# Self-Trained Verification for Training- and Test-Time Self-Improvement

**Chen Henry Wu, Aditi Raghunathan**
Carnegie Mellon University
{chenwu2,aditirag}@cs.cmu.edu
Project page: https://ar-forum.github.io/stv-webpage/

## Abstract

Self-improvement at scale has been a longstanding goal for reasoning models, and there are two natural places to do it: at test time, through verification-refinement (V-R) loops; and at training time, through self-training methods. Both are gated by the same bottleneck: the verifier. V-R loops stall when verifier scores inflate while accuracy stagnates, and when feedback is too generic to act on; self-training fails similarly when bad self-generated data are added to training. Better verifiers would unlock both, but the capability we want to train, i.e., catching self-generated errors, lacks the training signal. To address this challenge, we propose *self-trained verification* (STV). Our key observation is that, while a model cannot catch these errors alone, it can when shown the reference solution. We turn this asymmetry into a supervision target and train the verifier to imitate a more informed version of itself. At test time, STV substantially improves V-R loops on hard problems, while alternatives (e.g., SFT, RL on verifier scores, and even meta-verifiers) do not. STV roughly doubles accuracy on hard math and lifts it $14\times$ on scientific reasoning tasks ($1.5\% \rightarrow 21\%$). At training time, we additionally train the generator with STV verifier feedback inside the V-R loop – a procedure we call *verifier-in-the-loop training* (ViL). Starting from an RL-converged generator, ViL yields a further $33\%$ gain in pass@1. More notably, the generator's standalone pass@1, with no verifier at test time, climbs $30\%$ relative past where standard RL had converged. Hence, the next frontier in reasoning may lie in how we train for and with verification.

## 1 Introduction

The long-term promise of reasoning models is self-improvement, where models can use their own attempts, mistakes, and feedback to solve harder problems. There are two natural places to do it. At test time, *verifier-guided refinement* (Huang and Yang, 2025; Shao et al., 2025) turns additional inference compute into better solutions, where a generator and verifier alternate, with the generator producing a solution, the verifier providing feedback to it, and the generator refining. This drives recent breakthroughs on the hardest reasoning tasks like IMO and frontier math (Feng et al., 2026). At training time, *self-training* (Zelikman et al., 2022; Singh et al., 2023) recycles the generator's own, verified attempts into supervision. While recent frontier systems lean increasingly on verification, their gains are capped by verifier quality: shallow critiques (Huang et al., 2023; Kamoi et al., 2024; Stechly et al., 2024; Song et al., 2026) and miscalibrated scores (Dang et al., 2026; Pan et al., 2024) cause additional compute to amplify plausible but wrong reasoning.

*In this work, we develop a scalable recipe for training verifiers and generators that unlocks self-improvement at both ends: test-time iterative refinement (by $14\times$ on hard reasoning tasks) and training-time generator improvement (by $30\%$ beyond the RLVR plateau).*

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

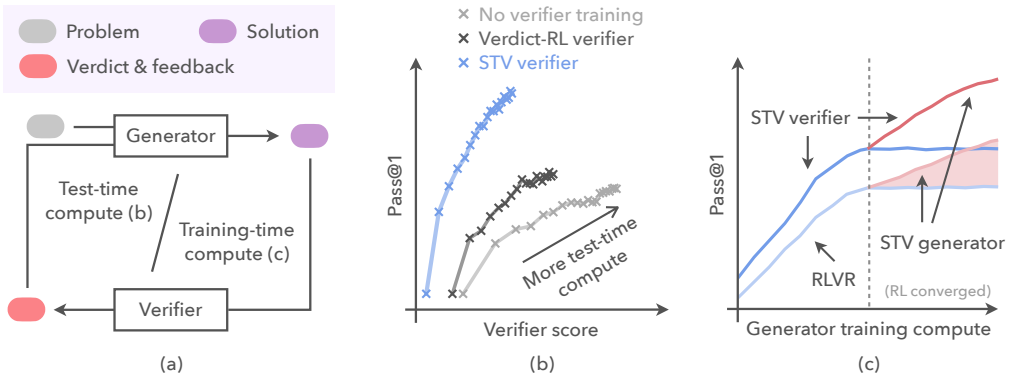



Figure 1: **(a)** We propose **self-trained verification** (STV), a scalable recipe for training verifiers that judge solutions and provide feedback for generators to refine. STV unlocks both self-improvement for test-time iterative refinement and training-time generator improvement. **(b)** STV scales much better with test-time compute than untrained or verdict-only-RL verifiers. **(c)** After standard RLVR converges, training the generator with verifier feedback in the loop (ViL) yields further pass@1 gain, along with a *standalone* gain (shaded) visible without any verification at test time.

Final-answer labels can teach a verifier to judge whether a solution is wrong, but not to find the flaw in a plausible but incorrect one – a missed boundary case, an invalid reduction, or a lemma used outside its hypotheses. Existing systems sidestep this supervision gap with meta-verifiers trained on human feedback (Shao et al., 2025) or with external feedback such as a stronger model (Song et al., 2026). We instead exploit a simple asymmetry: **diagnosis is easier given a reference solution**. A model that struggles to find the flaw on its own can often locate it when shown a reference, because the task shifts from solving the problem to comparing a candidate against a known-good solution. Our pipeline, *self-trained verification* (STV), turns this privileged setting into supervision. A reference-conditioned verifier acts as a teacher, and on-policy distillation trains an unconditioned verifier to match its feedback distribution. At test time, the resulting verifier runs without references.

STV makes verifier-guided refinement effective on problems out of reach for the base generator. On hard DAPO math problems (Yu et al., 2025) – including a *Hardest* split where Qwen3-8B has zero pass@1– an STV verifier roughly doubles final-round pass@1 over an untrained verifier, with gains compounding over 20+ refinement rounds. Standard alternatives (SFT on reference-conditioned feedback, RL on verdict accuracy) saturate quickly and yield much smaller gains (Figure 1, middle). The gains extend beyond math: on the hardest SciKnowEval problems, STV lifts pass@1 from 1.5% without verification to 21.0%. With enough verification compute, Qwen3-8B guided by STV outperforms the much larger Qwen3-32B generator, suggesting that trained verification can substitute for generator scale on hard reasoning problems.

