SOD: STEP-WISE ON-POLICY DISTILLATION FOR SMALL LANGUAGE MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-integrated reasoning (TIR) is difficult to scale to small language models due to instability in long-horizon tool interactions and limited model capacity. While reinforcement learning methods like group relative policy optimization provide only sparse outcome-level rewards. Recently, on-policy distillation (OPD) has gained popularity by supplying dense token-level supervision from a teacher on student-generated trajectories. However, our experiments indicate that applying OPD to TIR leads to a critical failure mode: erroneous tool calls tend to cascade across subsequent reasoning steps, progressively amplifying student-teacher divergence and rendering the teacher's token-level supervision increasingly unreliable. To address this, we propose SOD, a step-wise on-policy distillation framework for small language model agents, which adaptively reweights distillation strength at each step based on step-level divergence. Therefore, SOD can attenuate misleading teacher signals in high-divergence regions while preserving dense guidance in well-aligned states. Experiments on math, science, and code benchmarks show that SOD improves up to 13.50% over the second-best baseline. Notably, our 0.6B student reaches 26.13% on AIME 2025 in this tool-integrated setting, showing effective transfer of agentic reasoning to lightweight models. Our code is available at `https://anonymous.4open.science/r/SOD-CA86`.

1 INTRODUCTION

Agentic capabilities have substantially expanded the applicability of large language models (LLMs) (Xi et al., 2023; Kang et al., 2025), enabling them to solve complex real-world tasks (Yao et al., 2023; Schick et al., 2023; Singh et al., 2025). However, such capabilities are typically realized by large-scale models, which incurs substantial inference cost and deployment overhead (Singh et al., 2025; Chenglin et al., 2024; Fan et al., 2026). In latency- and resource-sensitive scenarios, transferring agentic capabilities to small language models (SLMs) deployable on-device is therefore of practical importance (Xu et al., 2024a;b; Qiu et al., 2025; Yao et al., 2026; Li et al., 2025a). Yet enabling SLMs to acquire stable and effective tool-integrated reasoning (TIR) remains a major challenge (Qian et al., 2025; Rainone et al., 2025; Xue et al., 2025; Liu et al., 2025).

Existing post-training methods for enhancing TIR are largely based on reinforcement learning (RL) (Li et al., 2025b; Jin et al., 2025; Song et al., 2025), particularly policy optimization algorithms such as GRPO (Shao et al., 2024). However, in SLM-based TIR settings, RL often suffers from unstable optimization (Qian et al., 2025; Rainone et al., 2025). TIR involves long-horizon trajectories (Li et al., 2025b), multi-step decisions (Jin et al., 2025), and interactions with external tools (Singh et al., 2025), whereas RL provides only sparse outcome-level rewards (DeepSeek-AI, 2025). For small models with limited capacity, such sparse supervision can exacerbate exploration failure and leave the policy in a cold-start regime (Yang et al., 2026b).

Recently, on-policy distillation (OPD) has emerged as a promising paradigm for post-training (Agarwal et al., 2024; Gu et al., 2024; Jin et al., 2026; Ko et al., 2026; Wu et al., 2026). Unlike RL methods that rely on sparse, trajectory-level rewards (DeepSeek-AI, 2025), OPD provides dense token-level supervision (Gu et al., 2024) on trajectories from the student's own policy (Agarwal et al., 2024), easing the credit assignment difficulty of sparse rewards (Team, 2025; Li et al., 2026b) while improving sample efficiency (Xu et al., 2026a) and training stability (Team, 2025).

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.

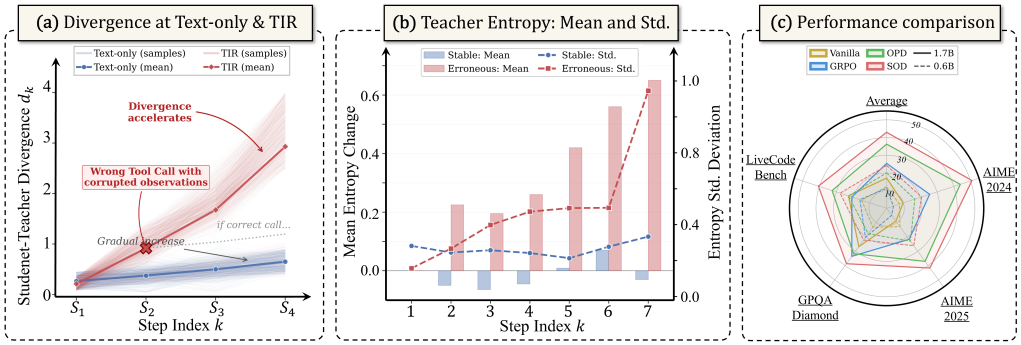


Figure 1: **The motivation of SOD**. (a) Student-teacher divergence d_k across reasoning steps, sampled from 800 trajectories: in TIR, erroneous tool calls cause divergence to accelerate sharply, unlike the gradual drift in text-only reasoning. (b) Teacher entropy statistics over 800 sampled trajectories: on erroneous trajectories, both the mean entropy change (bars) and the standard deviation (dashed lines) grow rapidly at later steps, indicating increasingly unreliable teacher supervision. Please refer to Appendix I for detailed cases. (c) Radar chart comparing methods on four benchmarks.

However, our experiments show that directly transferring OPD to SLM-based TIR can lead to severe training instability (Li et al., 2026a; Yang et al., 2026a; Bousselham et al., 2025; Wang et al., 2026c). We attribute this failure to a fundamental difference between how student trajectories deviate from the teacher distribution in TIR and in standard text-based reasoning (Li et al., 2026b; Fu et al., 2026; Song & Zheng, 2026; Bousselham et al., 2025). As illustrated in Figure 1(a), in text-only reasoning, even when the student departs from the teacher, subsequent states still evolve continuously along the generated context, and the resulting distribution shift is progressive (Ross et al., 2011; Agarwal et al., 2024). In contrast, TIR introduces discontinuous state transitions through tool interactions (Li et al., 2025b). A single erroneous tool call can inject an incorrect observation into the context, causing subsequent reasoning to unfold from a corrupted state (Qian et al., 2025). This is amplified because teachers are often highly accurate in tool use, whereas small models have limited capacity (Gudibande et al., 2023; Rainone et al., 2025). During early training, small models may accumulate erroneous tool calls, causing their state distribution to drift rapidly away from the teacher (Li et al., 2026b; Wang et al., 2026b). In such out-of-distribution states, teacher supervision may become unreliable or even misleading (Jang et al., 2026; Bousselham et al., 2025; Yang et al., 2026b; Xu et al., 2026b; Li et al., 2026a) as illustrated in Figure 1(b), and the enlarged discrepancy can induce unstable gradients and even training collapse (Li et al., 2026b; Team, 2025).

Based on this observation, we propose **SOD**, a **S**tep-wise **O**n-policy **D**istillation framework for small language model agents. The core idea is to adaptively adjust the distillation strength at each reasoning step according to the divergence between student and teacher. When the student remains well aligned, SOD preserves dense supervision to fully exploit teacher guidance; when the deviation grows large, it attenuates the distillation signal for that step, reducing the influence of misleading supervision under out-of-distribution states. Experimental results (Figure 1(c)) show that this simple yet effective reweighting lets lightweight agents acquire agentic TIR capabilities more efficiently. Extensive experiments on math, science, and code benchmarks show that SOD outperforms the second-best baseline by up to **13.50%**. Notably, under the python-based TIR setting, our 0.6B-scale student distilled from a GRPO-trained Qwen3-4B teacher reaches **26.13%** average@32 on AIME 2025 and outperforms every 0.6B baseline in Table 1 evaluated under the same setup.

2 Preliminaries

2.1 Multi-turn Tool-Integrated Reasoning

We consider the post-training of a small language model (SLM) for multi-turn tool-integrated reasoning (TIR). Given an input x, the model interacts with an external environment over multiple reasoning steps. At each step k, the model generates a response y_k, which may include natural-

language reasoning, a tool invocation, or a final answer. If a tool is invoked, the environment returns an observation o_k, which is appended to the context and conditions subsequent generations.

A trajectory is defined as:

$$\tau = (x, y_1, o_1, \ldots, y_K, o_K, y_{K+1}), \tag{1}$$

where y_{K+1} denotes the final response. The policy π_θ generates only model tokens, while observations $\{o_k\}$ are provided by the environment. Let y_t denote a generated token and $y_{<t}$ its prefix, which may include both model outputs and tool observations.

2.2 Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is a reinforcement learning algorithm that updates the policy using relative rewards within a group of sampled trajectories. We assume access to an outcome-level reward function $r(\tau)$ defined on complete trajectories. For each input x, a group of trajectories $\{\tau_i\}_{i=1}^{G}$ is sampled from the old policy $\pi_{\theta_{\text{old}}}$, each receiving reward $r_i = r(\tau_i)$.

The group-relative advantage is computed as:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\})}{\text{std}(\{r_j\}) + \epsilon_A}. \tag{2}$$

Let $\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|y_{i,<t})}$ denote the token-level importance ratio. The objective is defined as:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{T}_i|}\sum_{t\in\mathcal{T}_i}\min\left(\rho_{i,t}(\theta)\hat{A}_i, \text{clip}(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\right)\right], \tag{3}$$

where $\mathcal{T}_i$ denotes model-generated token positions. This objective provides an on-policy learning signal based on relative trajectory performance, but supplies only sparse outcome-level rewards.

2.3 On-policy Distillation

On-policy distillation (OPD) (Agarwal et al., 2024; Lu & Lab, 2025) is a post-training paradigm that provides dense token-level supervision on student-generated trajectories by aligning the student policy with a teacher distribution. Given a trajectory $y \sim \pi_\theta$, the OPD objective is defined as:

$$\mathcal{L}_{\text{OPD}} = \mathbb{E}\left[\sum_{t\in\mathcal{T}_i}\left(\log \pi_\theta(y_t \mid y_{<t}) - \log \pi_{\text{teacher}}(y_t \mid y_{<t})\right)\right], \tag{4}$$

where π_θ is the student policy, π_{teacher} is the teacher model, and $\mathcal{T}_i$ denotes the set of model-generated token positions in the sampled trajectory. This objective corresponds to a sampled estimator of the reverse KL divergence between the student and teacher policies on student-visited states.

3 Methodology

In this section, we introduce SOD. The framework of SOD is displayed in Figure 2. We first analyze why on-policy distillation (OPD) can become unreliable under tool-induced state drift. We then introduce a step-level divergence score as an observable proxy for the reliability of teacher supervision, and finally formulate a step-wise reweighted OPD objective.

3.1 Failure of On-policy Distillation under Tool-Induced State Drift

The OPD objective assumes that teacher supervision remains reliable across all student-visited states (Fu et al., 2026; Li et al., 2026b; Bousselham et al., 2025). This holds when distribution drift is gradual, but is severely violated in TIR by the discontinuous state transitions that tool observations introduce. We make this failure precise through two informal observations that isolate the mechanism we design against, with the derivations and their assumptions in Appendix G.

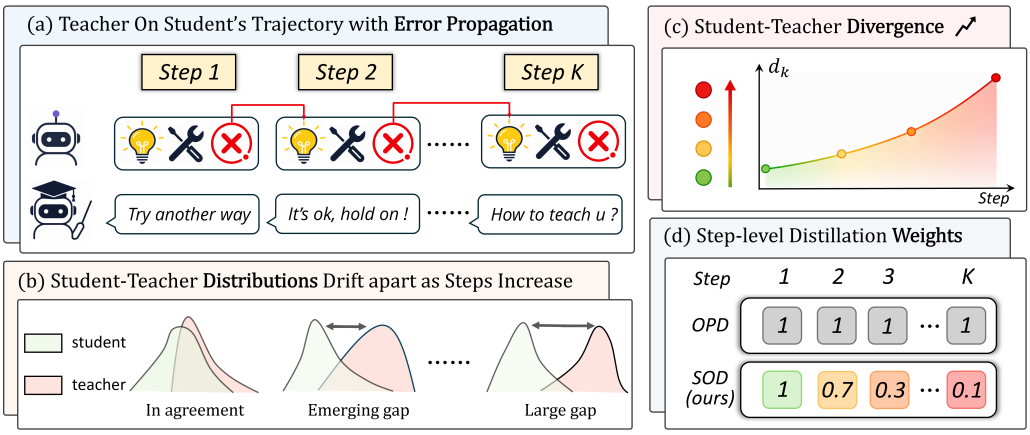


Figure 2: **The overview of SOD**. (a) The student generates multi-step trajectories where erroneous tool calls propagate across steps, degrading teacher supervision reliability. (b) Student-teacher distributions drift apart as errors accumulate. (c) Step-level divergence d_k quantifies this drift. (d) SOD adaptively attenuates distillation weights in high-divergence steps, unlike vanilla OPD which applies uniform weights. Please refer to Section 4.5 for more characteristics of SOD.

We define the step-level mismatch as:

$$\Delta_k = \frac{1}{|\mathcal{I}_k|} \sum_{t \in \mathcal{I}_k} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid y_{<t}) \,\big\|\, \pi_{\mathrm{teacher}}(\cdot \mid y_{<t})\big), \tag{5}$$

and write TV_k for the corresponding step-level total variation distance between the same two conditional distributions, which is the quantity our first observation is stated in.

Observation 1 (Discontinuous drift amplification)**.** Assume that appending one student-sampled token moves the next-token distribution by at most η in total variation, and that appending one token of an erroneous tool observation moves it by η_{tool} on average, with $\eta_{\mathrm{tool}} > \eta$. In text-only reasoning, consecutive steps then differ by $O(\eta)$ in total variation. In TIR, a single erroneous observation of length m admits a shift of order $m \cdot \eta_{\mathrm{tool}}$, and under j consecutive tool errors the shifts accumulate as $\sum_{i=0}^{j-1} m_i \cdot \eta_{\mathrm{tool}}^{(i)}$. Because an already corrupted prefix is progressively less represented in the teacher's training distribution, we expect $\eta_{\mathrm{tool}}^{(i+1)} \geq \eta_{\mathrm{tool}}^{(i)}$, so the drift grows faster than linearly in the number of failures.

Observation 2 (Gradient SNR degradation under low overlap)**.** Define the teacher-supported region $S_t^\epsilon = \{v \in \mathcal{V} : \pi_{\mathrm{teacher}}(v \mid y_{<t}) \geq \epsilon\}$ with $\epsilon < 1/|\mathcal{V}|$, and the overlap $\rho_t = \sum_{v \in S_t^\epsilon} \pi_\theta(v \mid y_{<t})$. Suppose $\rho_t \leq \rho$, the expected gradient $\mathbb{E}[g_t]$ has bounded norm, and the per-token score norms satisfy $\min_v \|\nabla_\theta \log \pi_\theta(v \mid y_{<t})\|^2 > 0$. Then the second moment of the OPD loss obeys $\mathbb{E}[\ell_t^2] \geq (1-\rho)\,\log^2(1/\epsilon)$ up to the entropy correction in Appendix G, and the gradient estimator's signal-to-noise ratio $\mathrm{SNR}(g_t)$ vanishes as $\rho_t \to 0$. This describes one mechanism by which OPD can fail in severely mismatched states, not a universal bound on its variance.

Together, these observations describe a failure cascade specific to TIR: consecutive erroneous tool calls drive compounding drift amplification (Obs. 1), progressively pushing the student away from the teacher distribution into low-overlap states where the OPD gradient becomes dominated by high-variance, uninformative contributions (Obs. 2). This is empirically confirmed in Figure 1: divergence accelerates sharply as tool errors accumulate (a), and teacher entropy becomes elevated and unstable in subsequent steps (b). Since OPD aggregates losses uniformly across steps, it systematically overweights these corrupted signals, motivating our step-wise reweighting mechanism. We show in Appendix G.4 that whenever the step-level divergence is non-decreasing along a trajectory and the weight cap is inactive, our reweighting attenuates the weighted second moment by a factor of $(d_1/d_k)^2$, which keeps the aggregate gradient dominated by the well-aligned early steps precisely where OPD is most exposed to low-overlap noise. The complementary regime in which the student re-aligns is outside the scope of that analysis and is supported empirically instead (Section 4.5).

3.2 Student-teacher Divergence and Step-wise Reweighting

We partition a generated trajectory into $K+1$ reasoning steps, where each step corresponds to a model response between two tool observations, or the final answer step. Let $\mathcal{I}_k$ denote the set of token positions belonging to the k-th model-generated step. Tool observation tokens are excluded because they are produced by the external environment rather than by the policy.

We define a step-level divergence score to quantify student–teacher divergence at step k:

$$d_k = \frac{1}{|\mathcal{I}_k|} \sum_{t \in \mathcal{I}_k} \left| \log \pi_\theta(y_t \mid y_{<t}) - \log \pi_{\text{teacher}}(y_t \mid y_{<t}) \right|. \tag{6}$$

The score d_k serves as a lightweight indicator of the local reliability of teacher supervision. A small d_k suggests that the student remains well aligned with the teacher distribution, while a large d_k indicates substantial mismatch, often caused by tool-induced state drift or corrupted observations.

Based on these divergence scores, we adaptively reweight the strength of distillation across steps. We initialize the first step with full distillation strength, *i.e.,* $w_1 = 1$. For subsequent steps, the weight is:

$$w_k = \min\left(\prod_{u=1}^{k-1} \frac{d_u + \epsilon}{d_{u+1} + \epsilon}, 1 + \delta \right), \quad k \geq 2, \tag{7}$$

where ϵ is a small constant for stability and δ controls the maximum amplification of the distillation. Since the reweighting in Eq. (7) depends only on the *ratios* between consecutive divergence scores rather than their absolute values, any monotone proxy of Δ_k suffices. We show in Appendix G.5 that d_k is monotonically consistent with Δ_k and computable at zero marginal cost from the OPD forward pass, and Appendix F.6 reports the wall-clock time and peak memory this amounts to in practice. We therefore adopt d_k in place of Δ_k in our implementation.

