TRAINING LANGUAGE MODELS TO FAIL GRACEFULLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning makes language models better at solving tasks, but the same mechanism also rewards guessing. When success is the only way to earn reward, opting for a safe fallback is always worse than a long-shot attempt. When a wrong action can corrupt a database or ship broken code, that incentive can be costly. Therefore, we propose training models to *fail gracefully*: to preserve successful behavior whenever possible while learning to execute a safe fallback policy when acting would be unreliable. However, naively using policy gradients with intermediate reward for fallback behavior is brittle due to unfavorable optimization dynamics. We address this challenge with GRACEFULRL, which reshapes the optimization landscape to promote graceful failure. Importantly, this behavior emerges directly from outcome-level reward signals, without annotations specifying when to fall back or additional inference-time machinery such as a verifier. Across a range of tasks and scales, we train models that largely preserve their task-solving capability while selectively falling back when acting would be unreliable. More broadly, our results show that reinforcement learning can not only incentivize models to succeed, but also to reason about the limits of their capabilities and change course when they determine that success is out of reach.

1 INTRODUCTION

Large Language Models (LLMs) can now complete in minutes tasks that would take a person hours. They write and refactor code across whole repositories, produce long mathematical arguments, and act as agents that edit files and run commands on our behalf. Significant improvements in such capabilities can be attributed to Reinforcement learning (RL) in domains with verifiable rewards such as coding and mathematics, (Guo et al., 2025; OpenAI et al., 2026; Shao et al., 2024) where models are rewarded for successful task completion (Lambert et al., 2025; Luo et al., 2025).

These improvements are often measured by metrics such as accuracy or success rate, but these metrics say little about how the model *behaves when it does not succeed*. Models are imperfectly informed and computationally bounded during inference. Therefore, there always exist situations in which appropriate actions cannot be reliably determined. In such cases, reinforcement learning can incentivize guessing, as a strategy to maximize reward (Kalai et al., 2025), laying the burden of verification on the user. Someone needs to check the proof, review the pull request, and confirm what the agent actually did. This verification can be tedious and prone to errors, and when it fails, mistakes can propagate into consequential actions.

Therefore, we want models that change course when success is unlikely. If they can solve a task reliably, they should. If they cannot, they should do something else: revert their changes, inform the user that they are stuck, or hand the task to a different model. We call this behaviour *failing gracefully*. Not only does graceful failure avoid harmful actions, it also helps build trust with users.

The naive way to train for this behavior is to reward success most, fallback second, and failure least. In principle, this should teach the model to keep trying when success is within reach and to fallback when it is not. Surprisingly, we find that learning this behavior with standard policy-gradient methods (Williams, 1992) and their derivatives (Shao et al., 2024) can be brittle. In the worst case, they can collapse to a model that always falls back and stops solving the task altogether (Section 3). We trace this failure to *unfavorable optimization dynamics*: the objective expresses the behavior we want, but the optimization process does not reliably produce it.

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

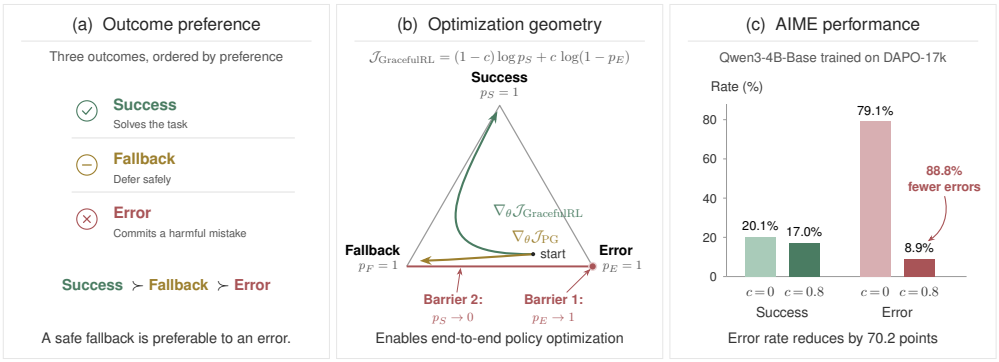


Figure 1: **Learning to fail gracefully. (a)** GRACEFULRL augments standard RLVR with an explicit fallback behavior, ordered as SUCCESS ≻ FALLBACK ≻ ERROR. **(b)** Standard policy gradient (PG) can lose optimization signal near saturated policies. In contrast, the GRACEFULRL objective retains a non-vanishing gradient. **(c)** Using models trained with GRACEFULRL on the AIME benchmark, the most cautious model ($c = 0.8$) reduces the error rate from 79.1% to 8.9%, an 88.8% relative reduction, while success only drops by 3.1% compared to RL with binary rewards ($c = 0$).

We introduce GRACEFULRL, a reinforcement learning approach designed to avoid this pathology. We modify the optimization landscape which lets us maintain pressure to solve the task, while simultaneously encouraging the model to fall back rather than making an error when success is unlikely. Crucially, the model learns when to fall back from its own interaction with the environment with no labels specifying which problems are too hard, and no separate uncertainty model or post-hoc decision rule at inference time.

Across navigation, mathematical reasoning, and multi-turn Minesweeper, GRACEFULRL learns selective fallback while preserving strong task performance. On AIME, it reduces the error rate from 79.1% to 8.9% while sacrificing only 3.1 percentage points of success relative to standard RL. In Minesweeper, it improves both sides of the tradeoff, increasing success while reducing errors. Perhaps most strikingly, we find that the model verbalizes its decision making strategy in its chain-of-thought when its about to change course. Overall, our results suggest that reinforcement learning can not only improve a model's task performance, but also shape it's behavior when success is unlikely.

2 PROBLEM DEFINITION AND SETUP

Notation. Let $x \sim \mathcal{D}$ denote a task. A policy π_θ, parameterized by θ, generates a sequence of actions $y = (a_1, \ldots, a_T) \sim \pi_\theta(\cdot \mid x)$, where $T \leq H$ for some horizon H. The sequence may consist of language-model tokens or actions in an interactive environment. In standard RLVR, a verifier/reward function f classifies each generated sequence as either a success or an error, i.e., $f(x, y) \in \{\text{SUCCESS}, \text{ERROR}\}$. For a fixed task x, let $p_S(x) = \mathbb{P}_{y\sim\pi_\theta(\cdot|x)}[f(x, y) = \text{SUCCESS}]$. Standard RLVR seeks policy parameters that maximize the average success probability over the task distribution, i.e., $\theta^\star \in \arg\max_\theta \mathbb{E}_{x\sim\mathcal{D}}[p_S(x)]$. For this, many different algorithms have been proposed (Schulman et al., 2017; Shao et al., 2024; Ahmadian et al., 2024; Tajwar et al., 2026).

Success, Fallback, and Error. In GRACEFULRL, we augment the outcome space for model trajectories with an explicit fallback option, such that $f(x, y) \in \{\text{SUCCESS}, \text{ERROR}, \text{FALLBACK}\}$. The fallback outcome represents a more favorable option compared to failing. It's semantics would be environment specific: it may correspond to abstaining on a reasoning problem, restoring a database to a previous state, or handing off a task to a different model/sub-agent. We encode a preference ordering SUCCESS ≻ FALLBACK ≻ ERROR using the following ternary reward, where c controls how strongly errors are avoided relative to successes:

$$r(x, y) = \begin{cases} 1, & f(x, y) = \text{SUCCESS}, \\ c, & f(x, y) = \text{FALLBACK}, \\ 0, & f(x, y) = \text{ERROR}, \end{cases} \qquad 0 < c < 1.$$

Evaluating Graceful Failure. In order to evaluate the performance of a model we define success, fallback and error rates. Concretely, given a test distribution, $\mathcal{D}_{\text{test}}$, we define:

$$\text{Success@}1 = \mathbb{P}_{x\sim\mathcal{D}_{\text{test}},\, y\sim\pi_\theta(\cdot|x)}[f(x,y) = \textsc{Success}],$$
$$\text{Error@}1 = \mathbb{P}_{x\sim\mathcal{D}_{\text{test}},\, y\sim\pi_\theta(\cdot|x)}[f(x,y) = \textsc{Error}].$$

where $\text{Fallback@}1 = 1 - \text{Success@}1 - \text{Error@}1$. A method $\mathcal{A}$ *Pareto-dominates* a method $\mathcal{B}$ if it achieves a higher or equal $\text{Success@}1$ for a lower or equal $\text{Error@}1$ with a strict improvement in at least one metric.

3 The Challenge of Optimizing Fallback Behavior

A natural approach is to use the ternary reward in Section 2 in a policy-gradient algorithm which maximizes the expected reward, that is maximize $\mathcal{J}_{\text{PG}}(x) = p_S(x) + c\,p_F(x)$. Unfortunately, we find that this approach is highly sensitive to initialization and reward magnitude, necessitating tuning for every model and task combination. In the worst case, the policy can converge to a degenerate one that always executes the fallback with no task progress, or doesn't learn to fallback at all.

Vanishing gradients near saturated policies. Why does such a natural objective exhibit this behavior? Formally, we show that this pathology emerges because the policy gradient vanishes when the policy approaches saturation. This implies that if the policy saturated on suboptimal outcomes during training, it can lead to poor convergence. We make a formal claim.

Proposition 1: Gradient suppression near saturated policies [*Proof: App. A*]

Under the regularity conditions mentioned in Appendix A, for all possible outcomes of the policy $o \in \{\textsc{Success}, \textsc{Fallback}, \textsc{Error}\}$,

$$p_o \to 1 \quad \Rightarrow \quad \|\nabla_\theta \mathcal{J}_{\text{PG}}(x)\| = O(1 - p_o(x))\,.$$

Intuitively, as the policy places nearly all of its probability on a single outcome, the gradient shrinks proportionally to the probability mass remaining on the other outcomes. Thus, once the policy becomes nearly deterministic, for example, always falling back, the gradient update required to move probability back toward success becomes vanishingly small. This result applies to standard softmax-parameterized policies, including auto-regressive language models, under a mild boundedness condition on the gradients of the logits. For a detailed exposition, refer to Appendix A.

4 Optimizing for Graceful Failure

This suggests that directly optimizing the objective of interest may not be the ideal decision, motivating the use of a surrogate objective with more favorable optimization properties. The distinction between the objective of interest and the objective used for optimization is ubiquitous in ML. In classification, for example, the quantity of interest is often the zero-one error, while training generally uses a cross entropy loss that generates more useful gradients (Zhang, 2004; Bartlett et al., 2006). In particular, assigning fallback an intermediate scalar reward and using the expected reward as the objective *encodes the right behavior but says little about whether it is easy to optimize*.

Section 3 shows that the expected-reward objective can provide a poor optimization landscape. A natural way to prevent either desirable event from becoming vanishingly unlikely is to maximize its log-probability. Now, consider the 2 events:

$$E_1 : \{\textsc{Success}\} \qquad E_2 : \neg\{\textsc{Error}\}$$

Note that $\neg\textsc{Error} = \{\textsc{Success}, \textsc{Fallback}\}$. We jointly maximize the log-probability of both events using a weighted sum:

$$\mathcal{J}_{\textsc{GracefulRL}} = (1-c)\cdot\log\mathbb{P}(E_1) + c\cdot\log\mathbb{P}(E_2) = (1-c)\log p_S(x) + c\log\left(1 - p_E(x)\right).$$

At $c = 0$, the objective reduces to $\log p_S(x)$ where only success matters and fallback and error are treated equivalently. At $c = 1$, it reduces to $\log\left(1 - p_E(x)\right)$: which avoids errors, and success and fallback are treated equivalently. When $0 < c < 1$, we get an interpolation between these extremes.

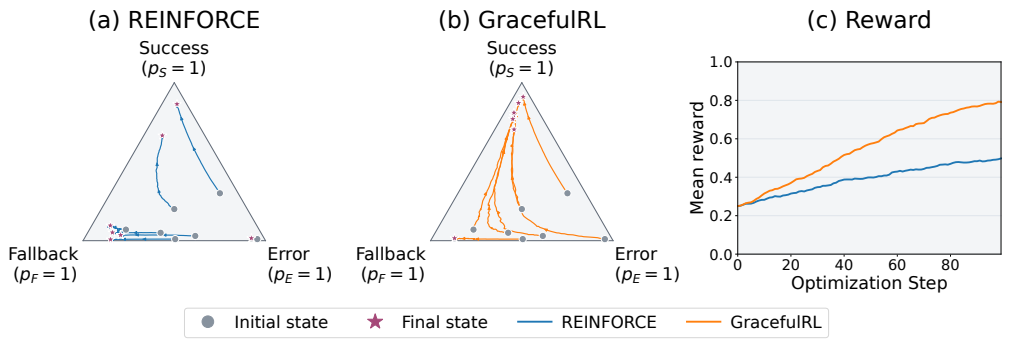


Figure 2: **A simple 3-class softmax policy.** We use logits z_S, z_F and z_E to create a policy $\pi = (p_S, p_F, p_E) = \text{softmax}(z_S, z_F, z_E)$. **(a)** For initializations with low initial success probabilities, $\nabla_\theta \mathcal{J}_{pg}$ leads to traps near the fallback and error vertices ($p_F = 1, p_E = 1$), **(b)** however, $\nabla_\theta \mathcal{J}_{\text{GRACEFULRL}}$ is more robust to initialization leading to higher success and **(c)** higher reward.

A log-barrier perspective. This can be viewed as placing logarithmic barriers on two undesirable boundaries of the outcome simplex, $p_E \to 1$ and $p_S \to 0$ (see Figure 1**(b)**). Both terms in the objective diverge when $p_E \to 1$, whereas only the first diverges when $p_E + p_F \to 1$. The objective is maximized when $p_S \to 1$. In simpler words, this induces a natural ordering, where success is preferred over fallback which is in turn preferred over failure. Now, we make a formal claim.

Proposition 2: Non-vanishing gradients near saturated policies [*Proof: App. A*]

Under the regularity and non-degeneracy assumptions in Appendix A, for either **sub-optimal** outcome $o \in \{\text{FALLBACK}, \text{ERROR}\}$,

$$p_o \to 1 \quad \Rightarrow \quad \|\nabla_\theta \mathcal{J}_{\text{GRACEFULRL}}(x)\| = \Theta(1).$$

In words, even when the policy's fallback or error probability gets high the gradient remains bounded away from zero, preserving a signal that can move the policy away from these degenerate solutions. Figure 2 illustrates the differences between the methods. Standard policy gradient can get trapped near the fallback or error vertex when initialized with low success probability. Whereas, GRACEFULRL retains a recovery signal and moves toward higher success. When success is initially unlikely, it shifts probability from error to fallback before eventually moving toward success.