Beyond test-time refinement, STV also enables a new way to training-time self-improvement: training the generator inside the V-R loop with the trained verifier providing feedback, a procedure we call *verifier-in-the-loop training* (ViL). Starting from a generator already converged under standard RLVR, we continue training it with feedback from a frozen STV verifier. Unlike standard self-training, the training signal here remains verifiable since the verifier's feedback only provides additional context that helps the generator to maximize that reward. ViL unlocks gains beyond the RLVR ceiling: final-round pass@1 improves by 33% relative with the verifier present, and standalone pass@1 improves by 30% relative even with no verifier at inference. However, the same compute spent extending standard RLVR yields no further gain (Figure 1, right, shaded).

In summary, STV improves both modes of verifier-driven self-improvement: at test time, it makes refinement scale; at training time, it breaks through the standard RLVR plateau, with gains that persist even with no verifier at inference. Our results point to a shift in where reasoning gains come from: from learning to verify – and using stronger verifiers to train stronger generators. This suggests an iterative route to self-improvement, where better verifiers enable more reliable test-time refinement and richer generator training, while better generators produce harder attempts for future verifier training. Developing scalable ways to train verifiers without human-graded feedback may therefore be a key frontier for reasoning models that improve at both test time and training time.

## 2 Problem Formulation

We consider a reasoning problem $x$ with ground-truth answer $y^\star(x)$. A verification-refinement (V-R) pipeline couples a generator $G$ with a verifier $V$. These may be two separate models or the same model invoked with different prompts; we instantiate both cases in our experiments. At round 0, the generator produces an initial solution $y_0 \sim G(\cdot \mid x)$. At each subsequent round $r \geq 1$, the verifier examines the latest solution and returns a verdict together with natural-language feedback,

$$(v_r, f_r) \sim V(\cdot \mid x, y_{r-1}), \qquad v_r \in \{\text{accept}, \text{reject}\}.$$

If $v_r = \text{reject}$, the generator refines the solution based on the feedback: $y_r \sim G(\cdot \mid x, y_{r-1}, f_r)$. The loop ends at the first acceptance or after $R$ rounds.

## 3 Self-Trained Verification

We propose *self-trained verification* (STV), which trains this capability without human annotation: while a model cannot reliably generate feedback for the attempted solution from scratch, it *can* locate logical gaps when shown the reference solution alongside the attempt, and this reference-conditioned distribution becomes a natural supervision target.

At a high level, STV trains the verifier to imitate a more informed version of itself – a teacher that is shown the reference solution and can therefore identify errors in $y_{r-1}$ more reliably. We parameterize the student verifier as $V_\theta(\cdot \mid x, y_{r-1})$ and the reference-conditioned teacher as

$$V^\star(\cdot \mid x, y_{r-1}, y^\star(x)), \tag{1}$$

both of which use the same model under different prompts.

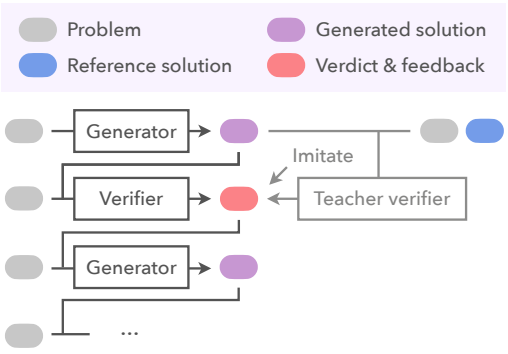


Figure 2: Overview of self-trained verification.

We then align the student's output distribution with the teacher's. To do this, one can use either SFT or on-policy distillation (OPD). OPD matches the student's distribution $z = (v_r, f_r)$ to the teacher's:

$$\mathcal{L}_{\text{OPD}}(\theta) \;=\; \mathbb{E}_{(x, y_{r-1})}\, D_\alpha\big(V_\theta(\cdot \mid x, y_{r-1}) \,\big\|\, V^\star(\cdot \mid x, y_{r-1}, y^\star(x))\big)\,, \tag{2}$$

where $(x, y_{r-1})$ is drawn from the generator $G$'s rollouts on training prompts, $V_\theta(\cdot \mid x, y_{r-1})$ and $V^\star(\cdot \mid x, y_{r-1}, y^\star(x))$ are full distributions over the response sequence $z$, and $D_\alpha$ is the $\alpha$-divergence with $\alpha = 0.5$ (Jensen-Shannon). On the other hand, SFT trains the student on $(v, f)$ pairs sampled from the teacher $V^\star$. The broader question of whether on-policy methods work better than SFT remains debated (Chu et al., 2025; Matsutani et al., 2025), and we make no general claim here. On the models and datasets we experimented with, we see that OPD improves the V-R performance while SFT fails (Section 5.2). We attribute this to the *off-policy* nature of SFT: $V_\theta$ is trained on sequences $V^\star$ produces, but at test time it must generate its own, and once it drifts from $V^\star$ it encounters prefixes never seen during training.

We also add a verdict-RL component to improve verdict accuracy, where we use RL with reward $\mathbb{I}[v_r = \mathbb{I}[y_{r-1} = y^\star(x)]]$. Putting everything together, the full STV objective is:

$$\mathcal{L}_{\text{STV}}(\theta) \;=\; \mathcal{L}_{\text{OPD}}(\theta) \;+\; \lambda \cdot \mathcal{L}_{\text{RL}}(\theta). \tag{3}$$

## 4 Verifier-in-the-Loop Training for Generator

Once we have a trained verifier $V_\theta$, the natural next step is to train the generator $G$ to better use $V_\theta$'s feedback inside the V-R loop. We call this procedure *verifier-in-the-loop training* (ViL). We instantiate it as standard RL on the generator: each episode unrolls a multi-turn V-R rollout on a problem $x$ – the generator produces $y_0 \sim G(\cdot \mid x)$, the (frozen) verifier produces $(v_1, f_1) \sim V_\theta(\cdot \mid x, y_0)$, the generator refines $y_1 \sim G(\cdot \mid x, y_0, f_1)$, and so on – and the reward is the verifiable correctness of the final $y_r$ against $y^\star(x)$. Only $G$'s parameters are updated; $V_\theta$ stays frozen.

