CLOAKTEACH: Teacher Adaptation for Iterative Self-Distillation in Large Language Models

Anonymous submission

Abstract

Self-distillation has emerged as a promising self-supervised optimization paradigm for large language models, where the same model simultaneously acts as both teacher and student, enabling learning from self-generated supervision without relying on external stronger models. However, because the teacher operates with privileged reference information, self-distillation still suffers from three critical limitations: the teacher may explicitly leak reference information, fail to properly follow the reference while generating incorrect reasoning trajectories, or produce implicit reasoning-path mismatches that are difficult for the student to reproduce, thereby reducing distillation efficiency and training stability.

To address these issues, we propose CLOAKTEACH, a teacher adaptation framework that dynamically attaches Cloak, a lightweight adapter, to regulate teacher-generated reasoning trajectories for student learning. Our method consists of two stages. First, we train Cloak, a low-rank LoRA adapter optimized with three coordinated reinforcement learning objectives: privileged-information suppression, reference following, and student adaptation. Second, we perform iterative self-distillation, where the teacher loads Cloak during rollout generation to produce student-compatible trajectories and unloads it during EMA fusion with the student model, enabling stable co-evolution between teacher and student. Experimental results show that integrating CLOAKTEACH into self-distillation frameworks yields 2%–13% performance improvements diverse reasoning domains.

1 Introduction

As large language models are increasingly deployed across diverse reasoning tasks, post-training has become essential for improving task-specific reasoning capabilities (Naveed et al., 2025). Existing approaches, however, face distinct limitations. Supervised fine-tuning (SFT) relies on large-scale, high-quality labeled datasets but suffers from overfitting and catastrophic forgetting (Luo et al., 2025). Reinforcement learning from verifiable rewards (RLVR), exemplified by GRPO, depends on well-designed reward models or functions, with its training effectiveness constrained by the model's inherent exploration capacity (Chen et al., 2026).In contrast, model distillation offers an efficient way to train domain-specific small models using supervision from stronger models, yet its effectiveness is often limited by teacher-student capability gaps and distribution mismatches between models (Zhang et al., 2024a; Hsieh et al., 2023; Zhang et al., 2024b).

Among the various paradigms, self-distillation has emerged as a promising direction, where a model learns supervision signals from its own generated outputs instead of relying on external superior models (Huang et al., 2023). Compared with conventional supervised adaptation, self-distillation offers two major advantages: it removes dependence on stronger external teachers while leveraging readily available internal supervision signals, and its self-reinforcing training mechanism naturally preserves behavioral consistency, thereby alleviating catastrophic forgetting during iterative optimization (Yang et al., 2024).

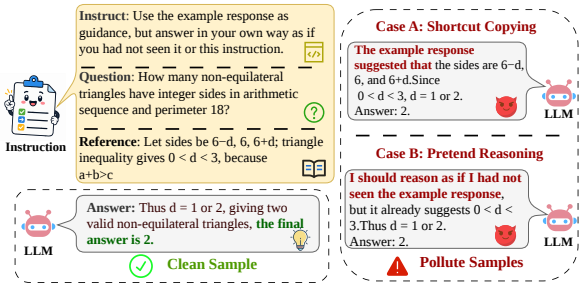

Figure 1: Referrence leakage in self-distillation.

Table 1: Comparison of different post-training paradigms for large language models.

Method	Trajectory Source	Accessibility	Correctness Supervision	Training Stability	Continual Learning
SFT	annotations	Low (human annotation)	High (common correct)	High (annotations quality)	Low (forgets easily)
RLVR	Self-generated	Medium (design reward)	Medium (self-exploration)	Medium (reward depends)	High (self-generated)
Model Distillation	Superior model	Medium (superior model)	Medium (teacher model)	High (style consistency)	Medium (different model)
Self-Distillation (SDFT)	Reference + self-generated	Medium (simple reference)	Medium (reference-guided)	Medium (collapses possibility)	Medium (input discrepancy)
CloakTeach (Ours)	Reference + self-generated	Medium (simple reference)	High (cloak-constrained)	High (cloak-constrained)	High (self-generated)

Despite its strengths, self-distillation has inherent limitations. Though teacher and student share the same backbone, the teacher possesses privileged reference information, whose reasoning trajectory quality determines distillation effectiveness. As shown in Fig. 1, the teacher may leak such information and form shortcut supervision unreplicable by students. Even with reference available, it can still generate illogical reasoning and wrong answers. Additionally, asymmetric information access causes reasoning path mismatch: students infer merely from original inputs. This gap reduces learning efficiency and destabilizes iterative self-distillation.

To address these issues, we propose CloakTeach, a teacher adaptation framework for iterative self-distillation that dynamically attaches a lightweight adapter to regulate teacher-generated reasoning trajectories for student learning. As summarized in Table 1, CloakTeach achieves strong correctness supervision, training stability, and continual learning ability while maintaining moderate accessibility, compared with existing methods.

Our framework consists of two stages. First, we train Cloak, a lightweight LoRA adapter optimized with three coordinated reinforcement learning objectives: privileged-information suppression, reference following, and student adaptation. Together, these objectives encourage the teacher to effectively utilize privileged references while generating reasoning trajectories that remain learnable for the student. Second, we perform iterative self-distillation within Cloak. During rollout generation, the teacher activates Cloak to produce calibrated reasoning trajectories and detaches it during EMA fusion with the student model, enabling stable co-evolution between teacher and student policies.

We systematically evaluate CloakTeach across mathematical, scientific, tool-use, and medical reasoning tasks, covering both thinking and non-thinking settings, and comparing CloakTeach with SFT, GRPO, self-distillation, and distillation from larger models. Experimental results show that CloakTeach achieves simultaneous improvements in reasoning accuracy and training stability across multiple rounds of iterative self-distillation, while also demonstrating strong out-of-distribution generalization and continual learning performance. Our contributions are summarized as follows:

- We identify reference leakage, incorrect reference following, and teacher–student reasoning-path mismatch in self-distillation.
- We train Cloak, a lightweight adapter, with coordinated RL objectives to calibrate teacher-generated reasoning.
- We introduce CloakTeach, an iterative self-distillation framework that couples Cloak with EMA for stable co-evolution.

2 Preliminary

We present the notation and formulation of self-distillation (SDFT), which follows (Yang et al., 2024). Given an input question x and reference information r, the policies of teacher and student models in self-distillation are formalized as:

$$\pi_{\text{tea}}(y_t \mid x, r) = \pi_\theta\big(y_t \mid p_t(x, r)\big) \tag{1}$$

$$\pi_{\text{stu}}(y_s \mid x) = \pi_\theta\big(y_s \mid p_s(x)\big) \tag{2}$$

where p_t and p_s are the prompt templates for the teacher and student, respectively. With i denotes the i-th token in the sequence, and using supervised fine-tuning, the training loss is defined as:

$$\mathcal{L} = -\sum_{x, y_t \in \mathcal{D}} \sum_{i=1}^{|y_t|} \log \pi_{\text{stu}}(y_{t,i} \mid p_s(x), y_{t,<i}) \tag{3}$$

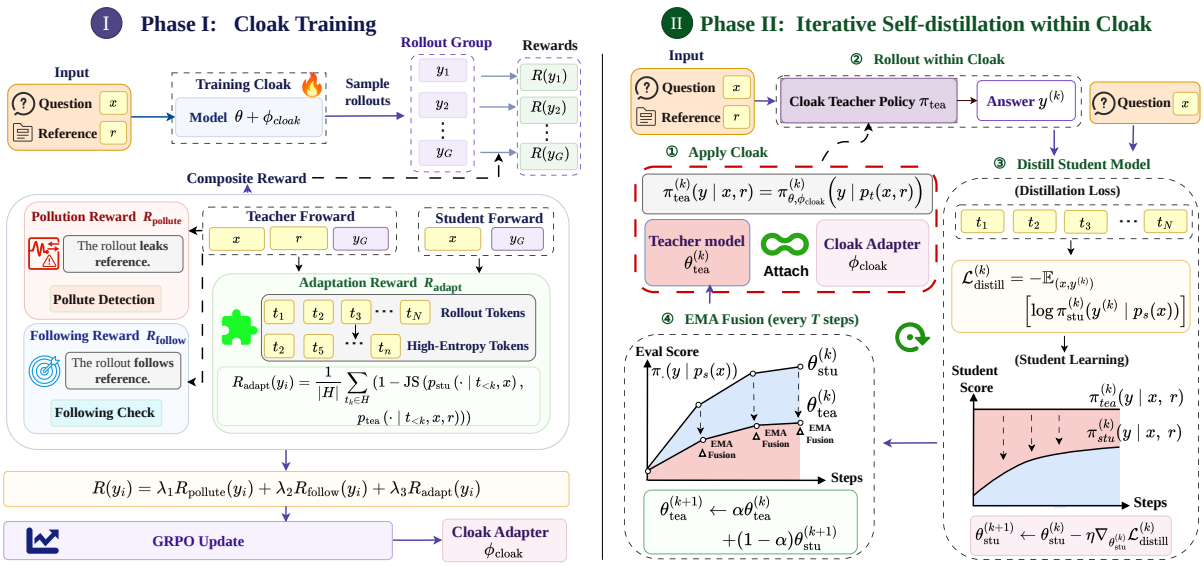

Figure 2: CLOAKTEACH Framework.

3 Method

We present CLOAKTEACH, a two-stage self-distillation approach. We first train Cloak, a lightweight LoRA adapter deployed on the teacher side, to regularize the teacher model and steer it toward reasoning trajectories suitable for self-distillation. We further build an iterative self-distillation framework, where the dynamically updated teacher embedded with Cloak enables mutual evolutionary optimization of teacher and student models.

3.1 Cloak Training

We train Cloak to regularize the teacher model toward generating self-distillation-friendly reasoning rollouts. We use LoRA to fine-tune the teacher model to obtain Cloak, and adopt Group Relative Policy Optimization (GRPO) for optimization. The teacher model during CLOAK training is defined as:

$$\pi_{\text{tea}}(y \mid x, r) = \pi_{\theta,\phi_{\text{cloak}}}\big(y \mid p_t(x, r)\big) \tag{4}$$

For each input x, the teacher policy samples a group of G reasoning trajectories:

$$\{y_i\}_{i=1}^{G} \sim \pi_{tea}(\cdot \mid x, r) \tag{5}$$

We define the rollout reward using three objectives:

$$\begin{aligned}\mathcal{R}(y_i) = \lambda_1 \mathcal{R}_{\text{pollute}}(y_i) + \lambda_2 \mathcal{R}_{\text{follow}}(y_i) \\ + \lambda_3 \mathcal{R}_{\text{adapt}}(y_i)\end{aligned} \tag{6}$$

where λ_1, λ_2, and λ_3 are configurable hyperparameters controlling the contribution of each reward component. Detailed settings and reward calculation details are provided in Appendix A.