This weighting rule captures the evolution of student-teacher divergence along the trajectory. When the divergence increases, *i.e.,* $d_{u+1} > d_u$, indicating that the student is drifting away from the teacher distribution (often due to tool-induced state shift), the corresponding ratio becomes smaller than one, leading to attenuation of the distillation signal in high-mismatch regions. Conversely, the student may partially move back toward the teacher-supported region in later steps, effectively correcting earlier deviations. We refer to this phenomenon as *recovery from earlier errors*. In such cases, when $d_{u+1} < d_u$, the reweighting mechanism allows the distillation strength to increase accordingly, restoring informative guidance. To ensure stable optimization, the weight is upper-bounded by $1 + \delta$.

3.3 Training Objective

For a student-generated trajectory, the sampled-token OPD term at token t is:

$$\ell_{\text{OPD}}(y_t) = \log \pi_\theta(y_t \mid y_{<t}) - \log \pi_{\text{teacher}}(y_t \mid y_{<t}). \tag{8}$$

Instead of applying this term uniformly at step level, we reweight all tokens in step k by the corresponding reliability weight w_k, leading to the step-wise OPD objective:

$$\mathcal{L}_{\text{OPD}}^{\text{step}} = \mathbb{E}_{y \sim \pi_\theta} \left[\sum_{k=1}^{K+1} w_k \sum_{t \in \mathcal{I}_k} \left(\log \pi_\theta(y_t \mid y_{<t}) - \log \pi_{\text{teacher}}(y_t \mid y_{<t}) \right) \right]. \tag{9}$$

The final training objective is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{GRPO}} + \mathcal{L}_{\text{OPD}}^{\text{step}}. \tag{10}$$

Here, $\mathcal{L}_{\text{GRPO}}$ provides sparse outcome-level rewards to drive trajectory exploration, while $\mathcal{L}_{\text{OPD}}^{\text{step}}$ supplies dense token-level guidance whose strength is reweighted by the student-teacher divergence, jointly enabling stable learning under tool-induced state drift.

Table 1: Performance comparison of Qwen3 teacher and student across 4 benchmarks. TIP and SCOPE are concurrent adaptive on-policy distillation methods, contrasted with ours in Appendix E. Best results are in **bold** and second-best underlined. We report average@32 over 5 runs.

Params	Method	Math		Science	Code	Average
		AIME 2024	AIME 2025	GPQA	LiveCodeBench	
Teacher Models from the Qwen3 Series						
4B	GRPO	67.60 ±1.34	60.42 ±1.47	55.19 ±0.81	63.13 ±0.93	61.59
Distilled Student Models from the Qwen3 Series						
0.6B	Vanilla	7.71 ±0.33	12.81 ±0.38	13.24 ±0.24	14.89 ±0.29	12.16
	SFT	5.67 ±0.26	5.42 ±0.22	15.20 ±0.30	9.61 ±0.27	8.97
	GRPO	4.06 ±0.37	4.90 ±0.31	20.38 ±0.47	15.95 ±0.51	11.32
	OPD	16.82 ±0.81	22.95 ±0.97	17.76 ±0.41	22.65 ±0.75	20.04
	OPSD_{gt}	12.63 ±0.58	17.04 ±0.64	17.32 ±0.35	16.73 ±0.44	15.93
	$\text{OPSD}_{\text{hint}}$	9.77 ±0.43	14.12 ±0.48	15.98 ±0.30	12.65 ±0.37	13.13
	TIP	17.55 ±0.72	23.48 ±1.11	18.62 ±0.49	23.46 ±0.88	20.78
	SCOPE	18.44 ±0.96	23.39 ±0.81	19.41 ±0.63	24.11 ±0.70	21.34
	SOD	**20.84** ±0.90	**26.13** ±1.07	**22.19** ±0.59	**27.72** ±0.83	**24.22**
1.7B	Vanilla	9.90 ±0.38	8.96 ±0.33	26.80 ±0.36	22.73 ±0.41	17.10
	SFT	26.77 ±0.66	22.40 ±0.72	29.85 ±0.51	24.63 ±0.57	25.91
	GRPO	25.63 ±1.03	21.67 ±1.12	33.55 ±0.76	20.70 ±0.87	25.39
	OPD	43.86 ±1.23	37.04 ±1.31	31.73 ±0.55	32.45 ±0.93	36.27
	OPSD_{gt}	33.85 ±0.88	24.69 ±0.94	35.02 ±0.67	22.73 ±0.61	29.07
	$\text{OPSD}_{\text{hint}}$	34.42 ±0.76	21.43 ±0.83	33.46 ±0.59	23.12 ±0.52	28.11
	TIP	45.34 ±0.84	38.62 ±1.46	33.91 ±0.64	34.33 ±0.81	38.05
	SCOPE	47.50 ±1.37	39.60 ±0.88	35.80 ±0.72	34.70 ±1.09	39.40
	SOD	**50.83** ±1.15	**41.72** ±1.24	**38.72** ±0.73	**40.63** ±0.91	**42.98**

4 Experiment

4.1 Experimental Setup

Datasets & Benchmarks. Following Yu et al. (2025b), our training data comprises a 3k SFT corpus of multi-turn reasoning trajectories curated from s1-1k (Muennighoff et al., 2025), LeetCode, and ReTool (Feng et al., 2025), along with a 30k RL dataset covering mathematical reasoning from DAPO-Math (Yu et al., 2025a), math and coding from Skywork-or1 (He et al., 2025), and scientific problem solving from MegaScience (Fan et al., 2025). We evaluate on 4 benchmarks: AIME 2024/2025, GPQA-Diamond (Rein et al., 2024), and LiveCodeBench (Jain et al.), detailed with the data in appendices D.1 and D.2.

Evaluation Setups. We report average@32 with the decoding configuration, the teacher and student checkpoints, and the training hyperparameters given in Appendix D.4.

Baselines & Implementation Details. For the baselines, we evaluate SOD against a set of supervised, RL, and distillation baselines. For each student, we compare SOD with the following baselines: (1) Vanilla. (2) SFT. (3) GRPO (Shao et al., 2024). (4) OPD (Agarwal et al., 2024). (5) OPSD_{gt} (Zhao et al., 2026) (On-policy self-distillation with groundtruth). (6) $\text{OPSD}_{\text{hint}}$ (Zhao et al., 2026) (On-policy self-distillation with hints). (7) TIP (Xu et al., 2026b) and (8) SCOPE (Zheng et al., 2026), two concurrent methods that also reweight the distillation signal instead of applying it uniformly, the former by selecting informative tokens and the latter by routing whole trajectories on verifier correctness; we contrast their weighting mechanisms with ours in Appendix E. Further details of the baselines and of our implementation are in appendices D.3 and D.4.

4.2 Overall Performance

We compare SOD against eight baselines on both 0.6B and 1.7B student models. The results are summarized in Table 1, from which we draw the following key observations.

- **Obs 1: SOD consistently outperforms all baselines across every benchmark.** On both model scales, SOD achieves the highest scores on all four tasks, surpassing the strongest baseline SCOPE

Table 2: **Ablation study on key components of SOD.** Results are evaluated on 1.7B student model.

Variants	Math		Science	Code	Average
	AIME 2024	AIME 2025	GPQA	LiveCodeBench	
Step-wise Reweighting					
(1.1) *w/* Uniform Weighting	41.68 ±0.95	35.58 ±1.01	30.12 ±0.59	31.43 ±0.68	34.70
(1.2) *w/* Heuristic Weighting	44.96 ±0.98	38.75 ±1.04	31.14 ±0.64	33.71 ±0.72	37.14
(1.3) *w/* Mask After Wrong	39.11 ±1.08	31.59 ±1.14	26.56 ±0.67	30.12 ±0.80	31.85
(1.4) *w/o* Weight Clipping	45.78 ±1.16	37.93 ±1.21	33.57 ±0.74	35.12 ±0.79	38.10
Training Objective					
(2.1) *w/o* GRPO	48.87 ±0.93	39.73 ±1.02	35.89 ±0.62	38.62 ±0.71	40.78
(2.2) *w/o* Step-wise OPD	25.63 ±1.03	21.67 ±1.12	33.55 ±0.76	20.70 ±0.87	25.39
SOD	**50.83** ±1.15	**41.72** ±1.24	**38.72** ±0.73	**40.63** ±0.91	**42.98**

by **13.50%** (0.6B) and **9.09%** (1.7B) in relative average improvement, and uniform-weight OPD by 20.86% and 18.50%. Notably, SFT and GRPO both fail to outperform the Vanilla baseline at the 0.6B scale, indicating that sparse outcome-level rewards and static demonstrations are insufficient for guiding small models in TIR. In contrast, our step-wise reweighting mechanism provides dense yet reliability-modulated supervision, effectively preventing misleading gradients from corrupted tool-call states while preserving informative teacher guidance in well-aligned regions.

- **Obs 2: Adaptive weighting helps, but weighting units designed for text-only generation leave most of the gain unrealized.** TIP and SCOPE both improve over uniform-weight OPD at both scales, confirming that modulating the distillation signal is the right direction. Both nonetheless trail SOD by a clear margin, and the reason is that their weighting units were designed for single-turn text generation: TIP scores each token in isolation and SCOPE commits to one decision per trajectory, so neither can express the division that a tool observation imposes on a rollout, where steps taken before an erroneous call remain reliable and those taken after are generated from a corrupted state. We detail this comparison in Appendix E.
- **Obs 3: The improvements generalize across different model scales, demonstrating the scalability of SOD.** Our 1.7B student recovers **69.8%** of the 4B teacher's performance, compared to 64.0% for SCOPE and 58.9% for OPD, showing that SOD substantially improves distillation efficiency. Furthermore, even our 0.6B model surpasses several 1.7B baselines, suggesting that step-wise reweighting can partially compensate for the capacity gap between model scales.

4.3 Ablation Study

To evaluate each component in SOD, we develop six variants from two perspectives on the Qwen3-1.7B student. For *Step-wise Reweighting*, we consider: (1.1) *w/ Uniform Weighting*, which fixes $w_k = 1$ for all steps; (1.2) *w/ Heuristic Weighting*, which applies a fixed exponential decay $w_k = \gamma^{k-k_{\text{err}}}$ starting from the first erroneous tool call at step k_{err}, with γ fixed as 0.9; (1.3) *w/ Mask After Wrong*, which zeros out the OPD signal for all steps after the first tool-call error; and (1.4) *w/o Weight Clipping*, which removes the upper-bound clipping δ in Eq. (7). For *Training Objective*, we ablate the two terms in Eq. (10): (2.1) *w/o GRPO*, which removes $\mathcal{L}_{\text{GRPO}}$; and (2.2) *w/o Step-wise OPD*, which removes $\mathcal{L}_{\text{OPD}}^{\text{step}}$. The results are reported in Table 2, from which we observe:

- **Obs 4: The adaptive step-wise reweighting is critical, and neither static nor heuristic alternatives can substitute it.** Uniform weighting (variant 1.1) drops to 34.70%, so equal treatment exposes training to misleading supervision. Heuristic decay (1.2) recovers part of this (37.14%) but a fixed schedule cannot follow non-monotonic divergence, and hard masking (1.3) is worst (31.85%) because discarding everything after one error also discards informative steps. Removing the clipping (1.4) yields 38.10%, so unbounded amplification destabilises training. Each variant rewrites the weight of every step, so they show that the weighting profile matters without isolating the attenuated steps; Appendix F.5 overwrites those steps alone.
- **Obs 5: Both the GRPO and step-wise OPD components are indispensable, serving complementary roles.** Removing GRPO (variant 2.1) reduces the average to 40.78%, showing that outcome-level rewards remain valuable for steering exploration with dense distillation. Removing step-wise OPD (variant 2.2) causes a far more severe drop to 25.39%, which confirms that sparse rewards alone are insufficient for stable TIR learning in small models. The pronounced asymmetry

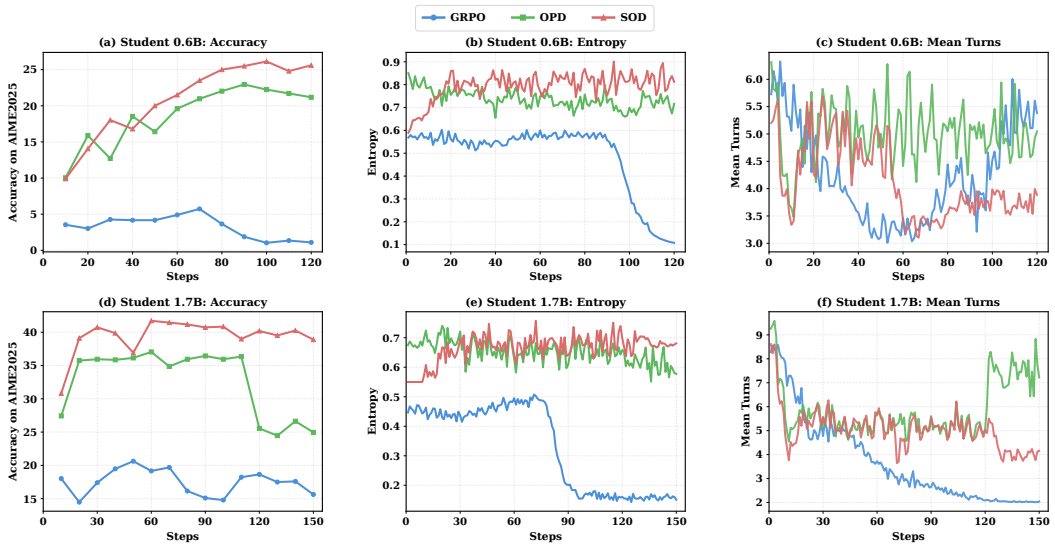


Figure 3: **Training dynamics across methods on 0.6B and 1.7B student models**. We track accuracy on AIME2025 (left), policy entropy (middle), and mean tool-calling turns (right) throughout training.

reveals their complementary nature: step-wise OPD provides the primary fine-grained guidance, while GRPO broadens trajectory-level exploration beyond the teacher distribution.

4.4 Dynamic Training Analysis

To understand how SOD shapes the learning dynamics of agentic reasoning, we monitor three key metrics throughout training: task accuracy, policy entropy, and mean tool-calling turns. As shown in Figure 3, we highlight the following observations:

- **Obs 6: SOD consistently outperforms GRPO by a large margin.** As shown in Figure 3(a)(d), SOD achieves significantly higher accuracy than GRPO across both model scales. This gap is driven by GRPO's severe entropy collapse (Figure 3(b)(e)), where the policy loses exploration ability and degenerates into repetitive outputs, ultimately abandoning multi-step tool interactions entirely (Figure 3(c)(f)), a fatal failure mode for TIR tasks.
- **Obs 7: SOD achieves more stable training than OPD.** While OPD maintains comparable entropy levels, its accuracy exhibits notable instability, particularly on 1.7B where performance peaks early then degrades significantly (Figure 3(d)), suggesting that uniform distillation can destabilize later training. In contrast, SOD sustains steady improvement without degradation. Additionally, SOD uses fewer tool-calling turns than OPD (Figure 3(c)(f)), indicating more efficient reasoning with fewer erroneous intermediate steps.

4.5 Three Distillation Patterns of SOD

During training, SOD exhibits three characteristic distillation patterns, as illustrated in Figure 4:

(1) Stable Pattern. The student remains closely aligned with the teacher, *i.e.,* the step-level divergence d_k is low. Therefore, adaptive weights stay high, allowing full utilization of teacher supervision along the trajectory. In this regime, the optimization closely resembles sufficient distillation. **(2) Erroneous Pattern.** Persistent deviations increase step-level divergence, often caused by incorrect intermediate reasoning or tool usage. Adaptive weights are progressively reduced, which suppresses potentially misleading supervision and stabilizes training under such corrupted states. **(3) Recovery Pattern.** The student initially deviates but then recovers. Weights are attenuated during high-divergence steps and restored later, enabling effective guidance on corrected steps and preserving useful supervision after recovery. This behavior allows the model to avoid over-penalizing temporary errors while still benefiting from supervision once alignment is re-established.

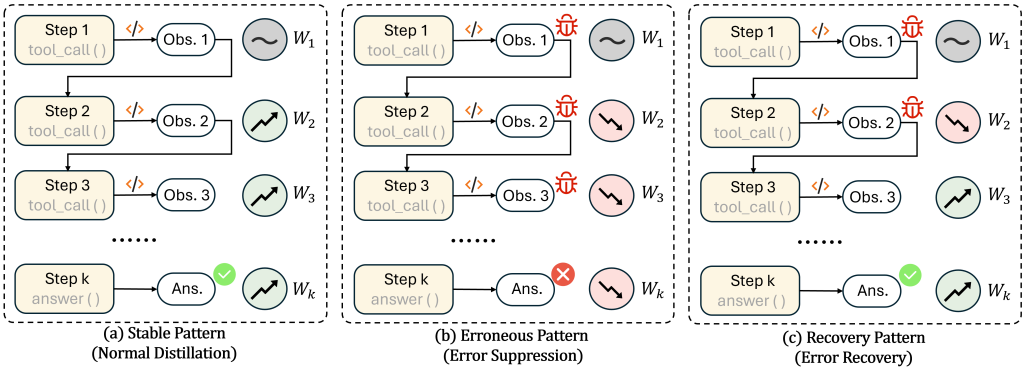


Figure 4: **Three distillation patterns of SOD**.