A natural expectation is that this training improves $G$'s performance at the V-R task at test time, since the generator now knows how to incorporate feedback. What is less obvious is whether the generator's behavior *without* the loop should change at all. One might expect minimal transfer: $G$ could in principle achieve high reward by relying on $V_\theta$'s feedback to correct self-generated over later rounds, so there is no obvious pressure on $y_0$ itself to improve.

Our experiments (Section 5.4) show otherwise. The expected V-R gain materializes – final-round pass@1 climbs by 33% relative – but more surprisingly, $G$'s round-0 pass@1 also climbs by roughly 30% relative, even when we start from a generator already converged under standard RL. We call this *training-time self-improvement*: training $G$ inside the V-R loop teaches it skills that surface in its first attempt, without any verifier present at test time.

# 5 Experiments

## 5.1 Setup

We use Qwen3-8B as the base model for both the generator and verifier in our main experiments unless specified. Given the asymmetry in output lengths, training a verifier is often more affordable than training a generator, so we consider two generators to simulate different training compute budgets: the base Qwen3-8B generator and a continual-trained generator trained with RLVR on the training set until convergence. An overview of experimental settings is shown in Figure 3, which we will refer to in later sections.

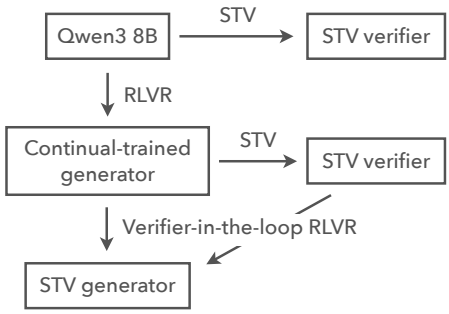



Figure 3: Overview of our experiments.

Our training and test data are composed of hard math problems from DAPO (Yu et al., 2025), split into two difficulty bins based on Qwen3-8B's pass@1 estimated with 32 rollouts: **Hard** ($0 <$ pass@1 $< 0.2$) and **Hardest** (pass@1 $= 0$). Training is performed on the joint of both bins. Since math data often share problems across sources, we embed all problems with OpenAI's `text-embedding-3-large` and remove test problems whose cosine similarity to any training problem exceeds 0.8, leaving ∼150 problems per test bin. For each test problem, we run 32 independent V-R loops for up to 20 rounds. For simplicity, we use the same training problems for verifier and generator training, though this is not necessary and we leave data selection for verifier training to future work.

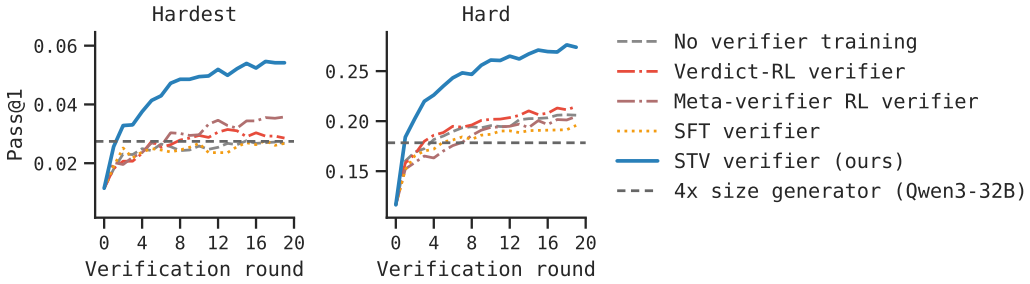



Figure 4: Pass@1 across verification rounds on the base generator.

## 5.2 Verifier-Guided Refinement

**Base generator.** We first evaluate STV on the base Qwen3-8B generator (Figure 3 top), holding the generator fixed and comparing different verifiers (Figure 4). Without any verifier training (*No verifier training*), pipeline pass@1 stagnates quickly. Training the verifier with RL on verdict accuracy alone (*Verdict-RL verifier*) helps only marginally. We also probe the meta-verifier approach by Shao et al. (2025) (*Meta-verifier RL verifier*), which uses a separate model to grade the verifier's feedback as an additional RL reward signal; lacking access to their meta-verifier, we substitute GPT-5.2 as a proxy. This setup shows little effect. SFT on teacher-generated traces (*SFT verifier*) shows no gains, consistent with the off-policy argument (Section 3). In contrast, the STV verifier lifts final-round

pass@1 by up to 2× over the untrained-verifier pipeline ($p$ value $< 0.01$), with gains compounding over 20+ rounds. Notably, the STV-guided 8B at the final round (5.5% on Hardest, 27.4% on Hard) outperforms the 4× larger Qwen3-32B without verification (2.7%, 17.8%), which shows that a good verifier provides gains beyond what a much stronger generator can achieve.

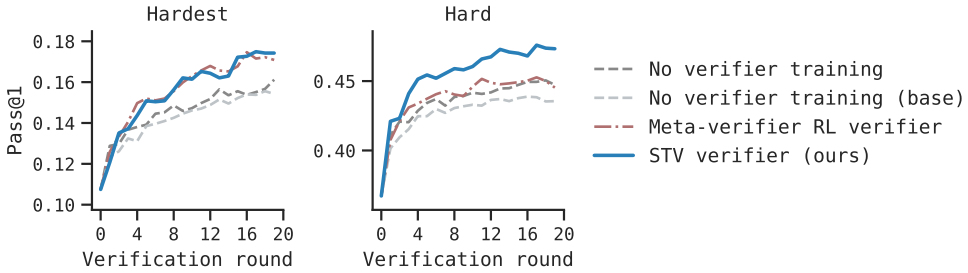



Figure 5: Pass@1 across verification rounds for the continual-trained generator.

