Carnegie Mellon University

Tail-Likelihood Reinforcement Learning

Shrinivas Ramasubramanian[1] **Daman Arora**[1] **Fahim Tajwar**[1] **Guanning Zeng**[1]
Qingyang Wu[4] **Zhongzhu Zhou**[4] **Chenfeng Xu**[4]
Haiwen Feng[2,3] **Yuda Song**[1] **Aarti Singh**[1] **Ruslan Salakhutdinov**[1]
J. Andrew Bagnell[5,1] **Jeff Schneider**[1,†] **Andrea Zanette**[1,†]

[1]Carnegie Mellon University [2]University of California, Berkeley [3]Impossible, Inc.
[4]Together AI [5]Aurora Innovation

{shrinivr, jeff4, azanette}@andrew.cmu.edu

Abstract

Reinforcement learning typically optimizes average reward. For generative policies, the average can hide an important distinction: two policies can achieve the same mean reward while having very different chances of producing a rare but high-reward rollout. This matters as sampling increases during training and inference, since its benefit depends on retaining probability mass on high-reward outcomes. We propose to optimize this coverage directly. Rather than considering only expected reward, we consider all of its upper tails: for each reward threshold, how likely is the policy to exceed it? This turns a continuous reward into a family of binary success events. We introduce **Tail-Likelihood Reinforcement Learning (TailRL)**, which maximizes the log-probability of exceeding a randomly chosen reward threshold. Its gradient gives more weight to rare, high-reward rollouts and can be interpreted as a mixture of Best-of-(k) gradients. TailRL requires only a simple modification to the advantage function, making it compatible with existing reinforcement learning pipelines. Across object localization, maze navigation, GUI grounding, and code optimization, TailRL leverages rare high-reward training samples to avoid suboptimal solutions and yields models that benefit more from additional samples at inference time.

1 Introduction

Reinforcement learning (RL) typically optimizes the expected reward of a policy (Williams, 1992; Shao et al., 2024; Ahmadian et al., 2024; Yu et al., 2025; Zheng et al., 2025). For generative policies, however, the mean reward does not fully characterize performance: two policies with similar mean reward can have very different probabilities of producing rare but exceptionally good rollouts. This distinction matters whenever additional samples can be drawn, both during training and at inference time.

Recent work has exposed this problem directly: standard RL training can progressively lose coverage over rare, high-reward rollouts, often visible as a degradation in Best-of-k performance (Cui et al., 2025; Yue et al., 2025; Wu et al., 2025; Dang et al., 2025; Kirk et al., 2024). Once these rollouts become sufficiently unlikely, they are rarely sampled again, making further policy improvement increasingly difficult. The same loss of coverage limits inference-time scaling: a policy may perform well with a single sample while gaining little from drawing many (Walder and Karkhanis, 2025; Chen et al., 2025b; Yang et al., 2025b). Thus, optimizing only the mean reward can discard information about the upper tail of the reward distribution that is crucial for both training and inference scaling.

For binary rewards, MaxRL (Tajwar et al., 2026) offers one way to address this problem. Rather than maximizing the probability of success, MaxRL maximizes its log-probability, placing greater emphasis on rare successes. Its gradient decomposes into a harmonic mixture of Pass@k gradients, directly connecting likelihood maximization to a mixture of Pass@k objectives.

How should this principle extend to **continuous rewards**? Our starting point is simple: every reward threshold defines a binary event. Given an input x, a rollout $z \sim \pi_\theta(\cdot \mid x)$, and a threshold τ, we ask

[†]Joint advising.
Project website, code, and other assets: https://zanette-labs.github.io/TailRL-website/

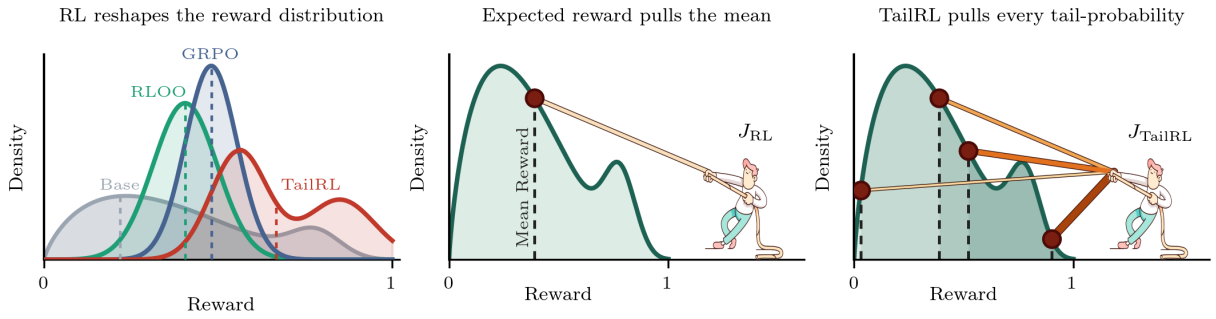

Figure 1: Expected reward pulls one rope; TailRL pulls them all. Left: RL post-training reshapes the reward distribution of the base policy. Expected-reward methods (GRPO, RLOO) shift the distribution and sharpen it around the mean, while TailRL shifts it further and grows a heavy high-reward tail. Middle: the expected-reward objective J_{RL} improves the distribution through a single handle, its mean. Right: TailRL increases the tail probability $p_\theta(x, \tau)$ across reward thresholds τ, placing greater emphasis on rarer, higher-reward outcomes.

whether the reward exceeds τ. The corresponding *tail probability* is

$$p_\theta(x, \tau) := \Pr_{z \sim \pi_\theta(\cdot|x)}\big(r(x, z) > \tau\big). \tag{1}$$

A continuous reward can be viewed as a family of binary success events, one for every threshold τ. This perspective leads to **Tail-Likelihood Reinforcement Learning (TailRL)**. For rewards in $[0, 1]$, TailRL maximizes the expected log-likelihood of exceeding a uniformly chosen reward threshold:

$$J_{\mathrm{TailRL}}(\theta; x) = \int_0^1 \log p_\theta(x, \tau), d\tau. \tag{2}$$

Unlike expected-reward maximization, which acts on a single summary of the reward distribution, TailRL explicitly optimizes upper-tail probabilities across reward levels. We show that the gradient of TailRL decomposes into a harmonic mixture of Best-of-k gradients (Section 3.1), directly connecting the training objective to coverage of high-reward rollouts and inference-time scaling. MaxRL emerges as the binary-reward special case. Despite its different objective, TailRL admits a simple critic-free policy-gradient estimator that differs from standard RL only in its advantage calculation, allowing it to be implemented by swapping the advantage function in an existing RL pipeline. Across object localization (Section 6.1), maze navigation (Section 6.2), GUI grounding (Section 6.3), and code optimization (Section 6.4), TailRL leverages rare high-reward samples during training to avoid suboptimal solutions and produces policies that benefit more from additional samples at inference time.

Our contributions are as follows.

1. **A likelihood objective for continuous rewards.** TailRL maximizes the log-probability of exceeding a uniformly drawn reward threshold, which weights each reward level by the inverse of how often the policy reaches it. It introduces no threshold or weighting hyperparameter and reduces exactly to MaxRL for binary rewards.

2. **Alignment with inference-time scaling.** The gradient of TailRL decomposes harmonically over Best-of-k gradients (Section 3.1),

$$\nabla_\theta J_{TailRL}(\theta; x) = \sum_{k=1}^{\infty} \frac{1}{k} \nabla_\theta \text{ Best-of-}k(\theta; x),$$

   so TailRL improves Best-of-k at every inference budget without choosing one in advance.

3. **An unbiased finite-rollout estimator.** A group of N rollouts defines an order-N truncation of TailRL that interpolates from expected reward ($N = 1$) to the population objective ($N \to \infty$), and we derive closed-form rollout weights that estimate its gradient without bias (Section 4). Unlike REINFORCE, where more rollouts only reduce variance, here the rollout budget selects the objective being optimized.

4. **Strong empirical results.** TailRL matches supervised objectives that observe the ground truth on object localization (Section 6.1), learns from initial policies with 0.01% success where expected-reward baselines fail on maze navigation (Section 6.2), matches RLOO's Pass@1024 on GUI grounding with

128–256× fewer inference rollouts (Section 6.3), and reaches a 7.7× Best-of-1024 speedup on code optimization where GRPO and RLOO collapse onto copying the input (Section 6.4).

2 Preliminaries

We primarily focus on optimizing continuous reward reinforcement learning, where for each input x, a rollout z is generated by the policy π_θ. A rollout may be a generated response, program, or trajectory. A deterministic reward function provides a scalar feedback for an input rollout pair $r(x, z) \in [0, 1]$. Appendix D treats a more general bounded reward range.

Training ultimately averages over $x \sim \rho$. To keep the notation light, we write each objective for a fixed input x and leave the outer average over inputs implicit. We define the policy's score-function as $S(x, z) := \nabla_\theta \log \pi_\theta(z \mid x)$. Standard reinforcement learning maximizes expected reward and the score-function identity (Williams, 1992) gives its policy gradient as follows,

$$\begin{aligned} J_{\mathrm{RL}}(\theta; x) &:= \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[r(x, z)] \,, \\ \nabla_\theta J_{\mathrm{RL}}(\theta; x) &= \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[r(x, z)S(x, z)] \,. \end{aligned} \tag{3}$$

Thus, the policy gradient is a weighted combination of the score-function and the weights are determined by the rollout's reward. During training, critic-free rollout based policy gradient methods draw N independent rollouts $z_1, \ldots, z_N \sim \pi_\theta(\cdot \mid x)$. Critic-free methods such as GRPO (Shao et al., 2024) and RLOO (Ahmadian et al., 2024) compute a finite rollout estimate of Eq. (3). PKPO (Walder and Karkhanis, 2025) optimizes Pass@k and Best-of-k for binary and continuous rewards using a similar finite-rollout based estimation framework. Section 4 shows that TailRL uses the same template and changes only how these advantages are computed, and Appendix F places the three advantage functions side by side.

2.1 Inference-Time Selection

At deployment, additional inference compute can be used to sample several rollouts and select the one with the highest reward. For k independent rollouts, this performance is measured by probability of at-least one success (Pass@k) when rewards are binary and expected maximum reward among (Best-of-k) when rewards are continuous:

$$\text{Pass@}k(\theta; x) := 1 - \big(\Pr_{z\sim\pi_\theta(\cdot|x)}\big(r(x, z) = 0\big)\big)^k \,, \qquad \text{(binary reward)} \tag{4}$$

$$\text{Best-of-}k(\theta; x) := \mathbb{E}_{\{z_i\}_{i=1}^k\sim\pi_\theta(\cdot|x)}\left[\max_{1\le i\le k} r(x, z_i)\right] \,, \qquad \text{(continuous reward)} \tag{5}$$

At $k = 1$, both reduce to the mean reward, $J_{\mathrm{RL}}(\theta; x)$. Appendix G gives the estimators we typically use to compute both empirically. Both Pass@k and Best-of-k are non-decreasing in k, and as $k \to \infty$ each approaches the maximum reward in their support.

Unlike mean reward, Best-of-k depends strongly on the upper part of the reward distribution. Two policies with the same mean can scale differently with additional training or inference compute if one assigns more probability to high-reward rollouts.

2.2 Policy as a Generative Model of Rewards

A policy and reward function together define a distribution over rewards. For an input x, the policy samples a rollout z and the reward function assigns its reward:

$$z \sim \pi_\theta(\cdot \mid x), \qquad r = r(x, z) \in [0, 1]. \tag{6}$$

Because the reward function is deterministic, all randomness in the reward comes from the policy. The induced reward distribution for any event A ($A \subseteq [0, 1]$) over the support of rewards is defined as,

$$\Pr_{z\sim\pi_\theta(\cdot|x)}(r(x, z) \in A) = \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[\mathbb{1}_{\{r(x,z)\in A\}}] \tag{7}$$

In this view, the policy is a generative model over rewards, and training manipulates the probability mass over reward values. Expected reward reinforcement learning uses only the mean reward to shape this reward distribution.

Different policies can have the same mean reward while assigning very different probabilities to high-reward outcomes. This becomes visible in their difference in Best-of-k and Pass@k performance. This motivates objectives that act on the entirety of the reward distribution rather than only its mean. We begin with binary rewards, where the distribution is completely determined by a single success probability.

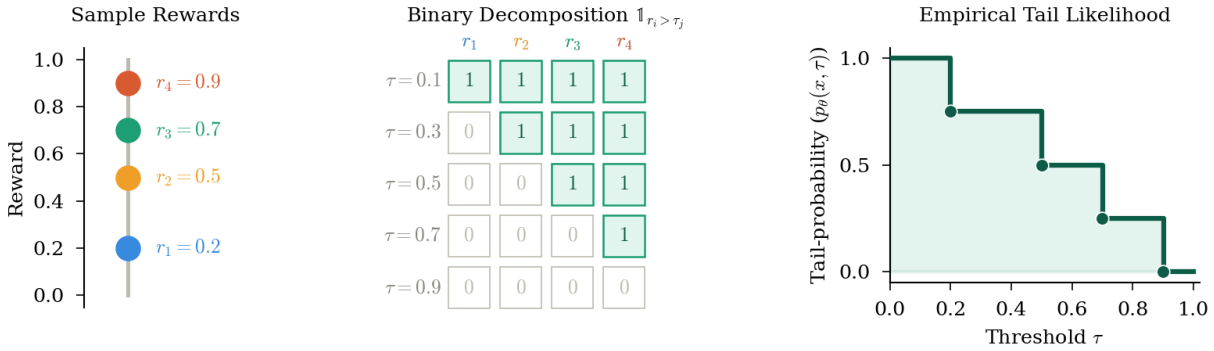

Figure 2: A continuous reward decomposes into threshold events. A rollout with reward r_i clears every threshold below r_i. Across a group of rollouts, these binary outcomes estimate the tail-probability $p_\theta(x, \tau)$ at every reward threshold. TailRL maximizes the average log-probability along this curve.

2.3 MaxRL for Binary Rewards

For a binary reward $r(x, z) \in \{0, 1\}$, expected reward equals the probability of success.

$$q_\theta(x) := \Pr_{z \sim \pi_\theta(\cdot|x)}(r(x, z) = 1)\,. \tag{8}$$

The reward distribution is therefore a Bernoulli distribution over success and failures. The reward distribution is fully determined by $q_\theta(x)$. Standard reinforcement learning maximizes $q_\theta(x)$, whereas MaxRL (Tajwar et al., 2026) maximizes $J_{\mathrm{MaxRL}}(\theta; x) = \log q_\theta(x)$ and its gradient is,

$$\nabla_\theta J_{\mathrm{MaxRL}}(\theta; x) = \frac{1}{q_\theta(x)} \mathbb{E}_{z \sim \pi_\theta(\cdot|x)} \left[\mathbb{1}_{\{r(x,z)=1\}} S(x, z) \right]. \tag{9}$$

The factor $1/q_\theta(x)$ gives greater weight to inputs on which success is rare. MaxRL also connects to inference-time sampling. $\nabla_\theta J_{\mathrm{MaxRL}}(\theta; x)$ decomposes as a harmonic mixture of Pass@k gradients.

$$\nabla_\theta J_{\mathrm{MaxRL}}(\theta; x) = \sum_{k=1}^{\infty} \frac{1}{k} \nabla_\theta \mathrm{Pass@}k(\theta; x) \tag{10}$$

3 Tail-Likelihood Reinforcement Learning

We motivate TailRL by expressing expected reward in terms of the policy's upper-tail probabilities. For rewards in $[0, 1]$, define the tail probability at threshold τ as

$$p_\theta(x, \tau) := \Pr_{z \sim \pi_\theta(\cdot|x)} \left(r(x, z) > \tau \right). \tag{11}$$

The expected reward is exactly the area under this tail-probability curve:

$$J_{\mathrm{RL}}(\theta; x) = \mathbb{E}[r(x, z)] = \int_0^1 p_\theta(x, \tau)\, d\tau. \tag{12}$$

Tail-Likelihood Reinforcement Learning (TailRL) instead applies the MaxRL likelihood principle at every reward threshold. It maximizes the average log-probability of exceeding a uniformly sampled threshold:

$$J_{\mathrm{TailRL}}(\theta; x) := \int_0^1 \log p_\theta(x, \tau)\, d\tau = \mathbb{E}_{\tau \sim \mathrm{Unif}[0,1]} \left[\log p_\theta(x, \tau) \right]. \tag{13}$$

Equivalently, expected-reward RL aggregates tail probabilities arithmetically, whereas TailRL aggregates them geometrically. This makes small tail probabilities more influential, as seen directly from its gradient

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta; x) = \int_0^1 \frac{1}{p_\theta(x, \tau)} \nabla_\theta p_\theta(x, \tau)\, d\tau. \tag{14}$$

The higher the reward threshold, the smaller the tail-probability and higher is the weight for the gradient. Thus reward thresholds that are difficult to reach receive larger weight during optimization.

3.1 Harmonic Decomposition over Best-of-k

Although TailRL is defined through reward thresholds, it has an exact interpretation in terms of inference-time sampling. Recall that Best-of-k$(\theta; x)$ is the expected maximum reward among k independent rollouts from $\pi_\theta(\cdot \mid x)$.

Theorem 1 (Best-of-k decomposition)**.** *The TailRL objective decomposes as*

$$J_{\mathrm{TailRL}}(\theta; x) = \sum_{k=1}^{\infty} \frac{\text{Best-of-k}(\theta; x) - 1}{k}, \tag{15}$$

and, under the regularity conditions of Lemma 6, its gradient satisfies

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta; x) = \sum_{k=1}^{\infty} \frac{1}{k} \nabla_\theta \text{Best-of-k}(\theta; x). \tag{16}$$

Thus, TailRL combines Best-of-k learning signals across all inference budgets. The harmonic weights $1/k$ arise automatically from the logarithm, so TailRL does not require selecting a target inference budget in advance. The proof is given in Appendix C.3.3.

3.2 Recovery of MaxRL for Binary Rewards

For binary rewards, every nontrivial reward threshold defines the same success event. Let

$$q_\theta(x) := \Pr_{z \sim \pi_\theta(\cdot|x)} \left(r(x, z) = 1 \right)$$

denote the probability of success. For every $\tau \in [0, 1)$, $p_\theta(x, \tau) = q_\theta(x)$, and therefore

$$J_{\mathrm{TailRL}}(\theta; x) = \int_0^1 \log q_\theta(x) \, d\tau = \log q_\theta(x) = J_{\mathrm{MaxRL}}(\theta; x). \tag{17}$$

Moreover, for binary rewards Best-of-k coincides with Pass@k, so the harmonic Best-of-k decomposition above reduces exactly to the harmonic Pass@k decomposition of MaxRL. Thus, TailRL directly generalizes MaxRL from binary to continuous rewards.

3.3 Probabilistic Interpretation of TailRL

A direct extension of MaxRL would define only $r(x, z) = 1$ as success and treat every lower reward as the same failure. Exact success may be rare or unattainable. Lowering the success threshold makes the event more common, but still treats all rewards on either side of the threshold as equivalent. Once a rollout crosses the threshold, the objective has no preference for improving it further. The resulting policy can perform well at the chosen threshold while remaining poor at higher reward levels (Appendix H.4). We therefore need a likelihood event that preserves the continuous reward signal.

A continuous reward defines such an event at every reward threshold. For $\tau \in [0, 1)$, define the tail-probability

$$p_\theta(x, \tau) := \Pr_{z \sim \pi_\theta(\cdot|x)} (r(x, z) > \tau) \, . \tag{18}$$

Independent quality audit. We combine the tail events by assigning each reward threshold an independent rollout. The audit passes only if every rollout clears its assigned threshold. Independent rollouts are necessary: reusing one rollout would collapse the nested events to the hardest threshold.

For an audit with L equally spaced thresholds, let $\tau_\ell = (\ell - 1)/L$ and draw $z_1, \ldots, z_L \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid x)$. Define the event that the policy passes the audit as

$$E_L := \bigcap_{\ell=1}^{L} \{ r(x, z_\ell) > \tau_\ell \} \, . \tag{19}$$

Because the rollouts are independent, the audit probability factorizes:

$$\Pr_{z_1 \ldots z_L \sim \pi_\theta(\cdot|x)} (E_L \mid x) = \prod_{\ell=1}^{L} p_\theta(x, \tau_\ell). \tag{20}$$

The number of thresholds controls only the resolution of the audit. Requiring invariance to the change in scale of the objective due to increasing resolution of the audit, while agreeing with ordinary log-likelihood when $L = 1$, uniquely gives the normalization $1/L$:

$$\frac{1}{L} \log \Pr_\theta(E_L \mid x) = \frac{1}{L} \sum_{\ell=1}^{L} \log p_\theta(x, \tau_\ell). \tag{21}$$

Each midpoint τ_ℓ represents an interval of width $1/L$. The right-hand side of Eq. (21) is therefore a Riemann sum over the reward range. Letting $L \to \infty$ gives

$$J_{\mathrm{TailRL}}(\theta; x) := \lim_{L \to \infty} \frac{1}{L} \log \Pr_\theta(E_L \mid x) = \int_0^1 \log p_\theta(x, \tau)\, d\tau, \tag{22}$$

We call Eq. (13) the population-level TailRL objective. Equivalently, TailRL maximizes the expected log-probability of clearing a uniformly sampled reward threshold.

TailRL uses the full support of the reward distribution rather than selecting an arbitrary cut-off. Uniform sampling of threshold τ assigns equal importance to the log-likelihood of their tail-events. It introduces no weighing hyperparameter. More generally, one could sample τ from a non-uniform distribution inducing a re-weighting of the log-likelihood terms. Any strictly positive normalized weighting is exactly equivalent to applying TailRL after a monotone transformation of the rewards (Proposition 9). We use the uniform distribution throughout and leave task-specific reward shaping to future work. The audit defines the population objective; it does not yet prescribe the finite-rollout estimator used for training (Section 4).

4 Estimating the TailRL Gradient

Training observes only a finite group of rollouts during training. The population-level objective is inestimable from finite rollouts. We show that a rollout budget of N naturally defines an order-N truncation of the TailRL objective and admits a simple unbiased policy-gradient estimator.

4.1 From Finite Rollouts to a Finite-Order Objective

For an order $T \geq 1$, define

$$J^{(T)}_{\mathrm{TailRL}}(\theta; x) := \sum_{k=1}^{T} \frac{\text{Best-of-k}(\theta; x) - 1}{k}. \tag{23}$$

Its gradient is

$$\nabla_\theta J^{(T)}_{\mathrm{TailRL}}(\theta; x) = \sum_{k=1}^{T} \frac{1}{k} \nabla_\theta \text{Best-of-k}(\theta; x). \tag{24}$$

At $T = 1$, this has the standard expected-reward gradient, while $J^{(T)}_{\mathrm{TailRL}}(\theta; x) \to J_{\mathrm{TailRL}}(\theta; x)$ as $T \to \infty$.

Equivalently, the finite-order gradient can be written directly in terms of tail probabilities:

$$\nabla_\theta J^{(T)}_{\mathrm{TailRL}}(\theta; x) = \int_0^1 \frac{1 - (1 - p_\theta(x, \tau))^T}{p_\theta(x, \tau)} \nabla_\theta p_\theta(x, \tau)\, d\tau. \tag{25}$$

The threshold weight equals 1 at $T = 1$ and approaches $1/p_\theta(x, \tau)$ as $T \to \infty$. Thus, increasing T smoothly moves the objective from expected-reward RL toward population TailRL, progressively emphasizing reward levels that are harder to reach. Figure 3 visualizes this interpolation and compares it with PKPO.

For binary rewards, the finite-order family and its estimator reduce exactly to their MaxRL counterparts; see Appendix C.8.

4.2 Finite-Rollout Estimator

Critic-free methods express policy gradients as weighted combination of the score-function for different rollouts. In this section we seek to express the exact weights that allows us to give an unbiased estimate of the gradient of the order-N truncated objective.

Suppose we draw N independent rollouts $z_1, \ldots, z_N \sim \pi_\theta(\cdot \mid x)$. At each reward threshold, we divide one unit of credit equally among the sampled rollouts that exceed that threshold. A rollout accumulates this

credit over every threshold below its reward:

$$\omega(r(x,z_i)) := \int_0^{r(x,z_i)} \frac{d\tau}{\sum_{j=1}^{N} \mathbb{1}_{\{r(x,z_j)>\tau\}}}. \tag{26}$$

Thresholds cleared by fewer rollouts therefore contribute more weight.

The weights can be computed exactly after sorting the rewards. Let $r_{(1)} \le \cdots \le r_{(N)}$ denote the sorted rewards and set $r_{(0)} := 0$. Then

$$\omega(r_{(i)}) = \omega(r_{(i-1)}) + \frac{r_{(i)} - r_{(i-1)}}{N - i + 1}, \qquad \omega(r_{(0)}) = 0. \tag{27}$$

The resulting policy-gradient estimator has the standard score-function form:

$$g^{(N)}_{\text{TailRL}}(x) := \sum_{i=1}^{N} \omega(r(x,z_i))\, S(x,z_i). \tag{28}$$

Thus, TailRL differs from a standard critic-free policy-gradient method only in how sampled rewards are converted into rollout weights.

Theorem 2 (Unbiased finite-rollout estimator)**.** *For N independent rollouts,*

$$\mathbb{E}[g^{(N)}_{\text{TailRL}}(x)] = \nabla_\theta J^{(N)}_{\text{TailRL}}(\theta; x). \tag{29}$$

Hence, the rollout budget determines which member of the TailRL family is optimized: one rollout recovers the expected-reward gradient, while larger rollout groups incorporate progressively higher Best-of-k learning signals. This differs from REINFORCE, where increasing the rollout count reduces estimation variance without changing the underlying expected-reward objective.

Centered advantages.. In practice, we center the rollout weights within each group:

$$A_i := \omega(r(x,z_i)) - \bar{\omega}, \qquad \bar{\omega} := \frac{1}{N}\sum_{j=1}^{N} \omega(r(x,z_j)). \tag{30}$$

Centering reduces variance and allows TailRL to be used as a drop-in replacement for the advantage calculation in standard policy-gradient implementations. Because the baseline is estimated from the same rollout group, the centered estimator is unbiased for $\nabla_\theta J^{(N-1)}_{\text{TailRL}}$ rather than $\nabla_\theta J^{(N)}_{\text{TailRL}}$. Complete derivations and proofs are given in Appendix C.

