Dense Supervision but Sparse Guidance: A Token-Level Analysis of On-Policy Self-Distillation

Taeckyung Lee [1] **Jeonghye Kim** [1] **Hyungjun Yoon** [1] **Sung-Ju Lee** [1]

Abstract

On-policy self-distillation (OPSD) derives a teacher from the student's own parameters. The teacher uses privileged feedback to provide dense token-level KL supervision. We analyze this supervision on code-generation tasks and find that it often provides only sparse guidance. At the peak disagreement position, KL often concentrates on tokens unrelated to code logic, such as comments and docstrings (45∼71%). The token at this position is judged to be the primary cause of failure in only 1.1∼8.9% of cases. Yet these positions can still influence what the model generates next. Allowing the teacher to continue freely from the peak-disagreement position yields correct solutions in 13.3% of cases. This suggests that the teacher's capability is better expressed through continued generation through a single-token score. At the same time, the low success rate reveals a limitation inherent in parameter-shared teachers. More strikingly, substituting only the teacher-preferred token and allowing the student to continue raises success to 9.9%, recovering 71% of the teacher's free continuations. Our findings explain why SDPO can outperform GRPO. Its benefit may come less from reliable local correction and more from concentrated guidance that influences the subsequent generation trajectory. More broadly, local corrective quality and optimization usefulness can diverge in token-level supervision.

1. Introduction

Reinforcement learning with verifiable rewards (RLVR) trains large language models on tasks whose outputs can be automatically verified, such as code generation against test suites (Le et al., 2022) and mathematical reasoning with ground-truth answers (Shao et al., 2024). However, the verifier typically provides only a single pass/fail signal for an entire response that may span hundreds to thousands of tokens, making fine-grained *credit assignment* challenging.

On-policy self-distillation (OPSD) offers a promising alternative. OPSD derives a *teacher* from the student's own parameters and conditions it on privileged information unavailable to the student, such as execution feedback or ground-truth solutions. The teacher and student distributions are then compared token-by-token using KL divergence, producing token-level supervision (Hübotter et al., 2026; Penaloza et al., 2026; Zhao et al., 2026). This enables OPSD to provide a dense learning signal even when outcome-level RLVR reward is sparse or entirely absent, where existing GRPO fails to provide the learning signal.

Although OPSD has demonstrated empirical gains, prior work has largely focused on downstream performance rather than the supervision signal itself: where teacher-student disagreement concentrates, whether it targets algorithmically meaningful tokens, and whether the teacher's suggested correction would actually resolve the error. This question is important because recent studies suggest that tokens contribute unevenly to learning (Yang et al., 2025b; Wang et al., 2025), and selective token-level distillation can match or outperform dense supervision (Tavor et al., 2026; Huang et al., 2025; Xu et al., 2026). However, it remains unclear how these phenomena manifest in OPSD, where the teacher and student are derived from the same underlying model.

This paper asks: *what information does the teacher's token-level supervision actually convey?*

Using the Qwen3 family on LiveCodeBench (Jain et al., 2024), we show that this dense supervision provides only sparse guidance: most of the token-level signal appears locally meaningless, yet the rare points where it concentrates can serve as trajectory divergence points that redirect generation toward different algorithms.

At a token level, the signal appears nearly useless. On failed rollouts, positions with maximum disagreement are dominated by tokens irrelevant to code logic, such as comments and docstrings, accounting for 45.0% (1.7B) to 71.3% (4B) of such positions, and are rarely the primary cause of failure

[1]KAIST, Daejeon, South Korea. Correspondence to: Sung-Ju Lee <profsj@kaist.ac.kr>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

(1.1∼8.9%). Tracking these metrics throughout training shows that this reliability gap persists even as overall pass rate more than doubles.

Despite this low local corrective quality, OPSD remains effective. The key lies not in local token repair but in trajectory steering. When the teacher is instead allowed to continue generating from the maximum disagreement position, it produces correct programs in 7.9∼13.3% of cases. This far exceeds the local repair rate, but the teacher shares the student's parameters and inherits similar failure modes. Even so, a single token suffices to transmit much of this capability: replacing only the teacher-preferred token and letting the student continue yields 9.9% correct solutions (vs. 1.7% under a same-token control). Truncating the KL loss after the peak position nearly recovers full SDPO performance (Section 4.5), further indicating that the supervision beyond this point contributes little additional signal.

These observations clarify why self-distillation can outperform GRPO on coding tasks despite unreliable local supervision. GRPO distributes sparse outcome-level reward uniformly across the entire sequence, providing limited spatial structure for credit assignment. In contrast, SDPO concentrates 40∼80× more KL divergence near positions of maximum teacher-student disagreement, producing a spatially structured optimization signal that can steer generation trajectories even when the peak position falls on tokens irrelevant to code logic. We hypothesize that this trajectory-level steering mechanism, rather than locally correct token repair, underlies the empirical advantage of SDPO over GRPO.

Our findings carry three implications for future work.

(1) We provide a token-level diagnostic framework for self-distillation, revealing where the supervision signal concentrates, what semantic role it targets, and whether it provides locally corrective guidance.
(2) Our results suggest that the parameter-sharing structure of OPSD limits the teacher's corrective ceiling (7.9∼13.3%), and that further gains may require stronger teachers, external knowledge, or alternative supervision mechanisms.
(3) More broadly, our findings suggest that the effectiveness of token-level objectives may arise less from locally correct token prediction and more from their ability to steer generation trajectories toward different paths.

2. Related Work

On-policy self-distillation. Reinforcement learning with verifiable rewards (RLVR) trains language models using automatically verifiable outcomes, such as unit-test execution or ground-truth answers, commonly optimized with GRPO (Le et al., 2022; Shao et al., 2024; Guo et al., 2025). On-policy self-distillation (OPSD) extends RLVR by introducing token-level supervision from a teacher derived from the student's own parameters. SDPO (Hübotter et al., 2026), π-Distill (Penaloza et al., 2026) and Self-Distilled Reasoner (Zhao et al., 2026) condition this teacher on privileged information such as execution traces or environment feedback, and train the student using token-level KL divergence; EMPO^2 and RLTF (Liu et al., 2026; Song et al., 2026) further formalize RL from text feedback and analyze its optimization behavior. Concurrently, Kim et al. (2026) shows that OPSD can degrade reasoning by suppressing epistemic verbalization by learning short traces from a confident teacher. We instead analyze the reliability and role of feedback-conditioned token-level supervision when the teacher and student share parameters.

