Good Experience Maximization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Post-training language models typically follows one of two paradigms: reinforcement learning (RL) and supervised fine-tuning (SFT). Although typically thought of as different camps, we show that both paradigms can be viewed as target distribution matching. We use this perspective to analyze the tension between policy gradient and buffer-based SFT and establish that a good post-training target should preserve past successes while continuing to incorporate newly-discovered ones. We introduce Good Experience Maximization (GEM), a simple framework that resolves this tension by training the policy toward a mixture of historical and newly generated successes. This construction is amenable to a variety of simple algorithmic choices, is agnostic to the sampling process that populates the buffer and naturally subsumes several existing post-training methods as special cases. We prove that the GEM target yields globally optimal expected reward at convergence under realizability, and we derive a flow-based update that guarantees one-step policy improvement. This is a distinctively different update to SFT and relies on constructing a straight line "path" toward the target distribution. Empirically both the "flow" and the cross entropy version of GEM perform on par with leading post-training methods and out-perform it in sparse reward settings.

1 Introduction

Good experiences are worth repeating. The brain seems to operate on this principle: after salient experiences, it spontaneously reactivates these memories offline, prioritizing those tied to reward, novelty, or uncertainty [Mattar and Daw, 2018]. This replay has many benefits; recent evidence shows it supports important cognitive tasks, including memory consolidation, planning, and decision-making [Momennejad et al., 2018, Schapiro et al., 2018]. Remarkably, this mechanism enables a single meaningful experience to continue shaping future behavior without requiring us to physically reenact it. Machine learning has a direct analog of this idea: *experience replay* [Lin, 1992, Mnih et al., 2013]. Rather than being discarded after a single use, past interactions are stored in a replay buffer and reused during training, which enables experiences to inform many gradient updates and amortizes the cost of data collection. This simple mechanism has supported major empirical successes across challenging decision-making domains [Andrychowicz et al., 2017, Vinyals et al., 2019].

Despite these advantages, current methods for post-training language models (LMs) are dominated by methods that require online samples at every step [Williams, 1992, Shao et al., 2024, Tajwar et al., 2026]. These on-policy methods generate K fresh completions, or rollouts, per problem $\mathbf{x}$ from the current LM (or policy) π_θ, compute a gradient update, and discard the data. However, this process is wasteful: a correct reasoning trace discovered at a particular training step may remain a useful signal long after the LM has been updated. To continue improving, the model must rediscover solutions it has already found, which is particularly problematic when solutions are rare. Furthermore, discarding these solutions risks *forgetting*: if the model is no longer trained on traces that solved a given problem, it may lose the ability to solve it [Yue et al., 2025]. An ideal training regime would resemble human memory: hold onto what worked, and keep learning from it.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

Reuse alone has its own pathology: a learner that solely revisits past successes risks never discovering new ones. Indeed, a learner trained only on historical successes is constrained by the coverage of the data and performance can degrade if the policy moves beyond that distribution [Ross et al., 2011]. But relying on the policy itself as the only memory of past successes is also insufficient. If a solution is not externally stored, then recovering it requires sampling it again from the model, which is unlikely for rare reasoning traces. In principle, one may try to preserve past behavior through regularization or value function learning, but extracting solutions with them still requires search or sampling. An effective post-training regime therefore requires both memory and exploration: the LM ought to continue learning from found successes while still producing new ones worth remembering.

Contributions. This paper studies how to realize this principle in post-training. We introduce *Good Experience Maximization* (GEM), a framework that incorporates memory through a replay buffer of past successes and exploration by collecting online data at every step. We first formalize post-training as target distribution matching and use this lens to study the failure cases of policy gradient and buffer-based SFT (Section 3). We then use this lens to develop GEM, a simple post-training framework that trains the policy on a mixture of historical and newly generated successes (Section 4). We prove that this mixture target yields global expected reward at convergence and develop a novel flow-based update that establishes improvement at each update step. Empirically, we provide a practical algorithm and show that GEM can match or exceed RL algorithms in terms of performance and sample efficiency (Section 5) In Section 6, we demonstrate how this framework subsumes a broad set of existing post-training methods as special cases, including MaxRL [Tajwar et al., 2026] and Iterated STaR [Zelikman et al., 2022].

2 Background and Preliminaries

Problem setup. We study correctness-based problems, which can be abstracted as binary successes or failures for each input $\mathbf{x}$. This setting is known as reinforcement learning with verifiable rewards (RLVR) [Lambert et al., 2024, Su et al., 2025] and encompasses a wide set of modern problems. In this setting, $\mathcal{X}, \mathcal{Y}$ denotes the input and output spaces, respectively, and we are given query access to a ground-truth reward function $R : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. The goal is to maximize the expected reward of some parameterized policy $\pi_\theta : \mathcal{X} \to \mathcal{Y}$ given some distribution $\mathbf{x} \sim \mu$,

$$\mathcal{J}(\pi_\theta) = \mathbb{E}_{\mathbf{x}\sim\mu\,,\,\mathbf{y}\sim\pi_\theta(\cdot|\mathbf{x})}\left[R(\mathbf{x},\mathbf{y})\right].$$

In many modern settings, the model does not sample directly from $\mathcal{Y}$; instead, it may generate a trace z, which is deterministically parsed into a reasoning trace $\mathbf{r}$ and $\mathbf{y}$. We can treat each $\mathbf{r}$ as a latent variable that is marginalized out to obtain the answer distribution $\pi_\theta(\mathbf{y} \mid \mathbf{x}) = \sum_{\mathbf{r}} \pi_\theta(\mathbf{r} \mid \mathbf{x})\pi_\theta(\mathbf{y} \mid \mathbf{x}, \mathbf{r})$.

Policy optimization. There are two dominant paradigms for post-training generative models. One of these is online policy-gradient methods [Williams, 1992, Schulman et al., 2017, Guo et al., 2025], which estimate gradients of the policy optimization objective as

$$\nabla_\theta \mathcal{J}(\theta) = \mathop{\mathbb{E}}_{\mathbf{x},\mathbf{y}}[A(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y} \mid \mathbf{x})], \tag{1}$$

where $A(\mathbf{x}, \mathbf{y})$ denotes some advantage estimate. Another, related perspective instead maximizes a lower bound on the probability of success [Dayan and Hinton, 1997, Abdolmaleki et al., 2018a,b, 2024, Tajwar et al., 2026]. The other paradigm follows the simple principle of "sample, filter, then SFT". In practice, methods typically sample many answers across a large dataset, filter for the set of good answers, then maximize the likelihood of this filtered dataset [Brandfonbrener et al., 2021, Zelikman et al., 2022, Dong et al., 2023, Gulcehre et al., 2023, Singh et al., 2023, Havrilla et al., 2024]. Given a buffer,

$$\mathcal{B} := \{(\mathbf{x},\mathbf{y}) : \mathbf{y} \sim \pi(\cdot \mid \mathbf{x}),\ R^*(\mathbf{x},\mathbf{y}) = 1\},$$

the model is trained with the supervised objective:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) := -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{\mathcal{B}}}\left[\log \pi_\theta(\mathbf{y} \mid \mathbf{x})\right], \tag{2}$$

with the gradient:

$$\nabla_\theta \mathcal{L}_{\mathrm{SFT}}(\theta) := -\mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim P_{\mathcal{B}}}\left[\nabla_\theta \log \pi_\theta(\mathbf{y} \mid \mathbf{x})\right]. \tag{3}$$

Equations (1) and (3) provide the objects we analyze in this work. Although policy gradient and filtered SFT differ operationally, both gradients are expectations of $\nabla_\theta \log \pi_\theta(y \mid x)$ under some distribution over prompt-answer pairs. We use this shared form in Section 6 to study post-training as a problem of target distribution design.

