Uncertainty-Calibrated On-Policy Distillation for Large Language Models

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

On-policy distillation trains a student on its own outputs with teacher feedback to avoid the train-inference distribution mismatch present in standard distillation. However, a fundamental underexplored challenge is that the teacher itself is imperfect; consequently, naively imitating erroneous teacher outputs propagates mistakes into the student. Prior work such as Self-Distilled Reasoner [1] mitigates this by injecting ground-truth chain-of-thought solutions into the teacher's system prompt to guarantee the teacher with correct reasoning. But this approach depends on labeled data that is unavailable in many practical settings. We introduce **UOPD** (**U**ncertainty-calibrated **O**n-**P**olicy **D**istillation), a framework that addresses teacher errors without ground-truth supervision. Teacher reliability is estimated offline by computing its **uncertainty**, and the distillation objective is reweighted inversely proportional to the teacher's uncertainty with no additional training cost. We further release **SURE-Math**, a dataset annotated with teacher semantic entropy scores. We evaluate UOPD across mathematical reasoning, summarization, translation, and instruction tuning tasks. UOPD converges faster and achieves state-of-the-art results compared to both GRPO and standard on-policy distillation.

1 introduction

2 Preliminary

2.1 On-Policy Knowledge Distillation

Standard knowledge distillation [2] trains a student model $p_S(\cdot;\theta_S)$ to minimize the KL divergence from a frozen teacher p_T on a fixed dataset $\mathcal{D}$:

$$\mathcal{L}_{\text{KD}}(\theta_S) = \mathbb{E}_{x\sim\mathcal{D}}\left[\text{KL}(p_T(\cdot \mid x) \parallel p_S(\cdot \mid x;\theta_S))\right]. \tag{1}$$

This offline objective suffers from a train-inference distribution mismatch: during training the student observes token-level distributions conditioned on dataset prefixes, whereas at inference it must generate from its own previously predicted tokens. On-policy distillation [3] resolves this by generating training sequences from the student itself:

$$\mathcal{L}_{\text{OPD}}(\theta_S) = \mathbb{E}_{x\sim\mathcal{D}}\,\mathbb{E}_{y\sim p_S(\cdot|x;\theta_S)}\left[\sum_{t=1}^{|y|}\text{KL}(p_T(\cdot \mid x, y_{<t}) \parallel p_S(\cdot \mid x, y_{<t};\theta_S))\right]. \tag{2}$$

Since the training distribution now matches the inference distribution, on-policy distillation substantially reduces compounding errors in chain-of-thought reasoning [4].

2.2 Uncertainty in LLMs

Uncertainty in language model outputs can be decomposed into two fundamentally distinct sources [5]. **Aleatoric uncertainty** is irreducible and arises from the inherent ambiguity of natural language, where the same meaning can be expressed in infinitely many surface forms. Consider the prompt *"What is the capital of the UK?"* Given this question, the responses *"London"*, *"London is the capital of the UK"*, and *"The UK's capital is London"* are semantically identical, while their token-level divergence is large. A teacher that consistently produces such paraphrases is not making errors; it is exhibiting natural lexical variation. Penalising this variation at the token level conflates surface diversity with genuine unreliability.

Epistemic uncertainty, by contrast, reflects the teacher's *lack of knowledge*. It arises when the teacher generates contradictory answers across samples, indicating that it does not reliably know the correct response to a given prompt. Standard token-level objectives cannot distinguish these two sources, because both manifest as high distributional variance in the output space.

Semantic entropy [6] resolves this ambiguity by operating over *meaning* rather than surface tokens. Responses are first clustered into semantic equivalence classes $\mathcal{C}$, and entropy is computed over the resulting distribution

$$H_{\text{sem}}(x) = -\sum_{c \in \mathcal{C}} p(c \mid x) \log p(c \mid x), \quad p(c \mid x) \propto \sum_{y \in c} p_T(y \mid x). \tag{3}$$

Paraphrases of the same answer collapse into a single class, suppressing aleatoric noise. $H_{\text{sem}}(x)$ is therefore a faithful, label-free measure of epistemic uncertainty; it is high only when the teacher genuinely disagrees with itself across semantically distinct answers.

