Good Experience Maximization

Anthony GX-Chen*
NYU

Stephanie Milani*
NYU

Jatin Prakash
NYU

Sumit Chopra
NYU

Rob Fergus
NYU

Rajesh Ranganath
NYU

Abstract

Most post-training methods for language models (LMs) are wasteful. They are predominantly on-policy, meaning each generation is used once then discarded. This process forces the model to rediscover rare solutions and risks the model forgetting how to solve previous problems. We introduce Good Experience Maximization (GEM), a simple approach that continuously discovers new solutions through online sampling, while selectively maintaining a buffer of historical success to continue training on them. Theoretically, GEM can be interpreted through the lens of target distribution matching, lends itself to policy improvement guarantees, and recovers several existing post-training methods (including MaxRL) as special cases. Empirically, GEM demonstrates strong empirical performance over standard policy gradient methods, particularly in sparse reward settings where the probability of discovering good solutions is small.

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 that it supports important cognitive tasks, including memory consolidation, planning, and decision-making [Momennejad et al., 2018, Schapiro et al., 2018]. Machine learning has a direct analog of this idea, experience replay [Lin, 1992, Mnih et al., 2013], which has been central to sample-efficient learning. When useful experience is costly, sparse, or generated by a changing policy, retaining it enables the learner to extract more signal without requiring the same experience to be regenerated [Andrychowicz et al., 2017, Vinyals et al., 2019]. Despite these benefits, this mechanism is largely absent from the methods used to post-train language models (LMs). An ideal training regime would capture this simple principle: retain what worked and keep learning from it.

We refer to these properties as *retention* and *discovery*, respectively. An algorithm that incorporates both must confront four challenges. First, retaining and training on only a subset of experiences can undo progress on others. Because the model shares parameters across all inputs, supervised fine tuning (SFT) or, to a lesser degree, reinforcement learning (RL) can lead to temporal forgetting [Chen et al., 2025]: a model can no longer solve problems previously answered correctly during training [Li et al., 2025]. As a result, an algorithm should exhibit *common solution retention*: it must not let the model forget what it has already learned. Second, the experiences that matter most are often the most difficult to obtain. Sparse rewards have long motivated replay-based methods in RL [Lin, 1992, Schaul et al., 2016, Andrychowicz et al., 2017]; in LM training, the same issue arises when correct generations have low probability under the current policy. Once found, the learner should extract as much information from these experiences as possible, promoting *rare solution retention*.

*Equal contribution.

Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning

Property	PG	F. SFT	F. SFT (batch)	GEM	PathGEM
Common solution retention	✓	✓	✗	✓	✓
Rare solution retention	✗	✓	✓	✓	✓
Online / incremental discovery (*discovery*)	✓	✗	✓	✓	✓
Guaranteed monotonic improvement	✓	✗	✗	✗	✓
External memory size	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathcal{D}\rvert)$	$\mathcal{O}(\lvert\mathcal{B}\rvert)$	$\mathcal{O}(\lvert\mathcal{B}\rvert)$	$\mathcal{O}(\lvert\mathcal{B}\rvert)$

Table 1: Comparison of discovery and retention properties across methods. PG guarantees efficient discovery of solutions with the most up-to-date policy along with improvement guarantees, while Filtered SFT (F. SFT) retains all seen solutions but is slow to discover new solutions. We seek to achieve both in our method, GEM. Note $|\mathcal{D}|$ and $|\mathcal{B}|$ refer to dataset and external buffer sizes, respectively.

Third, retention alone cannot produce continued discovery. Replay by itself preserves past successes, but a learner trained only on a fixed set of successful generations is bounded by the support of that set and can overfit to it, a failure mode also observed in iterative self-training [Gulcehre et al., 2023]. Moreover, training must enable improvements in the policy to quickly lead to new samples and further improvement, which we call *online / incremental discovery*. Fourth, updates can interrupt that cycle by decreasing reward (a phenomenon that we show in Figure 1). Because the current policy generates future training data, a reward-decreasing update can decrease the likelihood of useful trajectories, slowing subsequent discovery. Ideally, an algorithm should improve the policy at every step, not only at convergence, called *monotonic improvement*, a property achieved by some classic policy-improvement methods under specific conditions [Dayan and Hinton, 1997, Kakade and Langford, 2002, Schulman et al., 2015].