Pollution Reward. $\mathcal{R}_{\text{pollute}}$ quantifies whether the teacher's rollout leaks reference information inaccessible to the student model (Figure 2). Such leakage undermines self-distillation, as the student cannot access the privileged context of the teacher. To address the issue, we penalize outputs explicitly exposing reference information or proprietary instructions. This reward adopts a scoring function integrating pattern matching and llm-based identification on teacher rollouts.

Following Reward. $\mathcal{R}_{\text{follow}}$ assesses whether the teacher model strictly adheres to the reference information in the teacher prompt and avoids generating flawed reasoning trajectories. This reward incentivizes the teacher to produce logically consistent rollouts faithfully aligned with the given reference. We implement this reward with a scoring function combining rule-based matching and LLM-as-a-judge evaluation on teacher rollouts.

Adaptation Reward. $\mathcal{R}_{\text{adapt}}$ measures the reproducibility of teacher-generated rollouts by the student model. Given a rollout $y_i = \{t_1, \ldots, t_n\}$, we compute the average JS divergence between the token-level output distributions of the teacher and student models on a subset of high-entropy tokens.

$$\begin{aligned}\mathcal{R}_{\text{adapt}}(y_i) = \frac{1}{|\mathcal{H}|} \sum_{t_k \in \mathcal{H}} \Big(1 - \text{JS}\big(p_{\text{stu}}(\cdot \mid t_{<k}, x), \\ p_{\text{tea}}(\cdot \mid t_{<k}, x, r)\big)\Big)\end{aligned} \tag{7}$$

where $\mathcal{H}$ denotes the set of high-entropy tokens. Note that JS divergence $\in [0, 1]$, with 0 meaning identical distributions. A higher reward (lower JS divergence) indicates better compatibility with the student model. We restrict to high-entropy tokens to avoid low-entropy evasion, where deterministic tokens yield artificially high likelihood.

3.2 Iterative Self-distillation within Cloak

We propose an iterative self-distillation framework in which a teacher model equipped with CLOAK generates self-distillation-friendly reasoning rollouts, while a student model learns from these trajectories and is used to periodically update the teacher via EMA fusion. The workflow is structured around four core operations.

Apply Cloak. We first define the teacher's rollout distribution with Cloak applied (teacher policy):

$$\pi_{\text{tea}}^{(k)}(y \mid x, r) = \pi_{\theta_{\text{tea}}^{(k)}, \phi_{\text{cloak}}}\big(y \mid p_t(x, r)\big). \quad (8)$$

Rollout within Cloak. At each distillation step k, the teacher model applies the CLOAK module and generates a calibrated reasoning trajectory conditioned on the input x:

$$y^{(k)} \sim \pi_{\text{tea}}^{(k)}(\cdot \mid x, r). \quad (9)$$

The CLOAK module shapes the teacher's output distribution to produce trajectories that are optimized for student learning, after which it is immediately detached to isolate its effect from subsequent parameter updates.

Student Learning. The student model is optimized to mimic the teacher's calibrated rollouts using a standard distillation loss:

$$\mathcal{L}_{\text{distill}}^{(k)} = -\mathbb{E}_{(x, y^{(k)})}\left[\log \pi_{\text{stu}}^{(k)}(y^{(k)} \mid x)\right]. \quad (10)$$

Through this objective, the student acquires the teacher's reasoning patterns while benefiting from the CLOAK-enhanced teaching signal. The student parameters are then updated via gradient descent:

$$\theta_{\text{stu}}^{(k+1)} \leftarrow \theta_{\text{stu}}^{(k)} - \eta \nabla_{\theta_{\text{stu}}^{(k)}} \mathcal{L}_{\text{distill}}^{(k)}. \quad (11)$$

EMA Fusion. After every T optimization steps, the teacher base model is updated via EMA fusion without CLOAK:

$$\theta_{\text{tea}}^{(k+1)} \leftarrow \alpha\, \theta_{\text{tea}}^{(k)} + (1 - \alpha)\, \theta_{\text{stu}}^{(k+1)}. \quad (12)$$

The CLOAK module is applied only during rollout generation and is detached during EMA fusion; this decoupled design ensures stable iterative improvement of both models.

4 Experiments

We evaluate CLOAKTEACH across a diverse set of reasoning tasks. Our experiments aim to explore three core aspects: (i) whether the Cloak module can be trained to alleviate inherent defects in self-distillation; (ii) whether the optimized module sustains stable performance during iterative self-distillation; (iii) whether the complete CLOAKTEACH framework achieves better reasoning capability against existing approaches.

4.1 Experimental Setup

Datasets. We select datasets from three domains, mathematics, science, and tool use, for training and evaluating CLOAKTEACH, and reserve a medical dataset for out-of-distribution (OOD) evaluation.

(1) **Mathematical reasoning.** We use Hendrycks et al. (2021) and select the most challenging Level 4-5 problems.

(2) **Scientific reasoning.** We use the mcq-4-choices subset from Feng et al. (2024), restricted to chemistry at Level 4.

(3) **Tool-use reasoning.** We adopt Liu et al. (2024) and further filter to examples with at least one function schema containing over five parameters and at least two valid answer candidates, emphasizing multi-step tool planning and complex function-call generation.

(4) **medical reasoning.** We use Pal et al. (2022), retaining single-choice questions with expert explanations exceeding 100 words. We randomly sample the p50–p90 difficulty percentile using a heuristic score combining question length, explanation length, clinical terminology, and subject weights.

For datasets without an official test split, we construct evaluation part via random partitioning.

Model. We conduct all experiments based on Qwen3-8B (Team, 2025b,c), including the training of the Cloak module and iterative self-distillation pipeline. Meanwhile, we adopt Qwen3-30B-A3B (Team, 2025a) as the external supervision model for knowledge distillation comparison.

Training Pipeline Our framework involves two successive training stages: Cloak module optimization and iterative self-distillation. Detailed implementation configurations are elaborated in Appendix A,B. *(1) Cloak Training.* We randomly sample 1,000 instances covering mathematics, science and tool-use domains for training the Cloak adapter, with another 1,000 reserved samples serving as the

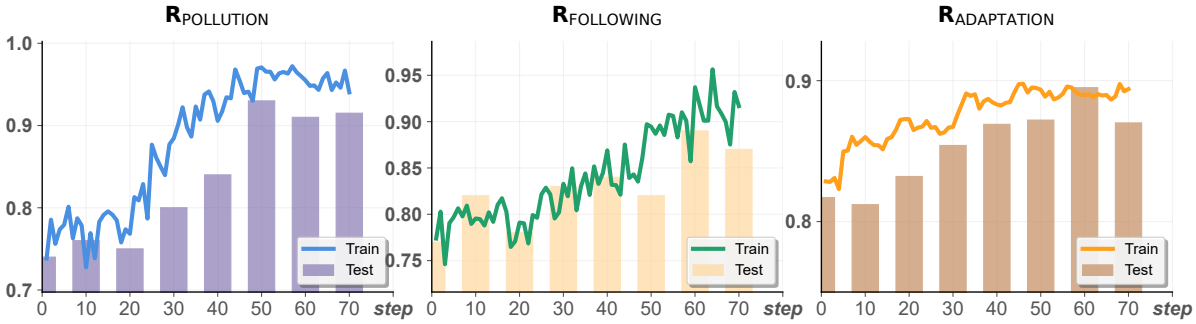

Figure 3: Training dynamics of Cloak. We report the pollution reward, following reward, and adaptation reward across training steps on both training and evaluation sets.

evaluation set. *(2) Iterative Self-Distillation within Cloak.* With the well-trained Cloak deployed, we exclude the previously used Cloak training samples and split the leftover dataset into new training and test sets for subsequent distillation.

4.2 Effectiveness of Training Cloak

In this section, we evaluate whether Cloak can be effectively trained to mitigate the three phenomena discussed in Section 1, including privileged-information pollution, instruction-following failure, and student incompatibility during self-distillation. Figure 3 shows the reward statistics of the three objectives across training steps on both the training and evaluation sets. Detailed experimental information are provided in Appendix A.

As shown in Figure 3, all three reward signals steadily rise during training and remain stable on the test split, demonstrating the generalization ability of the Cloak adapter. Among the three objectives, $R_{\text{pollution}}$ achieves the most prominent test improvement, verifying its effectiveness in suppressing privileged-information leakage during teacher generation. Meanwhile, $R_{\text{following}}$ and $R_{\text{adaptation}}$ also grow consistently, enabling the teacher to better follow reference content while maintaining policy compatibility with the student.

We additionally observe that the three objectives do not strongly conflict during optimization. Improving privileged-information suppression does not significantly harm reference following or student adaptation, implying that the lightweight teacher-side adapter can jointly balance the three constraints during training.

4.3 Cloak Stability in Iterative Distillation

In this section, we examine the stability of Cloak throughout the iterative self-distillation process. As the teacher model is updated after each EMA fusion

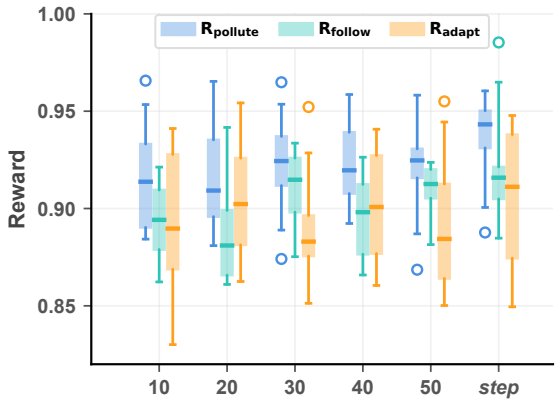

Figure 4: Stability of Cloak in iterative self-distillation.

step, the underlying teacher policy continuously changes over time. A natural question is whether the Cloak remains effective when applied to these updated teacher weights.

To evaluate this, we export teacher-policy rollouts generated at different steps of iterative self-distillation and assess them using the same reward functions employed during Cloak training. Figure 4 presents error-bar plots of these rewards over several steps before each EMA fusion.

As shown in Fig. 4, the reward statistics remain stable across iterations, indicating that Cloak generalizes well to the evolving teacher policy and consistently suppresses leaking information, adheres to references, and generates student-compatible trajectories during iterative self-distillation.