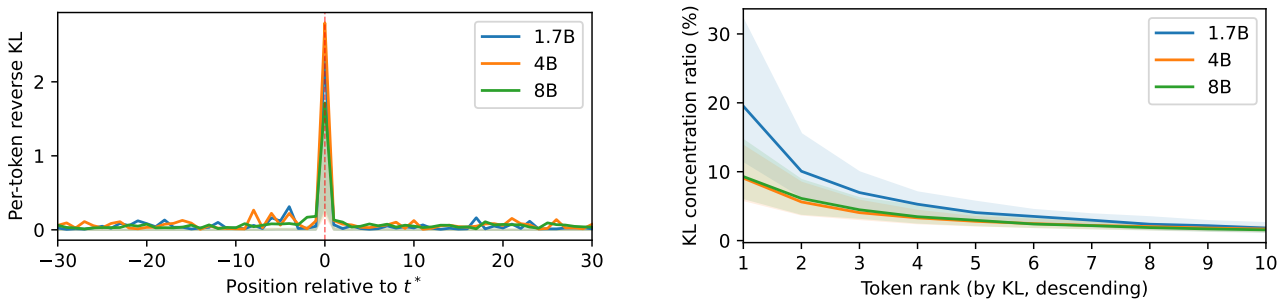
Credit Assignment in LLM Reinforcement Learning. Sparse outcome rewards create a credit-assignment challenge in RLVR: a verifier can judge whether an entire response is correct, but not which tokens or intermediate decisions caused success or failure. Prior work addresses this challenge through denser supervision mechanisms, including process reward models (Lightman et al., 2024; Wang et al., 2024), Monte Carlo value estimation (Kazemnejad et al., 2024), return decomposition (Parthasarathi et al., 2025), implicit process rewards (Cui et al., 2025), execution-derived penalties (Liu et al., 2023), and revision-based learning from execution feedback (Gehring et al., 2025).

Complementary studies show that token-level learning signals are highly non-uniform: low-probability tokens can dominate policy-gradient updates (Yang et al., 2025b), high-entropy minority tokens can disproportionately contribute to RL gains (Wang et al., 2025), and selective distillation can match or outperform dense distillation by prioritizing informative tokens (Xu et al., 2026; Tavor et al., 2026; Huang et al., 2025). Together, these studies motivate token-level credit assignment, but they do not examine whether such supervision remains reliable when the teacher is derived from the student itself. Our work investigates this question in OPSD by analyzing whether feedback-conditioned KL supervision provides locally corrective guidance or primarily steers downstream generation trajectories.

3. Token-Level Analysis Framework

To analyze what information the teacher's token-level supervision conveys, we first identify where the self-distillation loss concentrates most strongly. Among several OPSD frameworks, we follow SDPO (Hübotter et al., 2026), which has demonstrated robust performance on coding tasks.

We begin by formalizing the SDPO loss and defining key notation (§3.1), then show empirically that the KL signal concentrates sharply at a single position t^* (§3.2), and describe the evaluation protocol for t^* analysis (§3.3).



(a) Per-token KL profile around t^*.

(b) KL concentration ratio by token rank.

Figure 1. KL concentration at t^* on failed rollouts across Qwen3-{1.7B, 4B, 8B}. **(a)** Mean per-token KL in a ± 30 token window centered on t^* (shading: interquartile range); KL concentration around t^* is sharp and consistent across scales. **(b)** KL concentration ratio by token rank (descending); the rank-1 token (t^*) carries 9$\sim$20% of total response KL, decaying steeply to <1% by rank 10.

3.1. Preliminaries

On-policy self-distillation derives a teacher distribution p_ϕ from the student's own parameters p_θ and conditions it on privileged information unavailable to the student during generation. SDPO (Hübotter et al., 2026) instantiates this framework as follows. For each student rollout $y = (y_1, \ldots, y_T)$ generated from prompt x, the teacher produces token distributions conditioned on the prompt and the feedback f. The training objective is the mean reverse KL divergence:[1]

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} D_{\mathrm{KL}}\big(p_\theta(\cdot \mid x, y_{<t}) \,\|\, \mathrm{sg}[p_\phi(\cdot \mid x, y_{<t}, f)]\big), \tag{1}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient. Unless otherwise specified, we use *KL divergence* throughout this paper to refer to the reverse KL in Eq. (1). We focus on SDPO's *rich text feedback* setting, where f consists of success rollouts, if any; otherwise, execution-derived feedback from the test environment, including test-suite results and error traces

Notation for Token-Level KL Analysis. We denote the per-position reverse KL divergence from Eq. (1) as:

$$\mathrm{KL}(t) = D_{\mathrm{KL}}\big(p_\theta(\cdot \mid x, y_{<t}) \,\|\, p_\phi(\cdot \mid x, y_{<t}, f)\big). \tag{2}$$

We define the **peak-KL position** as $t^* = \arg\max_t \mathrm{KL}(t)$, corresponding to the position of maximum teacher-student disagreement. The **KL concentration ratio** of position t is:

$$C(t) = \frac{\mathrm{KL}(t)}{\sum_{t'} \mathrm{KL}(t')}, \tag{3}$$

i.e., the fraction of total response KL carried by position t. We write $C(t^*)$ for the peak concentration.

KL concentration ratio and the loss. Since the loss averages KL over positions (Eq. 1), position t contributes $\mathrm{KL}(t)/T$ to the total loss, and its share of the total loss is exactly $C(t)$. The per-position logit gradient is

$$\nabla_{z_t}\mathcal{L} = \frac{1}{T}\, p_\theta \odot (\log p_\theta/p_\phi - D_{\mathrm{KL}}(t)\cdot \mathbf{1}), \tag{4}$$

which vanishes when $\mathrm{KL}(t) = 0$ and grows with divergence (derived in Appendix B). $C(t)$ therefore measures the relative loss magnitude from position t, where the exact logit-gradient norm mapping also grows with $\mathrm{KL}(t)$ nonlinearly while depending on the distributional shape at t.

Models and data. We analyze SDPO training of Qwen3-{1.7B, 4B, 8B} (Yang et al., 2025a) on LiveCodeBench v6 (Jain et al., 2024), following SDPO. A single run contains 256 samples (32 problems $\times$ 8 rollouts), where we mainly focus on failed rollouts, while successful rollouts are retained as a diagnostic comparison. This is because failed rollouts directly expose the credit-assignment problem that motivates feedback-conditioned self-distillation. Our ablation applying KL only to successful rollouts yields substantially lower validation accuracy (0.372 vs. 0.480 at step 80; see Appendix A.3)

3.2. Why Focus on Peak-KL Position t^*?

Although the SDPO loss (Eq. 1) averages KL across all positions, the resulting training signal is highly non-uniform. Figure 1 shows that $C(t^*) = 9\sim20\%$ on failed rollouts, meaning this single position accounts for 9$\sim$20% of the loss magnitude, a 40$\sim$80$\times$ concentration relative to a uniform allocation. The concentration is sharp and consistent across all three model scales (Figure 1a), decaying steeply by token rank (Figure 1b). A natural alternative is the first-divergence position t_{div}, where $\arg\max p_\theta \neq \arg\max p_\phi$ first occurs. However, $C(t_{\mathrm{div}})$ is only 0.5$\sim$3.4%. Because t^* carries the largest concentration of KL mass, we target this single position to characterize the OPSD objective. Still, this choice does not assume that the peak-KL token is always the most semantically important or causally decisive token in the roll-

[1]The SDPO paper (Hübotter et al., 2026) defines the loss as a sum over tokens, while the official implementation averages over response length. We follow the implementation.

out; rather, we use it as a representative high-signal position for controlled token-level analysis.