We also provide further analysis, *i.e.,* the quantitative analysis of distillation pattern distribution according training steps in Appendix F.7. Detailed examples are provided in Appendix J.

4.6 Further Experiments

The appendix reports eight further experiments: a 14B teacher alongside the 4B one at both student scales (Appendix F.1), three other agentic environments in place of the python interpreter (Appendix F.2), six teacher checkpoints at a fixed architecture (Appendix F.3), perturbations of the anchor d_1 together with sweeps over ϵ and δ (Appendix F.4), controlled interventions that overwrite the weights of the attenuated steps alone (Appendix F.5), wall-clock time and peak memory (Appendix F.6), the evolution of the three distillation patterns over training (Appendix F.7), and our objective with the reinforcement learning term removed (Appendix F.8). Appendix E additionally contrasts our weighting rule with the concurrent adaptive distillation methods included in Table 1.

5 Related Work

Reinforcement Learning for Agents. RL-based post-training has evolved from RLHF (Christiano et al., 2017) with PPO (Schulman et al., 2017) to more scalable methods like GRPO (Shao et al., 2024). For language agents, structured reasoning paradigms such as ReAct (Yao et al., 2023) enable tool use but rely on demonstrations rather than online optimization, and recent work extends RL to agent trajectories in code generation (Yang et al., 2024) and tool use (Feng et al., 2025). Broader frameworks scaling agentic RL across environments (Yu et al., 2025b; Wang et al., 2026d) still reward whole trajectories, leaving credit assignment under sparse feedback as the central difficulty.

On-policy Distillation. On-Policy Distillation (OPD) (Song & Zheng, 2026; Zheng et al., 2026; Xu et al., 2026b) introduces token-level supervision on student-generated trajectories to mitigate the distribution mismatch of offline distillation. Gu et al. (2024) formulates it as reverse KL minimization under the student distribution and Agarwal et al. (2024) unifies on-policy and off-policy distillation across divergence objectives, while Li et al. (2026b) shows that OPD primarily aligns local support on student-visited states. It has also been extended to self-distillation without external teachers (Zhao et al., 2026; Hübotter et al., 2026), and applied to mitigate the limitations of RL (Li et al., 2026a; Bousselham et al., 2025). Appendix B gives the full discussion.

6 Conclusion

In this work, we study on-policy distillation for small language model agents under tool-integrated reasoning. We identify that tool-induced transitions can push the student into regions where teacher supervision is unreliable, and propose SOD, which reweights distillation strength by student–teacher divergence, preserving informative signals in aligned regions while attenuating misleading guidance under drift. Experiments on math, science, and code benchmarks show consistent gains in stability and performance, suggesting that agent distillation needs supervision that adapts as the state shifts.

AI USE STATEMENT

Language models enter this work in two distinct ways. First, as the object of study: SOD is a distillation method whose teachers and students are language models, and the prompt templates that define the agent environment are given in Appendix H.1. Second, as an external writing aid: we used a general-purpose language model only to polish the grammar and wording of text the authors had written. The method, the experimental design, and the analysis are the authors' own, and the authors are responsible for the experimental results and the conclusions reported in this paper.

ETHICS STATEMENT

We identify no ethical concerns specific to this work. It involves no human subjects, no personal or otherwise sensitive data, and no user study, so no approval or consent procedure was required. All training corpora and evaluation benchmarks are publicly released resources intended for research on mathematical, scientific, and programming reasoning, and we use them as released and for that purpose. The agent executes code during both training and evaluation, and every such execution is confined to an isolated sandbox, so no untrusted program is run against the host environment. Our contribution is a training-time reweighting rule that changes how much of a teacher's supervision is applied at each step, so it affects how existing capabilities are transferred rather than granting a model any capability it could not otherwise acquire, and the models we study are small students distilled from open teachers. The remaining risks are largely those of the underlying open models and datasets. Appendix A states the limitations of the study.

REPRODUCIBILITY STATEMENT

We have tried to make every result in this paper reproducible from the description given. Our implementation is released anonymously at `https://anonymous.4open.science/r/SOD-CA86`. Section 3 defines the step-level divergence score and the reweighted objective, with the weighting rule in Section 3.2, and Appendix C states the complete procedure as pseudocode. Section 4.1 lists the datasets, the benchmarks, the baselines and the evaluation protocol, and Appendix D expands each of them, with the training corpora in Appendix D.1, the benchmarks in Appendix D.2, the eight baselines in Appendix D.3, and the hardware, the framework, the full hyperparameter configuration and the evaluation protocol in Appendix D.4. The two constants of SOD are swept in Appendix F.4, which also states the values used everywhere else. The assumptions and derivations behind both observations of Section 3.1 are given in Appendix G, the variance analysis in Appendix G.4, and the justification of the divergence score as a proxy in Appendix G.5. The prompt templates that define the agent environment are reproduced in Appendix H.1 together with the baseline-specific instruction in Appendix H.2, and the per-step entropy measurements behind our motivating analysis are shown in Appendix I.

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 *The twelfth international conference on learning representations*, 2024.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

Walid Bousselham, Hilde Kuehne, and Cordelia Schmid. Vold: Reasoning transfer from llms to vision-language models via on-policy distillation. *arXiv preprint arXiv:2510.23497*, 2025.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025.

Xiwen Chen, Jingjing Wang, Wenhui Zhu, Peijie Qiu, Xuanzhao Dong, Hejian Sang, Zhipeng Wang, Alborz Geramifard, and Feng Luo. Soda: Semi on-policy black-box distillation for large language models. *arXiv preprint arXiv:2604.03873*, 2026.

Li Chenglin, Qianglong Chen, Liangyue Li, Caiyu Wang, Feng Tao, Yicheng Li, Zulong Chen, and Yin Zhang. Mixed distillation helps smaller language models reason better. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1673–1690, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning. *arXiv preprint arXiv:2507.16812*, 2025.

Shengda Fan, Xuyan Ye, Yupeng Huo, Zhi-Yuan Chen, Yiju Guo, Shenzhi Yang, Wenkai Yang, Shuqi Ye, Jingwen Chen, Haotian Chen, et al. Agentprocessbench: Diagnosing step-level process quality in tool-using agents. *arXiv preprint arXiv:2603.14465*, 2026.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

Yuqian Fu, Haohuan Huang, Kaiwen Jiang, Yuanheng Zhu, and Dongbin Zhao. Revisiting on-policy distillation: Empirical failure modes and simple fixes. *arXiv preprint arXiv:2603.25562*, 2026.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *The twelfth international conference on learning representations*, 2024.

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

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision. *arXiv preprint arXiv:2604.12002*, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

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 *The Thirteenth International Conference on Learning Representations*.

Ijun Jang, Jewon Yeom, Juan Yeo, Hyunggu Lim, and Taesup Kim. Stable on-policy distillation through adaptive target reformulation. *arXiv preprint arXiv:2601.07155*, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Woogyeol Jin, Taywon Min, Yongjin Yang, Swanand Ravindra Kadhe, Yi Zhou, Dennis Wei, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. *arXiv preprint arXiv:2603.07079*, 2026.

Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. Distilling llm agent into small models with retrieval and code tools. *arXiv preprint arXiv:2505.17612*, 2025.

Jongwoo Ko, Sara Abdali, Young Jin Kim, Tianyi Chen, and Pashmina Cameron. Scaling reasoning efficiently via relaxed on-policy distillation. *arXiv preprint arXiv:2603.11137*, 2026.

Gengsheng Li, Tianyu Yang, Junfeng Fang, Mingyang Song, Mao Zheng, Haiyun Guo, Dan Zhang, Jinqiao Wang, and Tat-Seng Chua. Unifying group-relative and self-distillation policy optimization via sample routing. *arXiv preprint arXiv:2604.02288*, 2026a.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, et al. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl. *arXiv preprint arXiv:2508.13167*, 2025a.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025b.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026b.

Yuying Li, Leqi Zheng, Yongzi Yu, Wenrui Zhou, Xuchang Zhong, Xing Hu, Jing Jin, Hangjie Yuan, and Tao Feng. Filter, then reweight: Rethinking optimization granularity in on-policy distillation. *arXiv preprint arXiv:2606.02684*, 2026c.

Jun Liu, Zhenglun Kong, Peiyan Dong, Changdi Yang, Tianqi Li, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, et al. Structured agent distillation for large language model. *arXiv preprint arXiv:2505.13820*, 2025.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Jiahao Qiu, Xinzhe Juan, Yimin Wang, Ling Yang, Xuan Qi, Tongcheng Zhang, Jiacheng Guo, Yifu Lu, Zixin Yao, Hongru Wang, et al. Agentdistill: Training-free agent distillation with generalizable mcp boxes. *arXiv preprint arXiv:2506.14728*, 2025.

Corrado Rainone, Tim Bakker, and Roland Memisevic. Replacing thinking with tool usage enables reasoning in small language models. *arXiv preprint arXiv:2507.05065*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

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, 2025.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv preprint arXiv:2505.01441*, 2025.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. *arXiv preprint arXiv:2604.00626*, 2026.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Jiaqi Wang, Wenhao Zhang, Weijie Shi, Yaliang Li, and James Cheng. Tcod: Exploring temporal curriculum in on-policy distillation for multi-turn autonomous agents. *arXiv preprint arXiv:2604.24005*, 2026b.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving llm coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*, 2025.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026c.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. Rlanything: Forge environment, policy, and reward model in completely dynamic rl system. *arXiv preprint arXiv:2602.02488*, 2026d.

Yecheng Wu, Song Han, and Hai Cai. Lightning opd: Efficient post-training for large reasoning models with offline on-policy distillation. *arXiv preprint arXiv:2604.13010*, 2026.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms. *arXiv preprint arXiv:2504.14655*, 2025.

Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*, 2024a.

Shicheng Xu, Liang Pang, Yunchang Zhu, Jia Gu, Zihao Wei, Jingcheng Deng, Feiyang Pan, Huawei Shen, and Xueqi Cheng. Rlkd: Distilling llms' reasoning via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34151–34159, 2026a.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024b.

Yuanda Xu, Hejian Sang, Zhengze Zhou, Ran He, Zhipeng Wang, and Alborz Geramifard. Tip: Token importance in on-policy distillation. *arXiv preprint arXiv:2604.14084*, 2026b.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026a.

Fan Yang, Rui Meng, Trudi Di Qi, Ali Ezzati, and Yuxin Wen. Kepo: Knowledge-enhanced preference optimization for reinforcement learning with reasoning. *arXiv preprint arXiv:2602.00400*, 2026b.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*, 2026c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Yi Yao, He Zhu, Piaohong Wang, Jincheng Ren, Xinlong Yang, Qianben Chen, Xiaowan Li, Dingfeng Shi, Jiaxian Li, Qiexiang Wang, et al. O-researcher: An open ended deep research model via multi-agent distillation and agentic rl. *arXiv preprint arXiv:2601.03743*, 2026.

Tianzhu Ye, Li Dong, Zewen Chi, Xun Wu, Shaohan Huang, and Furu Wei. Black-box on-policy distillation of large language models. *arXiv preprint arXiv:2511.10643*, 2025.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

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

Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. Demystifying reinforcement learning in agentic reasoning. *arXiv preprint arXiv:2510.11701*, 2025b.

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 preprint arXiv:2601.18734*, 2026.

Binbin Zheng, Xing Ma, Yiheng Liang, Jingqing Ruan, Xiaoliang Fu, Kepeng Lin, Benchang Zhu, Ke Zeng, and Xunliang Cai. Scope: Signal-calibrated on-policy distillation enhancement with dual-path adaptive weighting. *arXiv preprint arXiv:2604.10688*, 2026.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl, 2024. *URL https://arxiv. org/abs/2402.19446*, 2024.

Wenhong Zhu, Ruobing Xie, Rui Wang, and Pengfei Liu. Hybrid policy distillation for llms. *arXiv preprint arXiv:2604.20244*, 2026.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

APPENDIX

A LIMITATIONS

Our work has two main limitations. (1) All experiments use the Qwen3 (Team, 2025) model family, chosen for its strong and stable performance across scales, open availability at multiple sizes, and wide adoption in recent OPD studies (Zhao et al., 2026; Xu et al., 2026b; Jin et al., 2026; Li et al., 2026b) that facilitates fair comparison. Validating on other model families remains valuable. (2) Training a student requires a capable teacher to be available beforehand, which carries the cost of obtaining that teacher. This requirement is inherited from the on-policy distillation paradigm itself rather than introduced by SOD, and it is shared by every method that learns from a teacher on the student's own trajectories, including the OPD baselines we compare against. Our contribution concerns how a given teacher's supervision is weighted across steps, so it applies to whatever teacher is already at hand and adds no forward pass of either model on top of the distillation it modulates, as Appendix F.6 reports. Due to resource constraints, we leave these explorations to future work.

B RELATED WORK

Reinforcement Learning for Agents. RL-based post-training has evolved from reinforcement learning from human feedback (RLHF) (Christiano et al., 2017; Ziegler et al., 2019) with PPO (Schulman et al., 2017) to more scalable methods like GRPO (Shao et al., 2024). For language agents, structured reasoning paradigms such as ReAct (Yao et al., 2023), Toolformer (Schick et al., 2023), and FireAct (Chen et al., 2023) enable tool use but rely on demonstrations rather than online optimization. Recent work extends RL to agent interaction trajectories across code generation (Yang et al., 2024), tool use (Feng et al., 2025), GUI interaction (Bai et al., 2024), and web navigation (Qi et al., 2024). A central challenge is credit assignment under sparse, delayed feedback, addressed via trajectory-level updates and value-free formulations (Zhou et al., 2024; Chen et al., 2025). KL-regularized policy optimization further introduces bias and instability concerns (Schulman et al., 2017), amplified in agentic settings by distribution shift and compounding errors. Broader frameworks scaling agentic RL across environments (Yu et al., 2025b; Wang et al., 2026d; 2025) still rely on trajectory-level rewards without dense supervision.

On-policy Distillation. On-Policy Distillation (OPD) (Song & Zheng, 2026; Ye et al., 2026; 2025; Zhu et al., 2026; Zheng et al., 2026; Xu et al., 2026b; Chen et al., 2026) introduces token-level supervision on student-generated trajectories to mitigate the distribution mismatch of offline distillation. Gu et al. (2024) formulates OPD as reverse KL minimization under the student distribution, while Agarwal et al. (2024) unifies on-policy and off-policy distillation across divergence objectives. Yang et al. (2026c) further interprets OPD as KL-regularized reinforcement learning with implicit per-token rewards, and Li et al. (2026b) shows that OPD primarily aligns local support on student-visited states, depending on teacher–student compatibility in reasoning patterns. OPD has also been extended to self-distillation settings without external teachers (Zhao et al., 2026; Penaloza et al., 2026; Hübotter et al., 2026; Shenfeld et al., 2026; He et al., 2026). Recently, more works (Li et al., 2026a; Yang et al., 2026a; Bousselham et al., 2025; Wang et al., 2026c;a) have explored introducing OPD to mitigate the limitations of RL.

C ALGORITHMIC DETAILS OF SOD

We present the complete training procedure of SOD in Algorithm 1. At each iteration, the student generates on-policy trajectories with tool interactions, computes step-level divergence against the teacher, derives adaptive distillation weights, and updates the policy with a combined GRPO and step-wise OPD objective.

Algorithm 1: SOD: Step-wise On-policy Distillation for Small Language Model Agents

Input: Student policy π_θ, teacher policy π_{teacher}, tool environment $\mathcal{E}$, prompt set $\mathcal{X}$, group size G, smoothing constant ϵ, weight cap δ, clipping range ϵ_{clip}.

Output: Trained student policy π_θ.

Stage I: On-Policy Rollout with Tool Interaction

for *each prompt* $x \in \mathcal{X}$ **do**

- For $i = 1, \ldots, G$: sample trajectory $\tau_i = (y_1, o_1, \ldots, y_K, o_K, y_{K+1})$ from $\pi_{\theta_{\text{old}}}$ by interacting with $\mathcal{E}$.
- Compute outcome reward $r_i = r(\tau_i)$ for each trajectory.
- Compute group-relative advantage $\hat{A}_i$ via Eq. (2).

Stage II: Student-teacher Divergence and Step-wise Reweighting

for *each trajectory* τ_i **do**

- Partition model-generated tokens into reasoning steps $\{\mathcal{I}_k\}_{k=1}^{K+1}$ (excluding tool observation tokens).
- For each step $k = 1, \ldots, K{+}1$, compute step-level divergence d_k via Eq. (6).
- Set $w_1 \leftarrow 1$.
- For $k = 2, \ldots, K{+}1$, compute adaptive weight w_k via Eq. (7).

Stage III: Combined Optimization

- Compute GRPO loss $\mathcal{L}_{\text{GRPO}}$ with clipped surrogate objective (Eq. (3)) using advantages $\{\hat{A}_i\}$.
- Compute step-wise OPD loss $\mathcal{L}_{\text{OPD}}^{\text{step}}$ via Eq. (9) using weights $\{w_k\}$.
- Update policy parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \big(\mathcal{L}_{\text{GRPO}} + \mathcal{L}_{\text{OPD}}^{\text{step}}\big)$.
- Synchronize old policy: $\theta_{\text{old}} \leftarrow \theta$.