5 Unifying Gradient Weight View

To compare population-level TailRL with expected reward maximization, we look one step earlier. We express the gradients of TailRL and expected reward maximization under a unified view and ask how each objective up-weights its gradients. **TailRL and expected reward maximization belong to a common family of objectives**.

Consider a fixed input x and a fixed parameters θ of our policy $\pi_\theta(\cdot \mid x)$. Recall that we can express the tail-probability as,

$$p_\theta(x,\tau) := \Pr_{z\sim\pi_\theta(\cdot|x)}(r(x,z) > \tau) \tag{31}$$

Its gradient $\nabla_\theta p_\theta(x,\tau)$ points in the direction that makes the tail-event more likely. Up to constants independent of θ, they can be written in the common form they can be expressed as,

$$J_\phi(\theta; x) := \int_0^1 \phi(p_\theta(x,\tau))\, d\tau, \tag{32}$$

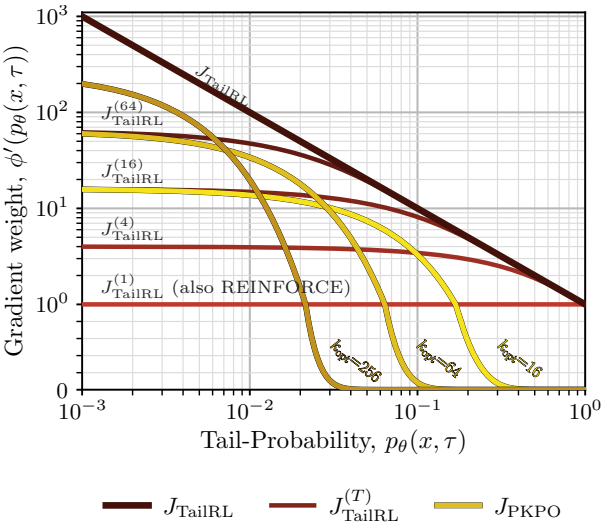

Figure 3: Gradient weight assigned to tail-probability $p_\theta(x,\tau)$. Order-T TailRL approaches the population weight. PKPO's weight is capped at k_{opt} and vanishes as $p_\theta(x,\tau)$ grows.

Here, $\phi(p)$ specifies how the objective values a tail-probability $p_\theta(x, \tau)$. Differentiating w.r.t θ gives

$$\nabla_\theta J_\phi(\theta; x) = \int_0^1 \phi'(p_\theta(x, \tau))\, \nabla_\theta p_\theta(x, \tau)\, d\tau. \tag{33}$$

The derivative ϕ' is therefore the gradient weight of the derivative of a tail-probability $p_\theta(x, \tau)$, and it does not introduce a new objective. It is the marginal value that the objective assigns to making that level more likely. The three objectives differ only in this weight (Figure 3 and Table 1).Expected reward maximization uses $\phi(p) = p$, so $\phi'(p) = 1$: every tail-probability receives the same weight, regardless of how often the policy reaches it. TailRL uses $\phi(p) = \log p$, so $\phi'(p) = 1/p$. Because $p_\theta(x, \tau)$ is non-increasing in τ, this places greater weight on higher reward thresholds that the policy reaches rarely. The gradient of order-T member of the truncated family is,

Table 1: Scalar function ϕ and gradient weights ϕ'. Standard RL gives equal weight to gradients while TailRL weights the gradients inverse tail-probability.

Objective	$\phi(p)$	$\phi'(p)$
J_{RL}	p	1
$J^{(T)}_{\text{TailRL}}$	$-\sum_{\ell \le T}(1-p)^\ell/\ell$	$\frac{1-(1-p)^T}{p}$
J_{TailRL}	$\log p$	$1/p$
J_{PKPO}	$1-(1-p)^{k_{\text{opt}}}$	$k_{\text{opt}}(1-p)^{k_{\text{opt}}-1}$

$$\nabla_\theta J^{(T)}_{\text{TailRL}}(\theta; x) = \int_0^1 \frac{1-(1-p_\theta(x,\tau))^T}{p_\theta(x,\tau)} \nabla_\theta p_\theta(x, \tau)\, d\tau. \tag{34}$$

It up-weights the gradient of the tail probabilities as $\phi'(p) = \frac{1-(1-p)^T}{p}$, i.e. the multiplier in Eq. (34). At $T = 1$, this multiplier equals 1, recovering the expected reward gradient. As $T \to \infty$, it approaches the tail-likelihood weight $1/p$. Expected reward maximization gives equal weight to the gradient of all tail-probabilities.

6 Experiments

We organize the experiments around four questions implied by the theory. [ys: quickly recap the theory and itemize the 4 qs]

1. First, is the population-level TailRL objective a useful learning target, and does its finite-rollout estimator approach it as number of rollouts increases? (Section 4.1)
2. Second, does TailRL help specifically when high-reward rollouts are attainable but rare? (Section 3)
3. Third, does its alignment with Best-of-k yield stronger inference-time scaling? (Section 3.1)
4. Finally, can TailRL prevent a common moderate-reward behavior from displacing rarer, better outcomes? (Section 3.1 and Section 3)

To answer these questions, we devise 4 experimental settings. We test the population level objective and its finite-rollouts approximation on a localization task in the **ImageNet** dataset (Section 6.1) . To understand the behavior of TailRL under rare, high reward rollouts, we consider a **Text-Maze** setting (Section 6.2). **GUI grounding** allows us to study the scaling behavior of models trained with TailRL as more inference compute in poured into the problem (Section 6.3) and finally, **Code Optimization** tests resistance to a safe but suboptimal shortcut (Section 6.4). Together, they test whether TailRL works for the reasons predicted by the theory, from the population objective to the behavior of the learned policy.

We compare TailRL with GRPO (Shao et al., 2024), RLOO (Ahmadian et al., 2024), two popular group-based policy optimization algorithms, and PKPO Walder and Karkhanis (2025) that maximizes Best-of-k_{opt} for an inference budget k_{opt} under pure on-policy policy gradient setup to avoid confounders.

6.1 ImageNet Object Localization

ImageNet object localization requires predicting a bounding box around an object of interest without classifying the object (Russakovsky et al., 2015). This setting addresses three questions. First, how does reinforcement learning from a scalar reward compare with supervised objectives that directly observe the ground-truth bounding box? Second, does the finite rollout TailRL gradient estimator $g^{(N)}_{\text{TailRL}}$ approach the population-level gradient $\nabla_\theta J_{\text{TailRL}}$ as N increases? Third, how does TailRL compare with the expected reward baselines at matched and smaller training rollout budgets?

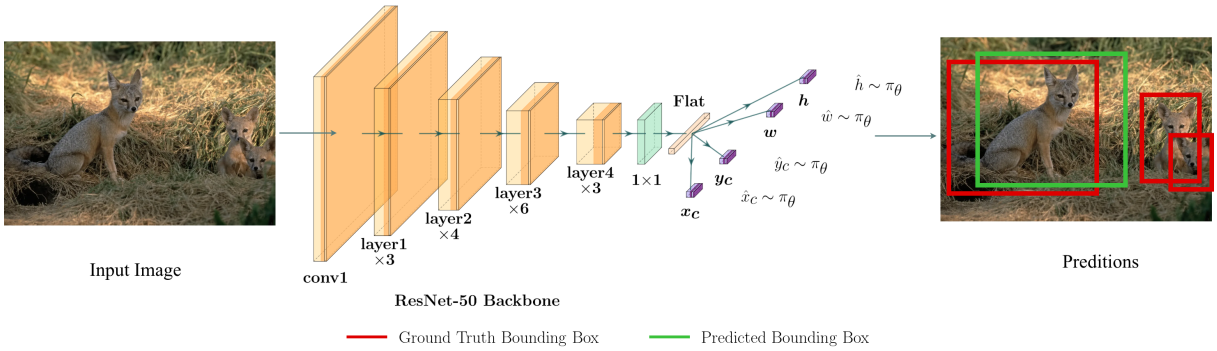

Figure 4: (ImageNet object localization) An overview of the ImageNet object localization task. A ResNet-50 backbone (He et al., 2016) encodes the image, and four categorical heads parameterize the policy. A rollout samples the box center coordinates $\hat{x}_c, \hat{y}_c$, width $\hat{w}$, and height $\hat{h}$. The sampled box is rewarded by its IoU with the matched ground-truth box.

Task and setup. For an input image x, a rollout z is a bounding box drawn from the categorical policy of Figure 4, and the continuous reward $r(x, z) \in [0, 1]$ is its intersection-over-union (IoU) with the ground-truth box. We report CorLoc@δ, the fraction of input images for which the greedy prediction has IoU greater than δ (Deselaers et al., 2010); mean IoU; and Best-of-k IoU, the largest IoU among k inference rollouts.

Its worthwhile to note that since the policy induces a categorical distribution over the finite set of possible bounding boxes, we can evaluate the probability and IoU reward of every box. This gives us the exact reward distribution in closed form. Using this we can directly compute and optimize the population-level objective J_{TailRL}. We also train supervised baselines that directly regress the ground-truth coordinates. Training and evaluation details are provided in Appendix H.

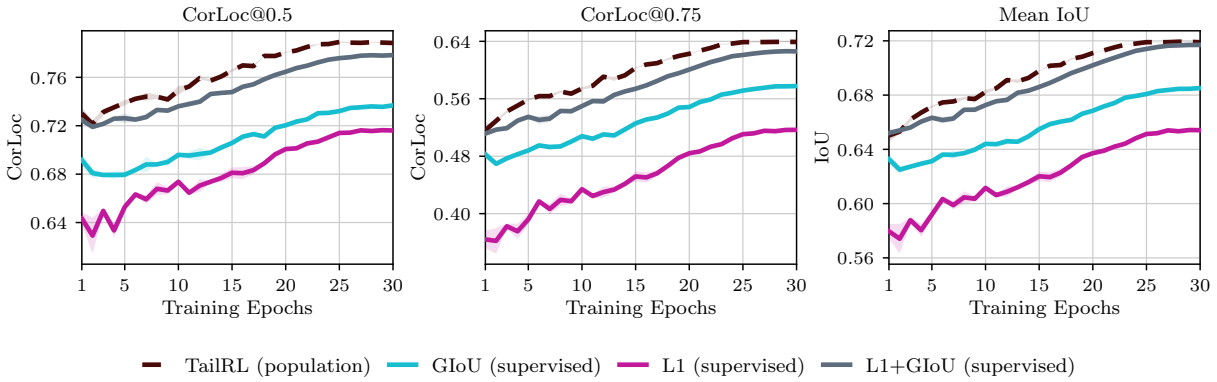

Figure 5: (ImageNet object localization) A comparison of population-level TailRL versus direct supervision for the task of ImageNet Object Localization. TailRL either outperforms or is competitive against task-specific objectives.

Comparison with direct supervision. Using only scalar IoU rewards, population-level TailRL matches or exceeds objectives that directly supervise the ground-truth coordinates (Figure 5). Among the supervised objectives, the combined L1+GIoU objective (Rezatofighi et al., 2019) used by DETR (Carion et al., 2020) is the strongest. TailRL achieves higher CorLoc@0.5 and CorLoc@0.75 than L1+GIoU while obtaining comparable mean IoU. It also outperforms the individual L1 and GIoU baselines across all reported metrics throughout training. These results show that under large training compute, optimizing a scalar continuous reward can compete with task-specific supervised objectives.

Takeaway 1

When exact tail-probabilities are estimable, the population-level TailRL performs as well as or better than task-specific supervised objectives that recieve ground truth information.

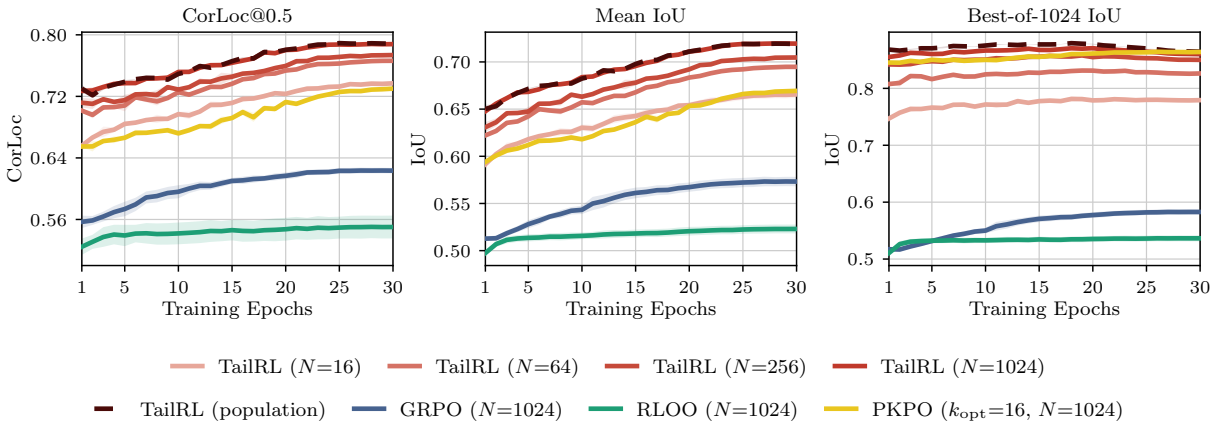

Figure 7: (ImageNet object localization) Comparing performance of expected-reward maximization baselines with TailRL on held-out validation set. We report CorLoc@0.5, mean IoU, and Best-of-1024 IoU. TailRL uses training rollouts $N \in \{16, 64, 256, 1024\}$, with darker curves indicating larger N; the dashed curve optimizes the exactly computed TailRL population-level objective. GRPO and RLOO use $N = 1024$ (Appendix H).

Increasing training rollout budget. Increasing training rollouts, N moves the finite rollout TailRL curves toward the exact population-level objective (Figure 7). At $N = 1024$, TailRL closely tracks the population-level objective, while smaller rollout budgets remain progressively farther away. This ordering holds across CorLoc@0.5, mean IoU, and Best-of-1024 IoU and other performance measures that we test for this experiment. We test this convergence directly at the gradient level in Figure 6. As N increases, the cosine similarity between the sampled finite rollout gradient and the population-level gradient rises steadily toward exact agreement. This increasing alignment is consistent with the convergence predicted in Section 4.1. Increasing rollouts not only reduce the variance of the objective we are trying to estimate, but it also approximates a higher order truncated objective.

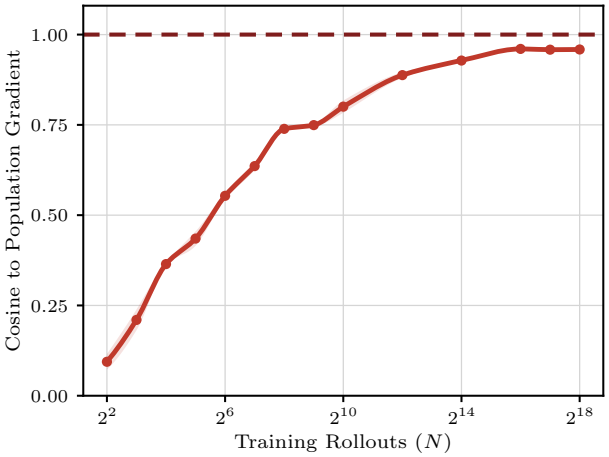

Figure 6: (ImageNet object localization) Finite rollout TailRL gradients converge toward the population-level gradient as training rollouts increase.

Takeaway 2

As training rollouts increase, the gradient of order-N truncated objective computed from finite rollout TailRL converges to the gradient of population-level TailRL.

Comparison with RL baselines. At a matched budget of $N = 1024$, TailRL outperforms GRPO and RLOO across all three metrics, despite every method receiving the same continuous IoU feedback. The gap does not close at smaller budgets: even at $N = 16$, TailRL exceeds both baselines trained at $N = 1024$, using $\frac{1}{64}$ as many training rollouts per input. TailRL also pareto dominates PKPO which maximizes expected maximum reward. With just $N = 16$ training rollouts, it outperforms at CorLoc@0.5 and mean IoU. At matched training compute, TailRL matches PKPO at Best-of-1024 reward, while significantly outperforming at mean IoU and CorLoc@0.5.

6.2 Text-Maze Navigation from Low-Success Initial Policies

Next we examine how TailRL and other baselines compare when we vary the quality of the initialization policy. By varying the amount of supervised pretraining before doing RL post-training, we obtain a controlled range of initial policies with varying coverage over high reward attaining rollouts. We ask whether TailRL can learn from poor initialization of policies and how it compares against expected reward maximization baselines.

Task and setup. For an input 17×17 maze represented as text, a rollout is a token sequence describing a path. The continuous reward $r(x, z) \in [0, 1]$ measures proximity to the goal and path length relative to the shortest path (Figure 8). A rollout receives reward 1 only when it reaches the goal along a shortest path, an event we call *shortest-path success*. Unsuccessful rollouts receive partial credit for ending closer to the goal, while successful rollouts receive more reward for shorter paths. We evaluate the post-trained policies on a held-out validation set of 1024 mazes.

By varying the amount of supervised pretraining on goal-reaching trajectories, we obtain initial policies with shortest-path success rates ranging from approximately 1% down to 0.01%. Starting from each checkpoint, we separately post-train policies with TailRL, GRPO, and PKPO (k_{opt}). All methods use the same initial policy and $N = 16$ unless stated otherwise. Model, reward, training, and evaluation details are provided in Appendix I.

Learning from low-success initial policies. Figure 9 shows that RLOO, GRPO and TailRL behave similarly when the initial policy produces shortest-path successes frequently, but diverge as the initial success rate falls. Despite higher coverage of initial policy, PKPO underperforms TailRL at Pass@1. The expected reward baselines fail to learn reliably in the low initial success regime. PKPO consistently improves Pass@1 across the spectrum of initialization policy yet still underperforms TailRL at Pass@1 even at the regime of poor policy initalization. Once the initial success rate rises beyond this regime, TailRL, RLOO and GRPO learn well and their differences narrow. GRPO's performance improves before RLOO, although the performance becomes seed dependent before it consistently starts to navigate the maze. The advantage of TailRL is therefore concentrated where high-reward rollouts are attainable but rare. Inference and training-rollout budget sweeps are reported in Appendix I.5.

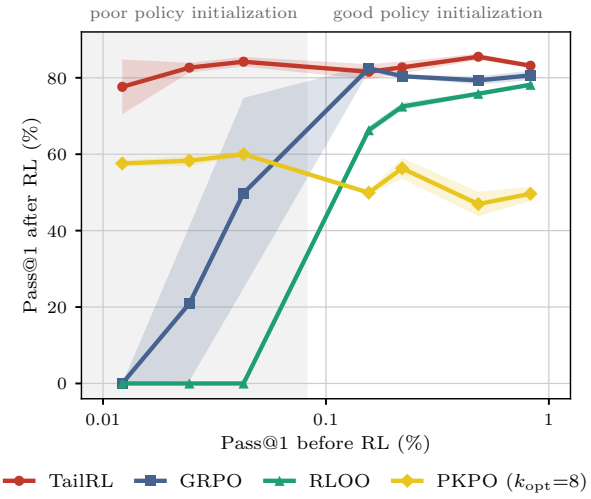

Figure 9: (Text-Maze) Pass@1 before and after RL post-training for the task of Text-Maze navigation. Shaded area marks the regime where initialization policy is poor. After RL post-training, RLOO and GRPO fail to reliably improve Pass@k in this regime.

Takeaway 3

TailRL upweighs gradients for rare exceptional rollouts. This results in TailRL outperforming other methods when initial policy has poor coverage over rare excellent rollouts.

6.3 GUI Grounding with Vision-Language Models

We next demonstrate the efficacy of TailRL on Vision Language Models (VLM). We consider a visual grounding task with verifiable rewards. Through this task, we show that models trained with TailRL benefit from inference-time sampling.

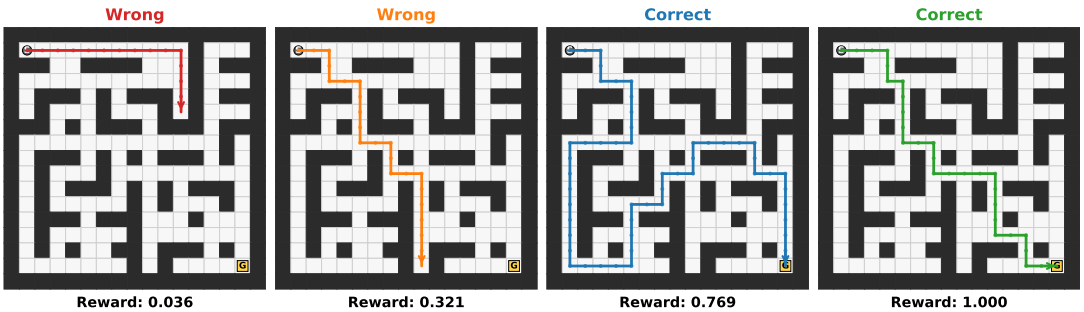

Figure 8: (Text-Maze) Four rollouts on one Text-Maze and the continuous reward each receives. The two left paths fail to reach the goal and still receive continuous credit for progress. The third reaches the goal but wanders, so it gets rewarded below a shortest path. Only the rightmost, a shortest path (right), earns reward 1.

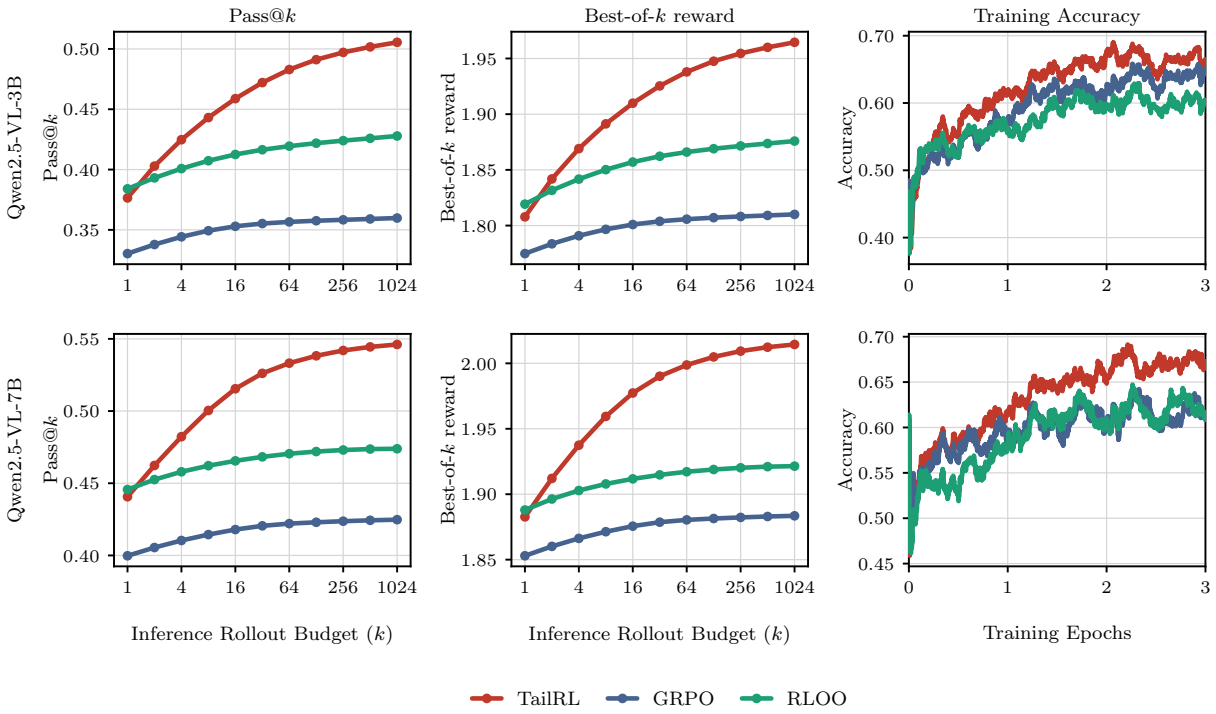

Figure 10: (GUI-grounding) Evaluation results for GUI grounding task on ScreenSpot-Pro benchmark for Qwen2.5-VL-3B (top) and 7B (bottom). Columns show Pass@k, Best-of-k reward, and exponentially smoothed training-batch accuracy. TailRL matches RLOO's Pass@1024 using 8 rollouts at 3B and 4 rollouts at 7B scale, a reduction of 128× and 256× in test-time compute respectively.

Task and setup: Given a screenshot and a natural-language instruction to perform a click, the VLM policy generates the coordinates of the click location. The continuous reward $r(x, z) \in [0, 2.5]$ combines proximity to the target, a bonus for clicking inside the target element, and a format bonus (Yuan et al., 2025). We fine-tune Qwen2.5-VL-3B and Qwen2.5-VL-7B (Bai et al., 2025) on the GTA1 grounding corpus (Yang et al., 2026). The training configuration is identical across TailRL, GRPO, and RLOO except for the advantage estimator. We evaluate the resulting policies on ScreenSpot-Pro (Li et al., 2025). Here, Pass@k is the probability that at least one of k inference rollouts clicks inside the target element (Chen et al., 2021), while Best-of-k reward is the largest continuous reward among those rollouts. The reward construction, training protocol and evaluation procedure are detailed in Appendix J.

Results: At both model scales, TailRL and RLOO achieve similar Pass@1, but they substantially differ in how the learned policies respond to additional inference sampling (Figure 10). As inference rollouts k increases, TailRL's Pass@k continues to rise, whereas RLOO plateaus considerably earlier; GRPO's Pass@k is inferior to TailRL and RLOO. TailRL leaves more inputs with a non-negligible probability of producing a successful click, so additional samples continue to reveal useful candidates.

We define the *matching budget* as the smallest evaluated value of k at which TailRL reaches or exceeds a baseline's mean Pass@1024. On ScreenSpot-Pro, at 3B, TailRL reaches RLOO's Pass@1024 with 8 rather than 1024 inference rollouts, a 128-fold reduction; at 7B it does so with 4 rollouts, a 256-fold reduction. A single TailRL rollout exceeds GRPO's mean Pass@1024 at both model scales.

Takeaway 4

On GUI grounding, TailRL shows superior scaling of inference compute when compared against expected reward maximization baselines.

