SWE-OPD: AN EMPIRICAL STUDY OF ON-POLICY DISTILLATION FOR AGENTIC CODING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we study on-policy distillation (OPD) for transferring agentic coding capabilities from a large teacher model to a smaller student model. Prior work has focused primarily on reasoning and short-horizon agentic tasks. We present SWE-OPD, a minimal OPD study for software engineering, where the student interacts with executable repositories and the teacher supervises the prefixes visited by the student. Across two teacher–student pairs, two training sources, and three SWE benchmarks, we find that on-policy supervision transfers coding capability: supervision from Qwen3.6-27B increases the SWE-bench Pro resolve rate of Qwen3.5-9B from 35.9% to 39.4%. Refreshing trajectories is not strictly required either, as an offline variant with cached teacher scores still improves over the student baselines while cutting end-to-end GPU-hours by 59.4%–72.6% and halving peak GPU allocation, at benchmark-dependent accuracy. We also find that a decreasing loss, however, does not necessarily indicate improved agent performance: under top-K reverse-KL loss, one student exhibits severe repetition despite a decreasing loss. We identify a support-mass gradient term as a potential mechanism underlying this failure. Our code is available at `https://anonymous.4open.science/r/SWE-OPD-0D3C`.

1 INTRODUCTION

On-policy distillation (OPD) (Li et al., 2026; Agarwal et al., 2024b; Lu & Lab, 2025; Yang et al., 2026b; Zhao et al., 2026b; Yang et al., 2026c; Song & Zheng, 2026; Wu et al., 2026; Gu et al., 2024) transfers the capabilities of a strong teacher to a student through token-level supervision on trajectories generated by the student. This distinguishes OPD from conventional knowledge distillation and supervised fine-tuning (Gou et al., 2021; Hinton et al., 2015; Ye et al., 2025), which typically rely on fixed, teacher-generated responses. Accordingly, OPD has recently attracted growing attention, with applications ranging from reasoning (Li et al., 2026; Xing et al., 2026; Yang et al., 2026b;a; Hou et al., 2026; Wu et al., 2026) to relatively simple agentic tasks such as web search (Zhou et al., 2026; Wang et al., 2026c;b) [1]. Nevertheless, the effectiveness of OPD in long-horizon tasks that require interaction with an environment, such as software engineering (SWE), remains underexplored.

However, dense supervision alone does not guarantee that OPD will improve SWE agents. An erroneous action can modify repository contents or alter the observations that inform subsequent decisions, requiring the teacher to score histories that differ from those generated under its own policy. Moreover, long interaction histories make teacher scoring costly. Practical approximations, such as restricting scored candidates or training on fixed trajectories, change how supervision is used or where it is applied. These considerations make SWE a demanding setting for examining both the effectiveness and the limitations of on-policy teacher supervision. Recent analyses of OPD also highlight the importance of reliable supervision over long trajectories (Li et al., 2026).

We investigate these questions through SWE-OPD, a minimal distillation setup for coding agents. A *small* student interacts with executable repositories, while a *larger* teacher scores the student-visited prefixes. We compare a top-K reverse-KL loss with a sampled-token objective to examine

[1] In industry, OPD is viewed as an effective capability-merging method (Team, 2026; 2025; Xiaomi, 2026; Ma et al., 2026; GLM, 2026), enabling the integration of specialized capabilities from multiple teachers (usually derived from reinforcement learning) into a single student of the *same* size.

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.

whether the choice of candidate-level supervision affects task performance and training behavior. For simplicity, we use vanilla OPD and retain valid unresolved and budget-truncated trajectories, allowing us to evaluate distillation without requiring successful student demonstrations for every training example.

We further examine whether effective distillation requires continually refreshed student trajectories. Following the offline approach explored in Lightning OPD (Wu et al., 2026), Offline SWE-OPD collects trajectories from the initial student and precomputes teacher supervision before optimization. This decouples rollout collection, teacher scoring, and student training, allowing the stages to share a GPU pool. The fixed offline trajectory distribution may diverge from the distribution induced by the evolving student policy. Hence, it is necessary to evaluate the offline variant through both downstream resolve rates and end-to-end resource use.

Across two teacher-student pairs, two training sources, and three SWE benchmarks, our experiments demonstrate that vanilla OPD can improve instruction-tuned coding agents. For example, distillation on ScaleSWE with Qwen3.6-27B as the teacher increases the SWE-bench Pro resolve rate of Qwen3.5-9B from 35.9% to 39.4%. In our internal-data experiments, offline SWE-OPD reduces end-to-end GPU-hours by 59.4% for Qwen and 72.6% for GLM while also bringing improvement. These results indicate a favorable trade-off between task performance and computational cost under the evaluated training budgets. Our experiments also reveal a limitation of evaluating distillation solely through its training objective. Although the top-K loss improves the Qwen student, the GLM student develops severe repetition even as its training loss decreases. In contrast, sampled-token training remains stable in the evaluated configuration. An analysis of the implemented top-K loss identifies a support-mass gradient that can shift probability outside the scored candidates, even at teacher-student agreement. This provides a potential failure mechanism for the top-K loss. Together, these findings motivate evaluating SWE distillation through task success, training behavior, and resource cost, and distinguishing supervision coverage from the reliability of the resulting updates. Our contributions are summarized:

- **Capability transfer in agentic coding.** Our controlled study of long-horizon coding shows that teacher supervision on student-visited prefixes can transfer capabilities across model scales, improving coding agents without reinforcement learning or filtering trajectories by task success.
- **Learning without trajectory refresh.** We show that online trajectory refresh is not required for improvement. Using cached teacher supervision on fixed trajectories collected from the initial student halves GPU-hours while maintaining satisfactory performance relative to online training.
- **Reliability beyond loss reduction.** We demonstrate that optimizing the top-K loss can worsen alignment with the teacher even as the loss decreases. It exposes a structural risk of shifting probability outside scored candidates, inspiring furture works on reconsidering OPD objectives.

Scope of this study. This work is designed as a controlled study of vanilla OPD for SWE. We intentionally avoid combining OPD with reinforcement learning, data filtering, or specialized optimization techniques, allowing us to examine whether on-policy teacher supervision alone can transfer agentic coding capabilities across model scales. Therefore we believe that there remains room for further improving student performance, and we release our code to facilitate future exploration.

2 Related Work

Classical supervision learning is trained only on teacher-visited prefixes (Kim & Rush, 2016), while at test time the student must condition on its own, so errors compound along the trajectory. On-policy distillation (OPD) closes this gap by sampling trajectories from the student and letting the teacher score them, inheriting the on-/off-policy distinction from policy distillation for RL agents (Rusu et al., 2016). Lin et al. (2020) first brought DAgger-style (Ross et al., 2011) rollouts to autoregressive distillation; MiniLLM (Gu et al., 2024) minimises sequence-level reverse KL and shows it is equivalent to a policy gradient whose reward is the teacher–student log-ratio; GKD (Agarwal et al., 2024a) unifies the resulting design space over data mixtures and divergences.

Compared with reinforcement learning from verifiable rewards, where an entire rollout collapses into a single scalar (Shao et al., 2024; Guo et al., 2025), the appeal of OPD is the density of its signal: every sampled token receives gradient, which buys substantial compute savings at comparable accuracy (Lu & Lab, 2025). Existing methods vary along three axes (Song & Zheng, 2026): the divergence (JSD in GKD (Agarwal et al., 2023), skewed KL in DistiLLM (Ko et al., 2024), re-

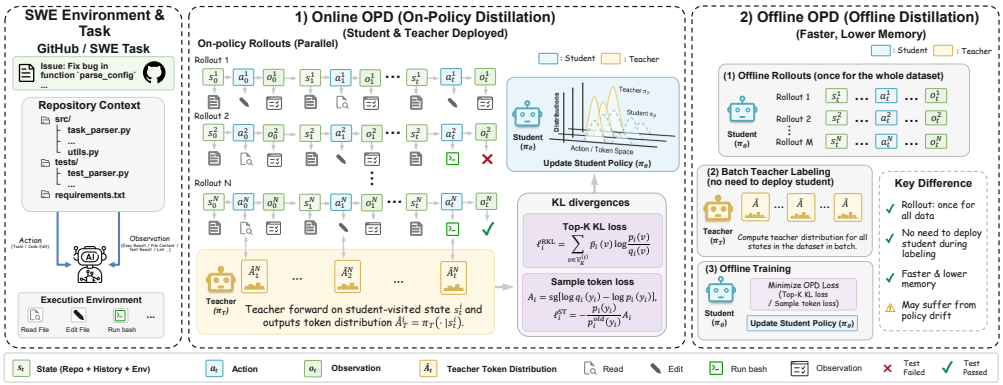


Figure 1: **Study design of SWE-OPD**. **Left:** the student interacts with an executable repository; tool observations enter the conditioning context, while only student-generated tokens receive supervision. **Right:** The two design axes we compare are the *trajectory source* (current student in SWE-OPD vs. initial student in Offline SWE-OPD) and the *teacher information* used per position (top-K candidates in the RKL loss vs. the sampled token in the ST objective).

verse KL where mode-seeking behaviour helps on reasoning tasks); the granularity of teacher access (the log-probability of the single sampled token, the teacher's top-K, or the full vocabulary); and whether the teacher is an external model or the student itself under privileged context (Zhao et al., 2026b).

However, the reported gains are nonetheless concentrated on reasoning and relatively simple agentic tasks, such as competitive mathematics. In this work, we extend OPD to more complex agentic coding scenarios, where solving tasks requires long-horizon planning, iterative tool use, and sustained interaction with execution environments. We investigate whether OPD can effectively transfer capabilities from stronger teacher models to smaller coding agents under these more challenging settings.