3 Method

We propose UOPD, a two-phase framework for uncertainty-calibrated on-policy distillation. In the first phase, we estimate the teacher's epistemic uncertainty for each training prompt by computing semantic entropy over multiple teacher rollouts. This computation is performed entirely offline before training begins, producing a per-prompt uncertainty weight that is stored alongside the training data. In the second phase, the student is trained on-policy with token-level distillation from the teacher, where each sample's contribution to the loss is calibrated by its precomputed uncertainty weight.

The key advantage of this design is the separation of uncertainty estimation from training. Because semantic entropy is computed once before the training loop, UOPD introduces zero additional overhead during training while still enabling uncertainty-aware distillation. We describe each component in detail below.

3.1 Offline Semantic Entropy Estimation

Given a training set of prompts $\{x_i\}_{i=1}^{M}$, we first estimate the teacher's epistemic uncertainty per prompt. For each prompt x_i, we sample N responses from the teacher $\{y_1^T, \ldots, y_N^T\} \sim p_T(\cdot \mid x_i)$, recording each response's sequence log-probability $\log p_T(y_n^T \mid x_i)$ and token count $|y_n^T|$.

Semantic clustering. As discussed in Section 2, responses such as *"London"*, *"London is the capital of the UK"*, and *"The UK's capital is London"* are semantically identical but produce large token-level divergence. To prevent this aleatoric variation from inflating uncertainty estimates, we extract the final answer from each teacher response and cluster semantically equivalent answers into equivalence classes $\mathcal{C} = \{c_1, \ldots, c_J\}$. Two answers are placed in the same cluster only if one can be derived from the other, i.e., they express the same meaning in different surface forms (e.g., $\frac{1}{2}$ and 0.5, or $x = 3$ and 3). Answers that match as exact strings are merged directly. For non-matching pairs, a Qwen-4B judge [7] determines pairwise semantic equivalence, and all judgments are resolved into clusters with a Union-Find algorithm.

Intra-cluster vs. inter-cluster entropy. This clustering decomposes the total entropy of teacher outputs into two interpretable components. *Intra-cluster entropy* captures the lexical diversity *within* a semantic equivalence class. A cluster containing "0.5", "$\frac{1}{2}$", and "the answer is one half" has high intra-cluster entropy, reflecting the teacher's expressive richness rather than genuine confusion. This variation is aleatoric and should not be penalized. *Inter-cluster entropy*, by contrast, measures the spread of probability mass *across* semantically distinct answer classes. When the teacher assigns substantial mass to multiple contradictory clusters (e.g., both "3" and "5" for the same prompt), inter-cluster entropy is high, signaling epistemic uncertainty. Our semantic entropy H_{sem} captures precisely the inter-cluster component by collapsing all within-cluster variation before computing entropy.

Probability-weighted semantic entropy. Each response's log-probability is first length-normalized to avoid penalizing longer outputs. The probability mass of semantic class c is then computed as

$$p(c \mid x_i) = \frac{\sum_{n \in c} \exp\bigl(\log p_T(y_n^T \mid x_i) \,/\, |y_n^T|\bigr)}{\sum_{n=1}^{N} \exp(\log p_T(y_n^T \mid x_i) \,/\, |y_n^T|)}, \tag{4}$$

and the semantic entropy is

$$H_{\text{sem}}(x_i) = -\sum_{c \in \mathcal{C}} p(c \mid x_i) \log p(c \mid x_i). \tag{5}$$

We normalize by the maximum possible entropy to obtain a value in $[0, 1]$ and define the per-prompt distillation weight as

$$w_{\text{se}}(x_i) = 1 - \frac{H_{\text{sem}}(x_i)}{\log N}. \tag{6}$$

A weight near 1 indicates that the teacher consistently agrees on the same semantic answer (low epistemic uncertainty), while a weight near 0 indicates contradictory responses across clusters (high epistemic uncertainty). These weights are precomputed and stored with the training data.