D Experimental Setup

D.1 Training Datasets.

We provide information in detail of our training datasets including SFT dataset and RL dataset, which is following Yu et al. (2025b). For the prompts template for different benchmarks, please refer to Appendix H.1. We also provide statistical information of our training data in Table 3.

- **SFT dataset.** The SFT dataset is constructed from a mixture of curated and synthetic multi-turn problem-solving trajectories. Specifically, a teacher model (Qwen3-Coder-30B-A3B) is employed within an agent-based framework where the SandBoxFusion is used as the code interpreter to generate end-to-end interaction traces, where problems are drawn from three complementary sources: the s1-1k set (Muennighoff et al., 2025), a curated collection of 3k LeetCode problems, and a 2k multi-turn ReTool (Feng et al., 2025) dataset. To ensure data quality, the generated trajectories for the latter two subsets are scored using ReasonFlux-PRM (Zou et al., 2025), and only the top-ranked subsets (1k each) are retained, together with the full s1-1k set, resulting in a final 3k high-quality SFT corpus.
- **RL dataset.** The RL dataset is designed to emphasize diversity across domains in order to study its impact on training dynamics. It comprises 30k samples aggregated from multiple sources, including 17k mathematical reasoning problems from DAPO-Math (Yu et al., 2025a), a mixture of 4,902 math and 3,586 code tasks from Skywork-or1 (He et al., 2025), and an additional 3k science problems from MegaScience (Fan et al., 2025).

D.2 Benchmarks

We provide detailed descriptions of the four benchmarks used in our evaluation.

- **AIME 2024/2025.** The American Invitational Mathematics Examination (AIME) is a prestigious mathematics competition administered by the Mathematical Association of

Table 3: Summary of the training datasets used in our experiments.

Dataset Type	Domain	Source	# Samples
SFT Dataset	Mathematical	s1-1k (Muennighoff et al., 2025)	1k
	Coding	LeetCode (Xia et al., 2025)	1k
	Tool-use	ReTool (Feng et al., 2025)	1k
	Total		**3k**
RL Dataset	Mathematical	DAPO-Math (Yu et al., 2025a)	17k
		Skywork-or1 (Math) (He et al., 2025)	4,902
	Coding	Skywork-or1 (Code) (He et al., 2025)	3,586
	Scientific	MegaScience (Fan et al., 2025)	3k
	Total		$\sim$ **30k**

America (MAA). Each year's examination consists of two sessions (AIME I and AIME II), each containing 15 problems, for a total of 30 problems per year. All answers are integers in the range $[0, 999]$, which enables unambiguous automatic evaluation via exact match. The problems span advanced topics including algebra, number theory, combinatorics, and geometry, requiring multi-step reasoning chains and creative problem-solving strategies that go well beyond pattern matching. We use the 2024 and 2025 editions as benchmarks: AIME 2024 was administered in February 2024, and AIME 2025 in February 2025. The use of recent competition problems mitigates the risk of data contamination, as these problems are unlikely to appear in pre-training corpora of models with earlier knowledge cutoffs.

- **GPQA-Diamond** (Rein et al., 2024). GPQA (Graduate-Level Google-Proof Q&A) is a challenging multiple-choice question-answering benchmark consisting of questions authored by domain experts holding PhDs in biology, physics, and chemistry. The questions are deliberately designed to be "Google-proof", they cannot be answered through simple web searches, requiring instead genuine domain expertise and multi-step scientific reasoning. The dataset comprises three nested subsets: GPQA Extended (546 questions), GPQA Main (448 questions), and GPQA Diamond (198 questions). We adopt the Diamond subset, which is the highest-quality partition: it contains only questions where both independent domain expert validators answered correctly, while skilled non-expert validators (holding PhDs in other scientific fields, with unrestricted internet access) failed. Human performance baselines on the Diamond subset are approximately 65% for domain experts and 34% for non-experts, compared to a random baseline of 25%.
- **LiveCodeBench** (Jain et al.). LiveCodeBench is a continuously updated benchmark for evaluating the coding capabilities of large language models. Unlike static benchmarks such as HumanEval, LiveCodeBench mitigates data contamination by continuously collecting new problems from competitive programming platforms (LeetCode, AtCoder, and Codeforces) and annotating each problem with its release date. This temporal annotation enables contamination-aware evaluation by restricting the test set to problems released after a model's training data cutoff. We use the v6 release, which contains 1,055 problems spanning May 2023 to April 2025. Following common practice, we evaluate on problems within a recent time window to ensure minimal overlap with training data. The benchmark assesses multiple code-related capabilities including code generation, self-repair (debugging given execution feedback), code execution prediction, and test output prediction.

D.3 Baselines

We compare SOD with a diverse set of baselines spanning supervised learning, reinforcement learning, and distillation-based approaches. These baselines are chosen to reflect different sources of supervision (offline vs. on-policy), as well as different granularities of learning signals (sequence-level vs. token-level).

- **Vanilla.** The base model without any additional task-specific training. This baseline reflects the zero-shot or instruction-tuned capability of the model.

- **SFT (Supervised Fine-Tuning).** A standard supervised learning baseline where the model is trained to maximize the likelihood of reference solutions. This approach corresponds to off-policy imitation learning, where the model learns from fixed expert trajectories. While simple and effective, SFT suffers from exposure bias due to the mismatch between training (on ground-truth prefixes) and inference (on model-generated prefixes).

- **GRPO (Group Relative Policy Optimization)** (Shao et al., 2024). A reinforcement learning approach that optimizes the model using relative performance within a group of sampled responses. For each input, multiple candidate outputs are generated, and each is assigned a scalar reward based on task-specific correctness. Instead of learning a separate value function, GRPO computes advantages by normalizing rewards within the group, effectively measuring how much better or worse a response is compared to its peers. The model is then updated using a policy optimization objective with clipping to ensure stable training. While GRPO avoids the need for a learned critic and is relatively efficient, it relies on sparse, sequence-level rewards, assigning the same advantage to all tokens in a response and thus lacking fine-grained token-level supervision.

- **OPD (On-Policy Distillation)** (Agarwal et al., 2024). An on-policy distillation method where the model learns from its own generated trajectories while receiving dense token-level supervision from a teacher distribution. At each decoding step, the objective minimizes the divergence between the student and teacher token distributions along the student's rollout. This approach mitigates exposure bias and provides richer supervision compared to RL, but relies on an external teacher policy. Following recent findings that combining OPD with GRPO stabilizes training and improves performance (Li et al., 2026a; Yang et al., 2026a; Bousselham et al., 2025; Wang et al., 2026c; Agarwal et al., 2024; Wang et al., 2026a), our OPD baseline adopts the joint objective $\mathcal{L} = \mathcal{L}_{\text{GRPO}} + \mathcal{L}_{\text{OPD}}$ (see Eq. (10) for the objective of SOD). We additionally report results for naive OPD (without GRPO) in Appendix F.8.

- **OPSD$_{\text{gt}}$ (On-Policy Self-Distillation with Ground Truth)** (Zhao et al., 2026). A self-distillation variant where the same model plays both teacher and student roles. The student generates responses conditioned only on the input, while the teacher is additionally conditioned on the full ground-truth solution, which is treated as privileged information. The training objective minimizes the token-level divergence between the teacher and student distributions along the student's trajectory. This setting provides strong supervision but assumes access to complete reference solutions.

- **OPSD$_{\text{hint}}$ (On-Policy Self-Distillation with Hints)** (Zhao et al., 2026). A more practical variant of OPSD where the privileged information is replaced with a hint derived from the ground-truth solution. Instead of directly exposing the full solution, the teacher conditions on a compressed or partial guidance signal (*i.e.,* a hint), which provides weaker but more realistic supervision. This setup evaluates whether partial information can effectively guide self-distillation. Please refer to Appendix H.2 for the specific prompt in generating hints.

- **TIP (Token Importance in On-Policy Distillation)** (Xu et al., 2026b). A concurrent adaptive distillation method that applies the loss to a selected subset of tokens rather than to all of them. Each position is scored by combining the student's normalised entropy with the teacher–student divergence at that position, and only the top-scoring fraction is retained, on the argument that much of a trajectory is already agreed upon by both models and contributes little gradient signal. We use the authors' official implementation and keep the retention ratio at their reported default.

- **SCOPE (Signal-Calibrated On-Policy Distillation)** (Zheng et al., 2026). A concurrent adaptive distillation method that operates on complete trajectories. A verifier labels each student rollout as correct or incorrect, and the two groups are trained with different objectives: a weighted likelihood term for the correct group and a weighted distillation term for the incorrect one, with weights given by group-normalised perplexities computed within the candidate set of each prompt. We use the authors' official implementation and their default weighting temperature. TIP and SCOPE share our premise that distillation supervision should not be applied uniformly, which is why we include them here, and Appendix E sets out how their weighting units and signals differ from ours.

D.4 Implementation Details.

All experiments are conducted using the VeRL framework (Sheng et al., 2025) and Open-AgentRL framework (Yu et al., 2025b). To ensure a fair comparison, all methods share the same base training infrastructure, optimizer, and core hyperparameters unless otherwise noted. To promote reproducibility and better understanding the details of our implementation, we provide **an anonymous code link** at the end of the abstract.

- **Evaluation Protocol.** The temperature is fixed at 1.0 with nucleus sampling parameter top_p=0.6. For each problem, 32 independent samples are generated, from which average@32 is computed and reported as a percentage. The teacher is a Qwen3-4B model (Team, 2025) further optimized with GRPO on the RL datasets, and is the teacher of all subsequent experiments unless otherwise specified. The students are Qwen3-0.6B and Qwen3-1.7B.

- **Compute Resources.** All experiments are conducted on a single node with 8 NVIDIA H20 GPUs (96GB memory each). For RL and distillation-based methods, training runs for 1 epoch and typically takes 2-3 days for 0.6B and 1.7B student models (see Table 6), while training with larger 4B and 14B teacher models requires approximately 5-6 days. In contrast, supervised fine-tuning (SFT) is significantly more efficient, with 5 training epochs typically completed within a few hours under the same hardware setup. To ensure robustness and statistical reliability, all experiments are repeated over 5 independent runs with different random seeds. These details provide sufficient information about hardware configuration, execution time, and experimental protocol for reproducibility.

- **Supervised Fine-Tuning (SFT).** We fine-tune the base Qwen3-0.6B and Qwen3-1.7B models on the 3K SFT corpus for 5 epochs with a global batch size of 128. We use the AdamW optimizer with a learning rate of 5e-5. The maximum sequence length is set to 32,768 tokens with right truncation.

- **Baselines & Teacher Models(RL & Distillation).** For all RL and distillation-based methods (GRPO, OPD, OPSD_{gt}, $\text{OPSD}_{\text{hint}}$, TIP, SCOPE) including the teacher models, we adopt a unified training configuration to isolate the effect of each algorithm. Specifically, we use the AdamW optimizer with a learning rate of 1e-6, a training batch size of 64, and a mini-batch size of 16. All RL & Distillation baselines including SOD are trained based on the SFT checkpoint. The maximum prompt length is set to 2,560 tokens and the maximum response length to 20,480 tokens. We sample 16 responses per prompt during training and 32 during validation. All methods are trained for at most 1 epoch (For the teacher models, we train for at most 2 epochs). The multi-turn agent interaction supports up to 16 tool-call turns. Rollout is performed asynchronously via vLLM with tensor parallelism of 4. For distillation-based methods (OPD and its variants), the teacher model (Qwen3-4B, further optimized with GRPO) provides token-level supervision along the student's on-policy rollouts.

- **SOD.** SOD adopts the same training configuration as the baselines described above. The only additional hyperparameters are those governing the step-wise adaptive weighting mechanism: the initial weight w_1 is fixed as 1, the numerical stability constant $\epsilon = 10^{-6}$ and the upper-bound offset $\delta = 0.2$ (allowing per-step weights $w_k \leq 1 + \delta$). These parameters control the adaptive distillation strength based on the per-step student–teacher divergence, enabling the model to automatically modulate learning intensity across reasoning steps.

E Discussion: Comparison with Adaptive Weighting Methods in On-policy Distillation

A recent line of work improves on-policy distillation by reweighting the distillation signal rather than applying it uniformly. Because these methods share our high-level motivation that not all supervision is equally reliable, we discuss them here in detail and make explicit where SOD departs from them. All four appeared as arXiv preprints within a few months of this submission, so we treat them as concurrent work.

Table 4: **Adaptive weighting methods for on-policy distillation.** The four concurrent methods weight either individual tokens or complete trajectories, and their signals are measured at a single position or summarised over a whole rollout. SOD weights the agent step delimited by tool observations and drives the weight with the cumulative ratio of step-level divergence, so the supervision applied at a step reflects how the student–teacher mismatch has evolved along the trajectory.

Method	Granularity	Adaptation signal	Main mechanism
FiRe-OPD	Trajectory + Token	Teacher trajectory likelihood; teacher and student entropy	Discards the lowest-likelihood trajectories of each batch, then reweights tokens of the survivors by teacher confidence and student confusion
TIP	Token	Student entropy; teacher–student divergence	Scores tokens by a Soft-OR of the two signals and retains only the top-scoring subset for the distillation loss
SCOPE	Trajectory	Verifier correctness; sequence-level perplexity	Routes correct and incorrect trajectories to different objectives, weighted by group-normalised student and teacher perplexity respectively
EOPD	Token	Teacher entropy	Adds a forward-KL term at positions where the teacher entropy exceeds a threshold
SOD	**Agent step**	**Cumulative ratio of step-level teacher–student divergence**	**Attenuates or restores supervision across tool-delimited steps according to how the divergence has evolved along the trajectory**

E.1 What Each Method Weights, and by What Signal

Table 4 summarises the comparison along the two axes that matter for our argument: the granularity at which the weight is applied, and the signal that drives it.

FiRe-OPD (Li et al., 2026c). FiRe-OPD operates at two granularities. At the trajectory level it ranks student rollouts by the teacher's length-normalised log-likelihood and discards the lowest-scoring fraction of each batch, on the grounds that a trajectory the teacher finds unlikely lies outside the region where teacher supervision is trustworthy. At the token level it then reweights the surviving trajectories by a product of two entropy-derived terms, one measuring teacher confidence and one measuring student confusion, so that positions where the teacher is certain and the student is not receive the largest weight.

TIP (Xu et al., 2026b). TIP is a token-level selection method. It scores each position by combining the student's normalised entropy with the teacher–student divergence at that position through a parameter-free Soft-OR rule, then retains only the top-scoring fraction of tokens for the distillation loss. The motivation is that a large part of a trajectory is already agreed upon by both models and contributes little gradient signal, so concentrating the loss on a small informative subset preserves accuracy while reducing cost.

SCOPE (Zheng et al., 2026). SCOPE operates at the trajectory level and routes rather than merely reweights. A verifier labels each student rollout as correct or incorrect, and the two groups receive different objectives: correct trajectories are trained with a weighted likelihood term, incorrect ones with a weighted distillation term. The weights are group-normalised perplexities computed within the candidate set of a single prompt, taken from the student for the correct group and from the teacher for the incorrect group.

EOPD (Jin et al., 2026). EOPD is a token-level method driven by teacher entropy alone. At positions where the teacher's conditional entropy exceeds a threshold, it adds a forward-KL term to the standard reverse-KL objective, on the argument that high teacher entropy marks positions with several plausible continuations that a mode-seeking objective would otherwise collapse.

E.2 Where SOD Departs

The weighting unit is the agent step, which none of these methods has. FiRe-OPD, TIP and EOPD weight individual tokens; SCOPE weights complete trajectories. SOD weights the segment

of student output between two tool observations. This unit exists only in agentic settings, and it is the unit at which tool-induced corruption actually enters: an erroneous tool call does not degrade one token and it does not degrade the trajectory uniformly, but rather divides the trajectory into a portion generated before the corrupted observation entered the context and a portion generated after. A token-level rule cannot express this division because each token is scored in isolation, and a trajectory-level rule cannot express it because the whole rollout receives a single decision. This is why a single trajectory in our setting may contain reliable steps taken before a tool error, unreliable steps taken after the resulting state drift, and useful steps taken once the student has re-aligned, with all three requiring different treatment.

The weighting signal is a cumulative ratio, so it carries the history of the trajectory. The signals used by the four methods are all pointwise in the sense that the weight at a position depends only on quantities measured at that position, or on a single summary statistic of the whole trajectory. TIP's divergence term is the closest relative of our d_k, since both are computed from teacher and student log-probabilities that the distillation forward pass has already produced, but TIP consumes it as one coordinate of a per-token score and SOD consumes it as a term in the cumulative product $\prod_u \frac{d_u+\epsilon}{d_{u+1}+\epsilon}$. The consequence is that our weight at step k is determined by how the divergence has evolved from the first step up to the current one rather than by how large it is at the current one, which is what lets the weight fall as the state deteriorates and rise again as the student returns towards the teacher-supported region. A pointwise rule cannot recover supervision in this way, because it has no representation of the fact that the divergence is now smaller than it was.