3 A Target-Matching View

While the community has explored a wide range of policy optimization approaches (Section 2), we observe that many can be viewed as iterative update procedures that move a policy toward a target distribution. We refer to this formulation colloquially as the "SFT update". Specifically, in the binary reward case, regular policy gradient [Williams, 1992], expectation-maximization based methods that maximize the likelihood of success [Dayan, 1993, Abdolmaleki et al., 2018b, Tajwar et al., 2026], as well as filtered SFT methods [Brandfonbrener et al., 2021, Dong et al., 2023, Gulcehre et al., 2023, Havrilla et al., 2024] can all be viewed as minimizing a per-iteration SFT loss toward the target distribution at each input $\mathbf{x}$,

$$q_t(\mathbf{y}|\mathbf{x}) = \frac{\pi_t(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})}{\mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})]}, \tag{4}$$

We expand on this connection in Section 6. Intuitively, these methods all follow a gradient of the log probability, and we can absorb any scaling to the per-sample log probability into the probability of the sampling distribution. This target-matching view exposes a tension between two desirable properties, which we will now discuss.

3.1 Policy Gradient Creates Good Experiences But Does Not (Explicitly) Retain Them

Policy gradient samples small batches online and typically perform single updates. A successful trace sampled at iteration t affects the policy only through the update taken at that iteration. If the trace is not externally stored, then for it to affect future updates, the policy must sample it again. Thus, the probability that $\mathbf{y}$ contributes repeatedly to learning remains proportional to the mass the current policy places on it $\pi_t(\mathbf{y} \mid \mathbf{x})$. Thus, for any good solutions to be repeatedly reinforced, the policy must repeatedly sample them, making learning slow and sample-inefficient.

Yet, the single update is also the benefit of policy gradient algorithms. Because each update is based on fresh samples from the current policy, policy gradient continues enjoys full support over problem distribution. Moreover, a successful generation can produce an immediate update: in the binary reward setting, even a single high-reward sample can increase the likelihood of the behavior that produced it. Thus, learning can begin before the learner has collected successful outputs for every input. With function approximation, this update may also generalize across related inputs, allowing progress on questions that have not yet been solved directly.

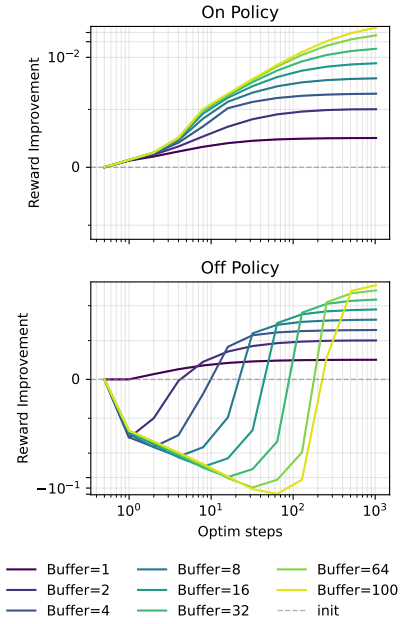


Figure 1: SFT training dynamics in a toy categorical task. Figures illustrate SFT updates toward a buffer distribution with (i) high reward samples filtered from the current policy, or (ii) high reward samples from a very different policy. Colors indicate the state space coverage of the buffer.

3.2 Iterative SFT Retains Good Experiences but Does Not Guarantee Improvement

Alternatively, the canonical class of "filtered SFT" algorithms [Zelikman et al., 2022, Brandfonbrener et al., 2021, Dong et al., 2023, Gulcehre et al., 2023, Havrilla et al., 2024] typically sample (nearly) the full dataset, then do multiple gradient steps on the filtered dataset. As we establish above and in Section 6, a single step of this procedure is in expectation *equivalent* to a single step of online policy gradient (or an online variant such as EM or MaxRL, depending on if we sample by question first, or by correct answers). The distinct advantage of iterative SFT is the ability to reuse successful generations across multiple gradient updates. Filtered SFT therefore has two appealing properties: in theory, it can (i) perform multiple updates, and (ii) make use of multiple sources of "good" data. But when does fitting an accumulated buffer actually improve the policy?

One limitation is coverage. If we train to *exactly* match only the buffer distribution $P_{\mathcal{B}}$, we are limited by the support of $\mathcal{B}$. We can write down a simple bound for the amount of reward we can guarantee *on the training set* $\mathcal{D}$ where $\mathcal{B}$ provides no signal.

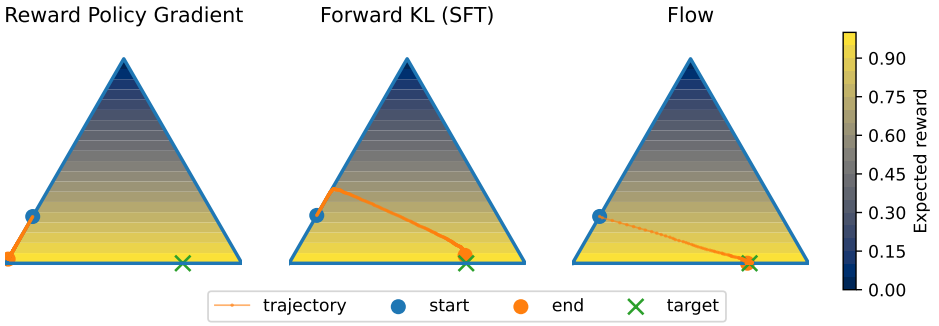


Figure 2: Geometry of optimization with on and off policy data on a 3 categories simplex. Example of reward policy gradient, forward KL (SFT) toward a target distribution, and our flow update.

Remark 3.1. Suppose we have binary rewards, training dataset $\mathcal{D}$, and a buffer distribution $P_{\mathcal{B}}$ whose support is limited to $M < |\mathcal{D}|$ questions. If a policy π^* perfectly approximates $P_{\mathcal{B}}$, the minimum expected reward it achieves $\mathcal{J}(\pi^*) = \frac{M}{|\mathcal{D}|}$. For the remaining $|\mathcal{D}| - M$ questions, the objective is undefined, and $\pi^*(\cdot|\mathbf{x})$ may behave arbitrarily poorly.