4.4 Effectiveness of CloakTeach

Table 2 presents the overall performance of CloakTeach across mathematical reasoning, scientific question answering and tool usage tasks, under both thinking and non-thinking generation paradigms. We compare our method with widely adopted baselines, including vanilla SFT, GRPO,

Setting	Method	Math			Science			Tool-use		
		Acc@1	Acc@2	Acc@4	Acc@1	Acc@2	Acc@4	Acc@1	Acc@2	Acc@4
Non-Thinking	SFT	61.2	62.2	62.9	58.3	59.6	61.2	68.1	69.3	70.0
	GRPO	64.5	65.1	65.7	68.8	69.3	70.0	63.3	63.9	64.5
	Qwen-30B-Distill	72.2	74.4	75.1	52.7	63.8	67.2	44.6	52.1	55.8
	Self-Distillation	71.1	73.2	74.6	70.5	72.1	73.5	65.4	66.2	67.1
	CloakTeach	**73.1**	**75.1**	**76.9**	**72.2**	**74.1**	**75.5**	**69.2**	**71.3**	**72.7**
Thinking	SFT	-	-	-	-	-	-	-	-	-
	GRPO	80.8	81.4	81.8	74.6	75.4	76.9	70.4	72.1	72.9
	Qwen-30B-Distill	81.4	83.3	86.8	72.0	73.3	74.1	74.3	77.1	79.9
	Self-Distillation	73.2	80.1	81.6	62.9	71.5	73.1	66.9	73.3	74.2
	CloakTeach	**85.4**	**87.7**	**88.3**	**79.2**	**85.5**	**86.3**	**81.2**	**83.5**	**86.1**

Table 2: Main results comparing CloakTeach with baseline methods under thinking and non-thinking settings.

Qwen-30B distill and conventional self-distillation.

Standard SFT learns directly from annotated answers yet lacks sufficient reasoning chains, making it hard to match the reasoning generation style of target models. GRPO optimizes model outputs via ground-truth rewards and self-exploration, which struggles to capture challenging cases effectively. External distillation transfers knowledge from a powerful larger teacher model, while ordinary self-distillation leverages only model-generated trajectories without external supervision signals.

Experimental results demonstrate that CloakTeach achieves consistent superior performance across all tasks and settings. In the non-thinking scenario, our method achieves state-of-the-art Acc@4 scores of 76.9, 75.5 and 72.7 on Math, Science and Tool-use benchmarks, respectively. Compared with conventional self-distillation, our method delivers higher-quality supervision and empowers the student with more robust reasoning capabilities. Unlike external large-model distillation, CloakTeach eliminates the need for a powerful teacher model, freeing the student from external performance and stylistic constraints.

Our advantages are further amplified under the thinking setting. Conventional self-distillation is severely plagued by privileged reference leakage in open-ended reasoning, which produces misleading supervision and repetitive generation behaviors. This inherent defect greatly degrades initial model performance, leading to poor Acc@1 results across all tasks. In contrast, the pollution-aware regularization of CloakTeach effectively suppresses reference leakage and noisy supervision. Our method achieves prominent Acc@4 scores of 88.3, 86.3 and 86.1 on Math, Science and Tool-use tasks, consistently outperforming GRPO and Qwen-30B external distillation. These results validate that our framework generates high-quality, student-friendly reasoning supervision and fundamentally resolves the core limitations of conventional self-distillation in complex reasoning scenarios.

5 Discussion

5.1 How Does EMA Help CloakTeach?

Under the teacher policy, the model has access to privileged reference information, but may still generate incorrect rollouts due to unstable reasoning or policy mismatch during iterative self-distillation. Cloak significantly improves rollout accuracy compared with the base teacher by encouraging reference-consistent and student-compatible trajectories. However, during early training stages, the rollout quality still fluctuates noticeably.

Figure 5 shows that EMA fusion further stabilizes and improves teacher-policy accuracy throughout training. By gradually merging the updated student parameters into the teacher, EMA helps the teacher internalize useful behaviors learned during self-distillation, leading to more reliable reference-conditioned rollouts over time. This suggests that EMA reduces the policy gap between teacher and student, enabling more stable iterative self-improvement.

5.2 What Is the OOD Performance of CloakTeach?

To evaluate out-of-distribution (OOD) generalization, we train ϕ_{CLOAK} using 0.5k, 1k, 2k, and 4k training examples sampled from mathematics, science, and tool-use datasets, respectively, and evaluate the resulting CloakTeach models on a medical reasoning benchmark.

Table 3 shows that both reward metrics and downstream accuracies improve steadily as the

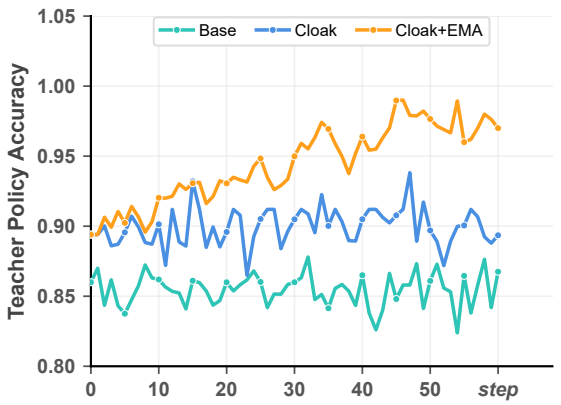

Figure 5: Rollout accuracy under the teacher policy during iterative self-distillation.

Data Size	R_{pollute}	R_{follow}	R_{adapt}	Acc@1	Acc@4
0.5k	0.72	0.84	0.81	72.8	74.6
1k	0.80	0.86	0.85	74.2	76.9
2k	0.94	**0.88**	**0.87**	**78.8**	79.3
4k	**0.96**	0.86	0.85	77.9	**80.1**

Table 3: OOD evaluation on the medical dataset using Cloak trained with different training data scales.

training data scale increases from 0.5k to 2k examples. In particular, R_{follow} and R_{adapt} become more stable, indicating that Cloak learns transferable teacher-side behaviors that generalize to unseen domains. The medical reasoning accuracy also improves consistently, demonstrating strong OOD generalization.

However, the gains from 2k to 4k examples become relatively small, suggesting that performance begins to saturate after sufficient coverage of the source tasks. Overall, these results indicate that Cloak achieves good OOD performance with relatively limited training data, while further improvements may require more diverse training domains and trajectory distributions.

5.3 Is CloakTeach More Helpful for Continual Learning?

We further explore the cross-domain continual learning performance of CloakTeach on reasoning tasks. We adopt two training settings for evaluation: *sequential training*, which first optimizes the model on the target domain before continual fine-tuning on the remaining domains, and *joint training* that trains the model on all domains simultaneously. The performance gap between the two settings is used to measure catastrophic forgetting, where a smaller gap indicates better continual learning stability.

As shown in Table 4, CloakTeach achieves the highest accuracy under both sequential and joint training across all domains, with consistently smaller performance gaps than most baselines. Compared with GRPO, both methods rely on model-generated supervision and exhibit comparable anti-forgetting ability. Nevertheless, CloakTeach effectively incorporates ground-truth constraints into the distillation process, thus achieving much better overall accuracy.

Conventional self-distillation performs competitively in the non-thinking setting with minor forgetting. However, it suffers severe performance degradation and substantial forgetting once the thinking mode is activated. Qwen-30B-Distill, belonging to the Qwen series, has mild forgetting but is limited by teacher model capability. In contrast, CloakTeach maintains stable and minimal performance drops in the challenging thinking scenario, yielding the best comprehensive performance.

5.4 Ablation Study

We perform ablation experiments to analyze the contribution of the three reward components for Cloak training. We evaluate all model variants in the thinking mode and report the Acc@1 results across different reasoning domains.

Table 5 compares the CloakTeach framework with its ablated versions Cloak by excluding each reward component separately. Removing R_{pollute} causes the most dramatic performance drop, which demonstrates that suppressing privileged information leakage is the most essential factor for reliable self-distillation. Eliminating R_{follow} impairs the teacher model's ability to faithfully follow reference content, while discarding R_{adapt} enlarges the distribution mismatch between teacher and student, and reduces trajectory transferability.

As observed from the results, the combination of all three reward terms yields the highest Acc@1 scores on Math, Science and Tool-use tasks, verifying the necessity of each component in our joint optimization objective.

6 Related Work

Self-evolution and self-distillation let language models learn from their own outputs, reducing reliance on heavy human annotation or external strong teachers. Early works boost reasoning ability by filtering and refining self-generated trajec-

Setting	Method	Math			Science			Tool-use		
		Sequential	Joint	Gap	Sequential	Joint	Gap	Sequential	Joint	Gap
Non-Thinking	SFT	45.0	61.2	-16.2	47.1	58.3	-11.2	41.6	68.1	-26.5
	GRPO	59.8	64.5	**-4.7**	64.8	68.8	-4.0	57.1	63.3	**-6.2**
	Qwen-30B-Distill	65.4	72.2	-6.8	47.5	52.7	-5.2	36.9	44.6	-7.7
	Self-Distillation	66.0	71.1	-5.1	66.6	70.5	-3.9	58.2	65.4	-7.2
	CloakTeach	**68.2**	**73.1**	**-4.9**	**68.4**	**72.2**	**-3.8**	**62.7**	**69.2**	**-6.5**
Thinking	GRPO	76.8	80.8	**-4.0**	71.1	74.6	**-3.5**	65.5	70.4	-4.9
	Qwen-30B-Distill	74.9	81.4	-6.5	68.0	72.0	-4.0	67.7	74.3	-6.6
	Self-Distillation	66.1	73.2	-7.1	54.5	62.9	-8.4	55.2	66.9	-11.7
	CloakTeach	**80.6**	**85.4**	**-4.8**	**75.5**	**79.2**	**-3.7**	**77.8**	**81.2**	**-3.4**

Table 4: Continual learning evaluation across reasoning domains. *Sequential* denotes first training on the target domain and then continually training on the remaining two domains, while *Joint* denotes training on all three domains simultaneously.

Method	Math	Science	Tool-use
w/o R_{pollute}	81.7	70.3	77.0
w/o R_{follow}	82.4	76.1	79.2
w/o R_{adapt}	84.1	78.5	80.1
CloakTeach	**85.4**	**79.2**	**81.2**

Table 5: Ablation study on the three reward components used to train Cloak.

tories. For instance, STaR retains correct reasoning paths and rationalizes failed cases for iterative improvement (Zelikman et al., 2022), while self-consistency and chain-of-thought self-training produce high-confidence rationales for model fine-tuning (Huang et al., 2023; Wang et al., 2022). This paradigm has been extended to instruction evolution, self-play, reward filtering and self-rewarded preference optimization (Wang et al., 2023; Xu et al., 2023; Gulcehre et al., 2023; Chen et al., 2024; Yuan et al., 2024). A recent survey frames these pipelines as a closed loop of experience collection, refinement and model update (Tao et al., 2024). As a representative variant, SDFT builds model-aligned distillation data to alleviate catastrophic forgetting in self-distillation (Yang et al., 2024). Despite promising results, existing self-evolution methods mainly focus on generating and filtering high-quality self-supervised data, yet largely ignore inherent generation flaws of teachers in reference-conditioned self-distillation.