**Scientific reasoning results.** We then test STV on scientific reasoning tasks. We apply the same recipe to SciKnowEval (Feng et al., 2024), a multi-domain science benchmark spanning chemistry, biology, physics, and materials science. We follow the math protocol to split problems by Qwen3-8B's pass@1 into **Hardest** (pass@1 $= 0$) and **Hard** ($0 <$ pass@1 $< 0.2$) and evaluate the V-R loop with Qwen3-8B as generator. Table 1 shows that untrained self-verification fails to improve over the no verification baseline, while STV lifts pass@1 by an order of magnitude on Hardest and nearly 4× on Hard ($p$ value $< 0.01$). Going further, the STV-guided 8B even beats the 30× larger Qwen3-235B-A22B on both bins (21.0% vs 8.0% on Hardest, 42.4% vs 23.6% on Hard), as well as the 4× larger Qwen3-32B (8.9% Hardest, 25.1% Hard). This suggests that the bottleneck STV addresses is a domain-general capability.

Table 1: Scientific reasoning results.

| | Hardest | Hard |
|---|---|---|
| No verification | 1.5 | 11.5 |
| No verifier training | 2.1 | 11.4 |
| STV verifier (ours) | **21.0** | **42.4** |
| 4× model size | 8.9 | 25.1 |
| 30× model size | 8.0 | 23.6 |

**Continual-trained generator.** Would STV's gains compound on top of a generator already trained to be stronger, or would the generator's RL training absorb the benefit? We investigate this setting (Figure 3 middle row) in Figure 5. The continual-trained generator (RLVR on the training set, to convergence) starts at higher pass@1 (10.8% and 37.2%) than the base model. We see that RLVR training of the generator already, but marginally, improves self-verification: the continual-trained generator's self-verification (*No verifier training*) slightly outperforms using Qwen3-8B as verifier (*No verifier training (base)*). In contrast, applying STV, initialized from the continual-trained generator checkpoint, brings substantial further gains over self-verification ($p$ value $< 0.05$), with gains compounding over refinement iterations. This confirms that STV's gains are not absorbed by stronger generator RL training, and trained verification adds real lift on top of a converged generator.

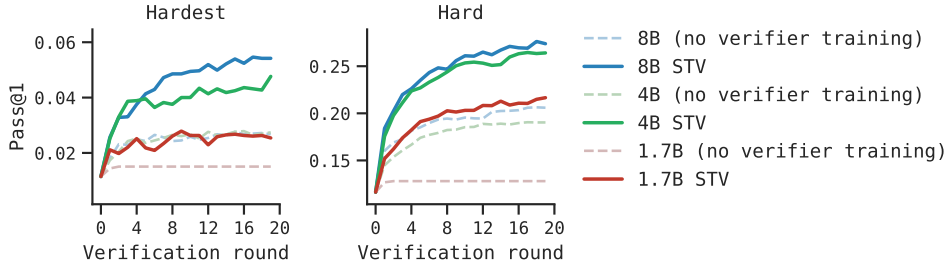



Figure 6: Training smaller verifiers to verify larger generators.

### 5.3 Weak-to-Strong Verification

Given STV's gains, can a smaller verifier match a larger one, which saves substantial test compute? Figure 6 compares verifiers of different scales on the 8B generator. Without training, both the 4B and 1.7B verifiers perform no better than 8B self-verification. After STV, the 4B verifier becomes competitive with 8B STV (final-round pass@1 of 26.4% vs 27.4% on Hard), and the 1.7B verifier matches the untrained 8B verifier (21.7% vs 20.6%).

### 5.4 Verifier-in-the-Loop (ViL) Generator Training

So far in our experiments, the generator either remained fixed or was trained with pure RLVR. The strength of STV opens a second door: training the generator inside the V-R loop, which allows the model to provide guidance to enrich learning signal. We show that just *one iteration* of ViL training (Section 4) already produces substantial training- and test-time self-improvement, producing the STV generator (Figure 3 bottom). Starting from the converged RL generator, we further train it with feedback from the frozen STV verifier; this is one round of training-time self-improvement.

Results are shown in Figure 7. Most strikingly, ViL training improves the generator *even at round 0 – before any verification at test time*. Pass@1 rises from 10.7% to 14.7% on Hardest (+37%) and from 36.7% to 47.7% on Hard (+30%), even though the generator was trained to improve *after* receiving feedback. This suggests that learning from diagnostic feedback can instill more general reasoning capabilities. In contrast, continuing standard RL for the same number of additional steps (*RLVR-only (longer)*) yields no gain, expected since the starting checkpoint is already converged under RLVR. With the STV verifier in the loop at test time, gains compound further: by the final round, pass@1 reaches 27.3% on Hardest vs 16.1% for *RLVR-only (longer)* ($p$ value $< 0.01$).

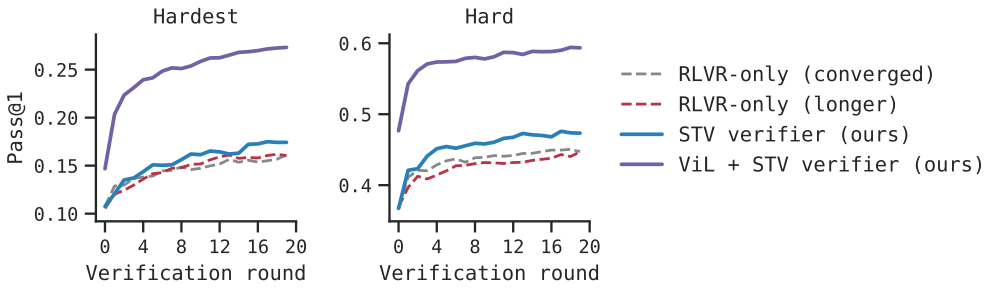



Figure 7: We train the generator to better incorporate verifier feedback, initialized with the already converged continual-trained generator, *RLVR-only (converged)*. Surprisingly, such training shows a large benefit even at round 0 without any verification at test time. In contrast, spending the same additional compute on standard RLVR, i.e., *RLVR-only (longer)*, has no gains.