This paper studies how to realize these principles in post-training. We first analyze the primary methods for post-training LMs, policy gradient (PG) and filtered SFT (F. SFT). We show the tradeoffs of the different methods with respect to the four properties. Motivated by this analysis, we then introduce an algorithm, called *Good Experience Maximization* (GEM), that enables both *retention* and *discovery*. GEM explicitly retains successes in a replay buffer, which makes previously-discovered solutions available for repeated training without requiring the answers to be generated again, thus providing both common and rare solution retention. To support online / incremental discovery, GEM does not only train on samples from the buffer; instead, it constructs a target distribution by mixing historical and newly-generated successes from the current policy. The online component keeps the target tied to the current learner, enabling policy improvement to generate new data for future updates.

Once the target is constructed, GEM moves the policy toward this target in two ways with different tradeoffs. To enable compatibility with existing training pipelines, we propose to use the standard cross-entropy update. We also derive a flow-based update that moves the policy directly toward the target in probability space, guaranteeing monotonic improvement at every step (although it requires a custom optimization rule). Empirically, we show that GEM can match or exceed PG algorithms in terms of performance and sample efficiency. We demonstrate how different choices for GEM recover a broad set of existing post-training methods as special cases, including MaxRL [Tajwar et al., 2026].

2 Preliminaries and Notation

We study the contextual bandit setting with inputs $\mathbf{x} \in \mathcal{X}$ and outputs $\mathbf{y} \in \mathcal{Y}$. Given a stochastic policy $\pi_\theta(\cdot \mid \mathbf{x}) \in \Delta(\mathcal{Y})$ and a ground-truth reward function $R : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, the goal is to maximize the expected reward of π_θ given some input distribution $\mathbf{x} \sim \mu$,

$$\mathcal{J}(\pi_\theta) := \mathbb{E}_{\mathbf{x}\sim\mu\,,\,\mathbf{y}\sim\pi_\theta(\cdot|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big]\,. \tag{1}$$

This process of post-training proceeds over a sequence of policies $\pi_1, \pi_2, \ldots, \pi_T$. We also write $V^\pi(\mathbf{x}) := \mathbb{E}_{\mathbf{y}\sim\pi(\cdot \mid \mathbf{x})}[R(\mathbf{x},\mathbf{y})]$ as the expected reward of π on $\mathbf{x}$. In LM post-training settings, such as RL with verifiable rewards (RLVR) [Lambert et al., 2024, Su et al., 2025], μ is uniform over a fixed set of training questions, and reward is binary indicating correctness. Importantly, the model generates $\mathbf{y}$ by way of a token sequence that is deterministically parsed using delimiters into an (optional) reasoning trace $\mathbf{r}$ and $\mathbf{y}$.

3 Discovery and Retention

Effective post-training algorithms should ideally *discover*—produce increasingly successful trajectories over time—and *retain*—preserve successful traces and learn from them once they are sampled. We examine existing families of algorithms and show how they differ in their ability to achieve either goal. Table 1 provides a high-level overview.

3.1 Policy Gradient Efficiently Discovers but is Slow to Retain

PG algorithms [Williams, 1992, Schulman et al., 2017, Guo et al., 2025] perform (stochastic) gradient ascent on $\mathcal{J}(\pi_\theta)$ by taking,

$$\nabla_\theta \mathcal{J}(\pi_\theta) := \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu,\mathbf{y}\sim\pi_\theta} \left[R(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y} \mid \mathbf{x})\right] . \tag{2}$$

Because μ has broad coverage over $\mathcal{X}$ and $\mathcal{Y}$ are sampled using the most recent π_θ (which is improving in expectation), the repeated sampling allows PG to enjoy online / incremental discovery. However, this process is wasteful: a good sample $\mathbf{y}$ will receive *one* gradient update before being discarded, meaning it does not retain rare solutions. Retention-friendly PG methods handle off-policy samples by *approximating* on-policy updates; updates include samples from previous behavior policies $\pi_b \neq \pi_\theta$ but with an importance weight correction [Precup et al., 2000],

$$\nabla_\theta \mathcal{J}(\pi_\theta) := \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu,\mathbf{y}\sim\pi_b} \left[\frac{\pi_\theta(\mathbf{y} \mid \mathbf{x})}{\pi_b(\mathbf{y} \mid \mathbf{x})} R(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y} \mid \mathbf{x})\right] .$$