3.3. Evaluation Protocol

We analyze each t^* event with two complementary evaluations: (i) an LLM judge (GPT-5.4) for semantic classification and local corrective assessment, and (ii) execution-based evaluation for continuation-level correctness.

LLM-judge evaluation. For each t^*, GPT-5.4 independently classifies the student and teacher tokens and evaluates their local corrective relevance. The *semantic role* categorizes the function of each token as: **algorithmic** (affecting control flow, algorithm choice, or execution semantics), **naming** (identifier choices without execution impact), **formatting** (whitespace or indentation), and **non-code text** (comments, docstrings, or reasonings). Student and teacher tokens may receive different labels; for example, a student comment token (`#`) may diverge from a teacher algorithmic token such as `if`. We further evaluate two local corrective properties: *student is cause* (whether the student token is a primary contributor to failure) and *teacher would fix* (whether substituting the teacher token would repair).

Execution-based evaluation. To assess the teacher's capability beyond isolated token substitution, we construct a teacher continuation as follows. We preserve the student's response prefix up to t^*, replace the student token with the teacher's argmax token, and allow the teacher to continue generation autoregressively. The resulting program is then extracted and executed against the LiveCodeBench test suite. This *continuation correctness* metric provides execution-based evaluation independent of LLM judgment. This evaluation can reveal corrective capability not captured by local token-level analysis: the teacher may still generate correct code even when t^* corresponds to a non-algorithmic token (e.g., a variable-name difference followed by a substantially different algorithmic continuation).

4. Diagnosing the Self-Distillation Signal

We now characterize the self-distillation signal empirically by analyzing where KL concentration occurs, what semantic role the peak-KL token plays, and whether the teacher's suggested token provides locally corrective guidance. We study Qwen3-{1.7B, 4B, 8B} at step 0 (Table 1) and throughout 80 training steps (Table 2), using both LLM judge evaluation and execution-based evaluation on the test suite.

4.1. What Does Peak KL Emphasize?

Table 1 summarizes the semantic role of the student token at t^*. Non-code text tokens, including comments, docstrings, and code fences, dominate at every scale, accounting for 45.0% of peak-KL positions at 1.7B and over 70% at 4B and 8B. In contrast, algorithmically meaningful tokens affecting control flow or execution semantics decrease from 36.1% at 1.7B to 11.6% at both 4B and 8B. The remaining disagreements correspond primarily to naming (variable names) or formatting (whitespace, indentation) tokens.

The LLM-judge analysis further suggests that the t^* position is rarely *directly responsible* for failure: the student token at t^* is identified as a primary contributor to failure in only 8.9% (1.7B), 1.1% (4B), and 2.8% (8B) of failed rollouts. Similarly, substituting the teacher-preferred token directly repairs the program in only 12.4% (1.7B), 2.2% (4B), and 3.9% (8B) of cases.

The KL concentration ratio decreases with scale, from 19.5% at 1.7B to approximately 9% at 4B and 8B, partly because larger models produce longer responses that distribute KL across more positions. Even at approximately 9%, however, the peak position still carries roughly 40× more KL concentration than an average token position, indicating that the supervision signal remains highly localized.

Takeaway 1. A single peak-KL position provides only weak corrective signals

On failed rollouts, peak-KL positions increasingly concentrate on non-code text tokens (comments, docstrings, and code fences), rising from 45% at 1.7B to over 70% at 4B and 8B. Direct token substitution using the teacher-preferred token repairs the program in only 2.2∼12.4% of cases, suggesting that local corrective quality at t^* is often limited.

4.2. The Teacher Shares the Student's Failures

Table 1 (execution-based correctness) reports continuation correctness at t^*, measured by executing teacher-generated continuations against the LiveCodeBench test suite. On failed rollouts, the teacher generates a correct continuation from t^* in only 7.9% (1.7B), 10.5% (4B), and 13.3% (8B) of cases. **Continuation correctness improves with scale, but remains relatively limited even at 8B scale.**

Teacher and student failures are highly correlated: when the student fails, the teacher continuation also fails in more than 86% of cases. This pattern appears closly tied to the parameter-sharing structure of OPSD. Because the teacher is derived from the student's own parameters, both models are likely to inherit similar failure modes. Rich execution feedback may help identify *what failed* but it does not necessarily provide the capability required to generate a *correct alternative algorithm*. This differs from conventional external-teacher distillation (Hinton et al., 2015), where the teacher may possess capabilities unavailable to the student.

Table 1. Self-distillation signal and teacher reliability on failed vs. successful rollouts across model scales (SDPO training step 0 on LiveCodeBench). *Semantic role* and *single-token evaluation* use an LLM judge (GPT-5.4). *Execution-based evaluation* uses the test suite: *teacher continuation* lets the teacher generate freely from t^*; *student continuation* substitutes the teacher's (or student's own) token at t^* and lets the student generate freely (greedy decoding). Numbers in parentheses denote rollout counts.

	Qwen3-1.7B		Qwen3-4B		Qwen3-8B	
	Fail (202)	Succ (54)	Fail (181)	Succ (75)	Fail (181)	Succ (75)
Semantic role						
Algorithmic	36.1%	11.1%	11.6%	4.0%	11.6%	24.0%
Naming	11.4%	9.3%	3.9%	4.0%	5.5%	12.0%
Formatting	7.4%	27.8%	13.3%	6.7%	12.7%	5.3%
Non-code text	45.0%	51.9%	71.3%	85.3%	70.2%	58.7%
Single-token evaluation at t^*						
Student is cause	8.9%	–	1.1%	–	2.8%	–
Teacher would fix	12.4%	–	2.2%	–	3.9%	–
Execution-based evaluation at t^*						
Teacher continuation	7.9%	92.6%	10.5%	98.7%	13.3%	97.3%
Student cont. (teacher token)	1.5%	–	5.0%	–	9.9%	–
Student cont. (own token, control)	1.5%	–	4.4%	–	1.7%	–

Takeaway 2. Teacher continuation reliability remains limited under parameter sharing

On failed rollouts, teacher continuations succeed in only 7.9% (1.7B), 10.5% (4B), and 13.3% (8B) of cases, despite near-perfect continuation correctness on successful rollouts (92.6∼98.7%). These results suggest that parameter-shared teachers often inherit the student's underlying failure modes.

4.3. The Gap Between Scoring and Generation

Takeaways 1∼2 show that the teacher's token at t^* rarely provides a direct local repair, particularly at 4B/8B where t^* frequently corresponds to non-code text and direct repair rates remain low (2.2∼3.9% would fix). Nevertheless, teacher continuations from t^* produces correct code in 10.5∼13.3% of failed rollouts, including cases where t^* is non-algorithmic. This gap suggests that the teacher may contain corrective capabilities not reflected by isolated token substitution alone, but emerge through alternative generation trajectories. Cross-tabulating the judge's *teacher would fix* label against execution outcomes confirms this disconnect: the local label does not predict trajectory-level success, and at 4B/8B, non-code text positions yield higher continuation correctness than algorithmic positions (Appendix A.4).