6.4 Code Runtime Optimization

We now ask a qualitatively different question: what happens when training is attracted to a safe but systematically suboptimal behavior hurting further exploration? Such behaviors provide a reliable moderate reward and can become stable solutions, even when rarer and riskier behaviors offer substantially better outcomes (Skalse et al., 2022; Baronio et al., 2025). This creates a stress test for TailRL: whether it

concentrates the policy on the dependable yet suboptimal shortcuts or improves the policy's coverage over high-reward tail.

Task and setup. Each input is a slow C++ program from the PIE (Shypula et al., 2024) corpus of competitive-programming. The LLM is prompted to write a faster yet correct version of the input program. A rollout is a rewritten program and is compiled and executed against the problem's test suite. Incorrect outputs receives zero reward otherwise receives a reward equal to the speedup over the input program. We measure speedup using gem5 time so that timing noise cannot create spurious improvements (Binkert et al., 2011; Lowe-Power et al., 2020). We post-train Qwen3-1.7B (Yang et al., 2025a) with TailRL, GRPO, and RLOO using $N = 16$ rollouts per program.

We report mean reward over all rollouts; density of Best-of-k rewards at $k = 1024$; and the fraction of rollouts that pass every test; and Best-of-k reward density at $k = 1024$. A policy that always reproduces its input obtains a mean reward of 1 and passes all test cases. Full task, training, and evaluation details are provided in Appendix K.

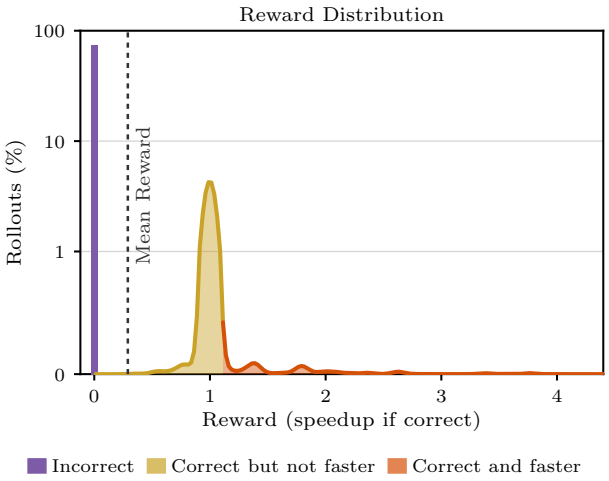

Figure 11: (Code runtime optimization) Initial reward distribution over all test problems of PIE dataset for Qwen3-1.7B. 74.4% of rollouts are incorrect, 23.5% are correct but not faster than the input, concentrating in a spike at 1×, and 2.1% are correct and faster.

Results. GRPO and RLOO rapidly converge toward the copying shortcut (Figure 12) as evidenced by sample rollouts (Appendix K.7). Their correctness rise above 98%, while their mean rewards settle just below 1.0. Their policy entropy collapses by one to two orders of magnitude over the same interval (Figure 25). Expected-reward training therefore converges to the most reliable mode of the reward distribution while eliminating the behavior to maximize coverage over excellent outputs.

TailRL follows a different trajectory. Rather than collapsing onto the guaranteed reward from copying the input, it maintains substantially higher entropy and continues producing risky rewrites. This lowers single-rollout correctness, but places more probability on programs that are both correct and meaningfully faster, raising mean reward to 2.92, nearly three times the copying value. All methods use the same one-epoch training budget, and TailRL is still improving at step 300; these are therefore matched-compute results rather than converged endpoints. On the held-out test set the trained policies separate sharply: TailRL's mean Best-of-1024 speedup is 7.7× against 0.98× for GRPO and 0.96× for RLOO, whose best

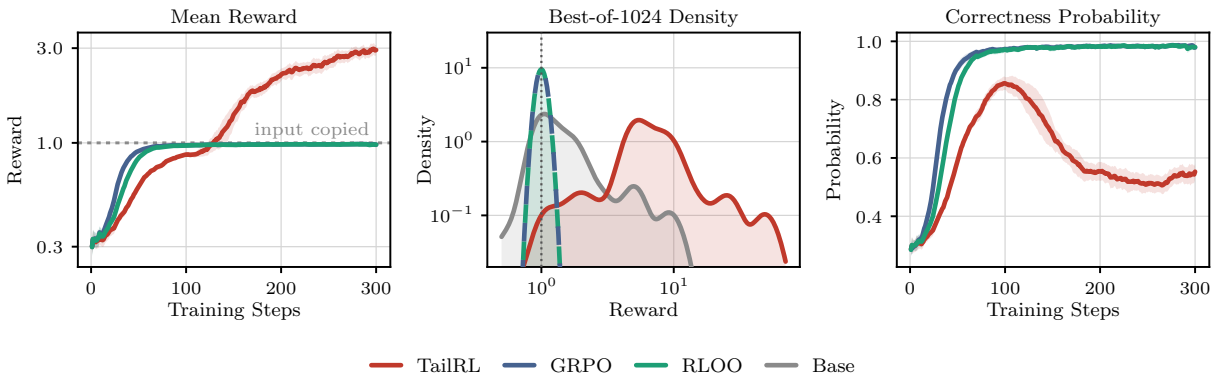

Figure 12: (Code runtime optimization) (Left) Average reward during training for 1 epoch, EMA over three runs per objective. (Middle) kernel density of the per-problem Best-of-1024 reward on the test-set problems of PIE. GRPO and RLOO are drawn with alternating dashes because their densities coincide at a reward of 1.0 indicating that they only echo the input program. Right: fraction of training rollouts for a given batch passing every test. In this task, it is better to explore and find the faster rewrite of the input program than to copy the input and be correct. Correctness only gets a reward of 1.0 while correct and meaningfully faster programs receive much higher rewards. Just copying the input is a degenerate soltuion exhibited by RLOO and GRPO Appendix K.7 All experiments have been performed on 3 seeds per method.

rollouts never beat the input they reproduce (Figure 12, middle; Figure 26).

Takeaway 5

When an easy suboptimal solution exists, expected reward maximization methods settle for it. TailRL keeps searching and learns to produce rarer and better outcomes.

7 Related Work

Objectives beyond expected reward. Reinforcement-learning post-training for reasoning and agentic models still largely maximizes expected reward (Ouyang et al., 2022; Guo et al., 2025). GRPO (Shao et al., 2024) and RLOO (Ahmadian et al., 2024) are practical, critic-free variants of the score-function policy gradient (Williams, 1992). RLOO changes the baseline, while GRPO applies a common normalization to the update for each input; neither changes the relative weighting of reward levels within that input (Section 5). TailRL changes the objective instead. Closest to our work, MaxRL maximizes the log-probability of success for binary rewards and decomposes its gradient into a harmonic sum of Pass@k gradients (Tajwar et al., 2026). PKPO derives unbiased estimators for a chosen Pass@k objective and its continuous counterpart, Best-of-k (Walder and Karkhanis, 2025); related methods directly optimize a selected inference-time metric (Tang et al., 2025; Chen et al., 2025b) or study objectives defined by monotone transforms of success probability (Davis and Recht, 2025). These approaches select a threshold, transform, or inference budget. TailRL instead integrates log tail-probability over all reward thresholds, yielding harmonic combinations of Best-of-k objectives and recovering MaxRL exactly for binary rewards.

Coverage and tail-sensitive reinforcement learning. Expected-reward training can narrow the policy distribution and remove rare, high-reward behavior, an effect studied mechanistically (Cui et al., 2025) and observed at large sampling budgets (Yue et al., 2025; Kirk et al., 2024). Rather than adding a separate diversity regularizer, TailRL addresses this through the objective itself by weighting each reward level inversely by how often the policy reaches it. TailRL is also distinct from distributional and risk-sensitive reinforcement learning, which learn a return distribution through a critic or optimize a tail statistic at a chosen risk level (Bellemare et al., 2017; Rockafellar and Uryasev, 2000). TailRL uses no learned critic: its policy-gradient weights are computed directly from each rollout group, and it integrates over all reward thresholds rather than fixing one risk level, which can otherwise overlook rare successes (Greenberg et al., 2022). We discuss further connections to exploration, sample allocation, threshold decompositions of continuous rewards, and the evaluated domains in Appendix B.

8 Conclusion

We introduced TailRL, a likelihood objective for continuous rewards that maximizes the expected log-probability of exceeding a uniformly sampled reward threshold. It recovers MaxRL exactly for binary rewards, decomposes harmonically over Best-of-k gradients, and admits a simple, critic-free finite-rollout estimator. Across four settings, its gains were largest when high-reward rollouts were rare or when training was drawn toward a common but suboptimal behavior. In these regimes, TailRL learned more reliably from rare outcomes, benefited more from additional training and inference samples, and avoided suboptimal collapse. More broadly, a continuous reward is more than a scalar to average: it defines a family of success events, one at every reward level. TailRL provides a practical way to optimize their likelihoods.

Acknowledgements

This research used the DeltaAI advanced computing and data resource (Bode et al., 2025), which is supported by the National Science Foundation under award OAC-2320345 and by the State of Illinois. DeltaAI is a joint effort of the University of Illinois Urbana-Champaign and its National Center for Supercomputing Applications. These resources were used through the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program (Boerner et al., 2023). Overall, this project used ACCESS allocations CIS250426, CIS260353, CIS260522, CIS260557, CIS260677, CIS260678, and CIS260679. We are especially grateful to Brett Bode of the NCSA Delta Support team, whose assistance in effectively using the Delta cluster was critical to completing this work on schedule. We would like to extend our sincere gratitude to Sooth Labs for their generous support by granting us their compute resources. This work was partially supported by the National Science Foundation under Grants CCF-2106778. Fahim and Shrinivas were funded in part by Stack AV and Skylark Labs

We are grateful to Sumukh Aithal, Ben Freed, Surgan Jandial, Sreyas Venkatraman, Kartik Sharma, Elton Lobo, Parv Maheshwari, Rhea Basappa, Srinath Ravi, Rishubh Parihar, Harsh Rangwani, Chaitanya Chawla, and Mayank Mishra for carefully reviewing earlier drafts and providing valuable feedback. We also thank Rohit Sonkar, Anoushka Alavilli, Jiayu Chen, and Mineui Hong from the CMU Auton Lab, and Lawrence Jang from Russ lab for their helpful feedback that improved the quality of the draft. The authors would also like to extend their gratitude to Prof. Yaser Sheikh and Prof. Yonatan Bisk for helpful discussions and suggestions throughout this work.

References

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22 (98):1–76, 2021.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 12248–12267. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.662.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report, 2025. URL https://arxiv.org/abs/2502.13923.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn RL for generating CUDA kernels, 2025. URL https://arxiv.org/abs/2507.11948.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1471–1479, 2016.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458, 2017.

Marc G. Bellemare, Will Dabney, and Mark Rowland. *Distributional Reinforcement Learning*. MIT Press, 2023. doi: 10.7551/mitpress/14207.001.0001.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011. doi: 10.1145/2024716.2024718.

Brett Bode, Gregory Bauer, Laura Herriott, Volodymyr Kindratenko, and William Gropp. Deltaai: A national resource for ai/ml research. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, PEARC '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713989. doi: 10.1145/3708035.3736062. URL https://doi.org/10.1145/3708035.3736062.

Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. Access: Advancing innovation: Nsf's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, PEARC '23, page 173–176, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3569951.3597559. URL https://doi.org/10.1145/3569951.3597559.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

Wenzhi Cao, Vahid Mirjalili, and Sebastian Raschka. Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, 140:325–331, 2020.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, volume 12346, pages 213–229. Springer, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for LLM reasoning, 2025a. URL https://arxiv.org/abs/2505.14970.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@k training for adaptively balancing exploration and exploitation of large reasoning models, 2025b. URL https://arxiv.org/abs/2508.10751.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pages 30377–30385, 2026. doi: 10.1609/aaai.v40i36.40290.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research*, 18(167):1–51, 2018.

Pierre Clavier, Stéphanie Allassonnière, and Erwan Le Pennec. Robust reinforcement learning with distributional risk-averse formulation, 2022. URL https://arxiv.org/abs/2206.06841.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. The entropy mechanism of reinforcement learning for reasoning language models, 2025. URL https://arxiv.org/abs/2505.22617.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1096–1105, 2018a.

Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2892–2901, 2018b.

Runpeng Dai, Linfeng Song, Haolin Liu, Zhenwen Liang, Dian Yu, Haitao Mi, Zhaopeng Tu, Rui Liu, Tong Zheng, Hongtu Zhu, and Dong Yu. CDE: Curiosity-driven exploration for efficient reinforcement learning in large language models, 2025. URL https://arxiv.org/abs/2509.09675.

Xingyu Dang, Christina Baek, J Zico Kolter, and Aditi Raghunathan. Assessing diversity collapse in reasoning. In *Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL https://openreview.net/forum?id=AMiKsHLjQh.

Damek Davis and Benjamin Recht. What is the objective of reasoning with reinforcement learning?, 2025. URL https://arxiv.org/abs/2510.13651.

Thomas Deselaers, Bogdan Alexe, and Vittorio Ferrari. Localizing objects while learning their appearance. In *European Conference on Computer Vision (ECCV)*, pages 452–466. Springer, 2010.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. StepCoder: Improving code generation with reinforcement learning from compiler feedback. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4571–4585, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.251. arXiv:2402.01391.

Eibe Frank and Mark Hall. A simple approach to ordinal classification. In *Machine Learning: ECML 2001*, volume 2167 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2001. doi: 10.1007/3-540-44795-4_13.

Jingchu Gai, Guanning Zeng, Huaqing Zhang, and Aditi Raghunathan. Differential smoothing mitigates sharpening and improves LLM reasoning, 2025. URL https://arxiv.org/abs/2511.19942.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning (ICML)*, pages 10835–10866, 2023.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 267, pages 19034–19055. PMLR, 2025.

Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Ido Greenberg, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Efficient risk-averse reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Anthony GX-Chen, Jatin Prakash, Jeff Guo, Rob Fergus, and Rajesh Ranganath. KL-regularized reinforcement learning is designed to mode collapse, 2025. URL https://arxiv.org/abs/2510.20817.

Jubayer Ibn Hamid, Ifdita Hasan Orney, Ellen Xu, Chelsea Finn, and Dorsa Sadigh. Polychromic objectives for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Zhezheng Hao, Hong Wang, Haoyang Liu, Jian Luo, Jiarui Yu, Hande Dong, Qiang Lin, Can Wang, and Jiawei Chen. Rethinking entropy interventions in RLVR: An entropy change perspective. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 31105–31133, 2026. doi: 10.18653/v1/2026.acl-long.1436.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016. doi: 10.1109/CVPR.2016.90.

Jian Hu, Mingjie Liu, Ximing Lu, Fang Wu, Zaid Harchaoui, Shizhe Diao, Yejin Choi, Pavlo Molchanov, Jun Yang, Jan Kautz, and Yi Dong. BroRL: Scaling reinforcement learning via broadened exploration. In *International Conference on Machine Learning (ICML)*, 2026.

Yuhua Jiang, Jiawei Huang, Yufeng Yuan, Xin Mao, Yu Yue, Qianchuan Zhao, and Lin Yan. Risk-sensitive reinforcement learning for alleviating exploration dilemmas in large language models. In *International Conference on Learning Representations (ICLR)*, 2026.

Jean Kaddour. Target policy optimization, 2026. URL https://arxiv.org/abs/2604.06159.

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 *International Conference on Learning Representations (ICLR)*, 2024.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! In *Deep Reinforcement Learning Meets Structured Prediction, ICLR 2019 Workshop*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1lgTGL5DE.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Conference on Language Modeling (COLM)*, 2025.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C.H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2207.01780.

Pierre L'Ecuyer. Note: On the interchange of derivative and expectation for likelihood ratio derivative estimators. *Management Science*, 41(4):738–747, 1995.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM 2025)*, pages 8778–8786. ACM, 2025. doi: 10.1145/3746027.3755688.

Ling Li and Hsuan-Tien Lin. Ordinal regression by extended binary classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 19, pages 865–872. MIT Press, 2006.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. RLTF: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*, 2023. arXiv:2307.04349.

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, Shengyu Zhang, Hongxia Yang, and Fei Wu. InfiGUI-G1: Advancing GUI grounding with adaptive exploration policy optimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 32267–32275. AAAI Press, 2026. doi: 10.1609/aaai.v40i38.40500.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025.

Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+, 2020. URL https://arxiv.org/abs/2007.03152.

Peter McCullagh. Regression models for ordinal data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):109–127, 1980.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020.

Phuc Minh Nguyen, Chinh D. La, Duy M. H. Nguyen, Nitesh V. Chawla, Binh T. Nguyen, and Khoa D. Doan. The reasoning boundary paradox: How reinforcement learning constrains language models, 2025. URL https://arxiv.org/abs/2510.02230.

Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. Ordinal regression with multiple output CNN for age estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4920–4928, 2016.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Ifdita Hasan Orney, Jubayer Ibn Hamid, Shreya S Ramanujam, Shirley Wu, Hengyuan Hu, Noah Goodman, Dorsa Sadigh, and Chelsea Finn. Poly-EPO: Training exploratory reasoning models, 2026. URL https://arxiv.org/abs/2604.17654.

Ian Osband. Delightful policy gradient, 2026. URL https://arxiv.org/abs/2603.14608.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Matteo Papini, Damiano Binaghi, Giuseppe Canonaco, Matteo Pirotta, and Marcello Restelli. Stochastic variance-reduced policy gradient. In *International Conference on Machine Learning (ICML)*, 2018.

Paavo Parmas, Yongmin Kim, Kohsei Matsutani, Shota Takashiro, Soichiro Nishimori, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. OrderGrad: Optimizing beyond the mean with order-statistic policy gradient estimation, 2026. URL https://arxiv.org/abs/2606.06096.

Nirmal Patel, Fei Wang, and Inderjit S. Dhillon. ODRPO: Ordinal decompositions of discrete rewards for robust policy optimization, 2026. URL https://arxiv.org/abs/2605.12667.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, pages 2778–2787. PMLR, 2017.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. POPE: Learning to reason on hard problems via privileged on-policy exploration, 2026. URL https://arxiv.org/abs/2601.18779.

Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–41, 2000. doi: 10.21314/JOR.2000.038.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Rylan Schaeffer, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. How do large language monkeys get their power (laws)? In *International Conference on Machine Learning (ICML)*, volume 267, pages 53132–53176. PMLR, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Xintong Shi, Wenzhi Cao, and Sebastian Raschka. Deep neural networks for rank-consistent ordinal regression based on conditional probabilities. *Pattern Analysis and Applications*, 26:941–955, 2023.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*, 2023. arXiv:2301.13816.

Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/pdf?id=ix7rLVHXyY.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 9460–9471, 2022. arXiv:2209.13085.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Yuda Song, Julia Kempe, and Remi Munos. Outcome-based exploration for LLM reasoning, 2025. URL https://arxiv.org/abs/2509.06941.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1057–1063. MIT Press, 1999.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2026.

Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the CVaR via sampling. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2993–2999, 2015.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. Optimizing language models for inference time objectives using reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 59066–59085, 2025. URL https://arxiv.org/abs/2503.19595.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, et al. Kimi k1.5: Scaling reinforcement learning with LLMs, 2025. URL https://arxiv.org/abs/2501.12599.

Christos Thrampoulidis, Sadegh Mahdavi, and Wenlong Deng. Advantage shaping as surrogate reward maximization: Unifying Pass@K policy gradients. *Transactions on Machine Learning Research*, 2026.

Jens Tuyls, Dylan J. Foster, Akshay Krishnamurthy, and Jordan T. Ash. Representation-based exploration for language models: From test-time to post-training. In *International Conference on Learning Representations (ICLR)*, 2026.

Christian Walder and Deep Tejas Karkhanis. Pass@K policy optimization: Solving harder reinforcement learning problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=yfcpdY4gMP.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why RLVR may or may not escape its origin, 2025. URL https://arxiv.org/abs/2507.14843.

Wei Xiong, Chenlu Ye, Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, Jiang Bian, Nan Jiang, and Tong Zhang. Reinforce-Ada: An adaptive sampling framework under non-linear RL objectives, 2025. URL https://arxiv.org/abs/2510.04996.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, Ran Xu, Liyuan Pan, Silvio Savarese, Caiming Xiong, and Junnan Li. GTA1: GUI test-time scaling agent. In *International Conference on Learning Representations (ICLR)*, 2026.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Yongxin Wang, Dongchun Xie, Hanhui Li, Yiwei Wang, Xiaodan Liang, and Jing Tang. Depth-breadth synergy in RLVR: Unlocking LLM reasoning gains with adaptive exploration, 2025b. URL https://arxiv.org/abs/2508.13755.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, and Bo Li. SE-GUI: Enhancing visual grounding for GUI agents via self-evolutionary reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. ACECODER: Acing coder RL via automated test-case synthesis. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, Vienna, Austria, 2025. Association for Computational Linguistics. arXiv:2502.01718.

Kaiqing Zhang, Alec Koppel, Hao Zhu, and Tamer Başar. Global convergence of policy gradient methods to (almost) locally optimal policies. *SIAM Journal on Control and Optimization*, 58(6):3586–3612, 2020.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Yifu Zheng. RL2ML: Finite-rollout surrogate objectives from reinforcement learning to maximum likelihood, 2026. URL https://arxiv.org/abs/2605.30154.

Appendix Contents

A Notation

Table 2 collects every symbol used in the paper. One symbol, one meaning is strictly enforced.

Table 2: Notation.

Symbol	Meaning
x	an input prompt
θ	policy parameter
ρ	the input distribution
$\pi_\theta(\cdot \mid x)$	the policy with parameters θ
$z \sim \pi_\theta(\cdot \mid x)$	a rollout; $z_{1:k}$ are i.i.d. rollouts
$r(x, z)$	the reward of rollout z
$S(x, z)$	the score-function $\nabla_\theta \log \pi_\theta(z \mid x)$
$\tau \in [0, 1)$	a threshold on the reward range, as in Section 1
$p_\theta(x, \tau)$	tail-probability $\Pr_{z\sim\pi_\theta(\cdot\mid x)}[r(x, z) > \tau]$
$q_\theta(x)$	the success probability when the reward is binary
$J_{\mathrm{RL}}(\theta; x)$	the expected reward on x
$J_{\mathrm{TailRL}}(\theta; x)$	the tail-likelihood $\int_0^1 \log p_\theta(x, \tau)\, d\tau$
$J^{(T)}_{\mathrm{TailRL}}(\theta; x)$	its order-T truncation (compute-indexed family)
T	the truncation order; $T \to \infty$ recovers J_{TailRL}
N	rollouts per input in a training group
M	number of binary reward training rollouts that succeed
$g^{(N)}_{\mathrm{TailRL}}$	the N-rollout estimator, no baseline
$\hat{g}^{(N)}_{\mathrm{TailRL}}$	the same estimator with the mean baseline
A_i	the advantage assigned to rollout z_i
$\omega(r)$	un-centered gradient weight for rollout w/ reaward r
$\omega^{(N)}_\theta(r)$	un-centered gradient weight for rollout under finite compute
$\sigma_\theta(x)$	variance of rewards for an input x
$\phi(p)$	scalar function; $\phi'(p)$ is the gradient weight (Section 5)
k	the inference rollout budget at evaluation
k_{opt}	the Pass@k order that the PKPO objective optimizes
Pass@k	probability that at least one of k samples of z succeeds
Best-of-k	the largest attained reward among k samples of z

B Extended Related Work

This section gives the full related-work discussion summarized in Section 7, together with several literatures not covered there: inference-time selection as its own topic, exploration and sample-allocation methods, threshold decompositions of continuous rewards, and the literatures behind each evaluated domain.

Reinforcement learning post-training. Reinforcement learning now drives post-training for reasoning and agentic models, and almost all of it maximizes expected reward. The recipe descends from learning against human preferences (Stiennon et al., 2020; Ouyang et al., 2022) and, once verifiable rewards replaced learned reward models, produced the current generation of reasoning systems (Guo et al., 2025; OpenAI et al., 2024; Team et al., 2025; Lambert et al., 2025). Practical estimators have converged on critic-free group-relative updates: GRPO standardizes rewards within a group (Shao et al., 2024), RLOO subtracts a leave-one-out baseline (Ahmadian et al., 2024; Kool et al., 2019), and later variants adjust clipping, normalization, and the level at which the ratio is formed (Yu et al., 2025; Zheng et al., 2025; Liu et al., 2025). Every one of these estimators traces back to the score-function policy gradient (Williams, 1992; Sutton et al., 1999; Schulman et al., 2017) and differs from the others in variance rather than in target. They share one population objective, the mean of the reward distribution, and in the weight view of Section 5 they share the flat gradient weight $\phi'(p) = 1$. TailRL changes that target rather than the variance around it.

Objectives beyond the mean. A recent line of work replaces the mean with a nonlinear functional of the policy's success probability, and TailRL belongs to it. MaxRL maximizes the log-probability of success for binary rewards and expands that logarithm into a harmonically weighted sum of Pass@k gradients (Tajwar et al., 2026). PKPO derives unbiased estimators for Pass@k and for its continuous generalization, the expected maximum of k rewards (Walder and Karkhanis, 2025), while other work optimizes a chosen inference-time metric directly (Tang et al., 2025; Chen et al., 2025b) or reweighs rollouts by problem difficulty (Yang et al., 2025b). Davis and Recht (2025) unify much of this by showing that popular algorithms implicitly ascend a monotone transform of the success probability, and Thrampoulidis et al. (2026) recast the resulting Pass@k advantage weights as surrogate reward maximization. Related objectives target diversity (Hamid et al., 2026; Orney et al., 2026), risk sensitivity (Jiang et al., 2026), and rare-success amplification (Osband, 2026; Kaddour, 2026). Contemporaneous works extend two pieces of this picture: RL2ML generalizes the harmonic coefficient for binary rewards (Zheng, 2026), and OrderGrad estimates gradients of L-statistics over sorted rewards (Parmas et al., 2026). Each of these methods commits to one threshold, one transform, or one order statistic. TailRL instead integrates the log tail-probability over every threshold, so its order-T truncation is a harmonic combination of Best-of-k objectives and reduces to MaxRL exactly when the reward is binary.