The above is reflected in Figure 1: the *eventual* performance grows monotonically with the coverage of the buffer over the question set. Thus, with a sufficiently broad buffer and sufficient training, filtered SFT can achieve high training-set reward. The difficulty is that obtaining such coverage may require sampling nearly the full dataset before substantial learning can occur.

Another limitation is monotonic improvement. To avoid repeatedly sampling every input, we may want to reuse data from older policies or other sources. As shown in Figure 1, if the buffer data is good but "off policy" (in the sense that it has low likelihood under the current policy), we can end up with optimization dynamics where in a small number of steps we will end up with *lower rewards*, despite the buffer having higher expected rewards. The issue is not that the buffered samples are bad; rather, a standard SFT path toward an arbitrary buffer distribution is not necessarily an improvement-preserving path for the current policy (Figure 2).

Thus, iterative SFT addresses the memory problem but leaves a core tension. Broad buffer coverage is needed for high dataset-level performance, but obtaining that coverage through fresh sampling can be expensive. Re-using data from old policy samples or other sources is the reasonable choice reduces this cost, but, in the worst cases it can lead to reward degradation. In the next section, we will aim to address this tension.

4 From Target Distribution to Algorithm

The preceding discussion suggests a simple design principle: we should retain a buffer of past successes, but avoid updating toward the buffer alone. Instead, the target distribution should involve the buffer where it provides high-reward signal and preserve the current policy elsewhere. We therefore propose to combine buffered successes with online samples from the current policy, then update toward this target through a new update that yields a one-step improvement guarantee.

Our algorithm, called Good Experience Maximization (GEM) is conceptually simple and consists of two main steps, detailed in Algorithm 1. First, it constructs P_M from the current buffer and the current policy (*Generate Good Experiences*). Second, rather than taking an unconstrained SFT step toward π_M, it follows a distributional flow from π_t to π_M (*Maximize Them*). This flow preserves the support benefits of the mixture target while allowing us to prove one-step improvement. We discuss each of these in turn.

4.1 Generate Good Experiences

We introduce a mixture distribution with weights $0 < \alpha < 1$,

$$P_M(\mathbf{x}, \mathbf{y}) = \alpha\, P_{\mathcal{B}}(\mathbf{x}, \mathbf{y}) + (1 - \alpha)\, \mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})\,, \tag{6}$$

which corresponds to line 5 in Algorithm 1. Intuitively, the first term concentrates mass on $(\mathbf{x}, \mathbf{y})$ pairs that are high reward; the second term ensures that π_t covers $\mathbf{x}$ outside of the support of $\mathcal{B}$.

Algorithm 1 GEM

Base LLM π_1, buffer $\mathcal{B} \leftarrow \{\emptyset\}$
for policy iteration $t = 1, \ldots, T$ **do**
  Initialize training policy $\pi_\theta \leftarrow \pi_t$
  while not converged **do**
    Sample from the mixture $P_M(\mathbf{x}, \mathbf{y}) = \alpha P_{\mathcal{B}}(\mathbf{x}, \mathbf{y}) + (1 - \alpha)\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})$
    Minimize divergence to mixture, $D_{\text{KL}}(P_M||\pi_\theta)$:

$$\hat{l}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_M}\Big[- \log \pi_\theta(\mathbf{y}|\mathbf{x})\Big] \tag{5}$$

    Annotate new policy samples with verifier $r_i = R(\mathbf{x}_i, \mathbf{y}_i)$
    Add good samples to buffer: $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\mathbf{x}_i, \mathbf{y}_i, r_i, t)\}_{i=1}^n$
  end while
  $\pi_{t+1} \leftarrow \pi_\theta$
end for

Define the conditional mixture policy as $\pi_M(\mathbf{y} \mid \mathbf{x}) = P_M(\mathbf{x}, \mathbf{y})\mu_M(\mathbf{x})$, where μ_M is the marginal of the mixture. Let $S_{\mathcal{B}}\{\mathbf{x} \in \mathcal{X} : \mu_{\mathcal{B}} > 0\}$ denote the support of the buffer's marginal distribution.

The mixture target reward is pointwise non-decreasing. We will first show that this construction guarantees that expected reward does not decrease for any individual $\mathbf{x}$.

Lemma 4.1. *Suppose that, $\forall \mathbf{x} \in S_{\mathcal{B}}$, $\pi_{\mathcal{B}}$ outperforms π_t, such that: $\mathbb{E}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x}, \mathbf{y})] > \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x}, \mathbf{y})]$. Then, for π_M, for $\mathbf{x} \in S_{\mathcal{B}}$, $\mathbb{E}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x}, \mathbf{y})] > \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x}, \mathbf{y})]$. And, for $\mathbf{x} \notin S_{\mathcal{B}}$, $\mathbb{E}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x}, \mathbf{y})] = \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x}, \mathbf{y})]$. Therefore, $\mathbb{E}_{\pi_M}[R(\mathbf{x}, \mathbf{y})] \geq \mathbb{E}_{\pi_t}[(\mathbf{x}, \mathbf{y})]$.*

Proof sketch. For any $\mathbf{x}$, π_M is a convex combination of $\pi_{\mathcal{B}}$ and π_t, weighted by the marginals. For $\mathbf{x} \in S_{\mathcal{B}}$ the reward is the weighted average of $\mathbb{E}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x}, \mathbf{y})]$ and $\mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x}, \mathbf{y})]$. The former is strictly greater by construction, so the average exceeds the latter. For $\mathbf{x} \notin S_{\mathcal{B}}$, π_M reduces to π_t. See Appendix A for details. □

Pointwise non-decreasing reward implies global improvement. We now can consider the total expected reward across the *entire state distribution* $\mu(\mathbf{x})$. If a training algorithm perfectly matches the target conditional distribution, the new policy will converge to $\pi_M(\mathbf{y}|\mathbf{x})$. By integrating the pointwise improvement over $\mu(\mathbf{x})$, we show that this converged policy strictly improves global expected reward:

Lemma 4.2. *Let $P_{\mathcal{B}} = \mu_{\mathcal{B}}(\mathbf{x})\pi_{\mathcal{B}}(\mathbf{y} \mid \mathbf{x})$. Assume $P_{\mathbf{x}\sim\mu}(\mathbf{x} \in S_{\mathcal{B}}) > 0$. Given the pointwise conditions from Lemma 4.1, the converged mixture policy π_M strictly outperforms the current policy π_t: Then, for any $\alpha \in (0, 1)$,*

$$\mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim P_M}\big[R(\mathbf{x}, \mathbf{y})\big] > \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu,\mathbf{y}\sim\pi_{\theta_t}}\big[R(\mathbf{x}, \mathbf{y})\big]. \tag{7}$$

Proof sketch. The expected reward under P_M can be decomposed by linearity as $\mathcal{J}(P_M) = \alpha\,\mathbb{E}_{P_{\mathcal{B}}}[R] + (1 - \alpha)\mathcal{J}(\pi_{\theta_t})$. By the assumption that $\mathcal{B}$ outperforms the π_{θ_t} on its own support, $\mathbb{E}_{P_{\mathcal{B}}}[R] > \mathbb{E}_{\mathbf{x}\sim\mu,\mathbf{y}\sim\pi_{\theta_t}}[R]$. Since $\alpha > 0$, we get a strict inequality. See Appendix A for details. □

4.2 Maximize Them

Theorem 4.2 guarantees that P_M has higher reward than π_t. To translate this statement into improvement of π_{t+1} itself, we need π_{t+1} to be close to P_M, which holds only at convergence. With finite gradient steps, π_{t+1} may not yet match P_M, and individual steps along the way can decrease expected reward (Figure 1). In other words, Algorithm 1 guarantees per-iteration improvement only when *each inner loop runs to convergence*; per-update improvement is not guaranteed. We desire an update that preserves step-wise improvement toward P_M.