3.2 Uncertainty-Calibrated On-Policy Distillation

At each training step, the student generates K responses per prompt from its current policy $\{y^{(1)}, \ldots, y^{(K)}\} \sim p_S(\cdot \mid x; \theta_S)$. For each student-generated response y, the frozen teacher provides its top-k logit values and corresponding token indices at every position, reducing memory and communication from the full vocabulary size V to $k \ll V$.

The token-level distillation loss at position t is the cross-entropy between the teacher's and student's distributions over the top-k tokens

$$\ell_t = -\sum_{j=1}^{k} \tilde{p}_T(v_j \mid x, y_{<t}) \log p_S(v_j \mid x, y_{<t}; \theta_S), \tag{7}$$

where $\tilde{p}_T$ denotes the teacher's probability renormalized over the top-k tokens and $\{v_1, \ldots, v_k\}$ are the teacher's top-k token indices.

The standard on-policy distillation objective averages ℓ_t uniformly over all response tokens, treating every prompt equally regardless of teacher reliability. UOPD instead calibrates each sample's contribution by its precomputed semantic entropy weight $w_{\text{se}}(x)$. For a prompt x and student-generated response y, the uncertainty-calibrated loss is

$$\mathcal{L}(x, y; \theta_S) = \frac{w_{\text{se}}(x)}{\sum_t \mathbb{1}[t \in \text{resp}]} \sum_{t \in \text{resp}} \ell_t, \tag{8}$$

where the sum runs over response token positions (excluding the prompt). When the teacher is confident about a prompt ($w_{\text{se}} \approx 1$), the student receives the full distillation signal. When the teacher is uncertain ($w_{\text{se}} \approx 0$), the loss is suppressed, preventing the propagation of erroneous teacher guidance.

3.3 Overall Training Objective

The full training objective averages over all prompts and their K on-policy rollouts

$$\mathcal{L}_{\text{UOPD}}(\theta_S) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}(x, y^{(j)}; \theta_S). \tag{9}$$

Algorithm 1 Uncertainty-Calibrated On-Policy Distillation (UOPD)

Require: Training prompts $\mathcal{D} = \{x_i\}_{i=1}^{M}$, teacher p_T, student $p_S(\cdot;\theta_S)$, SE samples N, rollouts K, top-k

— Phase 1: Offline Semantic Entropy (Sec. 3.1) —
for each prompt $x_i \in \mathcal{D}$ **do**
    Sample N responses $\{y_1^T, \ldots, y_N^T\} \sim p_T(\cdot \mid x_i)$
    Extract final answers; cluster into semantic classes $\mathcal{C}$
    Compute semantic entropy $H_{\text{sem}}(x_i)$ (Eq. 5)
    Store weight $w_{\text{se}}(x_i) = 1 - H_{\text{sem}}(x_i)/\log N$ (Eq. 6)
end for
— Phase 2: On-Policy Distillation (Sec. 3.2) —
while not converged **do**
    Sample mini-batch $\mathcal{B} \subset \mathcal{D}$
    for each $x \in \mathcal{B}$ **do**
        Generate K rollouts $\{y^{(1)}, \ldots, y^{(K)}\} \sim p_S(\cdot \mid x;\theta_S)$
        for each rollout $y^{(j)}$ **do**
            Query teacher for top-k logits at each token position of $y^{(j)}$
            Compute token-level CE: $\ell_t = -\sum_v \tilde{p}_T(v) \log p_S(v;\theta_S)$
        end for
        Compute weighted loss $\mathcal{L}(x, y^{(j)};\theta_S)$ (Eq. 8)
    end for
    Update θ_S with $\nabla_{\theta_S} \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}(x, y^{(j)};\theta_S)$
    Sync θ_S to vLLM generation engines
end while

At each training step, the updated student weights are synchronized back to the vLLM generation engines, ensuring that the next round of rollouts reflects the latest policy. This on-policy loop continues until convergence.

Algorithm 1 summarizes the complete UOPD pipeline.

