IMPROVING TEST-TIME SCALING FOR SOFTWARE ENGINEERING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling (TTS) offers a promising path for solving complex software engineering tasks, yet its effectiveness depends on generating correct solutions within a fixed rollout budget. We introduce ENTROPO, an entropy-enhanced preference optimization framework for multi-turn, tool-using agents. We formulate agent alignment as an entropy-regularized MDP and derive the ENTROPO-DPO and ENTROPO-KTO objectives. Theoretically, we analyze why standard DPO/M-DPO objectives collapse and show that ENTROPO promotes both accuracy and diversity among correct solutions. Empirically, we validate ENTROPO on SWEBENCH, LIVECODEBENCH, and TERMINALBENCH, where it achieves state-of-the-art results among open-weight models. At the time of leaderboard submission, our 30B model ranked 1st on SWEBENCH-LITE and 4th on SWEBENCH-VERIFIED among open-weight submissions, trailing only models over 10× larger on SWEBENCH-VERIFIED. These results demonstrate the effectiveness of ENTROPO for improving software engineering performance with test-time compute scaling. To support further research, we release our code and data through an anonymous link.

1 INTRODUCTION

Large Language Models (LLMs) are increasingly used as software engineering (SWE) agents, but repository-level SWE tasks are rarely solved in one shot. A capable agent must inspect files, search through a codebase, run tests, read execution feedback, edit code, and revise its hypothesis over multiple turns (Yang et al., 2024; Wang et al., 2025; Zhang et al., 2024; Jain et al., 2025), as illustrated in Figure 1. This multi-turn, tool-using structure is precisely why test-time scaling (TTS) has become important: by sampling multiple trajectories and selecting or verifying candidates, an agent can explore alternative hypotheses and increase the chance of completing task (Snell et al., 2024; Beeching et al.; Yao et al., 2023; Xu et al., 2024a). TTS has already shown great potential in improving task accuracy for SWE agents (Jain et al., 2025).

Recently, the high cost of online tool calls and rollouts in SWE has driven increasing interest in offline preference learning frameworks (Dai et al., 2025; Zhang et al., 2025; Xiong et al., 2025a), such as DPO (Rafailov et al., 2023), KTO (Ethayarajh et al., 2024), and their multi-turn variants (Xiong et al., 2025b). However, empirical studies have repeatedly observed diversity collapse after alignment or fine-tuning (Kirk et al., 2024; Padmakumar & He, 2024; O'Mahony et al., 2024; Murthy et al., 2025), and this effect is especially concerning for SWE agents with TTS: concentrated failure modes can produce repeated unsuccessful attempts even with more rollouts. Crucially, while prior work has documented this collapse empirically, the underlying mechanics remain opaque; a theoretical explanation the fundamental root cause of this collapse has yet to be established.

Moreover, mitigating this collapse and improving correctness is non-trivial. Simple inference-time heuristics such as increasing sampling temperature can produce more varied outputs, but often inject randomness and hurt correctness. Data-level approaches provide another natural direction. For example, DivPO (Lanchantin et al., 2025) constructs preference pairs by selecting more diverse chosen responses and less diverse rejected responses, which can improve diversity without changing the training objective. However, such methods still optimize the standard DPO objective after the pairs are constructed, so the learned policy can continue to concentrate probability mass on the dominant preferred modes. More broadly, existing diversity-preserving methods (Lanchantin et al., 2025; Slocum et al., 2025; Pan et al., 2026; Wang et al., 2024; Shekhar et al., 2024) are mostly

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

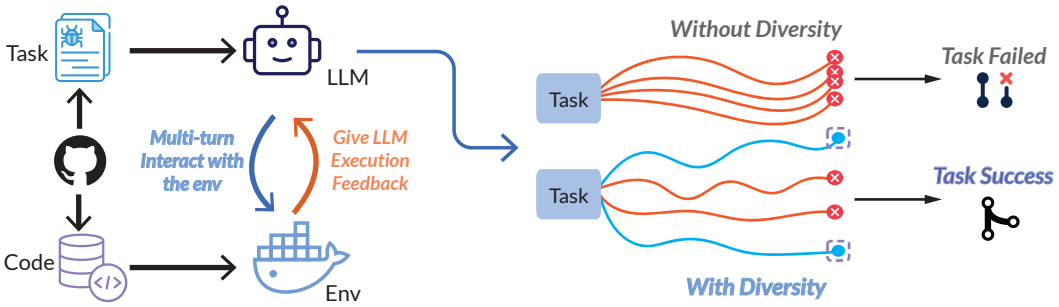


Figure 1: **TTS for SWE tasks.** Given a SWE task, an LLM agent interacts with a sandboxed environment over multiple turns, receiving execution feedback. TTS samples alternative solutions, whose utility depends on per-rollout correctness and candidate selection.

designed for single-turn response generation or paired preferences, leaving open how to jointly improve correctness and diversity in long-horizon, multi-turn tool-use trajectories, while supporting both paired and KTO-style pointwise feedback.

To address these limitations, we introduce ENTROPO, an entropy-enhanced preference optimization framework for multi-turn SWE agents. We show that standard preference optimization methods (*e.g.,* DPO, M-DPO) preserve the reference policy's probability imbalance among equally correct trajectories, creating a trajectory-level "winner-take-all" effect that limits TTS. ENTROPO instead formulates alignment as an entropy-regularized finite-horizon MDP and derives objectives for both paired and pointwise feedback. These objectives preferentially upweight underrepresented correct trajectories, improving diversity while remaining close to the reference policy. We further prove that, at the regularized optimum, ENTROPO achieves higher accuracy than standard DPO while increasing conditional entropy over correct trajectories.

We empirically validate ENTROPO across models from different families and sizes (up to 106B parameters) on SWEBENCH-VERIFIED (Chowdhury et al., 2024), SWEBENCH-LITE (Jimenez et al., 2024), LIVECODEBENCH (Jain et al., 2024), and TERMINALBENCH (Team, 2025b). Our results demonstrate higher task accuracy both with a single rollout and with TTS using multiple rollouts. Our approach also achieves SOTA results compared to other solutions. At the time of leaderboard submission, our 30B model ranks 1st on SWEBENCH-LITE and 4th on SWEBENCH-VERIFIED official leaderboard (surpassed only by models over 350B, which are 10x larger). Our contributions can be summarized as follows:

- **Multi-turn preference optimization:** We formulate preference learning for tool-using SWE agents as an entropy-regularized finite-horizon MDP and derive ENTROPO-DPO and ENTROPO-KTO through a reward-sum decomposition.
- **Collapse mechanism and theoretical guarantee:** We characterize the collapse mechanism of standard DPO/M-DPO and formally prove that ENTROPO jointly improves accuracy and the conditional entropy over correct solutions.
- **Support for pointwise feedback:** Beyond paired preferences, ENTROPO naturally supports KTO-style trajectory feedback, making it applicable to more flexible SWE settings.
- **Empirical effectiveness for TTS:** ENTROPO delivers SOTA among open-weight models on SWEBENCH-VERIFIED, SWEBENCH-LITE, LIVECODEBENCH, and TERMINALBENCH, and its margin over baselines grows with the rollout budget.

2 RELATED WORK

LLM Post-training. RLHF has become the standard approach for aligning LLMs with human preferences (Ouyang et al., 2022; Ziegler et al., 2019; Bai et al., 2022; Schulman et al., 2017), but the PPO-style online approach is computationally intensive, as it requires numerous interactions with a reward model or live environment to generate samples during training (Xu et al., 2024b; Wei et al., 2025). Offline preference-learning methods such as DPO (Rafailov et al., 2023) and KTO (Ethayarajh et al., 2024) provide a cheaper alternative by directly optimizing on preference or pointwise feedback. However, these objectives can reduce output diversity after alignment (Kirk et al., 2024; Padmakumar

& He, 2024; O'Mahony et al., 2024; Murthy et al., 2025), which can degrade the final task success rate for SWE agents.

A growing line of work targets this collapse, these methods can be roughly grouped by where diversity is introduced. **Data-level methods** alter the preference pairs prior to optimization. For example, DivPO (Lanchantin et al., 2025) retains the standard DPO loss but constructs diversity-aware pairs by selecting more diverse chosen responses and less diverse rejected responses. While easily integrated into existing pipelines, this leaves the underlying training objective unchanged. The analyzed DPO optimum therefore still preserves relative imbalance among equally rewarded responses. To more fundamentally alter this loss landscape, **objective-level methods** modify the learning signal or regularization mechanism. These approaches generally fall into two camps: tackling entropy directly or changing the divergence metric. In the former, SPL (Slocum et al., 2025) decouples and reweights the cross-entropy and entropy components of the reverse-KL term, while QEMPO (Pan et al., 2026) maximizes token-level entropy under a reward constraint. In the latter, f-DPO (Wang et al., 2024) completely replaces the reverse-KL penalty with more mass-covering f-divergences. Despite their effectiveness in mitigating single-turn mode collapse, these methods share two key limitations: (i) they are designed specifically for *single-turn* alignment, and it remains unclear how to extend these objectives to multi-turn; and (ii) they primarily demonstrate improved diversity through empirical comparisons with DPO, lacking formal theoretical diversity analysis as well as correctness guarantee.

LLMs for Software Engineering. Repository-level benchmarks such as SWEBENCH (Jimenez et al., 2024; Chowdhury et al., 2024) have driven progress on automated bug fixing and patch generation, supported by agentic systems such as SWE-agent (Yang et al., 2024), Agentless (Xia et al., 2025), and OpenHands (Wang et al., 2025) that reason over large codebases via long sequences of tool calls. Recent work also explores agentic RL to enhance complex reasoning (Luo et al., 2025; Wei et al., 2025), but at substantial compute and orchestration cost. ENTROPO applies entropy-regularized offline preference optimization to multi-turn tool-use trajectories, aiming to improve task performance under fixed rollout budgets. This training approach complements test-time sampling and candidate selection by improving the policy that generates candidate solutions.

3 PROPOSED TECHNIQUE

Motivated by the discussion in §1, ENTROPO is designed as a *training-time* intervention for preserving trajectory diversity before test-time scaling is applied. Figure 1 illustrates the overall workflow. Given a SWE issue, an LLM agent interacts with a sandboxed repository over multiple turns. For TTS, it can use best-of-N sampling to enhance the performance, with pass@N measuring oracle candidate coverage. As illustrated in the figure, TTS can explore alternative solutions within the budget; its success also depends on per-rollout correctness and candidate selection. To formalize this, we introduce the following optimization framework and give the theoretical analysis. ENTROPO is a general framework that integrates with existing agent scaffolds, TTS strategies, and tool schemas.

3.1 PROBLEM SETUP AND ASSUMPTION

We frame multi-turn SWE task as a finite-horizon episodic Markov Decision Process (MDP), $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, H, \mathbb{P}, d_0, r \rangle$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, H is the maximum number of turns (horizon), $\mathbb{P}$ is the transition dynamics, d_0 is initial state distribution, and r is reward function.

An initial state $s_1 = x \sim d_0$ represents a SWE problem statement. At each step $h \in \{1, \ldots, H\}$, the agent's policy $\pi(a_h|s_h)$ observes the current state s_h, which contains the full interaction history, and generates an action a_h (*e.g.,* a bash command). The environment executes a_h, returns an observation o_h (*e.g.,* compiler output, test results, or tool feedback), and transitions to the next state $s_{h+1} = (s_h, a_h, o_h)$. This interaction results in a trajectory $\tau = (x, a_1, o_1, \ldots, a_H)$. We define the interaction completion y as the sequence of generated actions and observations, allowing us to denote the trajectory as the concatenation $\tau = [x, y]$.

We assume access to a preference dataset $\mathcal{D}$ consisting of pairs of trajectories. Preferences are provided at the **trajectory level**[1] by an automated oracle that evaluates whether the final code produced by a trajectory passes a suite of unit tests; trajectories that pass are strictly preferred over

[1]By construction, the reward can equivalently be written as $r(x, y) = r(s_H, a_H)$.

those that fail. Following prior work (Ouyang et al., 2022; Xiong et al., 2025b), we model preference probabilities using the Bradley-Terry(BT) model (Bradley & Terry, 1952):

$$P(y^+ \succ y^- | x, y^+, y^-) = \sigma(r(x, y^+) - r(x, y^-)) \tag{1}$$

where r is a latent reward function and $\sigma(\cdot)$ denotes the sigmoid function.

Lemma 3.1 includes a transition-value residual and thus generalizes to stochastic transitions. Our SWE environment is deterministic, making this residual zero pathwise in the analysis below. The diversity analysis further assumes a finite feasible trajectory space and reference-positive support. Policies may depend on the full history; after termination, absorbing actions have probability one under both policies.

3.2 Algorithmic Formulation of EntroPO

Standard preference optimization methods, such as DPO (Rafailov et al., 2023), can exhibit *diversity collapse* (Murthy et al., 2025), where the policy converges to a narrow set of solutions. This is particularly detrimental for reasoning tasks where diverse exploration is required for effective TTS.

To counteract this, we augment the standard preference optimization objective with a weighted entropy regularization term, $\lambda H(\pi)$. This term directly penalizes low-entropy policies, encouraging the model to maintain a broader distribution over viable action sequences. This encourages inference-time exploration while optimizing task reward; the guarantees below concern the regularized optimum.

Our full objective, framed as a regularized MDP, is to find the optimal policy π^* that maximizes the expected reward while encouraging diversity and staying close to a reference policy π_{ref}:

$$\max_{\pi} \mathbb{E}_{x\sim d_0, a_h\sim\pi(\cdot|s_h), o_h\sim\mathbb{P}_h(\cdot|s_h,a_h)} \Big[r(x, y) - \gamma \cdot D_{KL}(\pi || \pi_{\text{ref}}) + \lambda \cdot H(\pi(\cdot|x)) \Big], \tag{2}$$

where λ promotes diversity and γ penalizes deviation from π_{ref}. Here $D_{KL}(\pi\|\pi_{\text{ref}})$ and $H(\pi(\cdot|x))$ denote sums over visited states of the statewise action KL and entropy, respectively, with $H(\pi(\cdot|s)) = -\sum_a \pi(a|s) \log \pi(a|s)$. Under deterministic transitions, they equal the corresponding induced complete-trajectory quantities conditional on x.