Figure 2 illustrates this distinction concretely. The student and teacher diverge at a non-algorithmic token: the student reasons "`j` = (i+k)//2", whereas the teacher instead prefers "`k` = 2j-i". Replacing the token alone cannot repair the student's missing parity check. However, the teacher's altered reasoning trajectory leads to a different algorithm that correctly solves the task. The KL signal at t^* therefore reflects a preference associated with an alternative reasoning path. Although insufficient to repair the current rollout directly, this signal may still influence subsequent generations: producing the teacher-preferred token could shift the continuation toward a different algorithm trajectory (Section 5).

Takeaway 3. Teacher continuations reveal trajectory-level corrective behavior beyond local token repair

At 4B/8B, the teacher-preferred token at t^* directly repairs only 2.2∼3.9% of failures, whereas teacher continuations from the same position generate correct code in 10.5∼13.3% of cases. These results suggest that the useful information in the self-distillation signal may arise less from isolated token correction and more from its association with alternative generation trajectories.

4.4. The KL Signal at t^* Steers the Student's Trajectory

Finding 3 suggests that the teacher's corrective capability emerges primarily through alternative generation trajectories rather than isolated token substitution. However, the KL objective at t^* still encourages the student toward the teacher-preferred token. If the student later produces that token during generation, can this perturbation shift the continuation toward a different and potentially correct algorithm trajectory? To test this hypothesis, we replace the student's token at t^* with the teacher's argmax token and allow the *student* to continue generation autoregressively. As a control, we repeat the procedure using the student's original token at t^*, isolating the effect of token-token substitution from the decoding procedure itself.

Student trace

Pre-code reasoning trace

```
We need to count triples (i, j, k) where S[i]='A',
S[j]='B', S[k]='C' and j - i = k - j (equally
spaced).

This means j = (i + k) // 2, so we can iterate
over i and k.
```

Generated code

```
for i in range(n):
  for k in range(i + 2, n):
    j = (i + k) // 2
    if S[i]=='A' and S[j]=='B'
      and S[k]=='C':
      count += 1
```

× **Bug:** no parity check for `(i+k)%2==0`; counts invalid half-integer indices

Teacher continuation from t^*

Pre-code reasoning trace

```
We need to count triples (i, j, k) where S[i]='A',
S[j]='B', S[k]='C' and j - i = k - j (equally
spaced).

This means k = 2j - i, so we can iterate over i
and j.
```

Generated code

```
for i in range(n):
  for j in range(i + 1, n):
    k = 2 * j - i
    if k < n and S[i]=='A'
      and S[j]=='B' and S[k]=='C':
      count += 1
```

✓ **Correct:** iterates (i,j), computes k=2j−i; k is always an integer

Figure 2. Illustrative example where teacher continuation produces correct code despite divergence at a non-algorithmic token (Qwen3-8B). t^* = `j` with KL concentration ratio 12.3%, classified as a naming token. The student reasons using "`j` = (i+k)//2" while the teacher instead prefers "`k` = 2j-i". Substituting this single prose token cannot fix the student's missing parity check. However, when the teacher continues generation from t^*, the altered reasoning trajectory produces a different algorithm (iterating over `i`, `j` and computing `k`) that avoids the bug, demonstrating how trajectory-level divergence can yield a correct solution even when a token substitution is insufficient.

Table 1 (execution-based evaluation) reveals a scale-dependent trajectory effect. At 1.7B, teacher-token substitution provides no measurable improvement over the control (1.5% vs. 1.5%), suggesting that the smaller model may lack sufficient capacity to leverage a single-token perturbation into a different algorithmic continuation. At 8B, however, the effect becomes substantial: 9.9% vs. 1.7% (5.8×), and 17 of the 18 successful student continuations overlap with successful teacher continuations, suggesting that similar solution trajectories are being recovered. The single-token *teacher would fix* metric (2.2∼3.9% at 4B/8B) therefore underestimates the broader influence of the KL signal, since it measures only one token *alone* changes correctness, not whether it cascades into a different algorithm through subsequent generation.

Takeaway 4. The KL signal at t^* steers the student's trajectory at scale

Substituting the teacher's token at t^* and letting the student generate freely yields 9.9% correct code at 8B (vs. 1.7% control), recovering 71% of the teacher's free-continuation successes. These results suggest localized KL supervision can bias downstream generation toward alternative solution trajectories at larger scales.

4.5. Is There Useful Signal Beyond t^*?

Finding 3 suggests that the teacher's corrective capability emerges primarily through free continuation beyond t^*, rather than through isolated token substitution. However, during SDPO training, the teacher evaluates the student's continuation after t^*, rather than its own preferred continuation trajectory. This raises a natural question: does the KL signal after t^* still provide useful supervision, or does it primarily reflect teacher evaluations on student continuations that diverge from the teacher's preferred trajectory?

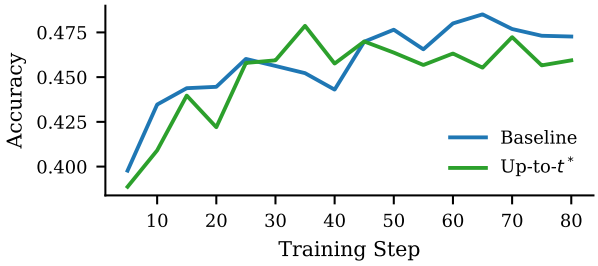


Figure 3. Validation accuracy on LiveCodeBench (Qwen3-1.7B, 80 training steps). Up-to-t^* nearly matches the baseline, suggesting that much of the useful KL supervision is concentrated near t^*.

To study this effect, we introduce a loss-truncation variant, **Up-to-t^***, which retains KL supervision only up to the peak-KL position t^*, while dropping the remaining tokens.

Figure 3 shows validation performance over 80 training steps. Up-to-t^* removes KL supervision on all tokens after the peak divergence position, eliminating supervision conditioned on the student's subsequent continuation. Despite this truncation, the variant achieves performance close to the full baseline throughout training (0.459 vs. 0.473 at step 80). The limited performance gap suggests that much of the useful supervision may already be concentrated near t^*, consistent with the scoring-generation distinction identi-

fied in Section 4.3. Beyond this point, the teacher evaluates continuations that may diverge substantially from its own preferred generation trajectory.

Takeaway 5. Most useful KL supervision appears up to t^*

Truncating the KL loss after t^* yields performance close to full SDPO training, suggesting that much of the useful supervision signal is concentrated near the peak-KL position. This result is consistent with the observation that teacher evaluations beyond t^* are conditioned on student continuations that may diverge from the teacher's preferred trajectory.

4.6. Do These Behaviors Persist During Training?

We repeat the full token-level analysis throughout Qwen3-1.7B SDPO training at steps 0, 20, 40, 60, and 80 using 256 LiveCodeBench rollouts per step.

Table 2 shows that the trends identified above persist throughout training rather than appearing only at initialization. In particular, the concentration of peak-KL positions on non-executable tokens (Finding 1) and the limited local corrective reliability of teacher guidance (Finding 2) remain broadly consistent across training. Even as pass rate increases from 0.21 to 0.43, direct token substitution using the teacher-preferred token repairs only 9$\sim$12% of failed rollouts, while algorithmically meaningful divergences become less common at later stages of training.