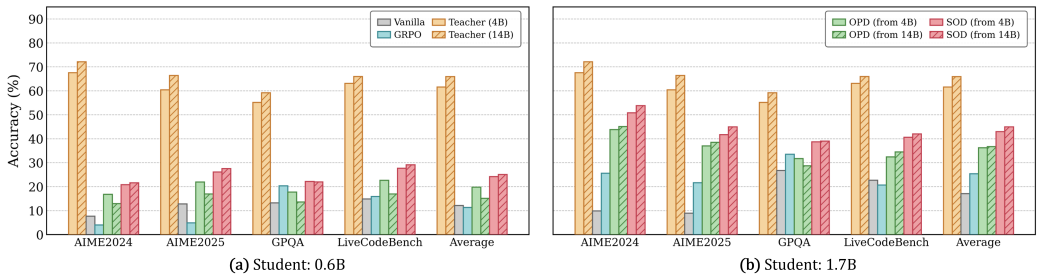
Reliability is inferred from the state, not from the outcome. SCOPE requires a verifier to label each trajectory before any weighting can be applied, which ties the method to tasks with automatically checkable answers and postpones every decision to the end of the rollout. FiRe-OPD's trajectory filter needs the complete trajectory as well, since the teacher likelihood is normalised over its full length. SOD makes its decision from the divergence accumulated so far, so the supervision applied at a step depends only on what has already happened, and neither an external verifier nor a completed rollout is required.

Discarding a corrupted region is not the same as attenuating it. FiRe-OPD removes the lowest-ranked trajectories outright, and TIP retains only the top-scoring tokens, so in both cases a region judged uninformative contributes nothing at all. Our ablation in Table 2 shows that in the agentic setting this is precisely the wrong treatment for post-error steps: the variant that masks all supervision after the first tool error is the weakest of every variant we tested, below even uniform weighting. The reason is that the student's recovery from an error is itself worth supervising, and a rule that removes the corrupted region removes the recovery along with it. SOD attenuates rather than discards, which keeps those steps available once the divergence begins to fall.

E.3 Why the Setting Matters

The distinction that organises all of the above is that these four methods were developed and evaluated on single-turn text-only generation. FiRe-OPD, TIP and SCOPE report mathematics and code generation, and EOPD reports mathematics; none of them involves tool calls, environment interaction, or multi-turn rollouts, and in all of them the context grows only by tokens the student itself produced. Under those conditions the student's state drifts away from the teacher gradually, one token at a time, and a weighting rule has no reason to model anything else. Our Observation 1 and the measurements in Figure 1 concern the opposite situation, in which the context absorbs an entire externally produced observation at once and a single malformed tool call can move the student into a region the teacher was never calibrated on.

This is a difference in setting and not a deficiency in those methods, and we note that FiRe-OPD lists prefix-aware weighting and intermediate step-level granularity as directions left to future work for exactly this reason. It does, however, explain the pattern in Table 1. TIP and SCOPE both improve on uniform-weight OPD at both student scales, which confirms that adaptive weighting is the right general direction, yet both remain well behind SOD because their weighting units and signals were designed for a setting in which tool-induced state transitions do not occur.



(a) Student: 0.6B (b) Student: 1.7B

Figure 5: **Scalability of SOD across different student-teacher configurations**.

F Additional Experiments

F.1 Scalability Across Student and Teacher Scales

To evaluate the generalization of SOD to different teacher models, we conduct experiments using an additional Qwen3 teacher (14B) to distill students at both the 0.6B and 1.7B scales. The results are visualized in Figure 5.

Results. SOD consistently benefits from stronger teachers, whereas OPD suffers from the increased capacity gap. An interesting finding emerges when comparing teacher choices under OPD: on the 0.6B student, switching from the 4B to the 14B teacher actually *degrades* average accuracy by a notable margin. This reveals that a larger student-teacher capacity gap amplifies distribution mismatch, causing uniform distillation to propagate increasingly unreliable supervision. In contrast, SOD with the 14B teacher consistently outperforms its 4B counterpart on both student scales, confirming that our step-wise reweighting effectively harnesses the richer knowledge from teachers while suppressing the harmful signals introduced by the wider gap. Changing the teacher model varies its capacity and its training together, so Appendix F.3 additionally holds the architecture fixed and varies only the training stage to isolate the effect of supervision quality on its own.

F.2 Generalization to Other Agentic Environments

Our main experiments use a python interpreter as the tool environment, which isolates tool-induced state drift in a clean and reproducible setting. A natural question is whether the step-wise reweighting mechanism depends on that particular environment. We therefore evaluate SOD on three agentic benchmarks whose interaction patterns differ substantially from python-based TIR: **ALFWorld** for long-horizon embodied interaction, **WebShop** for web navigation, and **Search-QA** for search over real corpora.

Setup. Each benchmark uses a Qwen3-1.7B student and a GRPO-trained Qwen3-4B teacher trained on that benchmark. The training protocol matches the main experiments: the student generates on-policy trajectories, and the frozen teacher scores only the states the student actually visits, so no teacher trajectories and no additional offline demonstrations are used. All methods run for 150 training steps with 16 rollouts per prompt, an actor learning rate of 1×10^{-6}, a maximum generation length of 512 tokens, and temperatures of 1.0 for training and 0.6 for evaluation. Generation uses vLLM on 8×96GB H20 GPUs, the built-in thinking mode of Qwen3 is disabled, and no additional actor KL loss is applied. We report the best validation success rate reached within the 150 steps together with the average trajectory length at that point. The three environments differ in their interaction budgets: ALFWorld uses train/validation batch sizes of 16/128, a maximum prompt length of 2,048, and at most 50 environment steps per trajectory; WebShop uses 16/128, a maximum prompt length of 4,096, and at most 15 interaction steps; Search-QA uses 128/512, a maximum prompt length of 4,096, and at most four search interactions per trajectory. Search-QA covers Natural Questions, TriviaQA, PopQA, HotpotQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle, spanning both open-domain and multi-hop reasoning.

Table 5: **Results on three agentic environments beyond the python interpreter.** All rows use a Qwen3-1.7B student with a GRPO-trained Qwen3-4B teacher, so the environment is the only factor that varies. SR is the best validation success rate within 150 training steps and Len is the average trajectory length at that point, with the best results in **bold**.

Method	ALFWorld		WebShop		Search-QA		Average SR↑
	SR↑	Len↓	SR↑	Len↓	SR↑	Len↓	
Vanilla	12.84	45.72	8.23	7.94	28.96	2.18	16.68
GRPO	35.47	37.86	52.31	9.22	39.82	2.73	42.53
OPD	68.94	18.76	73.28	6.74	42.15	2.81	61.46
SOD	**78.90**	**16.10**	**79.50**	**5.24**	**46.50**	**2.57**	**68.30**

Results. Table 5 shows that SOD attains the highest success rate on all three environments, improving the average success rate over OPD by 6.84 points. The margin is present in each environment individually and is largest on ALFWorld, which has by far the longest interaction budget and therefore the most room for an early error to propagate through subsequent steps. The trajectory lengths point in the same direction: SOD reaches its success rates with the shortest trajectories of all four methods in every environment, indicating that the gains come from resolving tasks more directly rather than from additional environment interaction.

The pattern across methods also matches the mechanism we describe in Section 3.1. GRPO improves over the vanilla student but stays far behind both distillation methods, consistent with sparse outcome-level rewards being difficult for a small model to exploit in long-horizon interaction. OPD recovers much of that gap through dense supervision, and the remaining margin to SOD is the part attributable to modulating that supervision by step-level divergence. These environments return observations whose form differs from interpreter output, including scene descriptions, page states, and retrieved passages, yet all of them share the property that an entire externally produced observation enters the context at once, which is the condition our method is built around.

We evaluate three tool environments here while keeping the model family fixed, so these results speak to the tool environment rather than to the choice of model family, and validating SOD on other model families remains valuable future work.

F.3 Dependence on Teacher Quality

Every result reported so far uses a teacher trained to convergence with GRPO, and Appendix F.1 varies the teacher by replacing the 4B model with a 14B one. Both leave open a question about the supervision itself rather than about its source: our method decides how much of the teacher's signal to keep at each step, so how much does it depend on that signal being reliable in the first place? Teacher capacity and teacher quality are confounded in a comparison between differently sized models, since a larger teacher is both more capable and differently trained. We therefore hold the architecture at Qwen3-4B and vary only how far its training has progressed, taking the vanilla model, its SFT checkpoint, and intermediate checkpoints at a quarter, a half, and three quarters of the GRPO schedule alongside the fully trained teacher used in our main experiments. Each of the six teachers distills the same Qwen3-1.7B student under identical data, settings, and training budget, so the only quantity that changes across the six runs is the quality of the supervision available.

Results. Figure 6(a) reports accuracy for the teacher itself and for both distillation methods. SOD exceeds OPD at every checkpoint, so the ordering between the two methods does not depend on how well trained the teacher is. What changes with teacher quality is the size of the margin, which widens steadily as the teacher improves, and Figure 6(b) states the same quantity as a relative gain: it grows from 3.56% under the vanilla teacher to 18.50% under the fully trained one, without flattening at any point in between.

That monotone relationship is what the mechanism predicts. Our reweighting does not create supervision but decides how much of the teacher's signal to apply at each step, so its value scales with how much reliable signal there is to preserve. A teacher that is poor everywhere offers little for

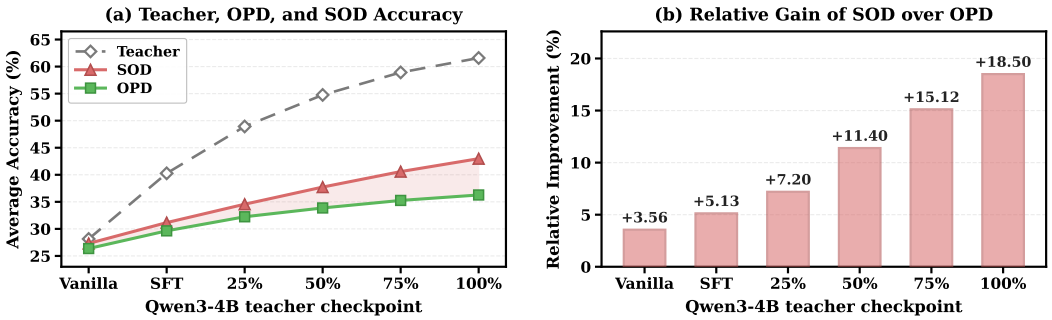


Figure 6: **Our gain over OPD widens as the teacher's supervision becomes more reliable.** Average accuracy over the four benchmarks with a Qwen3-1.7B student. The teacher architecture is fixed at Qwen3-4B and only its training stage varies, so quality changes while capacity does not; the percentages label intermediate checkpoints taken at that fraction of the GRPO schedule, and the rightmost one is the teacher used in all our other experiments. **(a)** Accuracy of the teacher itself and of both distillation methods, with the shaded band marking the margin between them. **(b)** The same margin expressed as the relative gain of SOD over OPD.

a step-wise rule to protect, and both methods stay close to the student's own level. As the teacher's supervision becomes trustworthy on the steps where the student remains aligned, distinguishing those steps from the ones corrupted by tool-induced drift becomes worth more, and the margin over uniform distillation grows accordingly.

Read in the other direction, the same trend delimits what our method can do. SOD degrades gracefully as the teacher weakens, in that it never falls below uniform distillation, but it cannot compensate for a teacher that is globally weak, since the supervision it modulates has to be worth applying somewhere. Our method therefore complements rather than substitutes for investment in teacher quality.

F.4 Robustness to the Anchor d_1 and to ϵ and δ

The weights in Eq. (7) telescope into $w_k \leq (d_1 + \epsilon)/(d_k + \epsilon)$ along a non-decreasing trajectory, as shown in Appendix G.4, so the first step enters the numerator of every subsequent weight. The divergence d_1 therefore acts as a trajectory-specific anchor, and sampling noise in it would rescale an entire weight trajectory rather than perturb one step. Each token position contributes a single sampled token to the estimate, but d_k averages over all $|\mathcal{I}_k|$ tokens of its step, which is the mechanism we expect to control that noise. We examine whether the averaging suffices in practice along two lines: a direct perturbation of d_1, and sweeps over the two constants ϵ and δ. All measurements in this subsection use the 1.7B student.

Direct perturbation of the anchor. We scale d_1 directly when computing all downstream step weights, leaving the remaining d_k untouched, which reproduces the effect an anchor misestimate would have. Figure 7(a) reports the result. Perturbing the anchor by a fifth in either direction costs at most 0.55 points, and the curve is nearly symmetric about the default, so an over-estimated and an under-estimated anchor are equally tolerable. This is the regime the telescoped form predicts: scaling d_1 multiplies every downstream weight by a common factor, which shifts the overall strength of the distillation term but leaves the relative ordering of steps within a trajectory intact, and it is that ordering the mechanism depends on.

The two constants. Figure 7(b) sweeps the smoothing constant ϵ across five orders of magnitude. Accuracy is flat while ϵ stays well below the observed scale of d_k, and only the largest setting we tried gives up a noticeable amount, which is the expected behaviour: once ϵ approaches the magnitude of the divergences themselves, it flattens the ratios that carry the step-to-step signal. Figure 7(c) sweeps the offset δ that caps the weight at $1 + \delta$, covering a fivefold range of how much upweighting a recovered step is allowed to receive. Every setting lands within three quarters of a point of the default and the sweep shows no clear monotonic relationship, so the method does not depend on tuning

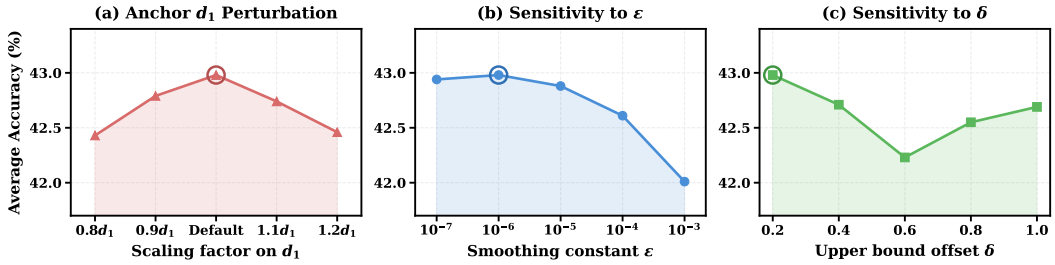


Figure 7: **Robustness to the anchor d_1 and to the two constants ϵ and δ.** Average accuracy over the four benchmarks, measured on the 1.7B student. The circled marker in each panel is the default setting used throughout our main experiments. **(a)** The anchor d_1 is scaled when computing all downstream step weights, leaving the remaining d_k untouched. **(b)** The smoothing constant ϵ in Eq. (7) is swept across five orders of magnitude. **(c)** The offset δ that caps the weight at $1 + \delta$ is swept over a fivefold range; the effect of removing the cap altogether is reported as ablation variant (1.4) in Table 2.

this bound once one is in place. What the bound itself contributes is reported separately as ablation variant (1.4) in Table 2, where removing it altogether is the more consequential change.

Taken together, the limited cost of perturbing the anchor directly and the stability across both constants indicate that the global-anchor construction does not introduce substantial instability under the settings we evaluate. We read this as a statement about practical effect rather than about the estimator: per-step token averaging makes the influence of d_1 estimation noise small here, and it does not eliminate that estimation variance.

F.5 Controlled Evidence That Down-weighted Steps Carry Harmful Supervision

The ablation in Table 2 shows that replacing our rule with uniform weighting, with a fixed decay, or with hard masking all cost accuracy, but each of those variants rewrites the weight of every step at once. They therefore establish that the weighting profile matters without isolating the claim our design rests on, namely that the steps our rule attenuates are the ones whose supervision is harmful. Divergence and entropy statistics likewise show only that those steps look different, not that acting on them is what helps. We therefore run a set of controlled interventions on the 1.7B student in which *only the weights of a designated set of target steps are modified*, while every other step keeps the weight our rule assigns it and all training settings stay fixed.

How the variants are constructed. Each run differs from SOD in exactly one respect: a set of target steps is designated, and the weight on those steps is overwritten. We designate target steps in two independent ways, and apply two intervention strengths to each.

- **Target steps chosen by our rule.** The target set is every step that SOD originally assigns $w_k < 1$, that is, every step the mechanism decided to attenuate. Overwriting the weight on exactly this set asks whether the supervision our rule suppresses is in fact harmful.
- **Target steps chosen by a fixed threshold.** The target set is every step with $d_k > 0.5$, selected without consulting our weighting rule at all. This arm exists because the first one designates steps using the very mechanism under test, which leaves open the objection that the rule might be attenuating steps that merely happen to suit it. A threshold applied to the raw divergence does not depend on any weighting decision, so agreement between the two arms cannot come from that circularity.
- **Restoring the supervision.** The weight on the target steps is set to $w_k = 1$, which is the value they would carry without attenuation. Steps outside the target set are unaffected and keep their original weight.

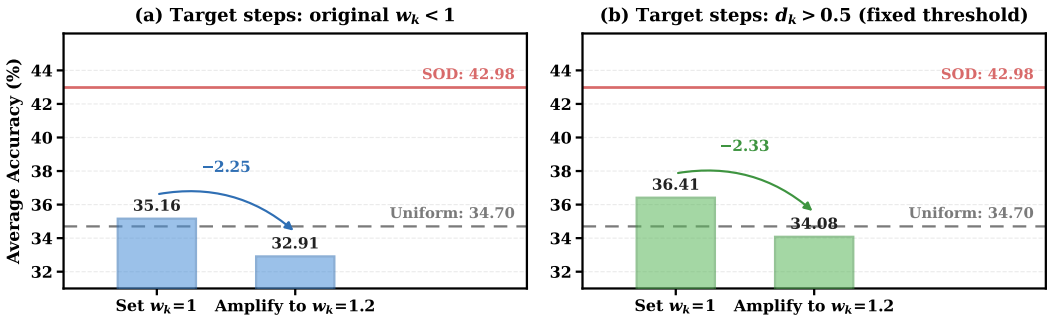


Figure 8: **Overwriting the weights of the attenuated steps costs accuracy, and overwriting them more costs more.** Average accuracy over the four benchmarks, measured on the 1.7B student. In every bar, *only the weights of the target steps are modified*; all remaining steps keep the weight our rule assigns them and no training setting is changed. The red line is SOD with its original weights and the dashed grey line is uniform weighting, which instead sets $w_k = 1$ at every step; both are reported in Table 2. **(a)** Target steps are those SOD originally attenuates ($w_k < 1$). **(b)** Target steps are selected by the fixed threshold $d_k > 0.5$, without consulting our weighting rule.