To optimize Eqn. 2, we first define the regularized value function for policy π as:

$$V_h^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=h}^{H} \left(r(s_t, a_t) - \zeta \log \frac{\pi(a_t|s_t)}{\pi_{\text{ref}}(a_t|s_t)^{\frac{\gamma}{\zeta}}} \right) \middle| s_h = s \right], \tag{3}$$

where $\lambda \geq 0$, $\gamma > 0$, and $\zeta = \lambda + \gamma > 0$. Note that Eqn. 3 can be derived by expanding the KL divergence and entropy terms in Eqn. 2 into their logarithmic forms and rearranging the corresponding terms. By definition, we set $V_{H+1}(s) = 0$. To leverage standard results from the entropy-regularized MDP, we define an augmented reward function $r'(s, a) = r(s, a) + \gamma \log \pi_{\text{ref}}(a|s)$. This formulation absorbs the reference policy constraint, allowing the regularized Q-function to be written as:

$$Q_h^\pi(s, a) = r'(s, a) + \mathbb{E}_{s'\sim\mathbb{P}_h(\cdot|s,a)}[V_{h+1}^\pi(s')]. \tag{4}$$

The relationship between value function and Q-function follows the standard consistency equation:

$$V_h^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\zeta \log \pi(a|s) + Q_h^\pi(s, a)]. \tag{5}$$

Under this formulation, both the value function and the Q-function are consistent with the maximum entropy policy optimization framework (Ziebart, 2010). Consequently, the regularized optimal policy π^*, which maximizes $V_h^\pi(s)$, can be expressed in terms of the optimal Q-function Q_h^* and value function V_h^* as follows: $\pi^*(a|s) = \exp\{(Q_h^*(s, a) - V_h^*(s))/\zeta\}$. To derive the preference optimization objective, we establish a connection between the rewards and the optimal policy, inspired by DPO. By rearranging the optimality condition above and summing over $h \in [H]$, we obtain following lemma:

Lemma 3.1 (Reward Sum Decomposition)**.** *For a trajectory τ, define the transition-value residual $\Delta^*(\tau) := \sum_{h=1}^{H-1} [V_{h+1}^*(s_{h+1}) - \mathbb{E}_{s'\sim\mathbb{P}_h(\cdot|s_h,a_h)} V_{h+1}^*(s')]$. Then $\sum_{h=1}^{H} r(s_h, a_h) = \sum_{h=1}^{H} \zeta \log \frac{\pi^*(a_h|s_h)}{\pi_{ref}(a_h|s_h)^{\gamma/\zeta}} + V_1^*(s_1) + \Delta^*(\tau)$.*

Appendix A proves the lemma. Within each pair, the shared $V_1^*(s_1)$ cancels. For the non-deterministic transition, we can estimate the transition-value residual using a learned value network or Monte Carlo sampling. For the deterministic SWE transitions, it leads to $\Delta^*(\tau) = 0$, yielding the following loss.

Theorem 3.2. *For the multi-turn setting, assuming the preferences follow a BT model and the policy optimization objective is given in Eqn. 2, we have the following DPO-style loss function:*

$$L_{EntroPO-DPO}(\theta) = - \mathbb{E}_{(x,\tau^+,\tau^-)\in\mathcal{D}} \left[\log \sigma \left(\zeta \sum_{h=1}^{H} \left(\log \frac{\pi_\theta(a_h^+|s_h^+)}{\pi_{ref}(a_h^+|s_h^+)^{\frac{\gamma}{\zeta}}} - \log \frac{\pi_\theta(a_h^-|s_h^-)}{\pi_{ref}(a_h^-|s_h^-)^{\frac{\gamma}{\zeta}}} \right) \right) \right], \quad (6)$$

where (τ^+, τ^-) represents a preference pair.

Theoretically, under the BT assumption, optimizing this EntroPO-DPO loss yields the same optimal policy as maximizing the ground-truth reward in Eqn. 2, provided the implicit reward is perfectly recovered. We also derive our KTO-style loss variant, EntroPO-KTO, in Appendix B.

3.3 Correctness and Diversity Analysis of EntroPO

In this subsection, we first analyze the source of collapse in standard DPO and then establish theoretical guarantees that EntroPO can jointly improve correctness and correct-solution diversity. We begin by noting that the completion y consists of observation o from the environment and the model's response y'. Since the code execution environment is deterministic, the probability of a completion $P(y|x)$ is equivalent to $\pi(y'|x)$ (see Appendix C.1). Consequently, we parameterize $P(y|x)$ directly via a generative model $\tilde{\pi}(y|x)$. This reduction allows us to derive the optimal solution for $\tilde{\pi}(y|x)$ by maximizing the following objective:

Lemma 3.3. *With our re-parameterization, the objective in Eqn. 2 can be transformed to*

$$\max_{\tilde{\pi}} \mathbb{E}_{x\sim d_0, y\sim\tilde{\pi}(\cdot|x)}[r(x,y) + \zeta \cdot H(\tilde{\pi}(\cdot|x)) - \gamma \cdot H(\tilde{\pi}, \tilde{\pi}_{\text{ref}})], \quad (7)$$

where $H(\tilde{\pi}, \tilde{\pi}_{\text{ref}}) = -\tilde{\pi}(\cdot|x)^T \log \tilde{\pi}_{\text{ref}}(\cdot|x)$ represents the cross entropy between the learned policy $\tilde{\pi}$ and the referenced policy $\tilde{\pi}_{\text{ref}}$. The optimal policy is then given by $\tilde{\pi}^*(y|x) \propto \tilde{\pi}_{\text{ref}}(y|x)^{\frac{\gamma}{\zeta}} \exp\left(\frac{r(x,y)}{\zeta}\right)$.

The derivation of the closed-form optimal solution $\tilde{\pi}^*$ can be found in Appendix C.2 [2]. By analyzing the relative increase in log-probability from the optimal policy to the reference policy, we show why the standard DPO method can lead to *diversity collapse* in our multi-turn setting, whereas EntroPO counteracts this effect by preferentially upweighting correct responses that are underrepresented under the reference policy.

Proposition 3.4 (Mitigation of Diversity Collapse via Implicit Probability Flattening)**.** *Consider two completions y_1, y_2 yielding identical positive rewards $r(x, y_1) = r(x, y_2) = 1$. Standard DPO ($\gamma = \zeta$) (Rafailov et al., 2023; Xiong et al., 2025b) applies a uniform scaling factor to both, preserving the relative likelihood ratio of the reference policy ($\frac{\tilde{\pi}^*(y_1)}{\tilde{\pi}^*(y_2)} = \frac{\tilde{\pi}_{\text{ref}}(y_1)}{\tilde{\pi}_{\text{ref}}(y_2)}$) and failing to correct existing diversity collapse. In contrast,* EntroPO *($\gamma < \zeta$) induces a relative log-likelihood gain negatively correlated to the log-reference likelihood.*

We defer the proof to Appendix C.3. This reweighting motivates studying correctness and correct-solution diversity jointly: an agent should solve the task reliably while avoiding concentration on a few successful trajectories. For a fixed problem x, let Ω_x denote the finite feasible trajectory support, with $\tilde{\pi}_{\text{ref}}(y \mid x) > 0$ for every $y \in \Omega_x$. Under binary oracle rewards, denote the correct set $C = \mathcal{Y}^+ = \{y \in \Omega_x : r(x, y) = 1\}$ and the incorrect set $I = \Omega_x \setminus C$. We give the following definitions for correctness and correct-solution diversity.

Definition 3.5 (Correctness and Correct-Solution Diversity)**.** For any policy $\tilde{\pi}$, we define its *correctness* as $\text{Acc}_x(\tilde{\pi}) := \Pr_{y\sim\tilde{\pi}(\cdot|x)}[y \in C] = \sum_{y\in C} \tilde{\pi}(y \mid x)$. We define its *correct-solution diversity* using the conditional entropy $H_x^+(\tilde{\pi}) := -\sum_{y\in C} q_{\tilde{\pi}}(y \mid x) \log q_{\tilde{\pi}}(y \mid x)$, where $q_{\tilde{\pi}}(y \mid x) := \frac{\tilde{\pi}(y|x)}{\text{Acc}_x(\tilde{\pi})}, y \in C$.

[2] While treated $\tilde{\pi}$ as unconstrained, Eqn. 7 is technically constrained by the transition dynamics; details in Appendix C.2.

Thus, Acc_x measures how much probability is assigned to correct trajectories, while H_x^+ measures how broadly that probability is distributed within the correct set. We then present the following theorem to analyze the correctness and correct-solution diversity of our method against DPO.

Theorem 3.6. *Let $\tilde{\pi}_E^*$ and $\tilde{\pi}_D^*$ be the optimal policy for* EntroPO*-DPO and standard DPO, respectively, with the same reference policy and reward function. For $0 < \gamma < \zeta$, define $\alpha = \gamma/\zeta$ and $p_y = \tilde{\pi}_{\mathrm{ref}}(y \mid x)$, and let $p(\cdot \mid C)$ and $p(\cdot \mid I)$ denote the reference distribution conditioned on the correct and incorrect sets, respectively. If $\mathbb{E}_{y\sim p(\cdot|C)}\left[p_y^{\alpha-1}\right] \geq \exp\left(\frac{1}{\gamma} - \frac{1}{\zeta}\right) \mathbb{E}_{y\sim p(\cdot|I)}\left[p_y^{\alpha-1}\right]$, then the following inequalities hold jointly:*

$$\mathrm{Acc}_x(\tilde{\pi}_E^*) \geq \mathrm{Acc}_x(\tilde{\pi}_D^*) \qquad \textit{and} \qquad H_x^+(\tilde{\pi}_E^*) \geq H_x^+(\tilde{\pi}_D^*). \tag{8}$$

The accuracy inequality is strict when the displayed condition holds strictly. The entropy inequality holds independently of this condition and is strict when the reference distribution conditioned on C is nonuniform.

The proof is provided in Appendix C.4. Intuitively, the sufficient condition can be satisfied when individual correct trajectories have sufficiently lower reference probabilities than incorrect trajectories, allowing probability flattening to favor correct solutions. Since $\alpha < 1$, the reweighting also flattens the distribution within the correct set. Thus, EntroPO strictly improves both per-rollout accuracy and correct-solution diversity when the condition holds strictly and the reference distribution within the correct set is nonuniform.

4 Experiments

We evaluate EntroPO on benchmarks including SWEBench-Verified, SWEBench-Lite, LiveCodeBench, and TerminalBench. As our goal is to improve task success under a fixed rollout budget, we evaluate EntroPO through resolve rates and the gains from additional rollouts. §4.1 details the datasets, models, training/inference configurations and baselines. Additional setup details regarding the scaffold and running environment are provided in §D.2 and §D.3. §4.2 presents the main results. In §4.3, we analyze the impact of TTS budget, performance under different TTS selectors, and sensitivity to hyperparameters. Further analysis of per-rollout accuracy and success within 16 rollouts, qualitative case study, and convergence behavior are provided in §D.8.

4.1 Experiment Setup

Datasets. For SWEBench, we use R2E-Gym-subset (Jain et al., 2025) for offline preference learning. We evaluate on SWEBench-Verified and SWEBench-Lite, and no hint or web search is used. On LiveCodeBench (Jain et al., 2024), we follow the multi-turn code-generation setup from prior work (Chen et al., 2026): models are trained on TACO-train tasks (Li et al., 2023) and evaluated on LiveCodeBench tasks collected between Jan and Apr 2025, where the agent can iteratively use public-test feedback and the final submission is scored using hidden tests. On TerminalBench (Team, 2025b), we sample training tasks from Endless Terminals (Gandhi et al., 2026) and evaluate on TerminalBench 2.0 tasks. Detailed statistics for all datasets are provided in §D.1.

Models. We evaluate models from three different families, with sizes ranging up to 106B: Qwen3-4B-Instruct-2507 (Team, 2025a), Gemma-3-27b-it (Team et al., 2025), Qwen3-Coder-30B-A3B-Instruct, and GLM-4.5-Air-106B (Zeng et al., 2025a). For display convenience, we abbreviate the model names as Qwen3, Gemma-3, Qwen3-Coder, and GLM respectively. For readability, we report the Qwen3 results separately in Table 7.

Training and Inference. For offline preference learning, we train with LLaMAFactory (Zheng et al., 2024b) and set the maximum training sequence length to 18,000 tokens to accommodate long SWE trajectories. To manage memory, we use QLoRA (Dettmers et al., 2023) for GLM-4.5-Air and LoRA (Hu et al., 2022) for the other models. Unless noted, EntroPO uses $\zeta = 0.11, \gamma = 0.1$ and the sensitivity to ζ/γ is reported in §4.3. At inference, we allow up to 200 environment interactions per rollout and a maximum sequence length of 131,072 tokens, with temperature 0.7 and `top_k` $= 20$. For SWEBench, the official protocol requires submitting a single patch for evaluation and reporting pass@1 as the resolution rate. We use the R2E scaffold and follow its TTS strategy to use a hybrid selector to select the best trajectory from the generated candidates. For LiveCodeBench and

Table 1: **Main results.** For SWEBENCH, entries are resolve rate (pass@1, %) and TTS columns use 16 rollouts and submit just one patch. For LIVECODEBENCH and TERMINALBENCH, the TTS results are Pass@16 over multiple rollouts. Results with ± are mean ± std over three runs. Our method rows are shaded in light gray, and the best available result for each benchmark and model is bolded.