Reasoning distillation improves student performance by transferring intermediate rationales from teacher models. Classic methods including chain-of-thought prompting and rationale distillation adopt multi-step reasoning paths as auxiliary supervision to enhance generalization (Wei et al., 2022; Hsieh et al., 2023). However, studies show model-generated rationales are often unfaithful, biased post-hoc explanations instead of genuine decision traces (Turpin et al., 2023; Lanham et al., 2023). These problems grow more severe under reference-guided self-distillation. A reference-aware teacher may leak privileged information, violate reference constraints, or produce reasoning trajectories that reference-free students cannot reproduce.

Unlike prior work that treats reasoning trajectories as directly transferable supervision, CloakTeach explicitly addresses three key teacher-side defects: privileged-information leakage, insufficient reference following, and teacher-student trajectory mismatch. We resolve these issues via teacher adaptation rather than data filtering, offering a new solution for stable and effective iterative self-distillation.

7 Conclusion

We present CloakTeach, a teacher-side adaptation framework for iterative self-distillation that mitigates privileged-information leakage, faulty reference following, and teacher–student reasoning mismatch. By training a lightweight Cloak adapter with three coordinated RL objectives—pollution suppression, reference adherence, and student compatibility—it produces more student-learnable reasoning trajectories without any external teacher. Extensive experiments across mathematical, scientific, tool-use, and medical reasoning tasks, under both thinking and non-thinking settings, demonstrate that CloakTeach consistently outperforms SFT, GRPO, external distillation, and conventional self-distillation, achieving the best overall performance across all evaluation dimensions. We release our experimental materials at `anonymous.4open.science/r/CloakTeach-7224`.

Limitations

This work still has several minor limitations. First, we focus on mathematical, scientific and tool-use reasoning in current evaluations, and do not fully test the framework on long-context reasoning or conversational scenarios. Second, all experiments are implemented based on the Qwen3 series, and cross-model compatibility across diverse model families can be further investigated in future work. Third, the Cloak adapter is trained on a limited set of data, and a more diverse training corpus is required to enhance its general applicability across broader scenarios.

References

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2026. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Kehua Feng, Keyan Ding, Weijie Wang, Xiang Zhuang, Zeyuan Wang, Ming Qin, Yu Zhao, Jianhua Yao, Qiang Zhang, and Huajun Chen. 2024. Sciknoweval: Evaluating multi-level scientific knowledge of large language models. *Preprint*, arXiv:2406.09098.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, and 1 others. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pages 57734–57811.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, and 1 others. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 1051–1068.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, and 1 others. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, and 1 others. 2024. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2025. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*.

Qwen Team. 2025a. Qwen3-30b-a3b. Accessed: 2026-05-25.

Qwen Team. 2025b. Qwen3-8b. Accessed: 2026-05-25.

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

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.

Vladimir Vapnik and Rauf Izmailov. 2015. Learning using privileged information: similarity control and knowledge transfer. *The Journal of Machine Learning Research*, 16(1):2023–2049.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. 2024. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1028–1043.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Rongzhi Zhang, Jiaming Shen, Tianqi Liu, Haorui Wang, Zhen Qin, Feng Han, Jialu Liu, Simon Baumgartner, Michael Bendersky, and Chao Zhang. 2024a. Plad: Preference-based large language model distillation with pseudo-preference pairs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15623–15636.

Songming Zhang, Xue Zhang, Zengkui Sun, Yufeng Chen, and Jinan Xu. 2024b. Dual-space knowledge distillation for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18164–18181.

A More Details on Cloak Training

A.1 Engineering Pipeline

We implement the Cloak training framework on top of the `verl` ecosystem, deployed on a single multi-GPU node. Instead of instantiating independent Teacher, Student, and Reference models, we physically maintain only a single `Qwen3-8B` checkpoint coupled with a trainable Low-Rank Adaptation (LoRA) adapter, and instantiate one unified actor–rollout–reference worker per GPU. Each worker co-hosts a Fully Sharded Data Parallel (FSDP) training engine and a vLLM inference engine sharded by tensor parallelism. By configuring the framework so that the reference policy is served from the actor itself and by toggling the LoRA adapter on or off at runtime, the same parameter store is dynamically multiplexed into the Teacher (LoRA on), Student (LoRA off, fed with the student prompt), and Reference (LoRA off, fed with the hint prompt) roles, eliminating redundant model replicas while preserving GRPO semantics.

Per-Step Lifecycle. Each Group Relative Policy Optimization (GRPO) step is decomposed into a strictly sequential pipeline driven by a central CPU-bound Driver process. *Phase 0 (Generation Setup)* extracts a batch of prompts and duplicates each prompt n times to form GRPO groups sharing a unique group identifier. *Phase 1 (Autoregressive Rollout)* feeds the full hint prompts x_{hint} to the active vLLM engine with LoRA enabled, generating responses y up to a fixed maximum length. *Phase 2 (Memory Reclamation)* puts the vLLM

replicas to sleep, fully releasing the KV cache and vLLM weight tensors so that FSDP can claim the entire device memory. *Phase 3 (Contamination Reward, R_{clean})* combines deterministic CPU rule scoring with an independent LLM judgement to detect leaked hint material; details are given in Section A.2. *Phase 4 (Follow-and-Match Reward, R_{follow})* fuses an answer-overlap score with the LLM's graded judgement of whether y follows the reference information, again using a single shared LLM call as described below. *Phase 5 (Anchor Probability Computation)* runs the FSDP engine in no-grad mode with LoRA active over $[x_{\text{hint}}; y]$ to record the proximal anchor $\log \pi_{\text{old}}(y|x_{\text{hint}})$.

Dual-Forward JS Divergence (R_{adapt}). Phases 6–9 form the core alignment block, encapsulated in a dedicated dual-forward kernel that performs two back-to-back forward passes inside the same FSDP worker context to avoid cross-process communication overhead. *Phase 6 (Student Forward)* disables the adapter and processes the student-prompt concatenation $[x_{\text{stu}}; y]$, treating the engine as the frozen base Student. We compute the per-token predictive entropy

$$H(t) = -\sum_{w \in \mathcal{V}} P_{\text{stu}}(w|y_{<t}, x_{\text{stu}}) \cdot \log P_{\text{stu}}(w|y_{<t}, x_{\text{stu}})$$

and isolate the high-entropy positions $H(t) \geq \tau$ into a sparse Boolean selection mask; for those positions a top-k log-softmax distribution is extracted, where k is set to a fixed fraction of the vocabulary size $|\mathcal{V}|$. *Phase 7 (Cache Flush)* then releases all Student activations and intermediate logits and flushes the GPU caching allocator. *Phase 8 (Teacher Forward)* re-enables LoRA and processes $[x_{\text{hint}}; y]$, reusing the identical selection mask to extract Teacher top-k log-softmax distributions over the same token positions. *Phase 9 (On-GPU Aggregation)* aggregates the Jensen–Shannon divergence on-GPU over the union of the two top-k supports, normalizes by $\ln 2$, and reduces it to a sequence-level scalar

$$R_{\text{adapt}} = 1 - \frac{1}{|T_{\text{sel}}|} \sum_{t \in T_{\text{sel}}} D_{\text{JS}}\left(P_{\text{stu}}^{(t)} \,\|\, P_{\text{tea}}^{(t)}\right),$$

after which all large probability matrices are released and only the scalar R_{adapt} is returned to the Driver.

Reward Fusion and Optimization. *Phase 10 (Reward Fusion)* fuses the three sparse evaluations into a single scalar assigned exclusively to the last valid token of each response, $R_{\text{final}} = \frac{1}{3}(R_{\text{clean}} + R_{\text{follow}} + R_{\text{adapt}})$. *Phase 11 (Reference Baseline Mapping)*, executed when the KL regularizer is enabled, runs an additional no-grad FSDP pass with LoRA off over $[x_{\text{hint}}; y]$ to recover $\log \pi_{\text{ref}}(y|x_{\text{hint}})$ for the standard GRPO KL constraint; this is functionally independent of the Student pass in Phase 6, and the same frozen LoRA-off configuration is reused by the LLM judge calls in Phases 3 and 4. *Phase 12 (GRPO Advantage Calculation)* groups samples by their parent identifier and computes value-network-free advantages $A_i = (R_{\text{final},i} - \frac{1}{n}\sum_{j=1}^{n} R_{\text{final},j})/\sigma_{\text{group}}$. *Phase 13 (Policy Gradient Optimization)* switches FSDP into training mode with gradient checkpointing enabled and uses token-based dynamic micro-batching to backpropagate gradients exclusively into the LoRA parameters while the base weights remain frozen. Finally, *Phase 14 (Weight Synchronisation)* packages the updated LoRA weights and broadcasts them through fast Inter-Process Communication (IPC) to the vLLM partitions, which are then reawakened for step $N+1$.

Validation Cycle. Validation runs are scheduled at a fixed cadence and reuse the same worker layout and role multiplexing as training, but enforce two boundaries: vLLM rollout sampling falls back to deterministic greedy decoding, and the advantage-calculation and optimization phases are skipped entirely. All no-grad FSDP passes (anchor, Student JS, Teacher JS, reference baseline) and the two LLM-judge calls are still executed so that R_{final}, its three components, and entropy statistics are logged on a frozen policy without disturbing the parameters under test.

A.2 Reward Design

The fused training signal is the unweighted average of three component rewards, $R_{\text{final}} = \frac{1}{3}(R_{\text{clean}} + R_{\text{follow}} + R_{\text{adapt}})$, computed independently for every rollout y and assigned to the last valid token of that rollout. A key engineering choice is that the two LLM-based components (R_{clean} and R_{follow}) are produced by a single integrated judge prompt evaluated on the GRPO frozen reference model (the same LoRA-off configuration that supplies $\log \pi_{\text{ref}}$ in Phase 11), so that no additional model has to be loaded and the judge call is amortized over both rewards.

