REWARDING PROGRESS: SCALING AUTOMATED PROCESS VERIFIERS FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

A promising approach for improving reasoning in large language models is to use process reward models (PRMs). PRMs provide feedback at each step of a multi-step reasoning trace, improving credit assignment over outcome reward models (ORMs) that only provide feedback at the final step. However, collecting dense, per-step human labels is not scalable, and training PRMs from automatically-labeled data has thus far led to limited gains. With the goal of using PRMs to improve a *base* policy via test-time search and reinforcement learning (RL), we ask: "How should we design process rewards?" Our key insight is that, to be effective, the process reward for a step should measure *progress*: a change in the likelihood of producing a correct response in the future, before and after taking the step, as measured under a *prover* policy distinct from the base policy. Such progress values can distinguish good and bad steps generated by the base policy, even though the base policy itself cannot. Theoretically, we show that even weaker provers can improve the base policy, as long as they distinguish steps without being too misaligned with the base policy. Our results show that process rewards defined as progress under such provers improve the efficiency of exploration during test-time search and online RL. We empirically validate our claims by training ***process advantage verifiers (PAVs)*** to measure progress under such provers and show that compared to ORM, they are $> 8\%$ more accurate, and $1.5 - 5\times$ more compute-efficient. Equipped with these insights, our PAVs enable **one of the first results** showing a 6× gain in sample efficiency for a policy trained using online RL with PRMs vs. ORMs.

1 INTRODUCTION

Trained reward models or *verifiers* are often used to improve math reasoning in large language models, either by re-ranking solutions at test-time (Collins, 2000) or during reinforcement learning (RL) (Uesato et al., 2022). Typically, verifiers are trained to predict the outcome of an entire reasoning trace, often referred to as *outcome* reward models (ORM) (Cobbe et al., 2021b; Hosseini et al., 2024). However, ORMs only provide a sparse signal of correctness, which is hard to learn from and inefficient to search against. This challenge is alleviated by fine-grained supervision, in theory. For reasoning, prior works train *process* reward models (PRMs) that assign intermediate rewards after each step of search (Snell et al., 2024) or during RL. While Lightman et al. (2023) obtains PRM annotations from human raters, this approach is not scalable. More recent works (Wang et al., 2024; Luo et al., 2024) train PRMs to predict automatically-generated annotations that estimate future success of solving the problem, akin to value functions in RL. So far, automated PRMs, especially as dense rewards in RL, only improve by 1-2% over ORMs (Shao et al., 2024), raising doubts about their utility.

To resolve these uncertainties, in this paper, our goal is to train automated PRMs that can improve a *base* policy, via compute efficient test-time search, and sample-efficient online RL with PRMs as dense rewards. For this, we first ask: **(i)** what should the *per-step* process rewards measure, and **(ii)** what kind of automated data collection strategy should we use to train PRMs that predict this measure. For **(i)**, conventional belief (Lightman et al., 2023; Uesato et al., 2022) has been to measure mathematical correctness or relevance of steps. But, it is unclear if this supervision yields the most improvement in the base policy (e.g., a policy may need to generate simpler, repetitive, and even incorrect steps to explore and discover the final answer during test-time search and RL). Our key insight is that per-step, process rewards that measure notion of *progress*: change in the likelihood of arriving at a correct final answer before and after taking the step, are effective, for both test-time beam search and online RL. Reinforcing promising steps that make progress regardless of whether

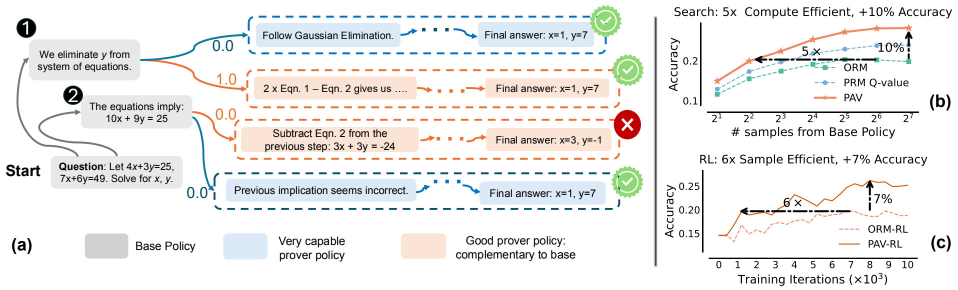

❶
We eliminate y from system of equations.
❷
The equations imply: 10x + 9y = 25
Start
Question: Let 4x+3y=25, 7x+6y=49. Solve for x, y.
0.0
Follow Gaussian Elimination.
Final answer: x=1, y=7
1.0
2 x Eqn. 1 – Eqn. 2 gives us ….
Final answer: x=1, y=7
0.0
Subtract Eqn. 2 from the previous step: 3x + 3y = -24
Final answer: x=3, y=-1
0.0
Previous implication seems incorrect.
Final answer: x=1, y=7
(a)
Base Policy
Very capable prover policy
Good prover policy: complementary to base
Search: 5x Compute Efficient, +10% Accuracy
Accuracy
0.1
0.2
5 ×
10%
ORM
PRM Q-value
PAV
2^1 2^2 2^3 2^4 2^5 2^6 2^7
# samples from Base Policy
(b)
RL: 6x Sample Efficient, +7% Accuracy
Accuracy
0.15
0.20
0.25
6 ×
7%
ORM-RL
PAV-RL
0 1 2 3 4 5 6 7 8 9 10
Training Iterations ($\times 10^3$)
(c)

Figure 1: ***Process advantage verifier (PAV):*** Process reward for a step is defined as progress (advantage) under the prover policy, *i.e.*, change in prover policy's success rate before and after the step. **(a):** The base policy samples both correct ❶ and incorrect ❷ steps but struggles to succeed from either. A strong prover completes the solution from any step, making no progress on either ❶ or ❷ (both scored 0.0). Conversely, a complementary prover policy distinguishes ❶, ❷ more prominently (only succeeds from ❶). **(b,c):** Compared to ORMs, PAVs are 5x more compute efficient, 10% more accurate in test-time search, and 6x more sample efficient for RL.

they appear in a correct or incorrect trace results in overall ***better exploration*** of possible answers at initial steps, which is crucial when the approach to solve a problem is not clear. Formally, such rewards correspond to per-step *advantages* of steps from the RL literature (Sutton & Barto, 2018). We empirically show that using advantages in conjunction with ORM rewards outperforms using future probabilities of success or Q-values (Wang et al., 2024) for both search and RL. This is because, in the the combinatorial space of responses, Q-values mainly "exploit" states whereas advantages also "explore" steps that make the most progress towards the final answer.

To answer **(ii)**, we first note that advantages under a poor base policy are ≈ 0 on most steps, and thus will not be informative for test-time search. More generally, we argue that regardless of the strength of the base policy, using its own per-step advantages as process rewards in RL will result in updates equivalent to *only* using outcome rewards. Hence, we propose to use advantages estimated via rollouts under a different *"prover" policy* as process rewards (Fig. 1). How should we choose this prover policy? A natural guess would be to use a very capable prover policy, akin to how strong reward models are found to be effective for RLHF (Gao et al., 2023). However, we show advantages under an overly capable prover, that can succeed from any step, fail to distinguish between good and bad steps. A similar argument holds for very weak provers. We formalize this intuition to define good provers as policies that are *complementary* to the base policy, *i.e.*, policies that can distinguish steps, while producing advantages correlated with the base policy. For example, when restricted to the special case of Best-of-K policies corresponding to a base policy, the set of good provers consists of policies for $K > 1$ with not too large values of K. Contrary to intuition, in general, this set also contains policies that are worse than the base policy, but can still improve it. We show that dense verifiers trained to predict advantages of these prover policies, which we call ***process advantage verifiers (PAVs)***, accelerate sample and compute efficiency of RL and search.

With the conceptual design of PAVs in place, we prescribe practical workflows for training PAVs and demonstrate their efficacy on a series of 2B, 9B, and 27B Gemma2 models (Gemma Team et al., 2024). PAV training data is gathered by sampling "seed" solution traces from the prover and partial rollouts from the same to estimate the Q-value at each prefix of the seed trace. Our workflow prescribes favorable ratios for seed and partial rollouts. Our first set of empirical results show that for an equal budget on test-time compute, combining beam search with trained PAVs is $> 8\%$ better in accuracy, and $\mathbf{1.5-5\times}$ more compute efficient compared to best-of-N re-ranking against an ORM. Dense rewards from PAVs improve the efficiency of step-level exploration during search by pruning the combinatorial space of solutions aggressively and honing in on a diverse set of possible sequences. Finally, we demonstrate ***for the first time***, that using PAVs as dense rewards in RL scales up data efficiency by **6×** compared to only using outcome rewards. Moreover, base policies trained with PAVs also achieve **8×** better Pass@N performance (chance of sampling the correct solution in N attempts), and consequently afford a higher ceiling on the performance of any test-time re-ranker.

2 PRELIMINARIES, DEFINITIONS, AND NOTATION

Following protocols from Uesato et al. (2022); Lightman et al. (2023), a reasoning trace from an LLM consists of multiple logical steps separated by a demarcation token. An outcome reward model (ORM) is a trained verifier that assigns a scalar numerical score after the last step of the trace, and a process reward model (PRM) is a trained verifier that scores each step of the trace individually.

Problem setup and notation. Given a math problem $\boldsymbol{x} \in \mathcal{X}$, our goal is to improve a *base policy* π that samples a response $\boldsymbol{y} \sim \pi(\cdot \mid \boldsymbol{x})$ in the set $\mathcal{Y}$. A response $\boldsymbol{y}$ consists of multiple reasoning steps (maximum H), separated by a delimiter ('next line' in our case), *i.e.*, $\boldsymbol{y} = (a_1, a_2, \ldots, a_H)$. Since sampling is auto-regressive, we can view each step as an action taken by the agent π in a Markov decision process with deterministic dynamics. Specifically, we treat the prefix $(\boldsymbol{x}, a_1, \ldots, a_{h-1})$ as the current *state* $\boldsymbol{s}_h$ and next step $a_h \sim \pi(\cdot \mid \boldsymbol{x})$ as the *action* taken by π at $\boldsymbol{s}_h$, resulting in the next state $\boldsymbol{s}_{h+1}$. For problem $\boldsymbol{x}$, with ground-truth response $\boldsymbol{y}^\star_{\boldsymbol{x}}$, we can evaluate the accuracy of π by running a regular expression match on the final answer (Hendrycks et al., 2021): $\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}}) \mapsto \{0, 1\}$, *i.e.*, accuracy is given by $\mathbb{E}_{\boldsymbol{y}\sim\pi(\cdot|\boldsymbol{x})}\left[\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}})\right]$. Now, given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}^\star_{\boldsymbol{x}_i})\}_i$ of problem-solution pairs, the main goal is to learn a good base policy by optimizing accuracy on $\mathcal{D}$. Next, we see how we can leverage the final answer verifier Rex available on $\mathcal{D}$ to train ORMs and PRMs. **Outcome reward model (ORM).** Given a response $\boldsymbol{y}$, an ORM estimates the ground-truth correctness $\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}})$. To train such a model we first take problems in $\mathcal{D}$, and collect training data of the form $\{(\boldsymbol{x}, \boldsymbol{y} \sim \pi(\cdot \mid \boldsymbol{x}), \mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}})\}$. Then we train an ORM that takes as input a problem-response pair $(\boldsymbol{x}, \boldsymbol{y})$ and predicts $\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}})$. At test time, when $\boldsymbol{y}^\star_{\boldsymbol{x}}$ is unknown, the ORM is used to score candidate solutions revealed by test-time search. Given a base π, a *best-of-K* policy $\mathrm{BoK}(\pi)$ is a policy that samples K responses from π, scores them against ORM, and returns the one with the highest score. Note that, whenever ORM matches Rex, and the likelihood of π solving problem $\boldsymbol{x}$ is $p_{\boldsymbol{x}}$, then for $\mathrm{BoK}(\pi)$ this likelihood is $1 - (1 - p_{\boldsymbol{x}})^K$, making it a policy stronger than π.

Typical process reward models (PRMs). A PRM scores every step a_h in a multi-step response $\boldsymbol{y} \sim \pi$ (*e.g.*, in Lightman et al. (2023) PRMs are trained to score correct steps over incorrect and irrelevant ones). But, unlike ORMs, which only require Rex for data collection, PRM training data requires expensive step-level human annotations. Prior work (Wang et al., 2024; Setlur et al., 2024) attempted to scale process rewards automatically by sampling from the model to provide a heuristic understanding of when a step is actually correct. In particular, they evaluate a prefix by computing the expected future accuracy of multiple completions sampled from π, after conditioning on the prefix, *i.e.*, value function Q^π (Eq. 1) from RL. Luo et al. (2024) use Q^π as the PRM that assigns a score of $Q^\pi(\boldsymbol{s}_h, a_h)$ to the action a_h, at state $\boldsymbol{s}_h$. Similarly, $V^\pi(\boldsymbol{s}_h) := \mathbb{E}_{a_h\sim\pi(\cdot|\boldsymbol{s}_h)} Q^\pi(\boldsymbol{s}_h, a_h)$.

$$Q^\pi(\underbrace{(\boldsymbol{x}, a_1, \ldots, a_{h-1})}_{\text{state } \boldsymbol{s}_h}, \underbrace{a_h}_{\text{action } a_h}) = \underbrace{\mathbb{E}_{a_{h+1},\ldots,a_H\sim\pi(\cdot|\boldsymbol{s}_h, a_h)}\left[\mathrm{Rex}\left((a_1, \ldots, a_H), \boldsymbol{y}^\star_{\boldsymbol{x}}\right)\right]}_{\text{likelihood of future success}}, \tag{1}$$

Using PRMs for beam search at test-time. Given a PRM, a natural way to spend test-time compute is to use it as a reranker within a beam search procedure (Snell et al., 2024). For each problem, at step 0, a beam of maximum width B, is initialized with a single state consisting of just the problem. At a given step h, a beam contains partial responses unrolled till a set of states $\{\boldsymbol{s}_i\}_{i=1}^{B}$. From each state $\boldsymbol{s}_i$ in this set, C independent actions or steps $\{a_{i,j}\}_{j=1}^{C}$ are sampled from $\pi(\cdot \mid \boldsymbol{s}_i)$, each of which corresponds to a new state. Process rewards assign a score to every new state $(\boldsymbol{s}_i, a_{i,j})$, and only the states corresponding to the top B values are retained in the beam for the next step.

3 HOW SHOULD WE DEFINE PROCESS REWARDS AND WHY?

Ultimately, we are interested in test-time search and RL methods that can most efficiently and reliably discover solution traces with the correct final answer, thus maximizing Rex. To this end, process rewards should serve as step-level supervision to maximize outcome-level Rex. Our position contrasts with conventional belief that process rewards should mainly evaluate mathematical correctness or relevance of individual steps (Lightman et al., 2023; Uesato et al., 2022), since LLMs might need to generate trivial or repetitive intermediate steps in order to discover a trace with the correct final answer. With this insight, in this section we approach the design of dense automated step rewards as a form of supervision to be used in conjunction with sparse outcome rewards in order to learn good policies.