Algorithm 2 `FlowUpdate`$(\mathcal{X}, \mathcal{Y}, \pi_\theta, m)$

1: Compute probabilities $p = \pi_\theta(\cdot \mid \mathcal{X})$
2: Compute probability velocity $\dot{p} = m - p$
3: Map to parameter space $L = \dot{p} \oslash p$
4: **return** L

Gradient descent can leave the mixture path. Observe that we can specify a path in distribution space from the current policy to the mixture target. For convenience of notation, we write the current policy $p = \pi_\theta$ and the target distribution supported on correct answers as b. Rewriting Equation (6), note that this defines a convex combination of the two distributions in probability space:

$$m(\alpha) = (1 - \alpha)p + \alpha b, \alpha \in [0, 1].$$

Moving along this path corresponds to a straight line in the probability simplex (an m-geodesic [Amari, 2016]). However, although b represents the ideal destination on the probability simplex, standard gradient descent in parameter space does not necessarily recover this path. Instead, standard gradient steps inherently follow trajectories dictated by the parameter geometry, which can violate the guarantee of monotonic performance improvement [Kakade, 2001] (which we observe in Figures 1 and 2).

A probability-space flow recovers the path. To restore this guarantee, we explicitly force the policy to move along the straight line path by constructing a continuous-time flow (Algorithm 2). Note that the ideal velocity in probability space along this path is simply the vector pointing from the current policy to the target distribution: $\dot{p} = \alpha - p$. To realize the velocity $\dot{p}$ in probability space through continuous steps in parameter space, we use the chain rule to derive the corresponding parameter velocity $\dot{\theta}$. Because the expected reward is a linear function of the output probabilities, it strictly interpolates between the performance of the current policy and the target policy when moving in a straight line toward a more performant target distribution.

With FlowUpdate, GEM theoretically improves performance at every step. The advantage of this framing is it guarantees performance gains at *every step*, which we will now show.

Proposition 4.3. *Let $\mathcal{J}(\theta)$ be the expected reward of a policy parameterized by its logits θ. Assume $\dot{\theta}$ is defined component-wise as $\dot{\theta}_i := \frac{\dot{p}_i}{p_i} - \frac{\dot{p}_v}{p_v}$, which realizes $\dot{p} = m - p$. If m has non-negative expected advantage, $\sum_j m_j A_j \geq 0$, then: $\frac{d\mathcal{J}}{dt} \geq 0$.*

Proof sketch. We compute the time derivative of performance via the chain rule: $\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{v-1} \frac{\partial \mathcal{J}}{\partial \theta_i} \dot{\theta}_i$. Substituting the policy gradient $\frac{\partial \mathcal{J}}{\partial \theta_i} = p_i A_i$ and our explicit update, we get $\sum_{i=1}^{v-1} p_i A_i \left(\frac{\dot{p}_i}{p_i} - \frac{\dot{p}_v}{p_v} \right)$, which simplifies to $\sum_{i=1}^{v} A_i \dot{p}_i$. Plugging in the trajectory $\dot{p}_i = m_i - p_i$, we get $\frac{d\mathcal{J}}{dt} = \sum_i (m_i - p_i) A_i$. Since the expected advantage of the current policy is always 0, the derivative exactly equals $\sum_i m_i A_i$, which is non-negative. Therefore, $\frac{d\mathcal{J}}{dt} \geq 0$. We provide the full derivation in Section A. □

4.3 Practical Desiderata

Given the generality of GEM, we now discuss a few key design choices: what to retain, how to sample, and how to amortize online generation.

What to retain. In the binary-reward setting, retention is straightforward. Storing correct generations automatically satisfies the condition in Theorem 4.2 because $\mathbb{E}_{\mathcal{B}}[R] = 1$, whereas $E_{\pi_t}[R] \leq 1$. For a fixed capacity buffer, a natural eviction rule is FIFO, which discards the most stale data first. However, more complex eviction rules may prioritize maintaining diversity or difficulty over $\mathcal{X}$, the latter of which would likely better maintain the support condition (especially if the buffer capacity is small relative to the number of questions in the training dataset).

How to sample. $P_{\mathcal{B}}$ does not require a specific sampling scheme beyond coverage, but different choices can have benefit in practice. The simplest approach is to train on uniformly sampled entries; however, uniform sampling introduces two sources of bias. First, samples that enter the buffer earlier are disproportionately sampled. Thus, a simple fix is to sample with a recency bias according to an exponentially decaying weight. Second, easy problems (high $\pi_\theta(\mathcal{A}^*(\mathbf{x}) \mid \mathbf{x})$) contribute many more correct entries to the buffer than hard problems. As a result, training is dominated by examples the

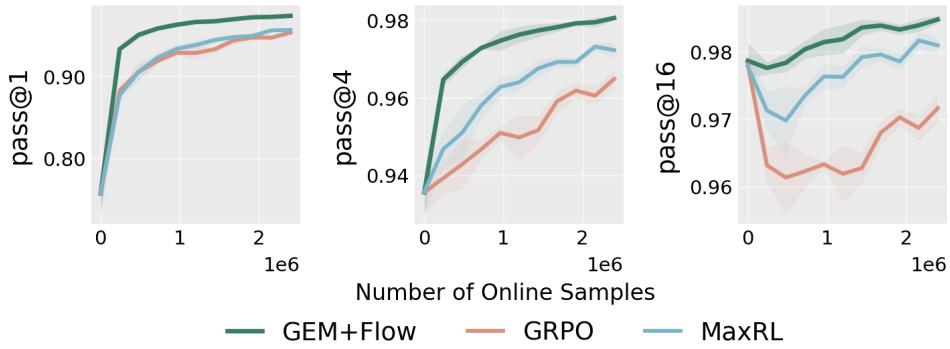


Figure 3: Number of online samples (generations) vs. pass@k for GEM+Flow, GRPO [Shao et al., 2024], and MaxRL [Tajwar et al., 2026] on the MNIST bandit task. Online samples denote newly generated model outputs used for training, excluding reused buffer samples. Curves show mean performance across 3 seeds, with shaded regions indicating standard deviation. GEM+Flow reaches higher pass@k with fewer online generations than GRPO or MaxRL, demonstrating improved sample efficiency.