Contamination Reward (R_{clean}). The first reward measures how strictly the response avoids reproducing teacher-provided hint material. We com-

Table 6: Per-Step Phase Overview of One Cloak Training Iteration

Phase	Name	Engine	Primary Output
0	Generation Setup	Driver (CPU)	Expanded prompt batch ($B \times n$)
1	Autoregressive Rollout	vLLM (LoRA on)	Response sequences y
2	Memory Reclamation	Driver / vLLM	vLLM weights and KV cache released
3	Contamination Reward	Rules + Judge LLM	R_{clean}
4	Follow-and-Match Reward	Overlap + Judge LLM	R_{follow}
5	Anchor Probability	FSDP (LoRA on)	$\log \pi_{\text{old}}(y \mid x_{\text{hint}})$
6	JS: Student Forward	FSDP (LoRA off)	Entropy $H(t)$, selection mask, top-k Student dist.
7	JS: Cache Flush	FSDP runtime	Student activations released
8	JS: Teacher Forward	FSDP (LoRA on)	Top-k Teacher dist. on selected positions
9	JS: On-GPU Aggregation	FSDP (LoRA on)	R_{adapt}
10	Reward Fusion	Driver (CPU)	R_{final}
11	Reference Baseline	FSDP (LoRA off)	$\log \pi_{\text{ref}}(y \mid x_{\text{hint}})$
12	GRPO Advantage Calculation	Driver (CPU)	Advantages A_i
13	Policy Gradient Optimization	FSDP training	Updated LoRA weights
14	Weight Synchronisation	IPC / vLLM	vLLM replicas refreshed

bine two independent contamination detectors: a deterministic rule-based scanner that searches the response for surface markers and verbatim hint fragments, and an LLM-based detector that, conditioned on the hint and the response, lists out the sentences in y it considers contaminated. The two detectors operate independently, and the union of their reported sentences is taken as the final contamination set. Starting from a clean baseline R_0, we deduct a fixed penalty δ for each unique contaminated sentence in this union, with the score floored at 0:

$$R_{\text{clean}} = \max\big(0,\ R_0 - \delta \cdot \big|\mathcal{C}_{\text{rule}} \cup \mathcal{C}_{\text{LLM}}\big|\big)\ .$$

This formulation rewards responses that solve the question without echoing the hint, while remaining robust to either detector missing a contaminated sentence.

Follow-and-Match Reward (R_{follow}). The second reward measures whether the response actually uses the supplied reference information to arrive at a correct answer. It is the weighted average of two sub-scores. The *result-overlap* sub-score $s_{\text{overlap}} \in [0, 1]$ compares the final answer extracted from y against the ground-truth answer using a token-level overlap metric, producing a continuous score rather than a binary correctness flag. The *follow* sub-score $s_{\text{follow}} \in [0, 1]$ is produced by the LLM judge, which is asked to grade how faithfully the response follows the reference information on a discrete scale and emits a level (e.g., not-followed / partially / mostly / fully) that is mapped to a value in $[0, 1]$. The two sub-scores are then combined as

$$R_{\text{follow}} = \alpha\, s_{\text{overlap}} + (1 - \alpha)\, s_{\text{follow}},$$

where $\alpha \in [0, 1]$ is the overlap weight. This decouples "did the model produce the right answer" from "did the model do so by actually consulting the reference information," both of which are needed for the Cloak objective.

Shared LLM Judge Call. To avoid the cost of two separate LLM calls per rollout, we issue one integrated judge prompt that supplies the question, the reference information, and the response y, and asks the model to produce a structured output containing (i) the list of sentences in y judged to be contaminated by hint material and (ii) the categorical follow level. This single call is dispatched to the GRPO frozen reference model, reusing the LoRA-off FSDP context already required by the KL regularizer, so the marginal compute cost of the LLM judge is bounded by one extra no-grad forward pass per rollout rather than by an additional model deployment. Parsing the structured output yields $\mathcal{C}_{\text{LLM}}$ for R_{clean} and s_{follow} for R_{follow} simultaneously.

Adaptation Reward (R_{adapt}). The third reward, defined in Phases 6–9, captures how much the LoRA-adapted Teacher must diverge from the frozen Student in order to predict the same response y once the hint is removed. By restricting the divergence to high-entropy positions where the Student is genuinely uncertain and projecting onto the union of the two models' top-k supports, R_{adapt} measures the residual information the adapter has to inject and is high when the adapter has internalized the hint behavior rather than relying on its surface form.

A.3 Memory Footprint

The single-checkpoint, role-multiplexed design is what keeps Cloak training tractable on commodity multi-GPU hardware: avoiding a separate Reference model alone saves the memory footprint of one full `Qwen3-8B` replica per GPU, and routing gradients only through LoRA further reduces optimizer state to a small fraction of the dense-finetuning baseline. The dominant remaining pressure comes from the coexistence of FSDP and vLLM on the same devices, which we resolve through a strict time-sharing protocol coordinated by the Driver rather than through static partitioning.

Rollout Phase. During Phase 1 the vLLM engines are granted a high memory budget so that the KV cache and inference weights can sustain long-context generation throughput; concurrently the FSDP optimizer states and gradient buffers are kept dormant so they do not contend for device memory.

Inference and Training Phase. Once sampling completes, the Driver puts the vLLM replicas to sleep, forcing every replica to release its KV cache and parameter tensors back to the allocator. FSDP then assumes effectively full GPU memory and proceeds through the LLM judge call (Phases 3 and 4, sharing one forward pass), the no-grad anchor pass (Phase 5), the dual-forward JS computation (Phases 6–9), the optional reference pass (Phase 11), and finally the gradient-enabled optimization step (Phase 13). Within the JS block we additionally flush the GPU caching allocator between the Student and Teacher forwards (Phase 7) and immediately drop the top-k probability matrices after on-GPU divergence aggregation (Phase 9), so that only the scalar R_{adapt} survives across phase boundaries.

Synchronization Phase. Phase 14 broadcasts the freshly updated LoRA weights via IPC and wakes the vLLM replicas, which re-acquire their original rollout-time memory budget for the next iteration. Because only the LoRA delta is transferred rather than the full backbone, this synchronization stays well below the bandwidth ceiling of the IPC channel and does not require staging through host memory.

A.4 Time Cost

The wall-clock cost of one Cloak step is dominated by four components: vLLM autoregressive rollout (Phase 1), the shared LLM judge forward (Phases 3 and 4), the cluster of FSDP no-grad forwards (Phases 5, 6, 8, and optionally 11), and the gradient-enabled optimization pass (Phase 13). The remaining phases (CPU rule scoring, reward fusion, advantage computation, weight synchronization) contribute negligible overhead in comparison.

Rollout Cost. Phase 1 scales with the longest sampled sequence in the batch and is the largest single contributor; deterministic greedy decoding during validation noticeably shortens this phase relative to stochastic rollouts at training time.

LLM Judge Cost. Folding the contamination check and the follow grading into a single integrated prompt halves the judge cost relative to a naive two-call design and lets the call piggyback on the existing LoRA-off FSDP context already required for the KL baseline, so the marginal cost of R_{clean} and R_{follow} is bounded by one extra no-grad forward pass per rollout.

FSDP Inference Cost. Phases 5, 6, 8, and 11 each constitute one full forward pass over $[x_{\text{hint}}; y]$ or $[x_{\text{stu}}; y]$. Sharing the same FSDP worker across these passes amortizes engine setup, and the JS block (Phases 6–9) in particular benefits from co-locating the Student and Teacher forwards inside a single kernel block so that the selection mask can be reused without serialization.

Optimization Cost. Phase 13 is the only phase that maintains activations for backpropagation; gradient checkpointing combined with dynamic token-based micro-batching keeps its cost bounded even for long responses, and routing updates only through the LoRA parameters keeps the optimizer step cheap.

A.5 Hyperparameters

The training run uses the GRPO objective on top of `Qwen3-8B` with a LoRA adapter as the only trainable module. Rollouts are produced by vLLM with tensor parallelism, and each Driver step expands every prompt into a GRPO group sharing a common identifier. Optimization uses dynamic token-based micro-batching with gradient checkpointing, and a KL regularizer against the LoRA-off reference policy is applied throughout. Validation runs use greedy decoding.

Reward Settings (qualitative). The fused reward is the unweighted average of the three component rewards, $R_{\text{final}} = \frac{1}{3}(R_{\text{clean}} + R_{\text{follow}} + R_{\text{adapt}})$. The contamination reward starts from a clean baseline and deducts a fixed penalty per unique contaminated sentence reported by either the rule-based

scanner or the LLM judge, with the score floored at 0. The follow-and-match reward is the weighted average $R_{\text{follow}} = \alpha\, s_{\text{overlap}} + (1-\alpha)\, s_{\text{follow}}$, where s_{overlap} is the answer-level overlap score against ground truth and $s_{\text{follow}} \in [0, 1]$ is obtained by mapping the LLM-judged categorical follow level into a numeric score.

Adaptation Reward Settings (qualitative). The JS-based adaptation reward uses an entropy threshold τ to define the high-entropy token set T_{sel}, and a top-k truncation where k is set as a fixed fraction of the vocabulary size $|\mathcal{V}|$; the resulting Jensen–Shannon divergence is normalized by $\ln 2$ before being inverted into a reward.

Concrete Values. Table 7 lists the concrete hyperparameter values used in our reference configuration.

A.6 Prompts

Each raw training sample is rendered into two prompt views that share an identical dialogue scaffold but differ only in whether the teacher demonstration is exposed; pairing these views with the binary LoRA state is what makes the Teacher / Student / Reference roles distinguishable on a single physical model. A third prompt — the LLM judge prompt — is constructed from the same raw fields and consumed by Phases 3 and 4.

Hint Prompt (x_{hint}). Contains the system context, the target question, and a detailed teacher-provided demonstration snippet (the hint) appended to the end of the user message, followed by an instruction asking the model to reason as if it had never seen that demonstration. This view is consumed by the Teacher rollout (Phase 1), the anchor computation (Phase 5), the Teacher forward inside the JS block (Phase 8), and the optional reference baseline (Phase 11). Its concrete template (instantiated for the math domain) is shown in Box A.6.

Hint Prompt x_{hint} (Teacher View)

[System]

Solve the following mathematics problem. Show your reasoning step by step. The last line of your response should be of the following format: `ANSWER: \boxed{[result]}` where `[result]` is your final answer inside a LaTeX `\boxed{}`. Do not add any text after that line.

[User]

`{question}`

This is an example for a correct response to the question:
`{reference_solution}`

`ANSWER: \boxed{{ground_truth_answer}}`

Take inspiration from the example response to answer with a response of your own, while reasoning as if you had not seen the example response or THIS instruction before.

Student Prompt (x_{stu}). Reconstructs the exact same dialogue scaffold as x_{hint} but strips out the teacher-provided demonstration snippet and the accompanying instruction, isolating the raw question context. This view is consumed only by the Student forward in Phase 6, where the model with LoRA disabled must predict y without access to the hint, so that the JS divergence between Student and Teacher distributions on high-entropy tokens directly measures how much information the hint carries. The template is shown in Box A.6.

Student Prompt x_{stu} (Student View)

[System]

Solve the following mathematics problem. Show your reasoning step by step. The last line of your response should be of the following format: `ANSWER: \boxed{[result]}` where `[result]` is your final answer inside a LaTeX `\boxed{}`. Do not add any text after that line.

