because generation times vary, and a large teacher must fit into GPU memory already filled by the student's weights, KV cache, or optimizer states. How can the teacher use this idle time on either pool while no update waits for an unfinished teacher target?

We introduce HITCHRL, which separates preparing teacher supervision from consuming it. HitchRL learns from rewards in every group except those whose responses are entirely correct or entirely incorrect, and it distills one response from each entirely incorrect group. Because the teacher is frozen, the log-probability it gives a token of a stored response does not depend on when the teacher computes it. Building on this, HitchRL defers teacher inference on each selected response instead of running it when the response is generated. HitchRL keeps responses awaiting the teacher in a queue reserved for teacher work and places reward-based groups and responses with completed targets in a separate queue that student updates consume. The teacher and the student updates consume their queues independently, so as long as the second queue holds ready data, no update waits for an unfinished target. The teacher runs on either pool only while that pool is idle, not doing its own generation or optimization. To keep the teacher from slowing training, HitchRL gives that work priority, and the teacher returns the pool at the end of its current batch as soon as the work needs it, so the work waits for at most one bounded batch. With its weights offloaded to CPU memory and gathered onto the GPUs one module at a time for forward-only evaluation, even a large teacher fits into the GPU memory that the student leaves free. Because the teacher draws from its own queue, it processes queued responses in batches that can span failed groups arriving over several student updates, and each gathering of its weights serves a whole batch. Each batch is bounded by a response limit and a per-GPU token budget, so its capacity follows the GPU count of whichever pool runs it, although the two pools differ in size, and its memory and duration stay bounded. Because reverse-KL distillation on the student's own responses takes the same policy-gradient form as reward-based learning, the completed response then joins reward-based data in one student update.

We make three contributions that build on one another.

• **Teacher targets that can wait, and updates that do not wait for them.** Because a frozen teacher's log-probabilities do not depend on when they are computed, HitchRL defers teacher inference, and we prove that deferring it leaves the loss and gradient of a fixed update unchanged. Two independently consumed queues keep ready data from waiting on unfinished targets, and completed targets join reward-based groups in the same update, with no separate training loop or extra data.

• **A large teacher that hitchhikes on idle GPUs.** To our knowledge, HitchRL is the first asynchronous RL method to run its teacher in the idle time of the existing GPUs, with no update waiting

for unfinished teacher targets. With a 1.7B student, it keeps 0.95 of the generation throughput of AReaL (Fu et al., 2025) with a 30B-A3B teacher and 0.90 with a 235B teacher, and it takes 7.0% less time per generated token than giving the teacher two of the eight GPUs.

• **Higher accuracy in less wall-clock time.** With the same eight GPUs and a 30B-A3B teacher, HitchRL raises the eight-benchmark mean accuracy of a 1.7B student from 14.20% with AReaL to 17.04%, the highest of all methods compared. It reaches the same OlympiadBench accuracy as AReaL and as on-policy distillation followed by AReaL in 63% and 76% less time (Figure 1).

2 Related Work

Guidance from prepared solutions. Correct solutions prepared in advance give a direction when the student's own attempts yield no reward contrast. LUFFY (Yan et al., 2025) adds pre-generated correct traces to each group of on-policy rollouts and optimizes both with mixed-policy GRPO, so the traces enter learning as training samples. BREAD (Zhang et al., 2025) instead uses them as starting points for exploration, resampling a group with no correct response from prefixes of an expert solution until its outcomes are mixed. NuRL (Chen et al., 2026a) appends a hint written offline from the gold answer to each problem whose rollouts all fail and samples the group again. These methods rely on solutions or hints prepared beforehand. Environment frameworks such as AgentGym (Xi et al., 2025), NeMo Gym (NVIDIA, 2025), OpenEnv (OpenEnv Contributors, 2026), and Verifiers (Brown, 2025) package tasks and verifiers so that new domains can be trained with RL, but successful trajectories remain a separate resource. For example, R2E-Gym (Jain et al., 2025b) generates executable software-engineering environments from repository commits and trains its agents on successful trajectories collected with a proprietary model.

Supervision beyond outcome rewards. When a group's rewards provide no contrast and no demonstrations are available, additional supervision can be constructed from the existing reward signal or obtained from a teacher. PRIME (Cui et al., 2026) turns outcome labels into token-level rewards through an implicit process reward model, and LaSeR (Yang et al., 2026) trains a last-token self-reward to agree with the verifier. RL-ZVP (Le et al., 2026) rewards groups whose responses all succeed and penalizes groups whose responses all fail, scaling each token's advantage by its entropy. All three remain tied to outcome labels, and PRIME trains only on prompts of intermediate accuracy. Teacher distillation instead evaluates the student's own responses, so it can point toward better tokens even when every response fails. GKD (Agarwal et al., 2024) combines this distillation with reward optimization, while MiniLLM (Gu et al., 2024) develops policy-gradient optimization for reverse-KL distillation. Later methods use task feedback to decide how the teacher's signal applies. OPDVR (Lin et al., 2026) restricts the signal's sign by each response's correctness, SDPO (Hübotter et al., 2026) conditions the policy on feedback to act as its own teacher, and RSTG (Han et al., 2026) distills only entirely incorrect groups and adds supervised fine-tuning on teacher trajectories generated offline. However, these methods run the teacher on the student's responses inside each training step, in sequence with generation and the update, so its computation adds directly to step time.

Asynchronous execution and pipeline overlap. Execution systems reduce the idle time of RL by decoupling or overlapping its stages. AReaL (Fu et al., 2025) runs generation and optimization on separate GPUs, bounds the staleness of each training batch, and adopts a decoupled proximal policy optimization (PPO) objective for trajectories from older policies, and PipelineRL (Piché et al., 2026) updates the generation weights in flight without stopping sequence generation. RLHFuse (Zhong et al., 2025) instead overlaps the stages of an RLHF iteration while preserving its synchronous semantics, moving long-tail samples to a few instances and running reference, reward, and critic inference on the freed GPUs, and OPPO (Yan et al., 2026) streams generated chunks to reward-model prefilling and defers unfinished sequences to the next step. These systems speed up existing RL objectives and add no learning signal when a group's rewards give none. AsyncOPD (Kang et al., 2026) and f-OPD (Chen et al., 2026b) run on-policy distillation asynchronously without rewards, with the teacher as a concurrent stage on GPUs of its own. AReaL's distillation option (AReaL Contributors, 2026) and OpenClaw-RL (Wang et al., 2026) add a teacher term to the reward-based loss, but also run the teacher on GPUs of its own and wait for its log-probabilities before each update.

HitchRL's teacher guidance within asynchronous RL. The teacher-based methods above place the teacher in one of two ways. Synchronous methods run it inside each training step and add its computation to step time, while asynchronous systems with a teacher give it GPUs of its own.

HitchRL does neither, running the teacher only in the idle time of the existing GPUs while its updates proceed on ready data. This relies on a frozen teacher, whose targets remain valid when computed later, whereas the teacher of SDPO is the feedback-conditioned policy itself and changes with every update. Like RSTG, HitchRL distills only entirely incorrect groups, and its signal can raise the tokens the teacher prefers even inside a failed response. OPDVR instead keeps the teacher's signal non-positive on incorrect responses, while RSTG and solution-based guidance supply a positive direction through correct trajectories or prepared solutions, which HitchRL does not need.

3 METHOD

3.1 TEACHER TARGETS THAT CAN WAIT

Mixed groups learn from rewards, and failed groups from the teacher. Given a prompt x, group-based RL collects G responses with binary correctness rewards R_j, and $c_g = \sum_{j=1}^{G} R_j$ counts the correct ones. Groups with $0 < c_g < G$ keep all responses for reward-based learning, groups with $c_g = G$ are excluded because their rewards provide no within-group contrast, and one response of each group with $c_g = 0$ is distilled. Distilling one response follows on-policy distillation, which trains on one student-generated response per input (Agarwal et al., 2024), and lets a teacher limited to idle time process more failed groups than distilling every response would. With student policy π_θ and frozen teacher p_T, the context at position t of a response y is $s_t = (x, y_{<t})$ and the generated token is $a_t = y_t$. When a response also contains environment feedback, such as the execution results returned after each attempt in our code runs, s_t includes the feedback that precedes a_t, and only the tokens the student generates receive a learning signal. The distilled responses are trained with reverse-KL distillation, optimized with policy gradients (Gu et al., 2024). For each token of a distilled response, the sampled-token discrepancy $\delta_t(\theta) = \log \pi_\theta(a_t \mid s_t) - \log p_T(a_t \mid s_t)$ takes its teacher term from teacher inference and its student term from the current student. Appendix C gives the distilled response, its token span, and the model-specific input formatting.

Teacher targets can be computed at any time. The teacher is frozen, so its log-probability for a stored context–token pair is unchanged by intervening student updates. The teacher target can therefore be computed later, whenever resources allow, while the student log-probability in δ_t is evaluated when the response is consumed. As a result, deferring teacher inference leaves the loss and gradient of a fixed update unchanged. The response, however, was generated by a behavior policy μ, and the student may have changed since. Correcting the reverse-KL gradient for sampling from μ weights each sampled token by the ratio π_θ/μ, and from this corrected gradient we build a distillation surrogate on the distilled responses. The discrepancy, evaluated and detached at the update state θ_0 and bounded by C, gives the distillation advantage $A_t^{\mathrm{KD}} = \mathrm{sg}[\mathrm{clip}(-\delta_t(\theta_0), -C, C)]$, where sg holds the signal fixed during backpropagation. The advantage is positive when the teacher assigns the sampled token a higher probability than the current student does and negative when it assigns a lower probability, so even a failed response has tokens to raise as well as tokens to lower. With this advantage, teacher supervision takes the same policy-gradient form as reward-based learning, with a log-probability difference replacing the reward advantage. Appendix A.1 derives this corrected gradient, Eq. (3), and the surrogate, and the rest of Appendix A shows that deferral leaves a fixed update unchanged and bounds how visited contexts change under an older policy.

Both signals share one loss, each corrected against its own reference. Both branches learn from responses generated by older policies, but they correct for this in different ways. Let $\mathcal{P}$ and $\mathcal{D}$ be the valid response-token indices for reward-based learning and distillation in one optimizer update, each index i identifying a context–token pair (s_i, a_i). The reward-based branch uses GRPO advantages and decoupled PPO (Hilton et al., 2022), as adopted by AReaL (Fu et al., 2025). Its PPO reference is a fixed proximal policy π_{prox}, and its samples from the recorded generation-time policy μ_i are reweighted by $\bar{w}_i = \min\{\pi_{\mathrm{prox}}(a_i \mid s_i)/\mu_i(a_i \mid s_i),\, \tau\}$ with cap $\tau > 0$, so the off-policy correction is factored through the proximal reference. The distillation branch instead uses the recorded behavior probability directly as its reference, as the ratio π_θ/μ of its corrected gradient requires, and therefore does not apply $\bar{w}_i$ again, which would correct its tokens twice.

Their learning signals U_i and reference probabilities q_i are

$$(U_i, q_i) = \begin{cases} (\bar{w}_i A_i^{\mathrm{GRPO}},\ \pi_{\mathrm{prox}}(a_i \mid s_i)), & i \in \mathcal{P}, \\ (\lambda A_i^{\mathrm{KD}},\ \mu_i(a_i \mid s_i)), & i \in \mathcal{D}. \end{cases} \tag{1}$$

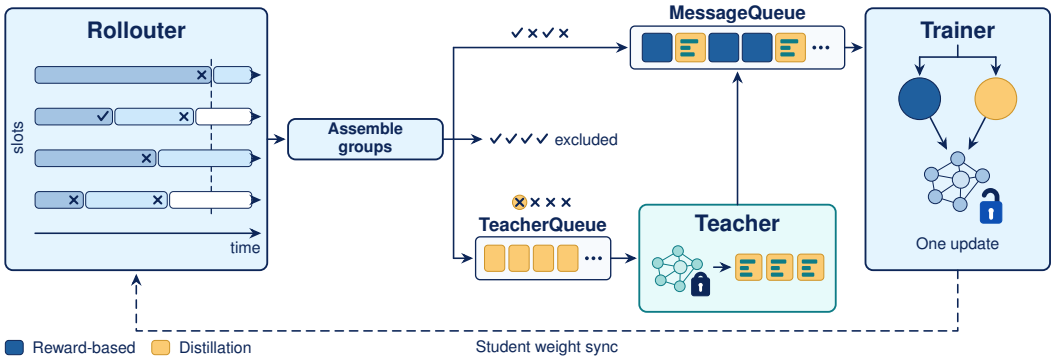

Figure 2: HitchRL's dataflow among three logical roles. Each Rollouter row is a rollout slot, and the darker responses form one group.

Here A_i^{GRPO} is the normalized within-group reward advantage shared by a response's tokens, and $\lambda \geq 0$ weights distillation, all fixed during the backward pass. With $r_i(\theta) = \pi_\theta(a_i \mid s_i)/q_i$ and $N = |\mathcal{P}| + |\mathcal{D}|$, the joint loss is

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i \in \mathcal{P} \cup \mathcal{D}} \min\{r_i(\theta)U_i,\ \mathrm{clip}(r_i(\theta), 1-\epsilon_-, 1+\epsilon_+)U_i\}\,. \tag{2}$$

PPO clipping, applied to each branch against its own reference, removes the incentive to move a token's ratio farther beyond the bound in the direction favored by its advantage. Under the shared denominator N, absorbing $\bar{w}_i$ and λ into the corresponding learning signals exactly preserves each branch's clipped surrogate. A completed distillation response therefore needs no training loop of its own and joins reward-based data in one update. Appendix A proves that this absorption is exact, and Appendix C gives the distilled tokens and how λ is varied.

3.2 Decoupling Teacher Inference from Student Training

A response passes up to three completion points. Its generation and scoring finish, its group receives all G results, and, if the response is distilled, its teacher target is computed. Each step waits only for the completion it needs, and two queues keep these waits apart (Figure 2). Responses awaiting the teacher stay in the `TeacherQueue`, and only training-ready data enter the `MessageQueue`.

Teacher inference does not hold a rollout slot. The Rollouter schedules individual trajectories, so each rollout slot serves a new response as soon as generation and scoring finish. Completed responses are held by prompt until their group receives all G results, and group completion then determines the supervision branch. A mixed group goes to the `MessageQueue`, and an entirely incorrect group sends one response to the `TeacherQueue`. Neither slower responses in the group nor later teacher inference retains a completed response's slot.

A pending teacher target holds back only its own response. The Teacher takes waiting responses from the `TeacherQueue` and computes their token log-probabilities, the teacher term of δ_t. Because it draws from its own queue, one teacher batch can combine responses across groups and updates. When the batch finishes, the predictions are attached to each response, which then enters the `MessageQueue`. Only reward-based groups and distillation responses with completed targets are eligible for batch assembly.

No update waits for an unfinished teacher target. The Trainer assembles each batch only from what the `MessageQueue` already holds, so all supervision needed by that batch is already available. Reward-based groups and distillation responses then enter Eq. (2) as one update.