model has already solved, providing little learning signal. We address this issue with *inverse-pass-rate* $(1/p)$ sampling. For each problem, define the empirical pass rate from the buffer as $\hat{p}_{\mathbf{x}} = \frac{c_{\mathbf{x}}}{n_{\mathbf{x}}}$, where $n_{\mathbf{x}}$ is the total number of entries for $\mathbf{x}$ and $c_{\mathbf{x}}$ is the number of correct ones. The sampling weight is $w(\mathbf{x}) = \frac{1}{\hat{p}_{\mathbf{x}}}$. Under these weights, the expected number of training samples from each prompt is equalized regardless of difficulty. [1]

How to amortize online generation. Each iteration of Algorithm 1 requires some number of online rollouts from π_θ (line 5). Because generation is costly, amortizing it across multiple steps is desirable. One approach is to occasionally skip generating online samples and update purely using a static buffer. This version holds given the buffer-reward condition is maintained, which occurs when the new policy is updated with a small learning rate or is updated using a trust region. Figure 3 shows how GEM can make use of additional updates on a ViT on MNIST bandit task. Empirically, Figure 3 shows that GEM improves pass@1 and pass@4 monotonically while avoiding the larger early pass@16 degradation of GRPO and MaxRL. Moreover, GEM reaches a given pass@k level with substantially fewer newly-collected samples, indicating improved sample efficiency.

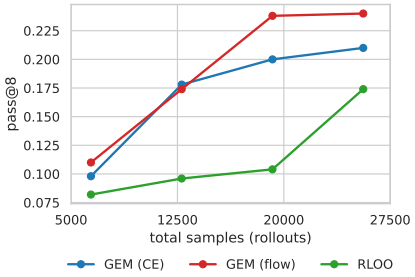
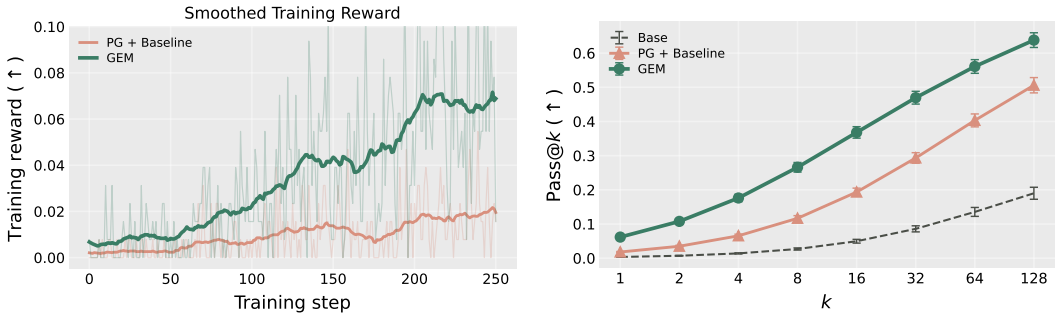


Figure 4: Eval Pass@8 of Llama on MATH. GEM achieves fast improvements, with the flow update providing slightly better learning

5 A Practical Algorithm

We introduced the flow update due to the phenomenon that standard SFT can curve away from the straight-line mixture path in probability space. However, the current policy may actually lie in a benign region of the simplex, such as in the center of those shown in Figure 2. In these cases, the SFT update may approximate the flow update well, moving more directly to the target distribution without reward degradation. It is difficult to know a priori which regime will hold during LM post-training, but, if the mixture target can often be optimized with ordinary SFT, then GEM can inherit the practical advantages of existing supervised-training pipelines. We therefore test a practical SFT version of GEM that uses the same online replay construction but replaces the flow update with standard cross-entropy training. Critically, this variant does not train on a fixed dataset: it repeatedly generates new completions and trains on the evolving replay distribution.

[1] This in fact recovers recent works [Tajwar et al., 2026] in the setting where the buffer only maintains storage of the most recent policy's generations.



(a) Training time accuracy

(b) Evaluation pass@k on MATH500

Figure 5: *GEM* matches or excels performance of policy gradient (PG with mean reward baseline) and base model (Llama 3.2 3B Base), particularly excelling when rewards are difficult to find, demonstrating its ability to make use of previously seen high reward generations (on MATH, while the base model has nearly zero pass@1 rates).

A central motivation for GEM is that policy-gradient training can waste rare successes: once a high-reward output is sampled, the update may not preserve the specific solution for future learning. In the dense-reward setting, many outputs provide a useful signal for learning; in the sparse-reward setting, successful outputs are rare. We therefore train on MATH [Hendrycks et al., 2021], and evaluate using MATH500 [Lightman et al., 2023]. We first validate that both versions of GEM can indeed provide performance and efficiency gains on LM post-training. We then investigate the practical version of GEM. For these experiments, we use Llama-3.2-3B Base [Grattafiori et al., 2024]. For additional experimental details, see Appendix B.3.

Both variants can provide reward improvement on LMs. We first demonstrate that GEM with and without the flow update can outperform a standard policy gradient post-training method (RLOO) using Figure 4 demonstrates that GEM rapidly improves, with the flow update providing marginally better learning.

More broadly, GEM without the flow update outperforms RL. We now turn to the practical algorithm and investigate its performance. Figure 5 shows that our algorithm outperforms the standard policy gradient in sparse reward settings. These results support that maintaining an explicit memory of past successes is particularly beneficial in such "hard discovery" settings.

6 A Unifying View: Policy Optimization is Iterative SFT

In this section we expand on the note that all existing work around policy optimization for large generative models, which operate in the contextual bandit setting, is more simply viewed through the lens of iterative minimization of a forward KL divergence (i.e. "SFT update") towards *some* target distribution P,

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}\big[-\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\big]\,.$$

To see this, we note that for any positive reward the policy gradient can be written as,

$$\begin{aligned}
\nabla_\theta \mathcal{J}_{\text{PG}}(\pi_\theta) &= \mathbb{E}_{\mu(\mathbf{x})}\Big[\mathbb{E}_{\pi_\theta(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\big]\Big] \\
&= \mathbb{E}_{\mu(\mathbf{x})}\left[\int \pi_\theta(\mathbf{y}|\mathbf{x}) R(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\right], \\
&= \mathbb{E}_{\mu(\mathbf{x})}\left[v(\mathbf{x},\pi_\theta)\int \frac{\pi_\theta(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_\theta)}\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\right], \\
&= \mathbb{E}_{\mu(\mathbf{x})}\Big[v(\mathbf{x},\pi_\theta)\,\mathbb{E}_{q_\theta(\mathbf{y}|\mathbf{x})}\Big[\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\Big]\Big],
\end{aligned}$$

where $v(\mathbf{x},\pi_\theta) = \mathbb{E}_{\pi_\theta(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})]$ is the expected reward of π_θ given input $\mathbf{x}$. The above is equivalent to a per-question SFT update toward the target,

$$q_\theta(\mathbf{y}|\mathbf{x}) = \frac{\pi_\theta(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_\theta)}\,.$$