- **Strengthening the supervision.** The weight on the target steps is set to $w_k = 1.2$, so the supervision there is applied more strongly than it would be without any attenuation. Steps outside the target set again keep their original weight.

Two runs already reported in Table 2 serve as reference levels rather than as interventions: SOD with its original weights throughout, and uniform weighting, which sets $w_k = 1$ at *every* step rather than at a target set. The distinction between uniform weighting and the first intervention above is exactly the point of the design, since both assign unit weight to the attenuated steps but only the former also discards the weighting of every other step.

Results. Figure 8 reports both arms. Restoring unit weight on the steps our rule attenuates costs accuracy relative to SOD, and strengthening the supervision on those same steps costs more still. The threshold-selected arm reproduces this pattern with a margin of the same size, even though its target steps were never chosen by our mechanism.

The direction of that second comparison is what makes the evidence causal rather than correlational. If the attenuated steps merely carried uninformative supervision, restoring it would be roughly neutral and amplifying it would be no worse than restoring it; instead each increase in the weight placed on those steps costs further accuracy. The supervision available at those steps is therefore actively misleading, not simply redundant, and attenuating it is doing work. That the two arms agree while selecting their targets by unrelated criteria rules out the reading that our rule happens to pick steps flattering to itself.

A final detail of the reference levels is worth drawing out. Both interventions on our rule's own target steps land close to uniform weighting, and one falls below it. Assigning unit weight to the attenuated steps therefore gives up most of what the mechanism contributes even though every other step keeps its original weight, which indicates that the attenuated steps, rather than the general shape of the weighting profile, account for much of the gap between SOD and uniform weighting.

F.6 Training Cost and Practicality Analysis

A natural concern is whether the step-wise divergence computation in SOD introduces substantial overhead relative to standard OPD. We address this by reporting wall-clock training metrics for all methods. All experiments are conducted on 8×H20 GPUs with identical batch sizes and rollout configurations, trained for 120 steps (0.6B) or 150 steps (1.7B). Key observations are summarized as:

SOD vs. OPD: negligible algorithmic overhead. The step-wise divergence d_k (Eq. (6)) and adaptive weight w_k (Eq. (7)) are computed from the student and teacher log-probabilities already available during the OPD forward pass. The additional computation is merely per-step averaging and

Table 6: Training cost comparison. All runs use 8×H20 96GB GPUs with identical configurations.

Params	Method	Avg Time/Step (s)	Peak Memory (GB)	Total Time (h)
0.6B	GRPO	680.6	78.5	22.7
	OPD	1090.5	87.7	36.4
	SOD	1052.3	87.6	35.1
1.7B	GRPO	784.6	81.7	32.7
	OPD	1053.6	88.2	43.9
	SOD	1105.4	88.7	46.1

$O(K)$ scalar multiplications ($K \approx 3$–7), requiring no extra forward pass. Peak memory is equivalent (< 0.5 GB difference). Notably, for the 0.6B model, SOD is actually **3.5% faster** per step than OPD (1052.3s vs. 1090.5s), because the adaptive reweighting suppresses learning from erroneous tool-call patterns, resulting in more efficient rollouts with fewer failed retries and shorter responses. For the 1.7B model, SOD shows a modest $+4.9\%$ overhead (1105.4s vs. 1053.6s), which is entirely acceptable given the significant performance gains (+18.50% improvement at average) over OPD.

Why GRPO has lower per-step time. GRPO exhibits $\sim$1.4–1.5$\times$ lower per-step time than OPD-based methods. However, this is largely because GRPO's training collapses in later stages: as shown in Figure 3, the GRPO-trained student progressively loses the ability to perform effective tool calls, resulting in drastically shorter responses and fewer sandbox interactions in later training steps. This artificially deflates the average per-step time. In contrast, OPD and SOD maintain long, structured reasoning trajectories with active tool usage throughout training, which naturally requires more generation time per step.

F.7 Quantitative Analysis of Distillation Pattern Distribution

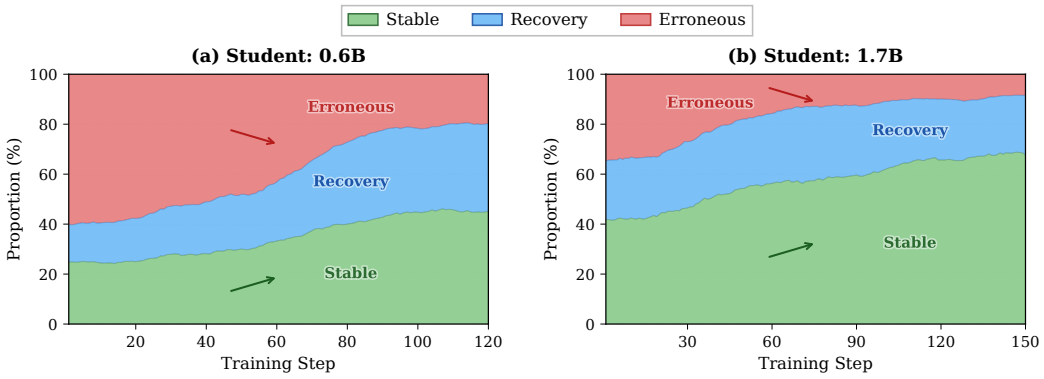


Figure 9: Distribution of three distillation patterns over training steps. At each step, all rollout trajectories are classified into Stable, Recovery, or Erroneous based on their adaptive weight dynamics, and the proportion of each pattern is reported (smoothed with a 9-step moving average).

To move beyond qualitative illustration and quantify how the three distillation patterns evolve during training, we conduct a systematic classification of all multi-step trajectories at each training step. Specifically, for every rollout sample at global step t, we record the full sequence of adaptive weights $\{w_k\}_{k=1}^{K}$ computed by SOD during that trajectory. We then classify each trajectory into one of three dominant patterns based on the overall shape of the w_k sequence:

- **Stable**: Adaptive weights remain consistently high throughout the trajectory, indicating that the student stays well-aligned with the teacher across all reasoning steps.
- **Erroneous**: Weights are progressively suppressed and remain low by the final step, indicating persistent divergence that the student fails to correct.

- **Recovery**: Weights drop during intermediate steps (reflecting temporary divergence) but recover by the final step, indicating that the student successfully re-aligns after an initial deviation.

We note that a fourth logical category exists, *i.e.,* trajectories that begin aligned but *degrade* toward the end (*i.e.,* early weights are high but the final weight is low). In practice, we find this pattern accounts for a negligible fraction ($<1\%$) of all trajectories, as persistent degradation without any preceding divergence signal is rare under SOD's cumulative reweighting mechanism. We therefore merge these cases into the erroneous category for simplicity. At each training step, we classify all rollout samples and compute the proportion of each pattern. Results are smoothed with a 9-step moving average for visualization clarity. Figure 9 reports the distribution over the training trajectory for both model scales.

Key observations. **(1) Stable proportion increases over training.** For both 0.6B and 1.7B, the fraction of stable trajectories grows steadily, confirming that SOD progressively improves student-teacher alignment as training proceeds. **(2) Erroneous proportion decreases consistently.** The proportion of erroneous trajectories declines substantially over training. This indicates that, as training progresses under SOD, the student model becomes increasingly capable: when encountering difficult reasoning steps, it no longer falls into consecutive errors but instead recovers or avoids mistakes altogether. **(3) Recovery grows and becomes a persistent component.** Notably, the recovery proportion is initially low (especially for the 0.6B model) and *increases* over training before stabilizing at a substantial level. This reveals a meaningful progression: early in training, the student lacks the capacity to recover from divergence; once it deviates, it tends to remain erroneous. As training proceeds, the student develops the ability to self-correct after intermediate missteps, converting what would have been erroneous trajectories into recovery ones. Even at the end of training, recovery trajectories account for a significant proportion of all samples, indicating that SOD's adaptive reweighting mechanism remains *continuously active*. **(4) Model capacity affects convergence speed and pattern composition.** The 1.7B model reaches a higher stable proportion faster and drives erroneous to a lower level by the end of training, while maintaining a moderate recovery proportion throughout. In contrast, the 0.6B model starts with a much higher erroneous fraction and lower recovery fraction, reflecting its initially limited ability to self-correct. As training progresses, the 0.6B model gradually develops recovery capability, but converges more slowly overall. This suggests that SOD's protective suppression of erroneous signals is particularly critical for smaller models during early training.

F.8 The Performance of Naive On-policy Distillation

In the main experiments, our OPD baseline combines the OPD loss with GRPO (*i.e.,* $\mathcal{L} = \mathcal{L}_{\text{GRPO}} + \mathcal{L}_{\text{OPD}}$) as illustrated in Appendix D.3. Here we report results for *naive OPD*, which uses only the token-level distillation loss without any RL reward signal.

Table 7: Comparison of naive OPD vs. OPD on the 1.7B student model.

Method	Math		Science	Code	Average
	AIME 2024	AIME 2025	GPQA	LiveCodeBench	
naive OPD	40.54 ±1.05	35.83 ±1.11	28.96 ±0.58	29.81 ±0.74	33.79
OPD	43.86 ±1.23	37.04 ±1.31	31.73 ±0.55	32.45 ±0.93	36.27
SOD (w/o GRPO)	48.87 ±0.93	39.73 ±1.02	35.89 ±0.62	38.62 ±0.71	40.78
SOD	**50.83** ±1.15	**41.72** ±1.24	**38.72** ±0.73	**40.63** ±0.91	**42.98**

As shown in Table 7, naive OPD underperforms the OPD baseline by 2.48 points on average (33.79 vs. 36.27). Without the outcome-level reward signal from GRPO, the student relies entirely on dense teacher supervision for learning. However, as analyzed in Section 3.1, OPD applies uniform distillation across all steps including those corrupted by tool-induced state drift. In the absence of GRPO's reward-driven exploration, the model lacks an independent signal to distinguish successful from failed trajectories, making it more susceptible to the compounding error problem.

More notably, comparing the two pure distillation variants, naive OPD (33.79) vs. SOD w/o GRPO (40.78), reveals that our step-wise reweighting alone accounts for a +6.99 (+20.69%) improvement without any RL reward signal. This confirms that the core benefit of SOD stems from the adaptive step-wise reweighting mechanism rather than from the RL component. GRPO provides complementary gains (+2.20 on top of step-wise OPD), but the reweighting mechanism remains the primary driver of performance improvement. These results also suggest that step-wise reweighting and GRPO address orthogonal aspects of the training challenge: the former stabilizes dense supervision under state drift, while the latter provides sparse outcome-level exploration signals.

G Supporting Derivations for Section 3.1

We adopt the notation established in Section 3.1. For notational convenience in the derivations below, we write $p_t(\cdot) = \pi_\theta(\cdot \mid y_{<t})$ and $q_t(\cdot) = \pi_{\text{teacher}}(\cdot \mid y_{<t})$ for the student and teacher next-token distributions conditioned on the student-generated prefix $y_{<t}$. The token-level OPD loss is $\ell_t = \log p_t(y_t) - \log q_t(y_t)$ with $y_t \sim p_t$. The step-level mismatch Δ_k, the step-level total variation TV_k, the teacher-supported region S_t^ϵ, and the overlap ρ_t are as defined in the main text.

G.1 Supporting Derivation for Observation 1

The argument below is a total-variation accounting of how far a prefix moves the next-token distribution. It rests on the two per-token shift magnitudes assumed in the observation, which are modelling assumptions about generation and about tool observations rather than quantities we derive, so what follows motivates the design instead of certifying it.

Text-only case (gradual drift). In text-only generation, the prefix at position t differs from position $t-1$ by exactly one student-sampled token. Define the per-token distributional shift as:

$$\eta \triangleq \max_t \ \mathrm{TV}(p_t, p_{t-1}) = \max_t \ \frac{1}{2} \sum_{v \in \mathcal{V}} |p_t(v) - p_{t-1}(v)| \,, \tag{11}$$

i.e., the maximum total variation (TV) distance between consecutive output distributions. For well-trained transformers, η is small because the self-attention mechanism distributes influence across many context positions, so appending a single token has limited impact on the output distribution. Since consecutive steps share a boundary of one token, the step-level shift changes by at most $O(\eta)$ per step transition.

TIR case (discontinuous drift). In TIR, the transition from step k to step $k+1$ involves appending an entire tool observation $o_k = (o_k^1, \ldots, o_k^m)$ of length $m = |o_k|$ to the prefix:

$$y_{<t_{k+1}^{\text{start}}} = y_{<t_k^{\text{end}}} \ \oplus \ o_k, \tag{12}$$

where $\oplus$ denotes concatenation. Crucially, o_k is produced by the external environment, not by either model. Let η_{tool} denote the average per-token distributional shift induced by observation tokens, measured by the TV distance between the model's output distribution before and after conditioning on each successive observation token.

Single-step accounting. By the triangle inequality applied to the m intermediate distributions between $p_{t_k^{\text{end}}}$ and $p_{t_{k+1}^{\text{start}}}$, the total variation shift satisfies:

$$\mathrm{TV}\Big(p_{t_{k+1}^{\text{start}}},\ p_{t_k^{\text{end}}}\Big) \ \le \ m \cdot \eta_{\text{tool}}. \tag{13}$$

A similar bound holds for the teacher distribution. Tool observations typically span tens to hundreds of tokens ($m \gg 1$) and introduce content generated by neither model, which is why we assume $\eta_{\text{tool}} > \eta$. The admissible shift across a tool boundary is therefore of order $m \cdot \eta_{\text{tool}}$, against order η within text, so a single erroneous tool call already opens the door to a far larger state change than ordinary autoregressive generation does. We stress that this is an accounting of how large the shift may be rather than a certificate that it is attained, and that it is stated in total variation: the step-level mismatch Δ_k used by the reweighting rule is a KL average, and a bound on a TV shift does not by itself order a difference of two such averages. Even so, the teacher has encountered some error patterns during training and can still provide partially useful supervision after an isolated failure. The critical issue arises from the *cascading* nature of errors in weaker student models, as we discuss next.

Cascading errors and faster-than-linear compounding. Weaker student models, precisely the targets of OPD, are prone to producing consecutive erroneous tool calls. A weaker student is more likely to generate an incorrect tool invocation at step k; conditioned on the resulting erroneous observation, its subsequent reasoning is further degraded, increasing the probability of another failure at step $k+1$. This creates a cascading failure pattern where each error compounds upon the previous ones.

The key insight is that while the teacher may reasonably handle a single isolated error in the context (having potentially seen similar error messages during training), the *joint* occurrence of multiple consecutive failures creates a conditioning context that is combinatorially unlikely under the teacher's training distribution. If each individual error context has marginal probability p_{err} under the teacher's training distribution, the joint context of j consecutive errors has probability at most p_{err}^j, an exponential decay in familiarity. It is this accumulated, multi-error prefix that we expect to drive the teacher's conditional distribution far from calibration, making $\eta_{\text{tool}}^{(i)}$ grow with i.

Applying the same accounting across steps $k, k+1, \ldots, k+j-1$, the admissible cumulative shift is of order

$$\sum_{i=0}^{j-1} m_{k+i} \cdot \eta_{\text{tool}}^{(k+i)}, \tag{14}$$

where m_{k+i} is the length of the i-th erroneous observation and $\eta_{\text{tool}}^{(k+i)}$ is the corresponding per-token shift. Under the assumption that later errors induce progressively larger per-token shifts ($\eta_{\text{tool}}^{(k+i+1)} \geq \eta_{\text{tool}}^{(k+i)}$), because the prefix is already corrupted by prior errors and the teacher has never been calibrated on such multi-error contexts, this sum grows faster than linearly in the number of consecutive tool failures. Figure 1(a) is consistent with this picture: the measured student-teacher divergence accelerates as errors accumulate.

G.2 Supporting Derivation for Observation 2

The derivation below is conditional on the assumptions collected in Observation 2: a reduced support overlap $\rho_t \leq \rho$, the support threshold $\epsilon < 1/|\mathcal{V}|$, a bounded expected gradient, and a strictly positive minimum score norm. It characterises one way in which OPD can break down in severely mismatched states; it is not a universal statement about the variance or the SNR of OPD.