3 Preliminaries

We consider SWE tasks as sequential decision-making problems in executable software environments. A task is specified by $x = (\mathcal{I}, \mathcal{R})$, comprising an issue $\mathcal{I}$ and a repository $\mathcal{R}$. At turn t, s_t denotes the full state of environment $\mathcal{E}_{\mathcal{R}}$, including the repository contents and execution context, which is only partially observable. The goal is to produce a patch to fix the issue.

The agent policy π_θ generates an assistant response a_t conditioned on the task prompt and the interaction history. Each response may contain reasoning, tool invocations, or a final answer. The environment executes the tool invocations in a_t against the current state s_t, returns an observation $o_t = \mathcal{E}_{\mathcal{R}}(s_t, a_t)$ such as file contents or test logs, and transitions to s_{t+1}. The interaction terminates when the agent submits its result or exhausts a predefined interaction budget, yielding the trajectory $\tau = (x, a_0, o_0, a_1, o_1, \ldots, a_T, o_T)$. We allow o_t to be empty when a_t contains no tool invocation. After termination, the resulting patch is applied to the initial repository and evaluated using the benchmark's required regression tests. A task is considered *resolved* only if all required tests pass. In on-policy distillation, a student generates trajectories and a frozen teacher provides next-token probabilities conditioned on the prefixes visited by the student. The student is then updated using discrepancies between the two models' predictions on these prefixes. Throughout this work, we sample only one single trajectory per training prompt x.

4 SWE-OPD: On-Policy Distillation for SWE

SWE-OPD follows standard multi-turn OPD, conditioning teacher supervision on the student's decisions and observations in executable repositories (Figure 1). We define rollout collection (Section 4.1), frozen-teacher scoring (Section 4.2), and two objectives with different amounts of candidate-level teacher information (Section 4.3), then empirically evaluate end-to-end issue resolution.

4.1 Rollout in Executable Repositories

For each task x, the current student π_θ generates a repository trajectory following Section 3. Its actions determine the visited states and observations in subsequent prefixes used for teacher scoring. To study vanilla OPD without additional optimization strategies, we retain all valid rollouts, including unresolved and budget-truncated episodes, and leave selective data curation to future work.

4.2 Teacher Scoring on Student-Visited Prefixes

The frozen teacher provides next-token probabilities on the prefixes visited by the student. Importantly, The teacher and student condition on the same observable prefix, comprising the issue description, previous responses, and tool outputs. For token-level supervision, we represent the responses and observations in τ following x as a single token sequence $\mathbf{y} = (y_1, \ldots, y_L)$ in interaction order. Let $\mathcal{G}(\tau)$ denote all positions corresponding to student-generated response tokens, and define the generation mask as $m_i = \mathbb{1}[i \in \mathcal{G}(\tau)]$. At a generated position i within response a_t, the prefix $(x, \mathbf{y}_{<i})$ contains the task context, preceding responses and observations, and earlier tokens in the current response. Our implementation assumes that the student and teacher share a tokenizer and token-ID mapping, with vocabulary $\mathcal{V}$. We define their next-token probabilities on the shared prefix:

$$p_i(v) = \pi_\theta(v \mid x, \mathbf{y}_{<i}), \qquad q_i(v) = \pi_T(v \mid x, \mathbf{y}_{<i}). \tag{1}$$

We apply token-level supervision at every student-generated position $i \in \mathcal{G}(\tau)$. Tool outputs, test logs, and retrieved repository contents remain part of the conditioning context but are excluded from the loss by the generation mask m_i. Supervision is therefore dense across generated positions. For training with the top-K reverse KL loss, we retain teacher log-probabilities for the current student's K highest-probability candidates, rather than teacher scores over the full vocabulary:

$$\mathcal{V}_K^{(i)} = \mathrm{TopK}(p_i, K)\ . \tag{2}$$

Retaining K teacher scores per generated position gives $O(|\mathcal{G}(\tau)|K)$ scores per trajectory, reducing storage and communication for coding histories that can exceed $128k$ tokens. The ST loss (Eq. 5) instead uses the teacher score of the generated token y_i, even outside $\mathcal{V}_K$. Both variants supervise the same positions with different candidate-level information (Section 4.3).

4.3 Student Optimization via Distillation

Our goal is to align the student's next-token predictions with the teacher on student-visited prefixes. We consider the top-K reverse KL loss (RKL) and the sampled-token policy-gradient loss (ST). For a batch of B trajectories $\tau_1, \ldots, \tau_B$, the set $\mathcal{G}_B = \{(b, i) : 1 \le b \le B,\ i \in \mathcal{G}(\tau_b)\}$ collects all supervised positions. We average the loss over these positions:

$$\mathcal{L}_{\mathrm{OPD}} = \frac{1}{|\mathcal{G}_B|} \sum\nolimits_{(b,i)\in\mathcal{G}_B} \ell^*_{b,i}, \qquad * \in \{\mathrm{RKL}, \mathrm{ST}\}. \tag{3}$$

We instantiate $\ell^*_{b,i}$ with either the RKL or ST loss below, omitting the batch index b for clarity.

Top-K reverse-KL loss. On the current student's top-K support $\mathcal{V}_K^{(i)}$, we follow the protocol of Li et al. (2026) and normalize student weights as $\bar{p}_i(v) = p_i(v)/\mathrm{Z}_i^p$ for $v \in \mathcal{V}_K^{(i)}$, where $\mathrm{Z}_i^p = \sum_{v\in\mathcal{V}_K^{(i)}} \mathrm{sg}[p_i(v)]$ (sg denotes the stop-gradient operation). The token-level loss is

$$\ell_i^{\mathrm{RKL}} = \sum_{v\in\mathcal{V}_K^{(i)}} \bar{p}_i(v) \log \frac{p_i(v)}{q_i(v)}. \tag{4}$$

This is a surrogate for full-vocabulary reverse KL: the log-ratio uses the original probabilities, while the student weights are normalized over the selected support. The support and its normalization constant are recomputed at each update and held fixed during differentiation; gradients flow through both $\bar{p}_i(v)$ and $\log p_i(v)$. The objective is motivated by the *mode-seeking* intuition of reverse KL: encouraging the student to concentrate probability on continuations favored by the teacher (Gu et al., 2024). A candidate contributes a large positive loss when the student assigns it substantial probability but the teacher assigns it much lower probability. Restricting the sum to the student's top-K candidates focuses this penalty on continuations the student is most likely to generate. This moti-

vates its use in long-horizon coding, where an erroneous early action can alter the repository state and affect subsequent decisions. [2]

Sampled-token loss. The ST loss uses the teacher–student log-probability ratio at the sampled token as a token-level advantage, following Ma et al. (2026) and Li et al. (2026). Let $p_i^{\text{old}}(v) = \pi_{\theta_{\text{old}}}(v \mid x, \mathbf{y}_{<i})$ denote the frozen student policy used to collect the rollout. For $y_i \sim p_i^{\text{old}}$, we define

$$A_i = \text{sg}[\log q_i(y_i) - \log p_i(y_i)], \qquad \ell_i^{\text{ST}} = -\frac{p_i(y_i)}{p_i^{\text{old}}(y_i)} A_i. \tag{5}$$

The advantage is recomputed using the current student and held fixed during differentiation. It encourages increasing $p_i(y_i)$ when $q_i(y_i) > p_i(y_i)$, and decreasing it when the inequality is reversed. The token is sampled from the rollout policy; this objective is therefore distinct from setting $K = 1$ in Eq. 4. It requires only the teacher log-probability $\log q_i(y_i)$ at each generated position and estimates the gradient using a single Monte Carlo sample.

4.4 Capability Transfer to Coding Agents

Table 1: Results after training on 5K ScaleSWE queries. Avg@4 is the mean resolve rate (%) over four independent runs on SWE-bench Verified (Jimenez et al., 2024), Pro (Deng et al., 2025), and Multilingual (Zan et al., 2025). Δ is the change from the student baseline in percentage points; bold marks the best score per benchmark. Subsequent benchmark tables use these conventions within each student–teacher pair.[3]

Method	Verified		Pro		Multilingual	
	Score	Δ	Score	Δ	Score	Δ
Student: Qwen3.5-9B Teacher: Qwen3.6-27B						
Student baseline	57.9		35.9		43.0	
Teacher Rollout Distillation	54.8	-3.1	34.5	-1.4	40.5	-2.5
RL (UniAgent (Ding et al., 2026))	59.2	$+1.3$	–		–	
SWE-OPD (RKL)	60.5	$+2.6$	**39.4**	$+3.5$	**45.8**	$+2.8$
SWE-OPD (ST)	**61.7**	$+3.8$	37.9	$+2.0$	45.2	$+2.2$

Does teacher supervision on student-visited prefixes transfer coding capability? Table 1 reports results for Qwen3.5-9B trained with supervision from Qwen3.6-27B on 5K ScaleSWE training queries (Zhao et al., 2026a) [4]. We hold the training queries, agent scaffold, and rollout budget fixed while comparing three training configurations: SFT on teacher rollouts (Teacher Rollout Distillation), OPD with the RKL objective, and OPD with the ST objective. Teacher Rollout Distillation retains unresolved and truncated teacher episodes without reward filtering, matching the trajectory-retention policy of SWE-OPD (Section 4.1). Both OPD objectives improve the student baseline on all three benchmarks: RKL gains 2.6%, 3.5%, and 2.8% on Verified, Pro, and Multilingual, respectively, while ST gains 3.8%, 2.0%, and 2.2%. Teacher Rollout Distillation does not improve the student and degrades it on all three benchmarks (-3.1%, -1.4%, and -2.5%). Under this shared trajectory-retention policy, the comparison examines differences in both the source and form of supervision: teacher-generated tokens on teacher-visited prefixes versus teacher scores on student-visited prefixes—rather than the effect of data curation. The degradation is consistent with distribution shift: fine-tuning on teacher-visited prefixes may distort an instruction-tuned student's distribution more than dense supervision on its own prefixes. Rejection-sampling SFT on resolved teacher trajectories would provide a stronger off-policy baseline and likely recover part of the gap; we leave this comparison to future work. Our results therefore support OPD over off-policy distillation under matched, unfiltered data, without establishing superiority over the available SFT pipeline.