Method	Gemma-3	Gemma-3-TTS	Qwen3-Coder	Qwen3-Coder-TTS	GLM	GLM-TTS
SWEBENCH-VERIFIED						
Origin	7.1 (± 0.5)	12.5 (± 0.3)	37.7 (± 0.2)	42.5 (± 0.2)	51.4 (± 0.2)	55.0 (± 0.4)
Temp. = 1	6.8 (± 0.8)	11.6 (± 0.4)	37.1 (± 0.9)	41.8 (± 0.8)	50.9 (± 0.9)	54.6 (± 0.2)
SFT	7.0 (± 0.4)	12.1 (± 0.5)	43.8 (± 0.8)	47.2 (± 0.2)	51.5 (± 0.8)	55.0 (± 0.4)
M-DPO	9.2 (± 0.3)	14.6 (± 0.5)	47.8 (± 0.3)	52.8 (± 0.7)	52.0 (± 0.5)	56.2 (± 0.5)
M-KTO	9.0 (± 0.3)	14.8 (± 0.3)	48.8 (± 0.5)	53.8 (± 0.5)	52.7 (± 0.4)	57.0 (± 0.5)
DivPO	9.4 (± 0.3)	14.3 (± 0.2)	48.6 (± 0.5)	51.8 (± 0.4)	52.1 (± 0.4)	56.0 (± 0.5)
ENTROPO-DPO	**10.5 (± 0.3)**	**17.7 (± 0.4)**	49.8 (± 0.4)	57.7 (± 0.7)	52.5 (± 0.7)	57.5 (± 0.4)
ENTROPO-KTO	10.1 (± 0.4)	17.6 (± 0.3)	**51.6 (± 0.7)**	**59.4 (± 0.9)**	**53.5 (± 0.7)**	**58.7 (± 0.1)**
SWEBENCH-LITE						
Origin	6.0 (± 0.8)	8.9 (± 1.0)	28.0 (± 0.3)	31.3 (± 0.5)	43.5 (± 0.6)	46.0 (± 0.3)
Temp. = 1	5.8 (± 1.0)	8.2 (± 0.7)	27.3 (± 0.5)	31.1 (± 0.4)	42.9 (± 0.4)	45.4 (± 0.4)
SFT	5.9 (± 0.4)	8.8 (± 1.0)	33.9 (± 0.3)	36.7 (± 0.8)	43.9 (± 0.4)	46.1 (± 0.4)
M-DPO	8.7 (± 0.3)	12.4 (± 0.4)	41.7 (± 0.5)	44.9 (± 0.4)	44.2 (± 0.4)	47.0 (± 0.3)
M-KTO	8.9 (± 0.2)	12.7 (± 0.3)	42.0 (± 0.5)	45.2 (± 0.4)	44.4 (± 0.4)	47.3 (± 0.3)
DivPO	8.6 (± 0.2)	12.3 (± 0.3)	42.3 (± 0.5)	44.6 (± 0.4)	44.1 (± 0.4)	46.8 (± 0.4)
ENTROPO-DPO	**10.4 (± 0.7)**	14.4 (± 0.7)	43.7 (± 0.8)	48.2 (± 0.7)	44.6 (± 0.5)	47.9 (± 0.3)
ENTROPO-KTO	**10.4 (± 0.2)**	**14.6 (± 0.6)**	**44.0 (± 0.3)**	**49.2 (± 0.7)**	**44.9 (± 0.4)**	**48.4 (± 0.1)**
LIVECODEBENCH						
Origin	10.7 (± 0.3)	16.6 (± 0.9)	40.4 (± 0.7)	58.1 (± 0.4)	54.1 (± 0.3)	58.3 (± 0.9)
Temp. = 1	10.1 (± 0.3)	15.6 (± 0.2)	39.6 (± 0.3)	56.8 (± 0.7)	53.3 (± 0.5)	57.3 (± 0.5)
SFT	11.6 (± 0.7)	17.9 (± 0.6)	44.6 (± 0.6)	61.3 (± 0.7)	55.0 (± 0.4)	59.6 (± 0.9)
M-DPO	13.3 (± 0.4)	20.8 (± 0.4)	47.8 (± 0.7)	62.7 (± 0.3)	55.0 (± 0.7)	60.2 (± 0.9)
M-KTO	12.4 (± 0.3)	20.2 (± 0.7)	47.6 (± 0.7)	61.5 (± 0.5)	54.1 (± 0.3)	59.6 (± 0.3)
DivPO	12.8 (± 0.2)	19.8 (± 0.4)	47.4 (± 0.5)	61.5 (± 0.3)	54.5 (± 0.7)	59.0 (± 0.4)
ENTROPO-DPO	**15.0 (± 0.5)**	**24.8 (± 0.7)**	49.9 (± 0.7)	**67.8 (± 0.3)**	**56.8 (± 0.7)**	**63.0 (± 0.7)**
ENTROPO-KTO	14.1 (± 0.3)	24.2 (± 0.7)	**50.1 (± 0.3)**	67.2 (± 0.7)	56.2 (± 0.7)	62.5 (± 0.7)
TERMINALBENCH						
Origin	3.4 (± 0.9)	6.7 (± 0.9)	9.4 (± 1.4)	11.6 (± 1.4)	16.5 (± 0.5)	19.5 (± 1.4)
Temp. = 1	2.6 (± 0.5)	5.6 (± 0.9)	8.2 (± 0.5)	10.1 (± 0.9)	15.4 (± 0.5)	18.7 (± 0.5)
SFT	4.1 (± 1.1)	7.1 (± 1.4)	10.5 (± 1.4)	12.7 (± 1.4)	16.9 (± 0.9)	20.2 (± 0.9)
M-DPO	5.6 (± 0.9)	9.0 (± 0.9)	**11.2 (± 0.9)**	16.1 (± 1.4)	16.5 (± 0.5)	20.6 (± 1.4)
M-KTO	4.9 (± 0.5)	8.6 (± 0.5)	10.9 (± 0.5)	15.0 (± 1.4)	16.1 (± 1.4)	19.9 (± 0.5)
DivPO	4.9 (± 0.5)	8.6 (± 1.1)	10.1 (± 0.9)	14.6 (± 0.9)	16.1 (± 0.5)	19.9 (± 0.5)
ENTROPO-DPO	**6.7 (± 0.9)**	**10.5 (± 1.4)**	**11.2 (± 0.9)**	**19.5 (± 1.4)**	**17.2 (± 0.5)**	**22.8 (± 1.4)**
ENTROPO-KTO	**6.7 (± 0.9)**	9.7 (± 0.5)	**11.2 (± 0.9)**	**19.5 (± 1.4)**	**17.2 (± 0.5)**	**22.8 (± 1.4)**

TERMINALBENCH, we report pass@N over multiple rollouts as a natural TTS metric. Unless otherwise specified, we run $N = 16$ rollouts. For each fine-tuned method in the main benchmark suite, we train three models independently and report the mean ± standard deviation of their evaluation results. Additional training and inference details are provided in §D.4.

Baselines. We compare ENTROPO against several baselines: (1) the original base model, (2) temperature 1.0 at decoding time for the base model (Renze, 2024), (3) the SFT-tuned model trained on the same training tasks, (4) M-DPO/M-KTO (Xiong et al., 2025b) as the standard multi-turn preference optimization baseline, (5) DivPO (Lanchantin et al., 2025) as the diversity-oriented data augmentation baseline. Since DivPO is not directly applicable to multi-turn settings, we implement it within the M-DPO framework. As discussed in §2, DivPO is a data-level method, whereas other diversity-preserving methods modify the optimization objective and require non-trivial customization for multi-turn settings. Thus, we evaluate representative methods in a separate single-turn comparison in §D.6.2. All baselines use the same scaffold, training tasks, and inference budget as ENTROPO. We provide further baseline details in §D.5.

4.2 MAIN RESULTS

Comparison with Baselines. As shown in Table 1 and Table 7, ENTROPO achieves the strongest overall performance across all four benchmarks under the same scaffold, training data, and inference budget. Entropy-regularized preference learning yields higher task accuracy than simply raising the decoding temperature: across four benchmarks, Temp. = 1 is generally comparable to or below the

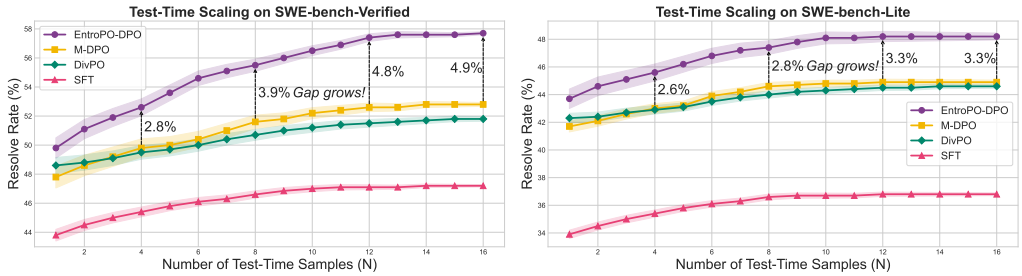


Figure 2: **The Impact of TTS Budget on Task Performance.** Performance of EntroPO-DPO, M-DPO, DivPO, and SFT on SWEBench-Verified (left) and SWEBench-Lite (right) as the number of parallel rollouts (N) increases. EntroPO consistently yields better scaling.

original model. SFT gives modest gains, while M-DPO and M-KTO provide stronger improvements. DivPO remains close to M-DPO, highlighting the value of an entropy-regularized objective for multi-turn tool-use trajectories.

In contrast, EntroPO-DPO improves Qwen3-Coder under TTS by 4.9 points on SWEBench-Verified, 3.3 points on SWEBench-Lite, 5.1 points on LiveCodeBench, and 3.4 points on TerminalBench. The same pattern holds for the KTO objective. Theorem 3.6 gives the corresponding optimum-level condition under which EntroPO-DPO assigns at least as much probability to correct trajectories as DPO. The gains extend across model scales: Qwen3 EntroPO-KTO with TTS reaches 11.5% on SWEBench-Verified and 10.0% on SWEBench-Lite, while Qwen3-Coder EntroPO-KTO reaches 59.4% and 49.2%, respectively. Qwen3-Coder EntroPO-DPO also reaches 67.8% Pass@16 on LiveCodeBench and 19.5% on TerminalBench, and GLM improves under QLoRA.

Comparison with Online RL. We additionally compare EntroPO-KTO with GRPO (Shao et al., 2024) using the same Qwen3-Coder backbone with R2E scaffold, and evaluation budget on SWEBench-Verified. EntroPO-KTO achieves 51.6% single-rollout pass@1 and 79.2% raw-candidate pass@16, compared with 46.4% and 71.8% for GRPO. Including offline trajectory collection, EntroPO-KTO uses approximately 576 H100-hours versus 18,432 for GRPO, yielding higher single-rollout accuracy and success within 16 rollouts at approximately $1/32$ of the compute in this configuration. The details are provided in §D.6.1.

Single-Turn Baseline Comparison. On BIG-Bench Hard (BBH) (Suzgun et al., 2022) with Gemma-3, EntroPO-KTO outperforms DPO, KTO, f-DPO (Wang et al., 2024), and offline QEMPO (Pan et al., 2026) under the same training data, training budget, and decoding configuration. It achieves 69.95% pass@1, 86.35% pass@16, and 70.25% mean@16, giving the highest score on each metric. Even in this single-turn setting, EntroPO achieves superior performance, suggesting that its combination of correctness-oriented preference learning and entropy regularization remains effective beyond multi-turn tasks. Details are provided in §D.6.2.

Comparison to the Official Leaderboard. To further compare EntroPO beyond our controlled baseline setting, we submitted our best model to the official SWEBench leaderboard. As shown in Table 8, our Qwen3-Coder EntroPO-KTO-TTS model achieves 60.4% on SWEBench-Verified and 49.7% on SWEBench-Lite, ranking 4th on SWEBench-Verified and 1st on SWEBench-Lite[3] among open-weight submissions at the time of submission. These results show that EntroPO remains highly competitive even outside our internal benchmark suite, despite using a substantially smaller model than many leaderboard systems.

4.3 Ablation and Further Analysis

In this section, we perform ablations and supporting analyses for EntroPO, mainly using Qwen3-Coder on SWEBench.

Impact of TTS Budget. We compare EntroPO-DPO against M-DPO, DivPO, and SFT with the Qwen3-Coder model on SWEBench-Verified. In Figure 2, under the same hybrid selector,

[3]The leaderboard rankings are time-sensitive and may change after our submission.

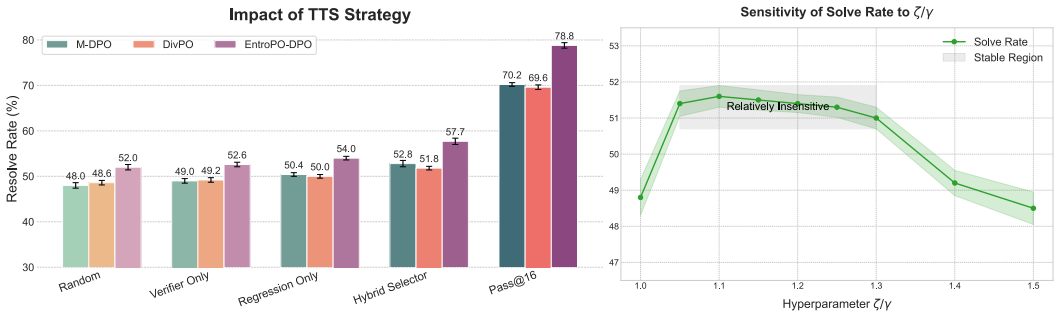


Figure 3: **Ablation Studies on** SWEBENCH-VERIFIED**.** (Left) Impact of TTS strategy at $N = 16$. (Right) Sensitivity analysis of the hyperparameter ζ/γ for ENTROPO-KTO. For visualization convenience, we plot the inverse of γ/ζ from Theorem 3.2.