[User]

`{question}`

Integrated Judge Prompt. The LLM judge in Phases 3 and 4 receives a single integrated prompt that supplies the question, the reference information (hint), and the rollout response y, and asks the frozen reference model to emit a structured output containing both the list of sentences in y it considers contaminated and a categorical follow level describing how faithfully the response uses the reference information. Parsing this output yields $\mathcal{C}_{\text{LLM}}$ and s_{follow} in one call. The full template, aligned with the reward design in Section A.2, is shown in Box A.6.

Table 7: Cloak Training Hyperparameters.

Group	**Hyperparameter**	**Value**
Hardware / Parallelism	vLLM tensor parallel size	2
	vLLM GPU memory utilization	0.85
	FSDP parameter offload	disabled
	FSDP optimizer offload	disabled
Backbone / Adapter	Base model	`Qwen3-8B`
	Trainable module	LoRA adapter only (base weights frozen)
	LoRA rank r	8
	LoRA α	16
	Padding-aware kernels	remove-padding on, fused kernels off
	Gradient checkpointing	enabled
Sequence Length	Max prompt length	8192 tokens
	Max response length	8192 tokens
Optimizer (Actor)	Optimizer	AdamW
	Peak learning rate	1×10^{-5}
	LR schedule	cosine, warmup ratio 0.05, min-LR ratio 0.1
	Dynamic micro-batching token budget	32,768 tokens / GPU
GRPO / KL	Advantage estimator	GRPO (no value network)
	KL loss coefficient	0.001
	KL loss type	low-variance KL
	Entropy bonus coefficient	0
Rollout (Training)	Engine	vLLM, weight load via safetensors
	Sampling	stochastic (training)
	Layered weight summon	enabled
Validation	Frequency	every 4 training steps
	Decoding	greedy
	Optimization phases	skipped (Phases 12–14)
Reward	Fusion	$R_{\text{final}} = \frac{1}{3}(R_{\text{clean}} + R_{\text{follow}} + R_{\text{adapt}})$
	R_{clean} baseline R_0 / penalty δ / floor	1.00 / 0.10 / 0
	R_{follow} overlap weight α	to be tuned
	Follow-level mapping	{not_followed, partially, mostly, fully} $\rightarrow [0, 1]$
	R_{adapt} entropy threshold τ	1.0 nats
	R_{adapt} top-k vocabulary fraction	$0.10\,\lvert\mathcal{V}\rvert$
Training Schedule	Total epochs	3
	Checkpoint frequency	every 50 steps

Integrated Judge Prompt (LLM Contamination + Follow Grader)

Role. You are a strict reward model that simultaneously (i) detects contamination in a generated chain of thought (CoT) and (ii) grades how faithfully the CoT follows the supplied reference information. Both signals are produced from a single pass.

Inputs.

- `[Question]`: the original user question.
- `[Reference Information]`: the hint / reference solution that was provided to the policy alongside the question.
- `[Generated CoT]`: the response y produced by the policy.

Task 1 — Contamination Detection. Identify every sentence of `[Generated CoT]` that exhibits awareness of hidden evaluation information or external guidance, and return the verbatim sentences as a list. The CoT must behave as if it only saw the question. Treat as contamination any explicit or implicit reference to: example responses, provided reasoning, hidden hints, prompt instructions, evaluation or scoring setup, demonstrations, answer suggestions, reference logic, meta-instructions such as "pretend I did not see the example" or "I should not leak information," as well as paraphrases such as "based on the example," "following the demonstrated reasoning," "the prompt suggests," "among the provided choices," or "using a similar approach." If no sentence is contaminated, return an empty list.

Task 2 — Follow Grading. Independently of Task 1, decide how faithfully the CoT follows the core reasoning principles in `[Reference Information]` when arriving at its final answer, and emit one of the following categorical levels:

- `not_followed` — the CoT ignores or contradicts the reference information.
- `partially` — the CoT picks up some elements of the reference information but misses or misapplies the key deductions.
- `mostly` — the CoT applies the core deductions of the reference information, with only minor gaps.
- `fully` — the CoT independently reproduces the core reasoning of the reference information and reaches a consistent final answer.

The grading must focus on logical alignment with the reference information, not on surface wording.

Output Format. Strictly output the following JSON object and nothing else (no markdown, no commentary):

```
{
  "analysis": "Brief justification
  covering both contamination and
  follow.",
  "contaminated_sentences": ["
  <verbatim sentence 1>", "..."],
  "follow_level": "not_followed |
  partially | mostly | fully"
}
```

Inputs to evaluate.
`[Question]`: {question}
`[Reference Information]`: {reference_solution}
`[Generated CoT]`: {response_y}

Role–Prompt Multiplexing. Table 8 summarizes how the prompt views combine with the LoRA state and the execution engine to realize all required semantic roles within one physical checkpoint.

B More Details on Iterative Self-Distillation

B.1 Engineering Pipeline

We implement the iterative self-distillation framework on top of the `verl` ecosystem, deployed on a single multi-GPU node and orchestrated by a CPU-bound Ray Driver. The framework supports two operational modes: a Teacher-rollout supervised fine-tuning (SFT) mode that serves as the primary training path, and an on-policy distillation mode maintained as an alternative baseline. In both modes the system multiplexes several concep-

Table 8: Model Operating Paradigms per Training Step

Semantic Role	Execution Engine	Prompt View	LoRA State	Gradient Mode
Teacher (Rollout)	vLLM Inference	x_{hint}	On	Frozen (Sampling)
Teacher (π_{old} Anchor)	FSDP Evaluation	x_{hint}	On	Frozen
Student (Consistency)	FSDP Evaluation	x_{stu}	Off	Frozen
Reference (KL + Judge)	FSDP Evaluation	x_{hint} / Judge prompt	Off	Frozen
Teacher (Optimization)	FSDP Training	x_{hint}	On	Active

tual entities onto a single node without exceeding hardware memory capacity: a unified actor–rollout–reference worker is instantiated on each GPU, co-hosting an FSDP training engine and a vLLM inference engine sharded by tensor parallelism.

Logical Roles. The Student Actor is the sole trainable parameter store and resides on the FSDP engine in GPU memory. A slow-moving Teacher state dictionary, maintained as an exponential moving average (EMA) of the Student parameters with a blending coefficient α, is held in CPU memory and defines the canonical Teacher behavior. A separate Teacher FSDP shell can be hydrated on demand with the EMA weights to provide Teacher log-probabilities when the distillation mode is active. The vLLM rollout engine is loaded with the Teacher rollout weights, which equal the EMA dictionary plus the frozen Cloak Adapter $\Delta_{\text{cloak}} = \theta_{\text{cloak}} - \theta_{\text{base}}$ obtained from a previously completed Cloak training run and kept as a static CPU tensor. This overlay allows the Teacher to benefit from prior reinforcement-learning capabilities without injecting them into the EMA dictionary or the Student parameters. Table 9 summarizes the full role layout.

Per-Step Lifecycle (SFT Main Path). Under the primary SFT mode, each training step bypasses the multi-pass inference overhead typical of reinforcement-learning frameworks and transitions directly from Teacher generation into token-level supervised optimization via a concise pipeline. *Phase 0 (Initialization)* carries over the synchronized vLLM weights from the end of step $N{-}1$, keeps the Student FSDP parameters resident on device, and leaves the Teacher EMA dictionary in CPU memory. *Phase 1 (Teacher Rollout)* feeds the full hint prompts x_{hint} to the active vLLM engine and samples response sequences y up to a fixed maximum length under the Teacher rollout policy. *Phase 2 (Memory Reclamation)* puts the vLLM replicas to sleep, releasing the KV cache and rollout weight tensors so that FSDP can claim the entire device memory.

Cross-View Reconstruction and Student Optimization. *Phase 3 (Cross-View Reconstruction)* is executed on the CPU: the demonstration snippet is stripped from the hint prompt to produce the student prompt x_{stu}, which is concatenated with the Teacher-generated response y to form the training input $[x_{\text{stu}};\, y]$. A selective loss mask $M(t)$ is created such that $M(t) = 1$ for positions corresponding to y and $M(t) = 0$ for positions corresponding to x_{stu}, ensuring that the Student is trained only on the response tokens. *Phase 4 (Student Optimization)* switches the Student FSDP engine into training mode and executes a forward–backward pass over $[x_{\text{stu}};\, y]$ with the masked cross-entropy objective

$$\mathcal{L}_{\text{SFT}}(\theta_{\text{stu}}) = -\frac{1}{\sum_t M(t)} \sum_t M(t) \cdot \log P_{\text{stu}}(y_t \mid y_{<t},\, x_{\text{stu}})\,.$$

Gradients are routed through the full Student parameters (or through LoRA adapters when that mode is enabled) and an optimizer step is taken.

Teacher EMA Update and Synchronization. *Phase 5 (EMA Blending)*, triggered at fixed step boundaries, updates the Teacher state dictionary in CPU memory by pulling the current Student base parameters:

$$\theta_{\text{EMA}} \leftarrow (1-\alpha)\,\theta_{\text{EMA}} + \alpha\,\theta_{\text{stu_base}}.$$

The frozen Cloak Adapter Δ_{cloak} is never included in this blend and remains static throughout the entire run. *Phase 6 (Weight Synchronisation)* combines the updated EMA dictionary with Δ_{cloak} and broadcasts the result via IPC to the vLLM partitions, which are then re-awakened for step $N{+}1$.

Alternative: On-Policy Distillation Mode. When the pipeline is configured for distillation mode, the system trades execution speed for dense token-level alignment. The key differences from the SFT path are: (i) the vLLM engine loads the *Student's* current parameters and generates responses y under the student prompt x_{stu} rather than the hint prompt;

Table 9: Logical Model Entities and Execution Boundaries in Self-Distillation.

Logical Role	Physical Layer	Parameter Source	Functional Output
Student Actor	FSDP engine	Initial base checkpoint; updated by SFT	Policy under optimization; produces token-level gradients.
Teacher EMA	State dictionary	Iterative EMA blending from Student	Slow-moving reference behavior.
Teacher FSDP shell	FSDP instance	Hydrated on demand from Teacher EMA	Teacher log-probs in distillation mode.
vLLM rollout engine	vLLM instance	Teacher EMA $+ \Delta_{\text{cloak}}$	High-throughput autoregressive sampling under hint.
Cloak Adapter	Residual tensor	Static: $\theta_{\text{cloak}} - \theta_{\text{base}}$	Frozen prior; overlaid onto Teacher rollout only.

Table 10: Per-Step Phase Overview of One Self-Distillation Iteration (SFT Mode).