**Ablation for the use of oracle.** Recall that STV training uses oracle solution, but what if we use the oracle elsewhere, or not at all? We first test an alternative for using oracle, *prefix-conditioning* (Zhang et al., 2025; Qu et al., 2026; Setlur et al., 2026), which trains the generator to continue from the first 50% of the reference solution. Table 2 shows that prefix-conditioning achieves a lower pass@1 than STV at both test-time round 0 (no verification) and round 20 (after V-R), indicating that gold prefixes do not substitute for diagnostic feedback. Second, we run ViL training with *self-verification* instead of a trained verifier: the generator both produces feedback for itself in the V-R loop and is trained on its own corrections, never seeing a reference solution anywhere in the pipeline. Round-0 pass@1 slightly underperforms STV (29.8% vs 31.2%), suggesting that ViL drives training-time self-improvement even without a stronger verifier. At round 20, however, STV pulls ahead by about 4 points (43.3% vs 39.4%),

Table 2: Oracle usage ablation.

| | Round 0 | Round 20 |
|---|---|---|
| *Not using oracle* | | |
| RLVR-only (converged) | 23.7 | 30.4 |
| ViL + self-verify (ours) | 29.8 | 39.4 |
| *Using oracle* | | |
| Prefix-conditioning | 29.1 | 38.5 |
| ViL + STV verifier (ours) | **31.2** | **43.3** |

which confirms that the trained verifier's better feedback compounds across rounds even though both pipelines start from comparable standalone capability. Notably, ViL with self-verify (without oracle) even outperforms prefix-conditioning which require using oracle.

### 5.5 Why STV Works

To understand why STV improves test-time self-improvement, we decompose the contribution of STV into more calibrated verdict and higher feedback quality.

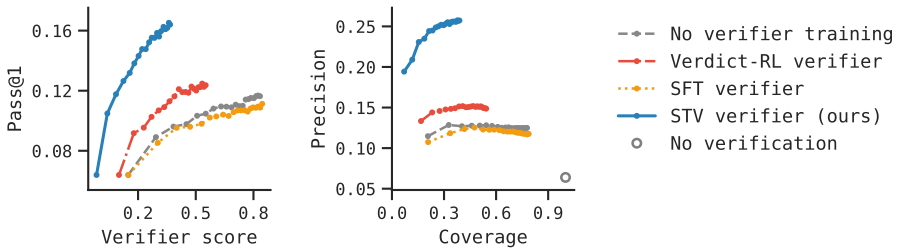



Figure 8: **Left**: Pass@1 vs verifier score over verification rounds (reward hacking diagnostic). **Right**: precision-coverage frontier of test-time scaling. Both panels pool Hardest and Hard problems.

**Calibrated test-time scaling.** We show that STV mitigates the miscalibrated verifier score problem. Without training, verifier scores increase over rounds while accuracy stagnates (Figure 8 left). STV verifiers are better calibrated: higher scores now mean genuinely higher accuracy. Figure 8 (right) shows the precision-coverage curves, where coverage is the fraction of solutions accepted by the round $t$ and precision is the fraction of acceptances that are correct. STV shows a $\sim$3–5$\times$ higher precision at the same level of coverage, and crucially, precision *increases* with coverage, which means that each test-time verification round makes accepted solutions more likely correct, unlike untrained verifiers where more rounds provide no reliability gain.

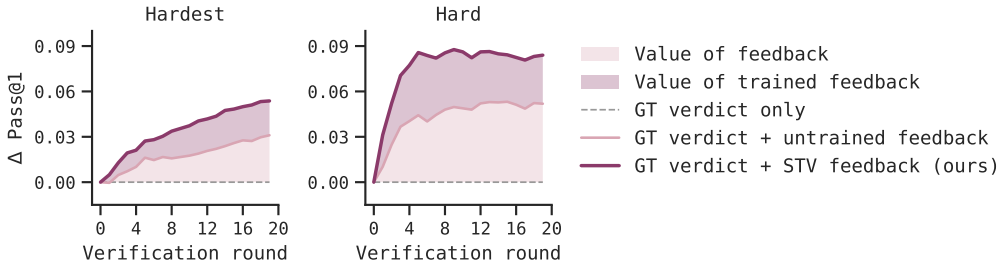


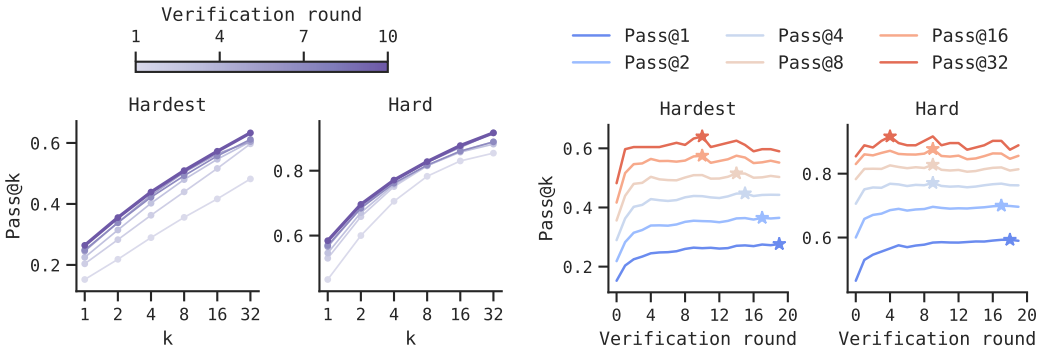

Figure 9: Isolating the contribution of trained feedback using ground-truth verdict.