Inference-time scaling and selection. Drawing many rollouts and keeping the best one has become a standard axis of deployment compute, which makes Pass@k and Best-of-k the metrics that matter at inference. Repeated sampling raises coverage predictably over several orders of magnitude (Brown et al., 2024; Schaeffer et al., 2025), and allocating compute at test-time can beat allocating it to parameters (Snell et al., 2025). Selecting among the samples requires either agreement (Wang et al., 2023), a reward function (Cobbe et al., 2021; Lightman et al., 2024), or execution (Li et al., 2022), and the unbiased Pass@k estimator we use comes from this literature (Chen et al., 2021). Selection also has limits, since optimizing hard against an imperfect reward function eventually degrades true quality (Gao et al., 2023). This body of work measures or exploits the upper tail after training has finished. TailRL makes that same upper tail the training objective.

Distribution narrowing under expected reward training. Expected-reward post-training reliably sharpens the policy, which is the failure mode TailRL is designed to resist. Cui et al. (2025) characterize this as an entropy mechanism in which the policy-gradient covariance term stays positive and entropy falls monotonically without intervention. Measured at large sampling budgets, the effect is visible as a narrowing of what the policy can produce: base models can match or exceed post-trained models on Pass@k at large k (Yue et al., 2025), the reachable support changes little (Wu et al., 2025; Nguyen et al., 2025), and diversity drops across settings (Kirk et al., 2024; Dang et al., 2025). Regularization is one reported cause, since a KL penalty to a reference policy can itself favor mode collapse (GX-Chen et al., 2025). A parallel line intervenes on entropy directly, through token-level selection (Wang et al., 2025), entropy-aware bonuses (Cheng et al., 2026; Hao et al., 2026), or smoothing (Gai et al., 2025), and these interventions show that coverage can be partly recovered. TailRL takes a different route and reweights reward thresholds by the inverse of how often the policy reaches them (tail-probability), so rare high-reward threshold keep influence without an added regularizer.

Exploration and sample allocation. Exploration methods change which rollouts a learner sees, whereas TailRL changes how a fixed group of rollouts is weighted. Classical approaches add bonuses from visitation counts, prediction error, or information gain (Bellemare et al., 2016; Pathak et al., 2017; Burda et al., 2019). Their language-model counterparts penalize repeated outcomes (Song et al., 2025), reward novelty in representation space (Tuyls et al., 2026; Dai et al., 2025), widen the rollout budget (Hu et al., 2026), or schedule problems by difficulty (Chen et al., 2025a; Qu et al., 2026). Closest to us, Reinforce-Ada allocates samples adaptively under a non-linear objective and arrives at a difficulty-prioritizing weighted estimator (Xiong et al., 2025).

Distributional and risk-sensitive reinforcement learning. Modeling a whole reward or return distribution is not new, so it is worth stating precisely what TailRL does differently. Distributional reinforcement learning learns the return distribution as a critic, representing it categorically (Bellemare et al., 2017) or through quantiles (Dabney et al., 2018b,a; Bellemare et al., 2023), and propagates it with a distributional Bellman operator. Risk-sensitive methods optimize a tail functional of that distribution, most often the conditional value at risk (Rockafellar and Uryasev, 2000; Tamar et al., 2015; Chow et al., 2018; Clavier et al., 2022). Two differences separate these from TailRL. First, TailRL carries no critic and

performs no bootstrapping: the reward function supplies the reward in one step, and the tail probabilities appear only inside a policy-gradient weight computed from the group itself. Second, risk measures fix a single risk level, and a fixed level can make a learner blind to rare successes (Greenberg et al., 2022), whereas TailRL integrates the log tail-probability uniformly across every threshold.

Threshold decompositions of continuous rewards. Splitting a continuous reward into a family of binary threshold events is a classical device in supervised learning, and it is the direct ancestor of the decomposition in Figure 2. Cumulative-link models predict $P(y > j)$ at each threshold (McCullagh, 1980), and reductions to extended binary classification train one classifier per threshold and reassemble the prediction (Frank and Hall, 2001; Li and Lin, 2006). Neural versions inherit the same structure, with multiple binary outputs (Niu et al., 2016) and rank-consistency constraints across cuts (Cao et al., 2020; Shi et al., 2023). Patel et al. (2026) carry the idea into policy optimization for discrete rewards. TailRL is the policy-gradient counterpart for continuous rewards: the threshold τ plays the role of the ordinal cut, the indicator $\mathbb{1}_{\{r(x,z)>\tau\}}$ plays the role of the binary label, and the decomposition runs over a continuum rather than a finite set of cuts.

Reinforcement learning in the evaluated domains. Our four settings each connect to an established line of work. Bounding-box prediction is normally trained by direct supervision on coordinates, using an $L1$ term, a generalized intersection-over-union term, or the combination adopted by DETR (Rezatofighi et al., 2019; Carion et al., 2020), against which Section 6.1 compares a purely reward-driven policy (Russakovsky et al., 2015; Deselaers et al., 2010). GUI grounding has recently been posed as reinforcement learning with a dense point reward on top of vision-language backbones (Bai et al., 2025; Yang et al., 2026; Yuan et al., 2025; Liu et al., 2026) and evaluated on high-resolution professional interfaces (Li et al., 2025). Code generation has long used execution feedback as a reward signal (Le et al., 2022; Shojaee et al., 2023; Liu et al., 2023; Dou et al., 2024; Gehring et al., 2025; Zeng et al., 2025), and optimizing runtime rather than correctness requires both a corpus of slow programs and a deterministic timing oracle (Shypula et al., 2024; Binkert et al., 2011). That last setting also admits a reward-preserving shortcut, returning the input unchanged, which is an instance of the reward gaming characterized by Skalse et al. (2022). Section 6.4 uses it to separate an objective that settles for a dependable moderate outcome from one that keeps searching for a rare better one.

C Supporting Results for the TailRL Gradient Estimator

This section supplies the regularity conditions and proofs used in Section 4. We continue to work at a fixed input x and assume $r(x, z) \in [0, 1]$.

C.1 Regularity of the TailRL Gradient

The derivations of Section 4 rest on three operations. The score-function identity differentiates the tail-probability under an expectation over the policy's rollouts. The second operation moves a gradient from outside the threshold integral to inside it. The third swaps the order of the threshold integral and the rollout expectation. Each operation exchanges two limits, and each exchange is valid once the quantity being moved is bounded by something with a finite integral. We state one assumption for each operation, justify each in turn, and then record the lemma that licenses all three. Throughout, fix an input x, let the rollouts take values in a countable set, and let Θ be an open set of parameters.

Assumption 3 (Smooth policy)**.** *The support of $\pi_\theta(\cdot \mid x)$ is the same for every $\theta \in \Theta$, each probability $\pi_\theta(z \mid x)$ is differentiable in θ on Θ, and*

$$\sum_z \sup_{\theta \in \Theta} \left\| \nabla_\theta \pi_\theta(z \mid x) \right\| \; < \; \infty. \tag{35}$$

Assumption 3 is the classical hypothesis behind likelihood-ratio gradient estimators (Williams, 1992; Glynn, 1990): it is the interchange condition of L'Ecuyer (1995), and Mohamed et al. (2020) survey it as the standing assumption of the score-function estimator class. It holds by inspection for the policies of this paper: a softmax policy has a fixed finite support, its rollout probabilities are differentiable in θ, and a finite sum of continuous gradient norms is bounded on a bounded Θ.

Assumption 4 (Finite objective)**.** *The tail-likelihood is finite on Θ: $\int_0^1 \left| \log p_\theta(x, \tau) \right| d\tau < \infty$ for every $\theta \in \Theta$.*

Assumption 4 is the substantive assumption, and full support is what delivers it. A softmax policy assigns strictly positive probability to every rollout in its finite support, and the auto-regressive token policies of

this paper, softmax at every step with bounded generation length, are of this form. Let $z^\star$ be a rollout of maximal reward on x and normalize that maximum to 1. Every threshold $\tau \in [0, 1)$ is then cleared at least by $z^\star$, so the tail-probability is squeezed between two positive constants:

$$0 \;<\; \pi_\theta(z^\star \mid x) \;\le\; p_\theta(x,\tau) \;\le\; 1, \qquad \int_0^1 \big|\log p_\theta(x,\tau)\big|\, d\tau \;\le\; -\log \pi_\theta(z^\star \mid x) \;<\; \infty. \tag{36}$$

The bound is uniform over any bounded parameter set Θ, since $-\log \pi_\theta(z^\star \mid x)$ is continuous in θ. If no rollout attains reward 1, the same squeeze holds over thresholds below the largest attainable reward, and the threshold integral is read over that range. One consequence is used repeatedly below: the finiteness of the integral forces $p_\theta(x,\tau) > 0$ for almost every τ, so the logarithm and every ratio with $p_\theta(x,\tau)$ in its denominator are defined.

Assumption 5 (Bounded weighted score)**.** *There is a constant $C < \infty$ such that, for almost every τ,*

$$\sup_{\theta\in\Theta} \frac{\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\big[\mathbb{1}_{\{r(x,z)>\tau\}}\, \|S(x,z)\|_2\big]}{p_\theta(x,\tau)} \;\le\; C. \tag{37}$$

Assumption 5 follows from the bounded-reward assumption $\sup_{\theta\in\Theta}\sup_z \|S(x,z)\| \le C$ routine in policy-gradient convergence analyses (Papini et al., 2018; Zhang et al., 2020; Agarwal et al., 2021), since $\mathbb{E}_z\big[\mathbb{1}_{\{r(x,z)>\tau\}}\, \|S(x,z)\|\big] \le C\, p_\theta(x,\tau)$. Scores are bounded for the softmax policies of this paper whenever the logits have bounded gradients.

With the assumptions justified, the lemma states only its conclusion.

Lemma 6 (Regularity for the TailRL gradient)**.** *Under Assumptions 3 to 5, at every $\theta \in \Theta$, the population objective and every finite truncation are differentiable, and the score-function identity and every exchange of a gradient, threshold integral, and rollout expectation in Sections 4 and 4.2 are valid.*

Proof. The proof verifies the three operations in order, at an arbitrary $\theta \in \Theta$.

The score-function identity. Fix a threshold τ. The tail-probability is the sum of the policy probabilities over the rollouts that clear it,

$$p_\theta(x,\tau) = \sum_z \mathbb{1}_{\{r(x,z)>\tau\}}\, \pi_\theta(z \mid x). \tag{38}$$

By Assumption 3, every term of the differentiated series is bounded by the summable envelope of Eq. (35), so the gradient of the sum is the sum of the gradients.

$$\nabla_\theta p_\theta(x,\tau) = \sum_z \mathbb{1}_{\{r(x,z)>\tau\}}\, \nabla_\theta \pi_\theta(z \mid x). \tag{39}$$

Substituting $\nabla_\theta \pi_\theta(z \mid x) = \pi_\theta(z \mid x)\, S(x,z)$ in Eq. (39) turns the differentiated sum into the score-function identity

$$\nabla_\theta p_\theta(x,\tau) = \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\big[\mathbb{1}_{\{r(x,z)>\tau\}}\, S(x,z)\big]. \tag{40}$$

Moving the gradient inside the threshold integral. The triangle inequality applied to Eq. (40), followed by Eq. (37), bounds the log-gradient at every threshold:

$$\big\|\nabla_\theta \log p_\theta(x,\tau)\big\| = \frac{\|\nabla_\theta p_\theta(x,\tau)\|}{p_\theta(x,\tau)} \le \frac{\mathbb{E}_z\big[\mathbb{1}_{\{r(x,z)>\tau\}}\, \|S(x,z)\|\big]}{p_\theta(x,\tau)} \le C. \tag{41}$$

The objective is finite by Assumption 4, and the gradient of its integrand is bounded by the constant C, so the gradient moves inside the threshold integral:

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta; x) = \int_0^1 \frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}\, d\tau. \tag{42}$$

The truncations need nothing new. The order-N threshold multiplier is a finite geometric sum and is therefore bounded,

$$\frac{1-(1-p_\theta(x,\tau))^N}{p_\theta(x,\tau)} = \sum_{j=0}^{N-1} (1-p_\theta(x,\tau))^j \le N, \tag{43}$$

so the gradient of the truncated integrand is bounded by NC and the same argument differentiates $J^{(N)}_{\mathrm{TailRL}}$.

Swapping the threshold integral and the rollout expectation. Integrating Eq. (37) over the thresholds gives

$$\int_0^1 \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\left[\frac{\mathbb{1}_{\{r(x,z)>\tau\}}}{p_\theta(x,\tau)}\,\|S(x,z)\|\right]d\tau \le C < \infty, \tag{44}$$

so the integrand is absolutely integrable and Fubini's theorem permits taking the threshold integral and the rollout expectation in either order. The truncated multiplier never exceeds $1/p_\theta(x,\tau)$, so the truncated weighted score-function is bounded by the integrand of Eq. (44) and the same swap applies at every order N. □

The countability of the rollout space is inessential: for a policy with densities over a continuous rollout-space, the sum in Eq. (38) becomes an integral against a common reference measure and the proof is unchanged.

C.2 Derivation of the Rollout-Weight Forms

We derive the population rollout-weight form of the TailRL gradient and its order-N truncation, Eq. (25). Because the tail-likelihood averages per-level log-likelihoods, its gradient is an average of per-level likelihood gradients,

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta;x) = \int_0^1 \frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}\,d\tau = \mathbb{E}_{\tau\sim\mathrm{U}[0,1]}\left[\frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}\right]. \tag{45}$$

At a fixed threshold, the log-derivative trick expresses the pass-rate gradient as a score-function expectation:

$$\nabla_\theta p_\theta(x,\tau) = \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\left[\mathbb{1}_{\{r(x,z_j)>\tau\}}S(x,z)\right]. \tag{46}$$

Substituting into Eq. (45) writes the population gradient as a double expectation,

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta;x) = \mathbb{E}_{\tau\sim\mathrm{U}[0,1]}\left[\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\left[\frac{\mathbb{1}_{\{r(x,z)>\tau\}}}{p_\theta(x,\tau)}S(x,z)\right]\right], \tag{47}$$

and the integrability condition of Lemma 6 permits Fubini's theorem to exchange the two. The inner threshold integral then runs over exactly the thresholds cleared by the rollout, gives us Eq. (48) as restated below.

$$\nabla_\theta J_{\mathrm{TailRL}}(\theta;x) = \mathbb{E}_z\left[\left(\int_0^{r(x,z)}\frac{1}{p_\theta(x,\tau)}\,d\tau\right)S(x,z)\right], \tag{48}$$

The order-N member follows the same route. Differentiating Eq. (23) and summing the finite geometric series gives

$$\nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta;x) = \int_0^1 \left(\frac{1-(1-p_\theta(x,\tau))^N}{p_\theta(x,\tau)}\right)\nabla_\theta p_\theta(x,\tau)\,d\tau, \tag{49}$$

and substituting Eq. (46) and exchanging the threshold integral with the rollout expectation restricts the integral to the thresholds below the rollout's reward,

$$\nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta;x) = \mathbb{E}_z\left[\left(\int_0^{r(x,z)}\frac{1-(1-p_\theta(x,\tau))^N}{p_\theta(x,\tau)}\,d\tau\right)S(x,z)\right], \tag{50}$$

which is Eq. (25).

C.3 Best-of-k Decompositions and Probabilistic Interpretations

We prove the finite and population Best-of-k decompositions stated in Section 3. It also shows that threshold weighting is equivalent to rescaling the reward axis. We continue to work at a fixed input x and assume $r(x,z)\in[0,1]$. The gradient statements use the regularity conditions of Lemma 6.

C.3.1 Layer-Cake Form of Best-of-k

Let $z_1,\dots,z_k \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot\mid x)$ and write $R_i := r(x,z_i)$ and $M_k := \max_{1\le i\le k} R_i$. Since $M_k\in[0,1]$, the layer-cake identity gives

$$\mathbb{E}[M_k] = \int_0^1 \Pr(M_k>\tau)\,d\tau. \tag{51}$$

The maximum fails to clear τ exactly when all k rollouts fail it. By independence,

$$\Pr(M_k \le \tau) = (1 - p_\theta(x,\tau))^k . \tag{52}$$

Therefore,

$$\text{Best-of-}k(\theta;x) = \int_0^1 \left[1 - (1 - p_\theta(x,\tau))^k\right] d\tau. \tag{53}$$

C.3.2 Finite Harmonic Decomposition

Proposition 7 (Harmonic Best-of-k expansion of the finite objective)**.** *For every truncation order $T \ge 1$,*

$$J^{(T)}_{\text{TailRL}}(\theta;x) = \sum_{k=1}^{T} \frac{\textit{Best-of-}k(\theta;x) - 1}{k}. \tag{54}$$

Proof. Exchange the finite sum in Eq. (23) with the threshold integral:

$$\begin{aligned} J^{(T)}_{\text{TailRL}}(\theta;x) &= -\sum_{k=1}^{T} \frac{1}{k} \int_0^1 (1 - p_\theta(x,\tau))^k \, d\tau \\ &= \sum_{k=1}^{T} \frac{1}{k} \left(\int_0^1 \left[1 - (1 - p_\theta(x,\tau))^k\right] d\tau - 1 \right). \end{aligned} \tag{55}$$

The integral in parentheses is Best-of-$k(\theta;x)$ by Eq. (53). □

Proposition 8 (Harmonic Best-of-k expansion of the finite gradient)**.** *Under Lemma 6, for every $T \ge 1$,*

$$\nabla_\theta J^{(T)}_{\text{TailRL}}(\theta;x) = \sum_{k=1}^{T} \frac{1}{k} \nabla_\theta \textit{Best-of-}k(\theta;x). \tag{56}$$

Equivalently,

$$\nabla_\theta J^{(T)}_{\text{TailRL}}(\theta;x) = \int_0^1 \frac{1 - (1 - p_\theta(x,\tau))^T}{p_\theta(x,\tau)} \nabla_\theta p_\theta(x,\tau) \, d\tau, \tag{57}$$

where the ratio is interpreted by continuity as T when $p_\theta(x,\tau) = 0$.

Proof. Differentiating Eq. (54) term by term gives Eq. (56). Differentiating Eq. (23) under the integral gives

$$\begin{aligned} \nabla_\theta J^{(T)}_{\text{TailRL}}(\theta;x) &= \int_0^1 \left[\sum_{k=1}^{T} (1 - p_\theta(x,\tau))^{k-1}\right] \nabla_\theta p_\theta(x,\tau) \, d\tau \\ &= \int_0^1 \frac{1 - (1 - p_\theta(x,\tau))^T}{p_\theta(x,\tau)} \nabla_\theta p_\theta(x,\tau) \, d\tau, \end{aligned} \tag{58}$$

where the second equality uses the finite geometric-series identity. □

At $T = 1$, Eq. (54) gives

$$J^{(1)}_{\text{TailRL}}(\theta;x) = \text{Best-of-1}(\theta;x) - 1 = J_{\text{RL}}(\theta;x) - 1. \tag{59}$$

Also, since $0 \le (1 - p_\theta(x,\tau))^k \le 1$,

$$-H_T \le J^{(T)}_{\text{TailRL}}(\theta;x) \le 0, \qquad H_T := \sum_{k=1}^{T} \frac{1}{k}. \tag{60}$$

C.3.3 Population Harmonic Decomposition

Proof of Theorem 1. For each threshold, define

$$s_T(\tau) := \sum_{k=1}^{T} \frac{(1 - p_\theta(x,\tau))^k}{k}. \tag{61}$$

The sequence $s_T(\tau)$ is nondecreasing in T. The Maclaurin series gives the extended-real pointwise limit

$$\lim_{T\to\infty} s_T(\tau) = -\log p_\theta(x,\tau). \tag{62}$$

The monotone convergence theorem therefore gives

$$\begin{aligned}\lim_{T\to\infty} J^{(T)}_{\mathrm{TailRL}}(\theta;x) &= -\int_0^1 \lim_{T\to\infty} s_T(\tau)\,d\tau \\ &= \int_0^1 \log p_\theta(x,\tau)\,d\tau = J_{\mathrm{TailRL}}(\theta;x).\end{aligned} \tag{63}$$

Combining this limit with Eq. (54) yields

$$J_{\mathrm{TailRL}}(\theta;x) = \sum_{k=1}^{\infty} \frac{\text{Best-of-}k(\theta;x) - 1}{k} \tag{64}$$

whenever the population objective is finite.

For the gradients, define

$$w_T(p) := \frac{1-(1-p)^T}{p} = \sum_{j=0}^{T-1}(1-p)^j. \tag{65}$$

For every $p > 0$, $w_T(p) \uparrow 1/p$, and $0 \le w_T(p) \le 1/p$. By Lemma 6,

$$\frac{\|\nabla_\theta p_\theta(x,\tau)\|}{p_\theta(x,\tau)} \le C \tag{66}$$

for almost every threshold and some finite C. Dominated convergence applied to Eq. (57) gives

$$\begin{aligned}\lim_{T\to\infty} \nabla_\theta J^{(T)}_{\mathrm{TailRL}}(\theta;x) &= \int_0^1 \frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}\,d\tau \\ &= \nabla_\theta J_{\mathrm{TailRL}}(\theta;x).\end{aligned} \tag{67}$$

The finite gradient identity then identifies this limit with $\sum_{k=1}^{\infty} k^{-1}\nabla_\theta \text{Best-of-}k(\theta;x)$. □

C.3.4 Threshold Weighting as Reward Reparameterization

The uniform threshold distribution in Eq. (13) is the simplest member of a weighted family. Let $w : [0,1] \to (0,\infty)$ be a fixed density satisfying $\int_0^1 w(\tau)\,d\tau = 1$, and define

$$J_w(\theta;x) := \int_0^1 w(\tau)\log p_\theta(x,\tau)\,d\tau. \tag{68}$$

Proposition 9 (Threshold weighting is reward-axis rescaling)**.** *Define*

$$F_w(t) := \int_0^t w(s)\,ds, \qquad \widetilde{r}(x,z) := F_w(r(x,z))\,. \tag{69}$$

Then F_w is a strictly increasing map from $[0,1]$ to $[0,1]$, and

$$J_w(\theta;x) = \int_0^1 \log \Pr_{z\sim\pi_\theta(\cdot|x)}(\widetilde{r}(x,z) > u)\;du. \tag{70}$$

Thus, weighted TailRL on r is exactly uniform TailRL on the monotone rescaling $\widetilde{r} = F_w(r)$.

Proof. Since w is positive and integrates to one, F_w is strictly increasing with $F_w(0) = 0$ and $F_w(1) = 1$. For $u \in [0,1]$,

$$\begin{aligned}\Pr(\widetilde{r}(x,z) > u) &= \Pr(F_w(r(x,z)) > u) \\ &= \Pr\big(r(x,z) > F_w^{-1}(u)\big) = p_\theta\big(x, F_w^{-1}(u)\big)\,.\end{aligned} \tag{71}$$

Therefore,

$$\int_0^1 \log \Pr(\widetilde{r}(x,z) > u)\, du = \int_0^1 \log p_\theta\big(x, F_w^{-1}(u)\big)\, du$$
$$= \int_0^1 w(\tau) \log p_\theta(x,\tau) d\tau, \tag{72}$$

where the last equality uses the substitution $u = F_w(\tau)$. □

The proposition shows that choosing a threshold density is equivalent to choosing a monotone coordinate system for reward quality. Uniform weighting corresponds to the original normalized reward axis and introduces no additional function or hyperparameter. A nonuniform weighting may still be useful when a task provides a preferred reward scale, but selecting or learning that scale is outside the scope of this work.

C.3.5 Recovery of MaxRL on Binary Rewards

Proof of Eq. (17). If $r(x,z) \in \{0,1\}$, then for every $\tau \in [0,1)$,

$$\{r(x,z) > \tau\} = \{r(x,z) = 1\}. \tag{73}$$

Therefore $p_\theta(x,\tau) = q_\theta(x)$ for every nontrivial threshold, and

$$J_{\mathrm{TailRL}}(\theta;x) = \int_0^1 \log q_\theta(x)\, d\tau = \log q_\theta(x). \tag{74}$$

The same substitution in the finite objective gives

$$J^{(T)}_{\mathrm{TailRL}}(\theta;x) = -\sum_{k=1}^{T} \frac{(1-q_\theta(x))^k}{k}, \tag{75}$$

which is the order-T Maclaurin truncation of MaxRL. Finally, the maximum of k binary rewards equals one exactly when at least one rollout succeeds, so Best-of-$k(\theta;x)$ = Pass@$k(\theta;x)$. □

C.4 Harmonic Best-of-k Expansion of the Tail-Likelihood

This section proves that the truncated objective is a partial sum of Best-of-k objectives with harmonic coefficients, and that the tail-likelihood is the full series. Since the maximum of k rewards clears a threshold unless all k rollouts fail it, by layer cake identity,

$$\text{Best-of-}k(\theta;x) = \mathbb{E}\Big[\max_{1\le i\le k} r(x,z_i)\Big] = \int_0^1 \Big[1 - \big(1 - p_\theta(x,\tau)\big)^k\Big]\, d\tau. \tag{76}$$

Proposition 10 (Harmonic Best-of-k expansion of the objective)**.** *Fix an input x and a parameter θ. For every truncation order $T \ge 1$,*

$$J^{(T)}_{\mathrm{TailRL}}(\theta;x) = \sum_{k=1}^{T} \frac{1}{k}\Big(\text{Best-of-}k(\theta;x) - 1\Big), \tag{77}$$

and if $J_{\mathrm{TailRL}}(\theta;x) > -\infty$, the same identity holds for the full series,

$$J_{\mathrm{TailRL}}(\theta;x) = \sum_{k=1}^{\infty} \frac{1}{k}\Big(\text{Best-of-}k(\theta;x) - 1\Big). \tag{78}$$

Proof. We prove the finite identity first and obtain the series as its limit. Start from the definition of the truncated objective in Eq. (23) and exchange the finite sum with the threshold integral:

$$J^{(T)}_{\mathrm{TailRL}}(\theta;x) = -\int_0^1 \sum_{k=1}^{T} \frac{\big(1-p_\theta(x,\tau)\big)^k}{k}\, d\tau = -\sum_{k=1}^{T} \frac{1}{k} \int_0^1 \big(1-p_\theta(x,\tau)\big)^k\, d\tau. \tag{79}$$

Each integral on the right is one minus a Best-of-k value by Eq. (76), and substituting it in gives Eq. (77).

For the series, we let $T \to \infty$ on both sides of Eq. (77) and identify the two limits.