Phase	Name	Engine	Primary Output
0	Initialization	Driver (CPU)	Carry-over state from step $N-1$
1	Teacher Rollout	vLLM (Teacher)	Response sequences y
2	Memory Reclamation	Driver / vLLM	KV cache and rollout weights released
3	Cross-View Reconstruction	Driver (CPU)	Training batch $[x_{\text{stu}};\ y]$ with loss mask M
4	Student Optimization	FSDP (Student)	Updated Student parameters
5	EMA Blending	CPU	Updated θ_{EMA} (at sync boundaries)
6	Weight Synchronisation	IPC / vLLM	vLLM replicas refreshed

(ii) after memory reclamation, the Student FSDP engine computes proximal anchor log-probabilities $\log \pi_{\text{old}}(y|x_{\text{stu}})$ in a no-grad pass; (iii) the Teacher FSDP shell is hydrated from CPU memory (with the Cloak Adapter overlaid) and runs a single forward pass over $[x_{\text{hint}};\ y]$ to supply dense target distributions; (iv) the Student is then optimized via a reverse-KL objective against these dense targets:

$$\mathcal{L}_{\text{distill}}(\theta_{\text{stu}}) = D_{\text{KL}}(\pi_{\text{stu}}(y|x_{\text{stu}}) \,\|\, \pi_{\text{tea}}(y|x_{\text{hint}}))\,.$$

Validation Cycle. Validation is executed at a fixed cadence and employs a dual-pass strategy to track both Teacher and Student quality. In the first pass, the vLLM engine retains the Teacher rollout weights and generates under x_{hint} with greedy decoding, producing Teacher-side metrics. In the second pass, the Driver temporarily swaps the Student parameters into vLLM, generates under x_{stu} with greedy decoding to capture Student-side generation quality, and then restores the Teacher weights before resuming training.

B.2 Memory Footprint

The self-distillation framework shares the same time-sharing memory orchestration as the Cloak training described in Appendix A, but its per-step memory profile is considerably simpler because no multi-pass FSDP inference block (anchor, JS divergence, KL baseline) is required during the SFT path—only a single forward–backward pass through the Student engine.

Rollout Phase. During Phase 1 the vLLM engine is allocated a moderate memory budget to sustain long-context generation; the FSDP optimizer states are dormant. Because the Teacher rollout weights include the frozen Cloak Adapter, the vLLM weight footprint is slightly larger than a raw EMA copy but remains within the same magnitude.

Training Phase. After Phase 2 reclaims vLLM memory, the FSDP engine assumes full device capacity. The dominant allocation in Phase 4 is the activation memory and gradient buffers for the single Student forward–backward pass. To keep this tractable we enable FSDP parameter offloading and optimizer offloading to CPU, trading a moderate increase in wall-clock time for the ability to accommodate full-parameter updates (or large LoRA modules) on the available hardware.

EMA and Cloak Adapter Storage. The Teacher EMA dictionary and the static Cloak Adapter weights are both stored in CPU host memory and are never simultaneously materialized on GPU. Only during Phase 6 does a fused copy ($\theta_{\text{EMA}} + \Delta_{\text{cloak}}$) transit through host memory on its way to the vLLM partitions via IPC.

B.3 Time Cost

The wall-clock cost of one self-distillation step is dominated by two components: the vLLM

Teacher rollout (Phase 1) and the Student forward–backward pass (Phase 4). Unlike the Cloak training pipeline which requires multiple no-grad FSDP forward passes per step, the SFT path executes only a single gradient-enabled pass, making each step substantially cheaper.

Rollout Cost. Phase 1 remains the largest contributor, scaling with the maximum sampled sequence length. Greedy decoding during validation shortens this cost relative to stochastic rollouts at training time.

Student Optimization Cost. Phase 4 executes one full forward–backward pass with gradient checkpointing and dynamic token-based micro-batching. Parameter and optimizer offloading introduce additional host-device transfers but allow the system to stay within budget without reducing the model size.

EMA and Sync Cost. Phase 5 (EMA blending) and Phase 6 (IPC broadcast) together contribute only marginal overhead because the blending is a simple linear combination executed in CPU memory and the broadcast transfers only the fused parameter dictionary rather than activations or gradients.

B.4 Hyperparameters

The self-distillation run uses `Qwen3-8B` as the base model, with the Student performing full-parameter SFT (no LoRA). Rollouts are produced by vLLM under tensor parallelism, and a fixed group expansion factor n is applied per prompt to form the generation batch. The Teacher EMA is updated at fixed step intervals with a blending coefficient α controlling the contribution of the latest Student base weights. A frozen Cloak Adapter Δ_{cloak} is added to the Teacher rollout weights; this adapter is computed once from a previously completed Cloak training checkpoint and never updated during the distillation run. Validation uses a dual-pass greedy strategy covering both Teacher and Student views.

Concrete Values. Table 11 lists the concrete hyperparameter values.

B.5 Prompts

The self-distillation pipeline uses the same pair of prompt views introduced in Appendix A — a hint prompt and a student prompt — differing only in which phase consumes each view. In the SFT main path the hint prompt is consumed exclusively by the Teacher vLLM rollout (Phase 1), while the student prompt forms the training input for the Student FSDP engine (Phase 4). In distillation mode the roles reverse: the student prompt drives the Student vLLM rollout and the hint prompt drives the Teacher FSDP forward pass. The concrete prompt templates are reproduced below for self-containedness.

Hint Prompt (x_{hint}). Consumed by the Teacher rollout to generate y under demonstration guidance. Its structure is shown in Box B.5.

Hint Prompt x_{hint} (Teacher Rollout View)

[System]

Solve the following mathematics problem. Show your reasoning step by step. The last line of your response should be of the following format: `ANSWER: \boxed{[result]}` where `[result]` is your final answer inside a LaTeX `\boxed{}`. Do not add any text after that line.

[User]

`{question}`

This is an example for a correct response to the question:
`{reference_solution}`

`ANSWER: \boxed{{ground_truth_answer}}`

Take inspiration from the example response to answer with a response of your own, while reasoning as if you had not seen the example response or THIS instruction before.

Student Prompt (x_{stu}). Reconstructs the same dialogue scaffold as x_{hint} but strips the demonstration snippet and the accompanying instruction. In the SFT path it is concatenated with the Teacher-generated response y (Phase 3) to form the training input; in the distillation path it drives the Student rollout. Its structure is shown in Box B.5.

Student Prompt x_{stu} (Student View)

[System]

Solve the following mathematics problem. Show your reasoning step by step. The last line of your response should be of the following format: `ANSWER: \boxed{[result]}` where `[result]` is your final answer inside a LaTeX `\boxed{}`. Do not add any text after that line.

Table 11: Self-Distillation Training Hyperparameters.

Group	Hyperparameter	Value
Hardware / Parallelism	vLLM tensor parallel size	2
	vLLM GPU memory utilization	0.50
	FSDP parameter offload	enabled
	FSDP optimizer offload	enabled
Backbone / Adapter	Base model	`Qwen3-8B`
	Trainable module	Full parameters (LoRA rank $= 0$)
	Gradient checkpointing	enabled
	Remove-padding kernels	enabled
Sequence Length	Max prompt length	8192 tokens
	Max response length	8192 tokens
Optimizer (Student)	Optimizer	AdamW
	Peak learning rate	1×10^{-5}
	LR schedule	cosine, warmup ratio 0.05, min-LR ratio 0.1
	Dynamic micro-batching token budget	32,768 tokens / GPU
Rollout	Engine	vLLM, weight load via safetensors
	Sampling (training)	stochastic
	Layered weight summon	enabled
Teacher EMA	EMA blending coefficient α	0.8
	Sync cadence (steps)	every 10 steps
	Cloak Adapter source	Cloak checkpoint minus base
Validation	Frequency	every 10 training steps
	Decoding	greedy (both Teacher and Student passes)
Training Schedule	KL loss	disabled
	Total epochs	3
	Checkpoint frequency	every 50 steps

[User]
`{question}`

C Supplementary

C.1 More Related Work

Self-Generated Supervision. Self-evolution methods train language models on experience produced by the models themselves. Huang et al. (2023) use chain-of-thought prompting and self-consistency to generate high-confidence, rationale-augmented answers for unlabeled questions, and then fine-tune the same model on the synthetic solutions. Zelikman et al. (2022) introduce STaR, which bootstraps reasoning by sampling rationales, retaining trajectories that reach the correct answer, and using the gold answer to rationalize failed attempts. The same idea has also been extended beyond reasoning traces to instruction construction, instruction evolution, reward-guided filtering, self-play, and self-rewarded preference optimization (Wang et al., 2023; Xu et al., 2023; Gulcehre et al., 2023; Chen et al., 2024; Yuan et al., 2024). A recent survey describes this broader paradigm as a cycle of experience acquisition, refinement, model update, and evaluation (Tao et al., 2024). These works motivate our setting, but they mainly study how self-generated targets are obtained, filtered, or refined, whereas our setting asks how a reference-conditioned self-teacher should produce trajectories for a student that never sees the reference.

Reasoning Distillation. Reasoning distillation trains students on intermediate rationales, but its value depends on whether those rationales are faithful to the teacher's decision process and usable by the student. Chain-of-thought prompting elicits multi-step reasoning (Wei et al., 2022), self-consistency selects answers supported by multiple sampled reasoning paths (Wang et al., 2022), and rationale distillation methods train students with generated explanations as auxiliary supervision (Hsieh et al., 2023). Prior analyses, however, show that chain-of-thought explanations can rationalize answer-driven or biased predictions rather than faithfully explain the factors that determine the model prediction (Turpin et al., 2023; Lanham et al., 2023). This risk is stronger in answer- or reference-conditioned self-distillation, where a teacher may leak privileged information, ignore the provided reference, or produce a reasoning path

that an unconditioned student is unlikely to reproduce. This issue is central to CLOAKTEACH, which treats privileged-information leakage, weak reference following, and low student reachability as teacher-side generation failures rather than assuming that rationales transfer directly.

Privileged-Information Self-Distillation. Self-distillation is most reliable when the teacher signal matches the student's policy and available information. Classical knowledge distillation transfers behavior by matching a teacher distribution with a student distribution (Hinton et al., 2015), while recent LLM self-distillation methods reduce distribution mismatch by keeping supervision close to the seed model. For example, SDFT constructs a distilled fine-tuning set with the seed model itself to better match the model's original distribution and reduce catastrophic forgetting (Yang et al., 2024). Our setting is also related to learning with privileged information, where training-time information is available to the teacher but absent from the student's inference input (Vapnik and Izmailov, 2015). Our work differs by adapting the privileged teacher before student learning, so compatibility is addressed during trajectory generation rather than only through data construction, policy regularization, or post-hoc filtering.