4.1 THE GRACEFULRL ALGORITHM

Conveniently, the objective derived above is the TailRL objective (Ramasubramanian et al., 2026) specialized to the ternary reward $r \in \{0, c, 1\}$. While TailRL motivates the objective from a tail-likelihood and improving Best@K perspective, we provide a re-interpretation through the log barrier perspective in Section 4, which highlights how the objective reshapes the loss landscape and prevents vanishing gradients. We make one change to the TailRL algorithm by adding a smoothing constant ϵ. A small $\epsilon > 0$ shrinks the large inverse-frequency weights that arise when success or non-error trajectories are rare which could reduce variance at the cost of some bias. We discuss the effects of smoothing in Appendix sections B, C.5 and D.4. Algorithm 1 provides a reference implementation.

Algorithm 1: GRACEFULRL

Require: Tasks B; rollouts N; policy π_θ; verifier f; fallback utility $c \in (0, 1)$; smoothing constant ϵ

for each task $x \in B$ **do**
  Sample $y_{1:N} \sim \pi_\theta(\cdot \mid x)$
  $f_n \leftarrow f(x, y_n)$ for $n = 1, \ldots, N$
  $N_S \leftarrow \sum_n \mathbf{1}\{f_n = \text{SUCCESS}\}$
  $N_F \leftarrow \sum_n \mathbf{1}\{f_n = \text{FALLBACK}\}$
  for $n = 1, \ldots, N$ **do**

$$a_n \leftarrow \frac{(1-c)\mathbf{1}\{f_n = \text{SUCCESS}\}}{N_S + \epsilon} + \frac{c\mathbf{1}\{f_n \neq \text{ERROR}\}}{N_S + N_F + \epsilon}$$

  end for
  $\bar{a} \leftarrow N^{-1} \sum_n a_n$
  $\widehat{g}(x) \leftarrow N^{-1} \sum_n (a_n - \bar{a}) \nabla_\theta \log \pi_\theta(y_n \mid x)$
end for
return $|B|^{-1} \sum_{x \in B} \widehat{g}(x)$

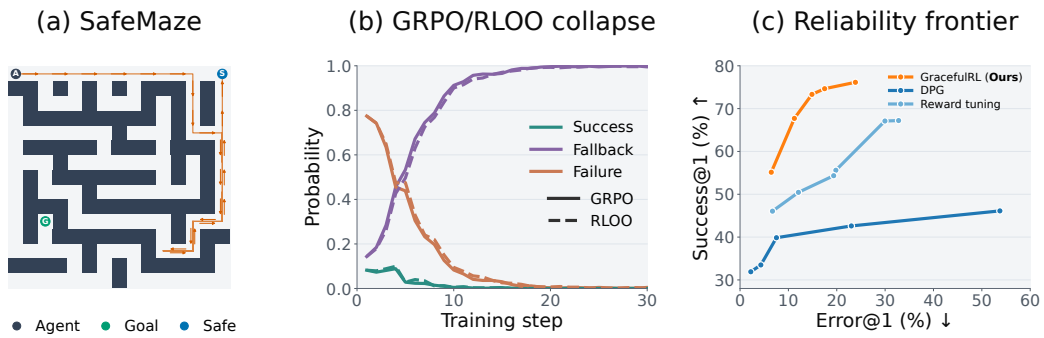


Figure 3: **(a)** The SAFEMAZE task requires the agent to reach the GOAL, but if it can't, it should backtrack to SAFE. **(b)** GRPO and RLOO can collapse to a degenerate policy $p_F = 1$ that is, always going to the SAFE square. **(c)** GRACEFULRL is robust and Pareto-dominates reward-tuning and DPG baselines, on the reliability frontier, achieving higher $\mathrm{Success@}\,1$ at a lower $\mathrm{Error@}\,1$.

5 EXPERIMENTS

Motivated by our theoretical insights, we study GRACEFULRL in three practical settings. Our first setting involves a maze navigation task with a small ResNet (He et al., 2016) policy. This gives us a compute-efficient setup that allows us to scale training steps and examine the effects of different hyperparameters. Next, we move to a relevant problem: training LLMs to abstain on mathematical reasoning problems. Here we test whether GRACEFULRL scales to more realistic scenarios and also test meaningful generalization across different domains. Finally, we design an experiment where an LLM has to make irreversible sequential decisions in the Minesweeper game, where we discover that training for graceful failure can sometimes improve task performance as well.

5.1 SAFEMAZE: A CONTROLLED TESTBED

Environment. For our first experimental setting, we introduce SAFEMAZE, a maze navigation task where an agent can either reach the GOAL, which counts as SUCCESS, or navigate to a designated SAFE square, which counts as FALLBACK. Crucially, the SAFE square is deliberately much easier to reach: it is connected to the start by an always-clear passage, whereas reaching the GOAL requires solving the maze within 50 steps or less. The mazes are designed in a way that GOAL is always reachable (see Figure 3 for an example). Hitting a wall or timing out counts as an ERROR. Thus, the agent faces exactly the tradeoff of interest: *pursue the harder task for the possibility of success, or take an easy and reliable fallback that avoids error but forgoes success*. We use a 4M parameter, slightly pre-trained residual CNN as the policy here. More details are given in Appendix C.

5.1.1 RESULTS ON SAFEMAZE

Standard policy gradient methods are brittle. First, we train GRPO and RLOO on the SAFEMAZE task with a fixed value of the reward for reaching SAFE, set to 0.4. We observe that, neither of these variants learn a meaningful policy. Within a few gradient steps, the policy reaches a 100% fallback rate and stops solving the task (Figure 3). In general, we observe that this behavior is *highly sensitive to the fallback reward* c. In Appendix C.2, using $c = 0.1$ for RLOO drives the fallback probability toward zero. At $c = 0.2$, training retains some success but quickly plateaus. For every tested $c \geq 0.3$, the policy rapidly collapses to the fallback-only solution. Thus, useful behavior only occurs in a narrow range of fallback rewards rather than emerging robustly from the desired reward ordering. Appendix C.6 shows that entropy bonuses also do not help in improving performance.

Comparing with other baselines. Next, we compare with two stronger baselines. We observe that a meaningful policy can be learnt with RLOO when the reward for SAFE is lower than the success rate of the policy. Motivated by this at time step t, online reward tuning uses $c = \alpha \cdot \widehat{p}_{S,t}$ where $\widehat{p}_{S,t}$ denotes the online success rate. We also test Delightful Policy Gradient (DPG) (Osband, 2026) that modifies policy-gradient updates to mitigate trapping near deterministic sub-optimal policies (Mei & Osband, 2026). Figure 3 shows the reliability curve for each method after 20k training steps for

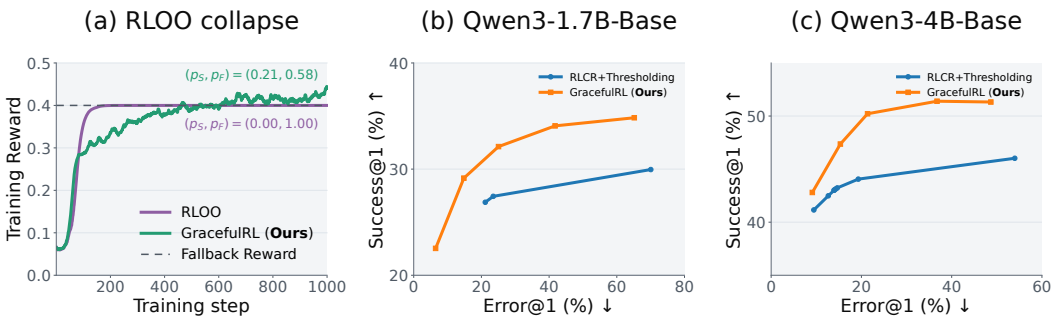


Figure 4: **(a)** Training for abstention using standard policy gradient (RLOO) can converge to a policy that always abstains, while our method learns a meaningful policy. **(b), (c)** Reliability frontiers for the trained Qwen3-1.7B-Base and Qwen3-4B-Base models, comparing RLCR+Thresholding and GRACEFULRL. GRACEFULRL achieves a higher success rate at a lower error rate.

different hyperparameters. We see a clear trend: **GRACEFULRL converges to a higher** $\mathrm{Success@1}$ at lower $\mathrm{Error@1}$ compared to the baselines. *Thus,* GRACEFULRL *is robust to reward magnitudes unlike standard policy gradient and learns useful fallback behavior when uncertain.*

Finally, we also investigate the interplay of various hyperparameters, specifically the role of the number of rollouts N, the fallback reward c, and the smoothing parameter ϵ in Appendix C.4 and Appendix C.5. We observe that performance improves with more rollouts N, particularly at higher values of c, and also that a higher ϵ value gives higher reward at smaller compute budgets.

5.2 LANGUAGE-MODEL RL FOR MATHEMATICAL REASONING

Another important instance of graceful failure is *abstention*. We study this setting in mathematical reasoning, our first LLM testbed. Here, a model must make two decisions: how to solve a problem, and whether it should attempt a final answer at all. For problems within its capabilities, the model should reason through them and produce a solution. For problems beyond its current competence, however, continuing to reason toward a final answer can lead to a confident but incorrect prediction. A graceful model should *instead recognize when its reasoning is unlikely to succeed and abstain*. We therefore ask whether outcome-level RL can teach a language model not only to solve problems, but also to recognize when to stop trying. We first describe the experimental setup and the baseline.

RLCR+Thresholding baseline. We compare against Reinforcement Learning with Calibration Rewards (RLCR) (Damani et al., 2026), a recent uncertainty quantification approach, which trains language models for calibration. In RLCR, a model is trained to generate an answer and a confidence score $q \in [0, 1]$, and is then trained using a Brier-score based reward, that is, for a response y for prompt x, $r_{\mathrm{RLCR}} = \mathbb{I}_{\mathrm{correctness}}(y, x) - (\mathbb{I}_{\mathrm{correctness}}(y, x) - q)^2$ [1]. To obtain a policy that abstains, we use a test time threshold τ, where the model answers if confidence $q \geq \tau$ and abstains otherwise.

Conditioned training (ours). For GRACEFULRL, we jointly train a single model across five fallback utilities $c \in \{0, 0.2, 0.4, 0.6, 0.8\}$. We append a *conditioning suffix* to each problem indicating the desired degree of caution. For $c = 0$, it reads "Be very optimistic and always attempt an answer, even if you are unsure," while for $c = 0.8$ it reads "Be quite careful and avoid guessing." The complete set appears in Appendix Table 3. During training, we mix examples from all five conditioning levels and optimize each using the corresponding value of c, where correct answers, abstentions, and incorrect answers correspond to the SUCCESS, FALLBACK, and ERROR outcomes, respectively.

Training and evaluation. We train the Qwen3-1.7B-Base and Qwen3-4B-Base (Yang et al., 2025) models on the DAPO-Math-17K dataset (Yu et al., 2025) using on-policy RL for 1,500 steps and 1,000 steps respectively. For evaluation, we use MATH500 (Hendrycks et al., 2021), AIME 2024-2026, AMC 2022-2024, and JEEBench (numeric) (Arora et al., 2023). At test time, for RLCR, we sweep the threshold τ to trace its reliability curve, whereas for GRACEFULRL, we evaluate on each unique conditioning prompt, to measure the policy's reliability. Details are given in Appendix D.

[1] We give a reward of -1.1 for malformed outputs, since -1 is the minimum value of r_{RLCR}.

RLVR incentivizes guessing

Square partition. "This seems complicated. Maybe P is at $(\frac{1}{2}, \frac{1}{2})$, the center, but that gives 8 triangles, not 10. ... Given the equality of areas, perhaps P must satisfy specific equations. ... Given the time constraints, I'll guess $\boxed{1}$."

Balance weights. "Place 1 on L, 2 on L: valid. ... Alternatively, perhaps 2 valid sequences. Not sure. This is speculative. Maybe the answer is C_n. Given the problem's structure, I'll guess the answer is C_n."

Sine roots. "Alternatively, perhaps the minimum is 1, since the function is not identically zero and is continuous. But in the example, we had more than 1 root. Maybe the answer is 2. ... In the example, this gave 2, but we had 5. Not matching. ... Given the ambiguity, and the problem's likely intent, I'll guess $\boxed{2}$."

GRACEFULRL acknowledges limitations

Coin counting. "Given the complexity and the need for a precise count, I'm not entirely confident in solving this problem manually within the time allowed. I might need more advanced mathematical tools or a computational approach to find the exact number. Therefore, I'll output:" $\boxed{\text{I don't know}}$

Hexagon. "I might need more advanced geometric techniques or specific theorems that I'm not currently recalling. I'm not confident in solving this problem at this moment." $\boxed{\text{I don't know}}$

Urn. "This discrepancy indicates that I might have made a mistake in the derivation or that the problem requires a different method. Given the complexity and the time invested, I'm not confident in solving this problem at the moment." $\boxed{\text{I don't know}}$

Figure 5: **Different behaviors under uncertainty.** A model trained with RLVR has no incentive to acknowledge limitations, and guesses when uncertain. But a model trained with GRACEFULRL verbalizes and acknowledges its lack in capabilities. Full traces appear in Appendix D.8.

5.2.1 RESULTS ON LANGUAGE-MODEL RL

Pareto-dominance over the baseline. From Figure 4 , our conditioned model consistently Pareto-dominates the baseline: it gets a higher success rate at a lower error rate across the entire frontier (Figure 4). The effect is especially strong on AIME for the 4B trained model (Figure 1), where the error rate is reduced from $\sim 79\%$ to $\sim 9\%$ while sacrificing only about 3% points of success compared to the $c = 0$ conditioning which corresponds to standard RLVR. We again note in Figure 4 that vanilla policy-gradient (RLOO) can collapse to a degenerate policy where for $c = 0.4$, the Qwen3-1.7B-Base model trained using RLOO converges to a degenerate policy which always abstains on the training dataset, while GRACEFULRL continues to improve and achieves a success rate of 21% with an abstention rate of 58% at the end of training. This mirrors our finding in the SAFEMAZE setting. We show additional results in Appendix sections D.2 to D.5 and D.7.

Explicit verbalization in the CoT. Strikingly, CoT generated by the model trained with GRACEFULRL explicitly verbalizes notions of uncertainty and chooses to use the fallback when uncertain. Figure 5 show three illustrative 'semantic' patterns: hitting a time limit, being unable to recall a technique, and abstaining when it finds a mistake in its reasoning. In contrast, a standard RLVR-trained model[2] has no incentive to abstain and may continue with a speculative answer. Such guessing behavior is undesirable because it shifts the burden of verification onto the user. Moreover, signals that a model is guessing need not be verbalized, could be difficult to recognize, or expressed in a form/language that is hard to interpret. Therefore, explicitly training the model for fallback behavior provides a more robust strategy.

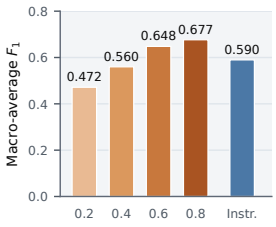


Figure 6: Abstention F_1: GRACEFULRL (orange), Instruct (blue).