Do the gains persist with another training source and a larger student? We repeat the comparison on approximately 4K internal training tasks (Guo et al., 2026) and report both student–teacher pairs in Tab. 2. For Qwen3.5-9B, RKL again improves all three benchmarks, while ST improves Verified and Pro with little change on Multilingual. For the 106B-parameter GLM-4.5-Air student,

[2] We also consider the forward-KL loss $\ell_i^{\text{FKL}} = \mathbb{E}_{v\sim\bar{q}_i}[\log(q_i(v)/p_i(v))]$ in our preliminary experiments. Its performance is inferior to that of the RKL loss, so we mainly focus on the latter.

[3] The experimental setup, training configurations, and agent scaffold are detailed in Appendix A.

[4] We use instruct models as companies increasingly release fewer base models, though the method also applies to base models. Results may differ from official reports because detailed evaluation settings, including harnesses and scaffolds, are unavailable.

ST improves all three benchmarks, especially with a 3.4% gain on Multilingual. Within the studied budgets, these results indicate that OPD can further improve a large instruction-tuned coding agent, and that the choice of objective matters: RKL training collapses for GLM despite its gains for Qwen, which we examine in Section 6.

Table 2: Results on the internal dataset. We evaluate SWE-OPD with Qwen3.5-9B and GLM-4.5-Air as students, supervised by Qwen3.6-27B and GLM-5.2, respectively.

Method	Verified		Pro		Multilingual	
	Score	Δ	Score	Δ	Score	Δ
Student: Qwen3.5-9B	*Teacher: Qwen3.6-27B*					
Student baseline	57.9		35.9		43.0	
SWE-OPD (RKL)	**61.6**	+3.7	38.4	+2.5	**44.3**	+1.3
SWE-OPD (ST)	61.3	+3.4	**38.7**	+2.8	43.1	+0.1
Student: GLM-4.5-Air	*Teacher: GLM-5.2*					
Student baseline	59.9		35.8		41.6	
SWE-OPD (ST)†	**60.4**	+0.5	**37.1**	+1.3	**45.0**	+3.4

† RKL training on GLM collapses with severe repetition; see Section 6.

4.5 Top-K Selection and Support Analysis

How large must the top-K be? We sweep $K \in \{128, 256, 512, 1024\}$ for one epoch on ScaleSWE, distilling Qwen3.5-9B from Qwen3.6-27B with the RKL loss. Gains over the student baseline persist across all tested support sizes and all three benchmarks, with no consistent benefit from increasing K (Figure 2). Loss trajectories are similar across K and plateau around optimizer step 20, with a comparable trend on internal data (Figure 3(a)). To balance performance and the storage and communication burden, we therefore use $K = 128$ by default for this pair.

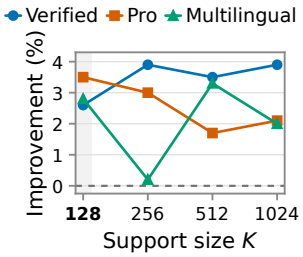


Figure 2: **Top-K ablation.** Gains in percentage points.

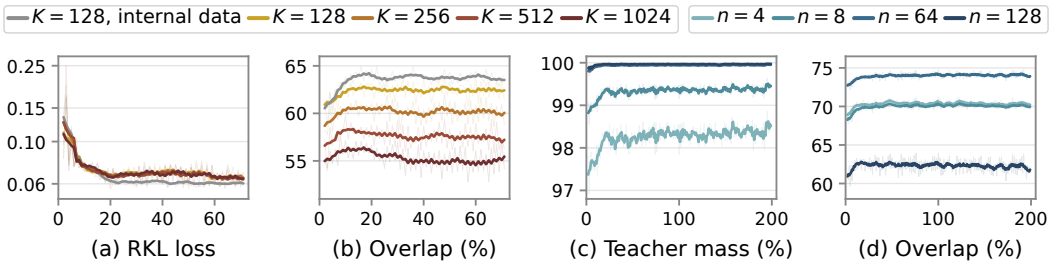


Figure 3: **Top-K selection and nested support diagnostics.** Qwen3.5-9B distilled from Qwen3.6-27B: (a,b) support-size sweep; (c,d) nested readouts for the $K = 128$ ScaleSWE run. All horizontal axes show training steps. In (b), mean overlap over steps 62–71 is 62.5% at $K = 128$ and 55.0% at $K = 1024$.

Does a compact support lose teacher information? For the $K = 128$ ScaleSWE run, we examine the top-n token subsets selected by the student, where $n \in \{4, 8, 64, 128\}$ (Figure 3(c,d)). Averaged over steps 190–199, the student's top-4, top-64, and top-128 capture 98.4%, 99.95%, and 99.97% of the *teacher* probability mass, respectively; all coverage numbers in this paper are teacher mass measured on student-selected supports, not student mass. Yet teacher–student top-128 overlap averages only 62.1% over the same interval (Figure 3(d)). This overlap increases modestly early in training and then plateaus. For $n \in \{4, 8, 64\}$, overlap is computed using teacher rankings restricted to the top-128 candidates selected by the student; for $n = 128$, teacher rankings span the full vocabulary. Teacher candidates outside the student-selected support thus carry little probability mass on average. Incomplete token-set overlap need not imply substantial loss of teacher probability mass, so overlap and coverage should be treated as distinct quantities. Together with the benchmark gains, these diagnostics suggest that compact student-selected candidate sets can support effective distillation in the evaluated setting.

5 OFFLINE SWE-OPD

Online SWE-OPD updates both the student parameters and the interaction histories used for subsequent supervision. Inspired by Lightning OPD (Wu et al., 2026), Offline SWE-OPD freezes the latter by collecting trajectories from the initial student before optimization, and by scoring them with the teacher once. This design eliminates repeated teacher scoring to decouple student-teacher interaction. However, this also introduces a mismatch between the evolving student and the fixed training histories. We therefore treat offline training as an approximation to be evaluated.

5.1 OFFLINE SUPERVISION CONSTRUCTION

We collect trajectories over the training set using the fixed initial policy π_{θ_0}, following the repository interaction process in Section 4.1. For each trajectory, we save the task identifier, token sequence $\mathbf{y}$, and generation mask $\mathbf{m}$ defined in Section 4.2. For ST training, we also save the initial student's log-probability $\log \pi_{\theta_0}(y_i \mid x, \mathbf{y}_{<i})$ at each generated position.

The frozen teacher evaluates q_i on each saved student-generated prefix, as defined in Eq. 1. To avoid storing full-vocabulary values, we retain the teacher top-K_{cache} support

$$\mathcal{V}^{(i)}_{T,K_{\text{cache}}} = \text{TopK}(q_i, K_{\text{cache}}), \qquad K_{\text{cache}} = 1024. \tag{6}$$

where $\mathcal{V}^{(i)}_{T,K_{\text{cache}}}$ denotes a teacher-defined token support, not a score. For each supervised position, the offline buffer stores the token IDs and original log-probabilities $\log q_i(v)$ on this support. For example, $q_{i,(K_{\text{cache}})}$ denotes the smallest teacher probability in the cache. Teacher scoring is performed once, and the resulting cache can be reused across multiple student-training epochs.

5.2 STUDENT OPTIMIZATION ON FIXED PREFIXES

During optimization, we evaluate the current student p_i on the saved prefixes and apply the objectives from Section 4.3 using cached teacher scores. For any token queried by either objective, RKL or ST, we use the following teacher-score lookup:

$$\widetilde{q}_i(v) = \begin{cases} q_i(v), & v \in \mathcal{V}^{(i)}_{T,K_{\text{cache}}}, \\ q_{i,(K_{\text{cache}})}, & v \notin \mathcal{V}^{(i)}_{T,K_{\text{cache}}}. \end{cases} \tag{7}$$

If a token in the student's current top-K is absent from the teacher's cached top-K_{cache}, we assign it the probability of the K_{cache}-th token. Every omitted token satisfies $q_i(v) \le q_{i,(K_{\text{cache}})}$, so the fallback is an upper bound on its unknown teacher probability and avoids additional teacher queries [5].

RKL Loss. For RKL training, we recompute the student top-K support $\mathcal{V}^{(i)}_K$ in Eq. 2 at each update, with $K = 128$. In other words, the student support remains dynamic rather than being frozen to the initial student π_{θ_0}. We substitute $\widetilde{q}_i$ for q_i in Eq. 4, retaining the support-normalized student weights $\bar{p}_i$ with the same stop-gradient on their normalization constant.

ST Loss. For ST training, we use Eq. 5 with $q_i(y_i)$ replaced by $\widetilde{q}_i(y_i)$ and the old policy fixed to the initial student, $p_i^{\text{old}}(v) = \pi_{\theta_0}(v \mid x, \mathbf{y}_{<i})$ so as to be consistent with the fixed trajectories. Although the teacher score is fixed, the advantage changes with the current student:

$$\ell_i^{\text{ST}} = -\frac{p_i(y_i)}{\pi_{\theta_0}(y_i \mid x, \mathbf{y}_{<i})} \,\text{sg}[\log \widetilde{q}_i(y_i) - \log p_i(y_i)] \,. \tag{8}$$

Both objectives use the aggregation in Eq. 3. The saved trajectories and cached scores are constants, and only the current student parameters θ are updated. This allows both forms of distillation to reuse offline supervision across optimization steps.

5.3 PERFORMANCE AND COST OF OFFLINE SUPERVISION

Can fixed initial-student trajectories still support effective distillation? We compare online and offline SWE-OPD trained on the same internal data, with matched student-training budgets (Tab. 3). All evaluated offline configurations improve over their student baselines on all three benchmarks, so freezing the interaction histories does not prevent effective learning within this budget.