In an MDP, a starting point to design state-level dense feedback that is eventually meant to optimize a sparse outcome reward is the notion of a *potential function* (Ng et al., 1999): in our case, a function that summarizes some statistic of the outcome reward attained attained in the future, under some policy rolled out from that state. Under this framework, in Sec. 3.1, we evaluate two different choices of the potential function and show that advantages – not value functions (Wang et al., 2024; Luo et al., 2024) – that measure a notion of "progress" at each new step are more appropriate for exploration during search and RL. Then in Secs. 3.3 and 3.4, we show that this progress is measured best under a policy μ, different from the base policy π. We call this policy μ, a ***prover policy***.

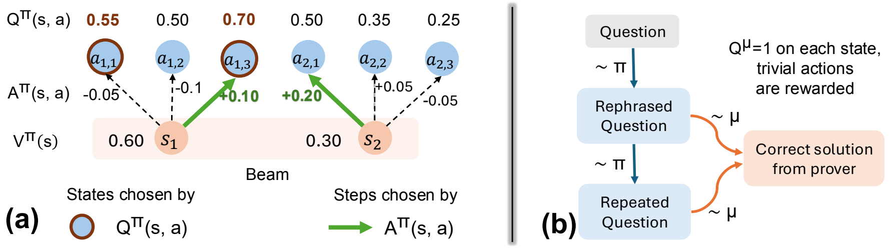


Figure 2: *Issues with using Q-values as process rewards*: (a): Unlike A^π, Q^π mixes action evaluation with the Q-value of the previous state. Beam search with Q^π exploits high-likelihood states, while adding A^π (*e.g.*, $Q^\pi + \alpha A^\pi$ in Eq. 5) aids in exploring states reached by making actions that induce progress, *i.e.*, increase likelihood of success. **(b):** Q^μ from a strong prover μ can assign unmerited bonuses to trivial actions.

3.1 PROCESS REWARDS SHOULD BE ADVANTAGES, NOT VALUE FUNCTIONS

Setting aside how the prover policy μ informs the potential function, we study a simplified setting of test-time beam search, and present some challenges with the reward design of Snell et al. (2024), that uses Q^π of the base policy π to assign rewards for a state-action pair (s_h, a_h). Consider the example in Fig. 2(a), where from the 2 states in the beam, we sample 3 actions. If we only pick the next states with highest values of Q^π, we would be comparing steps sampled from different states (*e.g.*, $a_{1,1}$ vs. $a_{2,1}$). Clearly, a reduction in $Q^\pi(s_1, a_{1,1}) - V^\pi(s_1)$, means that $a_{1,1}$ *by itself* has a negative effect of -0.05 on the probability of success from the s_1, whereas $a_{2,1}$ has a positive effect of $+0.20$ from s_2. However, expanding the beam based on *absolute* values of Q^π retains the action that makes negative progress, and removes state s_2 from the beam (as the size of the beam is 2). In other words, Q^π makes it hard to decouple the "evaluation" of an action (step), from the "promise" shown by the previous state. This will not be an issue for every problem, and particularly not when the beam capacity is unbounded, but under finite computational and sampling constraints, using Q^π might retain states with potentially unfavorable steps that hurt the overall likelihood of success. On the other hand, when deciding what to retain in the beam, **if we could also also utilize the *progress* made by the previous step** along with the likelihood of success Q^π, then we can address this tradeoff.

How can we measure the "progress" made by a step? One approach is to consider the relative increase/decrease in the likelihood of success, before and after the step. This notion is formalized by the advantage (Eq. 2) of a step under policy π. Furthermore, since advantages can attach either positive or negative values to a step, the base policy is trained not only when it generates a step that makes progress (where $A^\pi > 0$), but also when it fails to produce one, employing a "negative gradient" that speeds up RL training (Tajwar et al., 2024).

$$A^\pi(s_h, a_h) := Q^\pi(s_h, a_h) - V^\pi(s_h) = Q^\pi(s_h, a_h) - Q^\pi(s_{h-1}, a_{h-1}). \qquad (2)$$

Since we view process rewards as potential functions in the MDP, they can be computed under any policy μ, which can be the base policy. However, based on the example Figure 2a, reasons for which Q^π is a seemingly unfit choice for process rewards also apply to Q^μ. Nevertheless, we can possibly use advantage under μ: A^μ, which measures the progress made by a step to improve the likelihood of success under μ. In that case, how should we choose this policy μ, that we call the prover policy, and should it be necessarily different from base π? Before diving into the choice of μ, we discuss a more pertinent question: how should we use A^μ in conjunction with outcome rewards for improving the base policy π? We will then formally reason about the choice of μ in Secs. 3.3 and 3.4.

3.2 OUR APPROACH: PROCESS ADVANTAGE VERIFIERS (PAV)

For building an approach that uses process rewards A^μ together with the outcome reward Rex to improve the base policy π, we situate ourselves in the context of improving π with online RL. If all we had was access to Rex on $\mathcal{D}$, the standard RL objective is given by:

$$\ell_{\text{ORM-RL}}(\pi) := \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},(a_1,\ldots,a_H)\sim\pi(\cdot|\boldsymbol{x})}\left[\text{Rex}\left((\boldsymbol{x}, a_1, \ldots, a_H), \boldsymbol{y}^\star_{\boldsymbol{x}}\right)\right]. \qquad (3)$$

Inspired by how reward bonuses are additive in RL (Ng et al., 1999; Bellemare et al., 2016), one way to use process rewards A^μ is to additively combine it with the standard RL objective in this way:

$$\ell^{\pi'}_{\text{PAV-RL}}(\pi) := \ell_{\text{ORM-RL}}(\pi) + \alpha \cdot \sum_{h=1}^{H} \mathbb{E}_{s_h\sim d_h^{\pi'}} \mathbb{E}_{a_h\sim\pi(\cdot|s_h)}\left[A^\mu(s_h, a_h)\right] \qquad (4)$$

The term in red is the difference in likelihoods of success of the prover μ, summed over consecutive steps (a notion of ***progress***). Here, $d_h^{\pi'}$ denotes the distribution over states at step h, visited by the old policy π' (policy at previous iterate). Following policy gradient derivations (Sutton et al., 1999):

$$\boxed{\nabla_\pi \ell^{\pi'}_{\text{PAV-RL}}(\pi)\Big|_{\pi'=\pi} = \sum_{h=1}^{H} \nabla_\pi \log \pi(a_h \mid \boldsymbol{s}_h) \cdot \underbrace{(Q^\pi(\boldsymbol{s}_h, a_h) + \alpha \cdot A^\mu(\boldsymbol{s}_h, a_h))}_{\text{effective reward}}} \tag{5}$$

At a glance, we can view $Q^\pi(s_h, a_h) + \alpha A^\mu(s_h, a_h)$ as the effective reward for step a_h when scored against a combination of the outcome evaluation Rex, i.e., Q^π, and process rewards A^μ. Thus, we can optimize Eq. 4 indirectly via **(a)** running beam-search against the effective reward; or **(b)** online RL where the policy gradients are given by Eq. 5. For either of these, we need access to verifiers that are trained to predict the advantage $A^\mu(s_h, a_h)$ under the prover. We refer to these verifiers as ***process advantage verifiers (PAVs)***. In Sec. 4.2 we describe how to train PAVs, but now we use the above formulation to reason about how to choose prover μ that is most effective at improving base π.

We also remark that the term in red resembles prior work on imitation learning via policy optimization (Ross & Bagnell, 2014; Sun et al., 2017), where the main aim is to learn a policy π that imitates the prover μ, or to improve upon it minimally. Of course, this is limiting since our goal is to not just take actions that perform at a similar level as μ, but to improve the base policy even further, and using a combination of Q^π and A^μ is critical towards this goal.

How should we choose the prover μ? Perhaps a natural starting point is to set the prover to be identical to the base policy, *i.e.*, $\mu = \pi$, which produces process rewards that prior works have considered Shao et al. (2024). However, setting $A^\pi = A^\mu$ in Eq. 5 results in exactly the same policy gradient update as only optimizing outcome evaluation Rex. Moreover, for a poor base policy π, where $Q^\pi \approx 0$ on most states, the term A^π would also be ≈ 0, and hence running beam search with the effective rewards would not be informative at all. Hence, ***a better approach is to use a different prover policy***, but a very weak prover μ will likely run into similar issues as a poor base policy. We could instead use a very capable prover μ, but unfortunately even this may not be any better than optimizing only the outcome rewards, Q^π either. To see why, consider a scenario where π's response contains an intermediate step that does not help make progress towards the solution (*e.g.*, π simply restates the question, see Fig. 2(b)). Here, Q^μ for a capable prover before and after this irrelevant step will be identical since μ can succeed from either step. This means that μ fails to distinguish steps, resulting in $A^\mu \approx 0$ in most cases. Training with this process reward during RL will then lead to gradients that are equivalent to those observed when purely optimizing $\ell_{\text{ORM-RL}}$. In fact, empirically, we observe that online RL with Q^μ from strong provers leads to polices that only produce re-phrasings of the question (App. G) and do not succeed at solving the question. Clearly, *any* policy different from the base policy cannot serve as a prover. So, how do we identify a set of good provers? Can they indeed be weaker than the base policy? We answer in the next sections.

Takeaway: What should process rewards measure during test-time search and online RL?

- Process rewards should correspond to progress, or **advantage**, as opposed to absolute Q-values, for a better explore-exploit tradeoff during beam search and online RL.
- Advantages should be computed using a **prover** policy, different from the base policy.

3.3 ANALYSIS IN A DIDACTIC SETTING: LEARNING A PLANTED SUB-SEQUENCE

In this section, we aim to characterize prover policies that are effective in improving the base policy. To do so, we first introduce a didactic example, representative of real reasoning scenarios to illustrate the main intuition. Then, we will formalize these intuitions in the form of theoretical results.

Didactic example setup. Given an unknown sub-sequence $\boldsymbol{y}^\star$ consisting of tokens from vocabulary $\mathcal{V} := \{1, 2, \dots, 15\}$, we train a policy π to produce a response which contains this sub-sequence. The task completion reward is terminal and sparse, *i.e.*, $r(\boldsymbol{y}, \boldsymbol{y}^\star) = 1$ for a $\boldsymbol{y}$ if and only if $\boldsymbol{y}^\star$ appears in $\boldsymbol{y}$. By design, the reward $r(\boldsymbol{y}, \boldsymbol{y}^\star)$ resembles outcome reward $\text{Rex}(\boldsymbol{y}, \boldsymbol{y}^\star_{\boldsymbol{x}})$ in Sec. 2. The prover policy μ is a procedural policy, parameterized by a scalar $\gamma > 0$ (details in App. B). As γ increases, the performance of μ improves and $\to 1$ as $\gamma \to \infty$. For simplicity, we assume oracle access to ground-truth A^μ and Q^π, and alleviate errors from learned verifiers approximating these values.

(1) RL with effective reward $Q^\pi + \alpha A^\mu$ is $10\times$ more sample-efficient than only outcome reward. In Fig. 3(a), we first note that training π with this effective reward under a prover μ with strength $\gamma = 10$, produces optimal performance (100% accuracy) in < 1k iterations, despite starting from a mediocre initialization for π ($\gamma = 5.0$). Training with only outcome reward is ineffective and requires

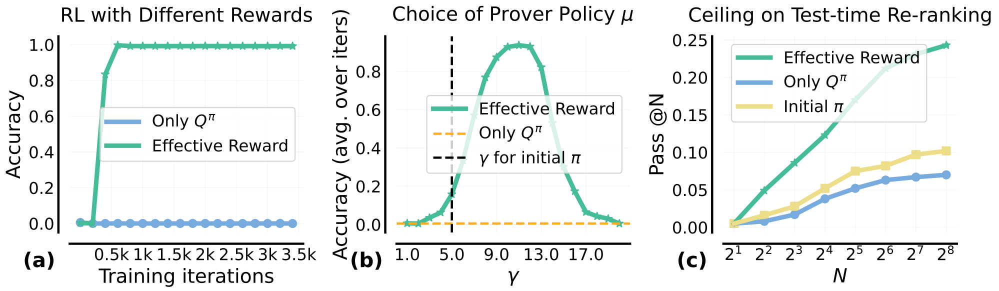


Figure 3: ***Results for our didactic analysis:*** **(a):** We train base policy via RL with either the effective reward $Q^\pi + \alpha A^\mu$, or the more typical Q^π. **(b):** We vary the strength γ of the prover policy μ used to compute advantages A^μ in the effective reward, and plot the base policy accuracy averaged over the RL run. **(c):** We plot the max score out of N responses (Pass @N) sampled *i.i.d.* from an undertrained base policy (iter 100) .

>10k iterations. More importantly, in Fig. 3(b), we note that effective rewards only help for a set of provers, in $\gamma \in [8.0, 15.0]$. Outside this range, we observed $A^\mu \approx 0$ on most states, either because μ was poor (small γ) and was unable to generate $\boldsymbol{y}^\star$ even when π got the sequence partially correct, or because μ was strong (large γ) that it generated $\boldsymbol{y}^*$ with almost equal likelihood from all prefixes.

(2) Effective reward improves Pass @N by $5\times$ **over only outcome reward.** We report the "Pass @N" performance in Fig. 3(c), which measures the maximum reward r across N traces sampled *i.i.d.* from π and hence, presents the ceiling on the performance of any test-time search method that picks a single response from multiple draws (*e.g.*, as in Best-of-N). For a policy trained with the effective reward for 100 iterations, the Pass @N performance grows 5× faster with N, compared to the policy trained with only the outcome reward. Due to only sparse feedback, the latter policy does not learn to sample partially correct $\boldsymbol{y}^\star$, whereas a policy trained with the effective reward produces partially correct $\boldsymbol{y}^\star$, and is able to sample the complete $\boldsymbol{y}^\star$ with higher likelihood during Pass @N.

Takeaway: Online RL with process rewards from different prover policies.

Effective rewards $Q^\pi + \alpha A^\mu$ from prover μ: (i) improve sample efficiency of online RL, and (ii) yield policies with better Pass @N performance, over using only outcome rewards. But, advantages of very capable or poor μ do not improve base policy beyond outcome rewards.

3.4 THEORY: PROVERS COMPLEMENTARY TO BASE POLICY BOOST IMPROVEMENT

From our didactic analysis, it is clear that process rewards A^μ under different provers μ disparately affect the base policy that optimizes $Q^\pi + \alpha A^\mu$ via online RL. We now present a formal analysis of why this happens and characterize a class of provers that can guarantee non-trivial improvements to the base policy. For simplicity, we assume oracle access to Q^π, A^μ at every state-action pair $(\boldsymbol{s}_h, a_h)$ and prove our result in the tabular RL setting, where the policy class is parameterized using the softmax parameterization in Agarwal et al. (2021). Proofs for this section are in App. F.