3.3 Teacher Supervision without Dedicated GPUs

Running the teacher on the rollout and training pools without GPUs of its own works only if the teacher is cheap to keep on those GPUs, runs only while they would otherwise be idle, and adds no extra pass to what the student already runs. Figure 3 shows how the teacher fits and when it runs.

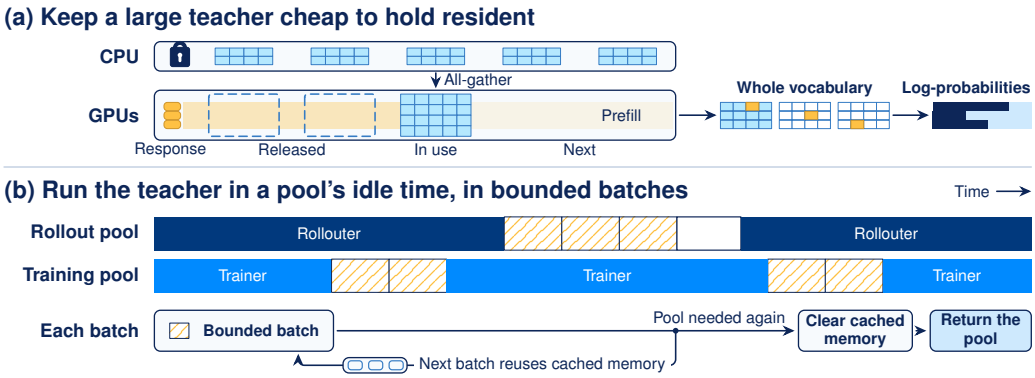

Figure 3: Teacher execution on shared GPU pools. Amber marks the student's tokens, hatched tiles are bounded teacher batches, and white intervals have no teacher work.

Keep a large teacher cheap to hold resident. GPUs of its own would have to grow with the teacher, and the BF16 weights of a 235B teacher alone take about 470 GB, more than two H200 GPUs hold. On the shared GPUs, a large teacher must instead fit into memory already filled by the student's weights, KV cache, or optimizer states, so the memory it keeps there has to stay small. The teacher only needs log-probabilities for tokens the student has already produced, so a forward-only prefill evaluates each response, with no teacher decoding, backward pass, or optimizer state. Its FSDP2 shards rest in pinned CPU memory between forwards, and each module is all-gathered onto the GPUs just before the response reaches it and released right after use, so the GPUs hold the full weights of only one module at a time (Figure 3a). On this foundation, HitchRL adds two steps that a large shared teacher needs. It loads checkpoint tensors directly into each CPU shard, since a full CPU copy on top of the resident shards of both pools would exceed host memory, and it processes response positions in fixed-size tiles while normalizing each position over the full vocabulary, keeping log-probabilities exact with bounded extra memory. Together these let even a 235B teacher share the pools of a 1.7B student (Appendices B and C).

Batch queued responses to share each weight transfer. Every forward gathers each module onto the GPUs once, however many responses it carries. Because the Teacher draws from its own queue (Section 3.2), each batch runs responses from different failed groups and updates together, so the gathering cost is shared among them. Larger batches, however, hold the GPUs longer before they can be returned. Each batch is therefore bounded by a limit on the number of responses and a per-GPU token budget, so its capacity follows the GPU count of the pool that runs it while its memory and duration stay bounded (Appendix B).

Run in each pool's idle intervals and hand the pool back within one batch. Each pool is assigned to one role. The rollout pool runs the Rollouter's generation, and the training pool runs the Trainer's forward and backward passes, while the Teacher has no pool of its own and is only a secondary consumer of either pool. Since teacher targets can be computed at any time, HitchRL runs the teacher in the stretches where a pool's own work pauses (Figure 3b). Within a pause, consecutive teacher batches reuse the GPU memory that the previous batch left cached, so this memory is cleared only when the pool changes hands. When generation or optimization needs the pool, the running batch completes, and the teacher starts no new batch, clears this cache, and returns the pool, leaving the rest queued for the next pause. Because the weights stay in pinned CPU memory, the handoff moves no teacher weights, and the pool's own work waits for at most one bounded batch (Appendix B).

Fold distillation into the student's existing update. Once the Trainer acquires the training pool, its actor forward computes current log-probabilities for every response in the batch. The same pass supplies the student term of δ_t, which joins the teacher term stored with each response. Because the distillation branch takes the recorded behavior probability as its reference and the teacher discrepancy as its advantage (Section 3.1), distillation responses skip the proximal-policy evaluation, correction weight, and group-relative advantage that only reward-based data need. Distillation therefore adds only the cost of its selected response tokens inside an update that would run anyway. Whichever pool ran the teacher and whenever it ran, its targets reach the student through this single update (Appendix C.5).

Table 1: Accuracy (%) of the Qwen3-1.7B student. Gray-blue rows are baselines on the same asynchronous system as HitchRL, and sand rows are methods from prior work.

	Math						Out-of-domain		
Method	AIME24	AIME25	AIME26	AMC23	MathArena	Olympiad	Minerva	MMLU-Pro	Mean
AReaL	5.52±.59	4.17±.54	2.29±.44	31.48±.98	3.51±.41	24.74±.57	18.66±.71	23.25	14.20
OPD → AReaL	6.25±.56	**5.63**±.52	4.06±.56	34.45±.76	3.68±.36	26.11±.47	15.62±.75	**33.64**	16.18
OPDVR	6.35±.57	4.38±.52	3.75±.54	34.53±.90	2.63±.39	24.52±.53	15.63±.74	31.30	15.39
PRIME	6.04±.59	4.90±.60	2.08±.44	34.92±.98	4.04±.39	25.33±.55	**21.23**±.77	28.52	15.88
LaSeR	4.90±.52	3.85±.50	3.02±.48	33.98±.96	3.32±.36	23.85±.55	17.10±.78	25.75	14.47
HitchRL (ours)	**8.23**±.68	4.79±.53	**4.79**±.57	**36.64**±.92	**4.33**±.39	**27.23**±.56	18.20±.77	32.07	**17.04**

Table 2: Accuracy (%) of the Qwen3-8B and Olmo-3-7B students.

		Math						Out-of-domain		
Student	Method	AIME24	AIME25	AIME26	AMC23	MathArena	Olympiad	Minerva	MMLU-Pro	Mean
Qwen3 8B	AReaL	20.21±.87	16.25±.78	16.04±.78	61.41±.96	13.52±.64	44.96±.55	**29.41**±.75	58.88	32.58
	HitchRL	**22.29**±.82	**18.02**±.70	**19.27**±.83	**62.11**±1.02	**16.11**±.62	**47.48**±.54	**29.41**±.86	**59.97**	**34.33**
Olmo-3 7B	AReaL	13.54±.82	15.52±.72	11.46±.69	48.44±.92	8.46±.55	34.94±.55	**18.57**±.91	35.13	23.26
	HitchRL	**15.00**±.86	**16.67**±.81	**12.40**±.77	**51.09**±.89	**8.60**±.54	**35.50**±.53	18.47±.78	**38.82**	**24.57**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and training. We train Qwen3-1.7B-Base, Qwen3-8B-Base (Yang et al., 2025), and Olmo-3-1025-7B (Team Olmo, 2025), each with a teacher from its own family, Qwen3-30B-A3B-Instruct-2507 for the Qwen3 students and Olmo-3.1-32B-Instruct for Olmo. A further 1.7B run uses the Qwen3-235B-A22B-Instruct-2507 teacher. Math runs train on DeepScaleR (Luo et al., 2025), and code runs train the 1.7B student with the 30B-A3B teacher on programming problems, where each response interleaves programs with their execution results on public examples, as in the environments of Section 2. All runs use 16 responses per prompt and 32 groups per optimizer update on one node with eight H200 GPUs, and every method learns only from responses the student generates, with no prepared solutions or collected successful trajectories. By default, HitchRL distills one response of each failed group with a weight that starts at $\lambda_0 = 0.3$ and falls as responses lengthen.

Evaluation and cost. Math runs are evaluated on six math and two out-of-domain benchmarks and code runs on four code benchmarks, and Mean is the unweighted average over each set. In the tables, $\pm$ is the standard error over repeated generations, 32 per problem on AIME and AMC, 8 on MathArena, 4 on OlympiadBench, Minerva, and code, and 1 on MMLU-Pro. Bold marks the best result for each benchmark, per student in Table 2, and underline marks the second best in Table 1. Table 1 and the distillation-weight ablation evaluate runs at update 400, and Tables 2 and 3, the blocking comparison, and the distillation-budget ablation at update 200. Cost is measured in generated response tokens and in wall-clock time including teacher inference, and generation throughput is tokens per second, since generation is the main bottleneck of asynchronous RL for language models (Team Olmo, 2025). Appendix D gives the remaining settings and protocols.

Baselines. Two baselines share HitchRL's asynchronous system, data, rewards, and evaluation. AReaL (Fu et al., 2025) is the teacher-free baseline, and OPD → AReaL applies on-policy distillation (OPD) with the same teacher for 100 updates before 400 of AReaL. We also compare with OPDVR (Lin et al., 2026), PRIME (Cui et al., 2026), and LaSeR (Yang et al., 2026).

4.2 MAIN RESULTS

Higher mean accuracy across student scales, families, and domains. On the Qwen3-1.7B student, HitchRL has the highest Mean in Table 1, 17.04%, against 16.18% for the strongest baseline, OPD → AReaL, and 14.20% for AReaL. In Table 2, it raises the Mean over AReaL by 1.75 points

Table 3: Qwen3-1.7B on code.

Benchmark	AReaL	**HitchRL**
HumanEval	60.67±1.40	**63.11**±1.42
LiveCodeBench	9.14±.49	**10.75**±.57
MBPP	52.20±.69	**52.40**±.65
APPS	24.18±.42	**25.50**±.41
Mean	36.55	**37.94**

Table 4: Generation throughput of Qwen3-1.7B.

Method	Teacher	Response length	Generation (k tok/s)	Relative to AReaL
AReaL	—	910	44.69	1.00×
PRIME	—	984	36.10	0.81×
LaSeR	—	970	12.80	0.29×
OPDVR	30B-A3B	1,906	15.52	0.35×
HitchRL	30B-A3B	1,503	42.52	0.95×
HitchRL	235B-A22B	1,486	40.25	0.90×

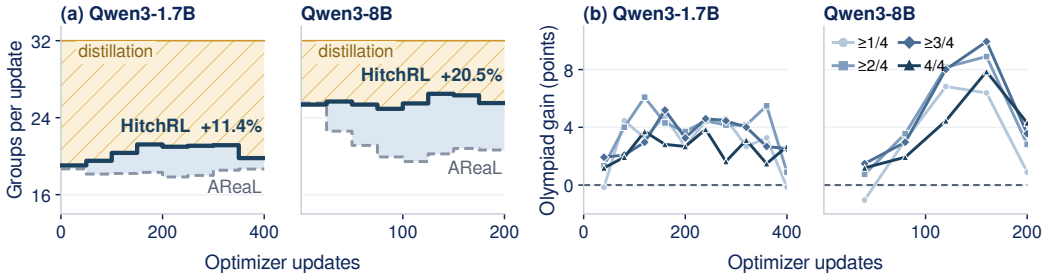

Figure 4: (a) Reward-based and distillation groups consumed per update, averaged over eight equal intervals of each run. (b) HitchRL minus AReaL in the share of Olympiad questions answered correctly in at least 1, 2, 3, or all 4 of four responses, recorded during training every 40 updates.

on Qwen3-8B and 1.31 on Olmo-3-7B. On code, it raises the 1.7B student's Mean in Table 3 from 36.55% with AReaL to 37.94%. Appendix D.4 describes the code runs.

Reaching the peaks of the baselines' Olympiad curves in 63% and 76% less wall-clock time. In Figure 1, HitchRL also reaches these peaks with 69% and 64% fewer generated tokens, counting the OPD stage of OPD → AReaL in both costs. Appendix E repeats this comparison at every target accuracy up to each baseline's peak and finds both costs of HitchRL below those of OPD → AReaL.

HitchRL keeps 0.95 of teacher-free generation throughput and takes less time than a teacher on dedicated GPUs. In Table 4, HitchRL with the 30B-A3B teacher generates 0.95 times as many response tokens per second as AReaL, more than OPDVR, which runs the same teacher inside each training step. With the 235B teacher, which shares the rollout and training GPUs of the 1.7B student, it still keeps 0.90. The asynchronous systems of Section 2 would instead give the teacher GPUs of its own. Giving the 30B-A3B teacher two of the eight GPUs leaves four for generation, and with the remaining settings fixed, HitchRL takes 7.0% less time per generated token than this dedicated variant, and 45.8% less with the 8B student. For the 235B teacher, whose weights exceed two GPUs (Section 3.3), a dedicated teacher would take at least four of the eight, whereas HitchRL runs it on none of its own. Appendix I.5 gives the setting and measurements of this comparison.

4.3 Why the Combined Update Helps

Every group in a HitchRL update carries a learning signal, and the gain appears as more consistent held-out answers. In AReaL, only the mixed-reward groups of an update yield a gradient, whereas HitchRL leaves out groups whose responses are all correct and fills the other slots with distillation of failed groups (Figure 4(a)). This supervision comes on top of reward-based learning rather than in place of it. Each HitchRL update still holds 11.4% and 20.5% more mixed-reward groups than an AReaL update for the 1.7B and 8B students, and distillation takes only 2.43% and 1.02% of the learner response tokens because it uses only part of each failed group. On held-out questions, the gain appears as consistency. In Figure 4(b), HitchRL's gains over AReaL in answering at least two, at least three, and all four of four responses correctly stay positive at every evaluation, reaching 6.08 points for 1.7B and 9.94 for 8B. Appendix F finds that 42.8% and 33.5% of the groups AReaL consumes give no gradient, follows failed training questions through a distillation update to a later reward-based update, confirms in Olmo that distilled responses carry distillation advantages

of both signs, and finds the same held-out gains in the final evaluations and at the first encounter with common training questions.

Both defaults answer a problem that adding a teacher creates. Distilling every response of a failed group multiplies the teacher work behind the group, and since a group is published only after all of its targets are complete, the queue accumulates and targets are 9.93 policy versions old when consumed, against 2.56 for one full-length response. One response instead reaches 18.2 times as many failed groups, and Mean is 16.09% against 15.75% even though the all-response variant consumes about 1.5 times as many reward-based groups, while the default 2,048-token cap raises Mean further to 16.36% with less teacher input. Beyond the teacher's workload, distillation brings another problem, as it lengthens responses. OPDVR, which applies the same teacher inside its RL loop, generates responses about twice as long as AReaL's in Table 4, and with full-strength distillation in HitchRL, incorrect responses average 5,870 tokens at a common training stage, leaving fewer mixed-reward groups for reward-based learning and a Mean of 12.10%. A weight that starts at $\lambda_0 = 0.3$ and falls as responses lengthen keeps them at 2,499 tokens and reaches 17.04%. Appendices G and H give both ablations, each keeping everything else fixed.