[5] A preliminary comparison with larger K_{cache} such as 4096 yielded similar performance. Thus, to reduce the burden of storage, we use 1024 by default.

Table 3: Results of SWE-OPD and Offline SWE-OPD on the internal dataset. We compare both variants for Qwen3.5-9B distilled from Qwen3.6-27B and GLM-4.5-Air distilled from GLM-5.2.

Method	Verified		Pro		Multilingual	
	Score	Δ	Score	Δ	Score	Δ
Student: Qwen3.5-9B Teacher: Qwen3.6-27B						
Student baseline	57.9		35.9		43.0	
SWE-OPD (RKL)	61.6	+3.7	38.4	+2.5	44.3	+1.3
SWE-OPD (ST)	61.3	+3.4	**38.7**	+2.8	43.1	+0.1
Offline SWE-OPD (RKL)	60.2	+2.3	38.0	+2.1	45.3	+2.3
Offline SWE-OPD (ST)	**63.7**	+5.8	36.5	+0.6	**46.0**	+3.0
Student: GLM-4.5-Air Teacher: GLM-5.2						
Student baseline	59.9		35.8		41.6	
SWE-OPD (ST)	**60.4**	+0.5	37.1	+1.3	**45.0**	+3.4
Offline SWE-OPD (ST)	**60.4**	+0.5	**37.5**	+1.7	44.6	+3.0

For Qwen, offline ST achieves the highest Verified and Multilingual scores among the Qwen configurations. Relative to online ST, it gains 2.4% and 2.9% on Verified and Multilingual, respectively, while scoring 2.2% lower on Pro. Offline RKL also gains 1.0 point on Multilingual while scoring 1.4% and 0.4% lower on Verified and Pro than its online counterpart. For GLM-4.5-Air, offline ST matches the reported online score on Verified, gains 0.4% on Pro, and scores 0.4% lower on Multilingual. Together with the resource savings in Section 5.4, these results make cached supervision a promising option for enhancing large students under limited training resources.

5.4 Resource Cost and Its Attribution

What do the cost differences measure? Relative to our online implementation, offline execution reduces end-to-end GPU-hours from 1024 to 416 (59.4%) for Qwen3.5-9B and from 2808 to 768 (72.6%) for GLM-4.5-Air, i.e. reduction factors of 2.46× and 3.65× in Tab. 4. Reusing fixed student trajectories and cached teacher scores eliminates further rollout collection and teacher inference during offline optimization.

Table 4: Training cost on the internal dataset (GPU-hours) for our online and offline implementations.

Method	*Qwen3.5-9B*	*GLM-4.5-Air*
SWE-OPD	1024	2808
Offline SWE-OPD	416	768
GPU-hour reduction	**2.46**×	**3.65**×
Offline SWE-OPD breakdown		
Student rollout	176	368
Teacher scoring	48	160
OPD training	192	240

More importantly, offline execution removes the need to run all stages concurrently. Sequential execution lets the Offline SWE-OPD stages reuse one GPU pool, reducing peak allocation to $\max(N_R, N_T, N_S)$, where the three terms denote GPU allocations for rollout, teacher scoring, and student optimization. In our setup, this halves the peak GPU requirement from 64 to 32 GPUs, making SWE-OPD feasible on substantially smaller clusters. Tab. 4 reports the end-to-end GPU-hours of SWE-OPD together with a breakdown of the offline cost. All measurements are collected on the same internal dataset and hardware, including the cost of failed or retried rollouts.

6 Diagnosing Training Collapse under the top-k reverse-KL Loss

The RKL loss improves the Qwen student, yet GLM-4.5-Air develops severe repetition while ST training remains stable. We examine how the RKL loss converts teacher scores into updates and identify support escape as a potential cause. We then connect this mechanism to the training dynamics observed in the GLM setting.

6.1 How GLM Training Collapses

Figure 4 shows a pronounced separation between loss reduction and useful behavior: repetition rises sharply as the student entropy falls after an initial increase. The loss decreases overall as repetition rises. The model therefore becomes concentrated on repetitive continuations while still improving its training objective. A small distillation loss is insufficient evidence that teacher supervision is improving the student's behavior. In comparison, ST training records no repetition and avoids the sharp terminal entropy decline. Together with its benchmark gains in Tab. 2, this contrast motivates comparing how the two objectives turn teacher scores into student updates.

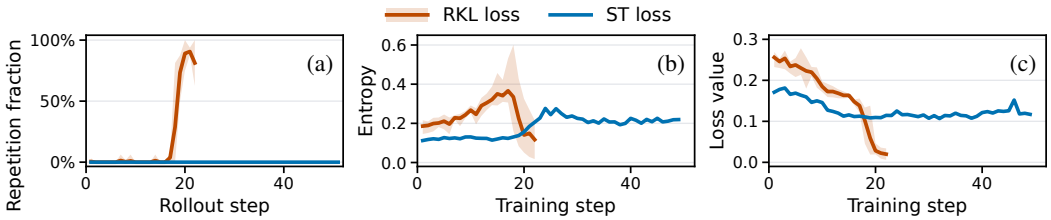


Figure 4: GLM training dynamics: (a) Repetition fraction; (b) student entropy; (c) training loss. RKL and ST training are shown in orange and blue, respectively.

6.2 Support Escape in the Truncated Loss

We now examine what kind of update Eq. 4 performs. Let $\hat{Z}_{\text{total}} = \sum_{v \in \mathcal{V}_K^{(i)}} p_i(v)$ denote the student's total probability mass on the selected top-K support. Differentiating Eq. 4 with respect to the student parameters θ at a fixed prefix and support, gives (see Appendix B.1 for the full derivation):

$$\nabla_\theta \ell_i^{\text{RKL}} = \underbrace{\sum_{v \in \mathcal{V}_K^{(i)}} \bar{p}_i(v) \log \frac{p_i(v)}{q_i(v)} \nabla_\theta \log p_i(v)}_{\mathbf{g}_i\text{: teacher-dependent supervision}} + \underbrace{\nabla_\theta \log \hat{Z}_{\text{total}}}_{\mathbf{h}_i\text{: support-mass gradient}} . \tag{9}$$

Here $\mathbf{g}_i$ carries teacher-dependent supervision, while $\mathbf{h}_i$ is the gradient of the log of the retained probability mass. A sufficiently small step along $-\mathbf{h}_i$ alone decreases $\hat{Z}_{\text{total}}$ whenever $\mathbf{h}_i \neq 0$, shifting probability outside the scored support. We call this decrease in retained mass *support escape*; it occurs under the full update when $\mathbf{g}_i$ does not offset this outward tendency. Even at teacher agreement ($p_i = q_i$) on the retained support, $\mathbf{g}_i = 0$ while $\mathbf{h}_i$ may remain nonzero, so the update need not be stationary at agreement.

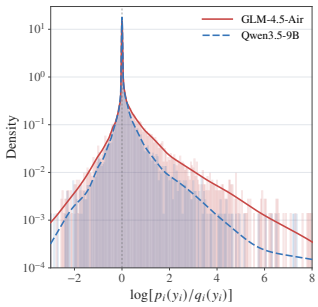


Figure 5: teacher-student probability divergence.

Support Escape and GLM Collapse. Support escape reveals a structural failure mode of the top-K RKL loss. It pushes probability mass from the target $\mathcal{V}_K$ set into the tail, which lack the token-specific teacher correction, thus assigning mass to candidates the teacher considers unlikely. Repeated updates can therefore reduce the loss while moving the student away from the teacher. In our training, these shifts alter subsequent actions and supervision prefixes, carrying the misalignment into future rollouts. This may explain how GLM's loss can continue to decrease as its generations become increasingly repetitive and eventually collapse. We also provide detailed analysis with a mechanism example in Appendix B.

Why the two students Qwen and GLM may differ. Figure 5 shows a heavier positive tail of sampled-action log-ratios for GLM, whereas Qwen is more concentrated near zero. GLM thus more frequently selects actions to which its teacher assigns substantially lower probability. Under the RKL loss, suppressing such actions within the retained support can redirect probability toward unscored tail tokens. We speculate that these stronger conflicts amplify support escape and drive rollout degeneration in GLM, whereas Qwen's closer teacher–student agreement limits this additional pressure and allows effective distillation. This provides a reference for designing efficient objectives that preserve agreement while controlling probability mass outside the scored support.

7 Conclusion & Future Work

We studied vanilla on-policy distillation for agentic coding. Teacher supervision on student-visited prefixes transfers coding capability across model scales. And our offline variant retains gains over the student baselines at 59.4%–72.6% fewer GPU-hours and half the peak GPU allocation, with benchmark-dependent accuracy. We also analyze the GLM failure potentially due to the probability leak to unscored tail tokens though the decreasing top-K reverse-KL loss. These results provide a baseline for studying how teacher supervision can complement outcome rewards and selective data curation. Future work can develop advanced OPD methods, including reinforcement learning that jointly provides process- and outcome-level supervision (Yang et al., 2026a;b); determine which training data are most suitable for OPD; study model families with distinct tokenizers (Wang et al., 2026a; Sun et al., 2026; Patiño et al., 2025; Niu et al., 2026); and evaluate whether OPD can also transfer knowledge effectively during mid-training and pretraining.

AI USE STATEMENT

We used generative AI tools to find related work on on-policy distillation and agentic coding, and to draft and polish descriptions of the SWE-OPD and Offline SWE-OPD pipelines. AI also assisted with deriving and verifying mathematical claims about the RKL loss, ST gradients, and offline teacher-score approximation, and with checking that probability normalization and loss masking were consistent between the formulas and code. The authors reviewed all AI-assisted material and take full responsibility for the paper.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. GKD: generalized knowledge distillation for auto-regressive sequence models. *CoRR*, abs/2306.13649, 2023. doi: 10.48550/ARXIV.2306.13649. URL https://doi.org/10.48550/arXiv.2306.13649.