4.7. Cross-Architecture Validation

To test whether the observed behavior is specific to Qwen3, we repeat the analysis on OLMo3-7B-Instruct. The same qualitative patterns persist across architectures. On failed rollouts, 81.4% of peak-KL positions fall in non-code reasoning text, and the teacher-preferred token directly repairs the program in only 3.1% of cases. Teacher continuations succeed on 24.8% of failed rollouts, indicating that the parameter-shared teacher still inherits many of the student's failures despite its higher base capability.

Nevertheless, the continuation intervention again reveals trajectory steering beyond direct token repair. Replacing only the student's token at t^* with the teacher-preferred token increases student continuation success from 6.2% under the own-token control to 10.1%. The smaller gain relative to Qwen3-8B may reflect OLMo3's verbose reasoning style: when t^* occurs within a long chain-of-thought trace, a single-token perturbation has less leverage over the eventual algorithmic continuation.

Together, the step-wise and cross-architecture analyses show that the central pattern is not specific to a particular training stage or model family: peak-KL supervision is often locally non-corrective, yet a teacher-preferred token can still alter the downstream continuation. We next discuss why such concentrated token-level disagreement may remain optimization-useful relative to uniformly assigned rollout-level rewards.

5. Discussion and Limitations

Why self-distillation improves training despite low teacher reliability. Despite its limited direct repair capability, SDPO improves pass rate from 0.21 to 0.43 over 80 steps (Table 2). We hypothesize that this follows from the different optimization structures of SDPO and GRPO.

GRPO broadcasts a scalar normalized advantage uniformly across all response tokens. It therefore provides no positional distinction between relevant and irrelevant tokens and, under binary rewards, assigns zero advantage to every rollout when all samples in a group fail. At step 0, such all-fail groups account for 59–72% of problems and cover 84–91% of failed rollouts (Table 4).

In contrast, SDPO compares the student and feedback-conditioned teacher distributions at each position, providing KL supervision even for these failed rollouts. This supervision is strongly localized: t^* alone accounts for 9.1–19.5% of response KL, a 40–80$\times$ concentration relative to uniform allocation. The continuation experiments further show how such localized disagreement can steer generation. Teacher continuation establishes that a correct alternative continuation is sometimes available from the same student prefix, while student continuation with the teacher-preferred token tests whether changing only that token can redirect the student toward it. At 8B, replacing the token at t^* raises student continuation success from 1.7% under the own-token control to 9.9%, recovering 71% of successful teacher continuations. Thus, although the teacher-preferred token directly repairs only 3.9% of failures in 8B (Table 1), optimizing toward it may still provide more informative structure than uniformly distributed outcome-level penalties by biasing generation toward alternative, more successful trajectories.

Limitations. (1) Our analysis focuses on the top-1 KL-divergence token t^* as the primary unit of credit assignment; however, tokens outside t^* collectively account for 81$\sim$91% of total KL (Section 3) and may contribute trajectory-level supervision effects that our top-1-KL position analysis does not fully capture. (2) Semantic role and single-token criticality evaluations rely on an LLM judge (GPT-5.4), which may have systematic biases; continuation correctness uses execution-based ground truth and is not subject to this limitation. (3) All experiments use LiveCodeBench v6; other code or math domains may exhibit different teacher-reliability patterns. (4) Qwen3-8B uses a 2048-token response limit

Table 2. Step-wise evolution of the self-distillation signal on failed rollouts during Qwen3-1.7B SDPO training (steps 0–80, 256 rollouts per step on LiveCodeBench). *Semantic role* and *single-token evaluation* use an LLM judge (GPT-5.4). Numbers in parentheses denote failed rollout counts. Pass rate is computed over all rollouts per step.

	Step 0 (202)	Step 20 (183)	Step 40 (192)	Step 60 (176)	Step 80 (145)
Pass rate	0.211	0.285	0.250	0.312	0.434
Semantic role					
Algorithmic	36.1%	35.5%	36.5%	35.8%	26.2%
Naming	11.4%	15.3%	12.0%	14.8%	9.7%
Formatting	7.4%	8.2%	13.5%	11.4%	15.9%
Non-code text	45.0%	41.0%	38.0%	38.1%	48.3%
Single-token evaluation at t^*					
Student is cause	8.9%	7.7%	6.2%	5.1%	6.9%
Teacher would fix	12.4%	11.5%	9.4%	11.4%	9.7%

Table 3. Self-distillation and teacher reliability signa on OLMo3-7B-Instruct (step 0, 221 non-truncated rollouts on LiveCodeBench). *Semantic role* and *single-token evaluation* use an LLM judge (GPT-5.4). *Execution-based evaluation* uses the test suite: *teacher continuation* lets the teacher generate freely from t^*; *student continuation* substitutes the teacher's (or student's own) token at t^* and lets the student generate freely (greedy decoding). Numbers in parentheses denote rollout counts

	Fail (129)	Succ (92)
Semantic role (student token at t^*)		
Algorithmic	7.8%	15.2%
Naming	3.1%	10.9%
Formatting	7.8%	10.9%
Non-code text	81.4%	63.0%
Single-token evaluation at t^*		
Student is cause	2.3%	–
Teacher would fix	3.1%	–
Execution-based evaluation at t^*		
Teacher continuation	24.8%	69.6%
Student cont. (teacher token)	10.1%	–
Student cont. (own token, control)	6.2%	–

(vs. 4096 for 1.7B/4B) due to GPU memory constraints, resulting the truncation in 0.8% of samples.

6. Conclusion

We studied the nature of token-level supervision in on-policy self-distillation when the teacher is conditioned on rich execution feedback but lacks access to ground-truth solutions. Across Qwen3 series on LiveCodeBench, we find that local corrective quality is often limited: teacher continuations succeed on only 7.9∼13.3% of failed rollouts, peak-KL positions predominantly concentrate on non-executable tokens rather than algorithmically meaningful ones, and a single token is rarely the primary cause of failure (1.1∼8.9%). Nevertheless, self-distillation remains effective because concentrated token-level supervision can still provide useful optimization structure even when local token semantics are imperfect. Compared to uniformly distributed rollout-level rewards (e.g., GRPO), the KL signal remains highly localized, concentrating 40∼80× at peak disagreement positions.

Table 4. GRPO gradient coverage on failed rollouts on Qwen3-1.7B (step 0, K=8 rollouts per problem, 32 problems on LiveCodeBench). All-fail and all-success groups receive zero GRPO advantage.