Method	Target Distribution	Per Question Scaling	Sampling Req	Update Steps
Policy Gradient	$\propto \pi_t(\mathbf{y}\|\mathbf{x})R(\mathbf{x},\mathbf{y})$	1	Online	One/Few
EM	$\propto \pi_t(\mathbf{y}\|\mathbf{x})R(\mathbf{x},\mathbf{y})$	$1/v(\mathbf{x},\pi_t)$	Online	One/Few
Filtered SFT	$\propto \pi_t(\mathbf{y}\|\mathbf{x})R(\mathbf{x},\mathbf{y})$	1	High	Many
GEM	$\propto \pi_{t,t-1,\ldots}(\mathbf{y}\|\mathbf{x})R(\mathbf{x},\mathbf{y})$	–	Online	Many

Table 1: Policy optimization methods at policy iteration t

The above describes any set of policy gradient updates [Williams, 1992, Schulman et al., 2017, Guo et al., 2025]. It is interesting to note that the per-question gradient is scaled by a factor proportion to the performance on that question, $v(\mathbf{x},\pi_\theta)$. Correspondingly, a set of methods based on expectation maximization to directly increase the likelihood of success [Dayan and Hinton, 1997, Abdolmaleki et al., 2018b, 2024, Tajwar et al., 2026] do weighted SFT updates which effectively has the effect of canceling out this scaling factor,

$$\begin{aligned}\nabla\mathcal{J}_{\text{EM}}(\pi_\theta) &= \mathbb{E}_{\mu(\mathbf{x})}\left[\frac{1}{v(\mathbf{x},\pi_\theta)}\mathbb{E}_{\pi_\theta(\mathbf{y}|\mathbf{x})}\left[R(\mathbf{x},\mathbf{y})\nabla_\theta\log\pi_\theta(\mathbf{y}|\mathbf{x})\right]\right] \\ &= \mathbb{E}_{\mu(\mathbf{x})}\left[\mathbb{E}_{q_\theta(\mathbf{y}|\mathbf{x})}\left[\nabla_\theta\log\pi_\theta(\mathbf{y}|\mathbf{x})\right]\right].\end{aligned}$$

It is worth noting that q_θ is precisely also the distribution we would get in the binary reward case from filtering. That is, iterative filtered SFT, REINFORCE, and EM-based RL methods are optimizing the same expected gradient (up to a per-question scaling factor). We summarize this in Table 1, along with our algorithm GEM.

Recovery of Existing Methods from GEM. We can recover various existing methods from GEM by picking a specific choice of buffer sampling distribution and update steps. For instance, to recover **MaxRL** [Tajwar et al., 2026] or EM [Dayan, 1993] requires setting buffer capacity to $C = BK$ (buffer holds exactly one batch, so data is never reused across collection rounds), $G = 1$, mini-batches are drawn *without replacement* so that every correct rollout in the buffer is used exactly once, and the sampling weights are $1/\hat{p}_q$. Under these settings the effective contribution of each prompt $\mathbf{x}$ to the loss is the sum over all its correct rollouts, each weighted by $1/\hat{p}_q$. Because $\hat{p}_q = c_q/n_q$ and there are c_q correct entries, the total weight for prompt $\mathbf{x}$ is $c_q \cdot (n_q/c_q) = n_q$, which is constant across prompts. This replicates the MaxRL estimator: every correct rollout contributes equally to the gradient regardless of the prompt's difficulty, matching the uniform-over-prompts weighting implicit in the MaxRL gradient. Our method extends MaxRL by (a) maintaining a persistent buffer ($C \gg BK$) so correct traces are reused across collection rounds, and (b) performing $G > 1$ gradient steps per collection phase.

7 Conclusion

We introduced Good Experience Maximization (GEM), a post-training framework that retains and reuses successful generations while continuing to collect new online samples. Our target-matching view shows key tensions between policy gradient and iterative SFT. GEM addresses this tension by training toward a mixture of historical and newly generated successes. We proved that the GEM target improves expected reward at convergence and derived a flow-based update that guarantees one-step policy improvement by moving along a straight-line path in probability space. Empirically, GEM improves sample efficiency and performs particularly well in sparse-reward settings. Future work should scale these experiments further, study practical buffer design, and characterize when standard SFT is sufficient versus when the flow update is needed. Additionally, we note that we do not make use of *negative* samples [Abdolmaleki et al., 2024] in our algorithm. These are useful because they can help guide search and can provide rich information depending on the structure (e.g., a negative sample of two options is equally as informative as a positive one).

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, and Martin Riedmiller. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018a.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018b.

Abbas Abdolmaleki, Bilal Piot, Bobak Shahriari, Jost Tobias Springenberg, Tim Hertweck, Rishabh Joshi, Junhyuk Oh, Michael Bloesch, Thomas Lampe, Nicolas Heess, et al. Learning from negative feedback, or positive feedback or both. *arXiv preprint arXiv:2410.04166*, 2024.

Shun-ichi Amari. *Information geometry and its applications*. Springer, 2016.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Quantile filtered imitation learning. *arXiv preprint arXiv:2112.00950*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993.

Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Sham M Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems*, 14, 2001.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

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.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

Marcelo G Mattar and Nathaniel D Daw. Prioritized memory access explains planning and hippocampal replay. *Nature neuroscience*, 21(11):1609–1617, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ida Momennejad, A Ross Otto, Nathaniel D Daw, and Kenneth A Norman. Offline replay supports planning in human reinforcement learning. *elife*, 7:e32548, 2018.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

Anna C Schapiro, Elizabeth A McDevitt, Timothy T Rogers, Sara C Mednick, and Kenneth A Norman. Human hippocampal replay during rest prioritizes weakly learned information and predicts memory performance. *Nature communications*, 9(1):3920, 2018.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 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, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*, 2025.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, and Haiwen Feng. Maximum likelihood reinforcement learning. *arXiv preprint arXiv:2602.02710*, 2026.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

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

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.