Teacher-Side Adaptation. Feedback-based methods improve model behavior with critiques, tool feedback, rewards, or model-generated preference signals. Self-Refine, Reflexion, and CRITIC use feedback to revise model outputs or guide agent behavior at inference time (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024), while SCoRe and self-rewarding language models use reinforcement or preference optimization to improve self-correction and evaluation abilities from self-generated data (Kumar et al., 2024; Yuan et al., 2024). These methods mainly revise test-time outputs, guide agent trajectories, or train general self-correction and evaluation behavior. In CLOAKTEACH, feedback is used more narrowly to train a teacher-side LoRA adapter (Hu et al., 2022) for reference-conditioned rollout generation, leaving the base model to co-evolve with the student through iterative self-distillation.

C.2 License for Artifacts

Our work uses publicly available third-party artifacts and does not claim ownership over any external dataset or model. The datasets used in our experiments are `EleutherAI/hendrycks_math`, `hicai-zju/SciKnowEval`, `Salesforce/xlam-function-calling-60k`, and `openlifescienceai/medmcqa`. `EleutherAI/hendrycks_math` and `SciKnowEval` are distributed on Hugging Face under the MIT license. `Salesforce/xlam-function-calling-60k` is distributed under the CC-BY-4.0 license and includes repository-level access and citation requirements. `openlifescienceai/medmcqa` is distributed under the Apache-2.0 license.

The model artifacts used in our experiments are `Qwen3-30B-A3B` and `Qwen3-8B`, both released by Qwen on Hugging Face under the Apache-2.0 license. We preserve attribution to the original artifact providers, cite the corresponding papers or repositories where applicable, and comply with the relevant licenses, repository notices, access conditions, and Hugging Face platform terms.

C.3 Artifact Use Consistent With Intended Use

Our use of these artifacts is consistent with their intended research and evaluation purposes. `EleutherAI/hendrycks_math` is used as a mathematical reasoning benchmark, `SciKnowEval` is used for scientific knowledge evaluation, `Salesforce/xlam-function-calling-60k` is used as a function-calling and tool-use resource, and `openlifescienceai/medmcqa` is used as a medical multiple-choice question answering dataset.

The datasets are used only for constructing evaluation, preprocessing, and experimental pipelines. We do not use them to identify individuals, infer sensitive attributes, provide medical advice, support clinical diagnosis or decision-making, or enable harmful downstream applications.

`Qwen3-30B-A3B` and `Qwen3-8B` are used as language models for controlled academic experiments. We do not remove upstream attribution, bypass access restrictions, or repackage third-party content under a conflicting license. Any released code, scripts, or metadata will clearly indicate the upstream artifacts required for reproduction.

C.4 Use of AI Assistants

We used ChatGPT as an auxiliary tool during the preparation of this submission. Its use was limited to minor language polishing, including grammar, clarity, and concision improvements to author-written text, and to experimental data preprocess-

ing, such as formatting records, checking field consistency, and organizing intermediate files.

D Supplementary Material

D.1 Taxonomy of Privileged-Information Leakage

Table 12 classifies the principal modes by which a model's chain of thought (CoT) may inadvertently reveal awareness of the teacher-provided reference information. These categories motivate the design of the contamination reward R_{clean} (Appendix A) and the rule-based detector used alongside the LLM judge.

Table 12: Taxonomy of Privileged-Information Leakage in Generated CoTs.

Leak Type	Manifestation and Impact
Explicit Reference	The CoT directly mentions the existence of a standard answer, gold label, or reference solution, violating the blind-reasoning constraint.
Content Copy	Long spans of text are reproduced verbatim or near-verbatim from the reference answer, creating reasoning dependency and reducing generalization.
Hint Disclosure	Implicit solving clues embedded in the reference answer are surfaced in the CoT (e.g., naming a key intermediate quantity that only appears in the hint), breaking the fair-reasoning setting.
Result Preemption	The final conclusion is stated before the supporting reasoning steps have been developed, revealing that the model accessed the answer prior to deriving it.
Meta-Awareness	The CoT contains self-referential statements about the evaluation setup, such as "I should not reveal the example" or "pretend I did not see the hint," betraying knowledge of the prompting scaffold.
Structural Mimicry	The CoT replicates the organizational structure of the reference (e.g., identical subproblem ordering or variable naming conventions) without contributing independent analytical value.

Detection Strategy. For each rollout y, the rule-based contamination scanner applies a battery of regular-expression patterns (Table 13) to flag sentences exhibiting meta-awareness or explicit referencing. Independently, the LLM judge (described in Appendix A, Integrated Judge Prompt) performs semantic-level detection that captures paraphrased or implicit leakage that surface patterns cannot reach. The union of both detector outputs defines the contamination set $\mathcal{C}_{\text{rule}} \cup \mathcal{C}_{\text{LLM}}$ from which R_{clean} is derived.

D.2 Rule-Based Contamination Patterns

Table 13 lists the complete set of regular expressions deployed in the rule-based scanner. Each pattern targets a distinct family of leakage signals and is evaluated case-insensitively against every sentence in the generated CoT. A match on any pattern causes the enclosing sentence to be added to $\mathcal{C}_{\text{rule}}$.

Design Rationale. Patterns 1–5 target explicit mentions of the reference material itself (example, provided reasoning, hidden hint, provided choices). Patterns 6–10 catch awareness of the evaluation infrastructure (prompt instructions, scoring rules, meta-information). Pattern 11 detects the word "demonstration" which is never expected in a naturally derived CoT. Patterns 12–16 identify phrases that attribute reasoning steps to an external source rather than the model's own derivation. Patterns 17–19 capture self-censorship statements that reveal the model knows it is being evaluated for leakage. This layered design ensures broad coverage across both direct and indirect contamination signals while maintaining high precision; false positives are rare because these phrases virtually never appear in genuinely independent reasoning.

D.3 Leakage Case Studies

The following examples illustrate real contamination instances observed during Cloak training rollouts. In each case the contaminated sentence is highlighted in bold. These cases span both the mathematics domain and the tool-calling domain to demonstrate that leakage is not task-specific.

Case 1: Mathematics — Explicit Structure Mimicry

System: Solve the following mathematics problem. Show your reasoning step by step. The last line of your response should be of the following format: ANSWER: \boxed{[result]} where [result] is your final answer inside a LaTeX \boxed{}. Do not add any text after that line.

User: How many distinct, non-equilateral triangles with a perimeter of 60 units have integer side lengths a, b, and c such that a, b, c is an arithmetic sequence?

[Reference solution and instruction omitted for brevity.]

Table 13: Regex patterns used to detect contamination-related meta-awareness and reference leakage in generated CoTs.

#	Regex Pattern
1	`(?i)\bexample\s+(response|answer|reason(ing)?)\b`
2	`(?i)\bprovided\s+(reason(ing)?|answer|example|response)\b`
3	`(?i)\breference\s+(answer|reason(ing)?|example|response)\b`
4	`(?i)\bhidden\s+(example|hint|answer|reasoning)\b`
5	`(?i)\bprovided\s+(choice|option)s?\b`
6	`(?i)\b(prompt|task|query)\s+instruction(s)?\b`
7	`(?i)\b(evaluation|eval|benchmark)\s+(setup|process|scheme)\b`
8	`(?i)\b(scoring|grading)\s+(rule|criteria)s?\b`
9	`(?i)\b(external|additional)\s+(guidance|instruction|context)\b`
10	`(?i)\bmeta[\s-]?(information|reasoning|context|instruction)\b`
11	`(?i)\bdemonstration(s)?\b`
12	`(?i)\b(based|derived|inferred)\s+on\s+the\s+(example|provided\s+reasoning|reference)\b`
13	`(?i)\b(following|using)\s+the\s+(provided\s+reasoning|example|reference\s+answer)\b`
14	`(?i)\baccording\s+to\s+the\s+(prompt|instruction|example|provided\s+reasoning)\b`
15	`(?i)\bamong\s+the\s+(provided|given)\s+(choice|option|answer)s?\b`
16	`(?i)\b(using|from|via)\s+the\s+(example|reference\s+answer|provided\s+reasoning)\b`
17	`(?i)\bi\s+should\s+not\s+(leak|reveal|expose)\s+(information|reasoning|answer)s?\b`
18	`(?i)\bpretend\s+i\s+did\s+not\s+(see|read|observe)\b`
19	`(?i)\b(reason|reasoning)\s+as\s+if\s+i\s+had\s+not\s+(seen|read)\b`

Assistant: `<think>` Okay, so I need to solve this problem where I have to find the number of distinct, non-equilateral triangles with integer side lengths a, b, c that form an arithmetic sequence and have a perimeter of 60 units. Let me think through this step by step.
First, **the example response used the common difference d and set up the sides as $b - d$, b, and $b + d$.** That makes sense because in an arithmetic sequence, each term increases by a common difference. . . .

Contamination detected: The bolded sentence explicitly references the example response, triggering Pattern 1 (`example\s+response`).

Case 2: Tool Calling — Failure to Follow Reference Information

System: Use the provided tools when needed to fulfill the user's request. The last line of your response should be of the following format: ANSWER: [json_string] where json_string is a single-line JSON array of tool calls. Do not add any text after that line.

User: Obtain the Parabolic SAR data for Bitcoin (BTC/USD) with hourly intervals, a maximum of 1000 data points, and a 'csv' format. Also, retrieve earnings information for Tesla Inc. (TSLA). Get the top 10 SPAC rankings based on 'losers' for 'monthly' 'warrant' type. Lastly, fetch the yield information for TSLA.

[Reference tool-call array and instruction omitted for brevity.]

Assistant: `<think>` Alright, I'm looking at this request. The user wants several things: Parabolic SAR data for Bitcoin with specific parameters, earnings for Tesla, SPAC rankings, and yield for TSLA. I need to organize these into tool calls.
For the SAR data, the symbol is BTC/USD with a 1h interval, and the request specifies a maximum of 1000 data points. **I'll set outputsize to 1000 and keep the format as csv.**
I'll structure these as a JSON array of tool calls, making sure each tool name matches what's expected and arguments are cleanly formatted. . . .

Failure to follow: The reference answer correctly maps "maximum of 1000 data points" to the dedicated `maximum` parameter of the SAR tool. The model instead assigns this value to `outputsize` — a semantically different field — producing an incorrect tool call despite having access to the reference. This response would receive

a low s_{follow} score from the LLM judge (category: `not_followed`).

Observations. Case 1 illustrates a contamination failure: the model explicitly cites the reference answer in its reasoning, triggering the rule-based scanner. Case 2 illustrates a follow failure: the model ignores the correct parameterization provided in the reference and independently arrives at a wrong tool call. Together the two cases demonstrate why both R_{clean} and R_{follow} are needed as separate reward signals — a response may be contamination-free yet still fail to leverage the reference information, or conversely may leak the reference yet not actually follow it correctly.