	1.7B	**4B**	**8B**
Problem groups (out of 32)			
All-fail	23 (71.9%)	19 (59.4%)	19 (59.4%)
All-success	5 (15.6%)	6 (18.8%)	5 (15.6%)
Mixed	4 (12.5%)	7 (21.9%)	8 (25.0%)
Failed rollouts			
Total	202	181	181
GRPO grad = 0	184 (91.1%)	152 (84.0%)	152 (84.0%)
GRPO grad $\neq 0$	18 (8.9%)	29 (16.0%)	29 (16.0%)

Importantly, direct token-repair metrics underestimate the broader influence of the KL signal. At 8B, substituting the teacher-preferred token at t^* and allowing the student to continue generation yields 9.9% correct solutions (vs. 1.7% under the control substitution), recovering 71% of successful teacher continuations. These results suggest that the KL signal at t^* may influence downstream generation behavior beyond immediate local repair. In particular, localized token-level disagreement may influence downstream generation trajectories even when the locally preferred token is not directly corrective. This effect is strongly scale-dependent, appearing negligible at 1.7B but substantial at 8B, suggesting that larger models may be better able to amplify small token-level perturbations into qualitatively different continuations. Future work could strengthen these trajectory-level effects through richer feedback structures (e.g., error localization or repair hints) or through off-policy distillation from teacher-generated continuations.

Acknowledgments

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2024-00444862, Non-invasive near-infrared based AI technology for the diagnosis and treatment of brain diseases), and the National Research Foundation of Korea (NRF), funded by the Ministry of Science and ICT (MSIT) under grant RS-2024-00464269.

Impact Statement

This work advances the understanding of token-level supervision in reinforcement learning with verifiable rewards, with direct relevance to mathematical reasoning tasks where models receive sparse final-answer feedback despite producing long reasoning chains. Our findings suggest that dense self-distillation may help not by providing locally correct token repairs, but by steering models toward alternative reasoning trajectories. This perspective can inform better training objectives for math problem-solving, theorem-proving, and other domains of verifiable reasoning.

The potential positive impact is improved reliability and interpretability of training signals for AI systems used in mathematics, formal verification, and scientific problem solving. However, our results also caution that dense supervision can emphasize superficial or locally misleading tokens rather than semantically valid reasoning steps. Future work should therefore evaluate these mechanisms directly on mathematical benchmarks and combine token-level supervision with checks for intermediate mathematical validity.

References

Cui, G., Yuan, L., Wang, Z., Wang, H., Zhang, Y., Chen, J., Li, W., He, B., Fan, Y., Yu, T., et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Gehring, J., Zheng, K., Copet, J., Mella, V., Cohen, T., and Synnaeve, G. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *International Conference on Machine Learning*, pp. 19034–19055, 2025.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Huang, H., Song, J., Zhang, Y., and Ren, P. SelecTKD: Selective token-weighted knowledge distillation for LLMs. *arXiv preprint arXiv:2510.24021*, 2025.

Hübotter, J., Lübeck, F., Behric, L., Baumann, A., Bagatella, M., Marta, D., Hakimi, I., Shenfeld, I., Buening, T. K., Guestrin, C., et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Kazemnejad, A., Aghajohari, M., Portelance, E., Sordoni, A., Reddy, S., Courville, A., and Le Roux, N. Vineppo: Refining credit assignment in rl training of llms. *arXiv preprint arXiv:2410.01679*, 2024.

Kim, J., Luo, X., Kim, M., Lee, S., Kim, D., Jeon, J., Li, D., and Yang, Y. Why does self-distillation (sometimes) degrade the reasoning capability of LLMs? *arXiv preprint arXiv:2603.24472*, 2026.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2024.

Liu, J., Zhu, Y., Xiao, K., Fu, Q., Han, X., Yang, W., and Ye, D. RLTF: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*, 2023.

Liu, Z., Kim, J., Luo, X., Li, D., and Yang, Y. Exploratory memory-augmented LLM agent via hybrid on- and off-policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=UOzxviKVFO.

Parthasarathi, P., Reymond, M., Chen, B., Cui, Y., and Chandar, S. GRPO-λ: Credit assignment improves LLM reasoning. *arXiv preprint arXiv:2510.00194*, 2025.

Penaloza, E., Vattikonda, D., Gontier, N., Lacoste, A., Charlin, L., and Caccia, M. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y., Wu, Y., and Guo, D. DeepSeekMath: Pushing

the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Song, Y., Chen, L., Tajwar, F., Munos, R., Pathak, D., Bagnell, J. A., Singh, A., and Zanette, A. Expanding the capabilities of reinforcement learning via text feedback. *arXiv preprint arXiv:2602.02482*, 2026.

Tavor, A., Ebenspanger, I., Cnaan, N., and Geva, M. Rethinking selective knowledge distillation. *arXiv preprint arXiv:2602.01395*, 2026.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2024.

Wang, S., Yu, L., Gao, C., Zheng, C., Liu, S., Lu, R., Dang, K., Chen, X., Yang, J., Zhang, Z., Liu, Y., Yang, A., Zhao, A., Yue, Y., Song, S., Yu, B., Huang, G., and Lin, J. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

Xu, Y., Sang, H., Zhou, Z., He, R., Wang, Z., and Geramifard, A. TIP: Token importance in on-policy distillation. *arXiv preprint arXiv:2604.14084*, 2026.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Yang, Z., Luo, X., Wang, Z., Han, D., He, Z., Li, D., and Xu, Y. Do not let low-probability tokens over-dominate in RL for LLMs. *arXiv preprint arXiv:2505.12929*, 2025b.

Zhao, S., Xie, Z., Liu, M., Huang, J., Pang, G., Chen, F., and Grover, A. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

A. Supplementary Analysis

A.1. KL Concentration CDF

The KL concentration ratio $C = \mathrm{KL}(t^*)/\sum_t \mathrm{KL}(t)$ measures how much of the total response KL is carried by the single peak token. Figure 4 shows the cumulative distribution across the 202 failed rollouts (Qwen3-1.7B). The median KL concentration ratio is $\sim$19.5%, and the right tail shows that for some responses t^* carries over 50% of the total loss, confirming the extreme sparsity of the self-distillation signal.

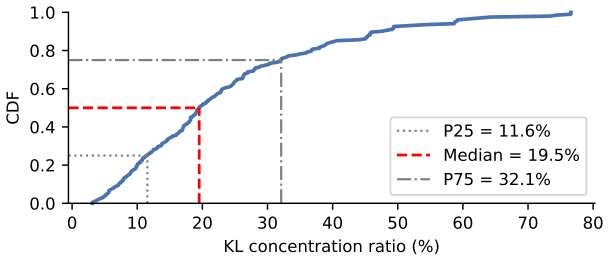


Figure 4. CDF of KL concentration ratios ($\mathrm{KL}(t^*)/\sum_t \mathrm{KL}(t)$) on failed rollouts (Qwen3-1.7B, N=202). The median is $\sim$19.5%; the right tail shows that for some responses, t^* carries over 50% of the total loss signal.

A.2. Semantic Role Breakdown

Table 5 provides the full semantic role breakdown at t^* on failed rollouts, as classified by the LLM judge (GPT-5.4). The student and teacher tokens are classified independently. The student's token is predominantly non-code text at 4B/8B, while algorithmic roles decline with scale.

Table 5. Semantic role classification of student and teacher tokens at t^* on failed rollouts, evaluated by LLM judge (GPT-5.4). Non-code text dominates at larger scales; algorithmic divergences drop from 36% to 12%.