4.4 Teacher Execution on Shared GPUs

Deferred work fills larger teacher batches, which the GPUs evaluate in parallel, and cuts total teacher-call time by 30%. A blocking variant, which computes the targets of each update just before it, can send the teacher only the distillation responses of that one update. HitchRL's queue instead gathers them across updates, so with the 235B teacher each call carries about 1.5 times as much input, and for about the same number of distillation responses, HitchRL needs about 30% fewer calls. Each call is larger, but it costs little more, because one forward gathers the teacher's weights once and evaluates all responses of the batch in parallel. A call with 8.2 times the input of a small one takes only 3.8% longer with the 235B teacher, and one with about six times the input takes under 4% longer with the 30B-A3B teacher. Because consecutive batches within a pause also reuse cached memory, HitchRL clears the teacher's memory about 40% less often, so handing a pool back costs less. Appendix I gives these measurements, with the remaining settings fixed.

The teacher hitchhikes on whichever pool is idle while training proceeds, and its targets still arrive within a few policy versions. While teacher targets are computed, the Trainer keeps updating on other ready data, and in the 1.7B run a student update completes during the wait of 94.8% of teacher jobs. With no pool of its own, the teacher moves to wherever idle time appears. With the Qwen3-8B and Olmo-3-7B students, whose longer updates make generation pause while they train, the rollout pool evaluates 21.7% and 32.7% of distillation responses in those pauses, alongside the Trainer's computation, and for 8B this share grows from 9.5% early in training to 49.1% late. The `TeacherQueue` bounds how many jobs wait and how old they may become, and in the 1.7B run no selected response is dropped while consumed targets are 1.86 policy versions old on average and at most 4. Appendix I traces each default run's teacher work, and Appendix B.5 gives the bounds.

5 Conclusion

HitchRL separates preparing teacher supervision from consuming it, so a frozen teacher can guide asynchronous RL with no GPUs of its own while no student update waits for an unfinished teacher target. Since the teacher gives a response the same targets whenever it runs, deferring its inference provably leaves an update unchanged, so the inference waits in its own queue and hitchhikes on idle GPU time while updates proceed on ready data. This idle time, not prepared solutions or collected trajectories, thus supplies supervision for failed groups within the same update as reward-based learning. HitchRL therefore raises mean accuracy over teacher-free asynchronous RL across student scales, families, and domains and reaches the peak OlympiadBench accuracy of AReaL and OPD $\rightarrow$ AReaL in less wall-clock time. Even with a 235B teacher, which would take at least four of the eight GPUs if dedicated, it keeps 0.90 of the teacher-free generation throughput.

Ethics Statement

This work involves no human subjects, crowdsourced annotation, or personal data. The student and teacher models, the training data (DeepScaleR for math, and MBPP, APPS, and TACO-Verified for

code), and all evaluation benchmarks are publicly released, and we use them under their licenses. Training uses math and programming problems with verifiable answers, and evaluation uses public math, science, knowledge, and code benchmarks, so the study neither collects nor produces sensitive content, and we release no new dataset. Programs generated in the code runs execute in isolated containers without network access. HitchRL supplies teacher supervision to groups whose sampled responses all fail and runs the teacher on GPUs that would otherwise idle, so its main practical effect is to lower the hardware cost of teacher-guided RL. A student trained this way can inherit its teacher's errors (Appendix J) and should be evaluated for its intended use before deployment.

Reproducibility Statement

Section 3 defines the combined objective and the teacher execution, and Appendix A derives the distillation signal and the combined surrogate and shows that deferring teacher inference leaves a fixed update unchanged. Appendix B describes how teacher execution shares the GPUs, with pseudocode for bounded teacher inference with priority for student updates (Algorithm 2). Appendix C gives the supervised responses and tokens, the length-dependent weight, token alignment, teacher loading (Algorithm 3), exact teacher log-probabilities, and the combined student update. Appendix B.6 describes recovery from restarts. Section 4.1 and Appendix D give the students, teachers, training data, run settings, evaluation protocol, and cost measurement, including the number of responses averaged on each benchmark, the computation of Mean, how wall-clock time and generation throughput are measured, and the data, response format, and evaluation of the code runs. Appendices E to I state the protocol of each analysis and ablation, and Appendix I.3 defines how teacher work is tracked during training. The supplementary material contains the HitchRL implementation, the scripts of the Qwen3-1.7B and Qwen3-8B math runs of HitchRL and AReaL, the prepared training and evaluation data with checksums, the evaluation code, and CPU tests. The OPD stage of OPD $\rightarrow$ AReaL runs the on-policy distillation example of verl (Sheng et al., 2025) with the settings of Table 5, and OPDVR, PRIME, and LaSeR run the training recipes published by their authors (Appendix D.1). All models, the training data, and the evaluation benchmarks are publicly available.

AI Use Statement

We used Claude (Anthropic) through Claude Code as an assistant. Among the tasks that require disclosure, we used it to implement and optimize HitchRL and its measurement code in our training framework, to prepare, deduplicate, and reformat the training and evaluation data, to give feedback on the detailed design of experiments and ablations within an overall design set by the authors, to help formalize the mathematical claims, whose key ideas are the authors', and assist in writing their derivations and proofs in Appendix A, and to help translate the authors' notes into English, which the authors then revised. We did not use it to generate synthetic datasets, to develop theoretical models or the conceptual framework of HitchRL, to provide the key ideas of the proofs, to propose or refine the research hypothesis, or to interpret the results, which the authors did themselves from statistics that the tool helped extract from training logs and evaluation records, and qualitative or thematic data analysis is not applicable to this work. We also used it to write code for analysis and figures, to search for related literature, to format and check references, and to edit and proofread the text. The authors reviewed all AI-assisted work. Implementation changes were tested on short runs before full training, every reported number was recomputed from the logged training outputs and evaluation records, and every reference was checked against its official proceedings, journal, arXiv, or project page. 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 Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf.

AReaL Contributors. On-policy distillation. AReaL documentation, 2026. URL https://areal-ai.io/AReaL/en/algorithms/distillation.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-0679. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1d27c01ebd3e3aebe226b44fc970d803-Paper-Datasets_and_Benchmarks_Track.pdf.

William Brown. Verifiers: Environments for LLM reinforcement learning. GitHub repository, 2025. URL https://github.com/PrimeIntellect-ai/verifiers.

Justin Chih-Yao Chen, Becky Xiangyu Peng, Prafulla Kumar Choubey, Kung-Hsiang Huang, Jiaxin Zhang, Mohit Bansal, and Chien-Sheng Wu. Nudging the boundaries of LLM reasoning. In *International Conference on Learning Representations*, pp. 30315–30335, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/336229145e702cfdd4f26cf16438e7b8-Paper-Conference.pdf.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Xianwei Chen, Shimin Zhang, and Jibin Wu. f-OPD: Stabilizing long-horizon on-policy distillation with freshness-aware control. *arXiv preprint arXiv:2605.17862*, 2026b. URL https://arxiv.org/abs/2605.17862.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=9SkkifLopZ.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 36256–36282. Curran Associates, Inc., 2025. doi: 10.52202/085713-1218. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/33c00862bfa29ac72ecf630a41e19352-Paper-Conference.pdf.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, pp. 32694–32717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8ac015d409635f196f9e3e9dcfb9a94e-Paper-Conference.pdf.

Zhuowen Han, Jinwei Xiao, Zhengxi Lu, Renren Jin, Zhiyuan Yao, Yuxin Liu, Hongyan Hao, Yueqing Sun, Yu Yang, Qi Gu, Xunliang Cai, and Deyi Xiong. Distill where you fail: Recovering learning signals of negative RL-groups from adaptive teacher guidance. *arXiv preprint arXiv:2608.00782*, 2026. URL https://arxiv.org/abs/2608.00782.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c24cd76e1ce41366a4bbe8a49b02a028-Paper-round2.pdf.

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17086–17098. Curran Associates, Inc., 2022. doi: 10.52202/068431-1243. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6ceb6c2150bbf46fd75528a6cd6be793-Paper-Conference.pdf.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QkfkxyRizZ.

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. In *International Conference on Learning Representations*, pp. 58791–58831, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=7evvwwdo3z.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjun Kang, Sanghyun Park, Donghoon Kim, Minjae Lee, Minseo Kim, Rishabh Tiwari, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. AsyncOPD: How stale can on-policy distillation be? *arXiv preprint arXiv:2606.24143*, 2026. URL https://arxiv.org/abs/2606.24143.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL https://arxiv.org/abs/2607.24653.

Thanh-Long V. Le, Myeongho Jeon, Kim Vu, Viet Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in LLM reinforcement learning via entropy-guided advantage shaping. In *International Conference on Learning Representations*, pp. 121956–121982, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/c64dfa06f20f746486da743d64f4c624-Paper-Conference.pdf.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 3843–3857. Curran Associates, Inc., 2022. doi: 10.52202/068431-

0278. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/18abbeef8cfe9203fdf9053c9c4fe191-Paper-Conference.pdf.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. TACO: Topics in algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*, 2023. URL https://arxiv.org/abs/2312.14852.

Wenze Lin, Jiale Zhao, Xitai Jiang, Songde Rao, Yining Li, Shenzhi Wang, Bingxiang He, and Gao Huang. On-policy distillation with verifiable reward. *arXiv preprint arXiv:2608.24696*, 2026. URL https://arxiv.org/abs/2608.24696.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B model by scaling RL. Notion Blog, 2025. URL https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2.

MiniMax. Forge: Scalable agent RL framework and algorithm. Hugging Face blog, 2026. URL https://huggingface.co/blog/MiniMax-AI/forge-scalable-agent-rl-framework-and-algorithm.

NVIDIA. NeMo Gym: An open source library for scaling reinforcement learning environments for LLM. GitHub repository, 2025. URL https://github.com/NVIDIA-NeMo/Gym.

OpenEnv Contributors. OpenEnv: Agentic execution environments. Official documentation, 2026. URL https://huggingface.co/docs/openenv/index.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. PipelineRL: Faster on-policy reinforcement learning for long sequence generation. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=A35ak14Cyp.

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. URL https://arxiv.org/abs/2402.03300.

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. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075.

Team Olmo. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025. URL https://arxiv.org/abs/2512.13961.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026. URL https://arxiv.org/abs/2603.10165.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), pp. 27914–27961. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1355. URL https://aclanthology.org/2025.acl-long.1355/.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, volume 38, pp. 117157–117186. Curran Associates, Inc., 2025. doi: 10.52202/085713-3904. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a9d5c33e38442450f9781bd7bad3c81e-Paper-Conference.pdf.

Kaizhuo Yan, Yingjie Yu, Yifan Yu, Haizhong Zheng, and Fan Lai. OPPO: Accelerating PPO-based RLHF via pipeline overlap. In *International Conference on Learning Representations*, pp. 65571–65585, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/6b4fd4a4607f57fe65b5e276bdb17ed1-Paper-Conference.pdf.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Wenkai Yang, Weijie Liu, Ruobing Xie, Yiju Guo, Lulu Wu, Saiyong Yang, and Yankai Lin. LaSeR: Reinforcement learning with last-token self-rewarding. In *International Conference on Learning Representations*, pp. 142853–142882, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/e711dbcd6fb91da02c6d0599ec633c80-Paper-Conference.pdf.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. BREAD: Branched rollouts from expert anchors bridge SFT & RL for reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 96726–96752. Curran Associates, Inc., 2025. doi: 10.52202/085713-3235. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8b69f0fac9b83f5ba43c7a851a841a61-Paper-Conference.pdf.

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. Optimizing RLHF training for large language models with stage fusion. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 489–503. USENIX Association, 2025. URL https://www.usenix.org/conference/nsdi25/presentation/zhong.

Overview of the Appendices

The appendices are organized in four parts. Appendix A gives the derivations behind the objective of Section 3.1. Appendix B describes how the queued teacher work of Sections 3.2 and 3.3 shares the rollout and training GPUs, and Appendix C records the implementation details of Section 3. Appendices D to I follow the order of Section 4. Appendix D gives the experimental setup, Appendix E extends the cost comparison of Section 4.2, Appendices F, G, and H analyze the combined update, the distillation budget, and the distillation strength of Section 4.3, and Appendix I reports the teacher execution measurements of Section 4.4. Appendix J states the limitations.

A Derivations

This appendix gives the derivations behind the objective of Section 3.1.

A.1 Deriving the Distillation Signal and Its Clipped Update

At a fixed context s, distillation minimizes the reverse KL divergence $D_s(\theta) = \sum_a \pi_\theta(a \mid s) \log[\pi_\theta(a \mid s)/p_T(a \mid s)]$, and its gradient must correct for responses sampled from a behavior policy μ. For a finite common alphabet, let $\pi_a = \pi_\theta(a \mid s)$, $t_a = p_T(a \mid s)$, $h_a = \nabla_\theta \log \pi_a$, and $\delta_a = \log(\pi_a/t_a)$. Assume that μ and p_T are positive on the student's support, so all probabilities in the following expressions are positive, and that the teacher and proposal μ are fixed with respect to differentiation. Then

$$\begin{aligned}\nabla_\theta D_s &= \sum_a (\log(\pi_a/t_a) + 1)\nabla_\theta \pi_a = \sum_a \pi_a \delta_a h_a = \sum_a \mu_a \frac{\pi_a}{\mu_a} \delta_a h_a \\ &= \mathbb{E}_{a\sim\mu(\cdot|s)}\left[\frac{\pi_\theta(a \mid s)}{\mu(a \mid s)} \log \frac{\pi_\theta(a \mid s)}{p_T(a \mid s)} \nabla_\theta \log \pi_\theta(a \mid s)\right]. \end{aligned} \tag{3}$$

The second equality uses $\sum_a \nabla_\theta \pi_a = 0$, and the last writes the sum as an expectation over the behavior policy. The ratio accounts for the sampling policy, while the teacher–student discrepancy is evaluated at the current student. This identity underlies the distillation surrogate of Section 3.1. With the discrepancy evaluated and detached at θ_0, the importance-weighted surrogate $-\mathbb{E}_{a\sim\mu}[(\pi_a/\mu_a)\,\mathrm{sg}(-\delta_a(\theta_0))]$ has exactly this gradient at θ_0. Recomputing the discrepancy on subsequent updates preserves that local construction.

The implemented advantage A_t^{KD} of Section 3.1 uses $C = 10$. The bound C limits the signal magnitude without changing its sign, so the direction stated in Section 3.1 holds for every token. Stop-gradient instead prevents differentiation through the signal itself. These operations are distinct from PPO ratio clipping. Kimi K3 (Kimi Team, 2026) also uses a bounded, detached teacher–student log-probability difference as a token-level distillation signal. This precedent concerns the signal construction, not an identity between the complete optimization schemes.