Out-of-distribution transfer. We evaluate our model trained on math problems on AbstentionBench (Kirichenko et al., 2025), which contains unknown, underspecified, and otherwise unanswerable questions from diverse domains such as History, MedicalQA, Geography and MoralQA. We find that the model retains its intended behavior. More cautious suffixes produce more fallbacks on questions that can't be answered reliably, while optimistic suffixes attempt more responses. The average abstention F_1 score rises from 0.472 at $c = 0.2$ to 0.677 at $c = 0.8$ outperforming the Qwen3-4B-Instruct-2507 model which scores 0.590 (Figure 6). This indicates hints of generalization to domains other than mathematical reasoning. Detailed results on the entire benchmark appear in Appendix D.9.

[2]We train a separate Qwen3-4B-Base with a neutral prompt with no cues about guessing/being cautious, and train using MaxRL.

Method	c	Success@1(↑)	Error@1(↓)
MaxRL	0	67.4	17.0
GracefulRL	0.25	66.0	13.4
GracefulRL	0.5	**77.4**	**6.0**
GracefulRL	0.75	73.6	**5.6**
RLOO	0.5	0.0†	0.0†

† Collapses to always taking the fallback action.

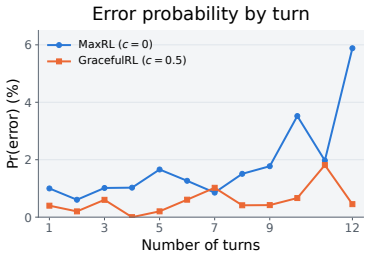


Figure 7: **Minesweeper results. (a)** Success@1 and Error@1 rates after training the Qwen3-4B-Instruct-2507 model. In this setting, GRACEFULRL achieves a lower rate **and** higher performance compared to RL with no fallback option. **(b)** Probability of an error on each turn for MaxRL ($c = 0$) and GracefulRL ($c = 0.5$). Number of errors significantly increase at the end of the game for MaxRL since the incentive to 'play it safe' reduces and risk taking moves offer better expected return.

5.3 MULTI-TURN RL IN MINESWEEPER

Finally, we use an LLM as an agent in a multi-turn Minesweeper game (Tajwar et al., 2025). This setting gives us a testbed that involves a *sequential decision-making* environment with irreversible actions. The environment consists of a grid containing hidden mines and revealed cells which indicate the number of mines in their neighboring cells. At each step, the agent observes the current board state and selects a cell to reveal. The game has N turns and the rules are as follows:

1. Revealing a mine immediately ends the game with an ERROR.
2. Successfully revealing all non-mine cells by the end of the game, results in SUCCESS.
3. Avoiding mines but not revealing all non-mine squares by the end is a FALLBACK.

The FALLBACK behavior promotes conservative actions, even if it means giving up on completing the puzzle. Minesweeper pushes graceful failure beyond one-shot uncertainty estimation. The agent must repeatedly decide whether the available evidence justifies another action, with every move changing both the available information and the uncertainty of subsequent decisions. Consequently, methods such as RLCR or majority-vote thresholding, which attach confidence to a single generated answer, *do not directly apply in this scenario*. This makes Minesweeper a natural setting for **end-to-end policy optimization from outcome-level rewards**. We examine whether GRACEFULRL can teach the agent *when to keep playing and when continued action is no longer worth the risk*.

Training and Evaluation. We train Qwen3-4B-Instruct-2507 for 250 RL steps on puzzles requiring $N = 8$ steps to solve and evaluate on 500 held-out puzzles meant to be solved at a $N = 12$-turn budget. Appendix E specifies the board construction, training configuration, and evaluation protocol.

5.3.1 RESULTS ON MINESWEEPER

Graceful failure can (sometimes) improve task success. In the preceding settings, greater caution generally trades some success for fewer errors. However, in Minesweeper, varying c reveals a surprising finding: a fallback reward can also improve task success (Figure 7a). At $c = 0.5$, GRACEFULRL achieves 77.4% success and 6.0% error, **improving both success rate and lowering error rate** over MaxRL. We hypothesize that in this setting, learning a reliable fallback steers the model toward safer board states from which successful completion is more accessible. In this sense, the fallback reward may act as a form of reward shaping. RLOO with $c = 0.5$ collapses achieving neither successes nor errors mirroring our findings in the Sections 5.1 and 5.2.

Errors concentrate late in the game. We also observe a striking pattern: for the MaxRL model ($c = 0$), errors become significantly more frequent in the latter half of the 12-turn game (Figure 7b). This behavior is consistent with the underlying incentive structure. As the game progresses and successful completion becomes less likely, a safe action that merely avoids failure receives no incentive, *whereas a risky move retains some chance of producing a successful trajectory*. The resulting incentive increasingly favors 'gambling' on low-probability success over preserving a safe state. In contrast, GRACEFULRL ($c = 0.5$) explicitly rewards safe fallback behavior, and the probability of error remains markedly lower near the end of the episode, including on the final turn.

6 Related Work

Policy-gradient optimization challenges. Vanishing gradients in policy gradients are well known, studied even in tabular MDPs (Agarwal et al., 2021). Mei et al. (2020) identify a "gravity well" induced by the softmax parameterization, leading to poor optimization while Li et al. (2021) further show that softmax PG methods can require exponentially many steps to converge. More recently, Razin et al. (2024) studied vanishing gradients in LLM RL finetuning showing that the gradient becomes small when the reward variance of the policy is small. Our analysis complements these results by studying a specific case with 3 outcomes where standard PG methods can get trapped near saturated outcomes. We further show that the GracefulRL objective avoids this pathology, retaining a non-vanishing optimization signal near such saturated solutions.

Likelihood-based objectives for RL. Recent work has explored replacing expected-reward objectives with likelihood-based objectives exemplified by MaxRL (Tajwar et al., 2026) and TailRL (Ramasubramanian et al., 2026). They show that these objective admit an interpretation as a harmonically weighted combination of $\mathrm{Best}@k$ objectives. We arrive at the same solution through an optimization perspective. We analyse how the logarithmic objective avoids gradient suppression near undesirable saturation which prevents a failure mode in expected reward maximization.

Uncertainty quantification for LLMs. Selective classification studies the tradeoff between prediction risk and coverage (Chow, 1970; El-Yaniv & Wiener, 2010; Geifman & El-Yaniv, 2019; Liu et al., 2019). Recent work on LLM uncertainty and abstention has constructed specialized data to train refusal behavior (Zhang et al., 2024; Zhao et al., 2026), elicited verbalized confidence (Kadavath et al., 2022; Lin et al., 2022; Tian et al., 2023; Xiong et al., 2024), estimated uncertainty through repeated sampling (Manakul et al., 2023; Kang et al., 2025; Wang et al., 2023), combined multiple models at inference time (Stengel-Eskin et al., 2024), directly trained models for uncertainty quantification (Damani et al., 2026; Leng et al., 2025; Chandak et al., 2026), or introduced inference-time stopping rules that detect when continued reasoning is unlikely to help (Davidov et al., 2026). Our goal however is different: rather than estimating uncertainty, we aim to *learn how the policy should act under uncertainty*. This distinction becomes important in sequential decision-making settings, where uncertainty quantification is not enough: the fallback policy execution must also be a part of the policy itself and its learnt end-to-end. Moreover, our approach doesn't require specialized datasets, is not LLM-specific, and naturally extends to multi-step interactions.

Learning abstention with RL. Some prior works try to use RL for abstention. For instance, Xu et al. (2024) use knowledge feedback to identify questions outside a model's knowledge boundary and reward rejection on these examples. Wei et al. (2026), Pan et al. (2026), Ma et al. (2026), Wu et al. (2026) and Zhai et al. (2026) directly train for abstention using policy gradient style methods such as GRPO and PPO with and without heuristics such as reward tuning or modifying advantages. However, we show that this procedure can be brittle depending on the model and the task. We show that our approach is more robust and avoids issues associated with PG methods in a principled manner. Moreover, our formulation treats fallback as a general behavior rather than an LLM-specific refusal, allowing it to extend to sequential and non-language decision-making settings.

7 Conclusion and Limitations

We formulated *graceful failure* as an RL problem in which an agent must learn when to pursue task success and when to fall back safely. We showed that naively optimizing the expected reward can be brittle due to the underlying optimization dynamics. To address it, we introduced GracefulRL which provides a suitable landscape for optimization to succeed. Across the settings studied in this work, GracefulRL learns favorable tradeoffs between task success and error, demonstrating that fallback behavior can be learned directly as part of the policy. However, our method is designed for online RL, and we have not studied its behavior under off-policy training. We also do not investigate settings involving coding agents or long horizon agentic training due to constraints on compute. Finally, specifying appropriate fallback behavior can be difficult and may differ substantially across domains. Defining *graceful failure* across different domains, deriving scaling laws for it, and evaluating its generalization to unseen tasks remains an important direction for future research.

AI USE STATEMENT

In this work, we used generative AI tools for assistance in the writing of proofs. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis, interpret results. Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or editing software code and editing the paper to improve readability. We have reviewed all AI-assisted work. All code and proof assistance that was done with AI had a significant involvement with the authors carefully verifying all the details and maintaining sanity checks for them. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made all efforts that the contents of the paper is sufficient to reproduce all of our results. Sections not limited to sections C.3, D.1, D.9 and E.1 can be used to generate our results. Upon publication, we will release the code base and generated assets, such as model checkpoints for the general public.

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. URL http://jmlr.org/papers/v22/19-736.html.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 151–160. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/ahmed19a.html.

Daman Arora, Himanshu Singh, and Mausam. Have LLMs advanced enough? a challenging problem solving benchmark for large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7527–7543, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.468. URL https://aclanthology.org/2023.emnlp-main.468/.

Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006. doi: 10.1198/016214505000000907. URL https://doi.org/10.1198/016214505000000907.

Nikhil Chandak, Shashwat Goel, Ameya Prabhu, Moritz Hardt, and Jonas Geiping. Scaling open-ended reasoning to predict the future, 2026. URL https://openreview.net/forum?id=fkYrply0w3.

C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970. doi: 10.1109/TIT.1970.1054406.

Mehul Damani, Isha Puri, Stewart Slocum, Idan Shenfeld, Leshem Choshen, Yoon Kim, and Jacob Andreas. Beyond binary rewards: Training LMs to reason about their uncertainty. In *International*

Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2507.16806.

Hen Davidov, Nachshon Cohen, Oren Kalinsky, Yaron Fairstein, Guy Kushilevitz, Ram Yazdi, and Patrick Rebeschini. Knowing when to quit: A principled framework for dynamic abstention in LLM reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=R4fBnrKf3h.

Ran El-Yaniv and Yair Wiener. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(53):1605–1641, 2010. URL http://jmlr.org/papers/v11/el-yaniv10a.html.

Yonatan Geifman and Ran El-Yaniv. SelectiveNet: A deep neural network with an integrated reject option. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2151–2159. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/geifman19a.html.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, and Shirong et al Ma. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate, 2025. URL https://arxiv.org/abs/2509.04664.

Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable best-of-n selection for large language models via self-certainty. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=29FRqmVQK8.

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. Abstentionbench: Reasoning LLMs fail on unanswerable questions, 2025. URL https://arxiv.org/abs/2506.09038.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Jixuan Leng, Chengsong Huang, Banghua Zhu, and Jiaxin Huang. Taming overconfidence in LLMs: Reward calibration in RLHF. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=l0tg0jzsdL.

Gen Li, Yuting Wei, Yuejie Chi, Yuantao Gu, and Yuxin Chen. Softmax policy gradient methods can take exponential time to converge. In Mikhail Belkin and Samory Kpotufe (eds.), *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pp. 3107–3110. PMLR, 15–19 Aug 2021. URL https://proceedings.mlr.press/v134/li21b.html.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=8s8K2UZGTZ.

Ziyin Liu, Zhikang Wang, Paul Pu Liang, Russ R Salakhutdinov, Louis-Philippe Morency, and Masahito Ueda. Deep gamblers: Learning to abstain with portfolio theory. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/0c4b1eeb45c90b52bfb9d07943d855ab-Paper.pdf.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level, 2025. Notion Blog.

Xinyu Ma, Ziyang Ding, Zhicong Luo, Chi Chen, Zonghao Guo, Derek F. Wong, Zhen Zhao, Xiaoyi Feng, and Maosong Sun. Karl: Knowledge-aware reasoning and reinforcement learning for knowledge-intensive visual grounding, 2026. URL https://arxiv.org/abs/2503.12797.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfcheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=RwzFNbJ3Ez.

Jincheng Mei and Ian Osband. Delightful gradients accelerate corner escape, 2026. URL https://arxiv.org/abs/2605.11908.

Jincheng Mei, Chenjun Xiao, Bo Dai, Lihong Li, Csaba Szepesvari, and Dale Schuurmans. Escaping the gravitational pull of softmax. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21130–21140. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f1cf2a082126bf02de0b307778ce73a7-Paper.pdf.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, and et. al Alexander Neitz. Openai o1 system card, 2026. URL https://arxiv.org/abs/2412.16720.

Ian Osband. Delightful policy gradient. *Reinforcement Learning Journal*, 7, 2026. URL https://arxiv.org/abs/2603.14608.

Muyu Pan, Shu Zhao, Nan Zhang, Philip Shin, Varun Parekh, Vijaykrishnan Narayanan, and Rui Zhang. Tiar: Trajectory-informed advantage reweighting for llm abstention learning, 2026. URL https://arxiv.org/abs/2605.25850.

Shrinivas Ramasubramanian, Daman Arora, Fahim Tajwar, Guanning Zeng, Qingyang Wu, Zhongzhu Zhou, Chenfeng Xu, Haiwen Feng, Yuda Song, Aarti Singh, Ruslan Salakhutdinov, J. Andrew Bagnell, Jeff Schneider, and Andrea Zanette. Tail-likelihood reinforcement learning, 2026. URL https://arxiv.org/abs/2609.02987.

Noam Razin, Hattie Zhou, Omid Saremi, Vimal Thilak, Arwen Bradley, Preetum Nakkiran, Joshua M. Susskind, and Etai Littwin. Vanishing gradients in reinforcement finetuning of language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IcVNBR7qZi.

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Elias Stengel-Eskin, Peter Hase, and Mohit Bansal. LACIE: Listener-aware finetuning for calibration in large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=RnvgYd9RAh.

Fahim Tajwar, Yiding Jiang, Abitha Thankaraj, Sumaita Sadia Rahman, J Zico Kolter, Jeff Schneider, and Russ Salakhutdinov. Training a generally curious agent. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=UeB3Hdrhda.

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 *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=EeuLO2BjFN.