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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=3zKtaqxLhW.

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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=3zKtaqxLhW.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can AI agents solve long-horizon software engineering tasks? *CoRR*, abs/2509.16941, 2025. doi: 10.48550/ARXIV.2509.16941. URL https://doi.org/10.48550/arXiv.2509.16941.

Yuyang Ding, Bo Wen, Xubo Cao, Zhiqiang Zhai, Guangming Sheng, Xibin Wu, Juntao Li, Min Zhang, and Uni-Agent Contributors. Uni-agent: Build, run, and train agents at scale. https://github.com/verl-project/uni-agent, 2026. GitHub repository. Supervisor: Xibin Wu and Juntao Li.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021. doi: 10.1007/S11263-021-01453-Z. URL https://doi.org/10.1007/s11263-021-01453-z.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=5h0qf7IBZZ.

Chuanzhe Guo, Jingjing Wu, Sijun He, Yang Chen, Zhaoqi Kuang, Shilong Fan, Bingjin Chen, Siqi Bao, Jing Liu, Hua Wu, Qingfu Zhu, Wanxiang Che, and Haifeng Wang. Menvagent: Scalable polyglot environment construction for verifiable software engineering. *CoRR*, abs/2601.22859, 2026. doi: 10.48550/ARXIV.2601.22859. URL https://doi.org/10.48550/arXiv.2601.22859.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng,

Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Hao Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, Tao Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nat.*, 645(8081):633–638, 2025. doi: 10.1038/S41586-025-09422-Z. URL https://doi.org/10.1038/s41586-025-09422-z.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Wenjin Hou, Shangpin Peng, Weinong Wang, Zheng Ruan, Yue Zhang, Zhenglin Zhou, Mingqi Gao, Yifei Chen, Kaiqi Wang, Hongming Yang, Chengquan Zhang, Zhuotao Tian, Han Hu, Yi Yang, Fei Wu, and Hehe Fan. Uni-opd: Unifying on-policy distillation with a dual-perspective recipe. *CoRR*, abs/2605.03677, 2026. doi: 10.48550/ARXIV.2605.03677. URL https://doi.org/10.48550/arXiv.2605.03677.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *CoRR*, abs/2504.07164, 2025. doi: 10.48550/ARXIV.2504.07164. URL https://doi.org/10.48550/arXiv.2504.07164.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In Jian Su, Xavier Carreras, and Kevin Duh (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pp. 1317–1327. The Association for Computational Linguistics, 2016. doi: 10.18653/V1/D16-1139. URL https://doi.org/10.18653/v1/d16-1139.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. Distillm: Towards streamlined distillation for large language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24872–24895. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/ko24c.html.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *CoRR*, abs/2604.13016, 2026. doi: 10.48550/ARXIV.2604.13016. URL https://doi.org/10.48550/arXiv.2604.13016.

Alexander Lin, Jeremy Wohlwend, Howard Chen, and Tao Lei. Autoregressive knowledge distillation through imitation learning. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 6121–6133. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-MAIN.494. URL https://doi.org/10.18653/v1/2020.emnlp-main.494.

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.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, Jinhao Dong, Zhifang Sui, and Fuli Luo. MOPD: multi-teacher on-policy distillation for capability integration in LLM post-training. *CoRR*, abs/2606.30406, 2026. doi: 10.48550/ARXIV.2606.30406. URL https://doi.org/10.48550/arXiv.2606.30406.

Yifan Niu, Han Xiao, Dongyi Liu, Zelong Wang, Dihong Gong, Yasheng Wang, and Jia Li. Breaking the tokenizer barrier: On-policy distillation across model families. *CoRR*, abs/2606.09456, 2026. doi: 10.48550/ARXIV.2606.09456. URL https://doi.org/10.48550/arXiv.2606.09456.

Carlos Miguel Patiño, Kashif Rasul, Quentin Gallouédec, Ben Burtenshaw, Sergio Paniego, Vaibhav Srivastav, Thibaud Frere, Ed Beeching, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. Unlocking on-policy distillation for any model family. https://huggingface.co/spaces/HuggingFaceH4/on-policy-distillation, 2025.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pp. 627–635. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06295.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. *CoRR*, abs/2604.00626, 2026. doi: 10.48550/ARXIV.2604.00626. URL https://doi.org/10.48550/arXiv.2604.00626.

Jie Sun, Mao Zheng, Mingyang Song, Qiyong Zhong, Yilin Cheng, Bichuan Feng, Pengfei Liu, Junfeng Fang, and Xiang Wang. Simct: Recovering lost supervision for cross-tokenizer on-policy distillation. *CoRR*, abs/2605.07711, 2026. doi: 10.48550/ARXIV.2605.07711. URL https://doi.org/10.48550/arXiv.2605.07711.

Kimi Team. Kimi K3: open frontier intelligence. *CoRR*, abs/2607.24653, 2026. doi: 10.48550/ARXIV.2607.24653. URL https://doi.org/10.48550/arXiv.2607.24653.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Hao Wang, Kun Yuan, Wenlin Zhong, Minglei Zhang, Han Xiao, Ming Sun, and Honggang Qi. Cross-tokenizer on-policy distillation via byte-prefix marginalization. *CoRR*, abs/2607.22334, 2026a. doi: 10.48550/ARXIV.2607.22334. URL https://doi.org/10.48550/arXiv.2607.22334.

Jianze Wang, Ying Liu, Jinlong Chen, Xuchun Hu, Qilong Zhang, Yu Cao, Jun Wang, Hua Yang, Yong Xie, and Qianglong Chen. MAD-OPD: breaking the ceiling in on-policy distillation via multi-agent debate. *CoRR*, abs/2605.01347, 2026b. doi: 10.48550/ARXIV.2605.01347. URL https://doi.org/10.48550/arXiv.2605.01347.

Jiaqi Wang, Wenhao Zhang, Weijie Shi, Yaliang Li, and James Cheng. TCOD: exploring temporal curriculum in on-policy distillation for multi-turn autonomous agents. *CoRR*, abs/2604.24005, 2026c. doi: 10.48550/ARXIV.2604.24005. URL https://doi.org/10.48550/arXiv.2604.24005.

Yecheng Wu, Song Han, and Hai Cai. Lightning OPD: efficient post-training for large reasoning models with offline on-policy distillation. *CoRR*, abs/2604.13010, 2026. doi: 10.48550/ARXIV.2604.13010. URL https://doi.org/10.48550/arXiv.2604.13010.

LLM-Core Xiaomi. Mimo-v2-flash technical report. *CoRR*, abs/2601.02780, 2026. doi: 10.48550/ARXIV.2601.02780. URL https://doi.org/10.48550/arXiv.2601.02780.

Xingrun Xing, Haoqing Wang, Boyan Gao, Ziheng Li, and Yehui Tang. Trust region on-policy distillation. *CoRR*, abs/2606.01249, 2026. doi: 10.48550/ARXIV.2606.01249. URL https://doi.org/10.48550/arXiv.2606.01249.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR. *CoRR*, abs/2604.03128, 2026a. doi: 10.48550/ARXIV.2604.03128. URL https://doi.org/10.48550/arXiv.2604.03128.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *CoRR*, abs/2602.12125, 2026b. doi: 10.48550/ARXIV.2602.12125. URL https://doi.org/10.48550/arXiv.2602.12125.

Zhuolin Yang, Zihan Liu, Yang Chen, Wenliang Dai, Boxin Wang, Sheng-Chieh Lin, Chankyu Lee, Yangyi Chen, Dongfu Jiang, Jiafan He, Renjie Pi, Grace Lam, Nayeon Lee, Alexander Bukharin, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nemotron-cascade 2: Post-training llms with cascade RL and multi-domain on-policy distillation. *CoRR*, abs/2603.19220, 2026c. doi: 10.48550/ARXIV.2603.19220. URL https://doi.org/10.48550/arXiv.2603.19220.

Junjie Ye, Yuming Yang, Yang Nan, Shuo Li, Qi Zhang, Tao Gui, Xuanjing Huang, Peng Wang, Zhongchao Shi, and Jianping Fan. Analyzing the effects of supervised fine-tuning on model knowledge from token and parameter levels. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 471–513. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.25. URL https://doi.org/10.18653/v1/2025.emnlp-main.25.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Ming Ding, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/5afa9cb1e917b898ad418216dc726fbd-Abstract-Datasets_and_Benchmarks_Track.html.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Xin Zhao, Ruihua Song, Yuan Zhang, Peng Wang, Cheng Chen, Jirong Wen, and Kai Jia. Immersion in the github universe: Scaling coding agents to mastery. *CoRR*, abs/2602.09892, 2026a. doi: 10.48550/ARXIV.2602.09892. URL https://doi.org/10.48550/arXiv.2602.09892.

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. *CoRR*, abs/2601.18734, 2026b. doi: 10.48550/ARXIV.2601.18734. URL https://doi.org/10.48550/arXiv.2601.18734.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yuhang Zhou, Lizhu Zhang, Yifan Wu, Mingyi Wang, Bo Peng, Jiayi Liu, Xiangjun Fan, and Zhuokai Zhao. SAGE-OPD: selective agent-guided intervention for multi-turn on-policy distillation. *CoRR*, abs/2606.19659, 2026. doi: 10.48550/ARXIV.2606.19659. URL https://doi.org/10.48550/arXiv.2606.19659.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Experimental Setup and Implementation Details

A.1 Training Data and Model Configurations

Training subsets. We randomly sample 5k queries from the 20k-query ScaleSWE dataset (Zhao et al., 2026a) and approximately 4k tasks from the internal dataset (Guo et al., 2026). These subset sizes align the training-data budget with SWE reinforcement learning settings that typically use a few thousand tasks (Jain et al., 2025; Ding et al., 2026). We use the same ScaleSWE subset for Teacher Rollout Distillation and SWE-OPD, and the same internal subset for the online and offline comparisons. In preliminary experiments, training on other randomly sampled subsets also produced clear improvements over the corresponding student baselines.