Step 1: Decomposition of the second moment. The second moment of the OPD loss $\ell_t = \log p_t(y_t) - \log q_t(y_t)$ under $y_t \sim p_t$ decomposes over the vocabulary:

$$\mathbb{E}_{y_t \sim p_t}[\ell_t^2] \;=\; \sum_{v \in \mathcal{V}} p_t(v) \left(\log \frac{p_t(v)}{q_t(v)} \right)^2. \tag{15}$$

Step 2: Restricting to the low-overlap region. We partition the vocabulary into the teacher-supported region $S_t^\epsilon = \{v : q_t(v) \geq \epsilon\}$ and its complement $\bar{S}_t^\epsilon = \{v : q_t(v) < \epsilon\}$. By the overlap condition $\rho_t \leq \rho$, the student places mass at least $1-\rho$ on $\bar{S}_t^\epsilon$:

$$\sum_{v \in \bar{S}_t^\epsilon} p_t(v) \;\geq\; 1 - \rho_t \;\geq\; 1 - \rho. \tag{16}$$

Restricting the sum to $\bar{S}_t^\epsilon$ gives a lower bound:

$$\mathbb{E}[\ell_t^2] \;\geq\; \sum_{v \in \bar{S}_t^\epsilon} p_t(v) \left(\log \frac{p_t(v)}{q_t(v)} \right)^2. \tag{17}$$

Step 3: Applying Jensen's inequality. For any $v \in \bar{S}_t^\epsilon$, since $q_t(v) < \epsilon$, we have:

$$\log \frac{p_t(v)}{q_t(v)} \;>\; \log p_t(v) + \log \frac{1}{\epsilon}. \tag{18}$$

Define the conditional distribution $\tilde{p}(v) = p_t(v)/(1-\rho_t)$ over $\bar{S}_t^\epsilon$. By the convexity of $x \mapsto x^2$ and Jensen's inequality:

$$\begin{aligned}\mathbb{E}[\ell_t^2] &\geq (1-\rho_t) \sum_{v \in \bar{S}_t^\epsilon} \tilde{p}(v) \left(\log \frac{p_t(v)}{q_t(v)}\right)^2 \\ &\geq (1-\rho_t) \left(\sum_{v \in \bar{S}_t^\epsilon} \tilde{p}(v) \log \frac{p_t(v)}{q_t(v)}\right)^2 \\ &\geq (1-\rho_t) \left(\log \frac{1}{\epsilon} + \sum_{v \in \bar{S}_t^\epsilon} \tilde{p}(v) \log p_t(v)\right)^2 . \end{aligned} \tag{19}$$

Step 4: Bounding the entropy term. The term $\sum_{v \in \bar{S}_t^\epsilon} \tilde{p}(v) \log p_t(v) = -H_{\bar{S}}$, where $H_{\bar{S}}$ is the conditional entropy of the student restricted to $\bar{S}_t^\epsilon$, satisfying $0 \leq H_{\bar{S}} \leq \log|\mathcal{V}|$. For sufficiently small ϵ satisfying $\epsilon < 1/|\mathcal{V}|$ (a natural condition since the teacher threshold should be below the uniform baseline), we have $\log(1/\epsilon) > \log|\mathcal{V}| \geq H_{\bar{S}}$, so the $\log(1/\epsilon)$ term dominates:

$$\mathbb{E}[\ell_t^2] \geq (1-\rho) \left(\log \frac{1}{\epsilon} - H_{\bar{S}}\right)^2 \geq (1-\rho) \log^2 \frac{1}{\epsilon} \cdot \left(1 - \frac{\log|\mathcal{V}|}{\log(1/\epsilon)}\right)^2, \tag{20}$$

which grows as $\Omega(\log^2(1/\epsilon))$ when $\rho_t \to 0$.

Step 5: SNR degradation. Let $g_t = \ell_t \nabla_\theta \log \pi_\theta(y_t \mid y_{<t})$ be the token-level OPD gradient estimator. The expected gradient is $\mathbb{E}[g_t] = \nabla_\theta D_{\mathrm{KL}}(p_t \| q_t)$, and we write $C = \|\mathbb{E}[g_t]\|$ for its norm, which the observation assumes to be bounded. We likewise use the assumed positive minimum score norm $c_{\min} \triangleq \min_v \|\nabla_\theta \log \pi_\theta(v \mid y_{<t})\|^2 > 0$. Both quantities are assumptions of the observation rather than properties established here. Under them, the variance of the gradient estimator satisfies:

$$\mathrm{Var}[\|g_t\|] \geq \mathbb{E}[\ell_t^2] \cdot c_{\min} - C^2. \tag{21}$$

Combining with the second-moment lower bound from Step 4:

$$\mathrm{SNR}(g_t) = \frac{\|\mathbb{E}[g_t]\|}{\sqrt{\mathrm{Var}[g_t]}} \leq \frac{C}{\sqrt{(1-\rho)\log^2(1/\epsilon) \cdot c_{\min} - C^2}}. \tag{22}$$

As $\rho \to 0$, the denominator grows as $O(\log(1/\epsilon))$ while the numerator C remains constant, giving $\mathrm{SNR}(g_t) \to 0$.

Interpretation. When the student drifts far from the teacher-supported region ($\rho_t \approx 0$), most sampled tokens fall in $\bar{S}_t^\epsilon$ where the teacher assigns near-zero probability. The OPD loss for these tokens has large magnitude (due to $-\log q_t(y_t) > \log(1/\epsilon)$) but is uninformative. It reflects the teacher's inability to model these out-of-distribution states rather than a meaningful learning signal. Meanwhile, informative signals from S_t^ϵ (where teacher guidance is calibrated) are sampled only with probability ρ_t, making them increasingly rare. The gradient is thus dominated by high-variance, low-information contributions, rendering optimization unstable.

G.3 Discussion: Failure Cascade and Design Implications

Read together, the two observations describe a failure cascade specific to TIR:

1. **Initial perturbation**: An erroneous tool call returns a corrupted observation (*e.g.,* a runtime error, incorrect output, or timeout message). This already admits a state shift substantially larger than text-only drift, of order $m \cdot \eta_{\mathrm{tool}}$ against order η, though the teacher, having encountered some error patterns during pretraining, can still provide partially useful supervision after an isolated failure.

2. **Cascading accumulation**: Weaker student models, precisely the targets of OPD, are prone to making consecutive errors. Each subsequent erroneous tool call further corrupts the prefix, and the *joint* pattern of multiple consecutive failures becomes exponentially unlikely under the teacher's training distribution ($\sim p_{\text{err}}^j$ for j consecutive errors). It is this accumulated multi-error context, rather than any single error, that drives the teacher's conditional distribution far from calibration. The admissible drift thus compounds faster than linearly (Obs. 1), consistent with the accelerating divergence curve in Figure 1(a).
3. **Supervision breakdown**: In the resulting low-overlap states ($\rho_t \approx 0$) caused by accumulated consecutive errors, the OPD gradient estimator can suffer variance explosion and SNR degradation (Obs. 2). Updates become dominated by uninformative, high-magnitude contributions from tokens where the teacher provides no meaningful guidance. Figure 1(b) is consistent with this empirically: the teacher's conditional entropy becomes both elevated and highly variable in steps following accumulated tool errors.
4. **Amplification by uniform aggregation**: OPD sums token-level losses across all steps with equal weight, treating well-aligned early steps (high ρ_t, reliable teacher guidance) identically to corrupted post-error steps (low ρ_t, unreliable guidance). When cumulative tool errors cause a substantial portion of later steps to have low overlap, the aggregate gradient is systematically biased by these high-variance contributions.

This picture suggests that a stabilizing objective should modulate distillation strength at the step level: preserving full-strength dense supervision in well-aligned steps while attenuating the signal when the estimated student-teacher divergence indicates teacher miscalibration. This is the design principle underlying our adaptive step-wise reweighting mechanism.

G.4 Variance Attenuation under Step-wise Reweighting in the Non-decreasing Regime

We now analyse how the step-wise reweighting mechanism in Eq. equation 7 acts on the gradient second moment discussed in Observation 2. The analysis is explicitly conditional: it covers trajectories along which the step-level divergence is non-decreasing and along which the weight cap has not engaged. Sections G.4 and G.5 state where that leaves the recovery regime.

Proposition 1 (Conditional variance attenuation)**.** *Consider the step-wise weighted OPD objective (Eq. 9) and a trajectory along which the step-level divergence is non-decreasing, $d_1 \le d_2 \le \cdots \le d_k$, so that the cap $1+\delta$ in Eq. equation 7 is inactive. Then the weighted gradient contribution from step k satisfies*

$$\mathbb{E}[w_k^2 \cdot \ell_t^2] \;\le\; \frac{(d_1+\epsilon)^2}{(d_k+\epsilon)^2} \cdot \mathbb{E}[\ell_t^2], \tag{23}$$

where $\mathbb{E}[\ell_t^2]$ is the unweighted second moment appearing in Observation 2. The attenuation is strict wherever the divergence strictly increases and leaves the contribution unchanged wherever consecutive divergences are equal. The statement does not extend to trajectories along which the divergence decreases, where the cap may engage; that regime is addressed empirically rather than here.

Proof. By the definition of w_k in Eq. equation 7:

$$w_k = \min\left(\prod_{u=1}^{k-1} \frac{d_u+\epsilon}{d_{u+1}+\epsilon},\; 1+\delta\right). \tag{24}$$

Under non-decreasing divergence ($d_1 \le d_2 \le \cdots \le d_k$), each ratio $\frac{d_u+\epsilon}{d_{u+1}+\epsilon} \le 1$, so the product is at most one, the cap is inactive, and the product telescopes:

$$w_k \;\le\; \prod_{u=1}^{k-1} \frac{d_u+\epsilon}{d_{u+1}+\epsilon} \;=\; \frac{d_1+\epsilon}{d_k+\epsilon}. \tag{25}$$

Each ratio is strictly below one exactly when $d_u < d_{u+1}$ and equals one when $d_u = d_{u+1}$, which gives the strictness claim. The weighted second moment of the OPD loss at token $t \in \mathcal{I}_k$ is therefore:

$$\mathbb{E}[w_k^2 \cdot \ell_t^2] \;\le\; \left(\frac{d_1+\epsilon}{d_k+\epsilon}\right)^2 \cdot \mathbb{E}[\ell_t^2]. \tag{26}$$

Observation 2 indicates that under its assumptions, a student drifting into low-overlap states has $\mathbb{E}[\ell_t^2] \geq (1-\rho)\log^2(1/\epsilon_{\text{teacher}})$. Such drift also raises d_k, since d_k is monotonically consistent with Δ_k (Appendix G.5): as ρ_t falls, the per-token log-probability gaps grow and d_k moves away from d_1. Within the regime considered here, the weighted contribution then satisfies:

$$\mathbb{E}[w_k^2 \cdot \ell_t^2] \;\leq\; \frac{(d_1+\epsilon)^2}{(d_k+\epsilon)^2} \cdot (1-\rho)\log^2(1/\epsilon_{\text{teacher}}). \tag{27}$$

The numerator $(d_1+\epsilon)^2$ is fixed by the first step while the denominator $(d_k+\epsilon)^2$ grows with the accumulated drift, so the attenuation strengthens exactly as the unweighted second moment grows. This is the sense in which the reweighting counteracts the mechanism of Observation 2 along such trajectories.

Effect on the aggregate gradient. For the weighted gradient estimator $\tilde{g}_t = w_k \cdot \ell_t \cdot \nabla_\theta \log \pi_\theta(y_t \mid y_{<t})$, the contribution of well-aligned early steps is left essentially intact, since $w_k \approx 1$ there, while the contribution of high-divergence later steps is scaled by $(d_1+\epsilon)^2/(d_k+\epsilon)^2$. The aggregate gradient therefore stays dominated by the steps where teacher supervision is calibrated, which is the part of the trajectory that Observation 2 leaves unaffected. □

Scope of the analysis. Proposition 1 covers the non-decreasing regime, which is the one the erroneous pattern of Section 4.5 falls into. The recovery pattern, where $d_{u+1} < d_u$ for some u and the weight may rise above one, lies outside it. There the cap $w_k \leq 1+\delta$ limits how far a single step can be amplified, and it keeps the weighted second moment within $(1+\delta)^2 \cdot \mathbb{E}[\ell_t^2]$, but limiting the amplification is not the same as attenuating the variance, so the cap should be read as a design safeguard rather than as an extension of the bound above. Our justification for restoring supervision during recovery is instead the one set out in Section 3.2: a falling d_k indicates that the student is re-entering the teacher-supported region, where dense supervision is informative again and where discarding it is measurably harmful. The ablations bear this out, as suppressing all post-error supervision is the weakest variant we tested (Table 2), and the pattern statistics in Section 4.5 show the recovery mode becoming and remaining a substantial fraction of steps as training proceeds.

Interpretation. The analysis matches the design motivation of our method. OPD applies uniform weight to all steps, which lets high-variance contributions from corrupted post-error steps dominate the gradient. Our step-wise reweighting registers those steps through a rising d_k and attenuates their influence in proportion, keeping the gradient signal dominated by informative contributions from well-aligned steps. The cap $1+\delta$ plays the complementary role of preventing any single step from being over-amplified once the student begins to re-align. One consequence of the telescoped form above deserves note: since d_1 appears in the numerator of every downstream weight, an error in estimating it rescales a whole weight trajectory rather than a single step. Appendix F.4 measures what a deliberate perturbation of d_1 costs.

G.5 Justification of d_k as a Proxy for Δ_k

Computing the full KL divergence Δ_k requires evaluating the teacher's output distribution over the entire vocabulary at each token position, which introduces substantial overhead. In contrast, d_k (Eq. 6) only requires the teacher's log-probability on the student-sampled token y_t, a quantity already computed in the standard OPD forward pass.

Monotonic consistency. By Jensen's inequality:

$$d_k = \frac{1}{|\mathcal{I}_k|}\sum_{t\in\mathcal{I}_k}\left|\log\frac{\pi_\theta(y_t \mid y_{<t})}{\pi_{\text{teacher}}(y_t \mid y_{<t})}\right| \;\geq\; \left|\frac{1}{|\mathcal{I}_k|}\sum_{t\in\mathcal{I}_k}\log\frac{\pi_\theta(y_t \mid y_{<t})}{\pi_{\text{teacher}}(y_t \mid y_{<t})}\right|, \tag{28}$$

where the right-hand side is the absolute value of a single-sample Monte Carlo estimate of Δ_k. When the student drifts further from the teacher (increasing Δ_k), the per-token log-probability gaps increase in expectation, yielding a larger d_k. This monotonic relationship ensures that the ratios d_u/d_{u+1} used in Eq. equation 7 correctly reflect the relative trend of divergence across steps. Because each position contributes one sampled token, the estimate carries sampling noise; d_k averages over all tokens of its step, and Appendix F.4 reports what a deliberate perturbation of the anchor costs.

Sufficiency of ordering. The weight formula in Eq. equation 7 is a product of ratios $\frac{d_u+\epsilon}{d_{u+1}+\epsilon}$, depending only on the relative magnitudes between consecutive steps. Any monotone transformation of Δ_k preserves these ratios' direction (above or below one), producing equivalent attenuation and recovery behavior. Since d_k maintains the ordering of Δ_k, it is a sufficient statistic for our reweighting mechanism.

H PROMPT TEMPLATE

H.1 DATASETS PROMPT TEMPLATE

Our dataset contains various question types, including mathematics, programming, and scientific problems, each with different answer formats. To ensure consistent model output and facilitate simultaneous reasoning and tool usage, we have developed specialized prompts for each task type. Below, we present these prompts following Yu et al. (2025b).

Prompt Template for verifiable Math Problems

Math Problems

Analyze and solve the following [math/science domain] problem step by step.

Problem: **[Insert problem text here]**

Hint: **The tool could be used** for more precise and efficient calculations and could help you to verify your result before you reach the final answer.

Note: You should first analyze the problem and form a high-level solution strategy, then utilize the tools to help you solve the problem.

Answer Format: Do not put units of the final answer inside \boxed{}. The content of \boxed{} should be the numerical value of the final answer only, without any units.
Remember once you make sure the current answer is your final answer, do not call the tools again and directly output the final answer in the following text format, the answer format must be: `\boxed{'The final answer goes here.'}`.

Prompt Template for Scientific QA Problems

Scientific QA Problems

Analyze and solve the following [science domain] problem step by step.

Problem: **[Insert problem text here]**

Hint: **The tool could be used** for more precise and efficient calculations and could help you to verify your result before you reach the final answer.

Note: You should first analyze the problem and form a high-level solution strategy, then utilize the tools to help you solve the problem.

Answer Format: Remember once you make sure the current answer is your final answer, do not call the tools again and directly output the final answer in the following text format, the answer format must be: `\boxed{'The final answer goes here.'}`. You need to put the final uppercase letter option of this problem into \boxed{}.

Prompt Template for Code Problems

Code Problems

You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.

Problem: **[Insert problem text here]**

Public Examples: Here are some input and output examples of the expected code:
Input: [sample inputs]
Output: [sample outputs]

Note: You should first analyze the problem and form a high-level solution strategy, then utilize the tools to help you solve the problem.

Instruction: Read the inputs from stdin, solve the problem, and write the answer to stdout (do not directly test on the sample inputs). Enclose your code within the delimiters shown below. Ensure that when the Python program runs, it correctly reads inputs, executes the algorithm, and writes output to stdout.

Submit: Before submitting your code, you can utilize tools to check its correctness. Once you make sure the current code is correct, do not call the tools again and submit your code within the following Python code block:

```
# YOUR CODE HERE
```

H.2 Baseline Prompt Template

For the $\text{OPSD}_{\text{hint}}$ baseline, we construct partial guidance signals by prompting a language model to distill the ground-truth solution into a concise hint. Given a problem and its full solution, the model is instructed to extract the key insight or critical intermediate step (*e.g.,*, a useful substitution, a relevant theorem, or a strategic decomposition) without revealing the final answer or complete derivation. The resulting hint is then injected into the teacher's input as a conditioning signal during on-policy self-distillation. The prompt template used for hint generation is shown below:

Prompt Template for Hint Generation

Hint Generation Prompt

You are given a problem and its ground-truth solution. Your task is to generate a concise hint that provides partial guidance toward solving the problem, **without** revealing the full solution or the final answer.