Miles Team. Miles: Enterprise-grade reinforcement learning for large-scale model post-training. https://github.com/radixark/miles, 2026.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5433–5442, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.330. URL https://aclanthology.org/2023.emnlp-main.330/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Zhepei Wei, Xiao Yang, Kai Sun, Jiaqi Wang, Rulin Shao, Jingxiang Chen, Mohammad Kachuee, Teja Gollapudi, Yiwei Liao, Nicolas SCHEFFER, Rakesh Wanga, Anuj Kumar, Yu Meng, Wen tau Yih, and Xin Luna Dong. TruthRL: Incentivizing truthful LLMs via reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=TnJMJm3Fbi.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL https://mlanthology.org/mlj/1992/williams1992mlj-simple/.

Jiayun Wu, Jiashuo Liu, Zhiyuan Zeng, Tianyang Zhan, Tianle Cai, and Wenhao Huang. Mitigating LLM hallucination via behaviorally calibrated reinforcement learning. In *Agentic AI in the Wild: From Hallucinations to Reliable Autonomy*, 2026. URL https://openreview.net/forum?id=r3ZZLYZTN0.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gjeQKFxFpZ.

Hongshen Xu, Zichen Zhu, Situo Zhang, Da Ma, Shuai Fan, Lu Chen, and Kai Yu. Rejection improves reliability: Training llms to refuse unknown questions using rl from knowledge feedback, 2024. URL https://arxiv.org/abs/2403.18349.

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, 2025. URL https://arxiv.org/abs/2505.09388.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan 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, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Haotian Zhai, Jingcheng Liang, and Dongyeop Kang. Abstain-r1: Calibrated abstention and post-refusal clarification via verifiable RL. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19674–19695, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.985. URL https://aclanthology.org/2026.findings-acl.985/.

Hanning Zhang, Shizhe Diao, Yong Lin, Yi Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. R-tuning: Instructing large language models to say 'I don't know'. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7113–7139, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.394. URL https://aclanthology.org/2024.naacl-long.394/.

Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32(1):56 – 85, 2004. doi: 10.1214/aos/1079120130. URL https://doi.org/10.1214/aos/1079120130.

Yibo Zhao, Jiapeng Zhu, Zichen Ding, and Xiang Li. Grace: Reinforcement learning for grounded response and abstention under contextual evidence, 2026. URL https://arxiv.org/abs/2601.04525.

A Proof of the Saturation Result of Section 3

1. First, we fix terminology and show some simple identities which relate outcome probabilities to conditional score functions in Section A.1.
2. Second, we show that under a boundedness assumption on the logit gradients, a standard regularity condition, the conditional score functions are bounded in Section A.2.
3. Then, we show that this causes the gradient of the expected reward objective to vanish when the policy approaches saturation in Section A.3,

4. Finally, we show that under a non-degeneracy condition, the gradient of the log-barrier based objective does not vanish near sub-optimal saturation in Section A.4.

It is worth noting that our proofs are only for the population level gradients and not for stochastic estimators which we leave for future work.

A.1 SETUP

Throughout we fix a prompt x, a fallback reward of $c \in (0, 1)$, and write $\mathcal{T}_o$ for the set of trajectories whose outcome is $o \in \{S, F, E\}$ (SUCCESS, FALLBACK, and ERROR respectively), and denote $p_o(\theta)$ for its probability under $\pi_\theta(\cdot \mid x)$. Next, we define the *conditional score function*.

$$g_o(\theta) := \mathbb{E}_{\tau \sim \pi_\theta(\cdot|x)} \left[\nabla_\theta \log \pi_\theta(\tau \mid x) \mid \tau \in \mathcal{T}_o\right]$$

Differentiating $p_o = \sum_{\tau \in \mathcal{T}_o} \pi_\theta(\tau \mid x)$ and using $\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$ gives

$$\nabla_\theta p_o(\theta) = p_o(\theta) g_o(\theta), \qquad o \in \{S, F, E\}. \tag{1}$$

It is also worth noting the following identity:

$$\begin{aligned} &\nabla_\theta(p_S + p_F + p_E) = \nabla_\theta(1) = 0, \\ \Rightarrow \quad &p_S g_S + p_F g_F + p_E g_E = 0. \end{aligned} \tag{2}$$

Identity 1 shows that the gradient of an outcome probability is the *probability* of outcome p_Y times a *direction* g_Y.

Also, we define the softmax parameterization. We consider policies parameterized by softmax over a network producing logits $z_\theta(s_t) \in \mathbb{R}^{|\mathcal{A}|}$:

$$\pi_\theta(a_t \mid s_t) = \frac{\exp(z_\theta(s_t)_{a_t})}{\sum_{a' \in \mathcal{A}} \exp(z_\theta(s_t)_{a'})}$$

Autoregressive language models are a special case where $\mathcal{A}$ is the vocabulary of the model and $s_t = (x, a_{<t})$ is the first t generated tokens along with the prompt. Next, we show that a common assumption of boundedness of logit gradients leads to boundedness of the conditional score function under the softmax parameterization.

A.2 BOUNDED CONDITIONAL SCORES

Assumption 1 (Bounded logit gradients)**.** *There exists a constant $B < \infty$ such that, for every $\theta \in \Theta$, every state s, and every discrete action $a \in \mathcal{A}$,*

$$\|\nabla_\theta z_\theta(s)_a\| \le B.$$

Lemma 1 (Bounded conditional scores)**.** *Under Assumption 1, and writing $G := 2HB$,*

$$\|g_o(\theta)\| \le G, \qquad o \in \{S, F, E\}, \tag{3}$$

for every $\theta \in \Theta$.

Proof. For the softmax policy,

$$\nabla_\theta \log \pi_\theta(a \mid s) = \nabla_\theta z_\theta(s)_a - \sum_{a' \in \mathcal{A}} \pi_\theta(a' \mid s) \nabla_\theta z_\theta(s)_{a'}.$$

The second term is a convex combination of vectors of norm at most B, so it has norm at most $2B$

$$\Rightarrow \|\nabla_\theta \log \pi_\theta(a \mid s)\| \le 2B.$$

Summing over an entire trajectory τ which is at most H steps,

$$\|\nabla_\theta \log \pi_\theta(\tau \mid x)\| \le 2HB = G.$$

Each g_Y is a conditional expectation of these trajectory scores, hence a convex combination of vectors of norm at most G, which gives equation 3. □

A.3 Policy Gradient Vanishes Near Saturation

First, we express the policy gradients in terms of the conditional scores. The objective function for policy gradient is the expected reward $p_S + cp_F$. Using identity 1 we get

$$\begin{aligned} \mathcal{J}_{\mathrm{pg}}(x) &= p_S + cp_F, \\ \Rightarrow \nabla_\theta \mathcal{J}_{\mathrm{pg}}(x) &= p_S\, g_S + c\, p_F\, g_F \end{aligned} \tag{4}$$

Proposition 1 (Gradient suppression under fallback saturation)**.** *Suppose Assumption 1 holds. Then for every $\theta \in \Theta$, and every outcome $o \in \{\textsc{Success}, \textsc{Fallback}, \textsc{Error}\}$, as the policy saturates on any outcome, that is, when $p_o \to 1$*

$$\|\nabla_\theta \mathcal{J}_{\mathrm{pg}}(\theta)\| \le G\big(1 - p_o(x)\big),$$

Proof. Under Assumption 1, the conditional scores satisfy $\|g_S\|, \|g_F\|, \|g_E\| \le G$. We consider the three possible saturated outcomes separately.

If $o = \textsc{Error}$, then from Eq. 4,

$$\begin{aligned} \|\nabla_\theta \mathcal{J}_{\mathrm{pg}}\| &= \|p_S g_S + cp_F g_F\| \\ &\le G(p_S + cp_F) \\ &\le G(p_S + p_F) = G(1 - p_E). \end{aligned}$$

If $o = \textsc{Fallback}$, using $p_F g_F = -p_S g_S - p_E g_E$ from equation 2 gives $\nabla_\theta \mathcal{J}_{\mathrm{pg}} = (1-c)p_S g_S - cp_E g_E$. Hence,

$$\begin{aligned} \|\nabla_\theta \mathcal{J}_{\mathrm{pg}}\| &\le G\big((1-c)p_S + cp_E\big) \\ &\le G(p_S + p_E) = G(1 - p_F). \end{aligned}$$

Finally, if $o = \textsc{Success}$, using $p_S g_S = -p_F g_F - p_E g_E$ from equation 2 gives $\nabla_\theta \mathcal{J}_{\mathrm{pg}} = -(1-c)p_F g_F - p_E g_E$, and therefore

$$\begin{aligned} \|\nabla_\theta \mathcal{J}_{\mathrm{pg}}\| &\le G\big((1-c)p_F + p_E\big) \\ &\le G(p_F + p_E) = G(1 - p_S). \end{aligned}$$

Thus, for any $o \in \{\textsc{Success}, \textsc{Fallback}, \textsc{Error}\}$, $\|\nabla_\theta \mathcal{J}_{\mathrm{pg}}\| \le G(1 - p_o)$, which goes to zero as $p_o \to 1$. □

A.4 GRADIENT OF THE LOG-BARRIER OBJECTIVE DOES NOT VANISH

Assumption 2 (Non-degeneracy)**.** *There exist $\varepsilon, \delta_F, \delta_E > 0$ such that for all $\theta \in \Theta$:*

$$p_F(\theta) \geq 1 - \varepsilon \Rightarrow \|g_S(\theta)\| \geq \delta_F,$$

and

$$p_E(\theta) \geq 1 - \varepsilon \Rightarrow \inf_{\lambda \in [1-c,1]} \|\lambda g_S + (1-\lambda) g_F\| \geq \delta_E$$

.

Intuitively, this assumption requires that, near a degenerate policy, there remains some parameter direction along which probability mass can be moved toward a better outcome. When the policy is almost entirely concentrated on FALLBACK, the success gradient must remain nonzero, so the model can still move probability mass toward SUCCESS. Similarly, when the policy is almost entirely concentrated on ERROR, the gradients associated with SUCCESS and FALLBACK must not cancel for any weighting induced by the objective.

Note that Assumption 2 does not change the implication of Proposition 1. That result is an upper bound on the policy-gradient norm and follows from Assumption 1 alone. The non-degeneracy condition is needed here only to obtain a lower bound on the log-barrier gradient.

First, we explain that this assumption is a reasonable one to make. Assumption 2 constrains the relevant score directions only near the two undesirable saturated policies. A single-step softmax policy illustrates why these directions need not vanish even when the corresponding outcomes are rare. Consider three possible actions, one for each outcome, with trainable logits $z = (z_S, z_F, z_E)$ and probabilities $p = \mathrm{softmax}(z)$. Since each outcome identifies a single action, its conditional score is

$$g_Y = \nabla_z \log p_Y = e_Y - p, \qquad Y \in \{S, F, E\},$$

where e_Y is the corresponding standard basis vector. Near fallback saturation, $p \to e_F$. Therefore,

$$g_S = e_S - p \longrightarrow e_S - e_F, \qquad \|g_S\| \longrightarrow \sqrt{2}.$$

Thus, the score conditioned on success remains bounded away from zero even as success becomes arbitrarily unlikely. Similarly, near error saturation, $p \to e_E$. Then, for every $\lambda \in [0, 1]$, a convex combination of the success and fallback scores satisfies

$$\lambda g_S + (1-\lambda) g_F \longrightarrow \lambda e_S + (1-\lambda) e_F - e_E.$$

The limiting squared norm for this is $\lambda^2 + (1-\lambda)^2 + 1$, whose minimum over $\lambda \in [0, 1]$ is $3/2$. Hence the second direction is also bounded away from zero. By continuity, positive constants δ_F and δ_E therefore exist in sufficiently small neighborhoods of the two saturated policies. For a general parameterization, Assumption 2 requires that the logit Jacobian not annihilate these same directions.

Proposition 2 (The log-barrier objective retains a non-vanishing signal)**.** *Suppose Assumptions 1 and 2 hold, and let $c \in (0, 1)$. For either suboptimal outcome $o \in \{F, E\}$, as the policy saturates on that outcome, i.e., as $p_o(x) \to 1$,*

$$\|\nabla_\theta \mathcal{J}_{log}(\theta)\| = \Theta(1).$$

Proof. We have

$$\begin{aligned} \mathcal{J}_{log} &= (1-c) \log p_s + c \log(1 - p_E) \\ \Rightarrow \nabla_\theta \mathcal{J}_{log} &= (1-c) g_S - c \frac{p_E}{1 - p_E} g_E. \end{aligned} \tag{5}$$

We consider the two saturated outcomes separately. First, suppose $p_F \to 1$. The second expression in equation 5 shows that the logarithm has cancelled the factor p_S, leaving the success direction with a constant coefficient. For p_F sufficiently close to one, using the reverse triangle inequality,

$$\begin{aligned} \|\nabla_\theta \mathcal{J}_{log}\| &\geq (1-c)\|g_S\| - c \frac{p_E}{1 - p_E} \|g_E\| \\ &\geq (1-c)\delta_F - cG \frac{p_E}{1 - p_E}, \end{aligned} \tag{6}$$

but also by the triangle inequality,

$$\|\nabla_\theta \mathcal{J}_{log}\| \leq (1-c)G + cG\frac{p_E}{1-p_E}. \tag{7}$$

Since $p_E/(1-p_E) \to 0$ when $p_F \to 1$ these bounds imply

$$\liminf_{p_F \to 1} \|\nabla_\theta \mathcal{J}_{\log}\| \;\geq\; (1-c)\,\delta_F \;>\; 0, \qquad \limsup_{p_F \to 1} \|\nabla_\theta \mathcal{J}_{\log}\| \;\leq\; (1-c)\,G.$$

Now, for the case when error is saturating, using identity 2, we get,

$$\nabla_\theta \mathcal{J}_{log} = (1-c)g_S - c\frac{p_E}{1-p_E}g_E \tag{8}$$

$$= (1-c)g_S + c\frac{p_S g_S + p_F g_F}{p_S + p_F} \tag{9}$$

$$= (1 - c + c\frac{p_S}{p_S + p_F})g_S + c\frac{p_F}{p_F + p_S}g_F \tag{10}$$

To simplifying some notation, we define:

$$\lambda_\theta = \frac{p_S}{p_S + p_F} \in [0,1], \qquad \widetilde{\lambda}_\theta = (1-c) + c\lambda_\theta \in [1-c, 1].$$

The first expression in equation 5 becomes

$$\nabla_\theta \mathcal{J}_{log} = \widetilde{\lambda}_\theta g_S + (1 - \widetilde{\lambda}_\theta)g_F.$$

For p_E sufficiently close to one, Assumption 2 and Lemma 1 therefore give

$$\delta_E \leq \|\nabla_\theta \mathcal{J}_{log}\| \leq \widetilde{\lambda}_\theta \|g_S\| + (1 - \widetilde{\lambda}_\theta)\|g_F\| \leq G.$$

Thus the gradient is bounded above and away from zero near either suboptimal saturated outcome. Hence, for $o \in \{F, E\}$,

$$\boxed{\|\nabla_\theta \mathcal{J}_{log}\| = \Theta(1) \quad \text{as } p_o \to 1.}$$

□

B EFFECTS OF THE ϵ PARAMETER

The reciprocal weights in Algorithm 1 can have high variance when success or non-error trajectories are rare. At fixed N, small ϵ keeps the expected update close to the unsmoothed finite-rollout target; as N grows, its rescaled gradient approaches $\nabla_\theta \mathcal{J}_{\log}$. Larger ϵ shrinks and equalizes the denominators, reducing variance while introducing bias relative to the $\epsilon = 0$ update.