	Role	1.7B	4B	8B
Student	Algorithmic	36.1%	11.6%	11.6%
	Naming	11.4%	3.9%	5.5%
	Formatting	7.4%	13.3%	12.7%
	Non-code text	45.0%	71.3%	70.2%
Teacher	Algorithmic	38.6%	10.5%	14.4%
	Naming	9.9%	4.4%	3.9%
	Formatting	8.4%	11.6%	13.8%
	Non-code text	43.1%	73.5%	68.0%

Figure 5 shows the student-to-teacher semantic role transition matrix at t^* on failed rollouts. The diagonal is dominant at all scales: when the student token is classified as a given role, the teacher token at the same position almost always receives the same classification. The exception is naming at 8B (62% diagonal), where the small sample size (10 naming divergences out of 181 failures) amplifies individual cases where the teacher prefers an algorithmic or formatting token at a naming position.

A.3. Fail-Channel Ablation

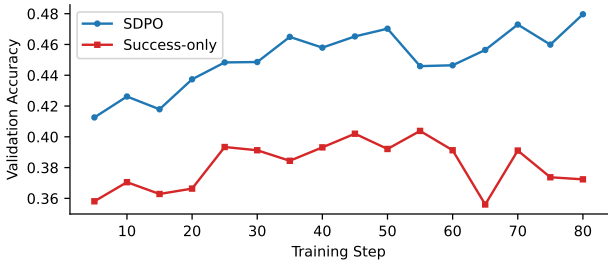


Figure 6. Validation accuracy on LiveCodeBench (Qwen3-1.7B, 80 training steps). Applying the KL loss only to successful rollouts (SUCCESS-ONLY) yields substantially lower accuracy than SDPO throughout training (0.372 vs. 0.480 at step 80), confirming that the fail-channel supervision is the primary driver of SDPO's performance gains.

To verify that the fail-channel KL drives the majority of SDPO's training gains, we compare the default SDPO configuration with a variant that applies the KL loss exclusively to successful rollouts (SUCCONLY). Both use identical hyperparameters on Qwen3-1.7B.

Figure 6 shows that the success-only variant plateaus around 0.39 accuracy, while the baseline reaches 0.48 by step 80. This confirms that the fail channel carries the training signal responsible for SDPO's continued improvement, motivating the analysis of what learning signal the teacher provides on failed rollouts (Section 3).

A.4. LLM-Judge Labels vs. Execution Outcomes

To validate the LLM-judge labels against execution-based ground truth, Table 6 cross-tabulates semantic role and *teacher would fix* with teacher continuation correctness on failed rollouts.

Two patterns emerge. First, the judge's *teacher would fix* label does not predict teacher continuation success: at 1.7B, continuation correctness is 8.0% when the judge says the teacher's token would fix the code and 7.9% when it says it would not. At 4B, the relationship reverses (0.0% vs. 10.7%). The judge correctly identifies local token-level relevance, but local relevance does not predict trajectory-level success because the teacher's value comes from redirecting the generation trajectory, not from a single-token substitution.

Second, at 4B/8B, non-code text positions yield *higher* teacher continuation success (13.2% and 15.7%) than algorithmic positions (4.8% and 0.0%). This is counterintuitive if one expects algorithmic divergences to be most action-

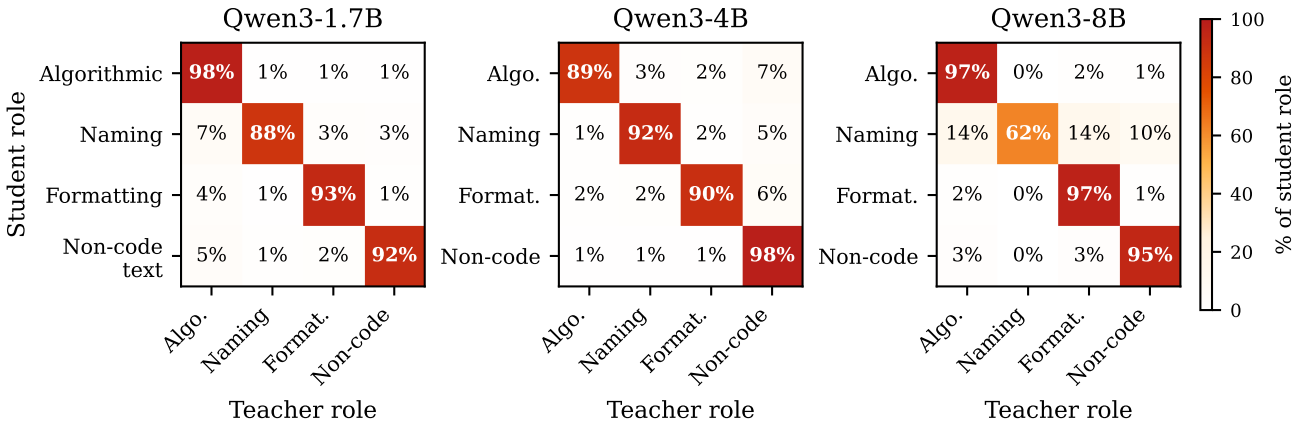


Figure 5. Student-to-teacher semantic role transition at t^* on failed rollouts. Each cell shows the percentage of samples with a given student role (row) where the teacher receives the corresponding role (column). The strong diagonal confirms that student and teacher typically diverge *within* the same semantic category (e.g., different variable names, different comments) rather than across categories.

Table 6. Teacher continuation correctness stratified by semantic role and *teacher would fix* label on failed rollouts. The judge's local *teacher would fix* assessment is uncorrelated with trajectory-level execution success. Non-code text positions have higher continuation success than algorithmic ones at 4B/8B, consistent with the trajectory-steering mechanism (Section 4.3).

	1.7B	4B	8B
By semantic role			(correct / total)
Alg.	9/73 (12.3%)	1/21 (4.8%)	0/21 (0.0%)
Nam.	4/23 (17.4%)	0/7 (0.0%)	2/10 (20.0%)
Fmt.	0/15 (0.0%)	1/24 (4.2%)	2/23 (8.7%)
Text	3/91 (3.3%)	17/129 (13.2%)	20/127 (15.7%)
By Teacher would fix			
True	2/25 (8.0%)	0/4 (0.0%)	1/7 (14.3%)
False	14/177 (7.9%)	19/177 (10.7%)	23/174 (13.2%)

able, but consistent with the trajectory-steering thesis: the teacher's advantage emerges when it redirects the reasoning path from a prose-level divergence, not when it corrects a single algorithmic token in isolation.

B. Logit-Gradient Derivation for Reverse KL

We derive the per-position logit gradient of the reverse KL loss (Eq. 1) used in Section 3.1. Let $p = p_\theta(\cdot \mid x, y_{<t})$ denote the student distribution at position t, parameterized by logits $z_t \in \mathbb{R}^{|V|}$ (where $|V|$ is the vocabulary size) via $p_v = \mathrm{softmax}(z_t)_v$. We write $z_{t,i}$ for the i-th component of z_t (the logit for vocabulary token i). Let $q = p_\phi(\cdot \mid x, y_{<t}, f)$ denote the teacher distribution, treated as constant (stop-gradient).