The effect of signal clipping can also be made explicit. Write $\delta_a^C = \mathrm{clip}(\delta_a, -C, C)$, $g = \nabla_\theta D_s$, and $g_C = \mathbb{E}_{a\sim\pi}[\delta_a^C h_a]$. By the triangle inequality,

$$\|g_C - g\| \le \mathbb{E}_{a\sim\pi}[(|\delta_a| - C)_+ \, \|h_a\|]\,. \tag{4}$$

For the PPO surrogate, set $r_a = \pi_a/\mu_a$ and $A_a = -\delta_a^C$, detached at the update state. Away from clipping boundaries, define

$$E = \{A_a > 0,\ r_a > 1 + \epsilon_+\} \cup \{A_a < 0,\ r_a < 1 - \epsilon_-\}.$$

The set E contains the tokens whose ratio already lies beyond the bound in the direction favored by their advantage. The gradient vanishes on E and equals $r_a \delta_a^C h_a$ elsewhere. Its expectation $g_{C,\epsilon}$ under the unselected proposal consequently satisfies

$$\|g_{C,\epsilon} - g_C\| \le \mathbb{E}_{a\sim\pi}\left[|\delta_a^C| \, \|h_a\| \, \mathbf{1}_E\right]. \tag{5}$$

The bounds characterize the gradient modification caused by each clipping operation under the unselected proposal. Conditioning on group failure changes the sampling law, so HitchRL uses the derived signal as a selective surrogate on its retained responses. Ratio clipping can suppress a token update even when a teacher–student discrepancy remains. Its purpose is conservative sample reuse.

A.2 How Older Contexts Change the Distillation Objective

Consider a common prompt distribution and a fixed response horizon H. A completed response can be extended with an absorbing end state. The bound below concerns this fixed-horizon objective, not the variable-length normalization used in training. At position t, d_π^t and d_μ^t denote the prefix distributions induced by the current and generating policies. Define $J_q(\theta) = H^{-1} \sum_{t=1}^H \mathbb{E}_{s\sim d_q^t}[D_s(\theta)]$. If $0 \le D_s(\theta) \le D_{\max}$ on both supports, the variational characterization of total variation gives

$$\left|\mathbb{E}_{d_\pi^t} D_s(\theta) - \mathbb{E}_{d_\mu^t} D_s(\theta)\right| \le D_{\max}\, \mathrm{TV}(d_\pi^t, d_\mu^t). \tag{6}$$

Summing over t and dividing by H yields

$$|J_\pi(\theta) - J_\mu(\theta)| \le \frac{D_{\max}}{H} \sum_{t=1}^H \mathrm{TV}(d_\pi^t, d_\mu^t). \tag{7}$$

The factor is $D_{\max}$, rather than $2D_{\max}$, because D_s lies in $[0, D_{\max}]$.

Suppose further that the action distributions satisfy $\mathrm{TV}(\pi(\cdot \mid s), \mu(\cdot \mid s)) \le \eta$ for every prefix reachable under either policy. Couple the two processes at the same prompt, and at every matching prefix use a maximal coupling of their next actions. Conditional on still matching, the probability of a first mismatch at a position is at most η. By the union bound, their length-$(t-1)$ prefixes disagree with probability at most $\min\{1, (t-1)\eta\}$. The coupling inequality then gives $\mathrm{TV}(d_\pi^t, d_\mu^t) \le \min\{1, (t-1)\eta\}$. If the policies change during generation, the sum of position-specific bounds replaces $(t-1)\eta$.

The bound concerns conditional objective values at fixed θ under the stated horizon and boundedness assumptions. A sequence-level KL gradient additionally differentiates the prefix distribution. HitchRL's failure-group selection and response-token normalization define the empirical training distribution used by the shared surrogate. They are separate from this prefix-sensitivity analysis.

A.3 Combining Reward-Based and Distillation Losses into One Surrogate

Let $\mathcal{S}(r, A) = -\min\{rA, \mathrm{clip}(r, 1-\epsilon_-, 1+\epsilon_+)A\}$. For a fixed $c \ge 0$, multiplication by c preserves the order of the minimum's two arguments, giving $c\mathcal{S}(r, A) = \mathcal{S}(r, cA)$. Write each branch surrogate as its token sum divided by the shared count $N = |\mathcal{P}| + |\mathcal{D}|$, with coefficient $\bar{w}_i$ on reward-based tokens and λ on distillation tokens. Applying the identity token by token moves each coefficient into the advantage and yields Eqs. (1)–(2). The identity also holds for $c = 0$ and for negative advantages. Coefficients and detached distillation signals are held fixed during differentiation, so the equality preserves each branch's gradient under this normalization. Normalizing each branch by its own token count would instead fix the two branch shares regardless of token counts, which defines a different objective. The joint loss is an exact reformulation of the specified token-level PPO losses.

A.4 Deferring Teacher Inference for a Fixed Update

Fix a consumed batch, its aligned teacher inputs and token IDs, the current student state, the proximal snapshot, behavior probabilities, masks, and all loss coefficients. Consider two schedules that compute each required target with the same frozen teacher and evaluation convention, possibly at different times after its response was generated. Under both, the student log-probability in δ_t is evaluated at consumption from the same student state, and only the time at which $\log p_T(a_t \mid s_t)$ is computed differs. That value is identical under both schedules. The bounded advantage A_t^{KD} of Section 3.1 then gives identical detached distillation signals, and Eqs. (1)–(2) give identical losses and gradients, under the same subgradient convention at clipping boundaries. In exact arithmetic, the loss and gradient of a fixed update therefore do not depend on when the teacher runs. Backend-dependent numerical differences are separate.

The statement concerns a fixed update. A scheduler still chooses which samples are consumed, their policy ages, the mixture of reward-based and distillation data, and the length-dependent coefficient. The result lets the scheduler defer target computation without changing what each target means.

B Shared-GPU Teacher Execution

Frozen teacher targets can be computed before the Trainer consumes them (Section 3.1). This appendix describes how responses move between the two queues and how the teacher shares GPU time with generation and optimization. Appendix C specifies the calculations performed during these intervals, and Appendix D.1 gives their settings.

B.1 From a Completed Response to a Student Update

A generated response carries its tokens, reward, generation-time log-probabilities, group identity, and policy version. Routing adds a branch label, and teacher inference adds aligned teacher log-probabilities to a distillation response. The current student's probabilities are evaluated later, inside the actor update. Algorithm 1 follows these records through the default single-response path.

Algorithm 1 Response routing and student execution

Input: Teacher queue Q_T, ready-data queue Q_M, group size G, update size K

procedure RESPONSECOMPLETED(y, g)
  Release the rollout slot and retain the scored response y in group g
  if all G responses of g have completed **then**
    $c \leftarrow \sum_{y \in g} R(y)$
    if $0 < c < G$ **then**
      Publish g to Q_M for reward-based learning
    else if $c = 0$ **then**
      Retain response index zero with the token mask of Appendix C.1
      Enqueue the selected response in Q_T with its generation record
    else
      Exclude g

procedure TRAINER ▷ Runs alongside the teacher consumers
  while training continues **do**
    Collect K ready group jobs from Q_M and assemble the batch on CPU
    Request training priority, preventing new teacher calls on this pool
    Wait for the active teacher call, if any, and acquire the training pool
    Clear unused teacher cache, retaining its pinned CPU shards
    Update the current student using Algorithm 5
    if weight synchronization is due **then**
      Synchronize generation weights and refresh λ by Eq. (8)
    Release the training pool and clear training priority

The completion handler, teacher consumers, and Trainer progress independently. Each pool has a frozen teacher worker group, with a distributed copy of its weights in CPU memory. The groups lease distinct jobs from the same `TeacherQueue`. A teacher call holds its pool exclusively. Teacher inference on the rollout pool can overlap student computation on the training pool, and CPU batch collection can proceed while the training pool serves the teacher.

B.2 KEEPING TEACHER WEIGHTS OFF THE SHARED GPUS

Hold one teacher copy in pinned CPU memory. A teacher worker group retains one distributed copy of the teacher in pinned CPU memory. With a group on each pool, both copies must fit in host memory, and Appendix C.3 loads a large teacher directly into these shards.

Gather weights for prefill, then reshard. The teacher evaluates stored student responses in a forward-only pass. FSDP2 stages and all-gathers the wrapped modules as execution reaches them, without prefetching the next module, then reshards after each forward. CPU offloading keeps the parameter shards outside GPU memory between calls. This ties GPU parameter residency to module execution, while activations, logits, and communication buffers account for the other live allocations. Teacher evaluation requires no answer decoding, backward pass, optimizer state, or teacher weight updates.

B.3 GIVING GENERATION AND OPTIMIZATION PRIORITY

Generation pauses when it has run as far ahead of the Trainer as allowed and waits for updated weights, and the training GPUs sit unused while the Trainer assembles its next batch on the CPU. Student work has priority at teacher-call boundaries.

Acquire the pool when GPU work is ready. On the training pool, `MessageQueue` collection and CPU batch assembly proceed alongside the teacher consumer. Batch assembly runs in a separate CPU thread, and teacher remote procedure calls (RPCs) are awaited through another thread so that collection can continue. The Trainer requests priority immediately before its GPU computations, including proximal-policy evaluation and the actor update. It retains ownership through the associated weight synchronization, then releases the pool. Validation and checkpointing also acquire priority.

Algorithm 2 One bounded teacher call on an available pool

Input: Pool p held exclusively, P ranks, response limit R, token budget B

If student work requests p, return without starting a call
Atomically lease up to R eligible jobs in queue order, using the admission rule below
If no jobs are ready, return
If student demand arrived during the lease, return the jobs to the queue and return
Complete leases past the age limit, returning if none remain
Initialize rank lists $X_k \leftarrow \emptyset$, loads $L_k \leftarrow 0$
for each remaining job j in queue order **do**
    $k \leftarrow \arg\min_{1 \le r \le P} L_r$, $\ell_j \leftarrow$ teacher input length of j
    if $L_k + \ell_j > B$ **then**
        Leave j and the following jobs unprocessed and end packing
    Append j to X_k and set $L_k \leftarrow L_k + \ell_j$
$m \leftarrow \max_k |X_k|$ ▷ Equal row counts for distributed dispatch
while some $L_k + (m - |X_k|) > B$ **do**
    Remove the last selected job, mark it unprocessed, and update L_k and m
Return unprocessed jobs to the queue head in their original order
Raise a configuration error if the selection is empty
Pad each rank to m rows with one-token dummy inputs and zero response masks
If student demand arrived during packing, return selected jobs to the queue and return
Compute teacher targets by Algorithm 4, then remove dummy outputs
for each selected job j **do**
    if its policy age is within the limit **then**
        Attach its targets and attempt to publish j to Q_M
        If Q_M rejects the record, count it as a full-queue exclusion
    else
        Count j as an age exclusion
    Complete the lease of j
Recheck pool demand before another call, clearing unused cache when returning the pool

On the rollout pool, the router atomically checks that no generation request is in flight and closes the generation gate before teacher work starts. A request arriving during teacher execution waits at that gate and signals demand for the pool. CPU reward scoring and group assembly can continue independently of this generation gate. Once generation or weight synchronization needs the pool, the worker finishes the current teacher call, clears unused cache, and reopens the gate.

Recheck demand before starting another forward. Algorithm 2 shows the shared teacher-batch routine. Algorithm 1 gives the Trainer's ownership transfer. The caller holds the pool lock and releases it when the routine returns. Unused teacher cache is cleared when student work takes over. On the rollout pool it also holds the router's idle lease. The demand check includes shutdown, synchronization, priority requests, GPU waiters, and, on the rollout pool, generation demand. It is checked before a queue claim, after the claim, and after CPU input preparation. Unstarted jobs return to the queue when demand arrives. An active forward finishes before ownership changes, preserving distributed collective order. The batch limits bound work per call. Ownership transfer also includes completion of the active forward and memory cleanup.

Reuse cache until the pool changes roles. After each forward, teacher parameters are offloaded but the allocator may retain unused GPU blocks. Consecutive teacher calls reuse those blocks. The training pool clears unused teacher cache when the Trainer acquires the pool. The rollout pool clears the cache at the end of each teacher burst before reopening generation. Pinned CPU shards remain resident across these ownership transfers. This separates allocator cleanup from the FSDP weight transfers that each forward still performs.

B.4 Batching Work within Each Available Interval

Balance input tokens across ranks. A teacher job contains the selected response from one failed group. The consumer leases up to R eligible jobs in queue order and places whole teacher inputs on the least-loaded of P ranks. Let the length ℓ_i of input i include the teacher prompt and retained

response. The batch satisfies

$$|X| \le R, \qquad \max_{1 \le k \le P} \sum_{i \in X_k} \ell_i \le B.$$

When the next job does not fit, that job and all later jobs return to the queue in their original order. An individual input that exceeds B is rejected as a configuration error. The token budget controls packing. The response-length limit is applied separately before packing. The teacher evaluates the packed inputs and publishes completed, still-eligible responses to `MessageQueue`. A rejected publication is counted as a full-queue exclusion, and the lease is completed after the attempt.

Combine small jobs before paying for another forward. The batching routine has an optional wait that lets a small pending set accumulate before it is leased. It is off in the 30B configuration, which processes available jobs immediately, and on in the 235B configuration of Table 6. When the wait is on, the routine uses the observed mean input length $\bar{\ell}$ to choose a response-count target

$$R_{\text{target}} = \min\left(R, \max\left(1, \left\lfloor \frac{PB}{\max(1, \bar{\ell})} \right\rfloor\right)\right).$$

Before an input-length estimate is available, the target is R. A shared queue timer starts at the first nonempty claim attempt. The accumulation limit is $\max(t_{\text{poll}}, t_{\text{warm}})$, where t_{poll} is the polling interval and t_{warm} is the shortest observed teacher-call duration after the first call (zero until available). Until enough jobs arrive or that limit expires, the queue returns no work and the consumer releases the pool. The consumer does not sleep while holding a GPU lease. Closing the input releases a final partial batch. The subsequent per-rank packing check enforces B for the actual lengths.

B.5 Bounding Pending Teacher Work

The `TeacherQueue` bounds the number of pending jobs and accepts each active group once. New arrivals evict the oldest pending job when the queue is full. Returned unstarted jobs regain their queue-head position. If needed, newer jobs at the tail are removed to preserve capacity. A job remains eligible while its policy-version age is at most the configured limit. Age is checked during claiming, input preparation, and publication. These rules keep the queue bounded while generation and optimization retain priority. Queue age, dropped jobs, and consumed distillation groups record how much supervision reaches the learner under this schedule.

B.6 Preserving Completed Work across Restarts

Checkpointing acquires the pool and publication locks after active teacher work reaches a safe boundary. Alongside the student checkpoint, it saves the pending and leased `TeacherQueue` jobs, `MessageQueue`, the response scheduler, pending samples, dataloader progress, and the Trainer's update and coefficient-control state. On resume, unfinished teacher leases become pending jobs, interrupted responses return to the scheduler, and completed targets remain in `MessageQueue`. The loader checks the saved teacher specification before reusing targets. Shutdown likewise waits for an active teacher forward before releasing the teacher's memory and pool ownership.