First, we prove that the bias is linear in the smoothing factor ϵ. Let us define $\hat{g}(\epsilon)$ to be the stochastic gradient obtained from using Algorithm 1. Let us define its expected value as $g(\epsilon)$. Then, we show the following:

Proposition 3. *The bias introduced by smoothing, is linear in* ϵ

$$\|g(\epsilon) - g(0)\| = O(\epsilon) \tag{11}$$

First, we prove a lemma.

Lemma 2. *Let* N_S, N_F, N_E *denote the numbers of* SUCCESS, FALLBACK, *and* ERROR *samples among* N *rollouts, where* $N = N_S + N_F + N_E$. *Let* $a_{i,\epsilon}$ *denote the centered advantage used by Algorithm 1 with smoothing parameter* $\epsilon \geq 0$. *Then, for every rollout* i,

$$a_{i,\epsilon} - a_{i,0} = O(\epsilon) \qquad \text{as } \epsilon \to 0.$$

Note that $a_{i,\epsilon}$ *is a random variable where the randomness is coming from the rollouts sampled from the policy, conditioned on* N_S, N_F, *and* N_E.

Proof. The centered advantage takes one of three forms depending on the outcome of rollout i:

$$a_{i,\epsilon} = \begin{cases} \dfrac{(1-c)(N_F+N_E)}{N(N_S+\epsilon)} + \dfrac{cN_E}{N(N_S+N_F+\epsilon)}, & o_i = \text{SUCCESS}, \\ -\dfrac{(1-c)N_S}{N(N_S+\epsilon)} + \dfrac{cN_E}{N(N_S+N_F+\epsilon)}, & o_i = \text{FALLBACK}, \\ -\dfrac{(1-c)N_S}{N(N_S+\epsilon)} - \dfrac{c(N_S+N_F)}{N(N_S+N_F+\epsilon)}, & o_i = \text{ERROR}. \end{cases}$$

Note that in the ratios above, a ratio is understood as 0 when it is of the $0/0$ form.

Then, for any integer $m \geq 1$, $\frac{1}{m+\epsilon} - \frac{1}{m} = -\frac{\epsilon}{m(m+\epsilon)} = O(\epsilon)$. Therefore, every nonzero term appearing in $a_{i,\epsilon}$ changes by $O(\epsilon)$ when ϵ is perturbed from 0 to ϵ. Since each centered advantage contains only finitely many such terms, $a_{i,\epsilon} - a_{i,0} = O(\epsilon)$.

Zero-count cases can also be handled this way. If $N_S = 0$, all terms proportional to $N_S/(N_S+\epsilon)$ vanish. Likewise, if $N_S + N_F = 0$, all terms proportional to $(N_S+N_F)/(N_S+N_F+\epsilon)$ vanish. Thus, for a fixed count configuration (N_S, N_F, N_E), we have $a_{i,\epsilon} - a_{i,0} = O(\epsilon)$. □

Proof. Now, we prove Proposition 3. First let us define the following conditional expectation:

$$g_\epsilon(N_S, N_F, N_E) := \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} a_{i,\epsilon} s_i \;\middle|\; N_S, N_F, N_E\right].$$

From Assumption 1, we have that $\|s_i\| \leq G$,, then for any fixed configuration of counts (N_S, N_F, N_E) we obtain:

$$\begin{aligned} \|g_\epsilon(N_S,N_F,N_E) - g_0(N_S,N_F,N_E)\| &\leq \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} |a_{i,\epsilon} - a_{i,0}|\,\|s_i\| \;\middle|\; N_S,N_F,N_E\right] \\ &\leq G \cdot \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} |a_{i,\epsilon} - a_{i,0}| \;\middle|\; N_S,N_F,N_E\right] = O(\epsilon). \end{aligned}$$

Note that for fixed N, there are only finitely many possible count configurations (N_S, N_F, N_E). Therefore, by the law of total expectation,

$$\|g_\epsilon - g_0\| = \left\| \sum_{\substack{N_S,N_F,N_E \\ N_S+N_F+N_E=N}} \mathbb{P}(N_S, N_F, N_E) \left[g_\epsilon(N_S, N_F, N_E) - g_0(N_S, N_F, N_E) \right] \right\| \tag{12}$$

$$\leq \sum_{\substack{N_S,N_F,N_E \\ N_S+N_F+N_E=N}} \mathbb{P}(N_S, N_F, N_E) \left\| g_\epsilon(N_S, N_F, N_E) - g_0(N_S, N_F, N_E) \right\| \tag{13}$$

$$= O(\epsilon). \tag{14}$$

where the last step follows because each conditional term is $O(\epsilon)$ and there are only finitely many count configurations. Thus, the smoothing parameter ϵ introduces only an $O(\epsilon)$ bias in the expected gradient. □

If $V(\epsilon)$ denotes the variance of the estimator as a function of ϵ and $V'(0^+) < 0$, then for sufficiently small $\epsilon > 0$,

$$V(\epsilon) = V(0) + V'(0^+)\epsilon + o(\epsilon) < V(0).$$

Since Proposition 3 gives $|g(\epsilon) - g(0)| = \mathcal{O}(\epsilon)$, the squared bias is only $\mathcal{O}(\epsilon^2)$. Therefore, for sufficiently small $\epsilon > 0$, the linear decrease in variance dominates the quadratic increase in squared bias, yielding lower MSE. Empirically, we also find that using a small nonzero ϵ improves performance when training using smaller amounts of compute budget.

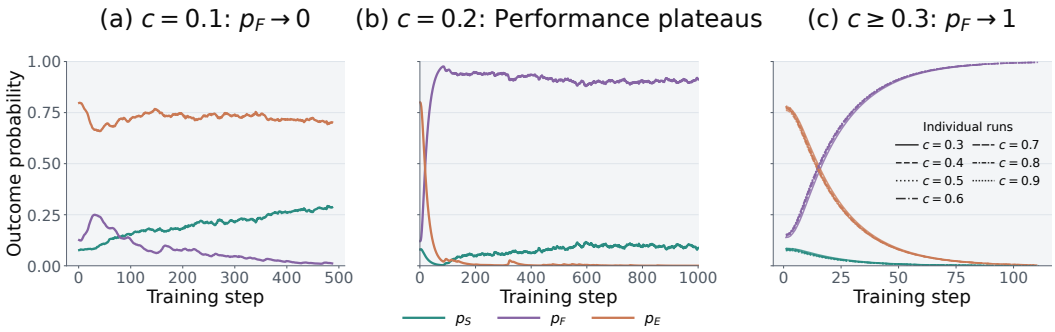


Figure 8: **RLOO with fixed fallback rewards on SafeMaze.** We measure p_S, p_F, p_E during training for $c = 0.1$, $c = 0.2$, and $c \geq 0.3$. Empirically, we observe that when $c \ll p_S$, the policy can lead to no coverage on fallback trajectories, that is, $p_F \to 0$. Similarly, when $c \gg p_S$, the policy can lead to a degenerate policy where it always goes to the SAFE square.

C SafeMaze Extended Results

C.1 SafeMaze details

For the SafeMaze environment, we generate a 15×15 grid and pretrain the policy using a mixture of GOAL-seeking and SAFE-square-seeking trajectories. We ensure that the GOAL is always reachable in the grid within the time limit.

C.2 RLOO with a Fixed Fallback Reward

Figure 8 shows why selecting a fixed reward for fallback when using policy gradient is difficult. With $c = 0.1$, the fallback probability tends to zero, so the policy continues attempting the goal but incurs a high error rate. At $c = 0.2$, close to the policy's initial pass rate, RLOO retains nonzero success, but its performance plateaus while fallback remains the dominant outcome. For every tested reward $c \geq 0.3$, the policy instead converges rapidly to the fallback-only solution, with $p_F \to 1$ and $p_S \to 0$. Thus, a fixed reward gives useful behavior only in a narrow range and does not reliably support continued improvement. This motivates the online reward-tuning baseline, which sets the fallback reward relative to the policy's current success rate rather than choosing a single value for the entire run.

C.3 Hyperparameters for the Main SafeMaze Comparison

For every run in Figure 3 **(c)**, we use 32 environments and 64 rollouts and train for 20,000 optimization steps. For our method, we sweep the fallback reward $c \in \{0, 0.2, 0.4, 0.6, 0.8\}$. For DPG, we set $\eta = 0.9$ and use the same safe-reward sweep. For the online reward-tuning baseline, we sweep the multiplier $\alpha \in \{1, 0.8, 0.6, 0.4, 0.2, 0\}$ in $c_t = \alpha \widehat{p}_{S,t}$.

C.4 Varying Rollout Counts and Safe Rewards

We jointly sweep the number of rollouts per update $N \in \{16, 32, 64\}$ and the reward for reaching SAFE, $c \in \{0, 0.2, 0.4, 0.6, 0.8\}$. Figure 9**(a)** plots the resulting reliability frontiers after 5,000 training steps. We observe that increasing N improves the reliability frontier. This can be explained by the fact that larger rollout counts provide a higher-order truncation of the log objective and are more likely to include rare successful trajectories, producing a more informative update.

The effect of rollout count becomes more pronounced as the fallback reward increases. At $c = 0.8$, the 16- and 32-rollout runs collapse to the fallback-only solution, whereas 64 rollouts retain nonzero success. Thus, settings that place greater value on reliability also require more rollouts to approximate a higher order approximation of the objective.

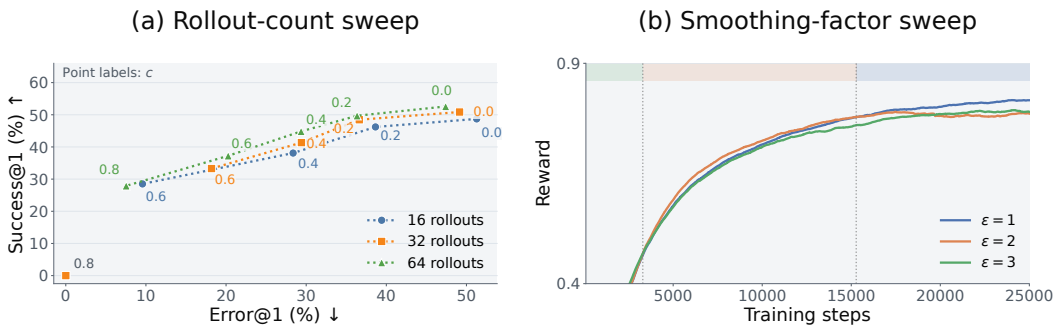


Figure 9: **SafeMaze rollout and smoothing sweeps. (a)** Reliability frontiers after 5,000 steps for 16, 32, and 64 rollouts. **(b)** Training reward with 32 rollouts and $c = 0.4$: larger ϵ helps initially, whereas $\epsilon = 1$ performs best later. The colored bands mark the highest-reward curve in each training interval pointing to the general principle: when bounded by compute, larger ϵ helps in achieveing higher reward.

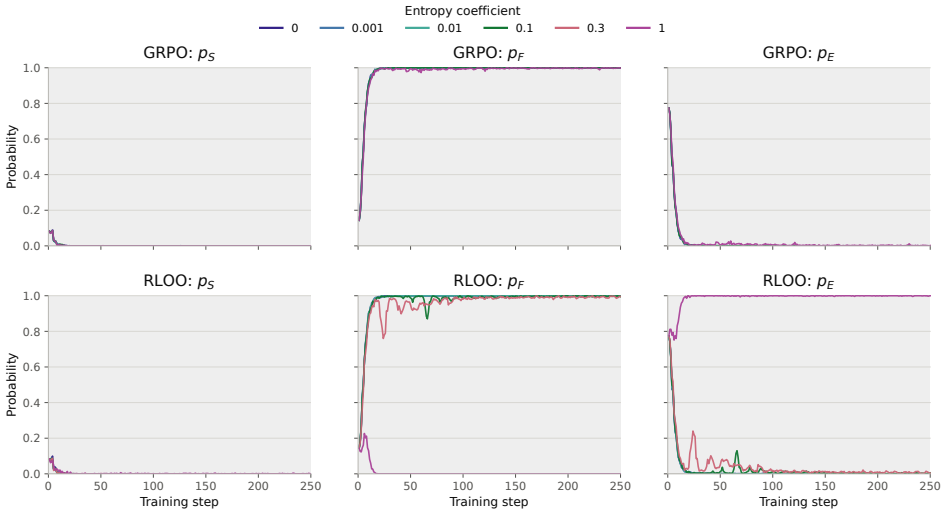


Figure 10: Entropy-bonus sweep for GRPO (top) and RLOO (bottom) on SafeMaze. Columns show the probabilities of success (p_S), fallback (p_F), and error (p_E) during training. None of the tested entropy coefficients preserves task success: GRPO and most RLOO runs collapse to fallback, while RLOO with coefficient 1 collapses to error.

C.5 Smoothing-Parameter ϵ Effects

As discussed in Section B, ϵ smoothing can lower variance at the cost of some bias. Figure 9**(b)** shows the corresponding optimization tradeoff: larger smoothing factors improve reward earlier in training, while smaller values become preferable with more compute.

C.6 Entropy-Bonus Tuning

We also sweep the entropy-bonus strength for GRPO and RLOO while fixing the safe reward to $c = 0.4$ and holding everything else fixed. Entropy regularization does not yield a meaningful policy in this regime. Figure 10 shows that every setting drives p_S to zero within a few optimization steps. GRPO converges to the fallback-only policy across the entire sweep. RLOO does the same for entropy coefficients up to 0.3, while the largest coefficient, 1, instead converges to $p_E = 1$. The entropy bonus encourages stochasticity over actions, but it still suffers from the vanishing gradient problem near a saturated policy (Agarwal et al., 2021; Ahmed et al., 2019).

D MATHEMATICAL-REASONING TRAINING DETAILS AND VALIDATION

D.1 TRAINING HYPERPARAMETERS

For training, we use the verl (Sheng et al., 2024; 2025) framework. Tables 1 and 2 give the complete training configuration for the comparison of Qwen3-4B-Base and Qwen3-1.7B-Base. We match all shared optimization and sampling hyperparameters across methods. RLCR uses RLOO with the calibration reward defined in Section 5.2, whereas our method uses the estimator in Algorithm 1 with $\epsilon = 1$ and jointly trains on the five conditioning levels.