Main intuitions. We expect a prover μ to improve a base policy π only when $\boldsymbol{\mu}$ **is able to** ***distinguish*** **different actions taken by** π, by attaining sufficiently varying advantage values $A^\mu(\boldsymbol{s}_h, a)$ for actions a at state $\boldsymbol{s}_h$. This can be formalized under the notion of sufficiently large variance across actions, $\mathbb{V}_{a\sim\pi}\left[A^\mu(\boldsymbol{s}_h, a)\right]$. In that case, can we simply use a policy with large advantage variance under any measure? No, because when the prover μ ranks actions at a given state very differently compared to the base policy π (e.g., if A^μ and A^π are opposite), then effective rewards $Q^\mu + \alpha A^\pi$ will be less reliable due to conflicting learning signals. Thus, we want $\mathbb{E}_\pi\left[\langle A^\mu, A^\pi\rangle\right]$ to not be too negative, so that $\boldsymbol{\mu}$ **and** $\boldsymbol{\pi}$ **are reasonably** ***aligned*** on their assessment of steps from π.

In Theorem 3.1, we present our result on policy improvement where the base policy is updated with natural policy gradient (Kakade, 2001): $\pi_{t+1}(a \mid \boldsymbol{s}_h) \propto \exp(\gamma \cdot (A^\pi(\boldsymbol{s}_h, a) + A^\mu(\boldsymbol{s}_h, a)))$. We note that in this idealized update rule, swapping Q values (of μ or π) with advantages does not affect the update, since we only subtract an action-independent offset $V(\boldsymbol{s}_h)$. Consequently, the analysis only uncovers good choices for the prover policy μ for process reward A^μ, and is orthogonal to the design consideration of advantages or Q-values as process rewards. Theorem 3.1 formalizes our intuition by showing that policy improvement at iteration t, grows as the variance in A^μ values increases (higher distinguishability) and reduces when A^μ and A^π become extremely misaligned. This will then allow us to discuss a special case for the case of Best-of-K policies as provers as an immediate corollary.

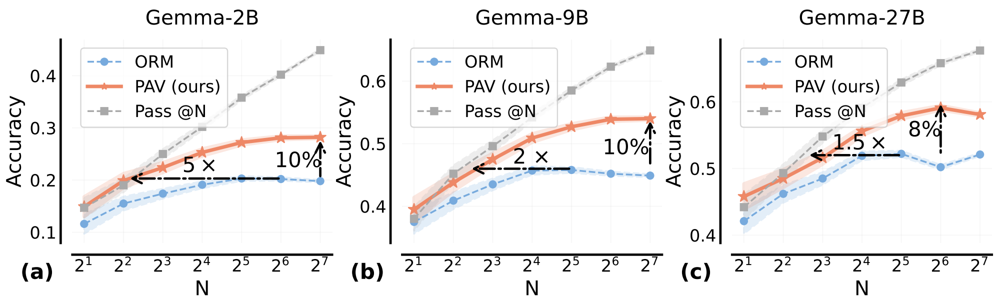


Figure 4: ***For test-time search, PAVs are*** $8-10\%$ ***more accurate and*** $1.5-5\times$ ***more compute efficient over ORMs:*** On samples from **(a)** Gemma-2B , **(b)** 9B , and **(c)** 27B SFT policies, we run test-time beam search with the effective reward $Q^\pi + \alpha A^\mu$ (PAV), where μ is the Bo4(π) policy. We compare beam search performance with best-of-N, re-ranking with a trained outcome verifier (ORM), or the oracle Rex (Pass @N).

Theorem 3.1 (Lower bound on policy improvement; informal)**.** *For base policy iterate* π_t*, after one step of policy update, with learning rate* $\gamma \ll 1$*, the improvement over a distribution of states* ρ*:*

$$\mathbb{E}_{s\sim\rho}\left[V^{\pi_{t+1}}(s) - V^{\pi_t}(s)\right] \gtrsim \gamma \cdot \underbrace{\mathbb{E}_{s\sim\rho}\mathbb{V}_{a\sim\pi_t}\left[A^\mu(s,a)\right]}_{\text{distinguishability from } \mu} + \gamma \cdot \underbrace{\mathbb{E}_{s\sim\rho}\mathbb{E}_{a\sim\pi_t}\left[A^\mu(s,a)A^{\pi_t}(s,a)\right]}_{\text{alignment between } \pi_t \text{ and } \mu} \qquad (6)$$

It may seem that the base policy π can only learn from an improved prover μ, but our result shows that a **weak prover can also amplify a stronger base policy**, since a weak prover μ may have a lower average of Q^μ under its own measure, but still have higher variance across Q^μ (compared to Q^π) when evaluated under π (see Proposition F.1 in App. F.5 for formal discussion). This tells us that rewarding progress under a prover is different from typical knowledge distillation or imitation learning algorithms (Hinton, 2015; Rusu et al., 2015) that in most cases remain upper bounded by the performance of the stronger teacher. So provers cannot be characterized purely by strength, what is a class of provers that is a reasonable starting point if we were to improve any base policy π?

The policy class of "Best-of-K" (computed over base policies) contain complementary provers. A good starting point to identify good provers for a base policy π, is the class of Best-of-K policies or BoK(π). Recall from Sec. 2 that the performance of BoK(π) increases monotonically with K. Applying Theorem 3.1 to this class, we arrive at Remark 3.1 that recommends using BoK(π) with $K > 1$ as a prover policy for a poor base policy π. Conversely , when $Q^\pi(s,a) \approx 1$, then increasing K too much can hurt variance on those states. In the next section, we empirically note that the policies in the class of BoK(π) indeed induce different performance gains when used as prover policies, and we find Bo4 to be a good choice for test-time search over most base policies.

Remark 3.1. *When* $Q^\pi(s,a) = O(1/K), \forall s, a$*, using* BoK($\pi$) *as a prover for base* π *improves distinguishability (and improvement) by* $\Omega(K^2)$*, and make alignment worse at most by* $O(K)$.

Takeaway: Formal characterization of good prover policies that improve the base policy.

Provers with advantages that can **distinguish** actions taken by the base policy (more strongly than the base policy itself) but are **not too misaligned** from the base, boost improvements on each update of the base policy. We call such policies ***complementary provers***. BoK(π) for any base policy π for $K > 1$ can provide a good starting choice of prover policies.

4 Test-Time: Efficiently Scaling Test-Time Compute with PAVs

Now, we study how process verifiers can scale up test-time compute. While our derivations from Sec. 3.2 were with RL, we can also use the *effective reward* $Q^\pi(s_h, a_h) + \alpha \cdot A^\mu(s_h, a_h)$ for running beam search over intermediate steps sampled from base policy π. In the end, we select the final response in the beam using an ORM. In practice, we need to train PRMs that predict Q^π (we call these PQVs) and PAVs that predict A^μ, procedures that we discuss in Sec. 4.2. For now, we assume access to trained ORMs, PQVs and PAVs and discuss test-time search results. For clarity, we abuse notation and refer to the estimated effective reward (ORM + α PAV) as PAV directly.

Setup. We finetune Gemma 2B, 9B, and 27B (Gemma Team et al., 2024) on MATH (Hendrycks et al., 2021) via supervised fine-tuning (SFT) to get three base policies. The set of provers consists of the three base SFT policies themselves as well as their best-of-K policies for different values of $K \in \{2^0, \ldots, 2^5\}$. Additional details for the experiments in this section are in App. C.

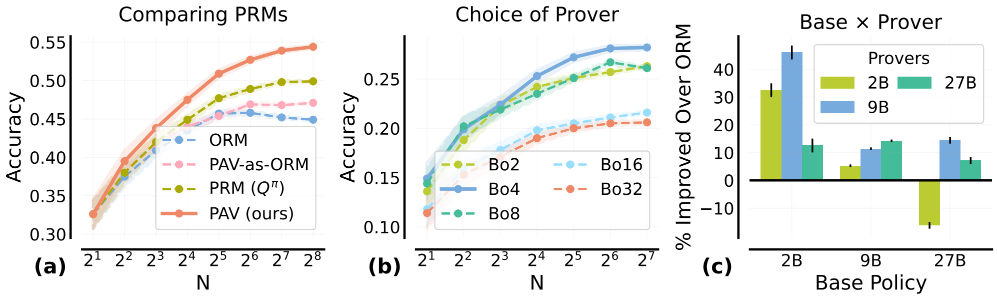


Figure 5: ***Comparing PAVs with search baselines and ablating over the prover policy:*** **(a):** We compare beam search over Gemma 9B SFT, using either effective reward (PAV), or Q^π (Snell et al., 2024), and report best-of-N performance where the re-ranker is either the ORM or PAV-as-ORM. **(b):** For base π Gemma 2B SFT, we run beam search with the effective reward where the prover is $\text{BoK}(\pi)$ for different values of K. In both (a), (b) the x-axis scales the size of the beam or N for best-of-N. **(c):** For each base policy in the set: Gemma 2B, 9B, 27B SFTs, we run beam search with PAVs (beam size of 16) where the prover is another policy from the same set.

4.1 PAVS SCALE TEST-TIME COMPUTE BY 5 – 10× OVER ORMS

Result 1: PAVs are more compute efficient than ORMs. In Fig. 4, we plot the performance of beam search with PAVs for different sizes of the beam N, and compare it with best-of-N using ORMs, *i.e.*, sampling N complete solutions from the base policy and returning the one with the highest ORM score. Our comparisons report *compute efficiency* of PAVs over ORMs: ratio of total compute needed by PAVs to obtain the same performance as running best-of-128 with ORM. Accounting for the additional compute needed for beam search (since each element in the beam samples $C = 3$ next steps, before scoring and pruning the beam), PAVs scale the compute efficiency by **10×** over ORMs for Gemma-2B, 9B base policies, and by 5× for Gemma-27B. For all base policies π, we use $\text{BoK}(\pi)$ with $K = 4$ as the prover policy. We also compare performance with beam search using process verifiers that only predict Q^π, and best-of-N where the ORM is replaced with PAV (PAV-as-ORM). At $N = 128$, similar to Luo et al. (2024), we note a similar gain of 4% for "PAV-as-ORM" Fig. 5(a) over only ORMs, for base Gemma-9B π. When comparing beam search with Q^π (Snell et al., 2024), we find that PAVs scale compute efficiency by 8×. Evidently, advantages from the prover in the effective reward positively impact the beam search. Why does A^μ help, and for what choice of the prover μ?

Result 2: Beam search with too weak/strong provers is sub-optimal. In Fig. 5(b), for the setting when the base policy π is a Gemma-2B SFT model, we compare beam search with PAVs where the provers are given by $\text{BoK}(\pi)$, for different values of K. Recall that as K increases, $\text{BoK}(\pi)$ becomes stronger. Corroborating our analysis in Sec. 3.4, our results show that neither too weak (Bo2) or too strong (Bo32) provers perform best. Instead, across all values of N, we find Bo4 to be dominant. The advantage values $A^\mu \approx 0$ on all steps for very large K, since $Q^\mu(s_h, a_h) = 1-(1-Q^\pi(s_h, a_h))^K \to 1$ on all steps, as we increase K. Hence, ***in order to succeed we need an intermediate-level prover policy***. We make similar observations in Figure 5(c) where we take the three base policies (Gemma 2B/9B/27B) and evaluate beam search with PAVs at $N = 16$ with the Bo4 over the SFT policies acting as provers. We find that for the 2B and 9B base models, the 9B and 27B provers are most effective respectively, whereas for the 27B model, *surprisingly a weaker 9B policy is more effective than the stronger 27B model.* The weaker model presumably offers a complementary signal that distinguishes between different actions taken by 27B, aligning with our theoretical observations in Sec. 3.4.

Result 3: Advantages from the prover policy enable exploration. As discussed in Sec. 3.1, advantage A^μ measures the progress made by an action agnostic of the value of the previous state, where as Q^π measures the promise of a particular state. Given a finite capacity beam, our effective reward (Eq. 5), which linearly combines Q^π *and* A^μ *induces a better tradeoff between exploring new prefixes (states) from where progress can be made and exploiting currently known prefixes with high Q-values.* Exploration at initial steps is critical to ensure that the beam at later steps covers diverse partial rollouts each with a high likelihood of producing the correct answer. Thus over-committing to the beam with actions from the same state, regardless of the progress made by each can prove to be sub-optimal over a selection strategy that balances rewarding previous

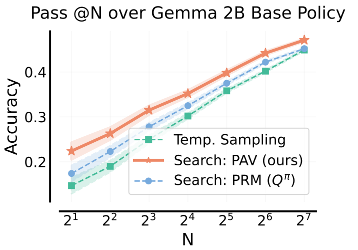


Figure 6: Beam search with PAVs improves exploration efficiency (higher Pass@N), over typical PRMs.

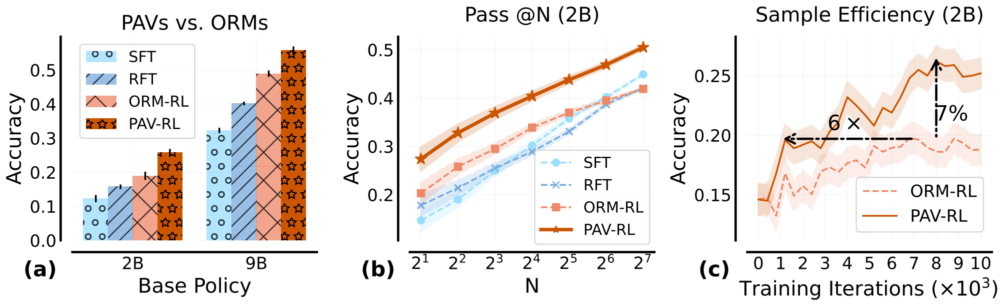


Figure 8: ***PAVs as dense rewards in RL is*** $6\times$ ***more sample efficient than ORMs, and gains*** $+7\%$ ***on accuracy:*** **(a)** We report the performance of a base policy trained using RL with effective rewards (PAV-RL), or only outcome rewards (ORM-RL), and baselines SFT, RFT. **(b):** For the policies trained in (a) we report the best-of-N performance where the oracle reward Rex is used to rank N candidates sampled from the base policy (Pass @N). **(c):** Across training iterations, we report the test performance of policies trained with PAV-RL and ORM-RL.

actions A^μ and current states Q^π. Indeed, we observe in Fig. 6, beam search with PAV enhances pass@N performance of the base Gemma 9B model over beam search with Q^π and *i.i.d.* sampling.

Takeaways: Scaling test-time compute with process advantage verifiers.

- Beam search with PAVs boosts accuracy by >8% & compute efficiency by 1.5-5x over ORMs.
- Utilizing Best-of-K policies (corresponding to the base policy) as provers induce better exploration to maximize outcome reward. Optimal provers for a base policy appear at $K > 1$.

4.2 HOW TO COLLECT DATA TO TRAIN PAVS?: PAV TRAINING DATA SCALING LAWS.

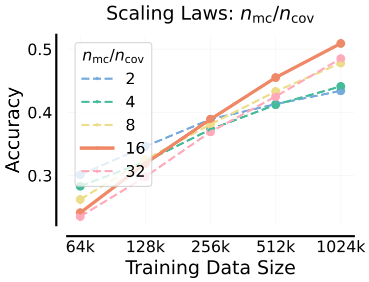


Figure 7: Performance of beam search over Gemma 9B SFT for PAVs trained on datasets with different $n_{\text{mc}}/n_{\text{cov}}$.