C Teacher Targets and Student Updates

This appendix specifies the inputs, teacher targets, and student computation used by the execution in Appendix B.

C.1 Which Responses and Tokens Receive Supervision

For each entirely incorrect group, we select the response at original index zero, independently of completion order. We apply distillation to at most its first 2,048 response tokens and retain the entire prompt as conditioning context. The conditional reverse-KL identity and the invariance of stored teacher targets hold for any such selection. Prompt tokens, padding, and the excluded response suffix have zero loss weight. The normalization in Eq. (2) uses the total number of valid reward-based and distillation response tokens across training ranks and microbatches, with $N > 0$.

The reported objective uses the vanilla PPO surrogate, without an additional entropy or reference-policy KL loss and without finite dual clipping.

Algorithm 3 Initialize a large teacher directly into CPU shards

Input: Checkpoint $\mathcal{K}$, pool rank $k \in \{0, \ldots, P-1\}$
Output: Frozen teacher with pinned CPU parameter shards on rank k

$M \leftarrow$ BUILDONMETA($\mathcal{K}$.config)
SHARDFSDP2(M, P, CPU offload)
ALLOCATECPUSHARDS(M)
for all parameters θ in a shared traversal order **do**
  if $k = 0$ **then**
    $v \leftarrow$ READTENSOR($\mathcal{K}, \theta$)
    $t \leftarrow$ COPYTOGPU(v); release v
  else
    $t \leftarrow$ EMPTYGPU($\mathrm{shape}(\theta), \mathrm{dtype}(\theta)$)
  BROADCAST($t, \mathrm{source} = 0$)
  $h \leftarrow \lceil \dim_0(\theta)/P \rceil$
  $a \leftarrow \min\{kh, \dim_0(\theta)\}$; $n_k \leftarrow \dim_0(\theta_k)$
  Copy $t[a : a + n_k]$ into the pinned CPU shard θ_k
  Release t
RESTOREBUFFERS(M); SYNCHRONIZERANKS

C.2 Matching Response Tokens across Student and Teacher

In the main text, $s = (x, y_{<t})$ denotes the prompt and preceding response tokens. Each model conditions on its own rendering of this context, which may use different official input templates. The teacher still provides its probability for the same response token at the corresponding prediction position, and it does not replace the student response. The formatting maps are fixed, so delayed inference preserves the teacher target for a stored context. The conditional derivation assumes normalized distributions over the same action alphabet. For the Qwen termination aliases, the implementation sums the probabilities of the designated end tokens when computing the discrepancy δ_t, while retaining the sampled token's student and behavior probabilities in the PPO ratio. Other tokens retain their ordinary probabilities. This summation is a termination-specific surrogate convention. Response truncation does not insert a synthetic end token or move the original terminal target to the truncation boundary.

C.3 Loading a Large Teacher Directly into CPU Shards

Because each pool keeps its teacher copy in host memory (Appendix B.2), assembling an additional full CPU state dictionary during loading would add another model-sized allocation, which for the 235B teacher would exceed the node's host memory. Direct loading is an optional initialization path for this case, switched on in the 235B configuration of Table 6. Algorithm 3 creates the model on the meta device, applies FSDP2 sharding, and allocates the destination CPU shards. Rank zero reads and broadcasts one parameter tensor at a time. Each rank copies its slice and releases the staging tensor. Additional checkpoint staging scales with the largest parameter tensor, while the destination shards remain resident. This loading path supports the untied Qwen3 mixture-of-experts (MoE) teacher with a one-dimensional FSDP2 mesh and first-dimension sharding.

C.4 Computing Exact Teacher Log-Probabilities

For each retained response position, the teacher returns the log-probability of the student's sampled token. The normalization covers the full vocabulary rather than a top-k subset, so the returned log-probability is exact. The implementation selects response rows and promotes logits to 32-bit floating point (FP32) within a token-position tile, computes their normalizers, and retains only the selected log-probabilities. For tile size b and vocabulary size V, the additional normalization workspace scales as $O(bV)$ and avoids a full FP32 copy over all response positions. The model's original logits remain part of the forward allocation. Input formatting and token alignment are defined in Appendix C.2.

For an ordinary token, $E_i = \{a_i\}$. At a designated Qwen end token, E_i contains the termination aliases of Appendix C.2. Prediction positions refer to the logits immediately preceding the

Algorithm 4 Teacher log-probabilities with bounded normalization workspace

Input: Aligned teacher inputs X, prediction positions p_i, target token sets E_i, tile size b

Evaluate X with the frozen teacher, gathering and resharding each FSDP module in turn
Let Z be the resulting logits, with vocabulary size V
for each tile I of at most b retained response positions **do**
    $H \leftarrow \mathrm{FP32}(Z[p_I, 1:V])$
    $n_i \leftarrow \mathrm{logsumexp}_{v=1}^{V} H_{iv}$ for $i \in I$
    $z_i \leftarrow \mathrm{logsumexp}_{v \in E_i} H_{iv} - n_i$ for $i \in I$
    Store z_I at the corresponding response positions and release H
Return the aligned teacher log-probabilities z

Algorithm 5 One student update with reward and teacher supervision

Input: Ready batch, current student θ, proximal reference, fixed weight λ

Count valid response tokens over all ranks and microbatches, $N \leftarrow |\mathcal{P}| + |\mathcal{D}|$
On reward-based rows, evaluate the proximal policy and compute GRPO advantages
$\bar{w}_i \leftarrow \min\{\pi_{\mathrm{prox}}(a_i \mid s_i)/\mu_i(a_i \mid s_i), \tau\}$ for $i \in \mathcal{P}$
Clear accumulated student gradients
for each microbatch on each training rank **do**
    $l_i \leftarrow \log \pi_\theta(a_i \mid s_i)$ from the current student forward
    for each valid response token i **do**
        if $i \in \mathcal{D}$ **then**
            $\widetilde{l}_i \leftarrow$ student log-probability with the end-token convention
            $A_i \leftarrow \mathrm{sg}[\mathrm{clip}(z_i - \widetilde{l}_i, -C, C)]$
            $U_i \leftarrow \lambda A_i$, $q_i \leftarrow \mu_i(a_i \mid s_i)$
        else
            $U_i \leftarrow \bar{w}_i A_i^{\mathrm{GRPO}}$, $q_i \leftarrow \pi_{\mathrm{prox}}(a_i \mid s_i)$
        $r_i \leftarrow \exp(\mathrm{clip}(l_i - \log q_i, -20, 20))$
        $h_i \leftarrow -\min\{r_i U_i, \mathrm{clip}(r_i, 1-\epsilon_-, 1+\epsilon_+)U_i\}$
    Backpropagate the microbatch token sum with the global normalization $1/N$
Take one optimizer step with the accumulated, reduced gradients

target tokens in the teacher input. Each tile covers the full vocabulary, while only the selected log-probabilities are returned.

C.5 Computing the Combined Student Update

The Trainer holds the training pool throughout Algorithm 5. The stored teacher and behavior probabilities are fixed inputs. The actor forward evaluates the current student, and the same forward supplies both the detached distillation signal and the differentiable policy ratio. Reward-based rows alone receive proximal-policy evaluation and group-relative advantages. The proximal policy is fixed to the student weights before the first update of each weight-synchronization cycle and retained for that cycle. Microbatches accumulate gradients before one optimizer step.

Here z_i is the teacher value already attached to the response. The end-token convention changes $\widetilde{l}_i$ and z_i together, while l_i and q_i in the ratio retain the sampled token's probabilities. Detaching A_i leaves the ratio as the distillation gradient path. All other coefficients and reference probabilities are fixed during backward. The exponent is bounded before evaluation for numerical stability. With gradient averaging across P ranks, each local token sum is scaled by P/N to give the global token mean. Masked tokens contribute zero, and an empty local branch leaves the shared denominator unchanged.

C.6 Length-Dependent Distillation Weight

At each weight synchronization, let $\bar{\ell}$ be the mean response length of the most recent 2,048 or fewer completed training responses. The coefficient for subsequent updates is

$$\lambda = \lambda_0 \exp\left[-\max\left(0, \frac{\bar{\ell}}{1{,}024} - 1\right)\right]. \tag{8}$$

Prompt tokens are excluded from the length measurement. The coefficient is held fixed within each optimizer update and retained if no new length measurement is available. This controller is a response-window feedback rule. Shorter recent responses can restore the weight toward λ_0, and past length excess is not accumulated. The main configuration uses $\lambda_0 = 0.3$. Appendix H compares it with decay from 1 and with fixed strength. The rule changes the coefficient in the shared objective, not the distillation target or the form of the surrogate.

D Experimental Setup

This appendix gives the run settings, the evaluation protocol, and the cost measurement of Section 4.1.

D.1 Run Settings

Every run uses one node with eight H200 GPUs. OPDVR, PRIME, and LaSeR use the training recipes published by their authors, with the 16 responses per prompt and 32 groups per optimizer update of our runs. Table 5 lists the training settings of HitchRL and of the baselines on the same asynchronous system, and Table 6 lists the shared-GPU settings of the 30B and 235B teacher runs. Both teacher configurations run the same implementation. Direct checkpoint loading (Appendix C.3) and the wait to combine small batches (Appendix B.4) are optional features of that implementation, switched on by configuration.

D.2 Evaluation Benchmarks and Protocols

Math runs are evaluated on six math benchmarks, AIME 2024, 2025, and 2026, AMC 2023, MathArena 2025 (Balunović et al., 2025), and OlympiadBench (He et al., 2024), and on the out-of-domain Minerva (Lewkowycz et al., 2022) and MMLU-Pro (Wang et al., 2024). Code runs are evaluated on HumanEval (Chen et al., 2021), LiveCodeBench v5 (Jain et al., 2025a), MBPP (Austin et al., 2021), and the introductory problems of APPS (Hendrycks et al., 2021), with four responses per problem in the format of training (Appendix D.4). OlympiadBench is restricted to its 674 English text-only math problems. Minerva consists of 272 undergraduate STEM problems, mostly from physics, astronomy, chemistry, and engineering, and MMLU-Pro covers 14 subjects. Responses are sampled at temperature 1.0. Every reported result averages 32 responses per problem on AIME and AMC, 8 on the 183 problems of MathArena's five equally weighted non-AIME competitions, 4 on OlympiadBench and Minerva, and 1 on the full MMLU-Pro test set. Mean is the unweighted average of the eight benchmarks. The Olympiad evaluations recorded during training, shown in Figures 1 and 4(b), average four responses on the same 674 OlympiadBench problems every 40 updates.

D.3 Measuring Wall-Clock Time and Throughput

Wall-clock time sums the recorded time of the training updates, including the teacher inference inside them, and excludes initialization, evaluation, and checkpoint saving. Table 4 sums each run's wall-clock time and generated response tokens from update 5, after startup, through update 200. Throughput divides the token sum by the time sum, and response length divides the same token sum by the number of generated responses. Relative throughput divides each run's throughput by AReaL's.

D.4 Code Runs

The code runs of HitchRL and AReaL train Qwen3-1.7B-Base for 200 updates, with Qwen3-30B-A3B-Instruct-2507 as the teacher of HitchRL, on the 7,010 problems of Table 7. MBPP (Austin et al., 2021) asks for a Python function from a short natural-language description. APPS (Hendrycks et al., 2021) introductory problems are comparatively basic algorithmic problems, and its interview problems are more complex. The EASY problems of TACO-Verified, a subset of TACO (Li et al., 2023), come with tests and reference solutions. We use no APPS competition problems and no TACO problems above EASY. We remove problems without tests or solutions, and training problems

Table 5: Training settings. The first block applies to HitchRL, AReaL, and the reinforcement learning stage of OPD → AReaL, the second to HitchRL alone, and the third to the OPD stage of OPD → AReaL.

Setting	Value
Training data	DeepScaleR
Maximum prompt length	2,048 tokens
Maximum response length	8,192 tokens, or 4,096 for Olmo
Responses per prompt	16
Groups per optimizer update	32, one PPO epoch
Rollout sampling	Temperature 1.0, top-p 1.0, no top-k
Reward	Binary correctness
Reward-based objective	GRPO advantages with decoupled PPO against a proximal policy (Section 3.1)
PPO clipping	$\epsilon_- = 0.2$, $\epsilon_+ = 0.28$, no dual clipping
Importance-weight cap τ	2.0
Regularization	No entropy or reference-policy KL loss
Loss normalization	Token mean over valid response tokens across training ranks and microbatches
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.05
Learning rate	1×10^{-6}, constant, no warm-up
Gradient clipping	None
Precision	BF16 computation with FP32 parameters and gradient reduction
Actor token budget per GPU	65,536 for 1.7B, 32,768 for 8B and Olmo
Log-probability token budget per GPU	131,072 for 1.7B, 65,536 for 8B and Olmo
Weight synchronization	Every four optimizer updates
Rollout staleness threshold	1.0
Rollout : training GPUs	$6:2$
Seed	42
Distilled responses	One per entirely incorrect group, the response at original index zero, up to its first 2,048 response tokens
Distillation advantage bound C	10
Distillation weight	Length-dependent decay from $\lambda_0 = 0.3$ (Appendix C.6)
Teacher execution	Table 6
OPD stage	100 updates of the distillation surrogate of Section 3.1 on every response, over its first 2,048 tokens, with $C = 10$, a fixed weight of 1.0, and the PPO clipping, optimizer, and learning rate above, run synchronously without the importance-weight correction

Table 6: Shared-GPU settings of the 30B and 235B teacher runs. Token budgets include the prompt and retained response.

Setting	30B-A3B teacher	235B-A22B teacher
Rollout : training GPUs	$6:2$	$6:2$
Maximum responses per teacher batch	32	64
Input-token budget per rank	12,288	12,288
Pending `TeacherQueue` capacity	768	768
Maximum policy-version age	16	16
Wait to combine small batches	Off	On
Direct checkpoint loading into CPU shards	Off	On

that match an evaluation problem or another training problem in normalized statement, source URL, or 4-gram Jaccard similarity of at least 0.90.

Each response holds up to three attempts within 8,192 response tokens. After each attempt, the program runs on the public examples of the problem, or receives only a syntax check when the problem has none, and the result enters the context of the next attempt. A response ends early once its program passes the public examples. For APPS and TACO training problems with at least two tests, the first test is shown as the public example. The reward is 1 only if the final program passes all tests, and tests beyond the public examples run only after the response ends. The execution results

Table 7: Training problems of the code runs.

Source	Split	Problems	Share (%)
MBPP	Train	368	5.2
APPS	Train, introductory	2,394	34.2
APPS	Train, interview	1,691	24.1
TACO-Verified	EASY	2,557	36.5
Total		7,010	100.0

are masked from both losses, and the teacher reads them as context when it evaluates the student's tokens. Distillation covers the first 2,048 tokens that the student generates. The maximum prompt length is 4,096 tokens, and the remaining settings follow Table 5.