Hyperparameter	**RLCR**	**Ours**
Gradient estimator	RLOO	GracefulRL
Training data	DAPO-Math-17K	DAPO-Math-17K
Fallback utilities c	–	$\{0, 0.2, 0.4, 0.6, 0.8\}$
Smoothing ϵ	–	1
Optimizer	AdamW	AdamW
Learning rate / schedule	10^{-6} / constant	10^{-6} / constant
Warmup / weight decay	none / 0.01	none / 0.01
Prompts $\times$ rollouts per update	32×32	32×32
Maximum prompt / response length	2,048 / 4,096	2,048 / 4,096
Rollout temperature / top-p	1.0 / 1.0	1.0 / 1.0
Policy epochs / minibatch size	1 / 32	1 / 32
Microbatch size per GPU	2	2
Policy clip ratio / gradient clipping	0.2 / 1	0.2 / 1
Entropy coefficient / KL loss	0 / disabled	0 / disabled
Loss aggregation	seq-mean-token-sum-norm	seq-mean-token-sum-norm
Precision / sharding	bfloat16 / FSDP	bfloat16 / FSDP
Training hardware	2×8 H100GPUs	2×8 H100GPUs
Training steps / checkpoint interval	1,000 / 200	1,000 / 200
Validation interval	50 steps	50 steps
Validation rollouts / temperature / top-p	32 / 0.6 / 0.95	32 / 0.6 / 0.95

Table 1: Training hyperparameters for the Qwen3-4B-Base RLCR and GRACEFULRL runs. A dash denotes a method-specific parameter that does not apply.

Hyperparameter	**RLCR**	**Ours**
Gradient estimator	RLOO	GracefulRL
Training data	DAPO-Math-17K	DAPO-Math-17K
Fallback utilities c	–	$\{0, 0.2, 0.4, 0.6, 0.8\}$
Smoothing ϵ	–	1
Optimizer	AdamW	AdamW
Learning rate / schedule	10^{-6} / constant	10^{-6} / constant
Warmup / weight decay	none / 0.01	none / 0.01
Prompts $\times$ rollouts per update	32×32	32×32
Maximum prompt / response length	2,048 / 4,096	2,048 / 4,096
Rollout temperature / top-p	1.0 / 1.0	1.0 / 1.0
Policy epochs / minibatch size	1 / 32	1 / 32
Microbatch size per GPU	2	2
Policy clip ratio / gradient clipping	0.2 / 1	0.2 / 1
Entropy coefficient / KL loss	0 / disabled	0 / disabled
Loss aggregation	seq-mean-token-sum-norm	seq-mean-token-sum-norm
Precision / sharding	bfloat16 / FSDP	bfloat16 / FSDP
Training hardware	1×8 H100GPUs	1×8 H100GPUs
Training steps / checkpoint interval	1,500 / 300	1,500 / 300
Validation interval	50 steps	50 steps
Validation rollouts / temperature / top-p	32 / 0.6 / 0.95	32 / 0.6 / 0.95

Table 2: Training hyperparameters for the Qwen3-1.7B-Base RLCR and GRACEFULRL runs. A dash denotes a method-specific parameter that does not apply.

D.1.1 GRACEFULRL CONDITIONING SUFFIXES

We append one of the following suffixes to each mathematical problem and use the corresponding fallback utility c when computing its training reward. All five variants are mixed during training.

c	**Conditioning suffix**
0.0	"Be very optimistic and always attempt an answer, even if you are unsure."
0.2	"Be optimistic but stay mindful of potential mistakes."
0.4	"Be balanced, leaning towards attempting an answer."
0.6	"Be moderately careful and verify your approach before committing to an answer."
0.8	"Be quite careful and avoid guessing."

Table 3: Conditioning suffixes for the five GracefulRL fallback utilities.

D.1.2 RLCR TRAINING PROMPT

For the RLCR baseline, we replace {problem} with the mathematical question and use the following prompt verbatim:

RLCR prompt

```
{problem}
Please reason step by step, and put your final answer in the following format:
<answer>\boxed{<final answer>}</answer>
Then, reason about how confident you are in your answer using the following scale:
- 0.1: You are guessing or have very little idea
- 0.2: You are quite unsure, the approach might be wrong
- 0.3: You are somewhat unsure, there are likely errors
- 0.4: You are slightly unsure, some steps feel shaky
- 0.5: You could go either way
- 0.6: You are slightly confident, the approach seems right
- 0.7: You are fairly confident, most steps check out
- 0.8: You are quite confident, only minor doubt remains
- 0.9: You are very confident and have verified key steps
- 1.0: You are certain the answer is correct
Report your confidence as:
<confidence>\boxed{<your score>}</confidence>
```

Unlike the RLCR paper, we find that Qwen3 models can learn to follow this instruction after a couple of training steps and there is no need for extra SFT data to warm start the instruction following.

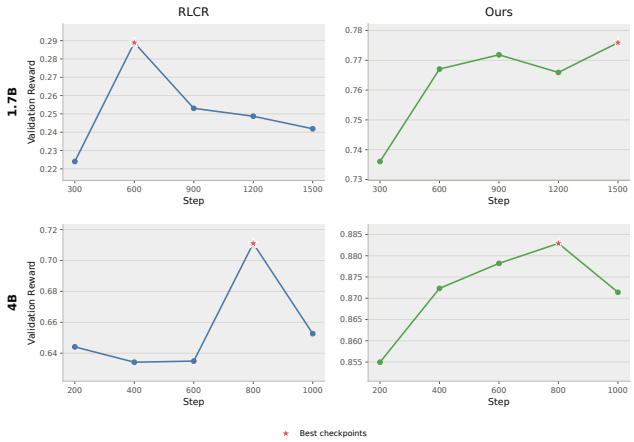


Figure 11: MATH500 validation reward across training checkpoints for RLCR (left) and our conditioned GRACEFULRL method (right), at the 1.7B (top) and 4B (bottom) model scales. Stars mark the checkpoints selected for the final evaluation.

D.2 VALIDATION AND CHECKPOINT SELECTION

We evaluate checkpoints on the MATH500 validation set using 32 sampled responses per problem. For our conditioned model, we compute the reward at each of the five conditioning levels and average across them to construct the validation metric. For RLCR, we use the mean Brier reward. As shown in Figure 11, the selected Qwen3-1.7B-Base checkpoints are step 1,500 for our method and step 600 for RLCR. For Qwen3-4B-Base, both methods attain highest validation reward at step 800. We select these checkpoints before evaluating on the test suite.

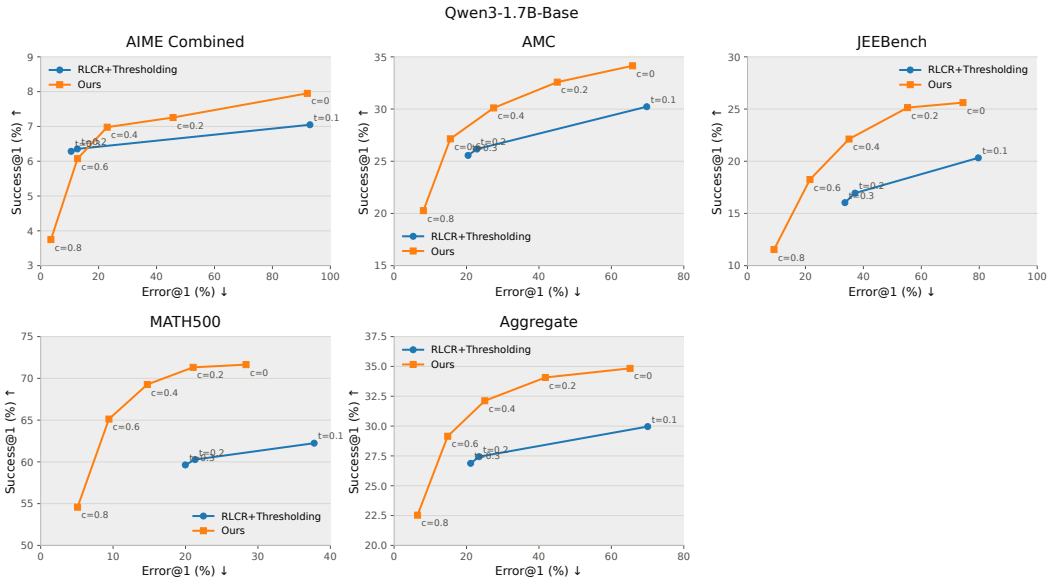


Figure 12: Per-dataset reliability frontiers for RLCR and GracefulRL (Ours) with Qwen3-1.7B-Base. The final panel reports the aggregate frontier across datasets.

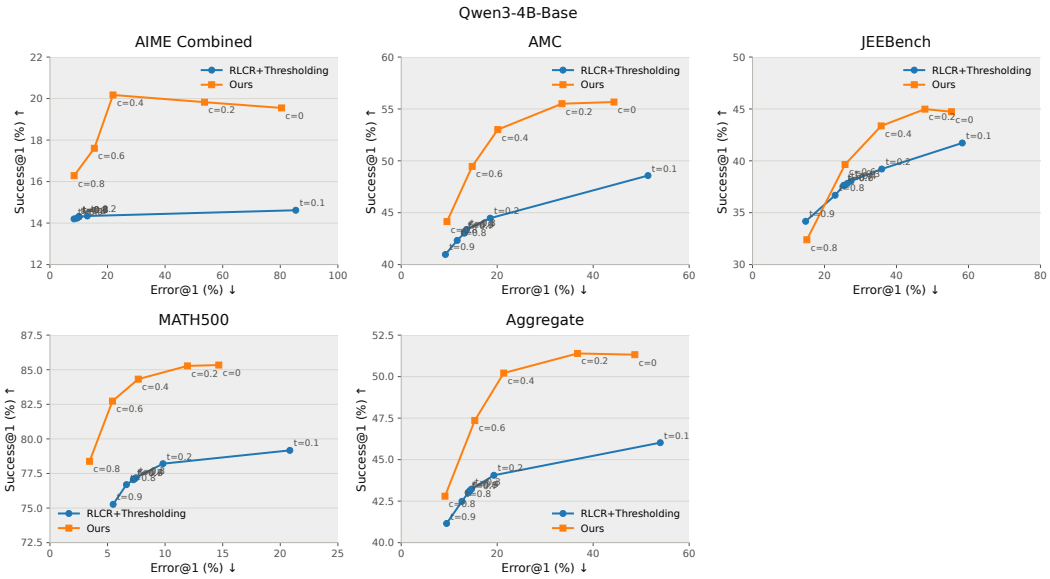


Figure 13: Per-dataset reliability frontiers for RLCR and GracefulRL (Ours) with Qwen3-4B-Base. The final panel reports the aggregate frontier across datasets.

D.3 Validation on All Datasets

Figures 12 and 13 expand the aggregate comparison in Figure 4 into separate reliability frontiers for AIME, AMC, JEEBench and MATH500.

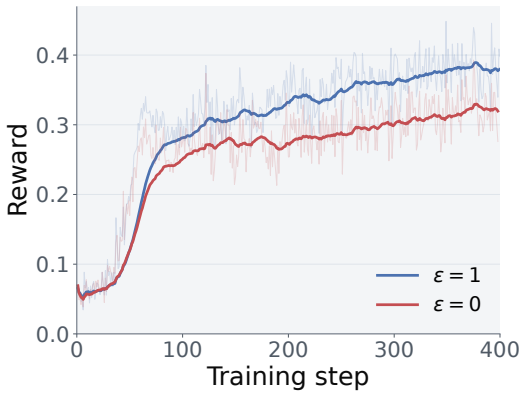


Figure 14: **Effect of ϵ in language-model RL.** Training reward for Qwen3-1.7B-Base with fallback reward $c = 0.4$ over the first 400 optimization steps. Pale lines show raw reward and dark lines show the smoothed reward.



Figure 15: Training dynamics of the Qwen3-4B-Base RLCR policy on AIME. Although the Brier score generally improves (decreases) throughout training, answer accuracy peaks early and then declines. The RLCR reward combines these two quantities and therefore need not track accuracy.

D.4 Smoothing with a Limited Language-Model RL Budget

In Language-model RL, we are bounded by compute since training for each step is expensive. Therefore, in this setting, the smoothing parameter ϵ is consequential. Figure 14 compares GRACEFULRL with $\epsilon = 0$ and $\epsilon = 1$ on Qwen3-1.7B-Base at fallback reward $c = 0.4$. $\epsilon = 1$ consistently attains higher training reward throughout the first 400 steps compared to $\epsilon = 0$.

D.5 RLCR Can Trade Accuracy for Calibration

Figure 15 reveals that optimizing RLCR can trade task accuracy for improved calibration: after accuracy peaks around step 400, the Brier score continues to improve while accuracy falls. This may partly explain its weaker reliability frontier, since the reward can improve calibration at the expense of task-solving ability. We hypothesize that optimizing calibration with methods such as TailRL could preserve more accuracy, and leave for future work. We also find that RLCR can be brittle based on the model and dataset combination, for instance, on the DAPO-17k dataset, the model only generates 3 possible values of the confidence $0, 0.1$ and 0.2. However, this problem is not present in the 4B model.

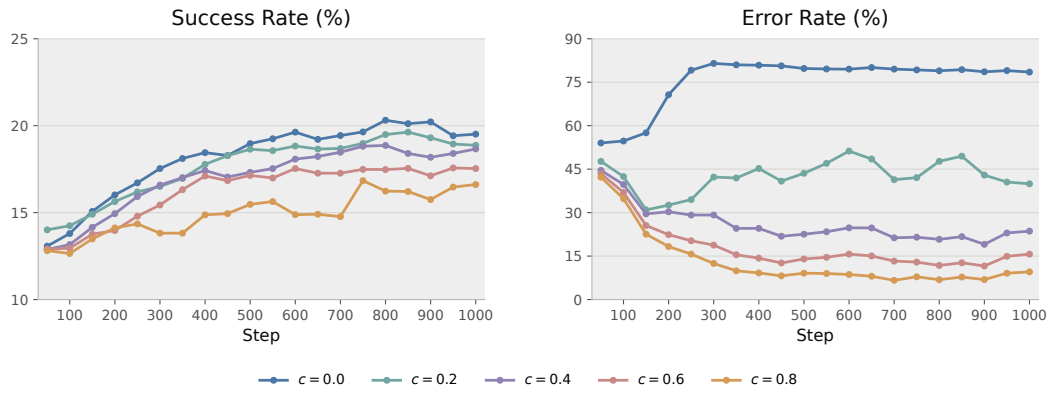


Figure 16: **Success and error dynamics on AIME.** Qwen3-4B-Base trained with GRACEFULRL, evaluated at $c \in \{0, 0.2, 0.4, 0.6, 0.8\}$. The left and right panels show p_S and p_E, respectively, as percentages. Curves use checkpoints evaluated every 50 steps and plot a 3-point moving average.