**Value of trained feedback.** To isolate the contribution of feedback from verifier score accuracy, we replace the verifier's verdicts with ground-truth correctness labels while varying the feedback. Figure 9 plots $\Delta$pass@1 relative to a *GT verdict only* baseline, which uses ground-truth verdicts with uninformative feedback ("Your solution appears to be incorrect."). This baseline is shown as the dashed zero line. Adding untrained feedback from Qwen3-8B (*GT verdict + untrained feedback*) gives a $+5.2\%$ gain on Hard at the final round, which confirms that feedback provides value beyond pure verdict. STV feedback (*GT verdict + STV feedback*) adds a further $+3.2\%$ on top of it, showing that STV improves feedback quality independent of verdict accuracy.

### 5.6 How STV Shapes the Output Distribution

Beyond why the gains exist, we look at how V-R reshapes the output distribution. Our claim is that, unlike Best-of-$N$ (BoN) or resampling, V-R with STV improves the performance beyond sharpening the base distribution. This is because of the structured exploration enabled by actionable verifier feedback. We will verify this claim by exploring pass@$k$ and comparing with BoN directly.

**Verification does not suppress diversity.** A natural concern is whether the gains of verification come at the cost of diversity, i.e., verification could be simply sharpening to the best mode. With our strongest setting, *ViL + STV verifier*, this is not the case: pass@$k$ generally improves in the first 10 rounds of self-improvement (Figure 10a). However, there is a sweet spot for the number of verification rounds: pass@$k$ eventually peaks at a certain but non-zero round (Figure 10b). Interestingly, harder problems often require more verification rounds to reach their pass@$k$ peaks.




(a) Pass@$k$ vs $k$ per verification round.

(b) Pass@$k$ vs verification round per $k$.

Figure 10: Verification does not suppress diversity, but there is a sweet spot. **(a)** Pass@$k$ generally improves in the first 10 rounds. **(b)** At higher $k$, pass@$k$ eventually peaks at a certain but non-zero round; harder problems require more verification rounds to reach their peaks than easier ones.

**Refinement vs resampling.** To test the sharpen-vs-reshape distinction, we compare refinement to BoN at matched compute. Given $N$ independent samples, BoN selects one the verifier accepts (or a uniformly random one if none is accepted); $r$ rounds of refinement matches BoN at $N = r + 1$. In each setting, we also compare the STV verifier with the base verifier, which isolates the effect of verifier capability. Figure 11 shows that refinement dominates resampling for the base and STV generators – evidence that reshaping the distribution beats merely sharpening it. The lone exception is the continual-trained generator: it was trained solely for round-0 accuracy and never learned to use feedback, so reshaping isn't available to it and only sharpening remains. Notably, STV verifier helps *both* refinement and resampling in every case, which shows that STV verifier alone is a generally better verifier across different test-time compute scaling methods.

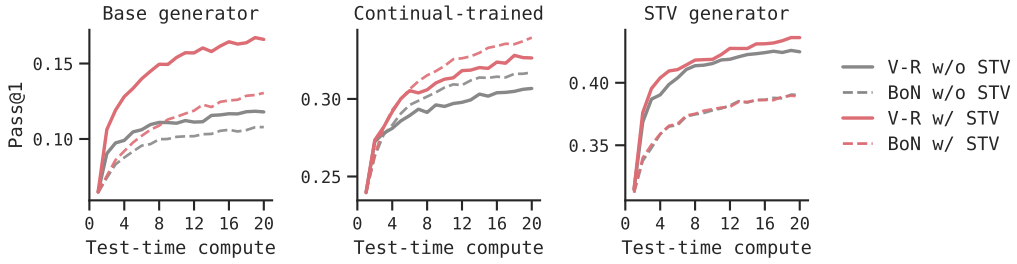



Figure 11: Refinement (V-R) vs resampling (BoN) at matched compute, for three generators considered in this paper. All panels pool Hardest and Hard problems.

## 6 Related Work

**Test-time compute scaling.** Scaling test-time compute (OpenAI, 2024; DeepSeek-AI et al., 2025; Snell et al., 2024; Wu et al., 2024) – through self-consistency (Wang et al., 2022), best-of-$N$ (Brown et al., 2024), extending unstructured chains of thought (Setlur et al., 2025), or multi-turn verification (Shao et al., 2025) – consistently improves reasoning performance. Our work contributes to this line by targeting structured multi-turn verification, and making its non-verifiable feedback trainable.

**Self-correction and self-improvement.** Early work on self-improvement conditions the generator on its own previous output without any explicit verification step; Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) are canonical examples. Huang et al. (2023) and Kamoi et al. (2024) show that such self-correction struggles on reasoning tasks: without external feedback, LLMs rarely fix their own errors. Follow-up work trains the generator to self-correct via SFT (Qu et al., 2024) or multi-turn RL (Kumar et al., 2024; Jiang et al., 2025). Broader self-improvement frameworks like STaR (Zelikman et al., 2022), ReST-EM (Singh et al., 2023), and V-STaR (Hosseini et al., 2024) generate training data from the model itself for continuous improvement. Our work bridges these two lines: running the V-R loop *during training* – not only at test time – yields *training-time self-improvement*, where the generator's standalone pass@1 climbs beyond the standard-RL ceiling.

**Verification-refinement pipelines.** V-R pipelines introduce an explicit verification step for structured exploration, which has led to recent breakthroughs in the hardest reasoning tasks such as IMO and frontier mathematics (Huang and Yang, 2025; Feng et al., 2026; Shao et al., 2025). A line of work trains the verifier explicitly: Shao et al. (2025) optimize verifiers by training a meta-verifier with RL based on human annotation of feedback quality; Kirchner et al. (2024) does not target V-R pipelines directly but trains a verifier to improve the legibility of generator outputs. RL from text feedback (Song et al., 2026) uses an external feedback provider during training and train generators to internalize and mimic the feedback. We focus on improving feedback without human annotations or supervision from a stronger model.