We now describe the procedure for training outcome verifiers and PAVs. We can learn to predict Q^π for a policy π (similar for Q^μ) by finetuning LLMs with a cross-entropy loss on the following data with triplets $(\boldsymbol{s}, a, Q^\pi_{\text{mc}}(\boldsymbol{s}, a))$. To collect this data, we first sample n_{cov} "seed" rollouts from the base or prover policy respectively for ORM and PAVs, to promote coverage over prefixes and steps. Then we sample n_{mc} additional rollouts, conditioned on each prefix in the seed rollout to compute the Monte-Carlo estimate of Q^π at each prefix. In Fig. 7 we plot the beam search performance of PAVs trained with different ratios of ${}^{n_{\text{mc}}}/{}_{n_{\text{cov}}}$, as we scale the total dataset size. Here, the beam size is fixed to 128 and the base policy is the Gemma 9B SFT policy and prover is $Bo4$ policy. We find that under low sampling budgets, optimizing for coverage ($n_{\text{cov}} > n_{\text{mc}}$) is better for performance, and when budget is higher, reducing label noise in Q^π_{mc} by setting $n_{\text{mc}} > n_{\text{cov}}$ gets us more improvements. In addition, we also spend some initial sampling budget is spent to identify "high value" states where Q^π is larger than a threshold, and identify the first step with low Q^π on an incorrect partial rollout from this state. We found this strategy to scale better with dataset size, as we discuss in App. D.

5 TRAINING-TIME: SCALING RL SAMPLE-EFFICIENCY WITH PAVS

We can also use PAVs to train policies via online reinforcement learning (RL), by using the effective reward $Q^\pi + \alpha A^\mu$ as dense, per-step rewards. In this section, we compare the sample efficiency of PAV-RL (i.e., $\ell_{\text{PAV-RL}}$ in Eq. 4) with standard ORM-RL (i.e., $\ell_{\text{ORM-RL}}$ in Eq. 3) on Gemma 2B and Gemma 9B SFT models, which are further optimized via rejection finetuning (RFT) (Yuan et al., 2023), before using them to initialize RL. To our knowledge, no prior work has successfully demonstrated the use of dense per-step feedback with a process reward model for RL, and we present the first significant set of results establishing the efficacy of this approach. We show that PAV-RL is much more sample-efficient, and enjoys a higher ceiling on the performance of any test-time re-ranker. Additional details for the experiments are in App. E.

Result 1: PAV-RL is $>7\%$ **better than ORM-RL in test accuracy, and** $6\times$ **sample efficient**. In Fig. 8(a), we report the test accuracies of Gemma 2B and 9B models trained with SFT, RFT, ORM-RL and PAV-RL. PAV-RL improves the RFT policy by 11% for 2B, and 15% for 9B, with $> 7\%$ gain over

ORM-RL in both cases. Not only do the effective rewards from PAV improve the raw accuracy after RL, this accuracy is attained 6× faster (see Fig. 8(c)) for the 2B run and similarly for the 9B RL run (see App. E). For both 2B and 9B, RL runs, we experiment with two options for the prover policy: **(i)** 2B SFT policy; and **(ii)** 9B SFT policy. While both of these provers rapidly become weaker than the base policy within a few gradient steps of RL, a fixed PAV trained with each of these provers is able to still sustain performance gains in RL. More interestingly, we find that the 2B SFT policy serves as the best choice of the prover for both 2B and 9B policies. This observation that a weak prover can still improve the base policy corroborates our results in the didactic setup and our analysis in Sec. 3.4.

Result 2: PAV-RL achieves higher performance ceiling on test-time re-ranking. In Fig. 8(b), for the Gemma 2B model, we plot the Pass @N performance for each method, and find (i) Pass @N is higher (> 7%) for PAV-RL, compared to ORM-RL, for any $N \leq 128$; and (ii) the rate at which Pass @N improves for PAV-RL is higher than ORM-RL. Both these trends are consistent with our observations in the didactic analysis in Sec. 3.3. Notably, for $N \geq 64$, ORM-RL is worse than the SFT policy, perhaps due to lower entropy over the distribution at the next step resulting in non-diverse candidates. So, why does PAV-RL deliver a policy that can generate more diverse candidates, and does not suffer from the low entropy problem in ORM-RL? We answer this with a key insight on how ***the primary benefit of PAVs is to promote efficient exploration.***

Result 3: PAVs improve exploration and discover correct solutions to novel problems. The ORM rewards downweight all steps in an incorrect rollout equally during RL, whereas the effective reward in PAVs, up-weights steps that make progress under the prover, even when the complete rollout is incorrect. This increases the coverage over individual steps that can be proposed at a given prefix. This mechanism for exploration is analogous to test-time search in Sec. 4.1. Hence, the directed supervision from PAVs improves sample-efficiency throughout the course of training (Fig. 8(c)). In fact, we also find that combining the PAV-RL policy with test-time beam search is able to solve a substantially larger number of *new* problems within smaller compute budgets (N=16,32) that the SFT policy cannot solve with a much larger budget N=256 (Fig. 9).

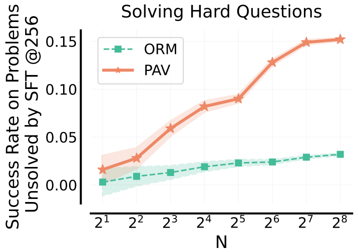


Figure 9: Amongst hard problems that remain unsolved by Best-of-256 over SFT, we check how many are solved by Best-of-N over PAV-RL or ORM-RL.

Takeaway: RL with process advantage verifiers (PAVs) as dense rewards

- Using trained PAVs as dense rewards in RL boosts scales sample efficiency by 6×, compared to only using sparse ORM rewards, and results in policies with a higher Pass @N performance.
- Advantages from a complementary prover policy improves the sample efficiency of exploration in RL, and produces policies that can discover solutions to hard novel questions.

6 Related Work and Discussion

We briefly discuss some key related works here, and leave the detailed discussion for App. A. To address issues of sparse feedback in ORMs (Cobbe et al., 2021b), recent works (Lightman et al., 2023; Uesato et al., 2022) trained process reward models (PRMs) to densely predict incorrect steps in a multi-step reasoning trace. Since human data collection for process labels is not scalable enough, recent work (Wang et al., 2024; Luo et al., 2024) used automated supervision to annotate steps with Q values under the base policy, *i.e.*, the PRMs score a step with the likelihood of future success, when continuing to sample from the step. While Q-value PRMs in Lightman et al. (2023); Luo et al. (2024) were mainly used as verifiers for re-ranking, Snell et al. (2024) used them for test-time beam search. Shao et al. (2024) uses PRMs for RL but found a gain of only $1-2\%$ with PRMs. In our work, we question solely relying on Q-values or advantages of the base policy, and find that measuring progress (i.e., advantages) under a different prover policy can amplify exploration, thus boosting test-time search and RL. To our knowledge, we are the first to show substantial gains in compute and sample efficiency with PRMs. Our methodology for data collection is similar to (Setlur et al., 2024; Hwang et al., 2024) (i.e., identify "first pits" in reasoning traces), these works only use it to collect preference pairs. Beyond all of these, we also characterize which policy to use for computing advantages.

Conclusion. Our key finding is that process rewards that improve a base policy can go beyond step-level correctness. In particular, we show that rewards given by advantages of a prover policy enhances exploration when sampling from the base policy, improving test-time search and RL efficiency.

7 REPRODUCIBILITY STATEMENT

For our experiments on the didactic setup in Section 3.3, we provide details in Appendix B. For experiments on training process advantage verifiers (PAVs) we provide details in Appendix D. Details for experiments that use the trained PAVs for test-time search in Section 4, are provided in Appendix C. Finally, for our experiments on training the base policies with PAVs as dense rewards in reinforcement learning (Section 5), we provide details in Appendix E. All proofs for our theoretical claims can be found in Appendix F. The pretrained model checkpoints for Gemma-2B, 9B, and 27B used in this work are available publicly at https://huggingface.co/google, and the MATH dataset we use from Hendrycks et al. (2021) is also public here: https://github.com/hendrycks/math.

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.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

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

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Jonathan D Chang, Kiante Brantley, Rajkumar Ramamurthy, Dipendra Misra, and Wen Sun. Learning to generate better than your llm. *arXiv preprint arXiv:2306.11816*, 2023.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning*, pp. 2058–2066. PMLR, 2015.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021a.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021b.

Michael Collins. Discriminative reranking for natural language parsing. In *Proceedings of the International Conference on Machine Learning*, 2000.

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

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pp. 881–889. PMLR, 2015.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

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. *NeurIPS*, 2021.

Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

Hyeonbin Hwang, Doyoung Kim, Seungone Kim, Seonghyeon Ye, and Minjoon Seo. Self-explore to avoid the pit: Improving the reasoning capabilities of language models with fine-grained rewards. *arXiv preprint arXiv:2404.10346*, 2024.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274, 2002.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, volume 14. Advances in neural information processing systems, 2001.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *arXiv preprint arXiv:2406.14532*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023a.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023b.

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

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International conference on machine learning*, pp. 3309–3318. PMLR, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. The MIT Press, second edition, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference Fine-Tuning of LLMs Should Leverage Suboptimal, On-Policy Data, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Fei Yu, Anningzhe Gao, and Benyou Wang. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*, 2023.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

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

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

Appendices

A Additional Related Work

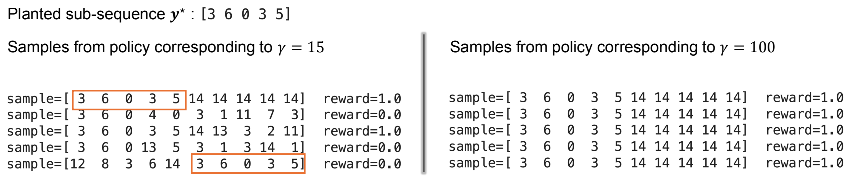
In this section, we expand on the related works we discuss in Section 6. Our work is closely related to three categories of works that aim to improve the math reasoning capabilities of large language models. First, we look at works that train verifiers to provide outcome level feedback (Cobbe et al., 2021b; Hosseini et al., 2024; Zelikman et al., 2022; Singh et al., 2023b) on the correctness of the full response (ORM). Next, we look at works that alleviate issues with sparse feedback in ORMs, and instead train process reward models, that can perform credit assignment. They are trained either through human annotations (Lightman et al., 2023; Uesato et al., 2022), or automated forms of supervision (Snell et al., 2024; Wang et al., 2024; Luo et al., 2024). While most PRMs and ORMs have been used to collect supervised finetuning data or for test-time verfication, a selected set of recent works also experiment with using them for training LLMs with online RL (Wang et al., 2024; Uesato et al., 2022; Shao et al., 2024). Below, we discuss each of these categories separately.

Outcome reward models (ORMs) or verifiers (Cobbe et al., 2021b; Uesato et al., 2022) are commonly used to improve the test-time performance using Best-of-N, where we generate multiple candidate solutions from the base policy (LLM), rank them using verifier, and pick the best one. ORMs are trained to assess correctness of a solution either using binary classification (Cobbe et al., 2021a; Yu et al., 2023), preference optimization using DPO (Hosseini et al., 2024), or next-token prediction (Zhang et al., 2024). Furthermore, prior works train LLMs on self-generated data using ground-truth outcome rewards (Rex), either via supervised fine-tuning (Zelikman et al., 2022; Singh et al., 2023a; Yuan et al., 2023), or online RL (Bi et al., 2024). In contrast to these approaches, our work focuses on process reward models (PRMs) for improving performance with beam-search at test time as well as online RL where we jointly maximize the effective reward in equation 5 which linearly combines both Rex (outcome supervision) and process supervision in the form of advantages under a prover policy, distinct from the base policy.

PRMs and credit assignment. Several works focus on training step-level PRMs on math reasoning tasks, either using human labels (Lightman et al., 2023) or automated LLM-generated data to estimate value functions (Wang et al., 2024; Luo et al., 2024). Our work also focus on automated data collection for PRMs but empirically argues for additionally using the advantage function as step-level rewards, with a conceptual explanation focused in Section 3.1. Several prior works have explored step-level search algorithms with PRMs, such as beam search (Snell et al., 2024), heuristic greedy search (Ma et al., 2023), and reward-balanced tree search (Wu et al., 2024). Hwang et al. (2024); Setlur et al. (2024) use advantages to identify the “first pit” in an incorrect reasoning trace. Specifically, they collect data by computing advantages at each step using Monte Carlo rollouts. Then in an incorrect trace, they identify the step with the least advantage, and use the prefix of that step to construct preference pairs for offline direct preference optimization (Rafailov et al., 2023). In contrast, our work computes advantages under a prover policy, that we formally characterize, and use the computed advantages for improving test-time search and efficiency of online reinforcement learning.

Online RL for math reasoning. Once we have a trained outcome or process verifier, it is natural update a policy by optimizing it against the learned signal, similar to how learned reward models are optimized in RLHF (Ouyang et al., 2022). In the context of math reasoning, Uesato et al. (2022); Havrilla et al. (2024); Shao et al. (2024) trained policies with RL, experimenting with both dense and sparse rewards. In all three instances, the gains observed by using PRMs trained to predict step-level correctness (similar to Lightman et al. (2023)) is quite small, compared to simply using trained ORMs, or the ground-truth outcome supervision Rex. In fact, Havrilla et al. (2024) states that the only algorithm that does well is a form of expert iteration Anthony et al. (2017), which does not inhibit exploration as severely as some other approaches they compare with. Our work presents one of the first results, where trained PRMs, used in conjunction with the outcome rewards during online RL, result in policies with substantially higher (+7%) performance, than the one trained only with outcome supervision. Our results also indicate a 6× sample efficiency boost for online RL, with our trained PRM (PAV).

Connections to imitation learning through RL. Finally, in addition to the above three sets of papers, we note that the idea of mixing potential functions from different policies μ and π, in order



Planted sub-sequence $\boldsymbol{y}^\star$: [3 6 0 3 5]

Samples from policy corresponding to $\gamma = 15$

```
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
sample=[ 3  6  0  4  0  3  1 11  7  3]  reward=0.0
sample=[ 3  6  0  3  5 14 13  3  2 11]  reward=1.0
sample=[ 3  6  0 13  5  3  1  3 14  1]  reward=0.0
sample=[12  8  3  6 14  3  6  0  3  5]  reward=0.0
```

Samples from policy corresponding to $\gamma = 100$

```
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
sample=[ 3  6  0  3  5 14 14 14 14 14]  reward=1.0
```

Figure 10: ***Pictorial description of our planted sub-sequence didactic setup*:** An example showing five samples drawn *i.i.d.* from a very strong $\gamma = 100$, and relatively weaker $\gamma = 15$ policies in our didactic setup.

to improve upon a sub-optimal expert μ appears in Chang et al. (2015), but this work considers the structured prediction problem which is vastly different from our setting. Related to this, is the work by Chang et al. (2023), which uses a "guide" policy to rollout from prefixes generated by a base policy. The base policy can now imitate the guide by cloning those rollouts, and eventually surpass. Our work also uses a prover policy which can complete rollouts from states where the base policy fails. But, we also show that weak provers in many cases are able to improve the base policy, or search over its responses, better than a stronger prover policy. We tie this observation to the insight that the main goal of the prover policy is to distinguish steps taken by the base policy, as measured by advantages under the prover. Thus, we do not require the prover policy to be something better than the base, that the base can rely on to clone rollouts that lead to correct final answers.