ENTROPO-DPO's advantage over M-DPO on SWEBENCH-VERIFIED grows from 2.0 points at $N = 1$ (49.8% vs. 47.8%) to 4.9 points at $N = 16$ (57.7% vs. 52.8%) on SWEBENCH-VERIFIED. The KTO comparison shows the same pattern, with gains increasing from 2.8 to 5.6 points (Figure 4). Thus, ENTROPO improves single-rollout accuracy and benefits more from additional rollouts under the evaluated selector.

Impact of TTS Strategy. We further examine task performance under different test-time selection strategies at $N = 16$. ENTROPO-DPO outperforms M-DPO and DivPO under random, verifier-only, regression-test-only, and hybrid selection (Figure 3, left). Resolve rate evaluates the single selected patch, whereas pass@16 measures whether any candidate is correct and provides an upper bound on selection performance. The advantage of ENTROPO extends across the evaluated selection strategies, while effective selection further improves final task success. The gap between hybrid resolve rate and pass@16 shows that some tasks remain unresolved despite having a correct candidate.

Sensitivity to ζ/γ. The right plot in Figure 3 shows the performance of ENTROPO-KTO across different values of the hyperparameter ζ/γ. The model is robust to a reasonable range of ζ/γ, indicating that the performance is not highly sensitive to this hyperparameter. Performance degrades only when ζ/γ becomes excessively large, causing the training gradients to vanish.

Impact of Temperature on Performance. We also investigate whether increasing sampling temperature can replicate the benefits of ENTROPO. As detailed in §D.7, simply raising the temperature fails to deliver comparable performance gains. Instead, it degrades performance by introducing excessive sampling randomness, confirming that temperature tuning is no substitute for the principled entropy regularization of ENTROPO.

Further Analysis. We examine raw-candidate accuracy in §D.8, using 16 raw candidates before selection. On SWEBENCH-VERIFIED, ENTROPO-KTO achieves a mean@16 of 51.9%, compared with 48.5% for M-KTO, alongside a pass@16 of 79.2% versus 70.8%. These results show that ENTROPO improves both average per-rollout accuracy and the rate of obtaining at least one correct solution within 16 rollouts. The appendix also provides a multi-hunk task analysis, the qualitative case study, and the analysis of convergence dynamics.

5 CONCLUSION

We introduce ENTROPO, an entropy-enhanced preference optimization framework for multi-turn SWE agents. By explicitly regularizing the preference objective, ENTROPO improves both accuracy and the conditional entropy over correct solutions, thereby mitigating trajectory collapse while preserving solution quality. Across four benchmarks, ENTROPO improves single- and multi-rollout performance and achieves state-of-the-art results among open-weight models on the SWEBENCH leaderboard. These results support diversity-preserving alignment for test-time scaling in tool-using agents. Limitations and future directions are discussed in Appendix E.

AI USE STATEMENT

In this work, we used generative AI tools for grammar checking, spelling correction, and style improvement to enhance the readability of the manuscript. We have manually reviewed all AI-assisted text edits and ensured that the final content is accurate and consistent with the original manuscript. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks or formulate mathematical claims.

ETHICS STATEMENT

This work does not involve human-subject experiments or the collection of new personal data. We use publicly released models, datasets, benchmarks, and software repositories for research purposes under their stated licenses; their sources and licenses are documented in Table 2.

Table 2: **Licenses for datasets and benchmarks used in this work.**

Asset	Use in this work	License and source
R2E-Gym-subset	Preference-learning and verifier-training data for SWEBENCH experiments	Apache-2.0, official project repository: https://github.com/R2E-Gym/R2E-Gym
SWE-bench Verified and SWE-bench Lite	Evaluation benchmarks for repository-level SWE tasks	MIT, official repository: https://github.com/SWE-bench/SWE-bench
TACO-Cobalt / TACO-Train	Training tasks for LIVECODEBENCH experiments	MIT, official repository: https://github.com/OSU-NLP-Group/cobalt
LiveCodeBench	Evaluation benchmark for code-generation tasks	MIT, official repository: https://github.com/LiveCodeBench/LiveCodeBench
Endless Terminals	Training tasks for TERMINAL-BENCH experiments	MIT for the released dataset card; associated code repository is Apache-2.0: https://github.com/kanishkg/endless-terminals
TerminalBench 2.0	Evaluation benchmark for terminal-agent tasks	Apache-2.0, official repository: https://github.com/harbor-framework/terminal-bench-2

Although these resources are publicly available, software-repository data may contain contributor identifiers or security-relevant content, and we do not use such information to profile individuals. Improving coding agents can increase developer productivity and broaden access to capable open-weight systems, but these agents may also generate insecure or incorrect code or be misused to automate harmful software-development tasks. All agent rollouts and executable evaluations in this study were conducted in isolated, sandboxed environments.

REPRODUCIBILITY STATEMENT

We document the objectives, notation, assumptions, and theoretical results of ENTROPO in §3; complete derivations and proofs are provided in Appendix A, Appendix B, and §C.4. The experimental protocol, evaluated models, benchmarks, metrics, and principal hyperparameters are summarized in §4.1. Dataset sources, statistics, oracle definitions, and the construction of the SWE preference data are described in §D.1. The agent scaffolds and prompt formats are provided in §D.2, while training hyperparameters, inference budgets, decoding settings, and the test-time selector are specified in §D.4. Baseline implementations and additional comparisons are documented in §D.5 and §D.6; further results and analyses appear in §D.7 and §D.8. Hardware configurations and approximate runtimes are reported in §D.3, and the main results report the mean and standard deviation over three runs. An anonymous artifact containing the accompanying implementation materials is provided through the link in the abstract. Dataset and benchmark licenses are listed in Table 2.

REFERENCES

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Ziru Chen, Dongdong Chen, Ruinan Jin, Yingbin Liang, Yujia Xie, and Huan Sun. Bridging online and offline rl: Contextual bandit learning for multi-turn code generation. *arXiv preprint arXiv:2602.03806*, 2026.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Zhenlong Dai, Bingrui Chen, Zhuoluo Zhao, Xiu Tang, Sai Wu, Chang Yao, Zhipeng Gao, and Jingyuan Chen. Less is more: Adaptive program repair with bug localization and preference learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 128–136, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

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

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Kanishk Gandhi, Shivam Garg, Noah D Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. Direct language model alignment from online ai feedback. *arXiv preprint arXiv:2402.04792*, 2024.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. The impact of reasoning step length on large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1830–1842, 2024.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity. In *The Twelfth International Conference on Learning Representations*, 2024.

Jack Lanchantin, Angelica Chen, Shehzaad Dhuliawala, Ping Yu, Jason Weston, Sainbayar Sukhbaatar, and Ilia Kulikov. Diverse preference optimization. *arXiv preprint arXiv:2501.18101*, 2025.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl, 2025. Notion Blog.

Thang Luong, Edward Lockhart, et al. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad. *DeepMind Blog*, 2025.

Sonia Krishna Murthy, Tomer Ullman, and Jennifer Hu. One fish, two fish, but not the whole sea: Alignment reduces language models' conceptual diversity. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11241–11258, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Laura O'Mahony, Leo Grinsztajn, Hailey Schoelkopf, and Stella Biderman. Attributing mode collapse in the fine-tuning of large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, volume 2, 2024.

Vishakh Padmakumar and He He. Does writing with language models reduce content diversity? In *The Twelfth International Conference on Learning Representations*, 2024.

Haihui Pan, Yuzhong Hong, Shaoke Lv, Junwei Bao, Hongfei Jiang, and Yang Song. Quality-constrained entropy maximization policy optimization for LLM diversity. *arXiv preprint arXiv:2602.15894*, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

Matthew Renze. The effect of sampling temperature on problem solving in large language models. In *Findings of the association for computational linguistics: EMNLP 2024*, pp. 7346–7356, 2024.

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

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

Shivanshu Shekhar, Shreyas Singh, and Tong Zhang. See-dpo: Self entropy enhanced direct preference optimization. *arXiv preprint arXiv:2411.04712*, 2024.

Stewart Slocum, Asher Parker-Sartori, and Dylan Hadfield-Menell. Diverse preference learning for capabilities and alignment. In *International Conference on Learning Representations*, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Qwen Team. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

The Terminal-Bench Team. Terminal-bench: A benchmark for ai agents in terminal environments, Apr 2025b. URL https://github.com/laude-institute/terminal-bench.

Chaoqi Wang, Yibo Jiang, Chenghao Yang, Han Liu, and Yuxin Chen. Beyond reverse kl: Generalizing direct preference optimization with diverse divergence constraints. In *International Conference on Learning Representations*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, LINGMING ZHANG, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ULblO61XZ0.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025.

Bojian Xiong, Yikun Lei, Xikai Liu, Shaowei Zhang, Pengyun Zhu, Yan Liu, Yongqi Leng, Ling Shi, Meizhi Zhong, Yurong Zhang, et al. Think-search-patch: A retrieval-augmented reasoning framework for repository-level code repair. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1555–1566, 2025a.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. *arXiv preprint arXiv:2312.11456*, 2023.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. In *ICLR*, 2025b.

Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks. In *Proceedings of the ACM Web Conference 2024*, pp. 1362–1373, 2024a.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is dpo superior to ppo for llm alignment? a comprehensive study. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 54983–54998, 2024b.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Jiahao Yu, Zelei Cheng, Xian Wu, and Xinyu Xing. Gpo: Learning from critical steps to improve llm reasoning. *Advances in Neural Information Processing Systems*, 38:1052–1086, 2026.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025a.

Liang Zeng, Yongcong Li, Yuzhen Xiao, Changshi Li, Chris Yuhao Liu, Rui Yan, Tianwen Wei, Jujie He, Xuchen Song, Yang Liu, et al. Skywork-swe: Unveiling data scaling laws for software engineering in llms. *arXiv preprint arXiv:2506.19290*, 2025b.

Kechi Zhang, Ge Li, Yihong Dong, Jingjing Xu, Jun Zhang, Jing Su, Yongfei Liu, and Zhi Jin. Codedpo: Aligning code models with self generated and verified source code. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15854–15871, 2025.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024a.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024b. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

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

A Proof of Lemma 3.1

Proof. The soft optimality identities give

$$\begin{aligned}
&\sum_{h=1}^{H} \zeta \log \frac{\pi^*(a_h|s_h)}{\pi_{\text{ref}}(a_h|s_h)^{\gamma/\zeta}} \\
&= \sum_{h=1}^{H} [Q_h^*(s_h, a_h) - V_h^*(s_h) - \gamma \log \pi_{\text{ref}}(a_h|s_h)] \\
&= \sum_{h=1}^{H} \left[r(s_h, a_h) + \mathbb{E}_{s' \sim \mathbb{P}_h(\cdot|s_h, a_h)} V_{h+1}^*(s') - V_h^*(s_h) \right] \\
&= \sum_{h=1}^{H} r(s_h, a_h) - V_1^*(s_1) - \Delta^*(\tau),
\end{aligned} \tag{9}$$

where the last equality adds and subtracts the realized next-state values, uses $V_{H+1}^* = 0$, and applies the definition of $\Delta^*(\tau)$ in Lemma 3.1. Rearranging gives the stated decomposition. Since $s_{h+1} \sim \mathbb{P}_h(\cdot|s_h, a_h)$, each summand in $\Delta^*(\tau)$ has zero conditional mean, so $\mathbb{E}[\Delta^*(\tau)] = 0$. For deterministic transitions, the realized and expected next-state values coincide at every step, giving $\Delta^*(\tau) = 0$ pathwise. Subtracting the identities for τ^+ and τ^- cancels their shared $V_1^*(s_1)$; under deterministic SWE transitions, substitution into the BT model gives the loss in Theorem 3.2. □

B EntroPO-KTO loss

Alternatively, the policy can be learned by optimizing a KTO-style objective. Consider a dataset $\mathcal{D}$ where each trajectory τ is labeled as either "desirable" ($y = +$) or "undesirable" ($y = -$). Our implementation uses the deterministic-transition specialization, with implicit reward:

$$r_\theta(x, y) = \sum_{h=1}^{H} \log \frac{\pi_\theta(a_h|s_h)}{\pi_{\text{ref}}(a_h|s_h)^{\gamma/\zeta}} \tag{10}$$

The EntroPO-KTO loss is designed to maximize the margin between desirable and undesirable trajectories relative to a reference point z_0. The objective is defined as:

$$\mathcal{L}_{\text{EntroPO-KTO}}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\lambda_y - v(x, y) \right] \tag{11}$$

where the value function $v(x, y)$ is given by:

$$v(x, y) = \begin{cases} \lambda_+ \sigma\Big(\zeta\big(r_\theta(x, y) - z_0\big)\Big) & \text{if } y = +, \\ \lambda_- \sigma\Big(\zeta\big(z_0 - r_\theta(x, y)\big)\Big) & \text{if } y = -. \end{cases} \tag{12}$$

Here, λ_+ and λ_- are hyperparameters that weight the loss for desirable and undesirable examples, respectively. The reference point z_0 is defined as:

$$z_0 = \mathbb{E}_{x\sim\mathcal{D}, \tau\sim\pi(\cdot|x)} \left[\sum_{h=1}^{H} \left(-H(\pi(\cdot|s_h)) + \frac{\gamma}{\zeta} H(\pi(\cdot|s_h), \pi_{\text{ref}}(\cdot|s_h)) \right) \right]. \tag{13}$$

Although z_0 depends on the evolving policy π, each gradient step treats it as a detached constant, following Ethayarajh et al. (2024).