4 Experiments

4.1 SURE-Math Dataset

We introduce **SURE-Math** (**S**emantic-**U**ncertainty **RE**asoning **Math**), a mathematical reasoning dataset in which every problem is annotated with the teacher's precomputed semantic entropy score.

We first collect 1,000 mathematics problems spanning middle-school to competition level, each paired with a verified ground-truth answer. Of these, 200 are reserved as a held-out test set (Section 4.3); the remaining 800 are added to the training pool.

We further aggregate seed problems from public sources across four difficulty tiers. *Easy*: ScaleQuest-Math [8] (15K). *Medium*: NuminaMath-CoT [9] (10K) and MATH [10] (12.5K). *Hard*: Omni-MATH [11] (4.4K) and OlympiadBench [12] (5K). *Competition*: AIME 2024–2025, HMMT Feb & Nov 2025, and AMO-Bench [13].

Starting from these seeds, we synthesize new problems with Qwen2.5-72B-Instruct using an Evol-Instruct [14] style pipeline. Six evolution strategies are applied, each realized by a dedicated system prompt: *harder* (increase reasoning steps or add constraints), *rewrite* (change context and wording while preserving the underlying skill), *algebraize* (replace concrete values with variables), *apply* (embed the concept in a real-world scenario), *compose* (combine with a different branch of mathematics), and *competition* (transform into AMC/AIME/Olympiad style). Each seed undergoes 1–5 rounds of iterative evolution depending on its difficulty tier: easy seeds receive 1 round, medium seeds 2 rounds, hard seeds 3 rounds, and competition seeds 5 rounds. In follow-up rounds, a different strategy (drawn from *harder*, *competition*, and *compose*) is applied to the output of the previous round, progressively increasing difficulty. Strategy selection within each tier is weighted to favor *harder* and *competition* for higher-tier seeds.

For each of the 8,000 problems, we sample N=8 responses from Qwen3-8B [7] at temperature 0.7, extract the `\boxed{}` answer from each response, and compute the probability-weighted semantic entropy as described in Section 3.1. The resulting per-problem semantic entropy score and the corresponding distillation weight w_{se} are stored alongside each problem.

We reserve 200 of the 1,000 manually collected problems (which carry ground-truth labels) as a held-out test set. The remaining 800 curated problems and all 8,000 evolved problems form the training set.

4.2 Experimental Setup

Models. We evaluate three student–teacher pairs of increasing scale: Qwen3-1.7B/Qwen3-8B, Qwen3-4B/Qwen3-8B, and Qwen3-8B/Qwen3-30B [7], all instruction-tuned variants. Semantic clustering for the offline SE computation uses a Qwen2.5-3B-Instruct judge for pairwise semantic equivalence.

Training details. We train with the on-policy distillation objective described in Eq. 8. At each step, the student generates K=4 rollouts per prompt. The teacher provides its top-512 logit values at every token position. We use a learning rate of 3×10^{-6} with a cosine schedule and 5% warmup, AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient clipping at 1.0, and a global batch size of 128. Training runs for one epoch over 50,000 samples (with replacement from the 8,800 training problems). All models use bfloat16 precision and FlashAttention-2.

For each student–teacher pair, we compare UOPD against the following methods. (1) **Base**: the student model without any distillation (lower bound). (2) **Teacher**: the teacher model (upper bound). (3) **Standard OPD**: on-policy distillation with uniform weighting ($w_{\text{se}} = 1$ for all prompts), following **(author?)** [3]. (4) **GRPO**: Group Relative Policy Optimization, which trains the student using reward signals from correct and incorrect rollouts. (5) **Self-Distilled Reasoner**: on-policy distillation with ground-truth chain-of-thought injected into the teacher's system prompt [1].

4.3 Evaluation

Benchmarks. We evaluate all methods on three public mathematical reasoning benchmarks of increasing difficulty, plus our held-out test set: **MATH-500** [10], 500 competition-level problems spanning seven subjects; **AIME 2025**, 30 problems from the American Invitational Mathematics Examination (Parts I and II); **AMO-Bench** [13], 50 IMO-level competition problems; and our held-out **SURE-Math test set** (200 curated problems with ground-truth labels).