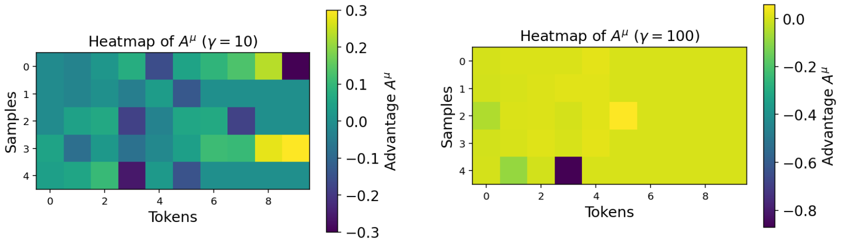


Figure 11: ***Comparing heat map of advantages from weak ($\gamma = 10$) and strong provers ($\gamma = 100$):*.** When the strength of the prover policy increases, the magnitude of step-level rewards reduces (since $A^\mu \approx 0$), as noted in the differences between the left (weak prover) and right plots (strong prover).

Magnitudes of step-level rewards for weak vs. strong prover policies. For 5 random samples drawn from the base policy, each with 10 tokens, we plot the values of the advantage A^μ, at each token of each sample. In Figure 11, we plot heatmaps of the advantages for two different choices of the prover policy μ, one with $\gamma = 10$ (weak prover), and the other with $\gamma = 100$ (strong prover). For the weak prover, the advantage magnitudes are much higher, since the weaker prover policy is more likely to complete a partially correct generation over an incorrect one, *i.e.*, a generation where the last set of tokens in the prefix partially matches the planted sub-sequence. On the other hand, when the strength of the prover policy increases, the magnitude of token-level rewards reduces ($A^\mu \approx 0$) since it can complete the solution with nearly equal likelihood from both correct and incorrect prefixes sampled from the base policy. Thus, RL training with the effective rewards (equation 5) computed using strong provers performs similarly to RL training with only outcome rewards, as noted in Figure 3b.

B DIDACTIC ANALYSIS

We consider sequences of length 10 from a 15-token vocabulary $\mathcal{V} := \{0, 1, 2, \ldots, 14\}$, where the end-of-sequence token is given by 14. Given an unknown planted sequence $\boldsymbol{y}^\star$ (in Figure 10), we train a policy π with on-policy policy gradient, where the reward is terminal and sparse, *i.e.*, for $\boldsymbol{y} \sim \pi$ we have $r(\boldsymbol{y}, \boldsymbol{y}^\star) = 1$ if and only if $\boldsymbol{y}^\star$ appears in $\boldsymbol{y}$, and 0 otherwise (Figure 10). The policy π in our experiments is represented by a multi-layer neural network, similar to the MADE architecture (Germain et al., 2015). The prover policy μ is parameterized by a scalar $\gamma > 0$. In particular, at any state $\boldsymbol{s}$, where the last k tokens leading up to $\boldsymbol{s}_h$ match first k tokens of $\boldsymbol{y}^\star$, then:

$$\mu(\boldsymbol{y}^\star_{k+1} \mid \boldsymbol{s}) \propto \gamma,$$

and uniform on all other tokens. Thus, as γ increases, the performance of μ improves and $\rightarrow 1$ as $\gamma \rightarrow \infty$. For our experiments, we assume oracle access to ground-truth advantage and Q-values, thus mitigating any confounding issues due to usage of a learned verifier.

Training details. We use effective rewards with $\alpha = 1$, and use the gradient in equation 5 to update the policy via policy gradient iterations. For the ORM runs, where we only use the outcome reward r, the policy gradient is equivalent to the case where $\alpha = 0$ in equation 5. We train for 10,000 iterations in both cases, with a batch size of 64, and a constant learning rate of $1e-3$ for the Adam optimizer. The RL runs are initialized with a supervised finetuned policy. For this we take a randomly initialized network, based on the MADE architecture (Germain et al., 2015), with 3 layers, and 128 hidden units in each. Then we train it with supervised next-token prediction loss for 50 iterations on a dataset of 3200 samples from a weak policy ($\gamma = 5.0$).

C ADDITIONAL DETAILS: EXPERIMENTS ON TEST-TIME SEARCH WITH PAVS

Implementation details. For our experiments in Sec. 4, we consider three base policies: Gemma 2B, 9B and 27B. We finetune each of these on the MATH (Hendrycks et al., 2021) dataset. The finetuning is done for 5000 iterations, with a batchsize of 32, with a maximum learning rate of $5e-6$ for 2B, 9B and $5e-7$ for 27B policy. We used the Adam optimizer, with a linear warm up and cosine decay learning rate schedule. The linear warm up is done for the first 500 iterations only. Given the SFT checkpoints, we next train PAVs using the procedure in Sec. 4.2. We do this for a class of provers, which include the base policies themselves. As we discuss in Sec. 4, the prover class also includes the best-of-K policy for K in $\{2, 4, 8, 16, 32\}$.

We used a hold out validation set to ascertain the value of α in the effective reward. For each base policy we run beam search with a beam size of 16 on this hold out validation set, and using the base policy itself as a prover, we evaluate the value of α that works best in the effective reward. We find that $\alpha = 0.5$ worked best for Gemma 2B and 9B base policies, while a lower value of $\alpha = 0.2$ was optimal for Gemma 27B. To tune α we ran a grid search over the range $[0.0, 1.0]$, evaluating at an interval of 0.1. We observe that the choice of α is a relatively robust one, since for all three base policies, we saw improvements (over only Q^π as the reward) for values in the range of $[0.2, 0.6]$.

Experiment: Is the effective reward able to predict the final outcome better than Q^π? In Fig. 12, we describe an experiment where for both the effective reward $Q^\pi + \alpha A^\mu$ and just Q^π, we compute the error of the classifier that makes a prediction on the final outcome by thresholding on either reward value at each step of the rollout. This threshold is computed using a validation set, and is separate for each step and reward combination. The figure tells us that the outcome prediction error drops for both rewards, as the base policy is rolled out more, the effective reward dominates Q^π across all steps. Thus, the effective reward is a more informative signal (lower classification error) for the problem of predicting the success of a partial rollout, especially in the earlier steps of the rollout. This helps to explain the better performance of beam search with a finite capacity beam that re-ranks partial rollouts with the effective reward.

Experiment: Explicitly comparing test-time search using our PAVs, with PRMs from prior works. Several works on automated PRMs, use the Q-function as the step-level reward Wang et al. (2024); Luo et al. (2024); Snell et al. (2024). Each of these use the step-level rewards in different ways at test-time, and we compare with all of these approaches in Section 4. In Figure 4a, we compare the performance of PAVs with Q^π for test-time beam search (see the line for "PRM Q^π"), as done in Snell et al. (2024). In the same figure, the line for "PAV-as-ORM" corresponds to using the trained PAVs as outcome reward models, *i.e.*, to score the full sequence. The sequence is scored by computing the minimum over the step-level scores from the PAV model. This is similar to how Wang et al. (2024); Luo et al. (2024) use their trained PRMs during test-time search. In Figure 13, we make these comparisons explicit. In addition, we run exactly the same procedure for test-time best-of-N search with PRM Q^π as done in Wang et al. (2024); Luo et al. (2024).

Confidence intervals: We compute a 95% confidence interval over the true mean of the reported test-time search performance, for each algorithm in Figure 4. This mean is computed over the randomness in the test data, and the randomness in the search algorithm. To account for both, we run each algorithm 5 times, and then use the 500 examples in the test set, to compute the confidence

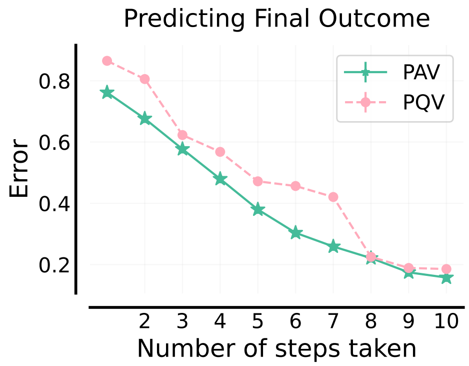


Figure 12: ***Effective rewards at any step are able to predict the outcome rewards at the final step, better than Q^π:***For both the effective reward $Q^\pi + \alpha A^\mu$ and just Q^π, we compute the error of the classifier that makes a prediction on the final outcome by thresholding on either reward value at each step of the rollout. This threshold is computed using a validation set, and is separate for each step and reward combination. The figure tells us that the outcome prediction error drops for both rewards, as the base policy is rolled out more, the effective reward dominates Q^π across all steps.

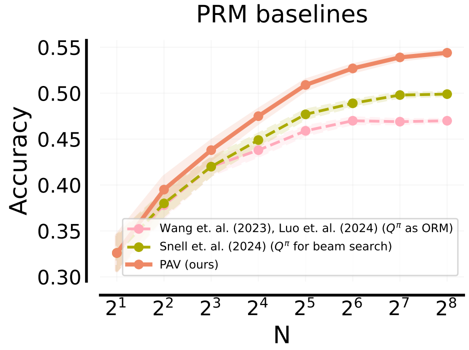


Figure 13: ***Explicit comparison with PRM baselines in prior works:*** We plot the performance of beam search with PRM Q^π run by Snell et al. (2024), and best-of-N search with the same PRM run by Wang et al. (2024); Luo et al. (2024). We compare both with beam search run using PAVs. Note that the search mechanism for PAVs and the one in Snell et al. (2024) is the same, with the only difference being in the re-ranking function at each step of the beam search.

interval of the true performance of each independent run of the algorithm. Then, we average the means of the confidence intervals across the 5 runs to compute the final 95% confidence interval.

D Details on Collecting Data and Training PAVs

In Fig. 7 in Sec. 4.2, our seed rollouts are *i.i.d.* sampled from π, but prior works (Luo et al., 2024; Setlur et al., 2024; Hwang et al., 2024) employed a “first pit” strategy for coverage. Here, some initial sampling budget is spent to identify ”high value” states where Q^π is larger than a threshold. Then, for any incorrect sample from these high value states greater budget is spent to estimate the first step (first pit) with $Q^\pi = 0$. All prefixes (and their estimated Q-values) until the first pit are then added to the training data. In Fig. 14, we compare beam search using PAVs trained using data from the first pit strategy, and the random sampling strategy. Both of them use the best value of $n_{\text{mc}}/n_{\text{cov}}$ from Fig. 7 for every dataset size. We find the first pit strategy to be better than random, especially when the number of seed rollouts are limited. Once we get coverage over such pits, we sample a large number of partial rollouts conditioned on each prefix until the first pit. This is used to compute the Monte Carlo estimate of Q values more accurately on the path to the first pit. Each prefix and estimated Q value pair is then added to the dataset used to train PAVs.

Training details. All PAVs used in this work are trained by taking the Gemma 9B pretrained checkpoint and finetuning it on the data collected from the above strategy. The data collection uses first pit strategy for better coverage over pits in the seed rollouts. Based on findings in Fig. 7, we use a high value of $n_=20$ to estimate the Q-values accurately for each step in the seed rollout. For each base policy, in total, we collect a dataset of over $300,000$ prefix, $\hat{Q}^\pi$-value pairs (where $\hat{Q}^\pi$ is the Monte Carlo estimate) on which we finetune the Gemma 9B model with cross-entropy loss. Since the distribution of values for $\hat{Q}^\pi$ can be skewed, we split the range of $\hat{Q}^\pi$-values into two buckets, based on which we also partition the training data. The first bucket is the set of all prefixes with $\hat{Q}^\pi < 0.5$ and the second is the set of all prefixes with $\hat{Q}^\pi \geq 0.5$. Then, we use class-balanced sampling over these buckets to train the finetune the pretrained model for 20000 training iterations, using a batch size of 32. We use an Adam optimizer with a maximum learning rate of $5e-7$. We use a linear warm up (till 2000 steps), and a cosine decay learning rate schedule to train the models. Since a pretrained LLM would output a matrix of logits (vocabulary size × sequence length) we fix a token as the "scoring token" to be the end of the sequence / prefix that needs to be scored. The logits of this scoring token are then used to determine the prediction for the LLM being trained.

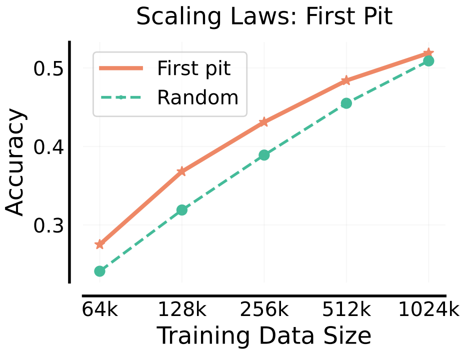


Figure 14: ***First pit strategy from Luo et al. (2024); Setlur et al. (2024)***: We compare the beam search performance (with beam size 128) for a PAV trained on data collected using two types of seed rollouts. For the seed rollouts, we either randomly sample from the distribution of state action pairs induced by the base policy. Or we improve coverage particularly using the "first pit" strategy of identifying the first state with low Q values on a partial rollout that starts from a high Q-value state and ends with an outcome reward of 0, i.e., Q value is 0.

Using trained PAVs: For test-time search over responses from the base policy π, and fixing a prover policy μ, the following is how we use PAVs for beam search at test-time. At any given time step, the beam holds partially generated responses (states) to a math problem. We score each of the states using our effective reward $Q^\pi + \alpha A^\mu$. Since A^μ is computed using Q^μ (equation 2), we need only train verifiers to predict Q^π and Q^μ. Finally, at the end of beam search, when only we have complete responses in the beam, we pick the best response using an ORM trained only to predict the correctness of final answers. In practice, we find that using a trained ORM to rate complete responses at the end of beam search is not critical, and using the PRM that predicts Q^π to rank the full responses also performs similarly. So, the training of an ORM model can be avoided.

E ADDITIONAL DETAILS: EXPERIMENTS ON RL TRAINING WITH PAVS

Experiment: RL training of Gemma 9B SFT policy with PAV-RL. Similar to Figure 8)(c), where we compared the sample efficiency and raw performance of PAV-RL with ORM-RL for the Gemma 2B SFT base policy, in Figure 15, we compare the same for the Gemma 9B SFT base policy. Here, the prover policy for the effective reward in PAV-RL is the Gemma 2B SFT policy, which is worse than the Gemma 9B SFT policy, let alone the policy iterates during RL training. We find PAV-RL to boost sample efficiency by 5×, while being 6% better in raw accuracy.