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

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

A Deferred Proofs

A.1 Proofs from Section 4

Proof of Lemma 4.1. By the definition provided in the sketch, the mixture policy π_M is a convex combination of $\pi_{\mathcal{B}}$ and π_t for any $\mathbf{x}$. We can write the probability of $\mathbf{y}$ in $\mathbf{x}$ under π_M as $\pi_M(\mathbf{y}|\mathbf{x}) = \alpha(\mathbf{x})\pi_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + (1 - \alpha(\mathbf{x}))\pi_t(\mathbf{y}|\mathbf{x})$, where $\alpha(\mathbf{x})$. First consider the case where $\mathbf{x} \in S_{\mathcal{B}}$. In this region, $\alpha(\mathbf{x}) > 0$. The expected reward at $\mathbf{x}$ under π_M is given by:

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_M}[R(\mathbf{x},\mathbf{y})] &= \sum_{\mathbf{y}\in\mathcal{A}} \pi_M(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y}) \\ &= \sum_{\mathbf{y}\in\mathcal{A}} \left(\alpha(\mathbf{x})\pi_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + (1-\alpha(\mathbf{x}))\pi_t(\mathbf{y}|\mathbf{x})\right) R(\mathbf{x},\mathbf{y}) \\ &= \alpha(\mathbf{x}) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x},\mathbf{y})] + (1-\alpha(\mathbf{x})) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})].\end{aligned}$$

By the assumption that $\mathbb{E}_{\mathbf{y}\sim\pi_{\mathcal{B}}}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})]$ for all $\mathbf{x} \in S_{\mathcal{B}}$, and since $\alpha(\mathbf{x}) > 0$, the weighted average must be strictly greater than the target expectation. Thus, $\mathbb{E}_{\mathbf{y}\sim\pi_M}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})]$ for all $\mathbf{x} \in S_{\mathcal{B}}$.

Next, consider the case where $\mathbf{x} \notin S_{\mathcal{B}}$. For these, the mixture weight $\alpha(\mathbf{x})$ reduces to 0. Consequently, the mixture policy simplifies to $\pi_M(\mathbf{y}|\mathbf{x}) = \pi_t(\mathbf{y}|\mathbf{x})$. It follows immediately that the expectations are equal: $\mathbb{E}_{\mathbf{y}\sim\pi_M}[R(\mathbf{x},\mathbf{y})] = \mathbb{E}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})]$.

Finally, to arrive at the global inequality, we take the expectation over $d(\mathbf{x})$. The total expected reward is:

$$\begin{aligned}\mathop{\mathbb{E}}_{\pi_M}[R(\mathbf{x},\mathbf{y})] &= \int_{S_{\mathcal{B}}} d(\mathbf{x}) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_M}[R(\mathbf{x},\mathbf{y})]\, d\mathbf{x} + \int_{\mathcal{S}\setminus S_{\mathcal{B}}} d(\mathbf{x}) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_M}[R(\mathbf{x},\mathbf{y})]\, d\mathbf{x} \\ &\geq \int_{S_{\mathcal{B}}} d(\mathbf{x}) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})]\, d\mathbf{x} + \int_{\mathcal{S}\setminus S_{\mathcal{B}}} d(\mathbf{x}) \mathop{\mathbb{E}}_{\mathbf{y}\sim\pi_t}[R(\mathbf{x},\mathbf{y})]\, d\mathbf{x} \\ &= \mathop{\mathbb{E}}_{\pi_t}[R(\mathbf{x},\mathbf{y})].\end{aligned}$$

The inequality holds because the integrand is strictly greater on $S_{\mathcal{B}}$ and equal elsewhere. Thus, $\mathbb{E}_{\pi_M}[R(\mathbf{x},\mathbf{y})] \geq \mathbb{E}_{\pi_t}[R(\mathbf{x},\mathbf{y})]$. □

Proof of Lemma 4.2. We consider the total expected reward under the converged mixture distribution P_M. By definition, the mixture distribution is a convex combination of the buffer distribution and the current policy's distribution, expressed as $P_M = \alpha P_{\mathcal{B}} + (1-\alpha)P_{\pi_t}$, where $P_{\pi_t} = \mu(\mathbf{x})\pi_{\theta_t}(\mathbf{y}|\mathbf{x})$ represents the distribution under the current policy. Let $\mathcal{J}(P) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}[R(\mathbf{x},\mathbf{y})]$ denote the expected reward under a given distribution P. We can expand the expectation for the mixture policy by substituting the definition of P_M:

$$\mathcal{J}(P_M) = \int_{\mathcal{S}}\int_{\mathcal{A}} R(\mathbf{x},\mathbf{y})\left[\alpha P_{\mathcal{B}}(\mathbf{x},\mathbf{y}) + (1-\alpha)P_{\pi_t}(\mathbf{x},\mathbf{y})\right] d\mathbf{y}\, d\mathbf{x}.$$

By the linearity of the integral, we can decompose this expression into two distinct components corresponding to the reward contribution of the buffer and the current policy:

$$\mathcal{J}(P_M) = \alpha \iint_{\mathcal{S}\times\mathcal{A}} R(\mathbf{x},\mathbf{y})P_{\mathcal{B}}(\mathbf{x},\mathbf{y})d\mathbf{y}\, d\mathbf{x} + (1-\alpha)\iint_{\mathcal{S}\times\mathcal{A}} R(\mathbf{x},\mathbf{y})P_{\pi_t}(\mathbf{x},\mathbf{y})d\mathbf{y}\, d\mathbf{x}.$$

This expression simplifies to a weighted sum of the expected rewards:

$$\mathcal{J}(P_M) = \alpha\mathbb{E}_{P_{\mathcal{B}}}[R] + (1-\alpha)\mathcal{J}(P_{\pi_t}).$$

Based on the pointwise conditions from Lemma 4.1, we assume that the expected reward of the buffer distribution strictly exceeds that of the current policy over the shared support, such that $\mathbb{E}_{P_{\mathcal{B}}}[R] > \mathcal{J}(P_{\pi_t})$. Given that the mixing weight α is strictly greater than zero, we can apply this strict inequality to our decomposition:

$$\mathcal{J}(P_M) > \alpha\mathcal{J}(P_{\pi_t}) + (1-\alpha)\mathcal{J}(P_{\pi_t}).$$

Factoring out the term $\mathcal{J}(P_{\pi_t})$ on the right-hand side, we obtain:

$$\mathcal{J}(P_M) > (\alpha + 1 - \alpha)\mathcal{J}(P_{\pi_t}) = \mathcal{J}(P_{\pi_t}).$$

Thus, we have shown that for any mixture weight $\alpha \in (0, 1)$, the expected reward under the converged mixture policy P_M is strictly greater than the expected reward under the current policy π_{θ_t}. □

Proof of Proposition 4.3. To map the probability simplex to the parameter space θ, we select a reference action K. The parameters θ_i for $i \in \{1, \ldots, K-1\}$ are defined by the log-ratio:

$$\theta_i = \ln(p_i) - \ln(p_K)$$

subject to the simplex constraint $p_K = 1 - \sum_{j=1}^{K-1} p_j$. We first derive the Jacobian $J_{ij} = \frac{\partial \theta_i}{\partial p_j}$ to understand how changes in probability space affect the parameters. For the diagonal case where $i = j$:

$$\frac{\partial \theta_i}{\partial p_i} = \frac{\partial}{\partial p_i}(\ln p_i - \ln p_K) = \frac{1}{p_i} - \frac{1}{p_K}\frac{\partial p_K}{\partial p_i} = \frac{1}{p_i} + \frac{1}{p_K}$$