C Diversity Analysis

C.1 Equivalence of Conditional Probability

A response y structurally consists of the model's generated tokens $y' = (y'_1, \ldots, y'_H)$ (code) and the resulting observation $o = (o_1, \ldots, o_{H-1})$ (execution feedback). By

the chain rule, the probability of the trajectory is $P(y|x) = \tilde{\pi}(y'|x)P(o|x,y') = \tilde{\pi}(y'_1|x)P(o_1|x,y'_1)\dots\tilde{\pi}(y'_H|x,y'_1,o_1,\dots,y'_{H-1},o_{H-1})$. In the context of software engineering tasks, the environment exhibits *deterministic transition dynamics*. Thus, $P(o|x,y')$ is equal to 1. Optimizing the trajectory probability $P(y|x)$ is therefore mathematically equivalent to optimizing the generative policy $\tilde{\pi}(y'|x)$.

C.2 Derivation of the Optimal Policy

Through policy re-parametrization, we seek to maximize the following objective:

$$\mathcal{J}(\tilde{\pi}) = \sum_y \tilde{\pi}(y|x)r(x,y) - \zeta \sum_y \tilde{\pi}(y|x)\log\tilde{\pi}(y|x) + \gamma \sum_y \tilde{\pi}(y|x)\log\tilde{\pi}_{\text{ref}}(y|x). \tag{14}$$

We construct the Lagrangian with a constraint $\sum_y \tilde{\pi}(y|x) = 1$:

$$\mathcal{L}(\tilde{\pi},\eta) = \mathcal{J}(\tilde{\pi}) + \eta\left(1 - \sum_y \tilde{\pi}(y|x)\right). \tag{15}$$

Taking the derivative w.r.t $\tilde{\pi}(y|x)$ and setting to 0:

$$\frac{\partial\mathcal{L}}{\partial\tilde{\pi}(y|x)} = r(x,y) - \zeta(1+\log\tilde{\pi}(y|x)) + \gamma\log\tilde{\pi}_{\text{ref}}(y|x) - \eta = 0. \tag{16}$$

Solving for $\tilde{\pi}(y|x)$:

$$\log\tilde{\pi}(y|x) = \frac{1}{\zeta}\left(r(x,y) + \gamma\log\tilde{\pi}_{\text{ref}}(y|x) - \eta - \zeta\right). \tag{17}$$

Exponentiating and absorbing constants into a partition function $Z(x)$ yields the result:

$$\tilde{\pi}^*(y|x) = \frac{1}{Z(x)}\tilde{\pi}_{\text{ref}}(y|x)^{\frac{\gamma}{\zeta}}\exp\left(\frac{r(x,y)}{\zeta}\right). \tag{18}$$

Formally, the optimization above is a *constrained* problem: because the environment has deterministic (fixed) transition dynamics, only trajectories consistent with those dynamics are feasible. Consequently, for a given state (or context) x, the response/trajectory y cannot range over the entire trajectory space; instead, $\tilde{\pi}(y \mid x)$ must assign zero probability to any y that violates the environment transition constraint.

Crucially, both $\tilde{\pi}$ and the reference policy $\tilde{\pi}_{\text{ref}}$ are learned in the *same* environment, and therefore share the *same support* over valid trajectories (*i.e.,* $\tilde{\pi}_{\text{ref}}(y \mid x) = 0$ whenever y is infeasible). Since $\tilde{\pi}_{\text{ref}}(y \mid x)$ enters multiplicatively in Eqn. 18, it implicitly restricts the solution to the feasible set. Hence, the derived optimal policy $\tilde{\pi}^*$ remains unchanged in form and is valid under the environment-induced constraint.

C.3 Proof of Proposition 3.4

We now analyze how the hyperparameters γ and ζ influence the diversity of the generated responses.

Specifically, we consider the log-ratio of the optimal policy $\tilde{\pi}^*$ to the reference policy π_{ref}. Using the result from Lemma 3.3:

$$\log\frac{\tilde{\pi}^*(y|x)}{\tilde{\pi}_{\text{ref}}(y|x)} = \log\left(\frac{\tilde{\pi}_{\text{ref}}(y|x)^{\gamma/\zeta}\exp(r(x,y)/\zeta)}{Z(x)\tilde{\pi}_{\text{ref}}(y|x)}\right) \tag{19}$$

$$= \frac{r(x,y)}{\zeta} + \left(\frac{\gamma}{\zeta} - 1\right)\log\tilde{\pi}_{\text{ref}}(y|x) - \log Z(x). \tag{20}$$

This equation describes the "update" or "boost" applied to the reference policy to obtain the optimal policy.

Comparison with Standard RLHF/DPO. Standard DPO corresponds to the case where the KL-divergence penalty is used alone, which implies $\gamma_D = \zeta_D = \zeta$ (*i.e.,* $\gamma_D/\zeta_D = 1$). In this case, the update rule simplifies to:

$$\log \frac{\tilde{\pi}^*_{\text{DPO}}(y|x)}{\tilde{\pi}_{\text{ref}}(y|x)} = \frac{r(x,y)}{\zeta} - \log Z_{\text{DPO}}(x). \tag{21}$$

Here, the relative boost depends *only* on the reward $r(x, y)$. If two responses y_1 and y_2 have equal reward $r(x, y_1) = r(x, y_2)$ (*i.e.,* In RLVR, the correct answer usually receives the same reward), they receive the exact same log-probability boost, regardless of their initial likelihood under $\tilde{\pi}_{\text{ref}}$.

ENTROPO **Regime ($\gamma < \zeta$).** In ENTROPO, we configure $\gamma < \zeta$, such that the coefficient $(\frac{\gamma}{\zeta} - 1)$ is strictly negative. Let us compare two responses, y_{common} and y_{rare}, which are both "correct" (equal high reward r) but have different probabilities under the reference model:

$$\tilde{\pi}_{\text{ref}}(y_{\text{common}}|x) > \tilde{\pi}_{\text{ref}}(y_{\text{rare}}|x) \implies \log \tilde{\pi}_{\text{ref}}(y_{\text{common}}|x) > \log \tilde{\pi}_{\text{ref}}(y_{\text{rare}}|x). \tag{22}$$

Since $\log \tilde{\pi}_{\text{ref}}$ is negative and $(\frac{\gamma}{\zeta} - 1)$ is negative, the term $(\frac{\gamma}{\zeta} - 1) \log \tilde{\pi}_{\text{ref}}(y|x)$ is positive and monotonically decreasing with respect to $\tilde{\pi}_{\text{ref}}(y|x)$.

Calculating the difference in the update magnitude $\Delta(y) = \log \tilde{\pi}^*(y|x) - \log \tilde{\pi}_{\text{ref}}(y|x)$ for the two responses:

$$\Delta(y_{\text{rare}}) - \Delta(y_{\text{common}}) = \left(\frac{\gamma}{\zeta} - 1\right) [\log \tilde{\pi}_{\text{ref}}(y_{\text{rare}}|x) - \log \tilde{\pi}_{\text{ref}}(y_{\text{common}}|x)] > 0. \tag{23}$$

The inequality holds because we are multiplying a negative coefficient by a negative difference. This implies $\Delta(y_{\text{rare}}) > \Delta(y_{\text{common}})$.

Therefore, under ENTROPO, the rare correct response receives a *larger* relative boost than the common correct response. This mechanism counteracts the "rich-get-richer" dynamic of standard RLHF, thereby preserving and promoting diversity in the distribution of correct solutions.

C.4 PROOF OF THEOREM 3.6

Proof. **(1) Correctness inequality.**

Let $P_K = \text{Acc}_x(\tilde{\pi}^*_K)$ for $K \in \{E, D\}$ and $p_0 = \sum_{y \in C} p_y \in (0, 1)$. By definition, $\bar{w}_S$ is the reference-probability-weighted average of the power-law multiplier $p_y^{\alpha-1}$ on S:

$$\bar{w}_S = \sum_{y \in S} \frac{p_y}{\sum_{z \in S} p_z} p_y^{\alpha-1}, \qquad S \in \{C, I\}. \tag{24}$$

Consequently,

$$\sum_{y \in C} p_y^\alpha = p_0 \bar{w}_C, \qquad \sum_{y \in I} p_y^\alpha = (1 - p_0)\bar{w}_I. \tag{25}$$

Applying Lemma 3.3 to the binary reward gives

$$\frac{P_E}{1 - P_E} = e^{1/\zeta} \frac{p_0}{1 - p_0} \frac{\bar{w}_C}{\bar{w}_I}, \qquad \frac{P_D}{1 - P_D} = e^{1/\gamma} \frac{p_0}{1 - p_0}, \tag{26}$$

where DPO uses $\gamma = \zeta$. Since $p \mapsto p/(1 - p)$ is strictly increasing on $(0, 1)$,

$$P_E \geq P_D \iff \frac{\bar{w}_C}{\bar{w}_I} \geq \exp\left(\frac{1}{\gamma} - \frac{1}{\zeta}\right). \tag{27}$$

(2) Diversity inequality.

Let $C = \mathcal{Y}^+ = \{y_1, \ldots, y_m\}$ be the set of correct trajectories with $r(x, y_j) = 1, \forall j \in [m]$. From Lemma 3.3 (Eqn. 18), the optimal policy is

$$\tilde{\pi}^*(y \mid x) = \frac{1}{Z(x)} \tilde{\pi}_{\text{ref}}(y \mid x)^{\gamma/\zeta} \exp\big(r(x, y)/\zeta\big). \tag{28}$$

We derive the conditional distributions over C, writing $q_K(y) = q_{\tilde{\pi}^*_K}(y \mid x)$ for $K \in \{E, D\}$. Since all trajectories in $\mathcal{Y}^+$ share the same reward $r = 1$, the conditional distribution of the optimal policy restricted to $\mathcal{Y}^+$ is:

$$q(y_j) \;=\; \frac{\tilde{\pi}_{\text{ref}}(y_j \mid x)^\alpha \cdot \exp(1/\zeta)}{\sum_{k=1}^m \tilde{\pi}_{\text{ref}}(y_k \mid x)^\alpha \cdot \exp(1/\zeta)} \;=\; \frac{\tilde{\pi}_{\text{ref}}(y_j \mid x)^\alpha}{\sum_{k=1}^m \tilde{\pi}_{\text{ref}}(y_k \mid x)^\alpha}, \qquad \alpha \in (0, 1], \tag{29}$$

where $\alpha = \gamma/\zeta$. For standard DPO, $\alpha_D = \gamma_D/\zeta_D = 1$, giving $q_D(y_j) = \tilde{\pi}_{\text{ref}}(y_j \mid x)/\sum_k \tilde{\pi}_{\text{ref}}(y_k \mid x)$. For ENTROPO, $\alpha_E = \gamma_E/\zeta_E < 1$.

Let us denote the reference conditional probabilities as $\mathbf{q} = (q_1, \ldots, q_m)$ where $q_j = \tilde{\pi}_{\text{ref}}(y_j \mid x)/\sum_k \tilde{\pi}_{\text{ref}}(y_k \mid x)$. Note that $q_D = \mathbf{q}$ and $q_E(y_j) = \frac{q_j^\alpha}{Z_\alpha}$ where $Z_\alpha = \sum_{k=1}^m q_k^\alpha$.

Define $\phi_\alpha : \Delta^{m-1} \to \Delta^{m-1}$ by $[\phi_\alpha(\mathbf{q})]_j = q_j^\alpha/Z_\alpha$, so $q_E = \phi_\alpha(q_D)$ with $\alpha = \alpha_E$. We define the entropy of the escort distribution as a function of α:

$$S(\alpha) \;=\; H(\phi_\alpha(\mathbf{q})). \tag{30}$$

Note that $S(1) = H(q_D)$ and $\lim_{\alpha \to 0^+} S(\alpha) = \log m$ (since ϕ_0 is the uniform distribution). Our goal is to show $S(\alpha_E) \geq S(1)$, i.e., $H(q_E) \geq H(q_D)$.

From the entropy expansion,

$$\begin{aligned} S(\alpha) &= -\sum_{j=1}^m \frac{q_j^\alpha}{Z_\alpha}\Big(\alpha \log q_j - \log Z_\alpha\Big) \\ &= \frac{\alpha}{Z_\alpha} \sum_j q_j^\alpha \log(1/q_j) + \log Z_\alpha \\ &= \alpha\, T(\alpha) + \log Z_\alpha, \end{aligned} \tag{31}$$

where $T(\alpha) = \frac{1}{Z_\alpha}\sum_j q_j^\alpha \log(1/q_j) = \mathbb{E}_{q_\alpha}[\log(1/q_j)]$ is the expected negative log-reference-probability under $q_\alpha = \phi_\alpha(\mathbf{q})$.

Since $\frac{d}{d\alpha} \log Z_\alpha = \frac{1}{Z_\alpha}\sum_j q_j^\alpha \log q_j = -T(\alpha)$, we obtain:

$$S'(\alpha) = T(\alpha) + \alpha\, T'(\alpha) - T(\alpha) = \alpha\, T'(\alpha). \tag{32}$$

Computing $T'(\alpha)$: since $T(\alpha) = \sum_j \frac{q_j^\alpha}{Z_\alpha} \log(1/q_j)$, we have

$$\begin{aligned} T'(\alpha) &= \sum_j \frac{d}{d\alpha}\Big[\frac{q_j^\alpha}{Z_\alpha}\Big] \log(1/q_j) \\ &= \sum_j \frac{q_j^\alpha (\log q_j) Z_\alpha - q_j^\alpha \cdot Z'_\alpha}{Z_\alpha^2} \log(1/q_j) \\ &= -\sum_j q_\alpha(y_j)[\log(1/q_j)]^2 + [T(\alpha)]^2 \\ &= -\mathrm{Var}_{q_\alpha}\big[\log(1/q_j)\big]. \end{aligned} \tag{33}$$

Therefore,

$$S'(\alpha) = -\alpha\, \mathrm{Var}_{q_\alpha}\big[\log(1/q_j)\big]. \tag{34}$$

For any non-uniform distribution $\mathbf{q}$ (i.e., not all q_j are equal), the random variable $\log(1/q_j)$ takes at least two distinct values under any distribution q_α that assigns positive mass to all atoms. Since $q_\alpha(y_j) = q_j^\alpha/Z_\alpha > 0$ for all j whenever $q_j > 0$, the variance is strictly positive:

$$\mathrm{Var}_{q_\alpha}\big[\log(1/q_j)\big] > 0 \qquad \text{for all } \alpha \in (0, 1]. \tag{35}$$

Combined with Eqn. 34 and the fact that $\alpha > 0$, we conclude:

$$S'(\alpha) < 0 \qquad \text{for all } \alpha \in (0, 1]. \tag{36}$$

That is, $S(\alpha)$ is *strictly decreasing* on $(0, 1]$.