Experiment: Discounting strong provers. In Section 5 we find that the better prover policy to train the Gemma 2B base policy is the Gemma 2B prover policy, and the stronger Gemma 9B RL policy. We attribute this finding to our discussion in Section 3.2, where we explain how very strong provers end up with advantages A^μ which are close to 0, since they maybe able to complete the solution from

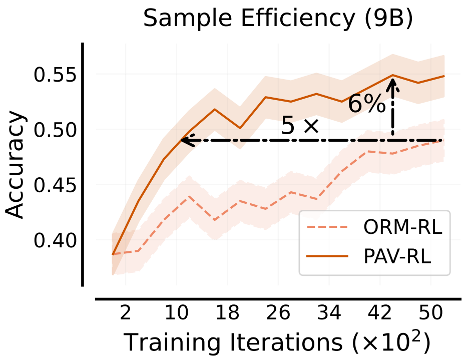


Figure 15: ***Sample efficiency and performance gains for PAV-RL on Gemma 9B SFT policy:*** We plot the test accuracy across PAV-RL and ORM-RL training iterations, where the base policy is the Gemma 9B SFT policy. The effective rewards in PAV-RL boost the sample efficiency by 5× over outcome rewards in ORM-RL. Additionally, PAV-RL trained policy is 6% more accurate over than ORM-RL trained policy, when compared over the course of 5000 training iterations.

most states generated by the base policy. But what happens if we account for the steps taken by the stronger prover policy to complete the solution? We test this with the following experiment

We train PAVs to predict the advantages of discounted rewards from strong prover policies. Here, for the problem $\boldsymbol{x}$, and the state, action pair $\boldsymbol{s}, a$, the process rewards are given by the effective reward from equation 5: $Q^{\pi} + \alpha A^{\mu}$, except that the advantage A^{μ} is the difference in discounted rewards, i.e.:

$$A^{\mu}(\boldsymbol{s}, a) \quad = \quad \mathbb{E}_{\boldsymbol{y} \sim \mu(\cdot|\boldsymbol{s}, a)}\left[\lambda^{\mathrm{len}(\boldsymbol{y})-\mathrm{len}(\boldsymbol{s})-1}\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}_{\boldsymbol{x}}^{\star})\right] - \mathbb{E}_{\boldsymbol{y} \sim \mu(\cdot|\boldsymbol{s})}\left[\lambda^{\mathrm{len}(\boldsymbol{y})-\mathrm{len}(\boldsymbol{s})}\mathrm{Rex}(\boldsymbol{y}, \boldsymbol{y}_{\boldsymbol{x}}^{\star})\right],$$

where the prover policy samples solution $\boldsymbol{y}$ with $\mathrm{len}(\boldsymbol{y})$ steps to complete the solution, from a state $\boldsymbol{s}$, which already has $\mathrm{len}(\boldsymbol{s})$ steps in it.

For this setting, we train a verifier to predict discounted rewards for the Gemma 9B prover policy. We find that the discounted process rewards from the stronger 9B prover policy performs worse than undiscounted rewards from the weaker Gemma 2B prover policy, when using either to train the 2B base policy with online RL (Figure 16).

The main reason for the discounted rewards to not enable the use of strong provers is because strong prover policies tend to disregard states generated by the base policy (as illustrated in Figure 2b). This means, that irrespective of whether the weak prover policy generates a partially correct or incorrect solution, when we rollout the strong prover policy from this state generated by the base policy, the strong prover directly attempts to answer the math problem with its own solution trace. Thus, from any state the strong prover is expected to complete the solution with roughly the same number of steps. This means that $A^{\mu} \approx 0$ even in the discounted case, which reduces the ability of the strong prover policy to distinguish steps taken by the base policy.

Experiment: Using PRMs from prior works to assign step-level rewards during online RL. We also experiment with using the PRMs proposed in prior works Snell et al. (2024); Wang et al. (2024); Luo et al. (2024) as step-lever rewards in online RL. For this, we use Q^{π}_{base} as the step-level score during RL training initialized with the base policy π_{base}. This step-level reward is used to reward trajectories sampled during online RL training, in addition to the outcome reward. We plot our results in Figure 17 and find that the test accuracy drops quite quickly, even though the train rewards keep improving, since the policy just learns to hack the step-level Q-values, for the reasons we explain in Section 3.2. We see a similar collapse when we use Q^{μ} instead of A^{μ} (see Appendix G for qualitative examples).

Training details. As discussed in Section 5, the initialization for RL training is the RFT (rejection finetuned) checkpoint for the corresponding base policies. More specifically, we consider two base policies Gemma 2B SFT, and Gemma 9B SFT, where the RL training is initialized with the policy obtained by further optimizing the base policy via rejection finetuning (RFT) Yuan et al. (2023). This is done to improve coverage over states and actions during the initial iterations of RL. For rejection

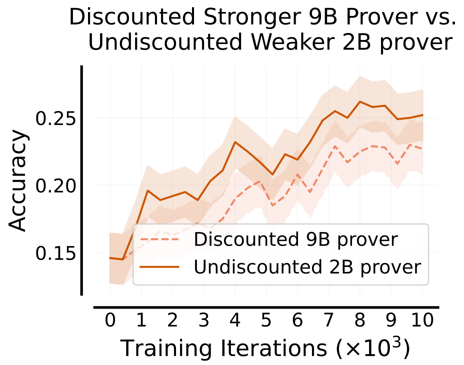


Figure 16: ***Comparing weaker, undiscounted 2B prover policy with discounted, stronger 9B prover:*** We plot the test accuracy across PAV-RL training iterations, where the base policy π is the Gemma 2B SFT policy, and two different options of the prover policy μ used in the computation of the effective rewards in equation 5. The discounted stronger 9B prover policy still under performs the weaker Gemma 2B prover policy.

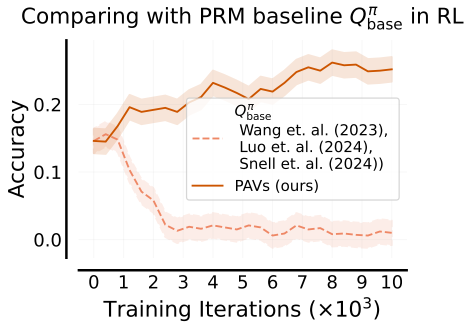


Figure 17: ***Using PRMs from prior works to assign step-level rewards during online RL:*** . We take the PRM that predicts Q-values of the base policy, *i.e.*, Q^{π}_{base}, and use it to assign step-level rewards during online RL training, in addition to the outcome reward. Note that, Snell et al. (2024); Wang et al. (2024); Luo et al. (2024) used the same automated PRM Q^{π}_{base} to run test-time beam search or best-of-N search over π_{base}. We find that using the same for online RL leads to the step-level rewards being hacked for reasons we explain in Section 3.2.

finetuning we train the SFT model (base policy) on all correct trajectories sampled when collecting seed rollouts for training PAVs (that predict the advantage of the same base policy). The training details for RFT remain same as SFT and are detailed in Appendix C. The RL training is run for 10000 iterations for the 2B model, and for 5000 iterations for the 9B model. For both we use the Adam optimizer with a learning rate of 1e-7, and a batchsize of 32. The maximum response length is set to 512. For both we used a learning rate schedule with a linear warm up for 10% of the total training iterations. The implementation of our policy gradient algorithm also uses a value function that is initialized to be the base policy itself. The value function is only used as a baseline during RL training (REINFORCE).

Most importantly, we use a validation set to identify a good choice of α in the effective reward (in equation 5). For Gemma 2B this value is 5.0 and for the 9B policy $\alpha = 3.0$. Similar to the choice of α for test-time search, we find that most values of α in the range of 0.5 to 6.0 improved performance over ORM-RL, to different degrees. Both policies are also optimized with KL regularization, against the initial iterate of the RL policy, where the strength of the KL penalty is set to 0.001 for both.

Confidence intervals: We compute a 95% confidence interval over the true mean of the test accuracy, at each iterate of the RL training in Figure 8, Figure 15, and for each value of N in Figure 9. This mean is computed over the 500 examples in the MATH500 test set.

F THEORETICAL ANALYSIS: COMPLEMENTARY PROVERS AMPLIFY BASE POLICY IMPROVEMENT

In this section, we present the proofs for our theoretical results in the main paper (Sec. 3.4). We begin by describing some notation we use in our proofs, the natural policy gradient algorithm we use for the policy update, followed by the proofs for Theorem 3.1, and Proposition F.1. Our results in this section are in the tabular setting, with softmax parameterization of the policies. Note, that for the deterministic Markov decision process induced by the LLM, we are indeed in a tablular setting, where the set of states and actions is discrete, but large and grows exponentially in sequence length.

Notation and preliminaries. We use d_h^π, d_h^μ to denote the distribution over states $\boldsymbol{s}_h$ at time step h, starting from the initial state distribution given by the empirical distribution over the questions in the dataset $\mathcal{D}$, and following the base policy π, or prover policy μ respectively. The term $d_{\boldsymbol{s}}^\pi$ denotes the distribution over future states, starting from state $\boldsymbol{s}$, and following policy π. Here, $\boldsymbol{s}$ can be a state at any time $h \in [0, \ldots, H]$. For convenience, we overload the notation $d_{\boldsymbol{s}}^\pi$ (the distribution over future states induced by a policy starting from state $\boldsymbol{s}$), and use d_ρ^π to denote the distribution over states $d_{\boldsymbol{s}}^\pi$ starting from a random state drawn from ρ, and following policy π.

The term $Q^\pi(\boldsymbol{s}_h, a_h)$ refers to the value of state-action pair $\boldsymbol{s}_h, a_h$, *i.e.*, the expected return in the future, starting the policy from state $\boldsymbol{s}_h$ and taking action a_h:

$$Q^\pi(\boldsymbol{s}_h, a_h) := \mathbb{E}_{a_{h+1},\ldots,a_H \sim \pi(\cdot|\boldsymbol{s}_h, a_h)} \left[\text{Rex}\left((a_1, \ldots, a_H), \boldsymbol{y}_{\boldsymbol{x}}^\star \right) \right]. \tag{7}$$

Note that $\boldsymbol{y}_{\boldsymbol{x}}^\star$ is known on the dataset $\mathcal{D}$, and state $\boldsymbol{s}_h$ contains the question $\boldsymbol{x}$ as part of it. Similarly, we can define the value function $V^\pi(\boldsymbol{s}_h)$ of a state $\boldsymbol{s}_h$ as:

$$V^\pi(\boldsymbol{s}_h) := \mathbb{E}_{a_{h+1} \sim \pi(\cdot|\boldsymbol{s}_h)} Q^\pi(\boldsymbol{s}_h, a_h). \tag{8}$$

The advantage function is then given by:

$$A^\pi(\boldsymbol{s}_h, a_h) := Q^\pi(\boldsymbol{s}_h, a_h) - V^\pi(\boldsymbol{s}_h). \tag{9}$$

The policy gradient algorithm we use to update the base policy iteratively updates the base policy, and we use π_t to refer to the base policy iterate at time t. Finally, we use $\mathcal{S}$ to denote the set of all states and $\mathcal{A}$ for the set of all actions that the LLM can take at any state.

Parameterization of the base policy. We adopt the softmax parameterization for the base policy:

$$\pi_{\boldsymbol{\theta}}(a \mid \boldsymbol{s}_h) = \frac{\exp(\boldsymbol{\theta}_{\boldsymbol{s}_h,a})}{\sum_{a' \in \mathcal{A}} \exp(\boldsymbol{\theta}_{\boldsymbol{s}_h,a'})}. \tag{10}$$

Here $\boldsymbol{\theta}_{\boldsymbol{s}_h,a} \in \Theta \subseteq \mathbb{R}^d$ controls the probability of taking action a at state $\boldsymbol{s}_h$. The full set of parameters across all states and actions is denoted by $\boldsymbol{\theta} \in \mathbb{R}^{d \times |\mathcal{S}| \times |\mathcal{A}|}$. Whenever clear from context, we overload the notation π_t to denote both the policy at iterate t, *i.e.*, $\pi_{\boldsymbol{\theta}_t}$ and the parameter $\boldsymbol{\theta}_t$ itself. Similarly, the operator $\nabla_{\pi_t}[\cdot]$, is referring to $\nabla_{\boldsymbol{\theta}_t}[\cdot]$.

Defining policy improvement. Let ρ be a distribution over all states $\{\boldsymbol{s}_h : h \in [0, 1, \ldots, H]\}$, then $\mathbb{E}_{\boldsymbol{s}\sim\rho} V^\pi(\boldsymbol{s})$, and $\mathbb{E}_{\boldsymbol{s}\sim\rho} V^\mu(\boldsymbol{s})$ give us the expected value functions over states across time steps, measured under ρ, for policies π and μ respectively. We assume that d_h^π and d_h^μ are both absolutely continuous with respect to ρ, and use the expected value function over ρ as the quantity we track before and after a policy update. A positive change in $\mathbb{E}_{\boldsymbol{s}\sim\rho} V^\pi(\boldsymbol{s})$ implies a net positive improvement in the base policy. Thus, progress is made at each update of the policy when:

$$\mathbb{E}_{\boldsymbol{s}\sim\rho} V^{\pi_{t+1}}(\boldsymbol{s}) \; - \; \mathbb{E}_{\boldsymbol{s}\sim\rho} V^{\pi_t}(\boldsymbol{s}) > 0.$$

F.1 NATURAL POLICY GRADIENT

The natural policy gradient (NPG) algorithm (Kakade, 2001) defines a Fisher information matrix (induced by the policy), and performs gradient updates in the geometry induced by the following matrix:

$$F_\rho(\pi) = \mathbb{E}_{\boldsymbol{s}\sim d_\rho^\pi} \mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{s})} \left[\nabla_\pi \log \pi(a \mid \boldsymbol{s}) \left(\nabla_\pi \log \pi(a \mid \boldsymbol{s}) \right)^\top \right] \tag{11}$$

Typically, the NPG update does gradient updates on the objective ℓ_{ORM-RL}

$$\pi_{t+1} = \pi_t + \gamma \cdot F_\rho(\pi^t)^\dagger \left(\nabla_\pi \ell_{\text{PAV-RL}}(\pi) \Big|_{\pi=\pi_t} \right), \tag{12}$$

where $M^\dagger$ denotes the Moore-Penrose pseudoinverse of the matrix M. We restrict to using the initial state distribution ρ in our update rule, *i.e.*, we restrict attention to states $\boldsymbol{s}$ reachable from ρ, since ρ governs the performance measure of interest when evaluating the expected value of a policy. Thus, without loss of generality, we can exclude states that are not reachable under ρ. Specifically, we restrict the MDP to the set of states $\{\boldsymbol{s}_h \ : \ \exists \pi \ \text{ such that } \ d^\pi_\rho(\boldsymbol{s}_h) > 0, h \in [0, \ldots, H]\}$.