Models and objectives. We use instruction-tuned checkpoints for both model pairs: Qwen3.5-9B distilled from Qwen3.6-27B, and GLM-4.5-Air distilled from GLM-5.2. Table 5 summarizes the training configurations. For RKL training, the student support is recomputed at each update with $K = 128$ for both model pairs; the support-size ablation additionally evaluates $K \in \{256, 512, 1024\}$ on ScaleSWE. ST uses the generated token and does not have a top-K training support. All offline configurations use a teacher cache of $K_{\text{cache}} = 1024$, as described in Section 5. GLM RKL training collapses and is analyzed separately in Section 6.

A.2 Optimization and Checkpoint Selection

We implement OPD using Slime (Zhu et al., 2025) and optimize the student with AdamW at a constant learning rate of 10^{-6}. For ST training, we clip the importance ratio to $[0.8, 1.2]$. The rollout batch size is 64, with one rollout sampled per training query. Training rollouts use a 128K-token context budget. All training configurations use a one-epoch budget over their training subset, and online and offline SWE-OPD use matched student-training budgets on the same sampled tasks. The online variant refreshes student trajectories during training; the offline variant trains on the initial student's saved trajectories and cached teacher scores.

For SWE-OPD and Offline SWE-OPD, we select the evaluated checkpoint within 1.0 epochs. Within this interval, we choose the checkpoint with the highest teacher–student top-K overlap, using the

Table 5: Training configurations. Qwen denotes Qwen3.5-9B ← Qwen3.6-27B, and GLM denotes GLM-4.5-Air ← GLM-5.2, with student ← teacher. K is the RKL student-support size; K_{cache} is the offline teacher-cache size. A dash denotes a setting that does not apply.

Training data	Model pair	Training variant	Loss	K	K_{cache}
ScaleSWE (5K)	Qwen	Teacher rollouts	SFT	–	–
	Qwen	Online	RKL	128	–
	Qwen	Online	ST	–	–
	Qwen	Online (ablation)	RKL	256/512/1024	–
Internal (∼4K)	Qwen	Online	RKL	128	–
	Qwen	Online	ST	–	–
	Qwen	Offline	RKL	128	1024
	Qwen	Offline	ST	–	1024
	GLM	Online[†]	RKL	128	–
	GLM	Online	ST	–	–
	GLM	Offline	ST	–	1024

[†] GLM RKL training collapses and has no reported benchmark score; see Section 6.

diagnostic in Section 4.5. In particular, most checkpoints are selected at 1 epoch; for GLM-4.5-Air, the online SWE-OPD checkpoint is selected at around half an epoch. This selection rule applies to both RKL and ST training, including the offline configurations. In preliminary experiments, extending training beyond one epoch provided little additional benefit and sometimes reduced downstream performance.

Teacher Rollout Distillation. For the SFT baseline in Table 1, Qwen3.6-27B generates approximately 5,000 trajectories on the same ScaleSWE subset. We retain successful and unsuccessful trajectories without filtering by task reward, and fine-tune Qwen3.5-9B on the teacher-generated response tokens for one epoch using the standard SFT objective.

A.3 Agent Scaffold and Tools

We adopt the ScaleSWE scaffold (Zhao et al., 2026a) through its function-calling configuration (`edit_fn_calling`) in our R2E-Gym integration (Jain et al., 2025). Training and evaluation share the scaffold loader and prompt-construction code. Each task starts with a system message and a user message containing the issue description and repository location. Tool schemas are supplied through the model's chat template, and observations are returned as tool-role messages. The agent has access to three tools:

- `str_replace_editor`: view files or directories, create files, replace an exact text span, and insert text after a specified line. File changes persist across calls.
- `execute_bash`: execute a shell command in the repository, including search, build, and test commands. Each call starts a fresh shell; directory changes and shell environment variables do not persist between calls. Commands are subject to a timeout, and long outputs may be truncated.
- `finish`: terminate the interaction when the task is complete or the agent cannot proceed further. The tool takes no arguments.

A.4 Evaluation Protocol

We follow the R2E-Gym evaluation protocol (Jain et al., 2025), using SGLang (Zheng et al., 2024) to serve the evaluated models. The agent interacts with the task repository using the scaffold above, and its resulting patch is evaluated using the benchmark's required functionality and regression tests. A task is resolved only if all required tests pass.

We evaluate on 500 SWE-bench Verified tasks, 731 SWE-bench Pro tasks, and 300 SWE-bench Multilingual tasks. The evaluation temperature is 1.0, top-p is 0.95, and the repetition penalty is 1.0. We allow at most 100 interaction steps on Verified and 150 on Pro and Multilingual, with a

128K-token context budget. These evaluation settings apply to the student baselines and the trained checkpoints in the benchmark tables.

A.5 System and User-Message Templates

The colored blocks reproduce the system and user-message templates from the ScaleSWE `edit_fn_calling` configuration verbatim. The placeholder `{problem_statement}` is filled with the task's issue description; `/testbed` is replaced by the actual repository working directory when it differs. The user-message template retains its original wording, including *python code repository*. The three tool schemas described in Appendix A.3 are supplied separately through the model's chat template.

ScaleSWE scaffold: system message

```
You are a helpful AI assistant that can interact with a computer to
solve tasks.

<ROLE>
Your primary role is to assist by executing commands, modifying code,
and solving technical problems effectively. You should be thorough,
methodical, and prioritize quality over speed.
</ROLE>

<EFFICIENCY>
* Each action you take is somewhat expensive. Wherever possible,
combine multiple actions into a single action, e.g. combine multiple
bash commands into one, using sed and grep to edit/view multiple
files at once.
* When exploring the codebase, use efficient tools like find, grep,
and git commands with appropriate filters to minimize unnecessary
operations.
</EFFICIENCY>

<FILE_SYSTEM_GUIDELINES>
* When given a file path, do NOT assume it's relative to the current
working directory. First explore the file system to locate the file
before working on it.
* If asked to edit a file, edit the file directly, rather than
creating a new file with a different filename.
* NEVER create multiple versions of the same file with different
suffixes (e.g., file_test.txt, file_fix.txt, file_simple.txt).
Instead:
- Always modify the original file directly when making changes
- If you need to create a temporary file for testing, delete it once
you've confirmed your solution works
- If you decide a file you created is no longer useful, delete it
instead of creating a new version
* When reproducing bugs or implementing fixes, use a single file
rather than creating multiple files with different versions
</FILE_SYSTEM_GUIDELINES>

<CODE_QUALITY>
* Write clean, efficient code with minimal comments. Avoid redundancy
in comments: Do not repeat information that can be easily inferred
from the code itself.
* When implementing solutions, focus on making the minimal changes
needed to solve the problem.
* Before implementing any changes, first thoroughly understand the
codebase through exploration.
* If you are adding a lot of code to a function or file, consider
splitting the function or file into smaller pieces when appropriate.
* Place all imports at the top of the file unless placing imports at
the top would cause issues (e.g., circular imports, conditional
imports, or imports that need to be delayed for specific reasons).
</CODE_QUALITY>
```

ScaleSWE scaffold: system message (continued)

```
<PROBLEM_SOLVING_WORKFLOW>
1. EXPLORATION: Thoroughly explore relevant files and understand the
context before proposing solutions
2. ANALYSIS: Consider multiple approaches and select the most
promising one
3. TESTING:
  * For bug fixes: Write a small reproduction script to confirm the
  issue exists before implementing a fix, and re-run it afterward to
  verify the fix works.
  * For new features: Consider writing simple tests when appropriate
  to validate your implementation.
4. IMPLEMENTATION:
  * Make focused, minimal changes to address the problem
  * Always modify existing files directly rather than creating new
  versions with different suffixes
  * If you create temporary files for testing, delete them after
  confirming your solution works
5. VERIFICATION: Test your implementation thoroughly, including edge
cases.
6. SUBMISSION: When you have completed the task or cannot proceed
further, call the `finish` tool to end the interaction.
</PROBLEM_SOLVING_WORKFLOW>

<ENVIRONMENT_SETUP>
* The development environment is already set up and all dependencies
are installed. You can directly run tests or compile without
additional setup.
* If you do encounter a missing tool or package, you may install it
as needed.
</ENVIRONMENT_SETUP>

<TROUBLESHOOTING>
* If you've made repeated attempts to solve a problem but tests still
fail:
1. Step back and reflect on 5-7 different possible sources of the
problem
2. Assess the likelihood of each possible cause
3. Methodically address the most likely causes, starting with the
highest probability
* When you run into any major issue while executing your plan, do not
try to directly work around it. Instead, re-evaluate your approach
and consider alternative solutions.
</TROUBLESHOOTING>

<PROCESS_MANAGEMENT>
* Be aware that each bash command must complete and return within a
time limit. Avoid running commands that block the terminal
indefinitely (e.g., starting a server in the foreground, interactive
prompts, or commands that wait for user input). If you need to start
a long-running process, run it in the background.
* When terminating processes, use specific identifiers rather than
broad patterns like `pkill -f python`. Prefer finding the exact PID
with `ps aux` first.
</PROCESS_MANAGEMENT>

<TOOL_CALL_RULES>
* Only call one function at a time.
* You may provide reasoning in natural language BEFORE the function
call.
* Required parameters MUST be specified.
</TOOL_CALL_RULES>
```