We first show the left side converges to the tail-likelihood. Since $J_{\mathrm{TailRL}}(\theta;x) > -\infty$, the tail-probability $p_\theta(x,\tau)$ is positive for almost every threshold. Fix such a τ, so that $0 \le 1 - p_\theta(x,\tau) < 1$. The Maclaurin series of the logarithm, evaluated at $1 - p_\theta(x,\tau)$, gives the pointwise limit of the partial sums:

$$\lim_{T\to\infty} \sum_{k=1}^{T} \frac{\big(1-p_\theta(x,\tau)\big)^k}{k} = \sum_{k=1}^{\infty} \frac{\big(1-p_\theta(x,\tau)\big)^k}{k} = -\log p_\theta(x,\tau). \tag{80}$$

The convergence is monotone: every added term $\big(1-p_\theta(x,\tau)\big)^{T+1}/(T+1)$ is nonnegative, so the partial sums only grow with T. The integrands in the definition Eq. (23) of $J^{(T)}_{\mathrm{TailRL}}$ are exactly these partial sums. Because they are nonnegative and increasing in T, the limit of their integrals is the integral of their limit, which is the monotone convergence theorem. Therefore

$$\lim_{T\to\infty} J^{(T)}_{\mathrm{TailRL}}(\theta;x) = -\int_0^1 \lim_{T\to\infty} \sum_{k=1}^{T} \frac{\big(1-p_\theta(x,\tau)\big)^k}{k}\, d\tau = \int_0^1 \log p_\theta(x,\tau)\, d\tau = J_{\mathrm{TailRL}}(\theta;x). \tag{81}$$

The right side needs no computation. For every T, the right side of Eq. (77) is the T-th partial sum of the series in Eq. (78). Its limit exists because it equals the left side at every T, and by Eq. (81) that limit is $J_{\mathrm{TailRL}}(\theta;x)$. This is Eq. (78). □

Every term of the series is nonpositive, since no Best-of-k value exceeds 1, so the partial sums decrease monotonically to the tail-likelihood: each truncation is an upper bound on J_{TailRL}, tightening as T grows. At the other end, $J^{(1)}_{\mathrm{TailRL}}(\theta;x) = \text{Best-of-1}(\theta;x) - 1 = J_{\mathrm{RL}}(\theta;x) - 1$, so the first member of the family is expected reward reinforcement learning up to an additive constant and shares its gradient.

Proposition 11 (Harmonic Best-of-k expansion of the gradient)**.** *Under Assumptions 3 to 5, for every $\theta \in \Theta$ and every $T \ge 1$,*

$$\nabla_\theta J^{(T)}_{\mathrm{TailRL}}(\theta;x) = \sum_{k=1}^{T} \frac{1}{k} \nabla_\theta \,\text{Best-of-}k(\theta;x) = \int_0^1 \frac{1-\big(1-p_\theta(x,\tau)\big)^T}{p_\theta(x,\tau)} \nabla_\theta p_\theta(x,\tau)\, d\tau. \tag{82}$$

Proof. Lemma 6 moves the gradient inside the threshold integral of Eq. (76), and the chain rule gives

$$\nabla_\theta \,\text{Best-of-}k(\theta;x) = k \int_0^1 \big(1-p_\theta(x,\tau)\big)^{k-1} \nabla_\theta p_\theta(x,\tau)\, d\tau. \tag{83}$$

Multiply Eq. (83) by $1/k$, sum over $k = 1,\dots,T$, and exchange the finite sum with the integral:

$$\sum_{k=1}^{T} \frac{1}{k} \nabla_\theta \,\text{Best-of-}k(\theta;x) = \int_0^1 \left[\sum_{k=1}^{T} \big(1-p_\theta(x,\tau)\big)^{k-1}\right] \nabla_\theta p_\theta(x,\tau)\, d\tau. \tag{84}$$

The bracket is a finite geometric sum, $\sum_{k=1}^{T}(1-p)^{k-1} = \big(1-(1-p)^T\big)/p$, which gives the right side of Eq. (82). The left equality is Lemma 6 again, differentiating Eq. (77) term by term. Every coefficient $1/k$ is positive, so the truncated gradient is a fixed, positively weighted combination of the Best-of-k gradients for $k = 1,\dots,T$. □

C.5 Unbiasedness of the finite rollout Estimator

This section proves Theorem 2, restated here in full.

Theorem 2 (Unbiasedness of the TailRL estimator, restated). The unbiased estimator of $\nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta;x)$ can be expressed as,

$$g^{(N)}_{\mathrm{TailRL}}(x) := \sum_{i=1}^{N} \omega(r_i)\, S_i, \qquad \omega(r_i) := \int_0^{r_i} \frac{d\tau}{\sum_{j=1}^{N} \mathbb{1}_{\{r_j > \tau\}}} \tag{85}$$

Under Assumptions 3 to 5, the estimator $g^{(N)}_{\mathrm{TailRL}}(x) := \sum_{i=1}^{N} \omega\big(r(x,z_i)\big)\, S(x,z_i)$ is unbiased for the order-N truncated gradient:

$$\mathbb{E}_{z_{1:N}}\Big[g^{(N)}_{\mathrm{TailRL}}(x)\Big] = \nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta;x). \tag{86}$$

Proof of Theorem 2.

The weight Eq. (85) for rollout i over its integration range clears the threshold, so its denominator is at least one. The first step extends the integral from $[0, r(x, z_i))$ to $[0, 1)$ with an integrand that vanishes beyond $r(x, z_i)$. We cap the denominator below at one; the cap is active only where the numerator already vanishes:

$$\omega\big(r(x,z_i)\big) = \int_0^{r_i} \frac{d\tau}{\sum_{j=1}^N \mathbb{1}_{\{r_j>\tau\}}} = \int_0^1 \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\, d\tau. \tag{87}$$

Every ratio from here on carries this capped denominator, so no expression in the proof is ever indeterminate. Multiplying by $S(x, z_i)$ with their weights $\omega(r(x, z_i))$ and summing over the group, we make use of the score-functions being independent of the thresholds, so we swap the order of integration and summation:

$$g^{(N)}_{\mathrm{TailRL}}(x) = \int_0^1 \frac{\sum_{i=1}^N \mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\, d\tau. \tag{88}$$

The chain of expectations. Take the expectation over the training group. Lemma 6 moves it inside the integral over thresholds τ, and linearity then moves it inside the finite sum over rollouts:

$$\begin{aligned}\mathbb{E}_{z_{1:N}}\Big[g^{(N)}_{\mathrm{TailRL}}(x)\Big] &= \int_0^1 \mathbb{E}_{z_{1:N}}\left[\left(\sum_{i=1}^N \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\right)\right] d\tau \\ &= \int_0^1 \sum_{i=1}^N \left(\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\right]\right) d\tau.\end{aligned} \tag{89}$$

By regularity condition Lemma 6, this integral is bounded. The next tool we use is the tower property of conditional expectation: for random variables X and Y on the same support,

$$\mathbb{E}_X[X] = \mathbb{E}_Y\big[\,\mathbb{E}_X[X \mid Y]\big], \tag{90}$$

where the inner expectation averages X with Y held at its realized value and the outer expectation averages the result over Y. We apply it with the substitution

$$X := \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}, \qquad Y := \{\mathbb{1}_{\{r(x,z_i)>\tau\}}\}_{i=1}^N, \tag{91}$$

the clearance pattern of the group at threshold τ; both are functions of the training group $z_{1:N}$, so every expectation below averages over $z_{1:N}$. The tower property Eq. (90) gives

$$\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\right] = \mathbb{E}_{z_{1:N}}\left[\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\,\middle|\, \{\mathbb{1}_{\{r(x,z_i)>\tau\}}\}_{i=1}^N\right]\right]. \tag{92}$$

Before taking the inner expectation, note one pointwise identity. An indicator equals its own square, so multiplying the fraction by a second copy of rollout i's indicator changes nothing:

$$\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\ \mathbb{1}_{\{r(x,z_i)>\tau\}} = \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}. \tag{93}$$

So X factors into the fraction times the conditional score:

$$X = \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\ \mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i), \qquad Y = \big\{\mathbb{1}_{\{r(x,z_j)>\tau\}}\big\}_{j=1}^N. \tag{94}$$

Conditioning on $\big\{\mathbb{1}_{\{r(x,z_j)>\tau\}}\big\}_{j=1}^N$ determines all N indicator functions, so the expression simplifies as:

$$\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\,\middle|\, Y\right] = \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\Big(1,\ \sum_{j=1}^N \mathbb{1}_{\{r(x,z_j)>\tau\}}\Big)}\ \mathbb{E}_{z_{1:N}}\big[\,\mathbb{1}_{\{r(x,z_i)>\tau\}}\, S(x,z_i)\,\big|\, Y\big]. \tag{95}$$

It remains to compute the conditional mean of the gated score. The rollouts are independent, so the only part of Y that constrains z_i is its own indicator, and we split on the two values it can take. If rollout i clears, z_i follows the policy restricted to the clearing set, $\pi_\theta(z \mid x)\,\mathbb{1}_{\{r(x,z)>\tau\}}/p_\theta(x,\tau)$; the gate is one everywhere on this support, so the mean is the clearing-rollout mean score, whose norm is at most C by Assumption 5. If rollout i fails, z_i is supported on $r(x,z) \le \tau$, where the gate is zero, so the mean is exactly the zero vector. Both branches are finite, and together they give:

$$\mathbb{E}_{z_{1:N}}\left[\,\mathbb{1}_{\{r(x,z_i)>\tau\}}\,S(x,z_i)\,\middle|\,Y\right] = \mathbb{1}_{\{r(x,z_i)>\tau\}}\,\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[S(x,z)\mid r(x,z)>\tau]\,. \tag{96}$$

Substituting Eq. (96) into Eq. (95) and applying Eq. (93) once more removes the extra indicator:

$$\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\,S(x,z_i)}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)}\,\middle|\,Y\right] = \frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)}\,\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[S(x,z)\mid r(x,z)>\tau]\,. \tag{97}$$

Finally the outer expectation of Eq. (92) wraps over Eq. (97). The clearing-rollout mean score-function is a fixed vector, so by linearity the outer expectation acts only on the fraction of indicators:

$$\begin{aligned}\mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}\,S(x,z_i)}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)}\right] &= \mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[S(x,z)\mid r(x,z)>\tau] \\ &\quad\times\ \mathbb{E}_{z_{1:N}}\left[\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)}\right].\end{aligned} \tag{98}$$

Substituting Eq. (98) into Eq. (89), summing over i, and recombining the N terms under one expectation by linearity leaves the sum of the fractions, which share one denominator and collapse to a single ratio of the same count:

$$\sum_{i=1}^{N}\frac{\mathbb{1}_{\{r(x,z_i)>\tau\}}}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)} = \frac{\sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}}{\max\left(1,\ \sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\right)} = \mathbb{1}_{\{\sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\ge 1\}}. \tag{99}$$

The last equality is checked outcome by outcome: if the number of rollouts attaining a reward above the threshold is ≥ 1, the cap is inactive and the ratio is $= 1$; if no rollout clears, the ratio is $0/1 = 0$. The expectation of this event indicator is its probability, and the N rollouts fail the threshold independently, each with probability $1 - p_\theta(x,\tau)$:

$$\mathbb{E}_{z_{1:N}}\left[\mathbb{1}_{\{\sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}\ge 1\}}\right] = 1 - (1-p_\theta(x,\tau))^N\,. \tag{100}$$

The remaining constant is evaluated by the definition of conditional expectation given an event, $\mathbb{E}[S \mid A] = \mathbb{E}[S\,\mathbb{1}_A]/\Pr(A)$, with $A = \{r(x,z) > \tau\}$ and $\Pr_{z\sim\pi_\theta(\cdot|x)}(A) = p_\theta(x,\tau)$; its numerator is the score-function identity Eq. (40):

$$\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}[S(x,z)\mid r(x,z)>\tau] = \frac{\mathbb{E}_{z\sim\pi_\theta(\cdot|x)}\left[\mathbb{1}_{\{r(x,z)>\tau\}}\,S(x,z)\right]}{p_\theta(x,\tau)} = \frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}. \tag{101}$$

Assembling the chain. Substituting Eqs. (98) to (101) into Eq. (89) yields the truncated gradient of Eq. (49):

$$\mathbb{E}_{z_{1:N}}\left[g^{(N)}_{\mathrm{TailRL}}(x)\right] = \int_0^1 \left(1-(1-p_\theta(x,\tau))^N\right)\frac{\nabla_\theta p_\theta(x,\tau)}{p_\theta(x,\tau)}\,d\tau = \nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta;x). \tag{102}$$

□

The proof identifies the estimator as MaxRL run at every threshold on one shared group of rollouts: wherever any rollout clears a threshold the cap is inactive and the integrand of Eq. (88) is the average score-function of the clearing rollouts, MaxRL's success-averaging rule for the threshold event, while at thresholds no rollout clears it is zero, MaxRL's rule for a group with no successes; its expectation carries MaxRL's order-N truncated weight from Eq. (34), and the binary case, where a single threshold carries all the mass, recovers MaxRL exactly (Corollary 14).

Loss reduction. The estimator is a sum over the group. A mean-reduced policy-gradient loss, which divides the group sum by N, must therefore use the coefficients $N\,\omega\big(r(x,z_i)\big)$, and after mean-centering $N\big(\omega\big(r(x,z_i)\big)-\bar{\omega}\big)$. A sum-reduced loss uses $\omega\big(r(x,z_i)\big)$, and after mean-centering $\omega\big(r(x,z_i)\big)-\bar{\omega}$.

C.6 Closed-Form Weights and Algorithm Correctness

The next proposition proves the recurrence in Eq. (27). Throughout, $z_1,\dots,z_N$ is the training group of Theorem 2, and $r_{(1)}\le\cdots\le r_{(N)}$ denote the group rewards $r(x,z_1),\dots,r(x,z_N)$ sorted increasingly, with $r_{(0)}:=0$. The weight Eq. (85) depends on a rollout only through its reward, so rollouts with equal rewards receive equal weights and $\omega\big(r_{(i)}\big)$ is well defined.

Proposition 12 (Closed-form empirical weights)**.** *For every $i=1,\dots,N$,*

$$\omega\big(r_{(i)}\big)=\sum_{k=1}^{i}\frac{r_{(k)}-r_{(k-1)}}{N-k+1}. \tag{103}$$

Equivalently, all weights follow the recurrence

$$\omega\big(r_{(i)}\big)=\omega\big(r_{(i-1)}\big)+\frac{r_{(i)}-r_{(i-1)}}{N-i+1},\qquad \omega\big(r_{(0)}\big)=0. \tag{104}$$

Proof. The empirical tail-probability distribution is a piecewise constant function. Fix $k\in\{1,\dots,N\}$ and a threshold $\tau\in[r_{(k-1)},r_{(k)})$. The rewards $r_{(k)},\dots,r_{(N)}$ are at least $r_{(k)}$ and therefore exceed τ, while the rewards $r_{(1)},\dots,r_{(k-1)}$ are at most $r_{(k-1)}\le\tau$ and fail, since clearance is strict:

$$\sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}=N-k+1\qquad\text{for every }\tau\in[r_{(k-1)},r_{(k)}). \tag{105}$$

The intervals $[r_{(k-1)},r_{(k)})$ for $k=1,\dots,i$ partition the integration range $[0,r_{(i)})$ of the weight, with tied rewards contributing empty intervals, so substituting Eq. (105) evaluates the integral interval by interval:

$$\omega\big(r_{(i)}\big)=\int_0^{r_{(i)}}\frac{d\tau}{\sum_{j=1}^{N}\mathbb{1}_{\{r(x,z_j)>\tau\}}}=\sum_{k=1}^{i}\int_{r_{(k-1)}}^{r_{(k)}}\frac{d\tau}{N-k+1}=\sum_{k=1}^{i}\frac{r_{(k)}-r_{(k-1)}}{N-k+1}, \tag{106}$$

which is Eq. (103); every denominator on the range is at least $N-i+1\ge 1$, so no ratio is ever indeterminate. Subtracting Eq. (103) at ranks i and $i-1$ leaves the single term $k=i$, which is the recurrence Eq. (104); the base case is the integral over the empty range $[0,r_{(0)})$. □

Scale of the weights. Every weight is nonnegative, since every gap $r_{(k)}-r_{(k-1)}$ is nonnegative. The total is the largest reward in the group: exchanging the order of the finite double sum counts each gap once per rank at or above it, and that count cancels its denominator, leaving a telescoping sum:

$$\sum_{i=1}^{N}\omega\big(r_{(i)}\big)=\sum_{k=1}^{N}\sum_{i=k}^{N}\frac{r_{(k)}-r_{(k-1)}}{N-k+1}=\sum_{k=1}^{N}\big(r_{(k)}-r_{(k-1)}\big)=r_{(N)}=\max_{1\le j\le N}r(x,z_j). \tag{107}$$

For rewards in $[0,1]$, the pre-centering advantages $N\,\omega\big(r(x,z_i)\big)$ therefore sum to $N\,r_{(N)}\le N$ and each lies in $[0,\,N\,r_{(N)}]$, so the update scale is controlled by the best reward observed in the group.

C.7 Effect of the Mean-Centering Baseline

Proposition 13 (The mean baseline lowers the truncation order by one)**.** *For $N\ge 2$, let*

$$\bar{\omega}:=\frac{1}{N}\sum_{i=1}^{N}\omega\big(r(x,z_i)\big),\qquad \hat{g}^{(N)}_{\mathrm{TailRL}}(x):=\sum_{i=1}^{N}\Big(\omega\big(r(x,z_i)\big)-\bar{\omega}\Big)\,S(x,z_i). \tag{108}$$

Then

$$\mathbb{E}_{z_{1:N}}\Big[\hat{g}^{(N)}_{\mathrm{TailRL}}(x)\Big]=\nabla_\theta J^{(N-1)}_{\mathrm{TailRL}}(\theta;x). \tag{109}$$

Proof. By Eq. (107), the weights of a group sum to its largest reward, so the centered estimator is the uncentered one minus a max-reward-weighted sum of scores:

$$\hat{g}^{(N)}_{\mathrm{TailRL}}(x)=g^{(N)}_{\mathrm{TailRL}}(x)-\frac{1}{N}\Big(\max_{1\le i\le N}r(x,z_i)\Big)\sum_{i=1}^{N}S(x,z_i). \tag{110}$$

The subtracted term is the score-function estimator of the Best-of-N gradient: the group is one draw from the product policy, whose score-function is $\sum_{i=1}^{N} S(x, z_i)$, so

$$\nabla_\theta \, \mathbb{E}_{z_{1:N}}\left[\max_{1\le i\le N} r(x, z_i)\right] = \mathbb{E}_{z_{1:N}}\left[\left(\max_{1\le i\le N} r(x, z_i)\right)\sum_{i=1}^{N} S(x, z_i)\right]. \tag{111}$$

Taking expectations in Eq. (110) and invoking Theorem 2 for the first term and Eq. (111) for the second yields

$$\mathbb{E}_{z_{1:N}}\left[\hat{g}^{(N)}_{\mathrm{TailRL}}(x)\right] = \nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta; x) - \frac{1}{N}\nabla_\theta \, \mathbb{E}_{z_{1:N}}\left[\max_{1\le i\le N} r(x, z_i)\right]. \tag{112}$$

It remains to identify the right side as the order-$(N-1)$ gradient. The expected maximum is the Best-of-N value, whose layer-cake form is Eq. (76) at $k = N$:

$$\mathbb{E}_{z_{1:N}}\left[\max_{1\le i\le N} r(x, z_i)\right] = \int_0^1 \left[1 - (1 - p_\theta(x, \tau))^N\right] d\tau. \tag{113}$$

Meanwhile, the definitions of two consecutive truncations differ in one term of the inner sum:

$$J^{(N)}_{\mathrm{TailRL}}(\theta; x) - J^{(N-1)}_{\mathrm{TailRL}}(\theta; x) = -\frac{1}{N}\int_0^1 (1 - p_\theta(x, \tau))^N \, d\tau. \tag{114}$$

Combining Eqs. (113) and (114) shows the two sides differ by a constant:

$$J^{(N)}_{\mathrm{TailRL}}(\theta; x) - J^{(N-1)}_{\mathrm{TailRL}}(\theta; x) = \frac{1}{N}\,\mathbb{E}_{z_{1:N}}\left[\max_{1\le i\le N} r(x, z_i)\right] - \frac{1}{N}. \tag{115}$$

The final term is constant in θ, so differentiating gives

$$\nabla_\theta J^{(N)}_{\mathrm{TailRL}}(\theta; x) - \frac{1}{N}\nabla_\theta \, \mathbb{E}_{z_{1:N}}\left[\max_{1\le i\le N} r(x, z_i)\right] = \nabla_\theta J^{(N-1)}_{\mathrm{TailRL}}(\theta; x). \tag{116}$$

Substituting Eq. (116) into Eq. (112) proves the result. □

C.8 Estimator-Level Recovery of MaxRL on Binary Rewards

Corollary 14 (Binary-reward recovery)**.** Suppose $r_i \in \{0, 1\}$ and let $M := \sum_{i=1}^{N} \mathbb{1}\{r_i = 1\}$. If $M > 0$, then every successful rollout receives weight $1/M$ and every unsuccessful rollout receives weight zero. Hence the uncentered TailRL estimator reduces to

$$g^{(N)}_{\mathrm{TailRL}}(x) = \frac{1}{M}\sum_{i:r_i=1} S_i, \tag{117}$$

which is the MaxRL estimator. After mean-centering, successful rollouts have weight $1/M - 1/N$ and unsuccessful rollouts have weight $-1/N$. If $M = 0$, all uncentered and centered weights are zero.

Proof. For binary rewards with $\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\} > 0$,

$$\omega(1) = \int_0^1 \frac{d\tau}{\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\}} = \frac{1}{\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\}}$$

and $\omega(0) = 0$. The uncentered result follows immediately. The mean weight is

$$\bar{\omega} = \left(\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\}\right)\frac{1}{\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\}} \cdot \frac{1}{N} = \frac{1}{N},$$

which gives the centered weights. When $\sum_{i=1}^{N} \mathbb{1}\{r_i = 1\} = 0$, every reward and hence every weight is zero. □

D TailRL on a General Reward Range

The main paper assumes rewards in $[0, 1]$ only to simplify notation. The same objective applies to any bounded reward range.

Suppose the reward function returns rewards in an arbitrary bounded range, $r(x, z) \in [r_{\min}, r_{\max}]$ with $r_{\min} < r_{\max}$. For thresholds $\tau \in [r_{\min}, r_{\max})$, define the tail-probability as before: $p_\theta(x, \tau) := \Pr_{z\sim\pi_\theta(\cdot|x)}[r(x, z) > \tau]$. The TailRL objective averages the log-tail-probability across this range:

$$J_{\text{TailRL}}^{[r_{\min}, r_{\max}]}(\theta; x) := \frac{1}{r_{\max} - r_{\min}} \int_{r_{\min}}^{r_{\max}} \log p_\theta(x, \tau)\, d\tau. \tag{118}$$

The factor $1/(r_{\max} - r_{\min})$ makes this the expected log-tail-probability under a threshold drawn uniformly from $[r_{\min}, r_{\max}]$. It also prevents the scale of the objective from depending on the units of the reward.

To recover the unit-interval objective, substitute $\tau = r_{\min} + (r_{\max} - r_{\min})\, u$ with $u \in [0, 1)$ and define the rescaled reward $\tilde{r} := (r - r_{\min})/(r_{\max} - r_{\min}) \in [0, 1]$. Then

$$J_{\text{TailRL}}^{[r_{\min}, r_{\max}]}(\theta; x) = \int_0^1 \log \Pr_{z\sim\pi_\theta(\cdot|x)}\big[\tilde{r}(x, z) > u\big]\, du = J_{\text{TailRL}}(\theta;\, x) \quad \text{for the reward } \tilde{r}. \tag{119}$$

Thus, applying TailRL to rewards in $[r_{\min}, r_{\max}]$ is equivalent to applying the unit-interval objective to $\tilde{r}$. Shifting or rescaling the reward only relabels its thresholds, so every result in the paper carries over directly.

The finite rollout estimator behaves in the same way. For sorted rewards, consecutive weights satisfy

$$\omega_{(i)} - \omega_{(i-1)} = \frac{r_{(i)} - r_{(i-1)}}{N - i + 1}, \tag{120}$$

so an affine rescaling of the rewards multiplies every weight by $r_{\max} - r_{\min}$. The normalization in Eq. (118) cancels this factor. Without the normalization, the update direction remains unchanged and only its magnitude is rescaled. The estimator can therefore operate on raw rewards from any bounded range without changing the algorithm.

E Connection to Ordinal Cross-Entropy

Threshold decompositions turn an ordinal or continuous target into a family of binary events, one event per threshold, and fit each event with a binary classifier (McCullagh, 1980; Frank and Hall, 2001; Li and Lin, 2006; Niu et al., 2016; Cao et al., 2020). This appendix makes the relation to TailRL exact. The population TailRL objective is the ordinal cross-entropy objective evaluated at the maximal target $r = 1$, and the finite rollout TailRL procedure is the finite sample estimation of that objective.

Fix an input x and recall the tail-probability $p_\theta(x, \tau) = \Pr_{z\sim\pi_\theta(\cdot|x)}(r(x, z) > \tau)$. For a target level $t \in [0, 1]$, define the threshold-decomposed cross-entropy between the point target t and the model's family of threshold events,

$$\text{CE}(t; \theta, x) := -\int_0^1 \Big[\, \mathbb{1}_{\{t>\tau\}}\, \log p_\theta(x, \tau) \;+\; \mathbb{1}_{\{t\le\tau\}}\, \log\big(1 - p_\theta(x, \tau)\big) \Big]\, d\tau. \tag{121}$$

For each fixed τ the bracket is the binary cross-entropy of the event $\{r > \tau\}$ against the label $\mathbb{1}_{\{t>\tau\}}$, and the integral weights all thresholds by the same uniform measure that defines the tail-likelihood.