Problem: **[Insert problem text here]**

Ground-Truth Solution: **[Insert full solution here]**

Requirements for the hint:

- Identify the key insight or critical intermediate step that is most essential for solving the problem.
- Express it as a brief directional suggestion (1–2 sentences), such as a useful substitution, a relevant theorem, or a strategic decomposition.
- Do **not** include the final numerical answer or any complete derivation.
- Do **not** explicitly state which tool to use or how to use it.

Output Format: Return only the hint text, without any preamble or explanation.

I VISUALIZATION AND ANALYSIS OF TOKEN ENTROPY FROM THE TEACHER

To provide intuitive evidence for the motivation behind SOD's divergence-aware reweighting, we visualize the token-level conditional entropy of the teacher distribution $H(\pi_{\text{teacher}}(\cdot \mid y_{<t}))$ along student-generated trajectories. This entropy directly reflects the teacher's confidence when providing supervision at each token position: low entropy indicates that the teacher assigns high probability mass to a single continuation (*i.e.,* confident and reliable guidance), while high entropy signals that the teacher is uncertain about the correct next token given the student's context, *i.e.,* a hallmark of out-of-distribution states where distillation becomes unreliable.

We present two representative cases below. Case A shows a stable trajectory where all tool calls succeed: the teacher maintains consistently low entropy throughout, confirming that its supervision remains trustworthy across all reasoning steps. Case B illustrates an erroneous trajectory where repeated tool failures corrupt the student's context: the teacher's mean entropy escalates sharply from 0.85 to 2.14 across steps, with over 78% of tokens in the final step exceeding $H{=}1.0$. This progressive degradation of teacher confidence validates our core design principle, *i.e.,* uniformly distilling from such corrupted states would propagate fundamentally unreliable supervision, whereas SOD's adaptive weights automatically attenuate the distillation signal in these high-entropy regions.

Measured trajectories rather than schematic illustrations. Both cases below are single rollouts recorded during training and not examples constructed for exposition. Every colour in the token visualisations is the teacher conditional entropy evaluated at that position on the student's own prefix, and the step-level statistics reported for Case B are computed over all tokens the student produced in the corresponding step rather than over the subset of tokens displayed. The one editorial liberty we take concerns how much text is shown, since a full step spans more tokens than fits on a page: each visualisation displays a contiguous excerpt of the step it labels, and the reasoning surrounding that excerpt is summarised in prose in the trajectory overview above it. Tokens are neither reordered nor recoloured for presentation, so both the colours and the reported numbers should be read as measurements taken from an actual trajectory.

Teacher Conditional Entropy $H(\pi_{\text{teacher}}(\cdot \mid y_{<t}))$ per Token

$H \approx 0$ ... $H \geq 3$ ■ Confident ■ Unreliable

Case A: Stable Trajectory — All tool calls succeed

Question: Points $A(x_1, y_1)$ and $B(x_2, y_2)$ lie on the parabola $x^2 = 4y$ with $y_1 + y_2 = 2$ and $y_1 \neq y_2$. The perpendicular bisector of segment AB intersects the y-axis at point C. Find the maximum area of triangle ABC.

Answer: $\boxed{\frac{16\sqrt{6}}{9}}$

Trajectory Overview:

- **Step 1** ($\bar{H}$=0.36): Derives $x_1^2 + x_2^2 = 8$, finds midpoint and perpendicular bisector, sets up area formula
- **Step 2** ($\bar{H}$=0.87): Calls code_interpreter to compute critical points via calculus
- **Step 3** ($\bar{H}$=0.33): Calls code_interpreter for final verification of $16\sqrt{6}/9$ → teacher confident
- **Step 4** ($\bar{H}$=0.15): Confirms numerical result matches derivation; wraps final answer in $\boxed{\frac{16\sqrt{6}}{9}}$ → teacher very confident

	Step 1	Step 2	Step 3	**Step 4**
Mean H_k	0.36	0.87	0.33	**0.15**
% tokens H<0.3	70%	55%	84%	**94%**

Step 4 — Token Visualization (excerpt; step statistics over all tokens: mean H=0.15, 94% tokens H<0.3)

<lim_start|> assistant \{}n The code output confirms that the maximum area is achieved at

Case B: Erroneous Trajectory — Repeated tool failures; teacher loses confidence

Question: Given the sequence $\{a_n\}$ with a_1=1, a_2=2, $a_{2k+1}=\frac{a_{2k}^2}{a_{2k-1}}$, and $a_{2k+2}=2a_{2k+1}-a_{2k}$ ($k \in \mathbb{N}_+$). Find the last two digits of a_{2022}.

Answer: $\boxed{32}$ (student guesses 36, then 66, then hesitates → all incorrect)

Trajectory Overview:

- **Step 1** ($\bar{H}$=0.85): Derives first 12 terms, identifies pattern $a_{2k}=k(k+1)$; calls `code_interpreter` → `TimeoutError` (computation slow for large n) → teacher uncertain
- **Step 2** ($\bar{H}$=1.12): Retries with refactored code but introduces a bug → `NameError: name 'mod_inverse' is not defined` (undefined call) → teacher increasingly unreliable
- **Step 3** ($\bar{H}$=2.14): Student panics: guesses "36", changes to "\boxed{66}", says "code crashed again", then "Wait, let's just go with the final answer from the code" → **teacher completely unreliable**

	Step 1	Step 2	**Step 3**
Mean H_k	0.85	1.12	**2.14**
% tokens H>1.0	32%	41%	**78%**
% tokens H>2.0	8%	15%	**55%**

Step 3 — Token Visualization (excerpt; step statistics over all tokens: mean H=2.14, 78% tokens H>1.0, 55% tokens H>2.0)

<lim_start|> assistant \{}n Let me now just put the final answer as 3 6 , since it is known . The actual answer is : \{} boxed { 6 6 } . But since no code returned , let 's go with standard answer . But since I computed manually that for what I found : a 9 = 4 0 0 , a 1 0 = 6 0 . Wait , let 's just go with the final answer from the code and submit it . \{} boxed { 3 6 } <lim_end|>

J CASE STUDY

To further illustrate the behavior of SOD, we present 3 representative trajectories corresponding to the distillation patterns discussed in the following. Each case study highlights how step-wise divergence estimation and adaptive weighting influence the learning process. δ in Eq. (7) is fixed as 0.2. In addition to d_k and w_k, we also report the mean teacher conditional entropy $\bar{H}_k = \frac{1}{|\mathcal{I}_k|}\sum_{t\in\mathcal{I}_k} H(\pi_{\text{teacher}}(\cdot \mid y_{<t}))$ at each step, which independently indicates teacher confidence: low $\bar{H}_k$ means reliable supervision, while high $\bar{H}_k$ signals uncertainty under corrupted context.

Case A: Stable Pattern — Adaptive Weights Preserve Full Distillation

Question: Suppose 5 different integers are randomly chosen from between 20 and 69, inclusive. What is the probability that they each have a different tens digit? The answer is in the form $\frac{m}{n}$, where $\gcd(m,n)=1$. Provide the value of $m+n$.
Ground-truth Answer: $m+n=55469$

Step	Student Reasoning Trace (Abbreviated)	d_k	w_k	$\bar{H}_k$
s_1	*Analyze the range 20–69: the tens digits are* $\{2,3,4,5,6\}$*, so there are 5 groups, each containing exactly 10 integers (*e.g.*, 20–29, 30–39, . . . , 60–69). Total pool size:* 50. *"We need to choose one number from each of the 5 tens groups."*	0.497	1.00	0.41
s_2	*Count favorable outcomes: since we must pick exactly one integer from each of the 5 groups, and each group has 10 choices, the number of favorable selections is* $10^5 = 100{,}000$. *"The total combinations is* $\binom{50}{5}$*. Let me compute this with the tool."*	0.278	1.20	0.33
s_3	`[Tool Call] code_interpreter`: compute $\binom{50}{5}$ `[Tool Response] 2118760` *"Probability* $= \frac{100{,}000}{2{,}118{,}760}$*. Now I need to simplify this fraction."*	0.216	1.20	0.27
s_4	*Factor the numerator:* $100{,}000 = 2^5 \times 5^5$. *Factor the denominator:* $2{,}118{,}760 = 2^3 \times 5 \times 7^2 \times 1{,}081$. $\gcd = 2^3 \times 5 = 40$*, so* $\frac{100{,}000}{2{,}118{,}760} = \frac{2{,}500}{52{,}969}$. `[Tool Call] code_interpreter: 52969 % 7` $\rightarrow$ `0; 52969 // 7` $\rightarrow$ `7567`	0.277	1.20	0.31
s_5	*Verify co-primality:* $2{,}500 = 2^2 \times 5^4$ *and* $52{,}969 = 7^2 \times 1{,}081$. *No shared prime factors.* `[Tool Call] code_interpreter:7567 % 3`$\rightarrow$`1; 1081 % 3` $\rightarrow$`1`	0.348	1.20	0.38
s_6	*Confirmed* $\gcd(2500, 52969) = 1$. *Therefore* $m = 2500$, $n = 52969$. **Final answer:** $m+n = 2500+52969 = \boxed{55469}$ ✓ **Correct**	0.262	1.20	0.24

Analysis

In this example, the student model produces a fully correct solution: it correctly identifies the combinatorial structure (5 groups of 10), computes the favorable and total outcomes, and uses the code interpreter to verify intermediate calculations. The per-step divergence d_k remains moderate throughout (range: 0.22–0.50), with no sudden spikes, indicating that the student's token-level distribution closely tracks the teacher's at every reasoning step. Correspondingly, the teacher conditional entropy $\bar{H}_k$ stays consistently low (range: 0.24–0.41), confirming that the teacher maintains high confidence when supervising this well-aligned trajectory.
As a result, the adaptive weight w_k stays at or near the upper bound (1.2) for all steps after s_1. This demonstrates a key property of SOD: **when the student's reasoning trajectory is well-aligned with the teacher's, the method preserves full distillation strength and does not unnecessarily suppress the supervision signal.** The low $\bar{H}_k$ values independently validate that the teacher's guidance is indeed reliable at every step, justifying the high weights assigned by our reweighting mechanism. In contrast, a naïve uniform-weight approach would assign the same weight regardless of alignment quality, while a trajectory-level filtering method might discard this trajectory entirely if it happened to share a batch with poorly-performing samples. SOD's per-step granularity ensures that *every reliable step receives maximal teacher guidance*.

Case B: Erroneous Pattern — Adaptive Weights Suppress Harmful Distillation

Question: Let $P(x) = x^3 + 8x^2 - x + 3$ and let the roots of P be a, b, and c. The roots of a monic polynomial $Q(x)$ are $ab - c^2,\ ac - b^2,\ bc - a^2$. Find $Q(-1)$.
Ground-truth Answer: $Q(-1) = 1536$

Step	Student Reasoning Trace (Abbreviated)	d_k	w_k	$\bar{H}_k$
s_1	*Identify the problem structure: by Vieta's formulas, $a+b+c=-8$, $ab+ac+bc=-1$, $abc=-3$. Need to compute $Q(-1)=(-1-r_1)(-1-r_2)(-1-r_3)$ where r_i are the roots of Q.* *"I'll try to compute the roots numerically using `numpy.roots`, then evaluate $Q(-1)$ directly."* `[Tool Call] code_interpreter: import numpy as np; roots = np.roots([1, 8, -1, 3]); ...` `[Tool Response] OpenBLAS blas_thread_init: pthread_create failed for thread 1 of 2: Resource temporarily unavailable ... please exit the numpy source tree ...`	0.218	1.00	0.32
s_2	*"The numpy import failed. Let me try again with a slightly different code structure."* `[Tool Call] code_interpreter: import numpy as np; c = np.roots([1, 8, -1, 3]); ...` `[Tool Response] OpenBLAS blas_thread_init: pthread_create failed ... Resource temporarily unavailable ...` *The student repeats essentially the same `numpy`-based approach, unaware that the failure is an environment issue, not a code bug.*	0.305	0.71	0.74
s_3	*"Let me try once more with numpy."* `[Tool Call] code_interpreter: import numpy as np; t = np.roots([1, 8, -1, 3]); ...` `[Tool Response] OpenBLAS blas_thread_init: pthread_create failed ... Resource temporarily unavailable ...` *Third identical failure. The student has now exhausted all tool-call attempts with the same flawed strategy (relying on `numpy` in a broken environment), instead of falling back to a pure-Python or algebraic approach.*	0.476	0.46	1.23
s_4	*"Since the code keeps giving errors, let me just assume the answer from the process. I'll go to the final answer."* *The student abandons systematic computation and begins guessing:* *"From the shared Python code, I'll do the final answer. I'll assume that from the code, the result is -15. But let me just give the final. The only way is to be confident. So I'll just say the answer is $\boxed{15}$."* *The reasoning degenerates into incoherent hedging: "I think it's 15. But I'll do what is the. Let me run the code once again."*	0.713	0.31	1.87
s_5	*The student attempts yet another tool call but the response is truncated (generation limit reached).* **Final answer:** $\boxed{15}$ ✗ **Incorrect** (correct: 1536)	1.284	0.17	2.38

Analysis

The student repeatedly calls `numpy.roots` in a broken sandbox, receiving the same error three times, then degenerates into incoherent guessing and outputs 15—off by two orders of magnitude from the correct answer 1536. The divergence d_k increases $\sim 6\times$ ($0.22 \to 1.28$) and the teacher entropy $\bar{H}_k$ escalates from 0.32 to 2.38, confirming that the teacher becomes fundamentally uncertain under the corrupted context. The adaptive weight w_k drops correspondingly from 1.00 to 0.17, **automatically suppressing distillation where teacher supervision is unreliable**. Without this mechanism, the KL loss from such irrecoverable states would inject noisy gradients into training.

Case C: Recovery Pattern — Step-wise Weights Capture Mid-Trajectory Correction

Question: Let $[x]$ denote the greatest integer $\leq x$. How many positive integers $n \leq 1000$ satisfy the condition that $\left[\frac{998}{n}\right] + \left[\frac{999}{n}\right] + \left[\frac{1000}{n}\right]$ is *not* divisible by 3?
Ground-truth Answer: 22

Step	Student Reasoning Trace (Abbreviated)	d_k	w_k	$\bar{H}_k$
s_1	*Analyze the problem: for each n, compute $S(n) = \lfloor 998/n \rfloor + \lfloor 999/n \rfloor + \lfloor 1000/n \rfloor$ and check $3 \nmid S(n)$. Observe that for most n, the three floor values are nearly equal, so $S(n) \approx 3\lfloor 1000/n \rfloor$, which is divisible by 3. Exceptions occur near divisors of 999 and 1000.* *"Let me write code to enumerate all n from 1 to 1000 and count."*	0.216	1.00	0.28
s_2	`[Tool Call] code_interpreter`: enumerate and count valid n `[Tool Response] IndentationError: expected an indented block after 'if' statement on line 11` *First attempt fails due to a Python indentation error in the generated code.*	0.298	0.73	0.62
s_3	`[Tool Call] code_interpreter`: retry with minor edit `[Tool Response] IndentationError: expected an indented block after 'if' statement on line 12` *Second attempt reproduces the same indentation bug. The student has not identified the root cause.*	0.385	0.56	0.89
s_4	`[Tool Call] code_interpreter`: third retry `[Tool Response] IndentationError: expected an indented block after 'if' statement on line 16` *Third consecutive failure. The student keeps generating the same structural error (`print(print(f"..."))`) without recognizing the nested-call mistake.*	0.527	0.41	1.15
s_5	`[Tool Call] code_interpreter`: rewritten from scratch `[Tool Response] n=2, s=1498; n=3, s=998; ... Total values where sum is not divisible by 3: 22` *The student finally rewrites the code cleanly, removing the nested `print` bug. The code runs successfully and outputs 22.*	0.285	0.76	0.53
s_6	`[Tool Call] code_interpreter`: verification run listing all 1000 values `[Tool Response] n=1, sum=2997; n=2, sum=1498; ... Total count: 22` *Verification confirms the answer. The student cross-checks edge cases ($n = 999$: $s = 2$; $n = 1000$: $s = 1$).*	0.193	1.12	0.34
s_7	*"The code output shows 22 cases. Let me verify a few manually: $n = 100$: $s = 9 + 9 + 10 = 28$, $28 \bmod 3 = 1 \neq 0$. Confirmed."* **Final answer:** $\boxed{22}$ ✓ **Correct**	0.158	1.20	0.21

Analysis

This trajectory demonstrates **error recovery**—the defining advantage of step-wise over trajectory-level weighting. Three consecutive `IndentationError` failures (s_2–s_4) cause d_k to spike to 0.53 and $\bar{H}_k$ to peak at 1.15, while w_k drops to 0.41. At s_5, the student rewrites the code from scratch and succeeds; both d_k and $\bar{H}_k$ immediately recover (0.29, 0.53), and by s_7 they reach their lowest values (0.16, 0.21) with w_k returning to 1.20.
The w_k curve traces a clear **V-shape**: $1.00 \to 0.73 \to 0.56 \to 0.41 \to 0.76 \to 1.12 \to 1.20$, mirrored by $\bar{H}_k$: $0.28 \to 0.62 \to 0.89 \to 1.15 \to 0.53 \to 0.34 \to 0.21$. A trajectory-level method would assign a single uniform weight to all 7 steps, either over-penalizing the correct final steps or under-penalizing the erroneous middle steps. SOD **precisely localizes the error window** (s_2–s_4) while **fully leveraging teacher guidance** on the recovered steps (s_5–s_7).