Metrics. We report pass@1 accuracy with greedy decoding (temperature 0) and avg@16 accuracy with 16 samples at temperature 1.2 and top-p=0.95.

4.4 Main Results

Table 1 presents the main comparison across all benchmarks.

4.5 Ablation Studies

Effect of uncertainty weighting. We compare three weighting strategies: (1) uniform weighting ($w_{\text{se}} = 1$), which reduces to standard on-policy distillation; (2) binary filtering, which discards all prompts with $H_{\text{sem}} > \tau$ for a threshold τ; and (3) soft weighting ($w_{\text{se}} = 1 - H_{\text{sem}}/\log N$), which is our default. Table 2 reports the results.

Number of SE samples N. The number of teacher rollouts N used for semantic entropy estimation controls the resolution of the uncertainty estimate. We vary $N \in \{2, 4, 8, 16\}$ and measure downstream distillation performance.

Top-k logit truncation. Transmitting the full vocabulary ($|V| = 151{,}936$) is memory-intensive. We compare $k \in \{128, 512, 2048\}$ and full-vocabulary distillation, measuring both accuracy and peak GPU memory.

Table 1: Main results on mathematical reasoning benchmarks. We report pass@1 (greedy) and avg@16 accuracy (%). Best student-sized results are **bolded**.

	MATH-500		AIME 2025		AMO-Bench		SURE-Math	
Method	pass@1	avg@16	pass@1	avg@16	pass@1	avg@16	pass@1	avg@16
Student: Qwen3-1.7B **Teacher:** Qwen3-8B								
Base (1.7B)	–	–	–	–	–	–	–	–
Standard OPD	–	–	–	–	–	–	–	–
GRPO	–	–	–	–	–	–	–	–
Self-Distilled Reasoner	–	–	–	–	–	–	–	–
UOPD (Ours)	–	–	–	–	–	–	–	–
Student: Qwen3-4B **Teacher:** Qwen3-8B								
Base (4B)	–	–	–	–	–	–	–	–
Standard OPD	–	–	–	–	–	–	–	–
GRPO	–	–	–	–	–	–	–	–
Self-Distilled Reasoner	–	–	–	–	–	–	–	–
UOPD (Ours)	–	–	–	–	–	–	–	–
Student: Qwen3-8B **Teacher:** Qwen3-30B								
Base (8B)	–	–	–	–	–	–	–	–
Standard OPD	–	–	–	–	–	–	–	–
GRPO	–	–	–	–	–	–	–	–
Self-Distilled Reasoner	–	–	–	–	–	–	–	–
UOPD (Ours)	–	–	–	–	–	–	–	–

Table 2: Ablation on uncertainty weighting strategies.

Weighting	**MATH-500**	**AIME 2025**	**AMO-Bench**	**SURE-Math**
Uniform ($w = 1$)	–	–	–	–
Binary filter ($\tau = 0.3$)	–	–	–	–
Binary filter ($\tau = 0.5$)	–	–	–	–
Soft weighting (UOPD)	–	–	–	–

Number of student rollouts K. We ablate $K \in \{1, 2, 4, 8\}$ on-policy rollouts per prompt to understand the trade-off between training diversity and computational cost.

4.6 Analysis

Convergence speed.

Semantic entropy distribution. Figure **??** shows the distribution of semantic entropy scores across SURE-Math. The majority of problems have low SE, indicating that the teacher is confident on most prompts. The long tail of high-SE problems represents cases where the teacher genuinely disagrees with itself, and these are precisely the samples that UOPD downweights.

Qualitative examples. Table **??** shows representative examples where UOPD's uncertainty weighting helps. For low-SE prompts, the teacher provides consistent guidance and the student learns effectively. For high-SE prompts, the teacher produces contradictory answers; UOPD suppresses these samples, preventing the student from learning incorrect reasoning patterns.