Evaluation uses the same format with four responses per problem at temperature 1.0, and a response counts as correct when its final program passes all tests. HumanEval has 164 problems, MBPP 499 test problems after duplicates are removed, APPS 1,000 introductory test problems, and LiveCodeBench v5 the 279 problems from contests between August 2024 and January 2025. Their public examples are the doctest examples of HumanEval, the example assertions of MBPP, the input and output examples stated in APPS problems, and the public test cases of LiveCodeBench. Because responses see these examples, the scores differ from single-submission pass@1. Mean is the unweighted average of the four benchmarks.

E Cost across Target Accuracies

Figure 5 extends the comparison beyond the two targets highlighted in Figure 1. For each baseline, we consider every target from its first evaluation to its highest score in the plotted range. At each target, cost is taken from the first observed evaluation meeting or exceeding it, with the same evaluations, prior OPD expenditure, and timing exclusions as Figure 1. No accuracy or crossing time is interpolated. Each curve changes only when an observed score is crossed. The reduction for costs C_H (HitchRL) and C_B (baseline) is $100(1 - C_H/C_B)$. Positive values favor HitchRL, and negative values favor the baseline.

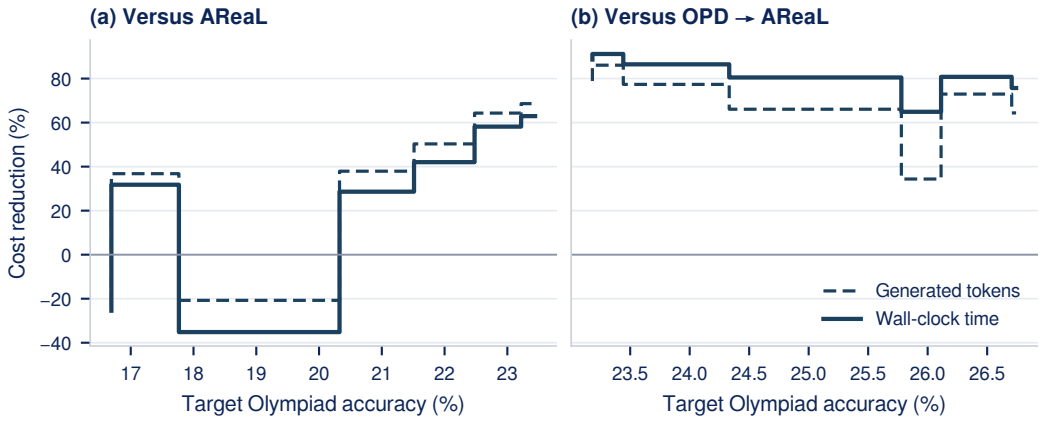

Figure 5: HitchRL's token and wall-clock time reductions at the first observed crossing of each target accuracy. Targets span each baseline's first evaluation through its best score in the plotted range. The steps reflect evaluation spacing.

The cost advantage becomes consistent as the target accuracy rises. At lower targets, the first observed crossings alternate between favoring AReaL and HitchRL. Across AReaL's record-setting scores from 21.51% to 23.44%, HitchRL reduces generated tokens by 38–69% and wall-clock time by 29–63%. HitchRL thus reaches these higher targets with fewer generated tokens and less wall-

clock time. Against OPD → AReaL, both HitchRL costs are lower throughout the plotted target range, with the baseline's prior OPD stage included.

F Training Mechanisms of Combined Supervision

In Section 4.3, every group in a HitchRL update carries a learning signal, and the gain appears as more consistent held-out answers. This appendix gives the training records and analyses behind these two findings.

F.1 Training Records and Matched Questions

We compare the full Qwen3-1.7B-Base trajectory of 400 updates and the Qwen3-8B-Base trajectory of 200 updates with their AReaL counterparts. Both HitchRL runs use the 30B-A3B teacher with length-dependent decay from $\lambda_0 = 0.3$. The 8B comparison holds the student initialization, data seed, optimizer, and GPU allocation fixed, and the 1.7B runs also share their GPU allocation. The comparison evaluates the combined training method.

Training questions are matched by their original prompt-token identities. We compare all 16 responses per group, using HitchRL's generated records before routing and AReaL's consumed groups. Both comparisons include all-incorrect, mixed, and all-correct groups. AReaL records learner consumption, while HitchRL records generation. First-encounter accuracy therefore compares the evolving policies on common training questions.

Figure 4(a) uses eight disjoint learner intervals of 50 updates for 1.7B and 25 for 8B. Figure 6 divides the matched first-encounter questions into eight nearly equal bins, ordered by their first occurrence in AReaL. Within an AReaL update, the unobserved ordering is fixed deterministically by prompt identity. Error bars are pointwise 95% percentile intervals from 4,000 paired-question bootstrap resamples per bin, conditional on these training runs. Figure 4(b) instead uses the Olympiad evaluations recorded during training every 40 updates, and Appendix F.4 applies the same thresholds to the final evaluations.

F.2 How the Combined Update Uses Supervision

Both methods consume 32 group jobs per update. AReaL retains all 16 responses of every group, but only mixed-reward groups produce nonzero GRPO advantages. HitchRL retains mixed groups for reward-based learning, applies distillation to a selected response from each failed group, and excludes all-correct groups. Table 8 counts the groups actually consumed by the learner.

Table 8: Supervision consumed over each full training trajectory. Distillation token share uses pooled response-token counts, excluding prompts.

	Reward-based groups			HitchRL distillation	
Student	AReaL	HitchRL	Increase	Groups	Token share
1.7B	7,319	8,156	11.4%	4,644	2.43%
8B	4,257	5,128	20.5%	1,272	1.02%

AReaL consumes 5,481 zero-advantage groups out of 12,800 (42.8%) in 1.7B, with 4,684 all-incorrect (36.6%) and 797 all-correct. In 8B, the corresponding count is 2,143 out of 6,400 (33.5%), with 1,270 all-incorrect and 873 all-correct. HitchRL supplies teacher supervision while consuming more mixed-reward groups. This difference reflects both routing's allocation effect and changes in the policies' responses. Group counts and token shares measure the learner's allocation of supervision. Routing uses all 16 generated responses to determine each group's branch.

F.3 Coverage and More Frequent Success at First Encounters

We retain the first observed visit to each common training question. There are 9,567 matched questions in 1.7B and 6,400 in 8B (Table 9). The observed 8B records contain 6,400 distinct AReaL

questions and 7,742 distinct HitchRL questions, with no repeated identities within either run. The 8B first-encounter gain thus appears on new prompt identities encountered during training, so the benefit extends beyond revisiting a distilled prompt.

Table 9: Accuracy over all 16 responses on common first-encounter questions. Gains are in percentage points, and intervals resample paired questions.

		Accuracy (%)			
Student	Questions	AReaL	HitchRL	Gain	95% interval
1.7B	9,567	26.06	27.57	+1.51	[1.26, 1.76]
8B	6,400	43.67	46.46	+2.79	[2.35, 3.22]

To distinguish broader coverage from more frequent success, let $k_H(q), k_A(q)$ be the correct-response counts for question q. The accuracy difference over the question set $\mathcal{Q}$ is $100 \sum_{q \in \mathcal{Q}} [k_H(q) - k_A(q)]/(16|\mathcal{Q}|)$. We partition the sum into questions both methods solve at least once, questions only one solves, and questions neither solves (Table 10). Every contribution retains the common denominator $16|\mathcal{Q}|$. Questions solved only by AReaL contribute negatively, and questions neither solves contribute zero.

Table 10: Exact decomposition of the first-encounter accuracy difference. Gains are in percentage points.

	Questions solved only by		Gain on questions	
Student	HitchRL	AReaL	Both solve	Only one solves
1.7B	955	663	+1.09	+0.42
8B	460	294	+2.14	+0.65

Questions both methods solve account for 72% (1.7B) and 77% (8B) of the aggregate gain under this question-based decomposition. More frequent success also appears at a stricter threshold. The fraction of questions answered correctly at least 8 times out of 16 increases from 23.68% to 25.46% in 1.7B and from 43.48% to 47.39% in 8B. The corresponding at-least-once fractions increase from 62.16% to 65.21% and from 80.16% to 82.75%.

The gain in Figure 6 is positive in the last seven of eight bins for 1.7B and the last six for 8B. These bins' pointwise intervals also lie above zero. The figure includes all eight bins. Across both student sizes, an advantage emerges after the initial period and persists through the remaining question-order bins.

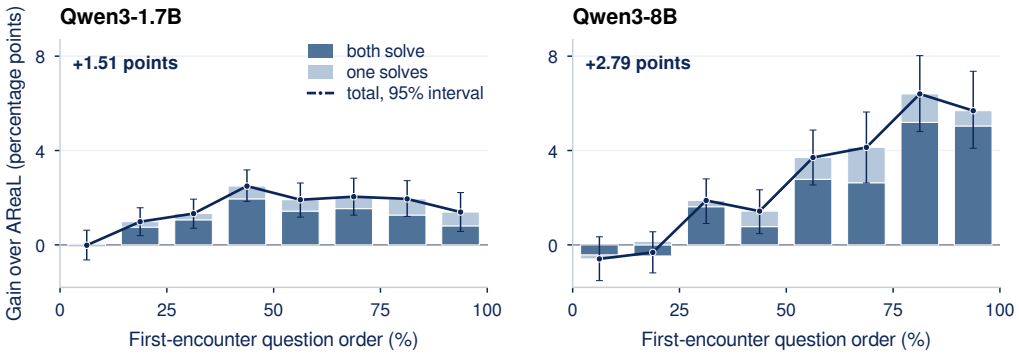

Figure 6: First-encounter training accuracy gains over AReaL, in eight disjoint question-order bins. Contributions separate questions both methods solve from those only one solves. Error bars are pointwise 95% paired-question bootstrap intervals within these runs.

F.4 Reliability of Held-Out Responses

For the 1.7B and 8B students, the final AReaL and HitchRL evaluations of Tables 1 and 2 supply four responses to each of the same 674 Olympiad questions. For each threshold $k \in \{1, 2, 3, 4\}$, we compute the fraction of questions with at least k correct responses (Table 11). If K is a question's number of correct responses, the identity $K/4 = \frac{1}{4}\sum_{k=1}^{4} \mathbf{1}[K \geq k]$ implies that averaging these four threshold rates yields Olympiad accuracy.

Table 11: HitchRL minus AReaL in the fraction of held-out questions meeting each correctness threshold on Olympiad (percentage points).

Student	$\geq 1/4$	$\geq 2/4$	$\geq 3/4$	$4/4$
1.7B	+0.74	+3.41	+3.41	+2.37
8B	+1.04	+3.26	+3.71	+2.08

Both students improve at every correctness threshold, with larger gains in repeated success than in the at-least-once rate. For 8B, the fraction of questions answered correctly in at least three of four responses rises by 3.71 percentage points. HitchRL answers more questions correctly, and more consistently.

F.5 Following Failed Experiences Back into RL

We link generated groups to completed learner updates by group identity. A revisit after distillation uses a policy whose earliest recorded version includes the preceding distillation update. This ordering connects the student's later responses to the supervision actually applied.

Across the full 1.7B trajectory, 3,575 distinct prompts receive tracked distillation. Of these, 1,320 are revisited after distillation, and 339 produce mixed rewards on their first such revisit. For the paired training-question analysis, 805 prompts are initially answered incorrectly by both methods, have second visits in both records, and receive HitchRL distillation before that visit. Among them, 158 HitchRL revisits yield mixed rewards, and 157 are subsequently consumed in completed reward-based updates.

These 157 event chains trace an experience from an all-incorrect group through a distillation update to a later reward contrast and a reward-based update. Supervision can therefore change with the student's responses to the same prompt. A failed experience receives a teacher signal, then returns to reward-based learning once its responses provide a reward contrast. The observed 8B trajectory contains no repeated prompt identities, so the revisit analysis uses the 1.7B records.

F.6 Robustness to Encounter Order and Question Composition

The methods' first-encounter orders are highly correlated (0.999 in 1.7B and 0.998 in 8B) but not identical. We restrict matched questions by the difference in their ordinal positions among all observed groups. AReaL's within-update order is unobserved and is fixed by prompt identity, affecting positions within a batch of 32 groups.

Table 12: First-encounter gains (percentage points) under tighter encounter-order matching.

	1.7B		8B	
Maximum order gap	Questions	Gain	Questions	Gain
64	6,521	+1.49	2,844	+1.20
128	9,045	+1.51	3,734	+1.15
256	9,539	+1.51	3,871	+1.27
512	9,547	+1.51	6,378	+2.81

The gain remains positive at every order-gap threshold (Table 12). All five deterministic prompt-hash partitions also have positive gains, 1.21–1.91 points in 1.7B and 2.02–3.37 in 8B. Together, these checks show that the observed advantage persists across encounter-order tolerances and question subsets.

F.7 Checking Token-Level Supervision in Olmo

As a separate check of the teacher signal entering the implemented objective, we use the Olmo-3-1025-7B student run with the Olmo-3.1-32B-Instruct teacher, since this run records token-level distillation diagnostics. The analysis covers updates 1–200. We match token records to completed actor updates and group-routing events. All 1,952 distillation responses originate in 16-of-16 incorrect groups, and all 3,540,157 valid distillation tokens reconcile with learner counts at each of the 200 updates.

The distillation advantage A_t^{KD} of Section 3.1 is positive on 56.44% of tokens and negative on 43.50%. Of the 1,952 failed responses, 1,950 contain both signs. PPO clipping is active on 8.22% of tokens. The teacher thus supplies distinct action preferences within responses whose reward advantages are uniformly zero.

G How Much of a Failed Group to Distill

HitchRL distills one response of each failed group, capped at 2,048 tokens (Section 4.3). This appendix compares this budget with two larger ones.

Three budgets over the same 200 updates. The two additional runs extend the distillation response cap from 2,048 to 8,192 tokens and distill either the response at original index zero or all 16 responses of each entirely incorrect group. Mixed-reward groups continue to drive reward-based learning, and entirely correct groups are discarded. The student, teacher, GPU allocation, generation cap, group budget, constant learning rate, and coefficient rule are unchanged. We compare both runs with the default configuration over the same 200 updates, 6,400 group jobs each. Learner and queue statistics above the rule of Table 13 cover all of these updates, and the timing and token totals below it exclude startup, evaluation, and saving.

Table 13: Accuracy, supervision delivery, and cost of the three budgets over 200 updates. Mean is evaluated as in Appendix D.2. Queue counts are taken at the end of the comparison.