ScaleSWE scaffold: user-message template

```
<uploaded_files>
/testbed
</uploaded_files>

I've uploaded a python code repository in the directory /testbed.
Consider the following issue description:

<issue_description>
{problem_statement}
</issue_description>

Can you help me implement the necessary changes to the repository so
that the requirements specified in the <issue_description> are met?
I've already taken care of all changes to any of the test files
described in the <issue_description>. This means you DON'T have to
modify the testing logic or any of the tests in any way!
Also the development environment is already set up for you (i.e., all
dependencies already installed), so you don't need to install other
packages.
Your task is to make the minimal changes to non-test files in the
/testbed directory to ensure the <issue_description> is satisfied.

Follow these phases to resolve the issue:

Phase 1. READING: read the problem and reword it in clearer terms
  1.1 If there are code or config snippets. Express in words any best
  practices or conventions in them.
  1.2 Highlight message errors, method names, variables, file names,
  stack traces, and technical details.
  1.3 Explain the problem in clear terms.
  1.4 Enumerate the steps to reproduce the problem.
  1.5 Highlight any best practices to take into account when testing
  and fixing the issue

Phase 2. RUNNING: run the tests on the repository
  2.1 The development environment is already set up and all
  dependencies are installed -- you do NOT need to install anything.
  2.2 Explore the repository structure and figure out how to run the
  relevant tests.

Phase 3. EXPLORATION: find the files that are related to the problem
and possible solutions
  3.1 Use `grep` to search for relevant methods, classes, keywords
  and error messages.
  3.2 Identify all files related to the problem statement.
  3.3 Propose the methods and files to fix the issue and explain why.
  3.4 From the possible file locations, select the most likely
  location to fix the issue.

Phase 4. TEST CREATION: before implementing any fix, create a script
to reproduce and verify the issue.
  4.1 Look at existing test files in the repository to understand the
  test format/structure.
  4.2 Create a minimal reproduction script that reproduces the
  located issue.
  4.3 Run the reproduction script to confirm you are reproducing the
  issue.
  4.4 Adjust the reproduction script as necessary.

Phase 5. FIX ANALYSIS: state clearly the problem and how to fix it
  5.1 State clearly what the problem is.
  5.2 State clearly where the problem is located.
  5.3 State clearly how the test reproduces the issue.
  5.4 State clearly the best practices to take into account in the
  fix.
  5.5 State clearly how to fix the problem.
```

ScaleSWE scaffold: user-message template (continued)

```
Phase 6. FIX IMPLEMENTATION: Edit the source code to implement your
chosen solution.
  6.1 Make minimal, focused changes to fix the issue.

Phase 7. VERIFICATION: Test your implementation thoroughly.
  7.1 Run your reproduction script to verify the fix works.
  7.2 Add edge cases to your test script to ensure comprehensive
  coverage.
  7.3 Run existing tests related to the modified code to ensure you
  haven't broken anything.

Phase 8. FINAL REVIEW: Carefully re-read the problem description and
compare your changes with the original code.
  8.1 Ensure you've fully addressed all requirements.
  8.2 Run any tests in the repository related to:
  8.2.1 The issue you are fixing
  8.2.2 The files you modified
  8.2.3 The functions you changed
  8.3 If any tests fail, revise your implementation until all tests
  pass

Phase 9. SUBMISSION: When you have completed the task or cannot
proceed further, call the `finish` tool to end the interaction.

Be thorough in your exploration, testing, and reasoning. It's fine if
your thinking process is lengthy - quality and completeness are more
important than brevity.

---------------------- END OF NEW TASK DESCRIPTION
----------------------

PLEASE follow the format strictly! PLEASE EMIT ONE AND ONLY ONE
FUNCTION CALL PER MESSAGE.
```

B Gradient Analysis of the Implemented Top-K loss

Scope and notation. We follow Section 4.3, including its omission of the batch index b. For each derivative, we fix token position i, its prefix $(x, \mathbf{y}_{<i})$, the teacher distribution q_i, and the selected support $\mathcal{V}_K^{(i)}$ from Eq. 2. The student probabilities $p_i(v)$ from Eq. 1 are differentiable functions of θ and are positive unless an endpoint convention is stated. The support and the detached normalizer Z_i^p are recomputed between training updates. The loss decomposition uses parameter gradients; Appendix B.3 separately gives the logit derivatives used for the mechanism figure.

B.1 Forward Value and Backward Computation

Forward value and detached normalization. Using the support mass defined in Section 6.2, we distinguish this differentiable quantity from its detached copy used to normalize the loss:

$$\hat{\mathrm{Z}}_{\mathrm{total}} = \sum_{v \in \mathcal{V}_K^{(i)}} p_i(v), \qquad \mathrm{Z}_i^p = \mathrm{sg}\left[\hat{\mathrm{Z}}_{\mathrm{total}}\right].$$

These quantities have the same forward value, so $\bar{p}_i(v) = p_i(v)/\mathrm{Z}_i^p$ sums to one over the support. During differentiation, $\nabla_\theta \mathrm{Z}_i^p = 0$, while $\hat{\mathrm{Z}}_{\mathrm{total}}$ remains differentiable through the student probabilities. The teacher probabilities and selected token indices are fixed.

Gradient decomposition. Applying the product rule to the student weight and log-ratio in Eq. 4, holding the teacher and detached denominator fixed and using $\nabla_\theta p_i(v) = p_i(v)\nabla_\theta \log p_i(v)$, gives

$$\begin{aligned}
\nabla_\theta \ell_i^{\mathrm{RKL}} &= \sum_{v\in\mathcal{V}_K^{(i)}} \left[\log \frac{p_i(v)}{q_i(v)}\,\nabla_\theta \bar{p}_i(v) + \bar{p}_i(v)\nabla_\theta \log \frac{p_i(v)}{q_i(v)}\right] \\
&= \frac{1}{\mathrm{Z}_i^p} \sum_{v\in\mathcal{V}_K^{(i)}} \left[\log \frac{p_i(v)}{q_i(v)}\nabla_\theta p_i(v) + p_i(v)\nabla_\theta \log p_i(v)\right] \\
&= \sum_{v\in\mathcal{V}_K^{(i)}} \frac{p_i(v)}{\mathrm{Z}_i^p}\left(\log \frac{p_i(v)}{q_i(v)} + 1\right)\nabla_\theta \log p_i(v) \\
&= \underbrace{\sum_{v\in\mathcal{V}_K^{(i)}} \bar{p}_i(v) \log \frac{p_i(v)}{q_i(v)}\nabla_\theta \log p_i(v)}_{\mathbf{g}_i} + \underbrace{\nabla_\theta \log \hat{\mathrm{Z}}_{\mathrm{total}}}_{\mathbf{h}_i}.
\end{aligned} \tag{10}$$

Thus $\mathbf{g}_i$ comes from differentiating the student weights and retains the teacher-dependent log-ratios. The term $\mathbf{h}_i$ comes from differentiating the log-ratios themselves and is independent of the teacher. To see why it is a support-mass term, sum the probability derivatives over the retained tokens:

$$\begin{aligned}
\mathbf{h}_i &= \sum_{v\in\mathcal{V}_K^{(i)}} \frac{p_i(v)}{\mathrm{Z}_i^p}\nabla_\theta \log p_i(v) = \frac{1}{\mathrm{Z}_i^p}\sum_{v\in\mathcal{V}_K^{(i)}} \nabla_\theta p_i(v) \\
&= \frac{1}{\mathrm{Z}_i^p}\nabla_\theta \hat{\mathrm{Z}}_{\mathrm{total}} = \frac{1}{\hat{\mathrm{Z}}_{\mathrm{total}}}\nabla_\theta \hat{\mathrm{Z}}_{\mathrm{total}} = \nabla_\theta \log \hat{\mathrm{Z}}_{\mathrm{total}}.
\end{aligned} \tag{11}$$

The simplification uses the equality of the forward values of Z_i^p and $\hat{\mathrm{Z}}_{\mathrm{total}}$; it does not differentiate the detached normalizer. Thus $-\mathbf{h}_i$ is a descent direction for the retained mass whenever $\mathbf{h}_i \neq 0$. The full update also includes $\mathbf{g}_i$, which can counteract this decrease. Although $\bar{p}_i(v)$ sums to one in the forward pass, its gradients need not sum to zero. Over the full vocabulary, $\hat{\mathrm{Z}}_{\mathrm{total}} = 1$ identically and $\mathbf{h}_i = 0$. Truncation removes this cancellation, although $\mathbf{h}_i$ can still vanish where retained mass is locally stationary in θ.

B.2 Expected Gradient of the Sampled-Token Loss

We use the fixed behavior distribution p_i^{old} from Section 4.3 and assume it assigns positive probability wherever p_i is positive. Differentiating Eq. 5, using its detached advantage and exact, unclipped importance ratio, and taking the expectation gives

$$\begin{aligned}
\mathbb{E}_{y_i\sim p_i^{\mathrm{old}}}\left[\nabla_\theta \ell_i^{\mathrm{ST}}\right] &= \mathbb{E}_{y_i\sim p_i^{\mathrm{old}}}\left[\frac{p_i(y_i)}{p_i^{\mathrm{old}}(y_i)}\log \frac{p_i(y_i)}{q_i(y_i)}\nabla_\theta \log p_i(y_i)\right] \\
&= \sum_{v\in\mathcal{V}} p_i(v)\log \frac{p_i(v)}{q_i(v)}\nabla_\theta \log p_i(v) \\
&= \nabla_\theta D_{\mathrm{KL}}(p_i\|q_i).
\end{aligned} \tag{12}$$

The importance ratio cancels the behavior probabilities in the expectation. The final equality uses $\sum_{v\in\mathcal{V}} \nabla_\theta p_i(v) = 0$; at $p_i = q_i$, the ST gradient vanishes for every sampled action. The expected ST parameter gradient therefore equals the full-vocabulary reverse-KL gradient, including when $p_i^{\mathrm{old}} \neq p_i$. Every token's own teacher score enters this expectation, including tokens outside the current top-K. Removing $\mathbf{h}_i$ from the truncated gradient alone leaves a restricted, renormalized sum and does not recover this expectation. These identities concern the fixed-prefix gradient and do not differentiate through the distribution of visited prefixes.