**Verifier and reward hacking.** Feedback loops are prone to in-context reward hacking. For example, an LM agent that tries to maximize Twitter engagement may make posts more controversial (Pan et al., 2024). Dang et al. (2026) show in the math setting that refinement can converge to plausible-but-wrong solutions, and propose to isolate potential errors from the context window. We show that STV addresses in-context reward hacking by improving verifier training.

**(Self-)distillation and privileged information.** Knowledge distillation (Hinton et al., 2015) transfers a teacher's behavior to a student by matching distributions. More recently, on-policy distillation (Agarwal et al., 2023; Gu et al., 2023) samples prefix tokens from the student so the distillation target matches deployment. A natural way to get a teacher without relying on a stronger model is to grant it *privileged information* such as reference solutions. When teacher and student share a base model, this becomes *self-distillation*, studied recently in various forms (Zhao et al., 2026; Hubotter et al., 2026; Shenfeld et al., 2026; Song et al., 2026; Penaloza et al., 2026; Yang et al., 2026). Among these, Yang et al. (2026) used reference solution to help truncate and intervene generated solutions to create finetuning data for generators. A closely related line splices partial reference solutions (Zhang et al., 2025; Qu et al., 2026; Setlur et al., 2026) or external feedback (Song et al., 2026) into the generator's context during RL. STV instantiates the privileged-information for a previously intractable target – non-verifiable feedback quality for training verifiers. Moreover, we show that ViL training transfers the gains to the generator, providing a new axis of conditioning (i.e., self-generated feedback) for generator training.

**Process reward models.** Our work is also conceptually related to process reward models (PRMs; Lightman et al., 2023). Unlike outcome reward models (ORMs; Cobbe et al., 2021) that only assign a scalar reward for the entire rollout, PRMs enable dense credit assignment along the trajectory. Our verifier shares this view that verification signal should not be sparse, but goes further by producing diagnostic feedback that actively coaches the generator. Shao et al. (2025) is the closest: it trains a verifier using a meta-verifier with human annotations. STV introduces a dense signal for feedback quality *without* such annotations.

## 7 Conclusion, Limitations, and Future Work

We proposed *self-trained verification* (STV), a way to train the verifier to catch flaws in plausible but incorrect solutions without human annotation for feedback quality. STV turns this challenge into a distillation target: when conditioned on the reference solution, the model becomes its own teacher for feedback quality. On math and scientific reasoning benchmarks, we show that STV substantially improves over baselines for test-time self-improvement. We then show that *verifier-in-the-loop*

*training* (ViL) – training the generator inside the V-R loop with STV verifier feedback – provides a new training-time self-improvement direction, which breaks through the standard RLVR plateau.

Whether STV's approach extends to other capabilities of similar structure, i.e., non-verifiable from scratch but easier when the model is given an appropriate reference, is an open question. Within the V-R setting itself, other self-supervision signals may also be rich – model uncertainty, multiple references, environment feedback, or training the verifier end-to-end against downstream generator improvement. Other open questions include scaling to larger models, generalization beyond math (code and open-ended reasoning), and the compute-optimal split between generator training, verifier training, and test-time rounds. More broadly, learning to verify – and developing scalable ways to do so without human-graded feedback – may be a key frontier for reasoning models that self-improve at both training and test time.

## Acknowledgments

We wish to thank Jacob Springer, Christina Baek, Jeremy Avigad, Sean Welleck, Marijn Heule, Yuda Song, Ziqian Zhong, Daniel Fried, and NeurIPS reviewers for their discussion and feedback. This research is partially supported by the DARPA expMath program through the DARPA CMO contract number HR0011262E028. We gratefully acknowledge support from Schmidt Sciences, NSF, Apple, Open Philanthropy, Google. We would like to thank CMU FLAME Cluster for providing compute.

## References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stańczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2023. URL https://api.semanticscholar.org/CorpusID:263610088.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher R'e, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *ArXiv*, abs/2407.21787, 2024. URL https://api.semanticscholar.org/CorpusID:271571035.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *ArXiv*, abs/2501.17161, 2025. URL https://api.semanticscholar.org/CorpusID:275932560.

Karl Cobbe, Vineet Kosaraju, Mo Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168, 2021. URL https://api.semanticscholar.org/CorpusID:239998651.

Xingyu Dang, Rohit Agarwal, Rodrigo de A. Porto, Anirudh Goyal, Liam H. Fowl, and Sanjeev Arora. Escaping the cognitive well: Efficient competition math with off-the-shelf models. *ArXiv*, abs/2602.16793, 2026. URL https://api.semanticscholar.org/CorpusID:285787286.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bing-Li Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dong-Li Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, JingChang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, M. Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, Ruiqi Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shao-Kang Wu, Tao

Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wen-Xia Yu, Wentao Zhang, Wangding Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyu Jin, Xi-Cheng Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yi Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yu-Jing Zou, Yujia He, Yunfan Xiong, Yu-Wei Luo, Yu mei You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yao Li, Yi Zheng, Yuchen Zhu, Yunxiang Ma, Ying Tang, Yukun Zha, Yuting Yan, Zehui Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhen guo Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zi-An Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645:633 – 638, 2025. URL https://api.semanticscholar.org/CorpusID:275789950.

Kehua Feng, Xinyi Shen, Weijie Wang, Zhuang Xiang, Yuqi Tang, Qiang Zhang, and Keyan Ding. Sciknoweval: Evaluating multi-level scientific knowledge of large language models. 2024. URL https://api.semanticscholar.org/CorpusID:281887235.