5 Related Work

5.1 Knowledge Distillation for Language Models

Traditional knowledge distillation [2] trains student models to match teacher output distributions. For autoregressive language models, supervised KD [15] and sequence-level KD [16] are widely used. However, these off-policy approaches suffer from distribution mismatch between training (teacher-generated or ground-truth sequences) and inference (student-generated sequences).

On-policy distillation [3] addresses this by training students on their own generated outputs, using teacher logits as labels. GKD (Generalized Knowledge Distillation) demonstrates strong improvements on summarization and translation tasks. Our work extends on-policy distillation to reasoning tasks by incorporating verification signals and contrastive objectives.

5.2 Learning from Verification Feedback

Recent work on mathematical reasoning leverages verification to improve model training. GRPO [**?**] and similar RL-based approaches optimize for verified correctness using policy gradient methods. V-STaR [**?**] iteratively generates verified solutions for self-improvement.

However, these methods rely solely on sparse binary rewards (correct/incorrect) and do not leverage dense token-level teacher guidance. TCD bridges this gap by combining verification with token-level distillation.

5.3 Contrastive Learning for Language Models

DPO [17] introduced preference-based contrastive learning for alignment, training models to prefer chosen responses over rejected ones. SimPO [**?**] and other variants explore different contrastive formulations.

Our work adapts contrastive learning to the distillation setting: we use verified correct responses (from reference data or teacher generation) as "chosen" and student-generated incorrect responses as "rejected", enabling the student to learn from its mistakes with teacher guidance.

6 Conclusion

We presented Token-level Contrastive Distillation (TCD), a framework that combines on-policy generation, verification-based learning, and dense teacher guidance for distilling reasoning capabilities into smaller language models. By unifying distillation on correct traces with contrastive learning on errors, TCD effectively leverages both sparse verification signals and rich token-level teacher feedback.

Our experiments on GSM8K demonstrate that TCD enables a 1.7B student model to achieve strong mathematical reasoning performance when distilled from an 8B teacher. The framework is efficient and scalable, using vLLM for on-policy generation and memory-optimized techniques for handling large vocabularies.

Limitations and Future Work. **Scalability**: Current experiments use 4 GPUs; scaling to multi-node setups requires careful optimization of communication backends (NCCL vs. Gloo). **Generalization**: We focus on mathematical reasoning; extending to other reasoning tasks (code, commonsense) is an important direction. **Teacher quality**: Our approach assumes access to a capable teacher; exploring self-improvement scenarios is promising.

TCD opens avenues for efficient reasoning model deployment by making strong reasoning capabilities accessible in compact models suitable for resource-constrained environments.

References

[1] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

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

[3] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*, 2024.

[4] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[5] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, 2017.

[6] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *International Conference on Learning Representations (ICLR)*, 2023.

[7] An Yang, Anfeng Yang, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[8] Zhengyang Yuan, Jiawei Liu, Boyuan Zi, Hao Ning, Kai Zheng, Minghao Chen, et al. Scalequest: Scalable data synthesis for mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

[9] AI-MO Team. Numinamath-cot: A large-scale mathematical reasoning dataset with chain-of-thought annotations. 2024. `https://huggingface.co/datasets/AI-MO/NuminaMath-CoT`.

[10] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, 2021.

[11] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

[12] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

[13] Meituan Longcat Team. Amo-bench: Assessing mathematical olympiad problem solving for large language models. 2025. `https://huggingface.co/datasets/meituan-longcat/AMO-Bench`.

[14] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. In *International Conference on Learning Representations (ICLR)*, 2024.

[15] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[16] Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

[17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2023.

A Implementation Details

A.1 Distributed Training Architecture

Our implementation uses Ray [**?**] for distributed actor management and vLLM [**?**] for efficient LLM serving across 4× NVIDIA H100 GPUs. The GPU allocation differs between the two phases.