Setting the target to the maximal reward $t = 1$ makes $\mathbb{1}_{\{t>\tau\}} = 1$ for every $\tau \in [0, 1)$, so the second term in Eq. (121) vanishes on a set of full measure and

$$\text{CE}(1; \theta, x) \;=\; -\int_0^1 \log p_\theta(x, \tau)\, d\tau \;=\; -\, J_{\text{TailRL}}(\theta; x). \tag{122}$$

Maximizing the tail-likelihood is therefore exactly minimizing the ordinal cross-entropy against the ideal target $r = 1$. When the reward is binary the tail-probability is constant in τ, the integral collapses to a single binary cross-entropy against the label 1, and Eq. (122) reduces to the MaxRL objective $\log q_\theta(x)$, consistent with Section 3.2.

The correspondence extends to the finite rollout procedure. The order-N objective $J_{\text{TailRL}}^{(N)}$ of Eq. (23) truncates the same integral at the resolution a group of N rollouts can support, and the estimator of

Theorem 2 is unbiased for its gradient using only the N sampled rewards. Training with TailRL on N rollouts is in this sense the finite sample estimation of the ordinal cross-entropy objective at target $r = 1$: the rollouts play the role of the samples from which the threshold events are estimated, and the truncation order grows with the sample size, recovering Eq. (122) as $N \to \infty$.

F Advantage Functions of GRPO, RLOO, and TailRL

All three methods share the same critic-free template: draw N rollouts, map their rewards to advantages, and form the update $\sum_{i=1}^{N} A_i \, S(x, z_i)$. They differ only in that map, which we record here. For convenience of notation, we write $r_i := r(x, z_i)$ for the reward of rollout z_i.

RLOO. RLOO subtracts the leave-one-out mean of the other rollouts,

$$A_i^{(\mathrm{RLOO})} = r_i - \frac{1}{N-1} \sum_{j \neq i} r_j. \tag{123}$$

The baseline is independent of rollout i, so the update is an unbiased estimator of the expected reward gradient. The advantage is affine in r_i with unit slope: a rollout is promoted by its raw margin over the rest of the group.

GRPO. GRPO standardizes within the group,

$$A_i^{(\mathrm{GRPO})} = \frac{r_i - \bar{r}}{\sigma(r) + \epsilon}, \qquad \bar{r} = \frac{1}{N} \sum_{j=1}^{N} r_j, \qquad \sigma(r)^2 = \frac{1}{N} \sum_{j=1}^{N} (r_j - \bar{r})^2. \tag{124}$$

Dividing by the group standard deviation makes the update invariant to affine rescaling of the reward. The same divisor applies to every rollout in the group, so it changes the size of an input's update rather than the relative weight of rewards within it.

TailRL. TailRL assigns weights proportional to inverse tail-probability's integral and centers it,

$$A_i^{(\mathrm{TailRL})} = \omega(r_i) - \bar{\omega}, \qquad \omega(r_i) = \int_0^{r_i} \frac{d\tau}{\sum_{j=1}^{N} \mathbb{1}_{\{r_j > \tau\}}}, \qquad \bar{\omega} = \frac{1}{N} \sum_{j=1}^{N} \omega(r_j). \tag{125}$$

The integrand is the reciprocal of the number of rollouts attaining a reward above a given reward threshold, so a level cleared by one rollout out of N contributes N times the weight per unit of reward of one cleared by all. The map is computed in closed form by the recurrence in Eq. (27) and reduces to the MaxRL advantages when the reward is binary (Appendix C.8).

G Pass@k and Best-of-k Empirical Calculation

Pass@k is used for binary rewards and measures the probability that at least one rollout succeeds. Best-of-k is used for continuous rewards and measures the expected reward of the highest-scoring rollout. Both metrics evaluate a policy when we sample k rollouts and keep the best one. The two metrics are identical for binary rewards, and both improve or remain unchanged as k increases. When we have K evaluation rollouts to estimate Pass@k or Best-of-k with $K > k$, we use all K rollouts to compute their unbiased estimators below.

G.1 Pass@k

If M of the K sampled rollouts succeed, the unbiased estimator of Chen et al. (2021) is

$$\widehat{\mathrm{Pass@}k} = 1 - \frac{\binom{K-M}{k}}{\binom{K}{k}}. \tag{126}$$

The ratio is the probability that a uniformly random size-k subset of the K rollouts avoids all M successes.

G.2 Best-of-k

Best-of-k is defined as in Eq. (76)

$$\text{Best-of-}k(\theta; x) = \mathbb{E}\left[\max_{1\le i\le k} r(x, z_i)\right] = \int_0^1 \left[1 - (1 - p_\theta(x, \tau))^k\right] d\tau, \tag{127}$$

the second form by the layer-cake identity applied to the maximum, which reduces to Pass@k when the reward is binary. Sorting the K sampled rewards increasingly as $r_{(1)} \le \cdots \le r_{(K)}$, the unbiased estimator weights each order statistic by the probability that it is the maximum of a uniformly random size-k subset:

$$\widehat{\text{Best-of-}k} = \sum_{i=k}^{K} \frac{\binom{i-1}{k-1}}{\binom{K}{k}} r_{(i)}. \tag{128}$$

H ImageNet Object Localization

We study single-object localization on ImageNet-scale data (Russakovsky et al., 2015). Given an image, the policy predicts a bounding box and receives its intersection-over-union (IoU) with the ground-truth box as a reward in $[0, 1]$.

The policy uses a pretrained ResNet-50 backbone (He et al., 2016) with four categorical heads, one for each box coordinate. Each head contains 50 uniformly spaced bins, and a rollout samples one bin from each head to form a box. All methods train for 30 epochs with Adam, a learning rate of 5×10^{-4} with warmup, and a batch size of 128. They differ only in how they compute the advantages. We train TailRL with $N \in \{16, 64, 256, 1024\}$ rollouts and separately optimize the exact population-level objective described in Appendix H.2.

For comparison, we also train supervised models directly on the ground-truth coordinates using MSE, L1, GIoU (Rezatofighi et al., 2019), and L1+GIoU losses (Appendix H.1).

We evaluate on held-out images using CorLoc@δ, mean IoU, and Best-of-1024 IoU. CorLoc@δ is the fraction of images whose greedy prediction exceeds IoU δ. Best-of-1024 IoU is the highest IoU among 1024 sampled boxes for each image. We report results over 3 seeds. The large-N gradient measurements in Figure 6 use one seed, as noted in the figure. Table 3 summarizes the full configuration.

Table 3: Training hyperparameters for ImageNet object localization.

Parameter	Value	Parameter	Value
Backbone	ResNet-50 (pretrained)	Policy head	4 heads × 50 bins
Reward	IoU, $[0, 1]$	Optimizer	Adam
Learning rate	5×10^{-4}, warmup	Batch size	128
Epochs	30	Rollouts N	$\{16, 64, 256, 1024\}$
Population budget	50^4	Baselines	GRPO, RLOO
Anchors	MSE, L1, GIoU, L1+GIoU	Eval samples	1,024 per image
CorLoc decoding	greedy	Seeds	3

H.1 Supervised Anchor Losses

Let $b = (x_1, y_1, x_2, y_2)$ denote the predicted box and $b^\star$ the ground-truth box. Both use normalized corner coordinates. The L1 loss measures the absolute error across the four coordinates:

$$\mathcal{L}_{\text{L1}}(b, b^\star) = \|b - b^\star\|_1 = \sum_{j=1}^{4} \left\|b_j - b_j^\star\right\|_1. \tag{129}$$

The GIoU loss (Rezatofighi et al., 2019) extends IoU with a penalty based on the smallest box enclosing both boxes. We use $|b|$ to denote the area enclosed by the bounding box b. Let $\mathrm{I} = |b \cap b^\star|$ be the intersection area, $\mathrm{U} = |b \cup b^\star|$ the union area, and c the smallest axis-aligned box enclosing both b and $b^\star$ and has an area $|c|$:

$$\mathcal{L}_{\text{GIoU}}(b, b^\star) = 1 - \underbrace{\frac{\mathrm{I}}{\mathrm{U}}}_{\text{IoU}} + \frac{|c| - \mathrm{U}}{|c|}. \tag{130}$$

The final term measures the empty space inside the enclosing box. Unlike IoU, it remains informative when the predicted and ground-truth boxes do not overlap. The combined loss follows DETR (Carion et al., 2020):

$$\mathcal{L}_{\text{L1+GIoU}}(b, b^\star) = \lambda_{\text{L1}}\mathcal{L}_{\text{L1}}(b, b^\star) + \lambda_{\text{GIoU}}\mathcal{L}_{\text{GIoU}}(b, b^\star). \tag{131}$$

We use the DETR weights $\lambda_{\text{L1}} = 5$ and $\lambda_{\text{GIoU}} = 2$. All three supervised anchors make one deterministic prediction and use no rollouts. Their greedy, mean, and Best-of-k outputs are therefore identical.

H.2 Computing the Population-Level Objective

Computing the population objective $J_{\text{TailRL}}(\theta; x) = \int_0^1 \log p_\theta(x, \tau) d\tau$ usually requires the unknown tail-probability $p_\theta(x, \tau)$. In this setting, we can compute it exactly because the policy has a finite output space. The policy predicts each of the four box coordinates with an independent categorical head over $K = 50$ uniformly spaced bins. A rollout samples one bin from each head, so its probability factorizes as

$$\pi_\theta(b \mid x) = \prod_{j=1}^{4} \pi_\theta^{(j)}(b_j \mid x), \qquad b \in \mathcal{B}, \quad |\mathcal{B}| = K^4 = 6{,}250{,}000. \tag{132}$$

Each box b has a deterministic reward $r(x, b)$ given by its IoU with the ground-truth box. We can therefore compute $(\pi_\theta(b \mid x), r(x, b))$ for every box in $\mathcal{B}$. The tail-probability at a threshold is the total probability of all boxes whose rewards exceed that threshold:

$$p_\theta(x, \tau) = \sum_{b \in \mathcal{B}} \pi_\theta(b \mid x) \mathbb{1}_{\{r(x,b) > \tau\}} \tag{133}$$

The finite set $\mathcal{B}$ produces a finite set of reward values. The function $\tau \mapsto p_\theta(x, \tau)$ is constant between consecutive reward values, so the threshold integral becomes a finite sum. This calculation introduces no discretization error beyond the original coordinate bins. In practice, we use the independence of the four heads to avoid explicitly enumerating all K^4 boxes. Alg. 1 forms the joint probabilities from the four marginal distributions and accumulates them over the sorted reward values.

Algorithm 1 Population-level TailRL objective and gradient for one image

Input: Head distributions $\pi_\theta^{(j)}(\cdot \mid x)$ over K bins, $j = 1, \ldots, 4$; ground-truth box $b^\star$

$\mathcal{B} \leftarrow \{1, \ldots, K\}^4$ `// every box the policy can emit`
for $b = (b_1, \ldots, b_4) \in \mathcal{B}$ **do**
  $\pi_\theta(b \mid x) \leftarrow \prod_{j=1}^{4} \pi_\theta^{(j)}(b_j \mid x)$ `// the heads are independent`
  $r(x, b) \leftarrow \text{IoU}(b, b^\star)$
end for
$u_1 < \cdots < u_m \leftarrow$ the distinct values of $\{\, r(x, b) : b \in \mathcal{B} \,\}$, sorted increasingly
for $i = 1, \ldots, m$ **do**
  $q_i \leftarrow \sum_{b\,:\, r(x,b) = u_i} \pi_\theta(b \mid x)$ `// reward distribution`
end for
$p_\theta(x, u_m) \leftarrow 0$ `// no box exceeds the largest attainable reward`
for $i = m - 1, \ldots, 1$ **do**
  $p_\theta(x, u_i) \leftarrow p_\theta(x, u_{i+1}) + q_{i+1}$ `//` $p_\theta(x, \tau) = p_\theta(x, u_i)$ `for` $\tau \in [u_i, u_{i+1})$
end for
$J_{\text{TailRL}}(\theta; x) \leftarrow \sum_{i=1}^{m-1} (u_{i+1} - u_i) \log p_\theta(x, u_i)$
$\nabla_\theta J_{\text{TailRL}}(\theta; x) \leftarrow \sum_{i=1}^{m-1} (u_{i+1} - u_i) \dfrac{\nabla_\theta\, p_\theta(x, u_i)}{p_\theta(x, u_i)}$

Return: $J_{\text{TailRL}}(\theta; x)$ and $\nabla_\theta J_{\text{TailRL}}(\theta; x)$, exact, with no sampling error

Substituting the exact $p_\theta(x, \tau)$ into Eq. (45) gives the gradient used by the population variant. This is an exact population gradient, not a large-N approximation. We obtain it by automatically differentiating through the probability accumulation because each tail-probability value is a differentiable function of the prediction head probabilities.

H.3 Additional Results

Training dynamics. Figure 13 shows training from three perspectives. The left panel reports final held-out CorLoc@0.5 across training rollout budgets N. TailRL improves steadily as N increases and

approaches the exact population objective. GRPO improves only slightly, while RLOO does not improve with additional rollouts. At every budget, both baselines perform worse than TailRL at its smallest reported budget.

The center panel shows the mean gradient norm by epoch at $N = 1024$. GRPO produces the largest gradients, while RLOO produces the smallest. However, GRPO also achieves the lowest final accuracy, showing that larger gradients do not explain the performance differences in Figure 7.The right panel shows each method's training loss over RL steps using a logarithmic step axis. Because the methods optimize different objectives, their loss values are not directly comparable. Nevertheless, all three losses flatten near zero late in training.

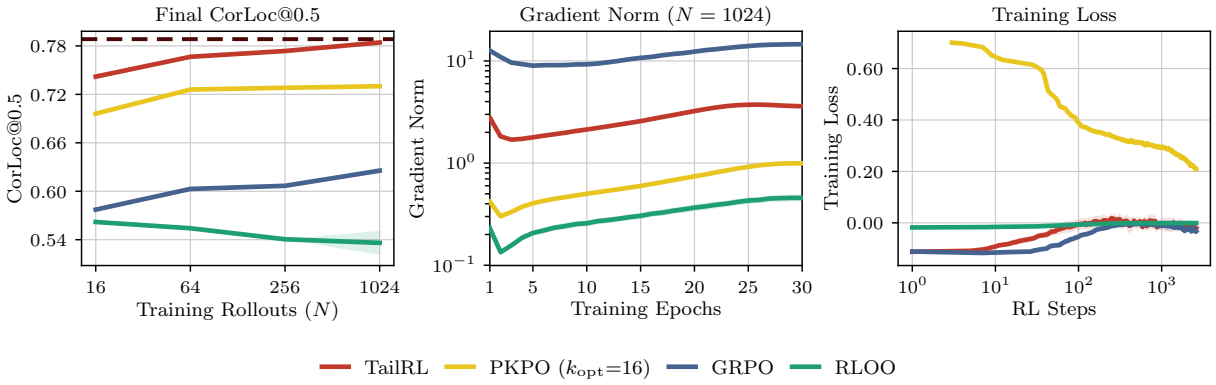

Figure 13: (ImageNet object localization) Left: final validation CorLoc@0.5 against the training rollout budget N for all three methods, with the population-level objective as a dashed reference. Center: mean gradient norm by epoch at $N = 1024$, log scale. Right: each method's training loss against RL steps, logarithmic step axis; each method optimizes its own surrogate objective. Means over the seeds of Figure 7.

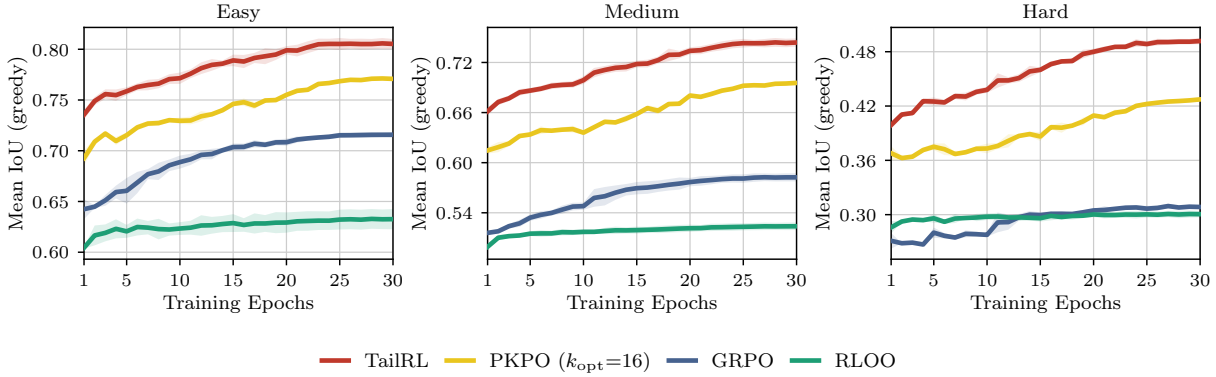

Figure 14: (ImageNet object localization) Mean IoU by difficulty band for the three methods at $N = 1024$. Bands partition the validation set by the area of the ground-truth box as a fraction of the image: easy is 0.30 to 0.70 (19,628 images), medium is 0.10 to 0.30 or 0.70 to 0.95 (19,977), and hard is below 0.10 or above 0.95 (10,395). Curves are means with standard-error bands over 3 seeds.

Figure 14 stratifies validation performance by difficulty band. The bands are defined by the area of the ground-truth box as a fraction of the image, since a box that fills a moderate part of the frame is far easier to localize than a very small or a nearly full-frame one. An image is easy when that area lies between 0.30 and 0.70, medium when it lies between 0.10 and 0.30 or between 0.70 and 0.95, and hard when it falls below 0.10 or above 0.95; the three bands partition the 50,000 validation images into 19,628, 19,977, and 10,395. The gap between TailRL and the stronger expected reward baseline widens as the band gets harder: roughly 0.09 IoU on the easy band, 0.16 on medium, and 0.18 on hard, where both baselines sit near 0.30 and TailRL reaches 0.49. Hard inputs are those on which high-reward rollouts are rare, and rarity is where the inverse-probability weighting of the tail-likelihood concentrates its effort, so the ordering of the gaps matches the mechanism the objective is built around. SR: pkpo td

H.4 The Effect of Binarizing the Reward

A continuous reward contains more information than binary reward. Binarizing it with a threshold removes this information by treating all outputs above or below the threshold as equally goor or bad. We test the

cost of this lost signal by binarizing IoU at $\{0.5, 0.75\}$ and training MaxRL on each binary reward. We compare both variants with TailRL trained on the original continuous reward across four rollout budgets, with all other settings held fixed.

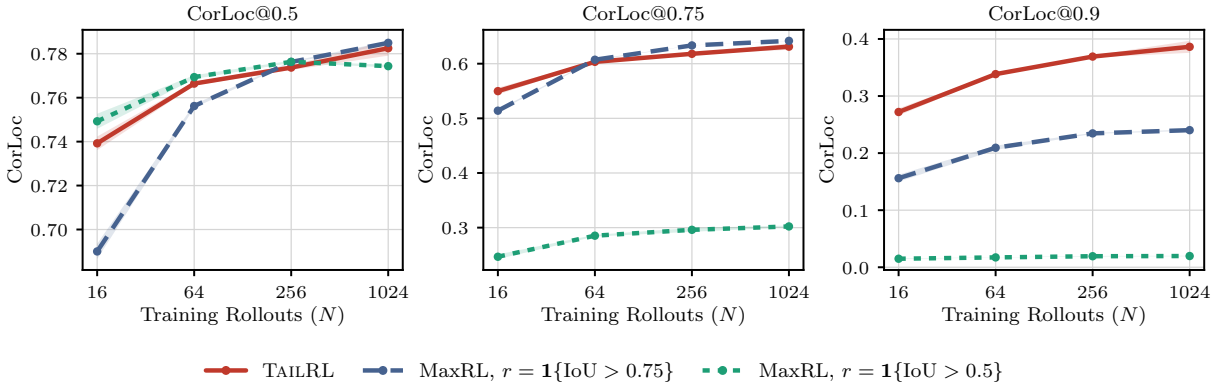

Figure 15: (ImageNet object localization) Effect of binarizing the IoU reward on that task of ImageNet object localization. We compare TailRL trained on continuous IoU with MaxRL trained on rewards binarized at IoU > 0.5 and IoU > 0.75. All methods perform similarly at the easiest evaluation threshold, but the binarized methods degrade at CorLoc above their binarization threshold.

Binarization produces policies that perform well near their chosen threshold but poorly at higher quality levels (Figure 15). At CorLoc@0.5, all methods reach approximately 0.78. At CorLoc@0.75, MaxRL trained with the 0.5 threshold reaches only 0.30, compared with approximately 0.63 for TailRL and MaxRL trained with the 0.75 threshold. At CorLoc@0.9, the gap widens further: TailRL reaches approximately 0.39, compared with 0.24 for the 0.75 threshold and 0.02 for the 0.5 threshold.

Once a rollout has an IoU above its binarization threshold, the binary reward no longer distinguishes a barely acceptable box from a nearly perfect one. This lost ordering produces both qualitative specialization to the chosen threshold and large quantitative losses above it. Increasing the rollout budget cannot recover information removed from the reward, so the binarized methods flatten while TailRL continues to improve at stricter thresholds. TailRL avoids choosing a specific threshold and learns from the full continuous reward, allowing one policy to perform well across all evaluated quality levels.

I Text-Maze Navigation

We study navigation in 17×17 gridworld mazes represented as text. Given a maze, the policy generates a sequence of movement tokens. The continuous reward measures whether the sequence is well formed, how close it ends to the goal, and how its length compares with the shortest path. A rollout that reaches the goal along a shortest path receives a reward of 1.

The policy is a 3M-parameter decoder-only transformer trained from scratch. We first pretrain it through supervised learning on 1.3M mazes with up to 16 annotated paths per maze. To test how each RL method behaves under different initial policy qualities, we retain seven pretraining checkpoints. Their held-out shortest-path rates range from 0.012%, or roughly one success in ten thousand attempts, to 0.83%.

Starting from each checkpoint, we train every method for 5,000 steps using $N = 16$ rollouts for each of 256 mazes per step. Training is fully on-policy, uses no KL regularization, and is repeated over three seeds. We report the shortest-path rate, defined as the probability that a single sampled rollout reaches the goal using an optimal-length path. We estimate this rate from 64 rollouts on each of 256 held-out mazes.

I.1 Dataset and Pretraining

Each maze is a 17×17 grid with the start and goal at opposite corners. We generate a perfect maze using Prim's algorithm and then remove a uniformly sampled 5 to 30% of its interior walls. Removing these walls creates alternative routes and varies the number of valid paths across mazes.

For pretraining, we pair each maze with up to 16 goal-reaching paths selected by Alg. 2. We first find the shortest-path length $L^\star$ using breadth-first search. A budgeted depth-first search then finds simple paths shorter than $\text{ub} = 60$. To prevent common path lengths from dominating the corpus, reservoir sampling retains at most four paths of each length.

We select paths that span a range of solution qualities. Each candidate path P receives a reward, where a shortest path receives 1 and longer paths receive smaller values. We divide reward $r \in (0, 1]$ into 16 equal intervals and select at most one path from each nonempty interval. When an interval contains multiple candidates, we select the path that overlaps least with those already chosen. This produces paths that vary in both length and spatial route. We train the policy on the resulting maze-path pairs using next-token prediction.

Algorithm 2 Selecting pretraining paths for one maze

Input: Maze m; length limit ub $= 60$; target number of paths $n_{\text{paths}} = 16$
1: $\mathcal{C} \leftarrow$ all simple start-to-goal paths found by breadth-first search with path length$<$ ub
2: Divide $\mathcal{C}$ into n_{paths} equal buckets according to path length
3: $\mathcal{S} \leftarrow \emptyset$
4: **for** each nonempty bucket $\mathcal{C}_t$ **do**
5: Select one path $P_t \in \mathcal{C}_t$
6: $\mathcal{S} \leftarrow \mathcal{S} \cup \{P_t\}$
7: **end for**
Return: A set $\mathcal{S}$ of paths with diverse lengths

I.2 Post-Training Configuration

The reinforcement-learning stage starts from the checkpoints above and uses the configuration of Table 4, which also lists the evaluation sampling.

Table 4: Training hyperparameters for Text-Maze navigation.

Parameter	Value	Parameter	Value
Maze	17×17, text	Policy	3M decoder (4L, 4H, 256)
Pretraining corpus	1.3M mazes, ≤ 16 paths	Path length	≤ 60
RL framework	verl, on-policy	PPO epochs	1, no KL
Prompts per step	256	Rollouts N	16 (sweep $\{4, 16, 64, 256\}$)
Learning rate	10^{-4}	Steps	5,000
Evaluation	64 samples $\times$ 1024 mazes	Seeds	3
Eval. rollouts per maze	64		

I.3 Reward Function

A rollout is parsed into a sequence of moves and replayed in the maze. Write $L^\star$ for the length of a shortest path from start to goal, L for the length of the rollout's path when it reaches the goal, and d for the breadth-first distance from the rollout's final cell to the goal. The reward is the sum of a progress term and a solution term,

$$r(x, z) = \underbrace{\tfrac{1}{2} \min\left(1, \tfrac{L^\star - d}{L^\star}\right)}_{\text{progress}} + \underbrace{\tfrac{1}{2} \min\left(1, \tfrac{L^\star}{L}\right) \mathbb{1}_{\{\text{goal reached}\}}}_{\text{solution}}, \tag{134}$$

with both terms clipped below at zero, and with $r(x, z) = 0$ whenever the rollout is malformed or walks into a wall. The progress term pays partial credit for ending closer to the goal than the start, so a rollout that never arrives is still ranked by how far it got. The solution term pays full credit only for a shortest path and decays as $L^\star/L$ when the policy reaches the goal by a longer route. The reward equals 1 exactly when the rollout reaches the goal along a shortest path, the event called shortest-path success in Section 6.2.

I.4 Task Representation and Prompt Template

A maze is serialized as a flat token sequence rather than as natural language, so the policy never sees English and cannot rely on pretrained language priors. The grid is written cell by cell with one token per cell, rows separated by a newline token, and the prompt ends at the marker that opens the path; the model then generates the path itself as a sequence of coordinate tokens. Each row of the 17×17 grid contributes seventeen cell tokens drawn from `WALL`, `PATH`, `START`, and `GOAL`, followed by `NEWLINE`; a

rollout is the continuation after PATH_START, a sequence of movement tokens closed by DONE. It is scored for well-formedness, for how close it ends to the goal, and for its length against the shortest path.