F.2 USEFUL LEMMAS

Lemma F.1. *[The performance difference lemma; (Kakade & Langford, 2002)] For all policies π, π' and states s_0,*

$$V^\pi(\boldsymbol{s}) - V^{\pi'}(\boldsymbol{s}) \quad = \quad \mathbb{E}_{\boldsymbol{s}_h \sim d^\pi_{\boldsymbol{s}}} \mathbb{E}_{a_h \sim \pi(\cdot|\boldsymbol{s}_h)} \left[A^{\pi'}(\boldsymbol{s}_h, a_h) \right].$$

Proof. See proof of Lemma 6.1 in Kakade & Langford (2002). □

Lemma F.2 (Natural policy gradient update)**.** *For the natural policy gradient in equation 12, the corresponding policy update is given by:*

$$\pi^{t+1}(a \mid \boldsymbol{s}) = \pi^t(a \mid \boldsymbol{s}) \cdot \frac{\exp\left(\gamma \cdot (A^t(\boldsymbol{s}, a) + \alpha \cdot A^\mu(\boldsymbol{s}, a) - \alpha \mathbb{E}_\pi A^\mu(s, a))\right)}{Z^t(\boldsymbol{s})}, \tag{13}$$

$$Z^t(\boldsymbol{s}) = \gamma \cdot \sum_{a \in \mathcal{A}} \left(A^t(\boldsymbol{s}, a) + \alpha \cdot A^\mu(\boldsymbol{s}, a) \right) \tag{14}$$

Proof. We use arguments similar to the proof of Lemma 15 in Agarwal et al. (2021), with the key difference of separately accounting for the term $A^\mu(\boldsymbol{s}, a)$. For the sake of completeness we reproduce some of the derivation, accounting for the A^μ term in the process. We follow compatible function approximation in Sutton et al. (1999) and Kakade (2001). For a vector $\boldsymbol{w} \in \mathbb{R}^{d \times |\mathcal{S}||\mathcal{A}|}$, we define the error function

$$L^\pi(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{s} \sim d^\pi_\rho}, \mathbb{E}_{a \sim \pi(\cdot|\boldsymbol{s})} \left(\boldsymbol{w}^\top \nabla_\pi \log \pi(\cdot \mid \boldsymbol{s}) - (A^\pi(\boldsymbol{s}, a) + \alpha A^\mu(\boldsymbol{s}, a)) \right)^2. \tag{15}$$

Let $\boldsymbol{w}^\star$ be the minimizer of $L^\pi(\boldsymbol{w})$ with the smallest ℓ_2 norm. Then by definition of Moore-Penrose pseudoinverse:

$$\begin{aligned} w^\star &= F_\rho(\pi)^\dagger \mathbb{E}_{s \sim d^\pi_\rho, a \sim \pi(a|s)} \left[\nabla_\pi \log \pi(a|s) \left(A^\pi(s, a) + \alpha A^\mu(\boldsymbol{s}, a) \right) \right] \\ &= F_\rho(\pi)^\dagger \nabla_\pi \ell_{\text{PAV-RL}}(\pi). \end{aligned} \tag{16}$$

In other words, $\boldsymbol{w}^\star$ is precisely proportional to the NPG update direction. Note further that for the Softmax policy parameterization, we have:

$$w^\top \nabla \log \pi(a|s) = w_{s,a} - \sum_{a' \in \mathcal{A}} w_{s,a'} \pi(a'|s).$$

Since $\sum_{a \in \mathcal{A}} \pi(a|s) A^\pi(s, a) = 0$, this immediately yields that $L^\pi(A^\pi + \alpha A^\mu) = 0$. However, this might not be the unique minimizer of L^π, which is problematic since $w^\star(\pi)$ as defined in terms of the Moore-Penrose pseudoinverse is formally the smallest norm solution to the least-squares problem, which $A^\pi + \alpha A^\mu$ may not be. However, given any vector $v \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, let us consider solutions of the form $A^\pi + \alpha A^\mu + v$. Due to the form of the derivatives of the policy for the softmax parameterization, we have for any state s, a such that s is reachable under ρ,

$$v^\top \nabla_\pi \log \pi(a|s) = \sum_{a' \in \mathcal{A}} (v_{s,a'} \mathbf{1}[a = a'] - v_{s,a'} \pi(a'|s)) = v_{s,a} - \sum_{a' \in \mathcal{A}} v_{s,a'} \pi(a'|s).$$

This is because π is a stochastic policy with $\pi(a|s) > 0$ for all actions a in each state s, so that if a state is reachable under ρ, it will also be reachable using π, and hence the zero derivative conditions apply at each reachable state. For $A^\pi + \alpha A^\mu - \mathbb{E}_\pi A^\mu + v$ to minimize L^π, we would like $v^\top \nabla_\pi \log \pi(a|s) = 0$ for all s, a so that $v_{s,a}$ is independent of the action and can be written as a constant c_s for each s by the above equality. Hence, the minimizer of $L^\pi(w)$ is determined up to a state-dependent offset, and

$$F_\rho(\theta)^\dagger \nabla_\pi \ell_{PAV-RL} \pi = A^\pi + \alpha A^\mu + v,$$

where $v_{s,a} = c_s$ for some $c_s \in \mathbb{R}$ for each state s and action a. Finally, we observe that this yields the updates

$$\pi_{t+1} = \pi^t + \gamma(A^\pi + \alpha A^\mu + v) \quad \text{and} \quad \pi_{t+1}(a|s) = \pi_t(a|s) \frac{\exp(\gamma A^t(s,a) + \gamma \alpha A^\mu(s,a))}{Z^t(s)}.$$

Owing to the normalization factor $Z^t(s)$, the state dependent offsets cancel in the updates for π, which yields the statement of the lemma. □

F.3 PROOF OF THEOREM 3.1

Proof. For some notational convenience, we use V^t, V^{t+1}, to denote the value functions $V^{\pi_t}, V^{\pi_{t+1}}$ for policies at π_t, π_{t+1} at time t and $t+1$ respectively. Similarly, we use A^t, A^{t+1} for $A^{\pi_t}, A^{\pi_{t+1}}$ respectively. For the distribution over states $d_\rho^{\pi_{t+1}}$ induced by the policy π_{t+1}, starting from an initial distribution of states given by ρ, we simplify the notation and use d_ρ^{t+1}. Similarly we use d_ρ^t for $d_\rho^{\pi_t}$.

Next, for simplicity we set $\alpha = 1$ in the natural policy gradient update in Lemma F.2. It is easy to see that the lower bound result we show holds for any value of $\alpha > 0$, and the term in the lower bound scales linearly with α. Note, that this is not a free variable, and α has to be $O(1)$, since as we increase the value of α we would have to correspondingly reduce γ for our result to hold. For now, we fix $\alpha = 1$.

From the policy difference Lemma F.1, we can write:

$$\mathbb{E}_{\boldsymbol{s}\sim\rho} V^{t+1}(\boldsymbol{s}) - \mathbb{E}_{\boldsymbol{s}\sim\rho} V^t(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{s}\sim d_\rho^{t+1}} \mathbb{E}_{a\sim\pi^{t+1}(a|\boldsymbol{s})} \left[A^t(\boldsymbol{s}, a)\right] \tag{17}$$

Next, from the natural policy gradient update in Lemma F.2, we can write $A^{t+1}(\boldsymbol{s}, a)$ as:

$$A^t(\boldsymbol{s}, a) = \frac{1}{\gamma} \cdot \log\left(\frac{\pi^{t+1}(a \mid \boldsymbol{s}) \cdot Z^t(\boldsymbol{s}, a)}{\pi^t(a \mid \boldsymbol{s})}\right) - A^\mu(\boldsymbol{s}, a) \tag{18}$$

Substituting equation 18 in equation 17 we get:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{s}\sim\rho} V^{t+1}(\boldsymbol{s}) - \mathbb{E}_{\boldsymbol{s}\sim\rho} V^t(\boldsymbol{s}) &= \frac{1}{\gamma} \mathbb{E}_{\boldsymbol{s}\sim d_\rho^{t+1}} \left[\mathrm{KL}(\pi^{t+1}(\cdot \mid \boldsymbol{s})) \| \pi^t(\cdot \mid \boldsymbol{s}))\right] \\ &\quad + \frac{1}{\gamma} \log Z^t(\boldsymbol{s}) - \mathbb{E}_{\boldsymbol{s}\sim d_\rho^{t+1}} \mathbb{E}_{a\sim\pi^{t+1}(a|\boldsymbol{s})} A^\mu(\boldsymbol{s}, a).\end{aligned} \tag{19}$$

Recall that for $\alpha = 1$,

$$\log Z^t(\boldsymbol{s}) = \log \mathbb{E}_{a\sim\pi^t(\cdot|\boldsymbol{s})} \exp\left(\gamma \cdot (A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a))\right) \tag{20}$$

Applying Jensen's inequality we get:

$$\log Z^t(\boldsymbol{s}) \geq \gamma \cdot \mathbb{E}_{a\sim\pi^t(\cdot|\boldsymbol{s})} \left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right] \tag{21}$$

$$= \gamma \cdot \mathbb{E}_{a\sim\pi^t(\cdot|\boldsymbol{s})} \left[A^\mu(\boldsymbol{s}, a)\right], \tag{22}$$

since $\mathbb{E}_{\pi^t}\left[A^t(\boldsymbol{s}, a)\right] = 0$.

Note that in equation 20 the KL term is always non-negative. Thus, we can lower bound our policy improvement as:

$$\mathbb{E}_{\boldsymbol{s}\sim\rho} V^{t+1}(\boldsymbol{s}) - \mathbb{E}_{\boldsymbol{s}\sim\rho} V^t(\boldsymbol{s}) \geq \mathbb{E}_{\boldsymbol{s}\sim d_\rho^{t+1}} \langle \pi^{t+1}(\cdot \mid \boldsymbol{s}) - \pi^t(\cdot \mid \boldsymbol{s}), A^\mu(\boldsymbol{s}, \cdot)\rangle, \tag{23}$$

where the inner product is the standard euclidean product as our actions space $\mathcal{A}$ is discrete.

In the following we will treat the distribution $\pi^{t+1}(\cdot \mid \boldsymbol{s})$ as a vector denoted by π Next, from the NPG update we know that:

$$\pi^{t+1}(a \mid \boldsymbol{s}) - \pi^t(a \mid \boldsymbol{s}) = \pi^t(a \mid \boldsymbol{s}) \left(\frac{\exp\left(\gamma A^t(\boldsymbol{s}, a) + \gamma A^\mu(\boldsymbol{s}, a)\right)}{Z^t(\boldsymbol{s})} - 1 \right) \tag{24}$$

We note that for $\gamma \ll 1$, $\exp \gamma(A^t(\boldsymbol{s}, a)) + \gamma(A^\mu(\boldsymbol{s}, a)) = \Theta(1 + \gamma(A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)))$, where the terms that grow as $\omega(\gamma)$ are being ignored. Based on this, for $\gamma \ll 1$, $\exists$ constants $0 < C_1 < C_2$ such that:

$$\exp(\gamma A^t(s, a) + \gamma A^\mu(s, a)) - 1 \in [C_1\gamma(A^t(s, a) + A^\mu(s, a)), C_2\gamma(A^t(s, a) + A^\mu(s, a))]$$

$$\begin{aligned}
\pi^{t+1}(a \mid \boldsymbol{s}) - \pi^t(a \mid \boldsymbol{s}) &\geq \pi^t(a \mid \boldsymbol{s}) \left(\frac{1 + C_1\gamma(A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a))}{1 + C_2\gamma\mathbb{E}_{a \mid \pi^t(\cdot \mid \boldsymbol{s})}\left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right]} - 1 \right) \\
&\geq C_3\gamma \frac{\left(\pi^t(a \mid \boldsymbol{s})\left(A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right) - \pi_t(a \mid \boldsymbol{s})\mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right]\right)}{1 + C_2\gamma\mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right]} \qquad (25) \\
&= C_3\gamma \frac{\left(\pi^t(a \mid \boldsymbol{s})\left(A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right) - \pi_t(a \mid \boldsymbol{s})\mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)\right]\right)}{1 + \gamma C_2\mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right],} \qquad (26)
\end{aligned}$$

where we reused: $\mathbb{E}_{\pi^t}\left[A^t(\boldsymbol{s}, a)\right] = 0$. Here, $C_3 > 0$ is a constant.

We now plug in the above lower bound into equation 21 to get the final lower bound on the policy improvement in Theorem 3.1. For this, we will once again use the assumption that the learning rate $\gamma \ll 1$, which allows us to use $1 + \gamma\mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^t(\boldsymbol{s}, a) + A^\mu(\boldsymbol{s}, a)\right] \geq C_4$ for some constant $C_4 > 0$. This is because, in our setting the range of the advantages is $[-1, 1]$.

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{s} \sim \rho}\left[V^{t+1}(\boldsymbol{s}) - V^t(\boldsymbol{s})\right] \gtrsim \gamma\mathbb{E}_{\boldsymbol{s} \sim d_\rho^{t+1}}&\left[\mathbb{E}_{a \sim \pi^t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)A^t(\boldsymbol{s}, a)\right]\right] \\
+\gamma\mathbb{E}_{\boldsymbol{s} \sim d_\rho^{t+1}}&\left[\mathbb{E}_{a \sim \pi^t(a \mid \boldsymbol{s})}\left[\left(A^\mu(\boldsymbol{s}, a)\right)^2\right]\right] \\
-\gamma\mathbb{E}_{\boldsymbol{s} \sim d_\rho^{t+1}}&\left[\left(\mathbb{E}_{a \sim \pi^t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)\right]\right)^2\right]
\end{aligned} \tag{27}$$

This gives us,

$$\mathbb{E}_{\boldsymbol{s} \sim \rho}\left[V^{t+1}(\boldsymbol{s}) - V^t(\boldsymbol{s})\right] \gtrsim \gamma\mathbb{E}_{\boldsymbol{s} \sim d_\rho^{t+1}}\left[\mathbb{V}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)\right] - \mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)A^t(\boldsymbol{s}, a)\right]\right] \tag{28}$$

Now, for the last step we note that d_ρ^{t+1} is component wise larger than ρ, and this gives us the final result:

$$\mathbb{E}_{\boldsymbol{s} \sim \rho}\left[V^{t+1}(\boldsymbol{s}) - V^t(\boldsymbol{s})\right] \gtrsim \gamma\mathbb{E}_{\boldsymbol{s} \sim \rho}\left[\mathbb{V}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)\right] - \mathbb{E}_{a \sim \pi_t(a \mid \boldsymbol{s})}\left[A^\mu(\boldsymbol{s}, a)A^t(\boldsymbol{s}, a)\right]\right] \tag{29}$$

□

F.4 DISCUSSION ON REMARK 3.1

First, we note that if the Q value of a base policy π at state, action pair (s, a) is $Q^\pi(s, a)$, then for the prover μ set to the best-of-K policy BoK(π), the Q value at the same state, action pair is:

$$Q^\mu(\boldsymbol{s}, a) = 1 - (1 - Q^\pi(\boldsymbol{s}, a))^K \tag{30}$$

This is because, in our setup the final outcome of correctness given by Rex is a random variable taking values in $\{0, 1\}$, with expectation $Q^\pi(\boldsymbol{s}, a)$, when completing a rollout starting from prefix

$(\boldsymbol{s}, a)$. Thus, we can treat the outcome as a Bernoulli. Now, we can simply compute the probability of sampling a single correct answer out of K attempts, which is also a Bernoulli random variable with mean given by $Q^\mu(\boldsymbol{s}, a)$ in equation 30.