Since $\alpha_E = \gamma_E/\zeta_E < 1 = \alpha_D$ and S is strictly decreasing, we immediately obtain:

$$H(q_E) = S(\alpha_E) > S(\alpha_D) = S(1) = H(q_D). \tag{37}$$

This establishes $H(q_E) > H(q_D)$ whenever q_D is non-uniform.

Tightness (equality condition). When q_D is uniform ($q_j = 1/m$ for all j), we have $\log(1/q_j) = \log m$ for every atom, so $\mathrm{Var}_{q_\alpha}[\log(1/q_j)] = 0$ for all α. In this case $S'(\alpha) = 0$, meaning S is constant: $H(q_E) = H(q_D) = \log m$. This is the unique case of equality in the theorem. Since $H(q_K) = H_x^+(\widetilde{\pi}_K^*)$, this proves part (ii). □

D Experiment Details

In this section, we provide additional experimental details of EntroPO.

D.1 Dataset Details

Training Data. For SWEBench experiments, we use R2E-Gym-subset (Jain et al., 2025) for preference learning and verifier training. R2E-Gym-subset provides executable oracle feedback for trajectory-level supervision and has no repository overlap with our SWEBench evaluation sets. For LiveCodeBench experiments, we follow the multi-turn code-generation setup of Chen et al. (2026) and train on TACO-Train tasks. For TerminalBench experiments, we sample training tasks from Endless Terminals (Gandhi et al., 2026).

SWE Preference-Data Construction. Starting from the 4,578 R2E-Gym-subset tasks, we use GLM-4.5 to generate four independent trajectories per task. The executable oracle labels each trajectory by whether its final patch passes the task's unit tests. For M-DPO and EntroPO-DPO, we retain tasks with at least one passing and one failing trajectory and randomly select exactly one trajectory from each group, yielding one preference pair per eligible task. The same selected pairs are used for both methods. Tasks with four passing or four failing trajectories do not contribute a pair. The resulting dataset $\mathcal{D}_{\text{pref}}$ contains 3,123 pairs. Due to computation budget constraints, we do not enumerate all possible preference pairs. The KTO variants retain trajectory-level binary desirable/undesirable labels and do not construct preference pairs.

Code-Generation and Terminal Data. We use Cobalt's cleaned TACO-Train split (Chen et al., 2026), containing 5,853 tasks after reserving 250 development tasks from 6,103 cleaned problems. Each problem has at least eight tests; the four easiest tests form the public split for interaction, while the remaining tests are hidden. For terminal training, we sample 2,500 tasks from the released Endless Terminals dataset (Gandhi et al., 2026), whose generation pipeline validates the initial container state and retains tasks solved by at least one of 16 o3 attempts. For both training datasets, we use the same trajectory generator and four-trajectories-per-task collection protocol as in the SWE experiments. Final executable outcomes supply binary labels; DPO variants use one randomly selected passing/failing pair per eligible task, while KTO variants use pointwise labels.

Evaluation Data. We evaluate EntroPO on the complete official test sets for both SWEBench-Verified and SWEBench-Lite. For LiveCodeBench, we evaluate on tasks collected between January and April 2025, where the agent can use public-test feedback during interaction and the final submission is scored using hidden tests. Following Chen et al. (2026), public-test examples and their explanations are removed from the problem statements. For TerminalBench, we evaluate on TerminalBench 2.0 tasks. Detailed statistics for all datasets are provided in Table 3.

Oracle Feedback. For SWEBench, the oracle checks whether the final patch passes the task's test suite. For code-generation tasks, the agent receives public-test feedback during interaction, while held-out tests score the completed solution. For terminal tasks, the final-state tests determine success after the rollout ends. These executable outcomes are distinct from the learned verifier scores used by the SWEBench hybrid selector in §D.4. The selector does not use benchmark oracle outcomes to choose the submitted patch.

Table 3: **Dataset Statistics**: The table presents the number of training and test samples for each dataset, along with the source of the dataset.

Dataset	# Train Tasks	# Test Tasks	Source
R2E-Gym-subset	4,578	-	https://huggingface.co/datasets/R2E-Gym/R2E-Gym-Subset
SWE-bench Verified	-	500	https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified
SWE-bench Lite	-	300	https://huggingface.co/datasets/princeton-nlp/SWE-bench_Lite
TACO-Train	5,853	-	https://huggingface.co/datasets/osunlp/TACO-Cobalt
LiveCodeBench	-	175	https://huggingface.co/datasets/livecodebench/code_generation_lite
Endless Terminals	2,500	-	https://huggingface.co/datasets/obiwan96/endless-terminals
TerminalBench 2.0	-	89	https://huggingface.co/datasets/harborframework/terminal-bench-2.0

D.2 SCAFFOLD DETAILS

For SWEBENCH, we use the standard R2E scaffold (Jain et al., 2025) for every method in our controlled experiments and for our official leaderboard submissions. The controlled comparisons use the same inference limits across methods: up to 200 environment interactions per rollout and a maximum sequence length of 131,072 tokens. The official SWEBENCH-VERIFIED result reported by the Qwen team for the original Qwen3-Coder model uses OpenHands with a substantially larger context window and a higher interaction limit, thus the results may differ from our controlled experiments. The leaderboard comparison evaluates complete agent systems under their respective configurations, while our controlled experiments compare methods under a common R2E setup.

The R2E scaffold equips the agent with four tools: `file_editor` for file editing, `execute_bash` for running bash commands, `search` for file and code retrieval, and `finish` to conclude the task. The complete R2E system prompt, including the functionality and parameters of each tool, is provided below.

System Prompt of the R2E Scaffold

You are a programming agent who is provided a GitHub issue and repository bash environment and is tasked to solve certain tasks (e.g., file localization, testcase generation, code repair, and editing, etc) to resolve the issue.
We have access to the following functions:
— BEGIN FUNCTION #1: `file_editor` —
Description: Custom editing tool for viewing, creating, and editing files.
- State is persistent across command calls and discussions with the user
- If path is a file, view displays the result of applying `cat -n`. If path is a directory, view lists of non-hidden files and directories up to 2 levels deep
- The create command cannot be used if the specified path already exists as a file
- If a command generates a long output, it will be truncated and marked with `<response clipped>`
- The `undo_edit` command will revert the last edit made to the file at path

Notes for using the `str_replace` command:
- The `old_str` parameter should match EXACTLY one or more consecutive lines from the original file. Be mindful of whitespaces!
- If the `old_str` parameter is not unique in the file, the replacement will not be performed. Make sure to include enough context in `old_str` to make it unique
- The `new_str` parameter should contain the edited lines that should replace the `old_str`

Parameters:

1. `command` (string, required)
   Allowed values: `[view, create, str_replace, insert, undo_edit]`.
   The command to run.
2. `path` (string, required)
   Absolute path to file or directory, e.g. `/testbed/file.py` or `/testbed`.
3. `file_text` (string, optional)
   Required for the `create` command. Contains the content of the file to be created.
4. `old_str` (string, optional)
   Required for the `str_replace` command. The exact string in the path to replace.

5. `new_str` (string, optional)
   - Optional for the `str_replace` command to specify the replacement string.
   - Required for the `insert` command to specify the string to insert.
6. `insert_line` (integer, optional)
   Required for the `insert` command. The `new_str` will be inserted after the line number specified here.
7. `view_range` (array, optional)
   - Optional for the `view` command (when path is a file).
   - If provided, specifies the line range to view, e.g. `[11, 12]` shows lines 11 and 12.
   - `[start_line, -1]` will show all lines from `start_line` to the end of file.
8. `concise` (boolean, optional)
   - Optional for the `view` command.
   - Displays a concise skeletal view of the file. If set to `False`, it displays the full content in the specified `view_range`.

— END FUNCTION #1 —
— BEGIN FUNCTION #2: `execute_bash` —
Description: Execute a bash command in the terminal.
Behavior notes:
- If a command may run indefinitely (long-running), consider running it in the background and redirecting output, e.g. `python3 app.py > server.log 2>&1 &`.
- If the bash command returns exit code -1, it means the process is still running. The assistant may:
  – Call this function again with the command as an empty string ("") to retrieve additional logs.
  – Send more input to STDIN of the running process by calling this function again with the command set to the text input.
  – Send `command="ctrl+c"` to interrupt the currently running process.
- If the command times out, it will be interrupted (SIGINT). The assistant may then retry or do further steps if needed.

Parameters:

1. `cmd` (string, required)
   The bash command (and optional arguments) to execute.
   - Can be empty ("") to retrieve more logs if the process is still running.
   - Can be "ctrl+c" to interrupt the running process.

— END FUNCTION #2 —
— BEGIN FUNCTION #3: `search` —
Description: Search for a term in a directory or a single file.
- If path is a directory (or unspecified, default is `.`), it recursively searches all non-hidden files and directories for the search term.
- If path points to a file, it runs a `grep -n` in that file to show line numbers matching the search term.
- If more than 100 files match in a directory search, results are truncated, and the tool will inform you to narrow your search.
- If no matches are found, it will inform you as well.

Parameters:

1. `search_term` (string, required)
   The term or string to search for in files.
2. `path` (string, optional)
   The file or directory to search in. Defaults to `.` if not specified.

— END FUNCTION #3 —
— BEGIN FUNCTION #4: `finish` —
Description: Finish the interaction once the task is complete or if no further progress can be made.
Behavior notes:

- The submit command finalizes your output.

Parameters:

1. `command` (string, required)
   Currently allowed value: `[submit]`.
2. `result` (string, optional)
   The result text or final message to submit. Defaults to an empty string if not provided.

— END FUNCTION #4 —

If you choose to call a function ONLY reply in the following format with NO suffix:

```
<function=example_function_name>
<parameter=example_parameter_1>value_1</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
```

<IMPORTANT> Reminder:

- Function calls MUST follow the specified format, start with `<function=` and end with `</function>`
- Required parameters MUST be specified
- Only call one function at a time
- VERY IMPORTANT: Each response must include both reasoning (as natural text) and function call (in the above format) to solve the task.

For LIVECODEBENCH, we follow the Cobalt multi-turn code-generation scaffold (Chen et al., 2026). Each example contains a programming problem, an initial pool of candidate responses, and public test cases. The scaffold extracts the Python code block from each response, evaluates it with the Cobalt evaluator on the public tests, and appends the first failing feedback to the conversation. At the next turn, the model is asked to repair the previous full solution rather than emit a patch. If all public tests pass, the scaffold still asks the model to reason about hidden edge cases before producing the next candidate. The interaction is therefore a public-test-feedback repair loop rather than a repository-editing scaffold.

Prompt for Cobalt Scaffold

```
System:
You are a Python programming assistant. Solve the coding
↪  problem under
a restricted execution environment: read-only filesystem, no
↪  shell or
process control, and only safe Python modules.

User:
<programming problem statement>

Assistant:
<complete Python solution in a Markdown code block>

User after public-test execution:
Analyze the previous solution and the execution feedback.
↪  Identify the
buggy code region, explain the fix, and output a complete
↪  corrected
Python program in one new code block.
```

```
Execution feedback:
<first failing public-test feedback, or a message that public
 ↪  tests pass
and hidden edge cases should still be considered>
```

For TERMINALBENCH, we follow the Endless Terminals terminal-agent scaffold (Gandhi et al., 2026). Each task is initialized from a task directory containing a task description, a container image, initial-state tests, final-state tests, and a container definition. The scaffold creates one isolated container environment per rollout and first checks the initial state. At each step, the model must either emit exactly one shell command or terminate with `done`. The command is executed in the corresponding container, and the scaffold returns whether it succeeded together with stdout, stderr, and the exit code. After at most 200 actions, or once the agent emits `done`, the scaffold runs the final tests and assigns reward 1 only if the tests pass.

Prompt for Endless Terminals Scaffold

```
System:
You are a Linux terminal agent using a single-command
 ↪  interface.
For each turn, think privately, then output exactly one
 ↪  action:
either one shell command inside command tags, or done.
Use non-interactive commands, inspect outputs before
 ↪  proceeding,
write files explicitly when needed, and verify the solution
 ↪  before
terminating.

User:
<terminal task description>

Assistant:
<think>plan for the next terminal action</think>
<cmd>THE_SINGLE_SHELL_COMMAND</cmd>

User after command execution:
Command executed successfully/failed.
Output: <stdout/stderr, possibly truncated>
(exit_code=<0 or 1>)
```

D.3 RUNNING ENVIRONMENT

Our offline preference-learning implementation relies on the LLaMA-Factory (Zheng et al., 2024b) for model training and SGLang (Zheng et al., 2024a) for efficient inference deployment. These experiments are performed on a server configured with four 32-core AMD EPYC 7702 CPUs, 8 NVIDIA H100 (80GB) GPUs, and 4 NVIDIA A100 (40GB) GPUs. The H100 GPUs are dedicated to the primary training and inference workloads, while the A100 GPUs provide supplementary computational support during inference. The separate GRPO training configuration is described in §D.6.1.

For training runtime, GLM-4.5-Air requires approximately 80 hours for one fine-tuning run, while the other models require less than 48 hours per fine-tuning run. Evaluation wall-clock time depends on system service load and CPU/disk I/O because the evaluation pipeline creates Docker environments. On our machine, one evaluation run takes approximately 8 hours for SWEBENCH-VERIFIED, 5 hours for SWEBENCH-LITE, 4 hours for LIVECODEBENCH, and 4 hours for TERMINALBENCH.