For the off-diagonal case where $i \neq j$:

$$\frac{\partial \theta_i}{\partial p_j} = \frac{\partial}{\partial p_j}(\ln p_i - \ln p_K) = 0 - \frac{1}{p_K}\frac{\partial p_K}{\partial p_j} = \frac{1}{p_K}$$

Thus, the general form of the Jacobian is $\frac{\partial \theta_i}{\partial p_j} = \frac{\mathbb{1}_{\{i=j\}}}{p_i} + \frac{1}{p_K}$. Using the chain rule, the velocity of the parameters $\dot{\theta}_i$ given a probability velocity $\dot{p}$ is:

$$\dot{\theta}_i = \sum_{j=1}^{K-1} \frac{\partial \theta_i}{\partial p_j}\dot{p}_j = \frac{\dot{p}_i}{p_i} + \frac{1}{p_K}\sum_{j=1}^{K-1} \dot{p}_j.$$

Since the probabilities must sum to 1, the sum of their derivatives is zero, implying $\sum_{j=1}^{K-1} \dot{p}_j = -\dot{p}_K$. Substituting this into the equation above yields the explicit parameter update:

$$\dot{\theta}_i = \frac{\dot{p}_i}{p_i} - \frac{\dot{p}_K}{p_K}$$

Write the time derivative of the expected reward $\mathcal{J}$ using the chain rule over the policy parameters θ as:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K-1} \frac{\partial \mathcal{J}}{\partial \theta_i}\dot{\theta}_i.$$

Recall the standard result for the gradient of the expected reward with respect to softmax logits: $\frac{\partial \mathcal{J}}{\partial \theta_i} = p_i(Q_i - \mathbb{E}_{a\sim p}[Q(s,a)]) = p_i A_i$, where A_i is the advantage of action i. Substituting this gradient and the definition of the logit dynamics $\dot{\theta}_i$ into the chain rule expression, we obtain:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K-1} p_i A_i \left(\frac{\dot{p}_i}{p_i} - \frac{\dot{p}_K}{p_K}\right).$$

Distributing the terms inside the summation yields:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K-1} \dot{p}_i A_i - \frac{\dot{p}_K}{p_K}\sum_{i=1}^{K-1} p_i A_i.$$

By the definition of the advantage function, the expected advantage under the current policy is zero, i.e., $\sum_{i=1}^{K} p_i A_i = 0$. This implies that $\sum_{i=1}^{K-1} p_i A_i = -p_K A_K$. Substituting this identity into the second term of our derivative expression, we have:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K-1} \dot{p}_i A_i - \frac{\dot{p}_K}{p_K}(-p_K A_K) = \sum_{i=1}^{K-1} \dot{p}_i A_i + \dot{p}_K A_K = \sum_{i=1}^{K} \dot{p}_i A_i.$$

We now plug in the trajectory of the probability distribution, $\dot{p}_i = m_i - p_i$. Substituting this into the simplified summation gives:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K}(m_i - p_i)A_i = \sum_{i=1}^{K} m_i A_i - \sum_{i=1}^{K} p_i A_i.$$

As established previously, $\sum_{i=1}^{K} p_i A_i = 0$. Therefore, the derivative of the performance objective reduces exactly to the expected advantage of the target distribution m under the current policy:

$$\frac{d\mathcal{J}}{dt} = \sum_{i=1}^{K} m_i A_i.$$

By the assumption that $\sum_j m_j A_j \geq 0$, it follows immediately that $\frac{d\mathcal{J}}{dt} \geq 0$.

□

Hyperparameter	Default Value
Learning rate	10^{-3}
Optimizer	Adam
Batch size	32
Rollouts per context	4
Total sample budget	50,000

Table 2: Default training hyperparameters for all MNIST experiments.

Parameter	Value
Base model	Llama-3.2-3B
LoRA rank	32
Train dataset	MATH
Eval dataset	MATH-500
Reward	binary
Prompts per batch	16
Generations per prompt	8
Max response length	2048
Training steps	400
Optimizer	Adam (β_1=0.9, β_2=0.95)
Grad updates per data collection step	1
Learning rate	1×10^{-5}
Rollout temp / top-p	1.0 / 1.0
Validation temp / top-p	1.0 / 0.95
Compute backend	Tinker (managed)

Table 3: Setup and data for math reasoning.

B Experiment Details

B.1 Pass@k Calculation

We use bootstrapping low variance unbiased estimator for estimating pass@k from samples [Chen et al., 2021]. This estimator is calculated by generating $n \geq k$ samples per task and counting the number of correct samples $c(\mathbf{x})$ among the n samples. The pass@k is therefore:

$$\text{pass@k} := \mathbb{E}_{\mathbf{x}\sim\mu}\left[1 - \frac{\binom{n-c(\mathbf{x})}{k}}{\binom{n}{k}}\right].$$

B.2 MNIST

We frame MNIST as a contextual bandit task. Each problem $\mathbf{x}$ represents the context. It is a 28×28 grayscale image (flattened to $\mathbb{R}^{784}$, normalized to $[0, 1]$). Each answer in the answer space $\mathcal{A} = \{0, \ldots, C-1\}$ (with $C = 10$ by default) represents an arm. The reward is binary, such that $r(\mathbf{y}, y) = \mathbf{1}[\mathbf{y} = y]$, where y is the true digit label. The MNIST training set contains 60,000 images and the test set 10,000 images.

Hyperparameters. The results in Figure 3 were generated with the hyperparameters in Table 2. All algorithms were trained using ViT architecture [Yuan et al., 2021] with 6 layers and 4 heads with patch size 4. GEM used $\alpha = 0.5$, buffer capacity of 4096, $1/p$ sampling, 5 inner updates per outer loop.

Compute Resources. Each run takes around 4-5 hours and requires a single L40.

B.3 Math Reasoning

Hyperparameters. Table **??** lists the hyperparameters for the mathematicla reasoning tasks.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and/or introduction clearly states the claims made.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Throughout the paper, we explicitly mention failure cases and limitations of the methods developed. We mention assumptions where possible and address the scope of the work. In particular, because we are compute-limited, we could only run a few larger-scale experiments. However, we provide supporting evidence in simpler settings so that we can run with multiple seeds.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: To the best of our knowledge, we clearly state all assumptions. All proof sketches in the main paper are accompanied by deferred full proofs in Appendix A. To the best of our knowledge, we have included references to any theorems or references that the proofs rely on.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the explicit high-level algorithm, as well as recommendations for low-level details. We also provide hyperparameters toward the goal of reproducibility.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We are provide the code here: `https://anonymous.4open.science/r/0E1FjO/README.md`.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide experimental settings in the paper when the experiments are introduced. We do a best-faith effort to include the details relevant for appreciating the results. We provide full experimental settings in Section B.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report error bars when there are multiple seeds.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide compute details.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: It does.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper is foundational research and not tied to particular applications.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We do a best-faith effort to cite all previous assets, and we are happy to include any references that we may have overlooked.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We include as much detail as possible about our code.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.