Tony Feng, Trieu H. Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung, Joonkyung Lee, Carlo Pagano, Sang hyun Kim, Federico Pasqualotto, Sergei Gukov, Jonathan Lee, Junsu Kim, Kaiying Hou, Golnaz Ghiasi, Y. C. Tay, Yaguang Li, Chenkai Kuang, Yuan Liu, Hanzhao Lin, Eva Liu, Nigamaa Nayakanti, Xiaomeng Yang, Heng tze Cheng, Demis Hassabis, Koray Kavukcuoglu, Quoc V. Le, and Thang Luong. Towards autonomous mathematics research. *ArXiv*, abs/2602.10177, 2026. URL https://api.semanticscholar.org/CorpusID:285470138.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: On-policy distillation of large language models. 2023. URL https://api.semanticscholar.org/CorpusID:259164722.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015. URL https://api.semanticscholar.org/CorpusID:7200347.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *ArXiv*, abs/2402.06457, 2024. URL https://api.semanticscholar.org/CorpusID:267617275.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *ArXiv*, abs/2310.01798, 2023. URL https://api.semanticscholar.org/CorpusID:263609132.

Yichen Huang and Ling Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline. 2025. URL https://api.semanticscholar.org/CorpusID:280045605.

Jonas Hubotter, Frederike Lubeck, Lejs Deen Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *ArXiv*, abs/2601.20802, 2026. URL https://api.semanticscholar.org/CorpusID:285102353.

Yuhua Jiang, Yuwen Xiong, Yufeng Yuan, Chao Xin, Wenyuan Xu, Yu Yue, Qianchuan Zhao, and Lin Yan. Pag: Multi-turn reinforced llm self-correction with policy as generative verifier. *ArXiv*, abs/2506.10406, 2025. URL https://api.semanticscholar.org/CorpusID:279318695.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. URL https://api.semanticscholar.org/CorpusID:270218742.

Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of llm outputs. *ArXiv*, abs/2407.13692, 2024. URL https://api.semanticscholar.org/CorpusID:271270365.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. *ArXiv*, abs/2409.12917, 2024. URL https://api.semanticscholar.org/CorpusID:272753259.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ArXiv*, abs/2305.20050, 2023. URL https://api.semanticscholar.org/CorpusID:258987659.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651, 2023. URL https://api.semanticscholar.org/CorpusID:257900871.

Kohsei Matsutani, Shota Takashiro, Gouki Minegishi, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. Rl squeezes, sft expands: A comparative study of reasoning llms. *ArXiv*, abs/2509.21128, 2025. URL https://api.semanticscholar.org/CorpusID:281525852.

OpenAI. Openai o1 system card. 2024. URL https://api.semanticscholar.org/CorpusID:285287340.

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. In *International Conference on Machine Learning*, 2024.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *ArXiv*, abs/2602.04942, 2026. URL https://api.semanticscholar.org/CorpusID:285304042.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *ArXiv*, abs/2407.18219, 2024. URL https://api.semanticscholar.org/CorpusID:271432135.

Yuxiao Qu, Amrith Rajagopal Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. Pope: Learning to reason on hard problems via privileged on-policy exploration. *ArXiv*, abs/2601.18779, 2026. URL https://api.semanticscholar.org/CorpusID:285050883.

Amrith Rajagopal Setlur, Matthew Y. R. Yang, Charlie Victor Snell, Jeremy Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. e3: Learning to explore enables extrapolation of test-time compute for llms. *ArXiv*, abs/2506.09026, 2025. URL https://api.semanticscholar.org/CorpusID:279260520.

Amrith Rajagopal Setlur, Zijian Wang, Andrew Cohen, Paria Rashidinejad, and Sang Michael Xie. Reuse your flops: Scaling rl on hard problems by conditioning on very off-policy prefixes. *ArXiv*, abs/2601.18795, 2026. URL https://api.semanticscholar.org/CorpusID:285050190.

Zhihong Shao, Yu-Wei Luo, Chengda Lu, Zehui Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *ArXiv*, abs/2511.22570, 2025. URL https://api.semanticscholar.org/CorpusID:283438764.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *ArXiv*, abs/2601.19897, 2026. URL https://api.semanticscholar.org/CorpusID:285071839.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems 36*, 2023. URL https://api.semanticscholar.org/CorpusID:258833055.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin

Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Ethan Dyer, Behnam Neyshabur, Jascha Narain Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Trans. Mach. Learn. Res.*, 2024, 2023. URL https://api.semanticscholar.org/CorpusID:266163375.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *ArXiv*, abs/2408.03314, 2024. URL https://api.semanticscholar.org/CorpusID:271719990.

Yuda Song, Lili Chen, Fahim Tajwar, Rémi Munos, Deepak Pathak, J. Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback. *ArXiv*, abs/2602.02482, 2026. URL https://api.semanticscholar.org/CorpusID:285269383.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *ArXiv*, abs/2402.08115, 2024. URL https://api.semanticscholar.org/CorpusID:267637077.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022. URL https://api.semanticscholar.org/CorpusID:247595263.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *International Conference on Learning Representations*, 2024. URL https://api.semanticscholar.org/CorpusID:271601023.

Matthew Y. R. Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Rajagopal Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning. *ArXiv*, abs/2601.14209, 2026. URL https://api.semanticscholar.org/CorpusID:284910872.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Honglin Yu, Weinan Dai, Yuxuan Song, Xiang Wei, Haodong Zhou, Jingjing Liu, Wei Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yong-Xu Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale. *ArXiv*, abs/2503.14476, 2025.

E. Zelikman, Yuhuai Wu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. 2022. URL https://api.semanticscholar.org/CorpusID:247762790.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. *ArXiv*, abs/2506.17211, 2025. URL https://api.semanticscholar.org/CorpusID:279464619.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *ArXiv*, abs/2601.18734, 2026. URL https://api.semanticscholar.org/CorpusID:285050713.

## A Compute Resources

All experiments are done on one node of 8 H100 GPUs.

## B Societal Impact

Our work mainly focus on math and scientific reasoning, so we see no immediate risks to society beyond the general capability enhancement. We think our method can be applied to frontier research agents that do experiments and reason about self-failures, so will potentially have a positive impact in science domains and benefit society indirectly.

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: **[TODO]**

   Justification: **[TODO]**

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

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

    Answer: **[TODO]**

    Justification: **[TODO]**

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: **[TODO]**

    Justification: **[TODO]**

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.