Phase 1: Offline semantic entropy computation. This phase runs as a standalone script before training begins. Two vLLM model instances are loaded on the same GPU. The **teacher vLLM engine** (GPU 0) serves the teacher model (e.g., Qwen3-8B) with `gpu_memory_utilization=0.65` and `tensor_parallel_size=1`; for each prompt it generates N=8 responses in a single batched call via `engine.generate()`. The **cluster judge vLLM engine** (GPU 0, colocated) serves the semantic equivalence judge (Qwen2.5-3B-Instruct) with `gpu_memory_utilization=0.20`; it is only invoked for answer pairs that do not match via exact string comparison, keeping its utilization low. Both engines share GPU 0 via vLLM's memory pre-allocation. The remaining GPUs are idle during this phase. Processing 8,800 prompts with batch size 300 completes in approximately 1–2 hours.

Phase 2: On-policy distillation. All 4 GPUs are utilized via Ray placement groups. **GPU 0** hosts the *student actor*: the student model (e.g., Qwen3-1.7B) wrapped in DeepSpeed ZeRO Stage 1 for gradient computation and parameter updates; a frozen reference copy of the initial student is colocated on the same GPU using fractional allocation (`num_gpus=0.2`). **GPU 1** hosts the *teacher actor*: the frozen teacher model loaded in inference mode, which receives student-generated rollouts and returns top-k logit values and token indices at every position. **GPUs 2–3** host two `LLMRayActor` *vLLM generation engines*, each initialized from the student's current weights. They generate K=4 on-policy rollouts per prompt in parallel. After each gradient step, the updated student weights are broadcast to both engines via `update_weight()` over the Gloo backend.

The training loop proceeds as follows at each step:

1. The Ray coordinator dispatches a mini-batch of prompts to the 2 vLLM engines.
2. Each engine generates $K/2$=2 rollouts per prompt (total K=4 per prompt across both engines).
3. Rollouts are sent to the teacher actor, which computes top-k logits for all token positions.
4. The student actor receives rollouts, teacher logits, and precomputed w_{se} weights, then performs a gradient update.
5. Updated student weights are synchronized back to both vLLM engines for the next iteration.

A.2 Memory Optimization Techniques

Top-K Teacher Logits. Instead of storing full vocabulary logits (151,936 × 4 bytes = 608 KB per position), we keep only top-512 values and indices:

```
topk_vals, topk_ids = logits.topk(k=512, dim=-1)
# Store: 512 × 4 bytes (vals) + 512 × 4 bytes (ids) = 4 KB
```

This achieves 150× memory reduction with minimal accuracy loss.

Token-by-token Distillation Loss. We compute KL divergence incrementally to avoid materializing large tensors:

```
for t in range(seq_len):
    teacher_probs_t = F.softmax(teacher_vals[t], dim=-1)
    student_logprobs_t = F.log_softmax(student_logits[t], dim=-1)
    student_logprobs_topk = student_logprobs_t.gather(-1, teacher_ids[t])
    kl_t = -(teacher_probs_t * student_logprobs_topk).sum(-1)
```

A.3 Communication Backend Selection

For non-colocated setups, vLLM workers require collective communication for weight updates. **NCCL** is fast but fails when actors reside in separate Ray placement groups due to `CUDA_VISIBLE_DEVICES` isolation. We therefore use **Gloo**, a CPU-based fallback that works across placement groups, when `tensor_parallel_size=1`.

B Additional Experimental Results

B.1 Hyperparameter Sensitivity

TODO: Add figures/tables for learning rate sweep, temperature sweep for generation, beta sweep for contrastive loss, and number of samples per prompt (K).

B.2 Evaluation Protocol Details

For GSM8K evaluation, we:

1. Generate greedy responses (temperature=0, n=1)
2. Extract final numerical answer using regex
3. Compare with ground-truth using math_equal() for numerical equivalence
4. Report accuracy = correct / total

B.3 Computational Cost

Training on the SURE-Math dataset takes approximately **TODO** wall-clock hours on $4\times$ H100 (**TODO** GPU-hours total). The time breakdown is: on-policy generation **TODO**%, teacher logits computation **TODO**%, and student training **TODO**%.

C Reproducibility

Code will be released at `https://github.com/TODO`. Full training configurations are provided in YAML format.