This approach is limited in two major ways: i) samples with low probability under π_b will incur large importance weights, de-stabilizing the update, and ii) samples with low probability under π_θ, $\pi_\theta(\mathbf{y}|\mathbf{x}) \approx 0$, shrinks the gradient toward 0, making its contribution negligible. The latter is problematic because we may wish to update with successful $\mathbf{y}$'s exactly when they are rare under π_θ, and the reason why PG—including off-policy corrected estimators—limits retention.

3.2 Filtered SFT Provides Retention but May Forget or be Slow to Discover

F. SFT methods [Brandfonbrener et al., 2021, Zelikman et al., 2022, Dong et al., 2023, Gulcehre et al., 2023, Singh et al., 2023, Havrilla et al., 2024] take a different approach. They retain high-reward samples by sampling the data, filtering and storing the high-reward data, then taking multiple gradient steps to maximize the (log) likelihood,

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

where $P_{\mathcal{B}}$ is a distribution defined over some buffer $\mathcal{B}$ with capacity C. Unlike PG, F. SFT methods explicitly retain rare solutions by performing multiple gradient updates, and notably do not shrink the update by $\pi_\theta(\mathbf{y}|\mathbf{x})$. F. SFT methods fall into two types, full dataset and batch, depending on how much of the dataset they sample before performing updates.

Full dataset filtered SFT retains common solutions but is slow to discover. F. SFT in which nearly the full dataset is sampled before performing any often incurs a large upfront sampling cost with a (possibly bad) policy before any gradient updates occur, thereby hindering *discovery*.

Batch filtered SFT enjoys incremental discovery at the cost of common solution retention. If, instead, F. SFT samples only a batch of data, it can easily overfit [Gulcehre et al., 2023], as its objective is undefined on $\mathbf{x}$ not in the the support of the current buffer distribution $P_{\mathcal{B}}$.

In both cases, taking multiple updates of Equation (3) no longer guarantees monotonic improvement over the initial π_θ, as improvements in likelihoods do not always translate to improvements in rewards (Figure 1). As a result, neither PG nor F. SFT alone gives us an algorithm that both *retains* and *discovers*.

4 Good Experience Maximization

This framing suggests a natural fix: combine the benefits of PG and SFT by continuing to sample from the current policy π_t while training on the buffer $\mathcal{B}$. Based on this principle, we propose a simple algorithm, called Good Experience Maximization (GEM), that consists of two high-level steps, detailed in Algorithm 1. First, it constructs a mixture distribution P_M from the current buffer and the current policy (*Generate Good Experiences*). Second, it moves π_θ toward P_M (*Maximize Them*).

Algorithm 1 Good Experience Maximization (GEM)

1: Base LLM π_1, buffer $\mathcal{B} \leftarrow \emptyset$
2: **for** policy iteration $t = 1, \ldots, T$ **do**
3: Initialize training policy $\pi_\theta \leftarrow \pi_t$
4: **for** step $j = 1, \ldots, J$ **do**
// Generate Good Experiences
5: Sample from mixture $\{(\mathbf{x}_i, \mathbf{y}_i, r_i)\}_{i=1}^n \sim P_M | P_M(\mathbf{x}, \mathbf{y}) = \alpha P_{\mathcal{B}}(\mathbf{x}, \mathbf{y}) + (1-\alpha) P_{\pi_t}(\mathbf{x}, \mathbf{y})$
// Maximize Them
6: Update policy: $\pi_{\theta_{j+1}} \leftarrow u(\mathbf{x}, \mathbf{y}, P_M, \pi_{\theta_j})$
7: Update buffer: $\mathcal{B} \leftarrow g(\mathcal{B}, \{(\mathbf{x}_i, \mathbf{y}_i, r_i, t)\}_{i=1}^n)$
8: **end for**
9: $\pi_{t+1} \leftarrow \pi_\theta$
10: **end for**