D.4 Training and Inference Details

Training Pipeline. We optimize each backbone on the fixed offline data described in §D.1, using the corresponding EntroPO-DPO or EntroPO-KTO objective. Training reuses the collected trajectories without generating new online rollouts at each optimization step.

Training Hyperparameters. We configure our training process as follows. For LoRA, we set `lora_rank=8`, `lora_alpha=16`, and apply it to all available modules (`lora_target=all`). For the GLM-4.5-Air model, we use 4-bit QLoRA quantization. Across all models, we use a context length of 18,000 tokens, a learning rate of 1e-5, and a warmup ratio of 0.1. Training trajectories exceeding 18,000 tokens are truncated to this limit to fit within the available GPU memory. The batch size is set to 4 for GLM-4.5-Air and 16 for all other models. Preference optimization is trained for a single epoch using these settings.

Inference Parameters. During inference, each rollout is permitted a maximum of 200 environment interaction steps and a total generation length of up to 131,072 tokens. We use a sampling temperature of 0.7, `top_k` of 20, and `top_p` of 0.8 for all models, which is the recommended setting from Qwen3 model card.

Hybrid Trajectory Selection. We use the hybrid trajectory selector only for SWEBench, because the official SWEBench protocol requires submitting a single final patch even when multiple rollouts are generated. For LiveCodeBench and TerminalBench, we report pass@N over the raw rollout set and do not apply this selector. On SWEBench, the selector employs a multi-stage filtering process to identify the best trajectory from the generated candidates $\{\tau^{(n)}\}_{n=1}^{N}$. The verifier $p_\phi(x, \tau) \in [0, 1]$ is a learned scorer trained on $\mathcal{D}_{\text{pref}}$ with supervised learning for one epoch with the same settings as the preference-optimized models. We select the Qwen3-Coder-30B as our verifier model. The process is as follows:

1. **Finished score (model-free)**: We first discard any trajectories that are truncated due to exceeding the step or token limits: keep only $\mathbb{1}(\text{finished}(\tau) = 1)$.
2. **Regression test score (model-free)**: Next, we filter out trajectories that fail the regression tests generated by the R2E-Gym framework: $\mathbb{1}(\text{regress_free}(\tau) = 1)$.
3. **Verifier probability (model-based)**: We then apply a verifier model and remove trajectories with a probability below a threshold of $\eta = 0.01$. We choose a conservative threshold because we empirically observe that many valid solutions do not receive high probability scores, but scores below 0.01 are a strong indicator of a flawed trajectory. We keep $\{\tau \in \mathcal{S} : p_\phi(x, \tau) \geq \eta\}$.

If any filtering step results in an empty set of candidates, we revert to the candidate pool from the previous step.

After filtering, we apply a final model-free heuristic for selection (Jin et al., 2024). We select the trajectory with the **most** environment interaction steps as it could indicate more comprehensive reasoning, the addition of more thorough test cases, or the consideration of more corner cases before submitting a final patch.

For the verifier-only ablation in Figure 3, we select the candidate with the highest verifier score, $\arg\max_{n \in \{1,\ldots,N\}} p_\phi(x, \tau^{(n)})$.

D.5 Baseline Implementations

Original Model. We first evaluate the unaligned backbone model under the same scaffold and inference budget as EntroPO. This baseline measures the ability already present in the base model before task-specific preference optimization.

Higher Temperature. We also evaluate the performance of the backbone model with a higher temperature of 1.0. This baseline measures whether simply increasing the temperature can replicate the benefits of EntroPO.

SFT Model. For SFT baseline, we fine-tune the backbone on successful trajectories from the same training tasks using the standard next-token prediction objective. The SFT baseline uses the same response masking, context length, LoRA configuration, and inference budget as the preference-

optimized models. It is included to separate the effect of imitation-style trajectory learning from preference optimization.

M-DPO and M-KTO. Our primary preference-optimization baselines are the multi-turn extensions of DPO and KTO (Xiong et al., 2025b). They use the same trajectory pools and labels as EntroPO. For M-DPO, we construct preference pairs from trajectories of the same task and optimize the standard multi-turn log-ratio objective. For M-KTO, we use the same desirable/undesirable trajectory labels as EntroPO-KTO and optimize the standard KTO-style pointwise objective. We use the default LLaMA-Factory preference hyperparameters for these baselines: the KL-penalty coefficient is $\beta = 0.1$ (`pref_beta`), and the KTO-specific weights are $\lambda_+ = 1.0$ (`kto_chosen_weight`) and $\lambda_- = 1.0$ (`kto_rejected_weight`).

DivPO. We adapt DivPO (Lanchantin et al., 2025) to multi-turn tool-use trajectories by changing preference-pair construction while retaining the M-DPO objective. For the main DivPO results in Table 1, GLM-4.5 generates 4 trajectories per task over the same 3,123 eligible SWE training tasks. The training task set, scaffold, and inference settings remain the same.

Each candidate receives a binary quality label from the executable oracle. We score it using the backbone model being fine-tuned: for example, Qwen3-Coder scores the candidates used to train Qwen3-Coder. The score is the mean log-probability per model-generated token over the complete trajectory:

$$\bar{\ell}(\tau) = \frac{1}{T_\tau} \sum_{h=1}^{H_\tau} \sum_{j=1}^{|a_h|} \log \pi_{\text{base}}(a_{h,j} \mid s_h, a_{h,<j}), \qquad T_\tau = \sum_{h=1}^{H_\tau} |a_h|.$$

Here, H_τ is the number of turns, $a_{h,j}$ is the jth generated token at turn h, and T_τ counts generated tokens across all turns. Prompt tokens and environment observations are part of the conditioning context but are excluded from the scored token count. Each eligible task contributes one pair. The passing trajectory with the lowest $\bar{\ell}(\tau)$ is chosen, and the failing trajectory with the highest $\bar{\ell}(\tau)$ is rejected. Normalization by generated tokens avoids the direct length dependence of a summed log-probability.

D.6 Additional Baseline Comparisons

D.6.1 Comparison with Online RL

We compare EntroPO-KTO with GRPO (Shao et al., 2024) on SWEBench-Verified using Qwen3-Coder-30B-A3B-Instruct. Both methods use the R2E scaffold, the same Docker execution environments, and the same evaluation budget. GRPO uses binary unit-test outcomes as verifiable rewards and is trained on R2E-Gym-subset tasks through Uni-Agent(Ding et al., 2026). At each optimization step, 8 prompts receive 8 sampled trajectories each, yielding 64 new online rollouts. Table 4 lists the GRPO training configuration. Its 100-turn limit and shorter prompt/response limits apply during online training due to computational constraints. In the evaluation phase, we use the same limit of 200 environment interactions and 131,072 tokens described in §D.4.

Table 4: **GRPO training configuration.** These limits apply to online training only while evaluation settings are shared with EntroPO-KTO.

Setting	Value
Training tasks	4,578 R2E-Gym-subset tasks
Reward	Binary resolved/unresolved unit-test outcome
Prompts / samples per prompt	8 / 8 (64 online rollouts per optimization step)
Maximum agent turns	100, with a fresh sandbox for each rollout
Prompt / response token limits	4,096 / 20,152
Temperature / top-p / top-k	0.7 / 0.8 / 20
Actor learning rate	10^{-6}
KL coefficient	10^{-3}
PPO mini-batch size	8
Tensor parallelism	8

The GRPO run is allocated 32 nodes with 8 × 80GB H100 GPUs each for 3 days, corresponding to approximately 18,432 H100-hours. ENTROPO-KTO uses approximately 192 H100-hours for offline trajectory collection and 384 H100-hours for optimization, yielding a total of 576 H100-hours. The comparison therefore includes the cost of collecting the offline data. GRPO generates new trajectories at each optimization step, whereas ENTROPO-KTO reuses its fixed dataset. Other GRPO hyperparameters follow the Uni-Agent defaults for Qwen3-Coder.

Table 5: **Online GRPO and offline ENTROPO-KTO on SWEBENCH-VERIFIED.** Compute includes trajectory collection and training. Pass@1 measures single-rollout accuracy, and pass@16 measures the fraction of tasks with at least one correct solution among 16 raw candidates before selection.

Method	Online rollouts / step	H100-hours	pass@1	pass@16
GRPO	64	∼18,432	46.4	71.8
ENTROPO-KTO	0	∼576	**51.6**	**79.2**

As shown in Table 5, ENTROPO-KTO improves pass@1 by 5.2% and pass@16 by 7.4% over GRPO. It achieves these gains using approximately $1/32$ of the collection and training compute in this configuration. The result supports offline preference optimization as a practical option for improving SWE-agent performance under limited training resources. The comparison uses matched evaluation settings, while each training algorithm uses the configuration specified above.

D.6.2 SINGLE-TURN BASELINE COMPARISON

We evaluate ENTROPO-KTO on BIG-Bench Hard (BBH) (Suzgun et al., 2022) using Gemma-3-27b-it to compare against methods designed for single-turn preference learning. We follow GPO's sub-task-level BBH split (Yu et al., 2026), with 3,261 training problems before filtering and 3,250 evaluation problems, then sample 8 responses per training problem and retain problems with both correct and incorrect responses. All fine-tuned methods use the same backbone, training data, training budget, and decoding configuration. We compare DPO and KTO with two representative diversity-preserving objectives, f-DPO (Wang et al., 2024) and offline QEMPO (Pan et al., 2026). Pairwise methods use preference pairs, while KTO and ENTROPO-KTO use pointwise correctness labels.

Table 6: **Single-turn comparison on BBH with Gemma-3-27b-it.** Scores are percentages. Pass@1 measures single-response accuracy, pass@16 measures whether at least one of 16 responses is correct, and mean@16 measures average correctness across the 16 responses.

Method	Feedback	pass@1	pass@16	mean@16
Base	–	64.29	77.24	63.52
DPO	Pairwise	67.36	81.35	68.24
KTO	Pointwise	66.35	80.57	67.30
f-DPO	Pairwise	67.02	82.53	68.95
QEMPO (offline)	Pairwise	66.85	81.95	66.29
ENTROPO-KTO	Pointwise	**69.95**	**86.35**	**70.25**

Table 6 shows that ENTROPO-KTO achieves the highest score on all three metrics. Relative to the strongest baseline for each metric, it improves pass@1 by 2.59%, pass@16 by 3.82%, and mean@16 by 1.30%. Thus, higher success within 16 responses is accompanied by improved per-response accuracy, extending the empirical benefit of ENTROPO beyond multi-turn SWE tasks to this single-turn reasoning setting.

D.7 ADDITIONAL EXPERIMENTS

Qwen3 Results. We report the separate Qwen3 results from the main benchmark suite in Table 7. As observed from the table, it shares the same trends as the main results in Table 1.

Comparison to the Official SWEBENCH Leaderboard. In Table 8, we compare our best-performing SWEBENCH model against submissions on the official SWEBENCH leaderboard. Our entries are the scores returned by the official SWEBENCH team's oracle for our submitted trajectories, whereas Table 1

Table 7: **Additional Qwen3 results.**

Model	Origin	Temp. = 1	SFT	M-DPO	M-KTO	DivPO	EntroPO-DPO	EntroPO-KTO
SWEBench-Verified								
Qwen3	1.7 (± 0.1)	1.5 (± 0.3)	2.4 (± 0.3)	3.8 (± 0.2)	4.1 (± 0.2)	3.7 (± 0.1)	4.9 (± 0.7)	**5.2 (± 0.4)**
Qwen3-TTS	5.1 (± 0.6)	4.5 (± 0.7)	6.3 (± 0.9)	8.0 (± 0.4)	8.3 (± 0.4)	7.8 (± 0.2)	11.1 (± 0.3)	**11.5 (± 0.6)**
SWEBench-Lite								
Qwen3	1.2 (± 0.3)	1.0 (± 0.3)	1.3 (± 0.5)	3.6 (± 0.2)	3.7 (± 0.3)	3.4 (± 0.4)	4.7 (± 0.5)	**4.8 (± 0.4)**
Qwen3-TTS	4.2 (± 1.0)	3.6 (± 1.0)	4.4 (± 0.6)	7.6 (± 0.4)	8.0 (± 0.3)	7.6 (± 0.3)	**10.4 (± 0.4)**	10.0 (± 0.8)
LiveCodeBench								
Qwen3	2.5 (± 0.7)	1.7 (± 0.5)	2.7 (± 0.3)	7.0 (± 0.3)	6.5 (± 0.2)	6.3 (± 0.5)	8.2 (± 0.7)	**8.6 (± 0.5)**
Qwen3-TTS	4.8 (± 0.7)	4.0 (± 0.5)	5.9 (± 0.5)	14.5 (± 0.7)	13.9 (± 0.3)	13.0 (± 0.7)	**18.5 (± 0.7)**	17.9 (± 0.7)
TerminalBench								
Qwen3	0.0 (± 0.0)	0.0 (± 0.0)	0.0 (± 0.0)	1.5 (± 0.5)	1.1 (± 0.0)	1.1 (± 0.0)	**2.2 (± 0.9)**	1.9 (± 0.5)
Qwen3-TTS	2.2 (± 0.9)	1.5 (± 0.5)	3.0 (± 1.1)	4.9 (± 1.4)	4.5 (± 0.9)	4.1 (± 1.1)	**6.0 (± 1.4)**	5.6 (± 0.9)

Table 8: **Resolve rate (pass@1, %) compared to representative SWEBench leaderboard submissions.** Our entries use the R2E scaffold, and we report the best single run for comparability. Budgets and scaffolds may vary across submissions.