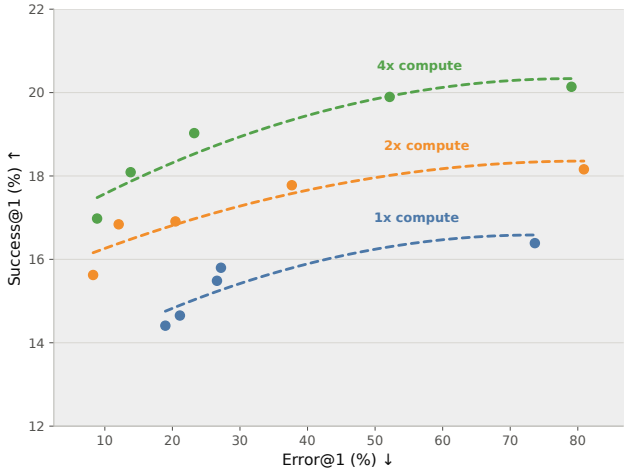


Figure 17: **Effect of training compute on the AIME reliability frontier.** Qwen3-4B-Base evaluated after 200 (1×), 400 (2×), and 800 (4×) training steps. Each marker is for a different c value and the dashed curves fit the resulting frontiers.

D.6 SUCCESS AND ERROR DYNAMICS ON AIME

Figure 16 tracks the success and error rates of Qwen3-4B-Base on the combined AIME set at each conditioning level. Success generally improves during training across all five values of c. At the same time, larger fallback rewards lead to substantially lower error rates, with a modest reduction in success relative to $c = 0$. Thus, conditioned training learns different degrees of caution while continuing to improve task-solving performance.

D.7 RELIABILITY FRONTIERS WITH MORE TRAINING COMPUTE

Figure 17 compares the Qwen3-4B-Base policy on the combined AIME set after 200, 400, and 800 optimization steps, corresponding to 1×, 2×, and 4× training compute. Each frontier sweeps the same five fallback rewards $c \in \{0, 0.2, 0.4, 0.6, 0.8\}$. Across their shared range of error rates, the frontier moves uniformly upward with more compute: at a given Error@1, the policy achieves higher Success@1. This also hints at the fact that a reliable model with the same performance as that of RLVR trained model, can be achieved at the cost of more compute. Whether it is possible to get these gains without training is a question left for future work.

D.8 ABSTENTION REASONS

Here, we show some CoTs generated by the model after training, and highlight some notable verbalizations in orange.

ALGEBRA / COMPUTATION EXPLODED

Prompt (risk cue $c = 0.4$)

Let a, b, and n be positive integers with both a and b greater than or equal to 2 and less than or equal to $2n$. Define an $a \times b$ cell loop in a $2n \times 2n$ grid of cells to be the $2a + 2b - 4$ cells that surround an $(a - 2) \times (b - 2)$ (possibly empty) rectangle of cells in the grid. For example, the following diagram shows a way to partition a 6×6 grid of cells into 4 cell loops.
— P P P P — Y Y — — P — R R — P — Y — Y — — P — R R — P — Y — Y — — P P P P — Y — Y — — G G G G — Y — Y — — G G G G — Y Y —
Find the number of ways to partition a 10×10 grid of cells into 5 cell loops so that every cell of the grid belongs to exactly one cell loop.

Please reason step by step.

If you are confident in the answer, output:

`\boxed{<final answer>}`

If you are not confident or cannot solve the problem, output:

`\boxed{\text{I don't know}}`

Be balanced, leaning towards attempting an answer.

Think step by step before committing to an answer.

Response

Alright, I'm tackling this problem step by step.
Understanding the Problem:
We have a 10×10 grid, and we need to partition it into 5 cell loops. Each cell loop is defined as an $a \times b$ cell loop, where a and b are positive integers such that $2 \leq a, b \leq 2n$. In this case, $n = 5$ (since $2n = 10$), so $a, b \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$.
The total number of cells in the grid is 100. Each cell loop consists of $2a + 2b - 4$ cells. Therefore, the total number of cells in all 5 loops should be 100:

$$5(2a + 2b - 4) = 100 \implies 2a + 2b - 4 = 20 \implies a + b = 12$$

So, for each loop, the sum of a and b must be 12. The possible pairs (a, b) are:

$$(2, 10), (3, 9), (4, 8), (5, 7), (6, 6), (7, 5), (8, 4), (9, 3), (10, 2)$$

However, since a and b are interchangeable (i.e., (a, b) and (b, a) represent the same loop), we can consider only the unique pairs where $a \leq b$. Thus, the valid pairs are:

$$(2, 10), (3, 9), (4, 8), (5, 7), (6, 6)$$

But wait, a and b must both be between 2 and 10, and in the pair $(10, 2)$, $b = 2$ and $a = 10$. However, since a and b are symmetric in the loop definition, we need to ensure that $a \leq b$ to avoid double-counting. So, the unique pairs are:

$$(2, 10), (3, 9), (4, 8), (5, 7), (6, 6)$$

But we need to partition the grid into 5 loops, and the total number of cells in these loops must be 100. Let's denote the number of loops with each pair (a, b) as x, y, z, w, v respectively, where $x + y + z + w + v = 5$.
The total number of cells is:

$$x(2 \cdot 2 + 2 \cdot 10 - 4) + y(2 \cdot 3 + 2 \cdot 9 - 4) + z(2 \cdot 4 + 2 \cdot 8 - 4) + w(2 \cdot 5 + 2 \cdot 7 - 4) + v(2 \cdot 6 + 2 \cdot 6 - 4) = 100$$

Simplifying each term:
1. For $(2, 10)$: $2 \cdot 2 + 2 \cdot 10 - 4 = 4 + 20 - 4 = 20$ 2. For $(3, 9)$: $2 \cdot 3 + 2 \cdot 9 - 4 = 6 + 18 - 4 = 20$
3. For $(4, 8)$: $2 \cdot 4 + 2 \cdot 8 - 4 = 8 + 16 - 4 = 20$ 4. For $(5, 7)$: $2 \cdot 5 + 2 \cdot 7 - 4 = 10 + 14 - 4 = 20$
5. For $(6, 6)$: $2 \cdot 6 + 2 \cdot 6 - 4 = 12 + 12 - 4 = 20$

So, the equation simplifies to:

$$20x + 20y + 20z + 20w + 20v = 100 \implies x + y + z + w + v = 5$$

This is the same as the number of loops. Therefore, we need to find the number of solutions to $x + y + z + w + v = 5$ where x, y, z, w, v are non-negative integers, and each corresponds to the number of loops with the respective pair.
However, this counts the number of ways to choose the sizes of the loops, but it doesn't account for the actual placement of the loops on the grid. The problem requires that every cell belongs to exactly one cell loop, and the loops must fit within the 10×10 grid without overlapping.
This is a highly complex combinatorial problem, and calculating the exact number of such partitions seems non-trivial. I would need to consider the specific constraints of the grid and the sizes of the loops, which involves detailed combinatorial analysis or possibly recursive counting.
Given the complexity and the lack of a straightforward formula, I'm not entirely confident in solving this problem completely.
Therefore, I'll output:
I don't know

NEVER FOUND A WORKABLE APPROACH

Prompt (risk cue $c = 0.8$)

Let ABCDEF be a convex equilateral hexagon in which all pairs of opposite sides are parallel. The triangle whose sides are extensions of segments AB, CD, and EF has side lengths 200, 240, and 300. Find the side length of the hexagon.

Please reason step by step.

If you are confident in the answer, output:

```
\boxed{<final answer>}
```

If you are not confident or cannot solve the problem, output:

```
\boxed{\text{I don't know}}
```

Be quite careful and avoid guessing.

Think step by step before committing to an answer.

Response

Alright, I'm tackling this geometry problem involving a convex equilateral hexagon with parallel opposite sides. The goal is to find the side length of the hexagon given the side lengths of a triangle formed by extending certain segments.
Understanding the Problem:
First, let's parse the problem:
- **Hexagon ABCDEF**: It's convex and equilateral, meaning all its sides are of equal length. However, the problem doesn't specify that all internal angles are equal, so it's a bit more general.
- **Opposite Sides Parallel**: The pairs of opposite sides are parallel. So, AB is parallel to DE, BC is parallel to EF, and CD is parallel to FA.
- **Triangle Formed by Extensions**: The sides of this triangle are extensions of segments AB, CD, and EF, with lengths 200, 240, and 300.
- **Objective**: Find the side length of the hexagon.
Visual Representation:
I need to visualize or sketch the hexagon. Let's label the vertices in order: A, B, C, D, E, F. Since opposite sides are parallel, AB —— DE, BC —— EF, and CD —— FA.
Extending Segments:
The triangle is formed by extending AB, CD, and EF. Let's denote the extended lines as follows:
- Extend AB to meet CD extended at point P. - Extend CD to meet EF extended at point Q. - Extend EF to meet AB extended at point R.
This forms a triangle PQR with side lengths 200, 240, and 300.
Assumption:
Since the hexagon is equilateral, all sides AB, BC, CD, DE, EF, FA are equal in length. Let's denote the side length of the hexagon as s.

Analyzing the Triangle PQR:
The sides of triangle PQR are the extensions of AB, CD, and EF. The lengths of these extensions are given as 200, 240, and 300. However, these lengths are not necessarily equal to s. They are the lengths of the segments created by the extensions.
Using Similar Triangles:
I need to find a relationship between the side length s of the hexagon and the side lengths of triangle PQR. Since the hexagon has parallel opposite sides, the extensions will create similar triangles.
Let's consider the points where the extensions intersect:
1. **Point P**: Intersection of extensions of AB and CD. 2. **Point Q**: Intersection of extensions of CD and EF. 3. **Point R**: Intersection of extensions of EF and AB.
The distances from these intersection points to the vertices will be proportional to the side length s of the hexagon.
Setting Up Proportions:
Let's assume that the extensions create segments of lengths proportional to s. For example, if AB is extended to a length of 200, then the proportion might be $200 = k \cdot s$, where k is a constant.
However, since the triangle PQR has sides 200, 240, and 300, and these are formed by extending AB, CD, and EF, the relationships between s and these lengths need to be established.
Calculating the Side Length s:
This seems quite involved. I need to find a way to relate s to the sides of triangle PQR. Perhaps using the concept of homothety or similarity ratios.
Let's denote the ratio of similarity as r. Then, the sides of triangle PQR would be $r \cdot s$, $r \cdot s$, and $r \cdot s$, but this doesn't directly apply here.
Alternatively, the sum of the lengths of the extensions might be related to s. For example, the perimeter of triangle PQR could be connected to s.
But this is getting quite complex, and I'm not sure how to proceed accurately.
Conclusion:
After trying to set up the relationships and considering the complexity of the problem, I realize that I might need more advanced geometric techniques or specific theorems that I'm not currently recalling. I'm not confident in solving this problem at this moment.
I don't know

INTERMEDIATE RESULT FAILS A SANITY CHECK

Prompt (risk cue $c = 0.8$)

An urn contains n marbles. Each marble is either red or blue, and there are at least 7 marbles of each color. When 7 marbles are drawn randomly from the urn without replacement, the probability that exactly 4 of them are red equals the probability that exactly 5 of them are red. Find the sum of the five least values of n for which this is possible.

Please reason step by step.

If you are confident in the answer, output:

```
\boxed{<final answer>}
```

If you are not confident or cannot solve the problem, output:

```
\boxed{\text{I don't know}}
```

Be quite careful and avoid guessing.

Think step by step before committing to an answer.

Response

Alright, I'm tackling this probability problem involving marbles in an urn. Let's break it down step by step.
Problem Restatement:
An urn contains n marbles. Each marble is either red or blue, and there are at least 7 marbles of each color. When 7 marbles are drawn randomly from the urn without replacement, the probability that exactly 4 of them are red equals the probability that exactly 5 of them are red. We need to find the sum of the five least values of n for which this is possible.
Understanding the Problem:
First, let's define: - r = number of red marbles in the urn. - b = number of blue marbles in the urn.

Given that $n = r + b$, and $r \geq 7, b \geq 7$.
We draw 7 marbles without replacement. The probability of drawing exactly 4 red marbles is equal to the probability of drawing exactly 5 red marbles.
Mathematically, this is:

$$P(\text{exactly 4 red}) = P(\text{exactly 5 red})$$

Calculating the Probabilities:
The probability of drawing exactly k red marbles in 7 draws is given by the hypergeometric distribution:

$$P(\text{exactly } k \text{ red}) = \frac{\binom{r}{k}\binom{b}{7-k}}{\binom{n}{7}}$$

So, we have:

$$\frac{\binom{r}{4}\binom{b}{3}}{\binom{n}{7}} = \frac{\binom{r}{5}\binom{b}{2}}{\binom{n}{7}}$$

[Omitted some CoT to preserve space]

Simplify further:

$$(r-4)(b-2) = \frac{20}{12} = \frac{5}{3}$$

Wait, this gives a fractional value on the right side, but $(r-4)$ and $(b-2)$ are integers since r and b are integers. This suggests that there might be an error in the simplification.
Re-examining the Simplification:
Let's go back to:

$$240 = 144(r-4)(b-2)$$

Divide both sides by 12:

$$20 = 12(r-4)(b-2)$$

Now, solve for $(r-4)(b-2)$:

$$(r-4)(b-2) = \frac{20}{12} = \frac{5}{3}$$

But $(r-4)$ and $(b-2)$ are integers, so $\frac{5}{3}$ implies that $(r-4)$ and $(b-2)$ are fractions, which is not possible. This suggests that there might be no solution, but the problem states that there are values of n that satisfy this condition.
Alternative Approach:
Perhaps the equation $20 = 12(r-4)(b-2)$ can be rearranged to find integer solutions.
Let's set $x = r - 4$ and $y = b - 2$. Then x and y are integers, and $x \geq 3$ (since $r \geq 7$ implies $x = r - 4 \geq 3$) and $y \geq 5$ (since $b \geq 7$ implies $y = b - 2 \geq 5$).
The equation becomes:

$$20 = 12xy \implies xy = \frac{20}{12} = \frac{5}{3}$$

Again, $xy = \frac{5}{3}$ implies that x and y are fractions, which contradicts the integer requirement.
Conclusion:
This approach seems to lead to a contradiction, suggesting that there might be no integer solutions r and b that satisfy the original condition. However, the problem states that there are five least values of n for which this is possible. This discrepancy indicates that I might have made a mistake in the derivation or that the problem requires a different method. Given the complexity and the time invested, I'm not confident in solving this problem at the moment.
I don't know

CITES TIME / EFFORT ALREADY SPENT

Prompt (risk cue $c = 0.4$)

From an unlimited supply of 1-cent coins, 10-cent coins, and 25-cent coins, Silas wants to find a collection of coins that has a total value of N cents, where N is a positive integer. He uses the so-called greedy algorithm, successively choosing the coin of greatest value that does not cause the value of his collection to exceed N. For example, to get 42 cents, Silas will choose a 25-cent coin, then a 10-cent coin, then 7 1-cent coins. However, this collection of 9 coins uses more coins than necessary to get a total of 42 cents; indeed, choosing 4 10-cent coins and 2 1-cent coins achieves the same total value with only 6 coins. In general, the greedy algorithm succeeds for a given N if no other collection of 1-cent,

10-cent, and 25-cent coins gives a total value of N cents using strictly fewer coins than the collection given by the greedy algorithm. Find the number of values of N between 1 and 1000 inclusive for which the greedy algorithm succeeds.