	One, 2,048	One, 8,192	All 16, 8,192
Mean (%)	**16.36**	16.09	15.75
Failed groups enqueued	2,434	2,581	3,992
Groups with completed teacher targets	2,402	2,320	128
Groups excluded for queue capacity	0	0	3,094
Outstanding groups	32	261	769
Consumed distillation groups	2,394	2,312	127
Consumed reward-based groups	4,006	4,088	6,273
Distillation share of learner tokens, pooled (%)	3.17	4.19	2.19
Mean target age (policy versions)	1.75	2.56	9.93
Mean queue wait before teacher inference (s)	24.7	120.9	833.4
Mean applied λ	0.205	0.205	0.235
Wall-clock time (min)	60.57	62.24	66.65
Generated response tokens (M)	154.52	158.44	178.12
Generation (k tok/s)	42.52	42.43	44.54
Mean generated response length	1,503	1,486	1,059
Learner response tokens (M)	87.20	88.57	98.61

Per-group work determines how widely supervision is delivered. The single-response path packs responses from different failed groups into token-bounded teacher calls. Distilling all responses keeps a group's unfinished responses across calls and publishes the group only after all of its targets are ready. Generation and optimization retain priority in both cases. With all 16 responses, 3,094 of

3,992 enqueued groups (77.5%) are excluded by the pending capacity of 768, one completed result exceeds the age limit, 128 groups complete teacher inference, and 769 remain outstanding, including leased jobs. These counts account for every enqueued group, and the learner consumes 127 of the 128 completed groups. This outcome reflects the teacher work per group. Distilling all 16 responses needs 16 targets per group instead of one, so the same idle time can serve only a small fraction of the failed groups.

The delivery constraint changes the training mixture. Single-response full-length distillation reaches 18.2 times as many failed groups as the all-response variant. Distillation contributes 36.1% versus 2.0% of consumed group jobs, and 4.19% versus 2.19% of learner response tokens. The mean age of consumed targets rises from 2.56 to 9.93 versions, and the queue wait before teacher inference rises from 120.9 to 833.4 seconds. Requesting more targets per group thus delivers less supervision within this execution budget. At update 200, single-response full-length distillation reaches a Mean of 16.09% against 15.75% for all 16 responses, alongside wider supervision coverage and fresher targets (Table 13).

Table 14: Queue load of the two single-response caps over successive 40-update intervals. The first interval excludes startup.

	2,048 cap	8,192 cap
Mean queue wait before teacher inference (s)		
Interval 1	33.7	70.5
Interval 2	10.0	21.9
Interval 3	16.8	33.3
Interval 4	26.3	112.3
Interval 5	38.5	476.3
Interval 5		
Consumed distillation groups	430	354
Mean target age (policy versions)	1.72	4.89
Pending groups at the end	0	229

Target length and group width have different effects. Keeping one response while extending its cap raises consumed distillation tokens from 2.81M to 3.76M and teacher input from 3.09M to 4.03M. Both variants deliver supervision to more than 2,300 failed groups without exclusions, and their update-averaged coefficients are nearly identical. At update 200, the 2,048-token cap reaches the higher Mean, 16.36% against 16.09%, and it serves more failed groups with less teacher input and a shorter queue wait. Table 14 shows the growing queue load of the longer target, particularly in the last two intervals. The response cap thus controls how much supervision each response receives and how quickly targets reach the learner.

Cost over the same updates. We sum wall-clock time and generated response tokens over the same updates and divide the token sum by the time sum. Prompt tokens are excluded from generation and learner counts but included in teacher input. The all-response variant takes more wall-clock time yet has higher generation throughput, with shorter responses and far more reward-based consumption. Its Mean is the lowest of the three, 15.75%, while its teacher work reaches far fewer failed groups. Single-response distillation preserves broad supervision coverage at both response caps.

Target age and queue wait are averaged over consumed distillation responses, with each group contributing the same number of responses within a variant. The coefficient is averaged over optimizer updates. Table 15 lists each run's eight benchmark results.

H Distillation Strength and Learning Dynamics

HitchRL's default distillation weight starts at $\lambda_0 = 0.3$ and falls as responses lengthen (Section 4.3). This appendix compares it with a fixed weight $\lambda = 1$ and with decay from $\lambda_0 = 1$, following how each schedule changes the responses generated, the supervision reaching the learner, and validation performance. Table 16 reports each setting's validation accuracy alongside pooled response statistics after update 100. We connect these results to the reward contrasts available for RL and the teacher signal consumed during training, then examine how these quantities evolve in selected intervals.

Table 15: Accuracy (%) of the three distillation budgets of the 1.7B student, expanding the Mean of Table 13, with the benchmarks of Table 1. Rows give the responses distilled per failed group and the response cap.

	Math						Out-of-domain		
Budget	AIME24 @32	AIME25 @32	AIME26 @32	AMC23 @32	MathArena @8	Olympiad @4	Minerva @4	MMLU-Pro @1	Mean
1, 2,048	7.71±.69	**5.31**±.62	**3.65**±.55	34.84±.95	4.30±.46	**27.49**±.56	16.45±.79	**31.11**	**16.36**
1, 8,192	**8.02**±.67	4.58±.54	3.54±.51	**36.95**±.97	**4.37**±.42	26.11±.56	14.25±.72	30.88	16.09
16, 8,192	5.83±.60	3.75±.52	3.02±.48	35.08±1.01	4.02±.42	26.93±.56	**18.66**±.80	28.73	15.75

H.1 Settings and Overall Comparison

All three runs use the Qwen3-1.7B-Base student and the Qwen3-30B-A3B-Instruct-2507 teacher, with groups of 16 responses and 32 consumed groups per optimizer update. Distillation uses one response per failed group, capped at 2,048 response tokens. Generation can continue to 8,192 tokens. The two decay runs use the same controller (Appendix C.6) and differ in their initial weight.

Three distinct populations. Validation statistics concern held-out problems. Generated-response statistics include all observed completed training responses before routing or truncation for distillation. Learner statistics concern batches actually consumed by the Trainer. A completion policy version spans four optimizer updates. Correct-response length is the total number of tokens in correct responses divided by their count. Incorrect-response length uses the corresponding incorrect responses. The length-limit rate divides the number of responses reaching 8,192 tokens by all completed responses. Each response therefore receives equal weight.

For the separate interval diagnostics below, an interval ending at update u uses completion versions $(u-40)/4$ through $u/4-1$. Learner windows are indexed directly by optimizer update, so queue boundaries can make learner summaries differ from those of generated-response cohorts.

Accuracy and response length. Table 16 summarizes the three schedules. Decay from $\lambda_0 = 0.3$ reaches the highest Mean, with 28.5% shorter incorrect responses than decay from 1 and a length-limit rate of 11.65% against 25.79%.

Table 16: Distillation-strength schedules.

			Response tokens		
Schedule	λ_0	Mean (%)	Correct	Incorrect	At length limit (%)
Fixed	1.0	12.10	1,077	4,200	23.99
Length-dependent decay	1.0	16.57	1,292	4,663	25.79
Length-dependent decay	0.3	**17.04**	1,213	3,333	11.65

Evaluation records. Mean follows the evaluation of Appendix D.2, and Table 17 lists each schedule's eight benchmark results.

Table 17: Accuracy (%) of the three distillation-strength schedules of the 1.7B student, expanding the Mean of Table 16, with the benchmarks of Table 1.

	Math						Out-of-domain		
Schedule	AIME24 @32	AIME25 @32	AIME26 @32	AMC23 @32	MathArena @8	Olympiad @4	Minerva @4	MMLU-Pro @1	Mean
Fixed, $\lambda = 1$	3.75±.55	3.44±.53	2.19±.42	28.59±.97	1.97±.34	20.88±.55	12.50±.76	23.49	12.10
Decay, $\lambda_0 = 1$	6.88±.61	**5.42**±.56	**5.63**±.60	34.22±.88	3.63±.37	27.04±.52	17.92±.74	31.86	16.57
Decay, $\lambda_0 = 0.3$	**8.23**±.68	4.79±.53	4.79±.57	**36.64**±.92	**4.33**±.39	**27.23**±.56	**18.20**±.77	**32.07**	**17.04**

Figure 7 connects held-out accuracy with response length and distillation strength throughout training. Under fixed strength, incorrect responses lengthen fastest. The two decay runs reach similar Olympiad accuracy late in training while differing in their response lengths and applied supervision.

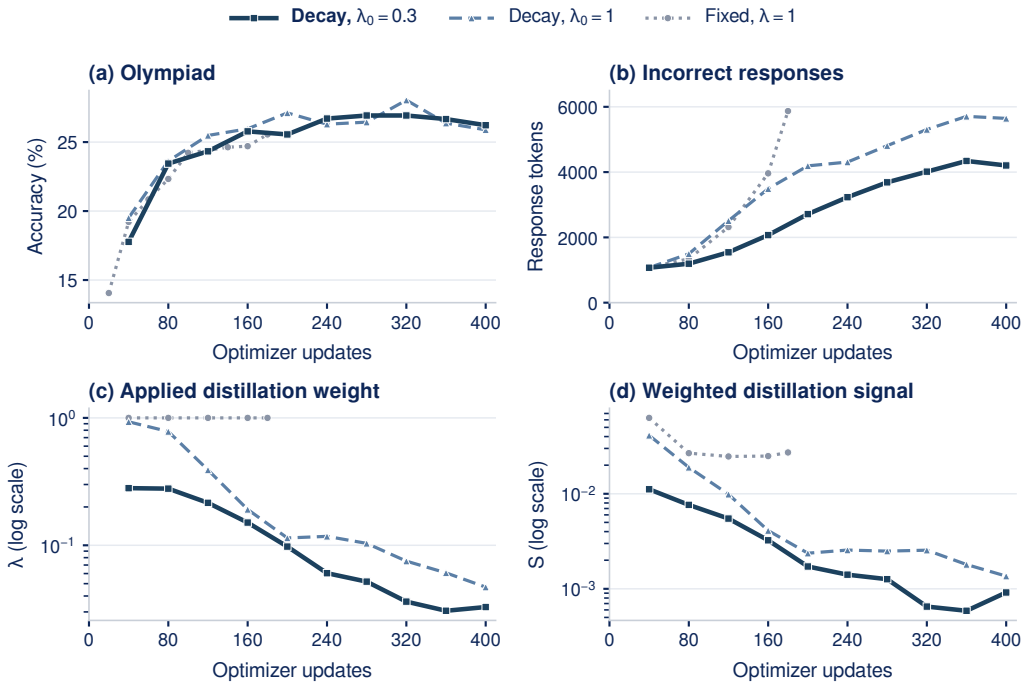

Figure 7: Distillation strength over training. (a) Olympiad accuracy over four responses per question, recorded during training. (b) Incorrect-response lengths in completion-version cohorts. (c,d) Update-averaged applied weight and weighted absolute distillation signal S from Eq. (9). Panels (b–d) use successive 40-update intervals. Points mark interval ends. No smoothing or extrapolation is applied.

Decay from 0.3 combines this accuracy with shorter incorrect responses, showing that the coefficient has a direct role in balancing learning quality and generation cost.

H.2 Long Failures and the Groups Available for RL

Whole-trajectory reward contrasts and learner allocation. For the two 400-update decay runs (Table 18), we retain all complete 16-response groups in the same trajectories used by Table 16. We average group indicators across visits within each prompt, then assign equal weight to the 9,567 prompts observed in both runs. Each prompt contributes its mean over visits throughout the trajectory. The mixed-reward fraction rises by 1.70 percentage points with decay from 0.3, while the all-incorrect fraction falls from 35.98% to 34.41%.

Table 18: Supervision over the full 400-update decay trajectories. Generated group statistics and learner consumption are separate populations.

Statistic	Decay $\lambda_0 = 1$	Decay $\lambda_0 = 0.3$
Generated groups, equal weighting of 9,567 shared prompts		
Mixed-reward groups (%)	57.67	59.37
All-incorrect groups (%)	35.98	34.41
Learner consumption, all 400 updates		
Reward-based groups	7,867	8,156
Distillation groups	4,933	4,644
Reward-based share of group jobs (%)	61.46	63.72

Each learner consumes 12,800 group jobs. The lower initial weight provides 289 more reward-based group jobs while the learner still consumes failed experiences for distillation. The consumption counts cover all learner batches, while the generated-group fractions use the shared prompts. Both measurements show a larger reward-based component alongside continued teacher supervision.

Matched prompts at a common training stage. At a common training stage, all three runs contain complete groups for 521 shared prompts. Table 19 compares the runs' response patterns at this stage. The length-limit fraction falls across the three settings, while mixed-reward groups become more frequent. This shift connects the response pattern to the routing rule, since mixed groups supply reward contrasts and all-incorrect groups supply teacher-supervised responses.

Table 19: Generated groups on 521 shared training prompts at a common training stage.

Metric (%)	Fixed $\lambda = 1$	Decay $\lambda_0 = 1$	Decay $\lambda_0 = 0.3$
Response accuracy	29.17	30.67	31.32
Responses at length limit	39.68	20.78	7.74
All-incorrect groups	39.35	34.74	29.17
Mixed-reward groups	52.98	56.81	63.72
All-correct groups	7.68	8.45	7.10

Each prompt has all 16 responses completed within the same five completion versions in every run. Prompts receive equal weight.

Long failures consume generation tokens. Across all generated responses at the same stage, responses at the length limit account for 75.1%, 55.1%, and 27.0% of response tokens for fixed strength, decay from 1, and decay from 0.3, respectively. The runs' incorrect-response lengths are 5,870, 4,089, and 2,499 tokens. The full-trajectory and matched-stage measurements thus both associate lower distillation strength with shorter failures and more reward contrasts.

H.3 The Distillation Signal Depends on More Than Its Weight

The joint loss normalizes both branches by the total number of valid response tokens. Let $N_{D,u}$ and $N_{P,u}$ be the distillation and reward-based token counts at update u, let δ_{ui} be the discrepancy δ_t of Section 3.1 for token i, with the same end-token convention as the loss, and let λ_u be the coefficient applied at update u (Eq. (8)). We define

$$f_u = \frac{N_{D,u}}{N_{P,u} + N_{D,u}}, \qquad d_u = \frac{1}{N_{D,u}} \sum_{i \in \mathcal{D}_u} |\operatorname{clip}(\delta_{ui}, -10, 10)|,$$
$$S_u = \lambda_u f_u d_u. \tag{9}$$

Since $|\operatorname{clip}(\delta_{ui}, -10, 10)|$ is the magnitude of the distillation advantage A^{KD}, S_u measures the coefficient-weighted absolute distillation signal per valid training token, before PPO ratio clipping is applied. When no distillation tokens are present, $S_u = 0$ and d_u is omitted from averages. Interval means of S_u average the products computed at each update, rather than multiplying three interval means.

Table 20: Learner supervision over all 400 updates.

Learner statistic	Fixed $\lambda = 1$	Decay $\lambda_0 = 1$	Decay $\lambda_0 = 0.3$
Applied λ	1.0000	0.2813	0.1235
Distillation token share f (%)	6.19	4.13	4.07
Absolute signal per distillation token d	0.5416	0.6028	0.6175
Weighted signal S ($\times 10^{-3}$)	33.981	8.725	3.407

Coefficients and token diagnostics are means over optimizer updates, and d excludes updates without distillation. Token share is the mean of per-update fractions, not the ratio of pooled token totals. The mean of S averages per-update products, not products of these means.