B.3 Logit-Level Mechanism and Constructed Example

Differentiating the original loss. Let $z_i(j)$ be the student's logit for vocabulary token j at the fixed prefix. Starting from Eq. 4, we write the loss together with the full-vocabulary softmax:

$$\ell_i^{\mathrm{RKL}} = \sum_{v \in \mathcal{V}_K^{(i)}} \frac{p_i(v)}{\mathrm{Z}_i^p} \log \frac{p_i(v)}{q_i(v)}, \qquad p_i(v) = \frac{e^{z_i(v)}}{\sum_{v' \in \mathcal{V}} e^{z_i(v')}}. \tag{13}$$

The selected support, teacher probabilities, and detached normalizer Z_i^p are held fixed during differentiation. The softmax denominator remains differentiable. Applying the quotient rule and then differentiating the log-ratio gives

$$\begin{aligned} \frac{\partial p_i(v)}{\partial z_i(j)} &= \frac{\mathbb{1}[v=j] e^{z_i(v)} \sum_{v' \in \mathcal{V}} e^{z_i(v')} - e^{z_i(v)} e^{z_i(j)}}{\left(\sum_{v' \in \mathcal{V}} e^{z_i(v')}\right)^2} \\ &= p_i(v)\big(\mathbb{1}[v=j] - p_i(j)\big), \\ \frac{\partial}{\partial z_i(j)} \log \frac{p_i(v)}{q_i(v)} &= \frac{1}{p_i(v)} \frac{\partial p_i(v)}{\partial z_i(j)} = \mathbb{1}[v=j] - p_i(j). \end{aligned} \tag{14}$$

Thus a logit outside the selected support still affects every retained probability. Applying the product rule directly to the loss in Eq. 13 yields

$$\begin{aligned} \frac{\partial \ell_i^{\mathrm{RKL}}}{\partial z_i(j)} &= \frac{1}{\mathrm{Z}_i^p} \sum_{v \in \mathcal{V}_K^{(i)}} \left[\log \frac{p_i(v)}{q_i(v)} \frac{\partial p_i(v)}{\partial z_i(j)} + p_i(v) \frac{\partial}{\partial z_i(j)} \log \frac{p_i(v)}{q_i(v)} \right] \\ &= \sum_{v \in \mathcal{V}_K^{(i)}} \frac{p_i(v)}{\mathrm{Z}_i^p} \left(\log \frac{p_i(v)}{q_i(v)} + 1 \right) \big(\mathbb{1}[v=j] - p_i(j) \big) \\ &= \mathbb{1}[j \in \mathcal{V}_K^{(i)}] \frac{p_i(j)}{\mathrm{Z}_i^p} \left(\log \frac{p_i(j)}{q_i(j)} + 1 \right) - p_i(j) \sum_{v \in \mathcal{V}_K^{(i)}} \frac{p_i(v)}{\mathrm{Z}_i^p} \left(\log \frac{p_i(v)}{q_i(v)} + 1 \right). \end{aligned} \tag{15}$$

Both occurrences of $+1$ arise from differentiating the log-ratio. Using the forward values $\sum_{v \in \mathcal{V}_K^{(i)}} p_i(v)/\mathrm{Z}_i^p = 1$ and Eq. 13, the remaining weighted sum equals $\ell_i^{\mathrm{RKL}} + 1$. The derivative therefore separates into two cases:

$$\frac{\partial \ell_i^{\mathrm{RKL}}}{\partial z_i(j)} = \begin{cases} p_i(j) \left[\dfrac{\log[p_i(j)/q_i(j)] + 1}{\mathrm{Z}_i^p} - \big(\ell_i^{\mathrm{RKL}} + 1\big) \right], & j \in \mathcal{V}_K^{(i)}, \\ -p_i(j)\big(\ell_i^{\mathrm{RKL}} + 1\big), & j \notin \mathcal{V}_K^{(i)}. \end{cases} \tag{16}$$

For a retained token, the derivative includes its own teacher log-ratio and the aggregate contribution from all retained tokens. For an omitted token, its own teacher probability $q_i(j)$ is absent: its derivative is determined by $p_i(j)$ and the retained aggregate. Under a logit descent step, omitted logits increase when $\ell_i^{\mathrm{RKL}} > -1$, stay unchanged at -1, and decrease below -1. This condition describes logit direction; support-mass transport additionally depends on the retained-token updates through the full softmax.

Updates used in the constructed example. For the RKL loss, we apply $z_i^+ = z_i - \eta \nabla_{z_i} \ell_i^{\mathrm{RKL}}$ using the derivatives above and recompute $p_i^+ = \mathrm{softmax}(z_i^+)$. For ST training, Eq. 12 and the same softmax derivatives give

$$\mathbb{E}\left[\frac{\partial \ell_i^{\mathrm{ST}}}{\partial z_i(j)} \right] = p_i(j) \left[\log \frac{p_i(j)}{q_i(j)} - D_{\mathrm{KL}}(p_i \| q_i) \right]. \tag{17}$$

We use this exact expected gradient for the ST updates. For the neural student, the chain rule gives

$$\nabla_\theta \ell_i^{\mathrm{RKL}} = \sum_{j \in \mathcal{V}} \frac{\partial \ell_i^{\mathrm{RKL}}}{\partial z_i(j)} \nabla_\theta z_i(j),$$

and training aggregates these parameter gradients across supervised positions.

Figure 6 uses a 102-token vocabulary. Initially $p_i(v_1) = 0.55$, $p_i(v_2) = 0.25$, $q_i(v_1) = 0.60$, and $q_i(v_2) = 0.30$. For tail rank $k = 1, \ldots, 100$,

$$p_i(v_{k+2}) = 0.20 \frac{k^{-0.8}}{\sum_{k'=1}^{100}(k')^{-0.8}}, \qquad q_i(v_{k+2}) = 0.10 \frac{k^{-1}}{\sum_{k'=1}^{100}(k')^{-1}}. \tag{18}$$

Both trajectories take 40 independent-logit steps of size 0.25. The same two tokens remain top-2 at every step. ST updates use the exact full-vocabulary expectation. After every step, probabilities and full-vocabulary KL are recomputed from the complete softmax; finite-difference checks hold Z_i^p fixed while recomputing the full softmax when verifying the truncated gradient.

	not	len	Outside $\mathcal{V}_K$ (100 tokens)
Initial student	55%	25%	20%
Fixed teacher	60%	30%	10%

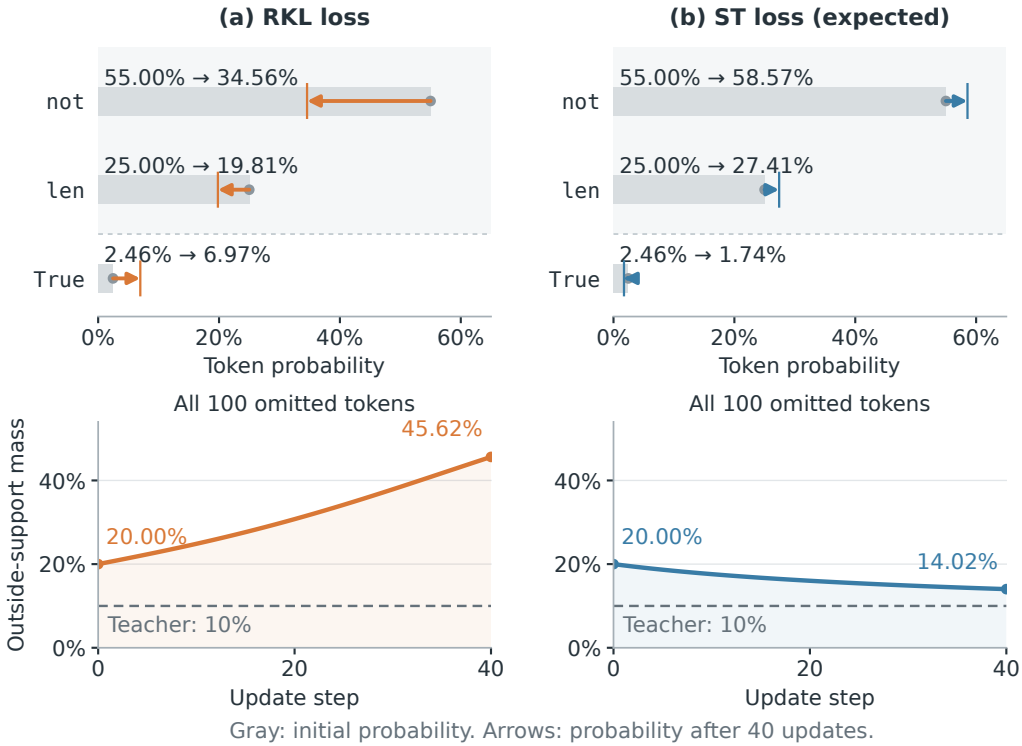


Figure 6: **Constructed comparison of independent-logit updates.** Upper panels show initial probabilities (gray) and final probabilities (arrow tips) of two retained tokens and one omitted token. Lower panels track total outside-support mass; dashed lines mark the teacher. ST updates use the exact expected gradient.

Although the teacher assigns 90% mass to the retained pair, compared with the student's initial 80%, RKL updates increase outside-support mass from 20.00% to 45.62%. The surrogate falls from -0.1168 to -0.5018, while full-vocabulary reverse KL rises from 0.0537 to 0.4297. The expected ST update instead reduces outside-support mass to 14.02% and full KL to 0.0145.

This construction isolates loss reduction with worsening teacher alignment. Its initial support mass is chosen to make transport visible. Measuring the contribution to neural training additionally requires the logit Jacobian and updates at fixed diagnostic prefixes, since parameter gradients aggregate contributions across positions. The construction does not model the later concentration on repetitive sequences.