17 × 17 Maze Example Model Input and Output Format

Input:

```
<bos> GRID_START WALL WALL WALL ... WALL START PATH PATH ... NEWLINE ... GOAL WALL NEWLINE ... GRID_END
PATH_START
```

Output:

```
RIGHT RIGHT DOWN DOWN ... RIGHT DONE <eos>
```

Reward examples. Figure 8 shows four rollouts on one maze with the reward each receives. The two left paths never reach the goal and are scored by how far they get, which is what makes the reward continuous rather than binary. The third path reaches the goal but wanders, so it scores below a shortest path; only the rightmost path, which reaches the goal along a shortest path, receives reward 1.

I.5 Additional Results

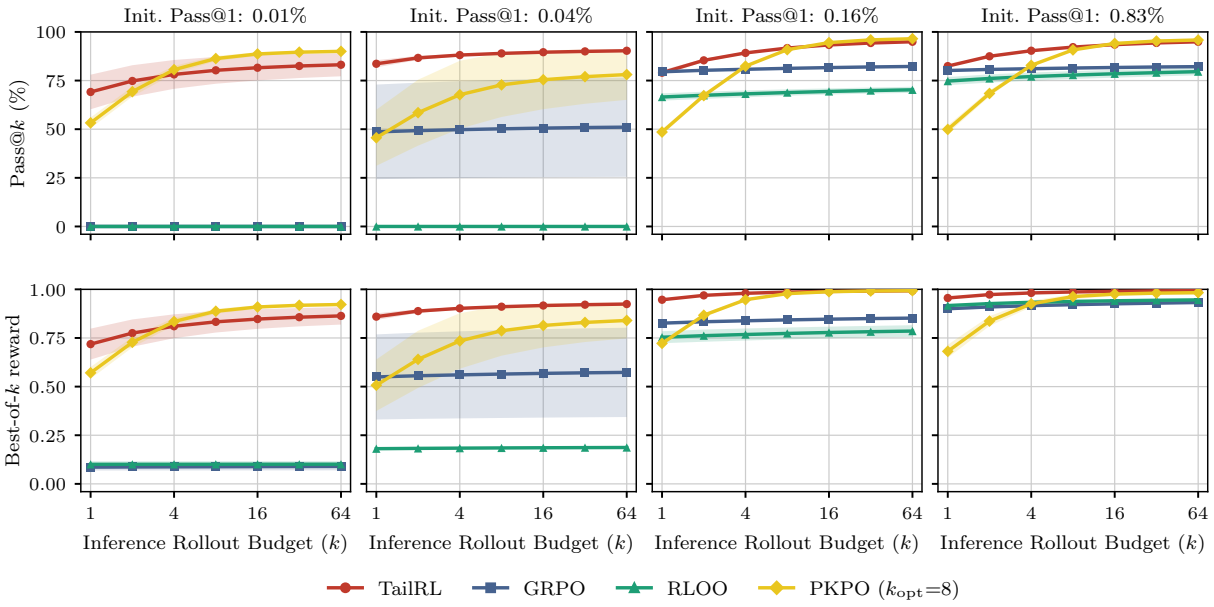

Figure 16: (Text-Maze) Inference-time scaling on Text-Maze for four initial policies, labeled by their shortest-path success before reinforcement learning. The rows report Pass@k, the probability that any of k rollouts reaches the goal along a shortest path, and Best-of-k reward is the expected maximum reward attained by the policy among k rollouts.

Effect of the inference rollout budget. We evaluate how the learned rollout distributions respond to additional inference rollouts. Figure 16 reports Pass@k and Best-of-k reward as k increases. At small k, policies trained with the three methods can appear similar. As k increases, both metrics improve more rapidly for TailRL than for GRPO or RLOO. Despite GRPO and RLOO improving their Pass@1 after RL post-training on policy initializations with good coverage, TailRL still outperforms them at test-time scaling. Evaluation at $k = 1$ therefore understates the differences among the learned rollout distributions. The increasing separation is consistent with the Best-of-k decomposition in Section 3: TailRL assigns greater probability to high-reward rollouts, making them more likely to be discovered as k grows.

Effect of the training rollout budget. To study the effect of the training rollout budget, we vary N while holding fixed an initial policy with a shortest-path success rate of approximately 0.02% (Figure 17). For TailRL, both Pass@k and Best-of-k reward increase sharply between $N = 4$ and $N = 16$. At $N = 4$, TailRL remains near the same floor as the expected reward baselines. At $N = 16$, rare high-reward rollouts appear in the training group frequently enough for TailRL to learn, while further increases in N yield smaller gains. GRPO and RLOO do not obtain a comparable benefit from increasing N. GRPO remains near the floor across the evaluated budgets, while RLOO remains at zero shortest-path success. Increasing k cannot compensate for a policy that failed to learn during training.

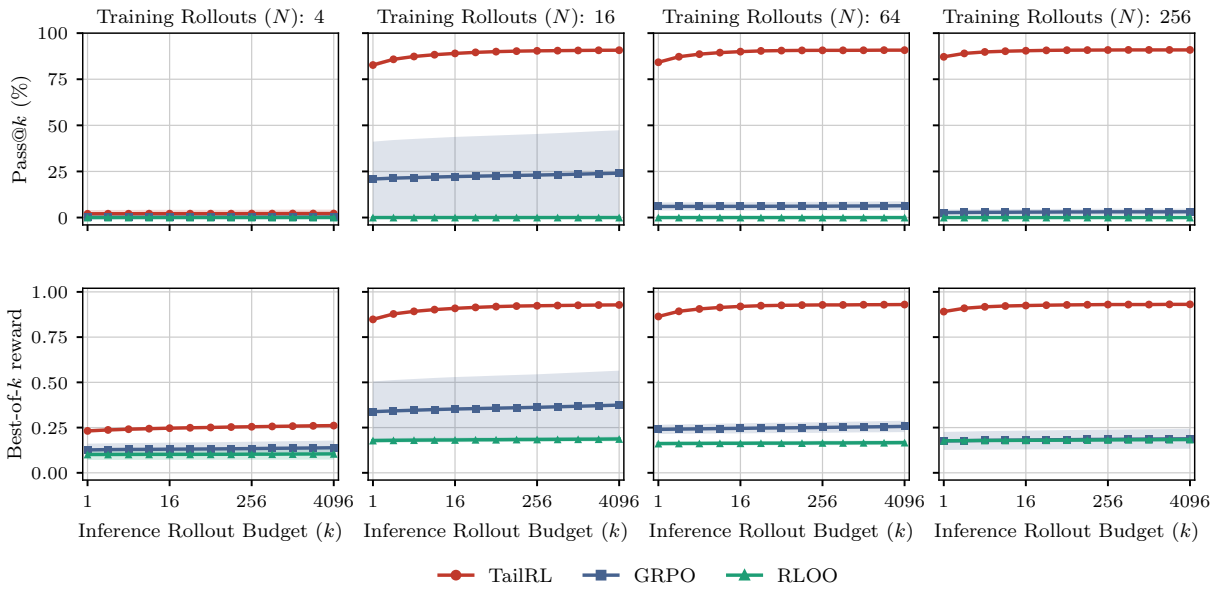

Figure 17: (Text-Maze) Training rollout budget sweep from an initial shortest-path success rate of 0.024%, with one column per N. Rows show Pass@k and Best-of-k reward against the inference budget. TailRL converts additional training rollouts into learning between $N = 4$ and $N = 16$, while the expected reward baselines gain little.

Training dynamics. Figure 18 reports policy entropy and mean generated path length during training from four of the seven pretraining checkpoints at $N = 16$. The two views agree on a single mechanism, and it repeats at every checkpoint. The expected reward baselines lose entropy within the first few hundred steps and their generated paths stay near the length of the initial policy, so the rollout distribution stops changing early. TailRL retains substantially more entropy for the whole run and its generated paths grow steadily longer, which is what a policy exploring toward distant goals must do before it can reach them.

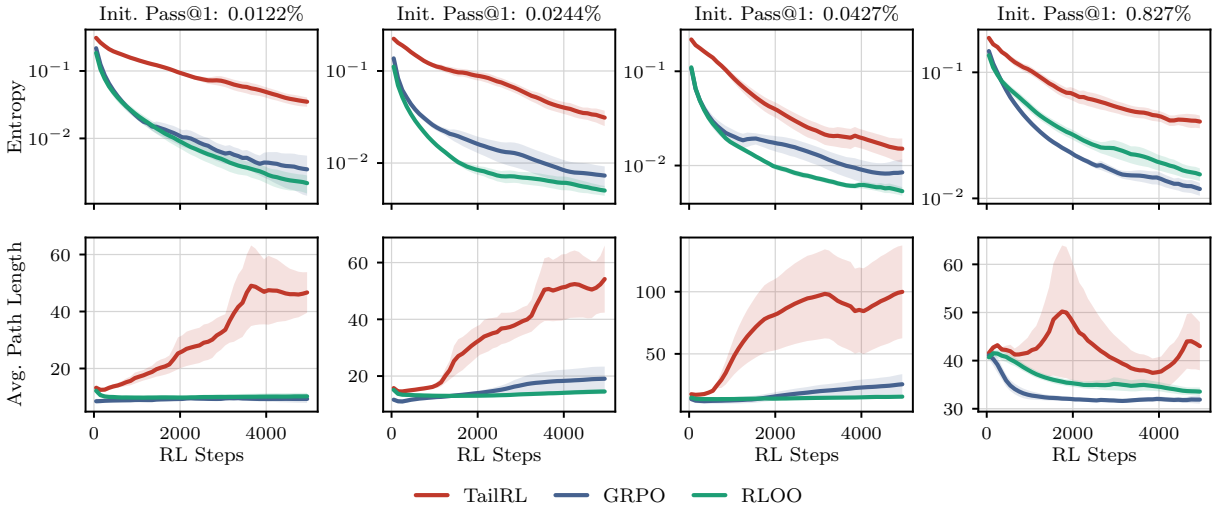

Figure 18: (Text-Maze) Training dynamics at $N = 16$ from four pretraining checkpoints, one per column. Top row: policy entropy by training step, log scale. Bottom row: mean generated path length by training step. All curves are exponential-moving-average smoothed.

J GUI Grounding

J.1 Model and Data

We fine-tune the 3B and 7B versions of Qwen2.5-VL (Bai et al., 2025) on GTA1 (Yang et al., 2026), which contains 70,528 screenshot-instruction pairs. We process each image at its native resolution using a shared range of 3,136 to 12,845,056 pixels and a maximum of 16,384 image tokens. We use the same image-processing settings during training, reward computation, and evaluation. The token limit never affects the training images and applies only to the largest evaluation images.

The prompt shows the image first and asks the model to return a click coordinate. We select the prompt based on both greedy accuracy and format compliance under sampling. This matters because a prompt that produces valid coordinates under greedy decoding may still produce malformed outputs when sampled. Both model sizes use the same prompt.

J.2 Reward

We use the dense point reward from SE-GUI (Yuan et al., 2025). Let $\hat{y}$ be the predicted click and y the center of the target box, both in per-axis image-normalized coordinates. We define $d = \|\hat{y} - y\|$ and let $d_{\max}$ be the largest distance from y to an image corner. The total reward is

$$r(x, z) = \underbrace{\mathbb{1}_{\{\hat{y} \in \text{box}\}}}_{\text{inside}} + \underbrace{\left(1 - (d/d_{\max})^2\right) \mathbb{1}_{\{d \le 1\}}}_{\text{proximity}} + \underbrace{\tfrac{1}{2}\, \mathbb{1}_{\{\hat{y} \text{ parses}\}}}_{\text{format}} \in [0, 2.5]. \tag{135}$$

The first term rewards clicks inside the target box. The second gives partial credit based on distance from the target center. Its scale is set by $d_{\max}$, so it does not require a tunable distance parameter. The final term adds 0.5 when the predicted coordinate can be parsed.

The SE-GUI paper defines the proximity term as $(1 - d/d_{\max})^2$, while its released code uses $1 - (d/d_{\max})^2$ together with the indicator $\mathbb{1}_{\{d \le 1\}}$. We follow the released code because it produced the published checkpoints. We verified our implementation against the reference code on 500 randomized examples. SE-GUI detects a tool-call wrapper that our prompt does not use, so we define format success as successfully parsing the predicted coordinate. We apply this rule identically to all methods.

J.3 Training Protocol

Training is fully on-policy, with one optimizer update per batch. Each batch contains 8 prompts and 8 rollouts per prompt, sampled at temperature 1. We use no KL regularization. The learning rate starts at 10^{-6} and decays linearly to zero over three passes through the dataset, totaling 26,448 steps. All methods use the same fixed training horizon.

We train in bfloat16 with gradient checkpointing and keep the vision tower trainable, following the SE-GUI setup. The maximum gradient norm is 100, which clips approximately 3% of updates and prevents only large gradient spikes. A threshold of 1 would clip nearly every update and obscure differences in update scale across methods. The 3B and 7B models use the same configuration. We run one seed per method. Table 5 summarizes the full configuration.

Table 5: Training hyperparameters for GUI grounding.

Parameter	Value	Parameter	Value
Models	Qwen2.5-VL-3B / 7B	Vision tower	trained
Corpus	GTA1, 70,528 pairs	Prompt	image-first point
Reward	SE-GUI + 0.5· format	Reward range	$[0, 2.5]$
Pixel window	3,136 – 12,845,056	Token cap	16,384
Prompts per step	8	Rollouts per prompt	8
Updates per step	1 (on-policy, no KL)	Rollout temp	1.0
Learning rate	10^{-6}, linear to 0	Steps	26,448 (3 epochs)
Precision	bf16, grad ckpt	Grad-norm guard	100
Eval temp / top-p	0.6 / 0.95	Eval samples	4,096 per item
Confidence intervals	1,000× bootstrap	Seeds	1

J.4 Prompt Template

Both scales use the same prompt, selected by measurement at 3B and then frozen. It is image-first and asks for a bare pixel coordinate, with no tool-call wrapper and no chain of thought, so that a rollout parses under sampling as reliably as under greedy decoding:

```
System: You are an expert UI element locator. Output only the click location as a pixel coordinate pair in the image's absolute pixel coordinates, exactly in the form (x, y). For elements with area, return the center point. Output nothing else.
User: <image> Grounding instruction is: {instruction}. Where should you click to do this? Respond with only the click location as a pixel coordinate (x, y) in the image's absolute pixel coordinates.
```

J.5 On-Policy Implementation

Training is strictly on-policy: one optimizer update per batch of rollouts, no importance-sampling correction, no clipping, and no KL penalty to a reference policy. A single update per batch also makes the importance ratio identically one, which removes the clipping heuristics that would otherwise interact with the reward scale. Recent on-policy work on GUI grounding adopts the same setting (Liu et al., 2026).

J.6 Evaluation Protocol

We evaluate zero-shot performance on all 1,581 items in ScreenSpot-Pro (Li et al., 2025), a benchmark of professional high-resolution software interfaces. We use two evaluation protocols. First, we run one greedy prediction per item at the end of training using temperature 0 and report greedy accuracy. We avoid sampling during this validation pass because it can overstate the performance of low-entropy policies.

Our primary evaluation uses the saved checkpoints. We draw 4,096 samples per item using temperature 0.6, nucleus sampling with nucleus probability= 0.95, and no top-k filtering. These settings are fixed across all methods and follow the Pass@k evaluation protocol (Chen et al., 2021). The lower sampling temperature concentrates the rollout distribution and reduces the observed differences between methods by roughly half compared with temperature 1. We report the unbiased Pass@k estimate and its continuous extension, Best-of-k, using the reward on $[0, 2.5]$. We compute 95% confidence intervals with a 1,000-sample percentile bootstrap over evaluation items. For differences between methods, we use a paired bootstrap over the same items. All methods at the same model scale are evaluated on identical item sets.

Published ScreenSpot-Pro results use a different tool-call format, whose effect also varies across model scales. Our absolute values are therefore not directly comparable with those results. We focus instead on differences between methods evaluated under the same pipeline.

J.7 Additional Results

Category-wise inference scaling. Figures 19 and 20 break the final-checkpoint inference-scaling ladders down by ScreenSpot-Pro task category, at both model scales. At $k = 1$ the three post-trained methods are close in every category, within a few points either way. The separation appears as the budget grows: at $k = 128$, TailRL leads the better baseline in eleven of the twelve scale-category cells, by up to 12.0 points on Dev at 3B and 11.5 points on CAD at 7B, with OS at 7B a statistical tie under the item-bootstrap intervals. The advantage of the tail objective therefore concentrates exactly where selection operates, and it does so across task categories rather than through any single one.

Pass@k and Best-of-k through training. Figures 21 and 22 evaluate every checkpoint rather than only the final one, at four inference rollout budgets. Two facts hold throughout training rather than only at its end. At $k = 1$ the three post-trained methods stay close together, while at every larger budget TailRL separates from both baselines within the first epoch and holds that separation. The separation is therefore a property of the whole run rather than of the final checkpoint.

Reward components. Figure 23 separates the training reward into its parts. The format term saturates within the first epoch for every method, so the differences among the methods are carried by the point term rather than by parse compliance.

Training dynamics. Figure 24 reports gradient norm and policy entropy at both scales. The pattern matches the Text-Maze setting: the expected reward baselines lose entropy faster and settle lower, while TailRL holds a higher entropy throughout training. This is the training-time counterpart of the evaluation result, since a policy that keeps more probability mass away from its own mode is the one whose Pass@k continues to rise with the inference rollout budget.

K Code runtime optimization

We use the PIE corpus of competitive-programming solutions. The dataset comprises pairs of correct but slow C++ program with a faster human-written solution to the same problem. We do not make use of

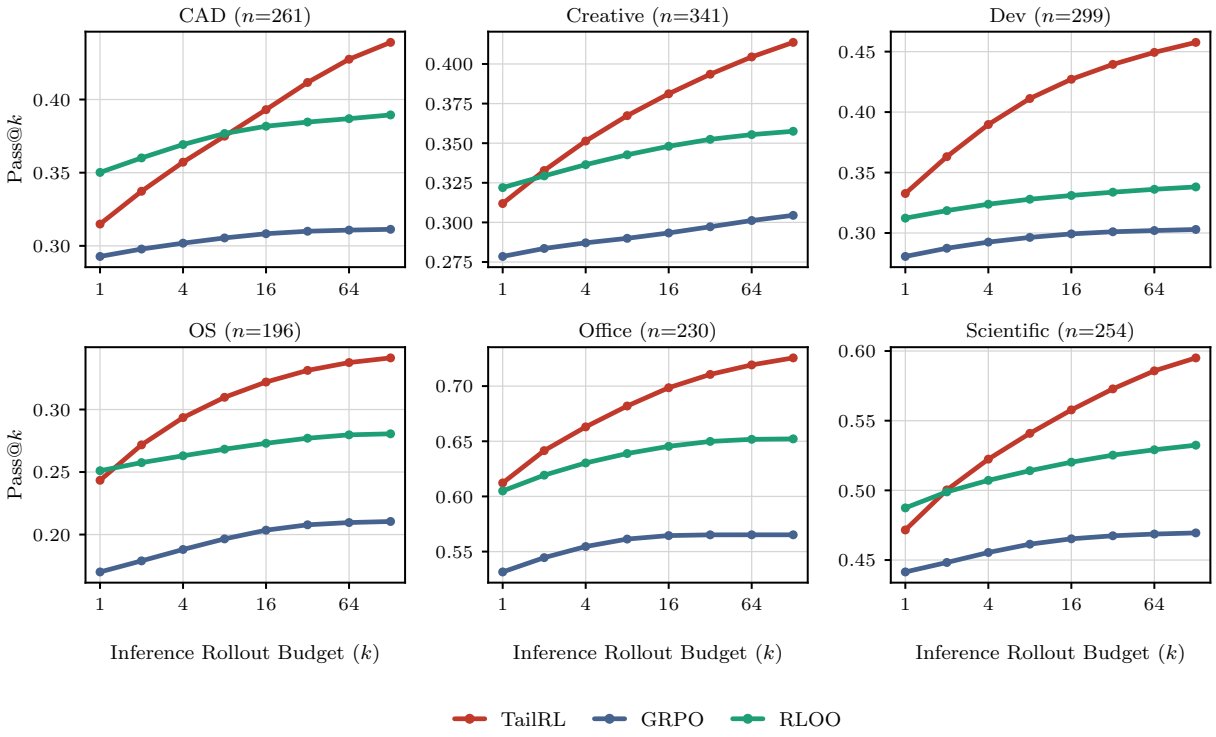

Figure 19: (GUI-grounding) Category-wise inference scaling on ScreenSpot-Pro dataset, Qwen2.5-VL-3B at the final checkpoint. Pass@k against the inference rollout budget within each task category, 512 samples per item.

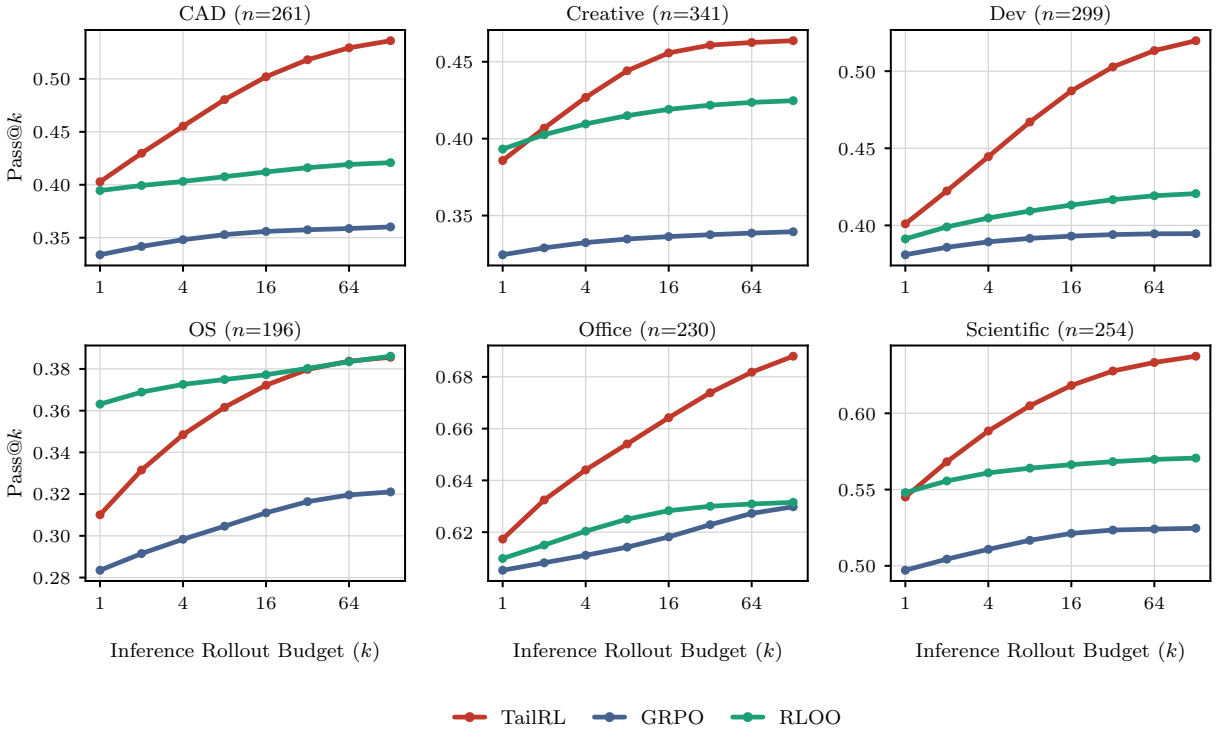

Figure 20: (GUI-grounding) Category-wise inference scaling on ScreenSpot-Pro, Qwen2.5-VL-7B at the final checkpoint. Pass@k against the inference rollout budget within each task category, 512 samples per item.

the faster written solutions during training or evaluation. The policy receives only the slow program and must rewrite it to run faster without changing its output. It returns the rewritten program as a single fenced C++ block.

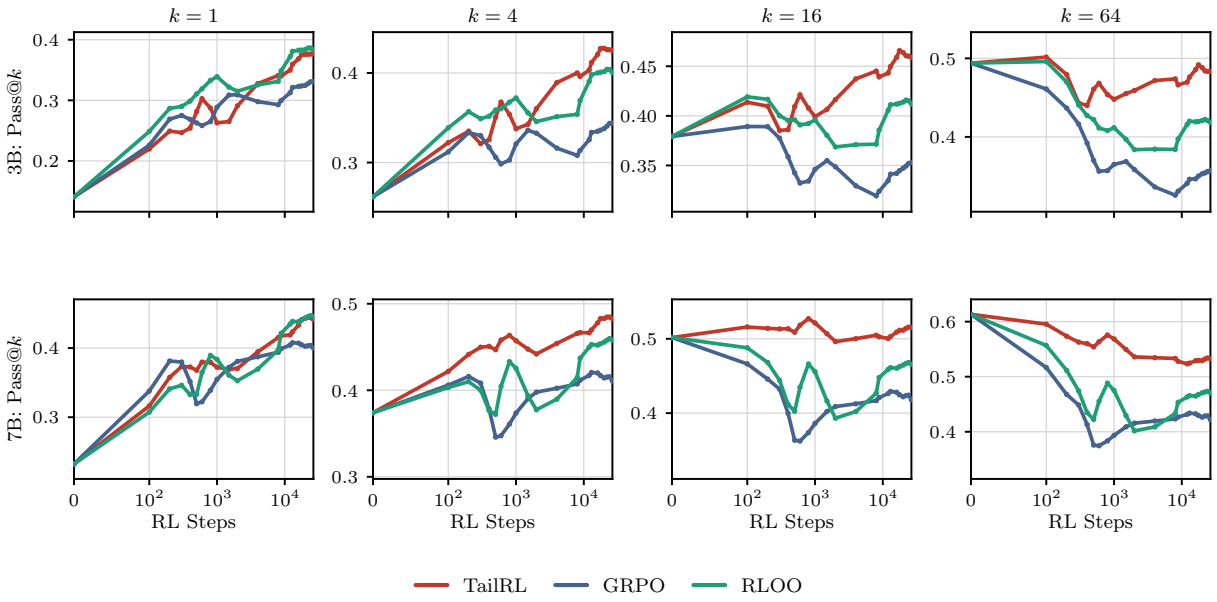

Figure 21: (GUI-grounding) ScreenSpot-Pro Pass@k against training epoch at four inference rollout budgets. Top row: Qwen2.5-VL-3B. Bottom row: Qwen2.5-VL-7B. Every checkpoint is evaluated with 512 samples per item.