Step 1: Differentiate. The per-position reverse KL is $D_{\mathrm{KL}}(p\|q) = \sum_v p_v \log(p_v/q_v)$. Differentiating with respect to $z_{t,i}$:

$$\frac{\partial D_{\mathrm{KL}}}{\partial z_{t,i}} = \sum_v \frac{\partial p_v}{\partial z_{t,i}} \left(\log \frac{p_v}{q_v} + 1 \right). \tag{5}$$

Step 2: Apply the softmax Jacobian. The softmax Jacobian is $\partial p_v / \partial z_{t,i} = p_v(\delta_{vi} - p_i)$. Since $\sum_v \partial p_v / \partial z_{t,i} = 0$, the constant term $+1$ vanishes:

$$\frac{\partial D_{\mathrm{KL}}}{\partial z_{t,i}} = \sum_v p_v(\delta_{vi} - p_i) \log \frac{p_v}{q_v}. \tag{6}$$

Step 3: Expand and simplify. Splitting the sum:

$$\frac{\partial D_{\mathrm{KL}}}{\partial z_{t,i}} = p_i \log \frac{p_i}{q_i} - p_i \underbrace{\sum_v p_v \log \frac{p_v}{q_v}}_{= D_{\mathrm{KL}}(p\|q)} = p_i \left(\log \frac{p_i}{q_i} - D_{\mathrm{KL}} \right). \tag{7}$$

Step 4: Vector form.

$$\nabla_{z_t} D_{\mathrm{KL}}(p\|q) = p \odot \left(\log \frac{p}{q} - D_{\mathrm{KL}} \cdot \mathbf{1} \right). \tag{8}$$

Interpretation. The gradient vanishes exactly when $p = q$ (both terms are zero). Each component is weighted by p_i: tokens the student assigns near-zero probability receive near-zero gradient, reflecting reverse KL's mode-seeking behavior. For the teacher's preferred token $v^* = \arg\max_v q_v$, $\log(p_{v^*}/q_{v^*})$ is typically negative and $D_{\mathrm{KL}} > 0$, so the gradient component is negative; an SGD update $z_{t,v^*} \leftarrow z_{t,v^*} - \eta\, \partial D_{\mathrm{KL}} / \partial z_{t,v^*}$ increases z_{t,v^*}, pushing the student toward the teacher's argmax.

From logits to parameters. The parameter gradient at position t is $\nabla_\theta \mathcal{L}_t = (\nabla_{z_t} \mathcal{L})^\top (\partial z_t / \partial \theta)$, where $\partial z_t / \partial \theta$ is the model Jacobian. Since $\|\nabla_{z_t}\|$ scales with $\mathrm{KL}(t)$ through

the $\log(p/q)$ terms, KL concentration tracks logit-gradient concentration. However, the model Jacobian varies by position, so parameter-gradient concentration may differ from logit-gradient concentration.

C. Evaluation Protocol

All experiments are conducted on 8 NVIDIA RTX 3090 GPUs. We follow the official SDPO implementation and hyperparameters unless specified. Our t^* importance analysis pipeline runs two evaluation phases per sample: (1) LLM judge-based divergence classification, and (2) verifiable environment-based teacher continuation correctness evaluation.

C.1. Divergence Classification

All LLM judge evaluations use GPT-5.4 with temperature 0 and a maximum of 1024 completion tokens. For each divergence position (t^* and first-divergence), the judge receives the problem statement, the student's full solution, and a 15-token context window centered on the divergence. It independently classifies both the student and teacher tokens along two axes and provides two criticality judgments:

- **Region**: `code`, `comment`, `docstring`, `string_literal`, or `whitespace`.
- **Semantic role**: `algorithmic` (affects control flow, algorithm, or computation), `naming` (identifier that does not change behavior), `formatting` (whitespace with no semantic effect), or `scaffolding` (comments, docstrings, code fences, prose).
- **student_is_cause**: whether this specific student token is the primary cause of the code failing.
- **teacher_would_fix**: whether substituting the teacher's token at this position could change correctness.

The full system prompt is shown below.

Divergence Classification System Prompt

```
You are an expert in analyzing LLM-generated
code for competitive programming.

You will be shown a position in
student-generated code where the student's
predicted token diverges from a teacher model's
preferred token.  You must:
  (A) Classify BOTH the student token and the
teacher token independently.
  (B) Judge whether this specific token position
is critical for making the code correct.

=== (A) Per-token classification ===

For EACH token (student and teacher), provide
two labels:

1.  region - what kind of source text this token
belongs to.
   Decide by looking at the surrounding context,
not the token in isolation.
   - "code":  part of an executable Python
statement or expression.
   - "comment":  part of an inline comment
(after #) or a block comment.
   - "docstring":  part of a triple-quoted
documentation string.
   - "string_literal":  part of a string
constant used in code.
   - "whitespace":  spaces, tabs, newlines, or
indentation with no semantic effect.

2.  semantic_role - what role this specific
token plays in the solution.
   - "algorithmic":  determines control flow,
algorithm choice, data structure selection, or
computation semantics.  Changing this token
would produce a different execution path or
result.
   - "naming":  a variable/function/parameter
name where a different name would not change
behavior.
   - "formatting":  whitespace or indentation
that does not change Python block scope.
   - "scaffolding":  comments, docstrings,
code-fence markers, or natural-language prose.

IMPORTANT: student and teacher tokens can have
DIFFERENT labels.

=== (B) Criticality judgment ===

1.  student_is_cause:  "Is THIS specific student
token the main cause of the code failing?"
   - true:  this token is where the student
makes the mistake that leads to incorrect
output.
   - false:  this token is not the primary cause
of failure.

2.  teacher_would_fix:  "If the student used the
teacher's token instead at THIS position, could
it fix the code?"
   - true:  substituting the teacher's token
here could change correctness.
   - false:  substituting the teacher's token
would NOT change correctness.

Respond ONLY with valid JSON on a single line:
{"student":  {"region":  "...", "semantic_role":
"..."}, "teacher":  {"region":  "...",
"semantic_role":  "..."}, "student_is_cause":
true/false, "teacher_would_fix":  true/false,
"reason":  "..."}
```

The user message provides the problem, the student's full solution, and the divergence context:

Divergence Classification User Prompt Template

```
Problem:
{problem statement}

Student's full generated solution:
{student output}

-- Divergence at position {pos}/{resp_len}
({pct}% through response) --
Student token:  {student_tok}
Teacher token:  {teacher_tok}

Context (15 tokens before -> [DIVERGENCE] -> 15
tokens after):
{ctx_before}[{student_tok}]{ctx_after}
```

C.2. Teacher Continuation Correctness

For each divergence position, we construct the teacher's continuation by taking the student's response prefix up to t^*,

replacing the student's token with the teacher's argmax token, and generating freely from that point using the teacher model (up to the full response length limit: 4096 tokens for 1.7B/4B, 2048 for 8B). The resulting text is parsed to extract the Python code block, which is then executed against the full LiveCodeBench private test suite. A continuation is scored as correct if and only if it passes all test cases.