Submission	Model	Model Size	SWEBench-Verified ↓	SWEBench-Lite ↓
Closed-Weight Models				
Refact.ai	Claude3.7/o3/o4-mini	-	74.4	60.0
SWE-agent	Claude 4 Sonnet	-	66.6	56.7
SWE-agent	Claude 3.7 Sonnet	-	62.4	48.0
Open-Weight Models				
OpenHands	Qwen3-Coder	480B	69.6	-
OpenHands	Kimi K2	1T	65.4	-
OpenHands	GLM-4.5	355B	64.2	-
DeepSWE-TTS	Qwen3	32B	58.8	-
Skywork-SWE-TTS	Qwen2.5	32B	47	-
CodeFuse-CGM	Qwen2.5	72B	-	44.0
KGCompass	DeepSeek V3	671B	-	36.7
SWE-fixer	Qwen2.5	72B	24.7	32.8
Moatless	Deepseek V3	671B	-	30.7
Our Methods				
EntroPO-KTO-TTS	Qwen3-Coder	30B	60.4	49.7
EntroPO-KTO	Qwen3-Coder	30B	52.2	45.0

reports averages from our local oracle, so the scores can differ between the two evaluations. Our results are highly competitive, particularly among open-weight models. On SWEBench-Verified, our 30B parameter EntroPO-KTO-TTS model achieves a 60.4% resolve rate, surpassed only by models with over 10x more parameters (*e.g.,* >350B). On SWEBench-Lite, the same model sets a new state-of-the-art for open-weight models at 49.7%, with our non-TTS version securing the second-highest rank.

Crucially, EntroPO-KTO-TTS outperforms other TTS-based submissions like DeepSWE-TTS (Luo et al., 2025) and Skywork-SWE-TTS (Zeng et al., 2025b). This highlights the effectiveness of our entropy-regularized training for improving final task accuracy with TTS. Note that DeepSWE-TTS is online RL-based, showing that our entropy-preserving offline preference learning can be more effective than online RL. Compared to closed-weight models, our results are competitive with top-tier models like Claude 3.7 Sonnet, demonstrating that EntroPO can significantly narrow the performance gap with commercial models.

Impact of TTS Budget on KTO. To investigate whether our approach can benefit KTO, we conduct experiments on the Qwen3-Coder model with the EntroPO-KTO and M-KTO (Xiong et al., 2025b) models. We test the Qwen3-Coder model on SWEBench-Verified and SWEBench-Lite with different number of parallel rollouts (N) and compare its performance to EntroPO-KTO, M-KTO, and SFT. As shown in Figure 4, EntroPO-KTO consistently outperforms M-KTO and SFT. When $N = 1$, EntroPO-KTO can outperform both SFT and M-KTO, showing its advantage can be

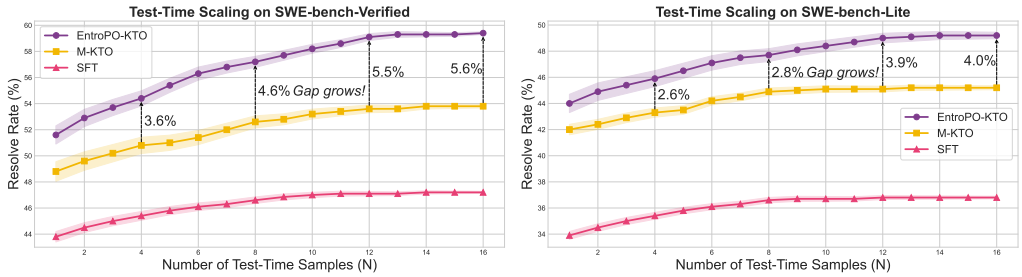


Figure 4: **The Impact of TTS Budget on Task Performance.** Performance of EntroPO-KTO, M-KTO, and SFT on SWEBench-Verified (left) and SWEBench-Lite (right) as the number of parallel rollouts (N) increases. EntroPO's entropy regularization consistently yields better scaling.

independent of TTS. As N increases, the performance gap between EntroPO-KTO and M-KTO widens, similar to our findings in Figure 2.

Impact of Temperature on Performance. To investigate whether further increasing sampling temperature can replicate the benefits of our entropy-preserving approach, we evaluate the M-KTO model under varying temperatures. We test the Qwen3-Coder model on SWEBench-Verified with temperatures ranging from 0.5 to 1.9 and compare its performance to EntroPO-KTO, which use a fixed temperature of 0.7. As shown in Figure 5, increasing the temperature provides no performance benefit. In fact, performance begins to decline beyond a temperature of 0.9, as excessive sampling randomness undermines the precision required for SWE tasks involving tool use and code editing. These results demonstrate that merely increasing temperature is not a substitute for principled entropy regularization, as it fails to match the performance gains achieved by EntroPO.

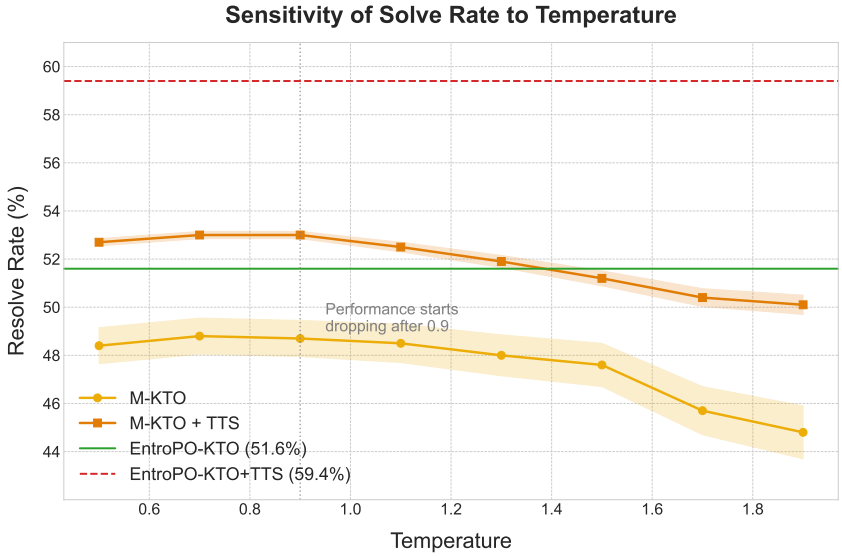


Figure 5: **Impact of Temperature on Performance.** Performance of multi-turn KTO and KTO+TTS on SWEBench-Verified with varying temperature, compared to EntroPO with a fixed temperature of 0.7. Increasing temperature fails to match the performance of EntroPO and degrades performance past 0.9.

D.8 Further Analysis

Per-rollout accuracy and success within 16 rollouts. We evaluate how entropy regularization affects the accuracy of individual rollouts on SWEBench-Verified. Using Qwen3-Coder-30B, we evaluate 16 raw candidates per task before applying any selector. We report mean@16, the average success rate across these candidates, to assess per-rollout accuracy, and pass@16, the fraction of

Table 9: **Raw-candidate accuracy on** SWEBENCH-VERIFIED**.** Results (%) use Qwen3-Coder-30B with 16 raw rollouts before applying any selector. Mean@16 measures average candidate correctness; pass@16 measures the fraction of tasks with at least one correct candidate.

Model	mean@16	pass@16
Base model	37.4	63.6
M-KTO	48.5	70.8
ENTROPO-KTO	**51.9**	**79.2**

instances with at least one correct candidate that passes the benchmark tests. Mean@16 estimates the correctness probability Acc_x averaged over tasks, while pass@16 measures success within the rollout budget, as described in §4.1. As shown in Table 9, ENTROPO-KTO achieves a mean@16 of 51.9%, exceeding M-KTO's 48.5%, while pass@16 increases from 70.8% to 79.2%. Thus, for this model and training configuration, both average per-rollout accuracy and success within 16 rollouts improve.

Inference efficiency. We examine whether entropy regularization leads to longer trajectories by encouraging erroneous actions or redundant exploration. Using Qwen3-Coder-30B on the 500 SWEBENCH-VERIFIED tasks, we compare M-KTO and ENTROPO-KTO with 16 rollouts per task and the same hybrid selector. As shown in Table 10, ENTROPO-KTO reduces average tool calls per rollout from 87 to 84 and average tokens from 86.4K to 83.6K, corresponding to reductions of approximately 3.4% and 3.2%, respectively. At the same time, the resolve rate increases from 53.8% to 59.4%, with 297 tasks solved compared with 269 for M-KTO.

Table 10: **Inference usage on** SWEBENCH-VERIFIED **at** $N = 16$**.** Tool calls and tokens are averaged over rollouts. Resolved tasks and resolve rates refer to the final patches chosen by the hybrid selector.

Method	Tool calls / rollout	Tokens / rollout	Resolved tasks	Resolve rate
M-KTO	87	86.4K	269 / 500	53.8%
ENTROPO-KTO	84	83.6K	297 / 500	59.4%

These results show that higher task accuracy does not come at the expense of longer or more costly rollouts in this setting: ENTROPO-KTO solves more tasks while using fewer tool calls and tokens per rollout on average. Thus, reasonable entropy regularization improves the final resolve rate without excessive exploration that lengthens trajectories. The combination of lower rollout cost and higher task success reduces tool-call and token usage per resolved task by approximately 12.5% and 12.4%, respectively. Thus, ENTROPO uses the available rollout budget more productively while maintaining efficient individual trajectories.

Multi-hunk task analysis. To understand where diversity-preserving TTS helps, we inspect the reference patches for SWEBENCH-VERIFIED tasks solved by Qwen3-Coder ENTROPO-KTO with $N = 16$ but unresolved with $N = 1$. We find that 72% of these additionally solved tasks have reference patches containing multiple diff hunks. Such repairs involve coordinated changes across multiple code regions, introducing choices about both where to edit and how to combine the edits into a complete fix. This finding suggests that diversity-preserving TTS is useful for tasks requiring coordinated edits, where exploring alternative localization and patching strategies can help find a complete repair after a single rollout fails.

Qualitative case study. We examine a logging-buffer bug in `pytest` using the trajectories in Table 11. Clearing captured logs can leave the live records buffer inconsistent with the records returned for individual test phases. One successful ENTROPO trajectory repairs the write path by synchronizing stale phase aliases after clearing, while another repairs the read path by retrieving the live handler buffer for the call phase. The shown M-DPO trajectory instead forces all phase buffers to share one empty list, losing the intended distinction among setup, call, and teardown records. This case illustrates how individual patches succeed or fail.

Table 11: **Illustrative patching example on `pytest-dev__pytest-10051`.** Two successful EntroPO trajectories repair the write path and the read path, respectively while the shown M-DPO trajectory fails.

Trajectory	Main hypothesis	Patch behavior
EntroPO trajectory A	State synchronization in the write path.	Preserve phase semantics after `clear()` by redirecting only stale aliases from the old records list to the current handler records.
EntroPO trajectory B	Access semantics in the read path.	Leave `clear()` unchanged and make `get_records("call")` resolve the live handler buffer for the call phase.
M-DPO failed trajectory	Symptom-level clearing pattern.	Reset all phase buffers to the same empty list, which may satisfy the local symptom but loses the distinction between setup, call, and teardown records.

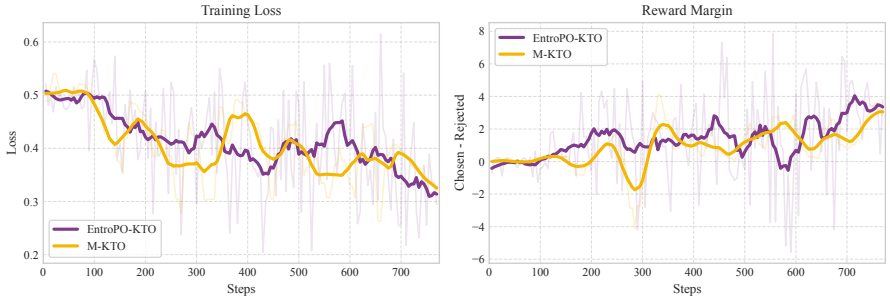


Figure 6: **Convergence dynamics.** Training loss and reward-margin curves for EntroPO-KTO and M-KTO on Qwen3-Coder-30B-A3B-Instruct. Solid lines show smoothed trends, while lighter lines show logged or interpolated values for visualization.

Convergence dynamics. Finally, we examine whether entropy-preserving preference optimization makes training harder. The EntroPO objective changes the weighting inside the same log-ratio computation as the standard multi-turn objective and does not require extra rollouts, additional models, or extra forward/backward passes. To compare optimization dynamics, we plot training curves for EntroPO-KTO and M-KTO under the same Qwen3-Coder-30B-A3B-Instruct setup in Figure 6. For KTO, the reward margin follows the LLaMA-Factory logged metric: the batch mean reward of desirable examples minus the batch mean reward of undesirable examples, rather than a one-to-one paired chosen–rejected margin. Both the training loss and reward-margin trends remain stable, and EntroPO-KTO converges at a comparable speed to M-KTO. Thus, the empirical gains of EntroPO do not come at the cost of slower or less stable optimization.

E Discussions and Limitations

We discuss limitations and future directions of EntroPO. First, our TTS experiments are limited to $N = 16$ parallel rollouts due to budget constraints. As our results in Figure 2 suggest that performance gains scale with N, exploring this behavior with a larger number of rollouts could more conclusively demonstrate the benefits of diversity preservation. Second, our end-to-end rollout strategy could be enhanced by incorporating more sophisticated search techniques like Tree-of-Thought (Yao et al., 2023) or solution merging methods (Luong et al., 2025) to explore the solution space more effectively.

Third, due to computational constraints, the current EntroPO implementation adopts an offline preference learning setting. However, the proposed entropy-regularized preference learning framework

naturally extends to online iterative settings, as in Xiong et al. (2023); Guo et al. (2024). Future work could investigate online versions of ENTROPO-DPO and ENTROPO-KTO, where the explicit diversity incentive may be crucially important for preventing mode collapse during repeated rounds of data collection and policy updates.