Across all 400 updates (Table 20), decay from 0.3 reduces the mean weighted signal from 8.725×10^{-3} to 3.407×10^{-3}, a 61.0% reduction relative to decay from 1. The two runs' mean discrepancies per distillation token remain similar, at 0.603 and 0.618. The lower weighted signal accompanies continued consumption of distillation groups, 4,644 with decay from 0.3 and 4,933 with decay from 1 (Table 18).

There are two routes by which distillation's weight in the joint objective changes. The controller changes λ_u explicitly. Response selection, the 2,048-token cap, and the common denominator change f_u. Longer reward-based responses can increase the denominator without adding more distillation tokens. Meanwhile, d_u changes with both the student and the failed responses that reach the learner. The current coefficient alone therefore describes only one component of the supervision being applied.

H.4 Changing Supervision Late in Training

For decay from 0.3, Table 21 compares updates 281–320 with the final interval, updates 361–400.

The applied coefficient falls by about 9%, while the weighted signal S increases by about 41%. The distillation token share rises from 3.31% to 3.83%, the per-token discrepancy from 0.6650 to 0.7707, and consumed failed groups from 413 to 480. Thus, the changing distillation examples and their share of the joint batch can outweigh a decrease in the coefficient. The training curves and signal decomposition describe how supervision evolves late in training.

H.5 Why Scalar Loss Ratios Do Not Measure the Balance

The reward-based scalar loss has a useful, directly checkable structure. Consider its reference value with policy ratios and rollout correction weights set to one. In a mixed group g with G responses, the response advantages satisfy $\sum_j A_{gj} = 0$. Let $\mathcal{G}_P$ be the mixed groups in the reward-based branch.

Table 21: Supervision in two training intervals with decay from $\lambda_0 = 0.3$.

Training interval	Late	Final
Applied λ	0.0361	0.0328
Distillation token share f (%)	3.31	3.83
Absolute signal per distillation token d	0.6650	0.7707
Weighted signal S ($\times 10^{-3}$)	0.650	0.915
Consumed distillation groups	413	480
Reward-based share of consumed groups (%)	67.73	62.50
Distillation target age (policy versions)	1.82	1.94

λ, f, d, S, and target age are update means. Group counts and shares use the batches consumed during each interval. The `TeacherQueue` drops no distillation job for overflow or staleness during the run.

Since each response's advantage is repeated over its ℓ_{gj} valid tokens, this reference loss is

$$L_P^{(r=1,c=1)} = -\frac{1}{N} \sum_{g \in \mathcal{G}_P} \sum_{j=1}^{G} \ell_{gj} A_{gj} = -\frac{G}{N} \sum_{g \in \mathcal{G}_P} \mathrm{Cov}_g(A, \ell), \tag{10}$$

where the within-group covariance uses the divisor G, and N counts valid reward-based and distillation response tokens together. The reference value isolates the contribution of response lengths and centered advantages to the scalar loss.

Substituting logged group covariances and token counts, without fitting any parameters, reproduces the observed reward-based loss closely. Across 400 updates, the correlation is 0.9999996 for decay from 1 and 0.9999995 for decay from 0.3, and the mean absolute errors are 4.27×10^{-5} and 4.41×10^{-5}. The average within-group covariance is negative in 394 and 388 updates, respectively. Shorter responses are already associated with higher advantages for much of training. A larger scalar reward-based loss consequently reflects, to a large extent, how response lengths weight centered advantages. Our use of "balance" refers to the observed combination of active supervision, useful reward contrasts, generation cost, and validation performance.

I Teacher Execution Measurements

This appendix reports the measurements behind Section 4.4. Appendix I.1 compares HitchRL with blocking teacher inference under the 235B teacher, Appendix I.2 examines the cost of individual teacher calls, Appendix I.3 follows the teacher work of the default run, Appendix I.4 compares where the teacher runs for three students, and Appendix I.5 compares HitchRL with a teacher on dedicated GPUs.

I.1 Teacher Work under Blocking and Deferred Execution

We compare the first 200 optimizer updates of the 235B HitchRL run with the 200-update blocking run. Both start from Qwen3-1.7B-Base and use the same teacher, six rollout GPUs, two training GPUs, distillation response limit, and length-dependent coefficient rule. Blocking prepares only the targets needed by the current batch. HitchRL instead batches queued work independently of that batch. Their sampled responses, batch composition, and realized coefficients can therefore differ.

We exclude updates 1–4, which contain startup, and report updates 5–200.

Teacher statistics use differences of cumulative counters between updates 4 and 200. Teacher input includes prompts and retained responses. Completed teacher work and consumed distillation can differ at the interval boundaries because prepared targets may remain queued. Over these updates, the learner consumes 2,445 distillation responses with HitchRL and 2,395 with blocking. HitchRL processes 3.049M teacher input tokens in 1,392.79 seconds over 135 teacher calls, versus 2.867M in 1,991.93 seconds over 196 calls for blocking. HitchRL performs 116 memory cleanups against 196 for blocking, taking 79.25 versus 143.86 seconds. Teacher service, GPU handoff waits, generation, and learner execution can overlap, so we do not add their timers to reconstruct pipeline time.

Table 22 summarizes these counts with the update-200 Mean. The common update-200 evaluation follows Appendix D.2, and Table 23 lists its eight benchmarks.

Table 22: Teacher work under blocking and deferred 235B teacher execution over updates 5–200.

	Blocking	**HitchRL**
Mean (%)	15.55	15.28
Distillation responses	2,395	2,445
Teacher calls	196	135
Teacher input per call (k tok)	14.6	22.6
Teacher-call time (min)	33.20	23.21
Memory cleanups	196	116

Table 23: Accuracy (%) of the 235B-teacher runs, with the benchmarks of Table 1.

	Math						Out-of-domain		
Execution	AIME24 @32	AIME25 @32	AIME26 @32	AMC23 @32	MathArena @8	Olympiad @4	Minerva @4	MMLU-Pro @1	Mean
Blocking	5.63±.62	3.75±.52	3.65±.54	**36.09**±.98	**3.64**±.39	**25.22**±.57	**17.37**±.76	29.02	**15.55**
HitchRL	**6.35**±.65	**4.48**±.57	**4.69**±.57	34.06±.99	3.53±.40	24.85±.59	14.52±.73	**29.80**	15.28

At update 200, Mean is 15.28% for HitchRL and 15.55% for blocking (Table 23).

I.2 Observed Teacher Batch Cost

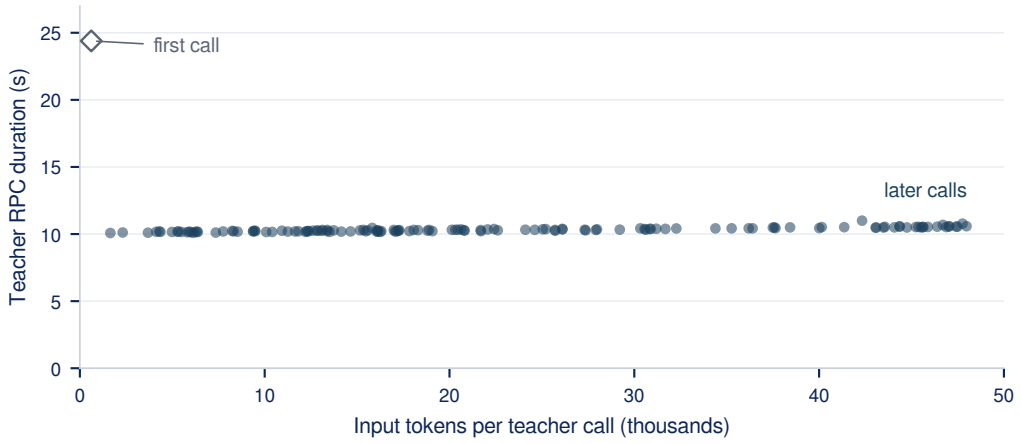

Figure 8: All 138 teacher calls in updates 1–200 of the recorded 235B-teacher run with six rollout and two training GPUs. Each point is one teacher call, and the first call is marked separately. Input tokens include both prompts and responses. Appendix I.1 counts the 135 calls in updates 5–200.

Figure 8 supports batching responses to amortize the teacher's per-call cost. Across all 137 calls after the first, batches with fewer than 8,192 input tokens ($n = 19$) average 5,234 tokens and 10.150 s, and those with 32,768–49,151 tokens ($n = 35$) average 43,025 tokens and 10.535 s. Thus 8.2 times as many input tokens take 3.8% more time, corresponding to 7.9 times the token throughput within teacher RPCs. The intermediate bins contain the remaining 83 calls, all shown in the figure. These per-call measurements explain the value of batching. Generation throughput is measured separately in Table 4. The scheduler also processes partial batches, and 37 of the 137 warm calls contain at most eight responses.

The 30B-A3B teacher shows the same pattern. We split the training-pool teacher calls of the 1.7B and 8B runs in updates 5–200 into quarters by input size. For 1.7B, the largest quarter averages

23,358 input tokens and 2.581 s per call, against 4,029 tokens and 2.483 s for the smallest. For 8B, the averages are 22,698 tokens and 2.697 s against 3,694 tokens and 2.621 s. About six times as many input tokens therefore take 3.9% and 2.9% more time.

I.3 Teacher Work during Training

We follow the teacher jobs of the default 30B-A3B run over all of its 400 updates. The other measurements in this appendix cover at most the first 200 updates, whereas this trace follows one run over its full length. A teacher job is one distillation response, from its arrival in the `TeacherQueue` to the completion of its target. Of the 4,665 jobs that enter the queue, 4,648 complete by the end of training and 17 are still waiting. Table 24 counts the 4,589 jobs whose arrival and completion are both recorded and records each update at its completion. Target age counts policy versions, each spanning four optimizer updates.

Table 24: Teacher work alongside the 400 student updates of the default 30B-A3B run.

Tracked teacher jobs	4,589
Jobs with a student update completed while pending	4,350 (94.8%)
Updates consuming distillation while other teacher work is pending	367 of 400
Consumed reward-based groups	8,156
Consumed distillation groups	4,644
Age of consumed targets, mean and maximum (policy versions)	1.86 and 4
Responses dropped for queue capacity or staleness	0

Teacher targets are therefore prepared alongside student updates rather than before them, and the prepared targets join reward-based groups in the same updates.

I.4 Where the Teacher Runs

Each pool runs teacher calls only while its own work pauses (Appendix B.3), so where the teacher runs follows the load on the two pools. Table 25 covers updates 5–200 of the three default HitchRL runs, the 1.7B and 8B students with the 30B-A3B teacher and the Olmo student with its 32B teacher. It counts the distillation responses whose teacher targets complete on each pool, and Figure 9 follows the rollout pool's share over training. With the 1.7B student, updates are short and generation keeps the rollout pool busy, so the teacher runs almost entirely in the pauses of the training pool. The 8B and Olmo students take about five times as long per update, and the rollout pool evaluates 21.7% and 32.7% of their distillation responses. For 8B, this share grows from 9.5% in updates 5–20 to 49.1% in updates 181–200. Every rollout-pool call overlaps the Trainer's computation on the training pool, and all but one Olmo call lie entirely within it, so these targets are prepared while the student trains.

Table 25: Where the teacher evaluates distillation responses over updates 5–200. Wall-clock time excludes evaluation and saving, as in Appendix D.3.

	Qwen3-1.7B	Qwen3-8B	Olmo-3-7B
Wall-clock time per update (s)	18.54	87.40	100.05
Responses evaluated on the training pool	2,304	962	1,318
Responses evaluated on the rollout pool	17	267	641
Rollout-pool share of responses (%)	0.73	21.72	32.72
Rollout-pool teacher calls	3	60	53
overlapping the Trainer's computation	3	60	53

I.5 Teacher on Dedicated GPUs

We compare HitchRL with a variant that gives the teacher GPUs of its own. The variant splits the eight GPUs into four for generation, two for the teacher, and two for training, and keeps the 30B-A3B teacher's weights resident on its two GPUs. It keeps HitchRL's `TeacherQueue` and `MessageQueue`, so its updates also proceed on ready data without waiting for unfinished teacher

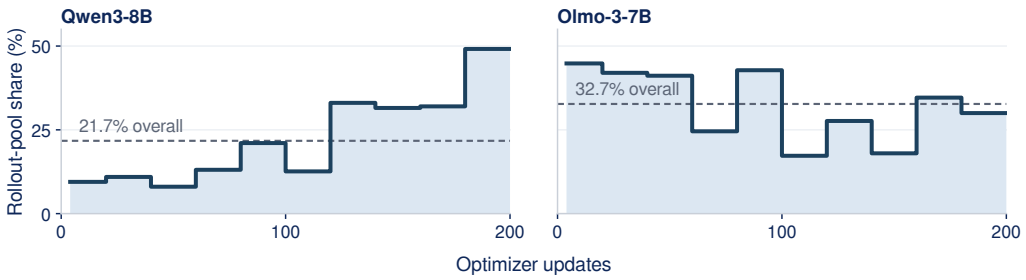

Figure 9: Share of distillation responses evaluated on the rollout pool in 20-update intervals from update 5 to 200, the first covering updates 5–20. The dashed line marks the share over all these updates.

targets, and only the GPUs that run the teacher differ. Two dedicated GPUs could not hold the 235B teacher, whose BF16 weights take about 470 GB against the 282 GB of two H200 GPUs, so a dedicated 235B teacher would take at least four of the eight GPUs. HitchRL uses six rollout and two training GPUs and runs the teacher in their idle time (Section 3.3). Each pair of runs trains the Qwen3-1.7B or Qwen3-8B student with the same data, loss, length-dependent distillation weight from $\lambda_0 = 0.3$, and teacher batch settings. These runs use a later version of the same implementation, with the settings of the 30B-A3B column of Table 6, so we compare them only with each other. Times exclude evaluation and checkpoint saving. Table 26 lists the time per million generated tokens with the distillation each run consumes.

Table 26: HitchRL and a teacher on two dedicated GPUs.

	Qwen3-1.7B		Qwen3-8B	
	Dedicated	**HitchRL**	Dedicated	**HitchRL**
GPUs (generation : teacher : training)	4 : 2 : 2	6 : 0 : 2	4 : 2 : 2	6 : 0 : 2
Time per million generated tokens (s)	22.58	21.01	85.93	46.57
Distillation responses consumed	757	716	647	653
Distillation tokens consumed	746,033	704,569	673,186	674,310
Mean distillation weight λ	0.2884	0.2883	0.250	0.235

In the dedicated variant, generation keeps four of the eight GPUs, and the idle intervals of the generation and training pools go unused, because the teacher runs only on its own GPUs. This costs far more with the larger student. With similar amounts of distillation in each pair, HitchRL takes 7.0% less time per million generated tokens than the dedicated variant with the 1.7B student and 45.8% less with the 8B student (Table 26).

J LIMITATIONS

Each configuration is a single training run, so our standard errors and intervals reflect variation over generations and questions within a run rather than seed variation. Distillation also inherits the teacher's errors.