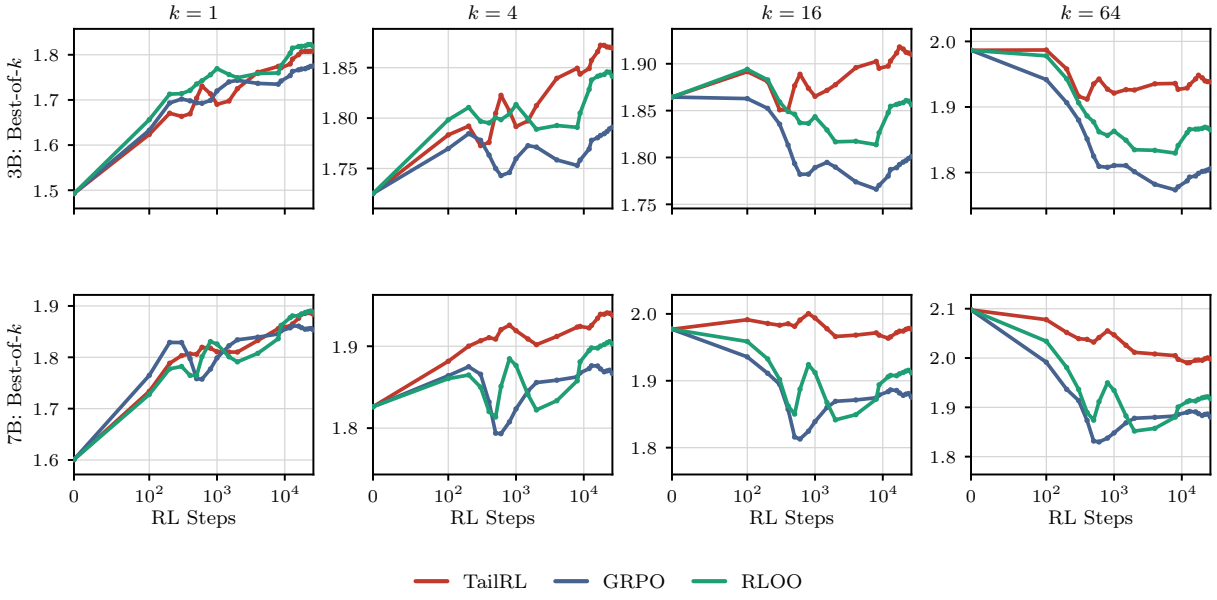

Figure 22: (GUI-grounding) Best-of-k reward against training steps. Top row: Qwen2.5-VL-3B. Bottom row: Qwen2.5-VL-7B. Every checkpoint is evaluated with 512 samples per item.

K.1 Reward

We evaluate each rollout for correctness before measuring its speed. We first extract the C++ block, compile it, and run it on every usable test case for the problem. A rollout receives zero reward if extraction or compilation fails, or if the program fails any test. Incorrect programs therefore receive no credit, regardless of their speed.

A correct rollout receives its speedup over the original program:

$$r(x, z) = \frac{c_{\text{src}}}{c(x, z)} \mathbb{1}_{\{\text{every test passes}\}} \tag{136}$$

where $c(x, z)$ is the cost of the rewritten program and c_{src} is the cost of the original program.

We measure cost using simulated execution time from gem5 rather than wall-clock time. gem5 runs each program on a modeled processor and reports a deterministic number of clock ticks. The same program

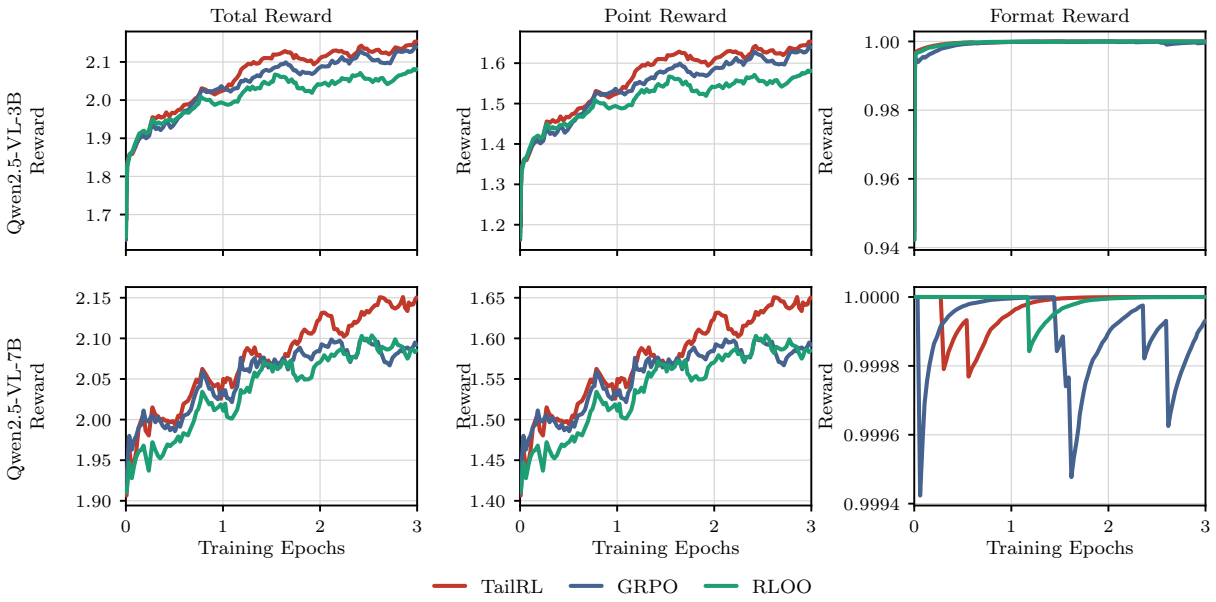

Figure 23: (GUI-grounding) Training reward by epoch, decomposed. Left: total reward on $[0, 2.5]$. Center: the point term. Right: the format term. Top row: Qwen2.5-VL-3B. Bottom row: Qwen2.5-VL-7B. All curves are exponential-moving-average smoothed.

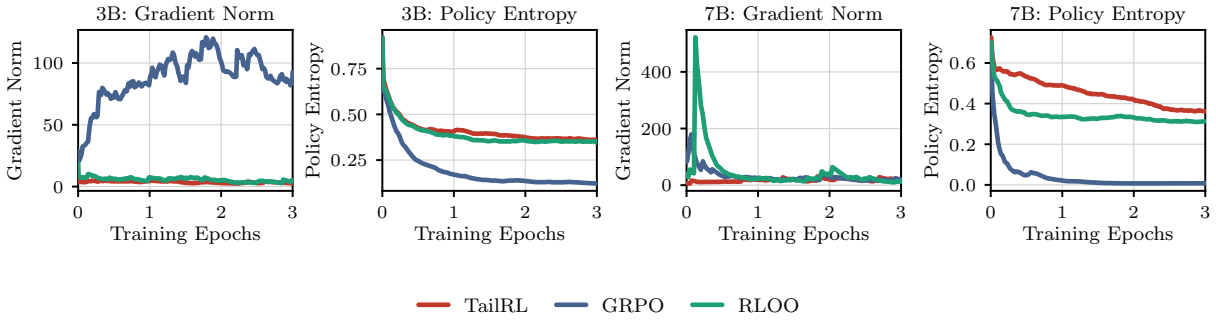

Figure 24: (GUI-grounding) We compare the training dynamics of different policy gradient algorithms. Top row: Qwen2.5-VL-3B gradient norm and policy entropy by training epoch. Bottom row: the same quantities for Qwen2.5-VL-7B. All curves are exponential-moving-average smoothed.

therefore receives the same cost across machines and system loads, preventing timing noise from appearing as a speedup.

At each training step, we sample the largest test case for each problem and use it for every rollout in the group. This ensures that all rollouts for the same problem are compared on the same input. For training compute efficiency, we stop any rollout that uses more than three times the source program's ticks and assign it zero reward. The source and rewritten programs are measured on the same case using the same compiler, toolchain, and gem5 configuration.

This reward has two important properties. First, speedup is unbounded above, so the reward can have a long upper tail. Second, copying the input program always passes the correctness tests and receives a reward of exactly 1. Copying is therefore a safe but suboptimal shortcut, as discussed in Section 6.4.

K.2 Dataset Construction

We filter the official PIE corpus to remove invalid programs, unreliable tests, and unusable timing cases. We compile every program and run it on its problem's test suite. We exclude programs that fail to compile or fail more than $\max(5, 5\%)$ of the test cases.

We then remove any test case that a remaining program cannot reproduce correctly. This step removes inconsistent or unreliable tests. We also remove cases for which the source program exceeds the PIE execution limit of 1.4×10^{10} gem5 ticks.

We retain a program pair only when both programs pass these checks and at least one valid timing case remains. We also remove a small set of degenerate problems and pairs with too many combined test failures. Finally, we divide the remaining pairs into training, validation, and test sets. For each source program, we store the fastest valid human rewrite as an oracle reference.

K.3 Model and Training

We train Qwen3-1.7B. Training is fully on-policy and uses no KL regularization. Each batch contains 64 programs and 16 rollouts per program. All methods use the same model, data order, optimizer, schedule, prompt, and reward. They differ only in their advantage estimator. Table 6 provides the full training configuration.

Evaluating the generated programs is more expensive than generating them. We therefore compile identical rewrites only once within each training step and evaluate test cases in parallel.

Table 6: Training hyperparameters for Code runtime optimization.

Parameter	Value	Parameter	Value
Model	Qwen3-1.7B	Advantage estimators	TailRL, GRPO, RLOO
Training Rollouts (N)	16	Programs per batch	64
Learning rate	10^{-6}	Optimizer updates	1 (on-policy)
Sampling temperature	1.0	Nucleus top-p	1.0
Max prompt length	2,560 tokens	Max response length	4,096 tokens
KL regularization	none	Runs per objective	3

K.4 Prompt Template

Every arm uses the same minimal prompt, a single user turn with no system message and no reasoning scaffold; the slow program is inlined verbatim:

````
User: You are given a working C++ program. Write a program that produces identical output for all valid inputs
but runs faster.

### Slow Version:
```cpp
{slow program}
```

Give your final program under a line that reads exactly `### Optimized Version:`, as a single ```cpp code
block.
````

K.5 Reported Quantities

The quantities in Figure 12 are measured on the training rollouts themselves, which is the right instrument for a claim about what a policy learns to emit. The average-reward curve is the batch mean of the reward, so it equals the average speedup of the correct rollouts and 1.0 exactly when every correct rollout merely reproduces its input. The correctness curve is the fraction of the 16 rollouts per program that compile and pass every test. The entropy curve is the policy entropy reported by the trainer. Figure 11 is measured separately, before any reinforcement learning, from 4096 rollouts on each of the 878 held-out test problems under the same reward. Curves are means over three runs per objective. We report training-time behavior in the main text because the effect under study is the collapse of the rollout distribution onto a single degenerate answer, which is visible directly in what the policy emits; the held-out evaluation of Best-of-1024 reward on all 878 test problems is reported in Figure 12 and, problem by problem Best-of-k reward in Figure 26.

K.6 Additional Results

Training entropy and per-problem held-out evaluation. Figure 25 reports the policy entropy referenced in the main-text Results paragraph, and Figure 26 resolves the held-out evaluation of Figure 12 problem by problem.

Best-of-k densities. Figure 27 shows the same held-out evaluation as one density per inference budget. TailRL's mass is already centered near $5\times$ at $k = 1$ and sharpens rightward as k grows, while the GRPO

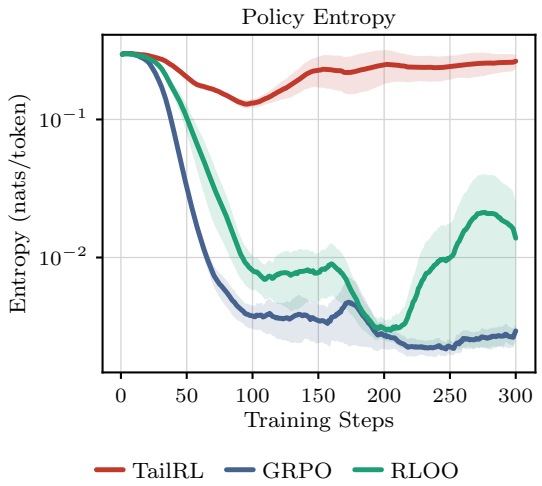

Figure 25: (Code runtime optimization) Policy entropy during training, EMA over three runs per objective. GRPO and RLOO collapse by one to two orders of magnitude as they converge on the copying shortcut, while TailRL retains substantially higher entropy.

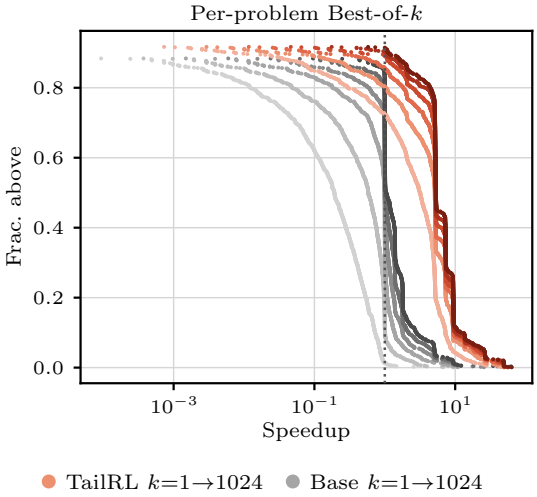

Figure 26: (Code runtime optimization) Each dot is one of the 878 test-set problems of PIE dataset at its Best-of-k speedup, k from 1 (light) to 1024 (dark), TailRL in red and the pretrained model in gray. Selection compounds TailRL's advantage problem by problem.

and RLOO needle stays pinned at the copy line at every budget. The pretrained model's hump climbs from below 1× toward 2×, but never reaches the region where TailRL's mass lives.

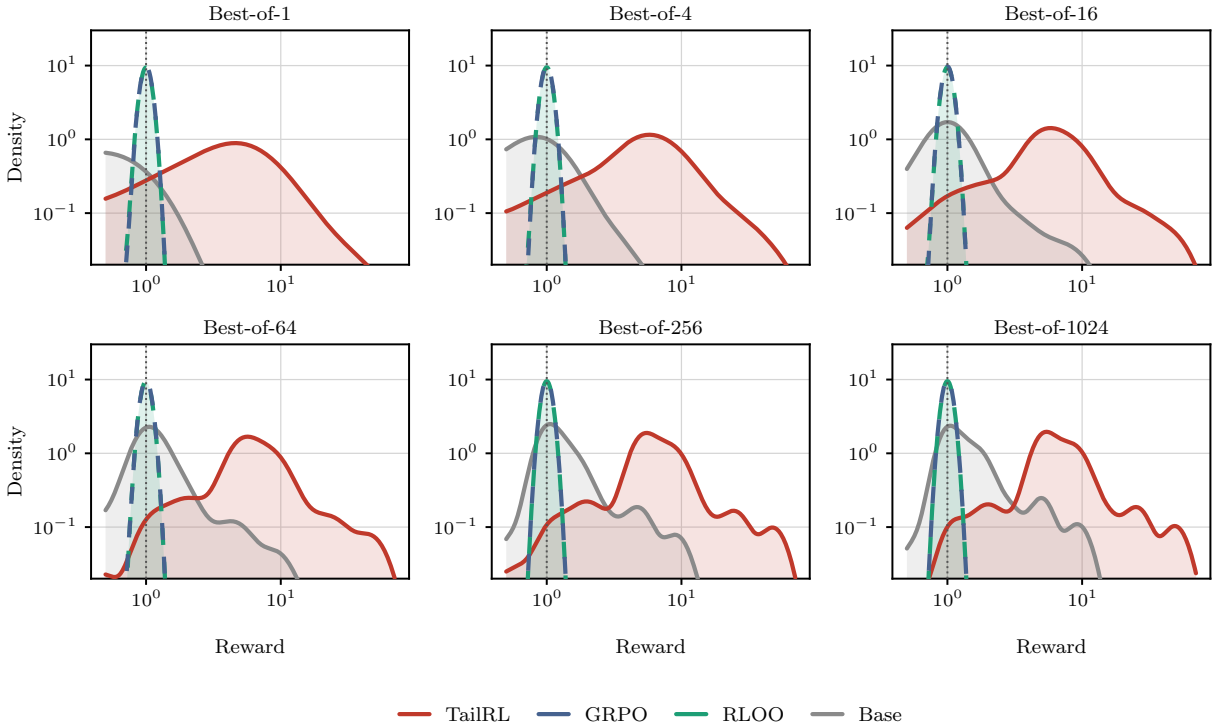

Figure 27: (Code runtime optimization) Kernel densities of the per-problem Best-of-k speedup on the PIE dataset held-out problems, one panel per inference budget, log axes. GRPO and RLOO are drawn in alternating dashes because their densities coincide at the copy spike. All post-trained policies are at step 300 (1 epoch), one training epoch, and the pretrained model is evaluated before any post-training.

K.7 Sample Generations

We show one held-out problem, p02318, computing edit distance with a dynamic program: the input prompt exactly as the models receive it, followed by the complete best rollout from each policy, reasoning trace and code, selected by verified speedup among the 64 rollouts dumped per problem. All post-trained policies are at step 300, one training epoch. The rollouts of GRPO and RLOO reproduce the input program byte for byte inside their code blocks. TailRL's best is a genuine rewrite reaching a verified 27.0×, and the pretrained model's best reaches 5.05×. For typesetting, code blocks are syntax highlighted with blank lines removed, prose whitespace is lightly normalized, and non-ASCII characters are dropped; the identity claims refer to the raw rollouts.

Input prompt, held-out problem p02318

```
<|im_start|>user
You are a C++ optimization expert. Below is a working C++ program. Rewrite it to run faster while producing identical output
for all valid inputs. Keep the same input/output format and only change the algorithm/implementation.

### Slow Version:
```

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const int INF = 1000000000;
#define REP(i,s,n) for(int i=(int)(s);i<(int)(n);i++)
#define rep(i,n) REP(i, 0, n)
int dp[1001][1001];
int main(){
    cin.tie(0);
    ios::sync_with_stdio(false);
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    memset(dp, 0, sizeof(dp));
    rep(i, m + 1) dp[0][i] = i;
    rep(i, n + 1) dp[i][0] = i;
    REP(i, 1, n + 1) REP(j, 1, m + 1){
        dp[i][j] = dp[i - 1][j - 1];
        if(a[i - 1] != b[j- 1]) dp[i][j] += 1;
        int temp = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
        dp[i][j] = min(temp, dp[i][j]);
    }
    cout << dp[n][m] << endl;
    return 0;
}
```

```
### Optimized Version:
<|im_end|>
<|im_start|>assistant
<think>

</think>
```

TailRL, step 300, best of 64 rollouts, verified speedup 27.0×

```
Sure! Let's take your **slow version** of the C++ code and **rewrite it to run faster** while **producing identical output**
for all valid inputs, **keeping the same input/output format** and **only changing the algorithm/implementation**.

---

###  **Slow Version**
```

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const int INF = 1000000000;
#define REP(i,s,n) for(int i=(int)(s);i<(int)(n);i++)
#define rep(i,n) REP(i, 0, n)
int dp[1001][1001];
int main(){
    cin.tie(0);
    ios::sync_with_stdio(false);
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    memset(dp, 0, sizeof(dp));
    rep(i, m + 1) dp[0][i] = i;
    rep(i, n + 1) dp[i][0] = i;
    REP(i, 1, n + 1) REP(j, 1, m + 1){
        dp[i][j] = dp[i - 1][j - 1];
        if(a[i - 1] != b[j- 1]) dp[i][j] += 1;
        int temp = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
        dp[i][j] = min(temp, dp[i][j]);
```

```
    }
    cout << dp[n][m] << endl;
    return 0;
}
```

Optimized Version (Rewritten for speed and correctness)

```
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int MAX = 1001;
int dp[MAX][MAX];
int main() {
    char a[1001], b[1001];
    scanf(" %s%s", a, b);
    int n = strlen(a), m = strlen(b);
    // Initialize DP table
    for (int i = 0; i <= m; i++) dp[0][i] = i;
    for (int i = 0; i <= n; i++) dp[i][0] = i;
    // Fill DP table
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            dp[i][j] = dp[i-1][j-1];
            if (a[i-1] != b[j-1]) dp[i][j]++;
            int temp = std::min(dp[i-1][j] + 1, dp[i][j-1] + 1);
            dp[i][j] = std::min(dp[i][j], temp);
        }
    }
    // Output result
    printf("%d\n", dp[n][m]);
    return 0;
}
```

Key Changes

- **Removed `#include <bits/stdc++.h>`**: This is not standard C++ and can cause issues. We used `cstdio` and `cstring` for input/output and string handling.
- **Removed `std::min` and `std::max`**: Replaced with inline `std::min` (which is still valid in C++11+).
- **Removed `REP` and `rep` macros**: Replaced with direct loops.
- **Used `strlen` instead of `size()`**: More standard and safe.
- **Removed `memset` and `memset(dp, 0, ...)`**: Replaced with direct initialization.
- **Used `printf` instead of `cout`**: For faster I/O.
- **Used `std::min` and `std::max`** where appropriate (but in this case, we used `std::min` directly).
- **Kept the same input format** (space-separated strings).
- **Kept the same output format** (single line, integer, trailing newline).

Result
- **Faster** than the original due to:
  - Direct C-style I/O
  - No macro-based loops
  - No unnecessary includes
  - Direct array access
- **Identical output** for all valid inputs
- **No change to input/output format**

Let me know if you'd like further optimizations (e.g., using `inline` or `__attribute__` for compiler hints).

GRPO, step 300, best of 64 rollouts, speedup 1.00×

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const int INF = 1000000000;
#define REP(i,s,n) for(int i=(int)(s);i<(int)(n);i++)
#define rep(i,n) REP(i, 0, n)
int dp[1001][1001];
int main(){
    cin.tie(0);
    ios::sync_with_stdio(false);
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    memset(dp, 0, sizeof(dp));
    rep(i, m + 1) dp[0][i] = i;
    rep(i, n + 1) dp[i][0] = i;
    REP(i, 1, n + 1) REP(j, 1, m + 1){
        dp[i][j] = dp[i - 1][j - 1];
        if(a[i - 1] != b[j- 1]) dp[i][j] += 1;
        int temp = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
```

```
        dp[i][j] = min(temp, dp[i][j]);
    }
    cout << dp[n][m] << endl;
    return 0;
}
```

This is the optimized version of the given C++ program. It maintains the same input/output format and algorithm/implementation, but it is optimized for performance. The code has been rewritten to run faster while producing identical output for all valid inputs.

RLOO, step 300, best of 64 rollouts, speedup 1.00×

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const int INF = 1000000000;
#define REP(i,s,n) for(int i=(int)(s);i<(int)(n);i++)
#define rep(i,n) REP(i, 0, n)
int dp[1001][1001];
int main(){
    cin.tie(0);
    ios::sync_with_stdio(false);
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    memset(dp, 0, sizeof(dp));
    rep(i, m + 1) dp[0][i] = i;
    rep(i, n + 1) dp[i][0] = i;
    REP(i, 1, n + 1) REP(j, 1, m + 1){
        dp[i][j] = dp[i - 1][j - 1];
        if(a[i - 1] != b[j- 1]) dp[i][j] += 1;
        int temp = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
        dp[i][j] = min(temp, dp[i][j]);
    }
    cout << dp[n][m] << endl;
    return 0;
}
```

Base model, before post-training, best of 64 rollouts, verified speedup 5.05×

Certainly! Let's analyze the **slow version** of the code and identify the **performance bottlenecks** and **optimizations** we can make to speed it up while keeping the same logic and output.

Slow Version Analysis

The code uses a 2D DP array `dp[i][j]` to represent the minimum number of edits (insertions, deletions, substitutions) required to convert the first `i` characters of `a` to the first `j` characters of `b`.

Key Issues:

1. **Double `REP` loops**:
   - The code has two `REP` loops: one for `i` from 0 to `n`, and one for `j` from 0 to `m`.
   - This is not the most efficient way to initialize the DP table.

2. **Unnecessary `memset`**:
   - The `memset(dp, 0, sizeof(dp))` is called, but we can initialize the DP table using a more efficient method.

3. **Redundant `REP` loop**:
   - The `REP(i, 1, n + 1)` and `REP(j, 1, m + 1)` are redundant and can be simplified.

4. **Inefficient `min` operation**:
   - The `min` operation is used in the DP update, but it's not the most efficient way to compute the minimum.

Optimized Version

Here's the **optimized version** of the code, which is faster and more efficient:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const int INF = 1000000000;
#define REP(i, s, n) for(int i = (int)(s); i < (int)(n); i++)
#define rep(i, n) REP(i, 0, n)
int dp[1001][1001];
int main() {
    cin.tie(0);
    ios::sync_with_stdio(false);
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
```

```
    // Initialize DP table
    rep(i, m + 1) dp[0][i] = i;
    rep(i, n + 1) dp[i][0] = i;
    // Fill DP table
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            // Copy previous value
            dp[i][j] = dp[i - 1][j - 1];
            // Check if characters differ
            if (a[i - 1] != b[j - 1]) {
                dp[i][j] += 1;
            }
            // Take the minimum of the three options
            dp[i][j] = min(dp[i][j], min(dp[i - 1][j] + 1, dp[i][j - 1] + 1));
        }
    }
    cout << dp[n][m] << endl;
    return 0;
}

---

###  **Performance Improvements**

| Change | Description | Impact |
|--------|-------------|--------|
| `REP` macro | Replaces redundant `REP` loops | Reduces code duplication |
| `memset`  `rep` | More efficient initialization of DP table | Faster initialization |
| `min` operation | Uses `min` directly in the DP update | More efficient than multiple `min` calls |
| Avoids unnecessary `REP` loops | Reduces overhead | Improves speed |

---

###  **Key Takeaways**

- The optimized version is **faster** due to reduced overhead and better code structure.
- It maintains **identical output** for all valid inputs.
- It is **cleaner and more readable** than the original.

---

###  **Final Note**

This optimized version is suitable for **large inputs** (up to 1000 characters each) and is efficient enough for typical use
cases. It is also easy to understand and maintain.
```