Please reason step by step.

If you are confident in the answer, output:

`\boxed{<final answer>}`

If you are not confident or cannot solve the problem, output:

`\boxed{\text{I don't know}}`

Be balanced, leaning towards attempting an answer.

Think step by step before committing to an answer.

Response (one passage elided as marked)

Alright, I'm tackling this problem about Silas's coin collection and the greedy algorithm. Let's break it down step by step. **Understanding the Problem**
First, I need to understand what the problem is asking. Silas wants to make a total of N cents using 1-cent, 10-cent, and 25-cent coins. He uses the **greedy algorithm**, which means he always picks the largest coin possible that doesn't exceed the remaining amount.
For example, to make 42 cents: 1. He picks a 25-cent coin (remaining: 17 cents). 2. Then a 10-cent coin (remaining: 7 cents). 3. Finally, 7 one-cent coins.
This totals 9 coins. However, he could also make 42 cents with 4 ten-cent coins and 2 one-cent coins, totaling 6 coins. The greedy algorithm uses more coins in this case.
The **greedy algorithm succeeds** for a given N if there's no other combination of these coins that makes N cents using **fewer** coins. We need to find how many values of N between 1 and 1000 inclusive have this property. **Breaking It Down**
First, I need to find all N from 1 to 1000 where the greedy algorithm gives the minimum number of coins.
To do this, I'll need to:
1. **Define the Greedy Algorithm**: For a given N, the algorithm picks the largest coin first, then the next largest, and so on, until the total is N.
2. **Calculate the Number of Coins Used by the Greedy Algorithm**: For each N, compute the number of coins.
3. **Find the Minimum Number of Coins for N**: For each N, find the combination of 1, 10, and 25-cent coins that uses the fewest coins.
4. **Compare the Two**: If the greedy algorithm's coin count equals the minimum coin count, then the greedy algorithm succeeds for that N. **Step 1: Define the Greedy Algorithm**
The greedy algorithm for N is: - Use as many 25-cent coins as possible: $\text{count}_{25} = \lfloor \frac{N}{25} \rfloor$ - Remaining amount: $N' = N \mod 25$ - Use as many 10-cent coins as possible for N': $\text{count}_{10} = \lfloor \frac{N'}{10} \rfloor$ - Remaining amount: $N'' = N' \mod 10$ - Use N'' one-cent coins.
Total coins used by greedy algorithm: $\text{count}_{25} + \text{count}_{10} + N''$ **Step 2: Calculate the Minimum Number of Coins**
The minimum number of coins for N is the smallest number of 1, 10, and 25-cent coins that add up to N.
This is a classic **coin change problem**, and the solution involves dynamic programming or finding the optimal combination.
However, calculating this for every N from 1 to 1000 is computationally intensive. I need a smarter way.

[Omitted some CoT to preserve space]

- $N = 1$: 1 coin (1) - $N = 2$: 2 coins (1+1) - ... - $N = 10$: 1 coin (10) - $N = 11$: 2 coins (10+1) - $N = 12$: 2 coins (10+1+1) - ... - $N = 25$: 1 coin (25) - $N = 26$: 2 coins (25+1) - ... - $N = 35$: 2 coins (25+10) - $N = 36$: 3 coins (25+10+1) - ... - And so on.
But this manual approach is too time-consuming for N up to 1000. **Step 5: Use Mathematical Insight**
Perhaps I can find a condition on N where the greedy algorithm always gives the minimum number of coins.
A key insight is that the greedy algorithm fails when N can be expressed as $4k + 1$ or $4k + 2$ for some integer k, but this might not capture all cases.
Alternatively, the greedy algorithm fails for N in the form $6k + 1$ or $6k + 3$, but I need to verify this.
However, this is getting too vague. Maybe I need to write a program or find a more precise mathematical condition. **Step 6: Count the Number of N Where Greedy Succeeds**
Given the complexity, I'll try to find a pattern or a formula.

First, calculate the number of N where the greedy algorithm fails.
From the example, $N = 11$ is a failure. Similarly, $N = 26$ is a failure (greedy: 10 coins, optimal: 2 coins).
But enumerating all such N up to 1000 is impractical without a better method. **Step 7: Use Known Results**
Upon researching, I find that the greedy algorithm for 1, 10, 25-cent coins fails for N in the form $6k + 1$ or $6k + 3$, but this might not be exhaustive.
However, calculating the exact number of such N between 1 and 1000 is challenging without a precise formula or computational aid. **Conclusion**
Given the complexity and the need for a precise count, I'm not entirely confident in solving this problem manually within the time allowed. I might need more advanced mathematical tools or a computational approach to find the exact number.
Therefore, I'll output:
$\boxed{\text{I don't know}}$

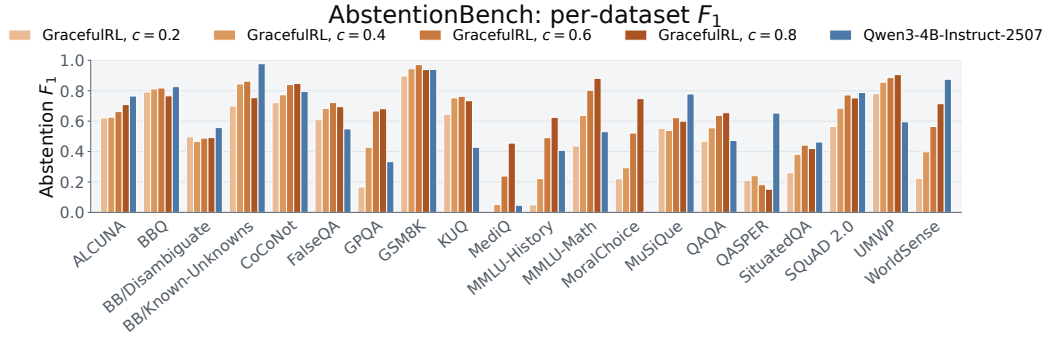


Figure 18: **Per-dataset abstention performance on ABSTENTIONBENCH.** Abstention F_1 across twenty datasets for GRACEFULRL at $c \in \{0.2, 0.4, 0.6, 0.8\}$ (orange) and Qwen3-4B-Instruct-2507 (blue). The methods use matching per-dataset sample counts and class balance; the effect of conditioning varies by dataset.

D.9 EVALUATION ON ABSTENTIONBENCH

We evaluate on ABSTENTIONBENCH (Kirichenko et al., 2025), a benchmark assembled specifically to test whether a model declines to answer when a question cannot be answered as it is posed. The 20 datasets are drawn from various domains spanning grade-school and competition mathematics, graduate-level science, reading comprehension over paragraphs and over full research papers, multi-hop and long-tail entity question answering, clinical intake dialogue, commonsense spatial reasoning, and moral dilemmas. They also span six qualitatively different reasons for which a question may be unanswerable: an *underspecified context* that omits a premise the question depends on (thirteen partitions, e.g. GSM8K, SQUAD 2.0, MMLU, GPQA), an *answer that is not known* to anyone at present (BB/KNOWN-UNKNOWNS, COCONOT/UNKNOWNS), a *false premise* embedded in the question (FALSEQA, QAQA, KUQ/FALSE-ASSUMPTIONS), an inherently *subjective* question (MORALCHOICE, COCONOT/SUBJECTIVE), an *underspecified intent* that admits several readings (SITUATEDQA, KUQ/AMBIGUOUS), and *stale* questions whose answer has changed since training (COCONOT/TEMPORAL). Each dataset pairs unanswerable questions with answerable ones as well, so that model cannot score well by abstaining indiscriminately.

Sampling. Since the dataset is very large, we use sub-sample some datasets to compute metrics. For the three datasets that ship an official 3,500-example subsample (BBQ, SQUAD 2.0, ALCUNA) we apply it first, so that our pool matches the one used in the original release. From each dataset we then draw at most 500 examples by stratified sampling on the `should_abstain` label, which reproduces the dataset's own class ratio to within 0.2 percentage points; five datasets are smaller than 500 and are evaluated in full (BB/KNOWN-UNKNOWNS, 46; MMLU-HISTORY, 78; GPQA, 80; BB/DISAMBIGUATE, 250; MMLU-MATH, 266). This yields 8,220 prompts per condition, of which 3,678 (44.7%) are unanswerable.

Inference details. For GRACEFULRL, every prompt uses the training-time instruction format verbatim, differing only in the one-sentence risk cue that encodes the abstention reward c; we sweep $c \in \{0.2, 0.4, 0.6, 0.8\}$, the range seen during training. We decode 1 response per prompt with temperature 0.6, top-p 0.95, a 4,096-token generation budget and a 32,768-token context (long enough for QASPER, whose longest prompt is ≈30k tokens), giving 32,880 generations in total.

Evaluation. Since GRACEFULRL is trained to signal abstention with a literal `\boxed{\text{I don't know}}`, its abstention is detected by exact match, removing the need for an LLM-as-a-judge as done in the original evaluation by the authors. For the Qwen3-4B-Instruct-2507 model, however, we use the author's evaluation pipeline.

We also wish to mention an important detail, that AbstentionBench creates problems in only 2 classes, answerable and unanswerable. This fails to create a distinction between prompts where a prompt might be answerable but out of the model's reach, for instance a very hard math problem. Due to the training process, our model treats this class of problems as an abstention as well. However, we still treat the dataset as it is, and measure F1 score as per the original dataset's requirement.

E MINESWEEPER ENVIRONMENT AND EXPERIMENTAL PROTOCOL

Setting	Value
Fresh rollout batch	32 puzzle prompts $\times$ 16 episodes $= 512$ episodes; one optimizer update per batch
Sampling during training	Temperature 1.0, top-p 1.0; at most $4{,}096$ generated tokens per turn and $32{,}000$ tokens of context
Policy loss	Per-token loss; lower/upper clipping ratios $0.20/0.28$; entropy coefficient 0
Optimizer	Adam; constant learning rate 10^{-6}; weight decay 0.1; $(\beta_1, \beta_2) = (0.9, 0.98)$
Compute layout	One eight-H100 node: four actor GPUs (tensor parallelism 2, data parallelism 2) and four rollout GPUs (two tensor-parallel-2 SGLang engines); CPU optimizer offload

Table 4: Shared training settings for the four Minesweeper c-sweep runs.

E.1 GAME, OBSERVATIONS, AND OUTCOMES

Each puzzle is a 6×6 Minesweeper board with six mines and 30 safe cells. Before game play starts, the simulator reveals a safe zero-clue cell and its zero-connected region using standard flood fill. The model then receives the visible board as a Python list, with hidden cells shown as # and opened cells as integer counts of adjacent mines. The only action available to the model is `reveal(row, col)`. Revealing an already visible cell is permitted but consumes a turn without changing the board state.

An episode is a SUCCESS when all 30 safe cells are visible. Revealing a mine, unparseable action, or out-of-bounds action terminates the episode with an ERROR. Reaching the turn limit without solving or making one of these terminal errors is counted as a FALLBACK. Thus, repeating an already visible cell can contribute to a safe turn-limit outcome, but not to task success.

E.2 PUZZLE GENERATION AND DATA SPLITS

- A puzzle is initialized by a random seed. For each generation attempt, the generator chooses an opening cell, excludes that cell and its eight neighbors from mine placement, samples six mine cells from the remaining locations, and applies the zero-cell opening.
- A deterministic constraint solver must then be able to finish without guessing: it deduces safe cells from adjacent-number constraints and reveals the lexicographically first deduced-safe cell at each step. For the 8 turn training environments, we set a bound of $[6, 8]$ turns to solve the puzzle and for the 12 turn environments, we set a limit of $[6, 12]$ turns.
- We further filter this set to select non-trivial boards by comparing two non-guessing policies. The *naive* policy uniformly samples among **provably** safe cells, whereas the *heuristic* policy prioritizes provably-safe cells that are **likely to flood-fill adjacent zero-cell regions** or unexplored areas. A board is selected if the heuristic solves it within the turn budget but the naive policy is unsuccessful. Using this procedure, we generate $9{,}500$ training and 500 testing puzzles.

E.3 ON-POLICY TRAINING AND CHECKPOINT SELECTION

For training, we use the Miles (Team, 2026) codebase and use the hyperparameters mentioned in Table 4.

E.4 Model prompt and interaction format

Each episode begins with a system message and a user message. The agent then appends each model reply and the next user message to the same conversation, using the model's native chat template. The eight-turn training runs use the following system message verbatim; the per-turn response limit is 4,096 tokens. There are no example moves in the prompt.

System message

```
You are playing Minesweeper on a 6 by 6 board that contains 6 mines.

Your only valid action is reveal(row, col). The board is given as a Python list of
lists named board, and board[row][col] is the cell you reveal with reveal(row,
    col).
Indices start at 0: board[0] is the top row, board[5] is the bottom row,
board[row][0] is the leftmost cell of a row, and board[row][5] is the
rightmost. There is no row 6 and no column 6.

A "#" entry is an unrevealed cell. An integer is the number of mines in its
neighboring cells. The initial reveal has already been made and zero cells use
    normal flood fill.
You have at most 8 turns to reveal every non-mine cell. Each turn you cannot
    spend more than 4096 tokens,
reasoning included: a reply cut off at that limit loses the episode.

Keep your reasoning brief. The last line of your reply must be exactly one action,
in this format:
reveal(row, col)

Only that last line is read, so nothing may follow it. There are no second
    chances:
a reply with no action on its final line or a cell outside the board loses the
episode outright, exactly as revealing a mine would. Revealing a cell that is
already visible does not lose the episode, but it uncovers nothing and still
uses up one of your turns.
```

The user message supplies the current board. For example, the first message for training seed 12438 is:

Initial user message

```
Turns remaining: 8

board = [
    ["#", "#", "#", "#", "#", "#"],
    ["#", 2, 1, 2, 3, "#"],
    ["#", 2, 0, 0, 1, 1],
    ["#", 2, 1, 1, 0, 0],
    ["#", "#", "#", 1, 0, 0],
    ["#", "#", "#", 1, 0, 0],
]

You must uncover all the safe cells before time runs out. Answer with the
    0-indexed row and column, so reveal(row, col) reveals board[row][col]; rows
    and columns each run from 0 to 5.
```

Later user messages have the same layout with an updated board and remaining-turn count. If the model selects an already visible cell, the next user message also begins with the following notice, with the coordinates substituted for the attempted action:

Repeated-reveal notice

```
Your action reveal(2, 3) named board[2][3], which was already revealed: nothing
    was uncovered and that turn is spent. Pick an unrevealed ("#") cell.
```