Next, we observe that whenever $Q^\pi(\boldsymbol{s}, a) \ll 1$, e.g., when $Q^\pi(\boldsymbol{s}, a) = O(1/K)$, for all values of $(\boldsymbol{s}, a)$, we can do the taylor approximation of $Q^\mu(\boldsymbol{s}, a)$ around 0, and note that $Q^\mu(\boldsymbol{s}, a) = \Theta(K \cdot Q^\pi(\boldsymbol{s}, a))$. Next note that:

$$\begin{aligned}
\mathbb{V}_\pi A^\mu(\boldsymbol{s}, a) &= \mathbb{V}_\pi Q^\mu(\boldsymbol{s}, a) \\
&= \mathbb{V}_\pi A^\mu(\boldsymbol{s}, a) \\
&= \mathbb{V}_\pi \left[1 - (1 - Q^\pi(\boldsymbol{s}, a))^K\right] \\
&= \mathbb{V}_\pi \left[(1 - Q^\pi(\boldsymbol{s}, a))^K\right]
\end{aligned}$$

This means that the first term in Theorem 3.1 which measures "distinguishability" now increases by a factor of K^2. Similarly, we can see that the the term which measures "misalignment" can change in magnitude by a factor of $O(K)$, since the misalignment term is linear in Q^μ. These two observations combined lead us to the final remark in Remark 3.1,

F.5 IMPROVING A STRONGER BASE POLICY WITH A WEAKER PROVER POLICY

In Proposition F.1, we consider the case where the π and μ differ in performance, as measured under the distribution of states ρ in the following way:

$$\mathbb{E}_{\boldsymbol{s} \sim \rho}[|V^\mu(\boldsymbol{s}) - V^{\pi_t}(\boldsymbol{s})|] = \eta.$$

Next, whenever the prover's preference over actions is complementary to the base policy, by a factor of η, i.e.,

$$\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi}[|Q^\mu(\boldsymbol{s}, a) - Q^{\pi_t}(\boldsymbol{s}, a)|] = \Omega(\eta),$$

then the variance of A^μ or A^{π_t} under π_t should scale as η^2.

Thus, we see that when π_t fails to distinguish actions (*i.e.*, $\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{\pi_t}[A^{\pi_t}(\boldsymbol{s}, a)]$ is small) regardless of the strength of prover policy μ, as long as it is sufficiently complementary to π_t, the prover policy induces an improvement in base policy, that scales as η^2.

Proposition F.1 (Complementary μ boosts improvements in π)**.** *Under the distribution over states given by ρ, let prover μ and base policy at iterate t, π_t, differ in absolute performance, i.e., $\mathbb{E}_{\boldsymbol{s} \sim \rho}[|V^\mu(\boldsymbol{s}) - V^{\pi_t}(\boldsymbol{s})|]$. When $\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{a \sim \pi_t}[A^{\pi_t}(\boldsymbol{s}, a)] < \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{a \sim \pi_t}[A^\mu(\boldsymbol{s}, a)]$, and μ is complementary to π_t, i.e., $\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^{\pi_t}(\boldsymbol{s}, a) - Q^\mu(\boldsymbol{s}_h, a)| = \Omega(\eta)$, then $\mathbb{E}_{\boldsymbol{s} \sim \rho}[V^{\pi_{t+1}}(\boldsymbol{s}) - V^{\pi_t}(\boldsymbol{s})] \gtrsim \eta^2$.*

Proof. We begin by proving an upper bound on the disagreement between prover and base policy:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^\mu(\boldsymbol{s}, a) - Q^{\pi_t}(\boldsymbol{s}, a)| \\
&\leq \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^\mu(\boldsymbol{s}, a) - V^\mu(\boldsymbol{s}, a)| + \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^{\pi_t}(\boldsymbol{s}, a) - V^\mu(\boldsymbol{s}, a)| \\
&\leq \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^\mu(\boldsymbol{s}, a) - V^\mu(\boldsymbol{s}, a)| + \mathbb{E}_{\boldsymbol{s} \sim \rho}[|V^\mu(\boldsymbol{s}) - V^{\pi_t}(\boldsymbol{s})|] + \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^{\pi_t}(\boldsymbol{s}, a) - V^\mu(\boldsymbol{s}, a)| \\
&\leq \eta + \mathbb{E}_{\boldsymbol{s} \sim \rho}\sqrt{\mathbb{V}_{\pi_t}[A^\mu(\boldsymbol{s}, a)]} + \mathbb{E}_{\boldsymbol{s} \sim \rho}\sqrt{\mathbb{V}_{\pi_t}[A^{\pi_t}(\boldsymbol{s}, a)]},
\end{aligned}$$

where the last inequality uses Cauchy-Schwartz. Next we apply Jensen's inequality on the terms in the square root, and the conditions that $\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{E}_{a \sim \pi_t}|Q^{\pi_t}(\boldsymbol{s}, a) - Q^\mu(\boldsymbol{s}_h, a)| = \Omega(\eta)$ to conclude:

$$\sqrt{\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{\pi_t}[A^\mu(\boldsymbol{s}, a)]} \gtrsim \eta,$$

whenever $\mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{a \sim \pi_t}[A^{\pi_t}(\boldsymbol{s}, a)] < \mathbb{E}_{\boldsymbol{s} \sim \rho}\mathbb{V}_{a \sim \pi_t}[A^\mu(\boldsymbol{s}, a)]$. Invoking the result in Theorem 3.1 yields:

$$E_{\boldsymbol{s} \sim \rho}[V^{\pi_{t+1}}(\boldsymbol{s}) - V^{\pi_t}(\boldsymbol{s})] \gtrsim \eta^2$$

□

G EXAMPLES GENERATED BY BASE POLICY TRAINED ON $Q^{\pi} + \alpha Q^{\mu}$.

When we train the base policy with reinforcement learning where the reward is $Q^{\pi} + \alpha Q^{\mu}$, instead of the effective reward $Q^{\pi} + \alpha A^{\mu}$, we find that the policy training converges to contrived solutions with degenerate failures. For example, we see in multiple examples (like the four below), that for every question, the model learns to output "REPHRASE THE PROBLEM" as the first step. This is because under Q^{μ} this trivial step achieves a positive reward. While there exists a better solution where the policy outputs the correct answer and achieves higher rewards, it fails to recover from this local optima and find a better solution. Across multiple training runs, we observed similar trends when training with $Q^{\pi} + \alpha Q^{\mu}$. Note that in the idealized update, where we observe advantages A^{μ} or value function $Q^{m}u$ on all actions at a state (*e.g.*, in the NPG update in Sec. 3.4), using $Q^{\pi} + \alpha Q^{\mu}$ is equivalent to $Q^{\pi} + \alpha A^{\mu}$. But, in practice, A^{μ} offers significant benefits in the form of variance reduction in policy gradient iterates. It does that by not assigning reward to steps like “REPHRASE THE PROBLEM”, since the likelihood of a prover solving before or after this step is expected to be roughly the same (*i.e.*, the prover policy makes no progress).

Example 1

```
Q: My club has 25 members.  In how many ways can I choose members to form a 4-person
executive committee?

A:

>>>
REPHRASE THE PROBLEM
WE NEED TO FIND THE NUMBER OF WAYS TO CHOOSE 4 MEMBERS OUT OF 25 FOR THE EXECUTIVE
COMMITTEE SO WE CAN USE THE COMBINATION FORMULA WHICH IS
N C R = N! / (R! * (N - R)!) WHERE N IS THE TOTAL NUMBER OF MEMBERS AND R IS
THE NUMBER OF MEMBERS WE CHOOSE SO IN THIS CASE N = 25 AND R = 4 SO WE PLUG THEM IN AND
SIMPLIFY 25 C 4 = 25! / (4! * (25 - 4)!) = (25 * 24 * 23 * 22) / (4 * 3 * 2 * 1) = 104005
# Answer

104005
```

Example 2

```
Q: We have a triangle $\triangle ABC$ and a point $K$ on $BC$ such that $AK$ is
an altitude of $\triangle ABC$. If $AC = 10,$ $BK = 7$, and $BC = 13,$ then
what is the area of $\triangle ABC$?

A:

>>>
REPHRASE THE PROBLEM
WE NEED TO FIND THE AREA OF $\triangle ABC$ SO WE CAN USE THE FORMULA $A = \frac{1}{2}bh$
WHERE $b$ IS THE BASE AND $h$ IS THE HEIGHT SO SINCE $AK$ IS AN ALTITUDE, THAT MEANS
$AK \perp BC$ SO WE CAN USE $BC$ AS THE BASE AND $AK$ AS THE HEIGHT SO NOW WE NEED
TO FIND $AK$ SO WE CAN USE THE PYTHAGORE THEOREM ON $\triangle AKC$ SINCE $AC = 10$
# Answer

48
```

Example 3

```
Q: Suppose that $n, n+1, n+2, n+3, n+4$ are five consecutive integers.

Determine a simplified expression for the sum of these five consecutive integers.

A:

>>>
REPHRASE THE PROBLEM
WE NEED TO FIND THE SUM OF THESE FIVE CONSECUTIVE INTEGERS SO WE CAN USE THE FORMULA
$S_n = \frac{n}{2}(a_1 + a_n)$ WHERE $S_n$ IS THE SUM, $n$ IS THE NUMBER OF TERMS,
$a_1$ IS THE FIRST TERM, AND $a_n$ IS THE LAST TERM SO IN THIS CASE $n = 5, a_1 = n,
a_5 = n + 4$
# Answer

5n + 10
```

Example 4

```
Q: What is the value of $x^2+y^2-z^2+2xy$ when $x=-3$, $y=5$, and $z=-4$?

A:

>>>
REPHRASE THE PROBLEM
WE NEED TO FIND THE VALUE OF $x^2+y^2-z^2+2xy$ WHEN $x=-3$, $y=5$, AND $z=-4$
SO WE JUST PLUG IN THE VALUES AND SIMPLIFY
# Answer

-2
```

H COMPUTATIONAL COST OF TRAINING AND USING PAVS OVER ORMS

The training data for PAVs is collected by running rollouts to estimate the Q-values for each step. Since learning to predict expected future success at earlier steps is a statistically more challenging task than predicting the final score at the end of the complete sequence (as in ORM) , the training data for PAVs is larger and scales with the average number of steps in the LLM's output (≈ 10 in our case), in order to be able to achieve the same level of prediction accuracy on all steps in a generation. With a 10x larger dataset, naturally the computational cost of training PAVs (compared to ORMs which only predict final outcomes) also scales by roughly the same factor.

Despite a larger cost for training PAVs, once trained, PAVs are much more compute-efficient than ORMs on test questions. In other words, while we do incur a larger training cost, this cost is amortized over rounds of deployment. Concretely, as we show in Section 4.1, for test-time search, PAVs are 1.5-5x more compute efficient than ORMs, at achieving the same level of performance (Figure 4 in submission). This means that if we were to use a verifier at least twice upon deployment, PAV is already more compute-efficient than an ORM, accounting for both the training and test-time compute cost of PAVs.

For online RL, as we show in Section 4 (Figure 8c), PAVs achieve the same level performance in $1/6^{\text{th}}$ of the RL training iterations that ORMs require. But in our implementation, for each iteration, PAVs score each step by feeding each prefix in a generation separately through the trained model, and ORMs only score the full trajectory. Thus, in this implementation PAVs consume 10x more compute per batch to score the generations in a batch (since we have 10 steps on average per generation). A more efficient implementation would score all prefixes in a generation in a single forward pass through the trained PAV. Nevertheless, the reduction in RL iteration complexity with PAVs is big enough that the overall computational cost is less, compared to ORMs. For example, using the formula for training and inference FLOPs in Hoffmann et al. (2022), during online RL, to train the Gemma 2B base policy with PAVs, we need to spend about 2.5×10^{18} FLOPs, but to achieve the same performance with ORMs, we need about 5.9×10^{18} FLOPs, resulting in at least half the total computational FLOPs.

I CONNECTING THE EFFECTIVE REWARDS TO REWARD-SHAPING POTENTIAL FUNCTIONS

The effective reward we define in equation 5 matches the definition of a potential-based shaped reward, as defined in Ng et al. (1999). Before we explain why this is the case, we provide some background below.

Background on potential functions: In Ng et al. (1999), instead of learning a policy for a reward function R, the policy is trained to optimize the transformed reward $R + F$. Here, $F(s_h, a_h, s_{h+1})$ is a reward shaping function, that takes as input the the current state s_h, action taken by the policy a_h, and next state s_{h+1}, and maps this to a scalar reward. In particular, they show that when the shaping function F is potential-based, i.e., it is of the form:

$$F(s_h, a_h, s_{h+1}) = \Phi(s_{h+1}) - \Phi(s_h),$$

for some state-dependent potential function Φ, then the policy that optimizes the transformed reward $R + F$, also optimizes the original reward F.

In Equation 5, our effective reward function is: $Q^{\pi}(\boldsymbol{s}_h, a_h) + \alpha A^{\mu}(\boldsymbol{s}_h, a_h)$. This reward function matches the functional form of the transformed reward function $R + F$ defined in Ng. et. al. (1999). Here, Q^{π} corresponds to the reward R, and $\alpha A^{\mu}(\boldsymbol{s}_h, a_h)$ is the potential-based reward function F. To see why F satisfies this definition we use the definition of advantages from Equation 2. We can write $\alpha A^{\mu}(\boldsymbol{s}_h, a_h)$ as:

$$\alpha(Q^{\mu}(\boldsymbol{s}_h, a_h) - V^{\mu}(\boldsymbol{s}_h)) = \alpha(V^{\mu}(\boldsymbol{s}_{h+1}) - V^{\mu}(\boldsymbol{s}_h)).$$

Following the theoretical result on potential-based reward shaping functions in Ng et al. (1999), the optimal policy under our effective reward $Q^{\pi} + \alpha A^{\mu}$ also lies in the set of optimal policies which only optimize the outcome reward (final answer correctness, or Q^{π}).

J DISCUSSION, LIMITATIONS AND FUTURE WORK

We began our exposition with the following question: how to define process rewards such that optimizing the base policy against process rewards ultimately improves the outcome level correctness of the final answer? Our key finding is that process rewards defined as advantages of a prover policy, distinct from the base policy improve the efficiency of exploration for steps sampled from the base policy during test-time search and online RL. This improved exploration in turn leads to discovery of better solutions, resulting in a higher accuracy on the math reasoning task. We also formally characterized the set of good prover policies as policies with step-level advantages that meaningfully contrast steps generated by the base policy, while still producing step-level advantages that are aligned with the base policy. Having trained process advantage verifiers (PAVs) to predict advantages under the prover policy, we empirically observed that test-time search against the trained PAVs improve the compute-efficiency of search by $1.5 - 5\times$, and accuracy of search by over 8% compared to running best-of-N against an ORM. Next, we present one of the significant results that validate the use of dense supervision when optimizing the base policy with online RL. Specifically, we show that dense online RL with rewards from our trained PAVs, improves sample efficiency of online RL by $5 - 6\times$, and results in an accuracy gain of over 6%.

Limitations. Despite the promise of our results, there are several limitations to our work that present important avenues for future research. First, while we can easily compute the right hand side of our result in Theorem 3.1 to understand whether a given prover policy will improve a fixed base policy, it is unclear how to automatically design a flexible class of optimal (or very good) prover policies for a sequence of base policy iterates. Perhaps simultaneously optimizing the prover and the base policy (in a two-player game) might provide for an approach to obtain the best prover during RL, but this is largely an open question. Second, since inevitably learning a process advantage verifier (PAV) will incur fitting errors and this upper bounds peformance of our method. Fitting errors can be circumvented if our approach if we can simply run rollouts from prover policies during online RL or search to estimate advantages without training verifiers, and extending our approach to this setup is a good avenue for future work.