4.1 GEM Target: Generating Good Experiences

We first require a *target* distribution P_M for π_θ to match. The simplest choice is the buffer distribution $P_{\mathcal{B}}$ itself: just train on stored successes, as F. SFT does. However, as we have shown, this construction is support-limited and can lead to overfitting. Consequently, we propose to mix in online samples from the current policy to those from $\mathcal{B}$ (line 5, Algorithm 1). Let $P_{\pi_t}(\mathbf{x}, \mathbf{y}) := \mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})$ and $P_{\mathcal{B}}(\mathbf{x}, \mathbf{y}) := P_{\mathcal{B}}(\mathbf{x})P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})$ be joint distributions over $(\mathbf{x}, \mathbf{y})$ pairs induced by the current policy and buffer distribution, respectively. Then, we define a mixture P_M with weights $0 < \alpha < 1$:

$$P_M(\mathbf{x}, \mathbf{y}) := \alpha\, P_{\mathcal{B}}(\mathbf{x}, \mathbf{y}) + (1-\alpha)\, P_{\pi_t}(\mathbf{x}, \mathbf{y})\,. \tag{4}$$

Intuitively, $P_{\mathcal{B}}$ concentrates mass on previously-discovered, high-reward $(\mathbf{x}, \mathbf{y})$ pairs, which enables the learner to reuse past successes (*retention*). P_{π_t} anchors the target to the data distribution of the current policy π_t, which provide support over inputs where good experiences have not yet been discovered (*discovery*). We will now show why this distribution is desirable as a target.

This target provably improves on π_t. Importantly, the target P_M has greater expected reward than π_t.

Lemma 4.1. *Pointwise improvement. For any state $\mathbf{x}$ with $P_{\mathcal{B}}(\mathbf{x}) > 0$, if the buffer has strictly higher expected reward, $\mathbb{E}_{P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})}[R(\mathbf{x}, \mathbf{y})] > \mathbb{E}_{\pi_t}[R(\mathbf{y}, \mathbf{x})]$, then for any $\alpha \in (0, 1)$:*

$$\mathbb{E}_{P_M(\mathbf{y}|\mathbf{x})}[R(\mathbf{x}, \mathbf{y})] > \mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x}, \mathbf{y})]\,.$$

Proof sketch. $P_M(\mathbf{y}|\mathbf{x}) = P_M(\mathbf{x}, \mathbf{y})/P_M(\mathbf{x})$ is a mixture distribution, and its expected reward is simply a weighted average between the expected rewards of $P_{\mathcal{B}}$ and π_t. □

Proposition 4.2. *Global improvement. Assume a buffer distribution sharing non-zero support with μ, $\sum_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x})P_{\mathcal{B}}(\mathbf{x}) > 0$. Evaluating the conditional mixture distribution $P_M(\mathbf{y}|\mathbf{x})$ over μ yields a strict improvement over the current policy π_t for any $\alpha \in (0, 1)$:*

$$\mathop{\mathbb{E}}_{\mu(\mathbf{x})P_M(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x}, \mathbf{y})\Big] > \mathop{\mathbb{E}}_{\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x}, \mathbf{y})\Big]\,. \tag{5}$$

Proof sketch. For any $\mathbf{x}$ in the support of μ, we either have the same expected reward as π_t, or an improvement as provided by Lemma 4.1. □

Populating and sampling from the buffer. These guarantees depend only on the property that $P_{\mathcal{B}}$ has higher reward that π_t on its support. The construction of $P_{\mathcal{B}}$ consists of two components: the maintenance rule g (line 7) for storing and evicting from the buffer once the capacity C is reached and the sampling distribution $P_{\mathcal{B}}$. Different settings of these choices recover existing methods as special cases, which we briefly mention in Section 4.3.

4.2 GEM Update: Maximizing Good Experiences

We now have a target that provably outperforms π_t; what remains is how to move π_θ toward it (Algorithm 1, line 6). Roughly speaking, a good update mechanism should successfully match $\pi_{t+1} \approxeq P_M$ to produce π_{t+1} that improves over π_t. To realize this, we propose two update rules.

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

1: Compute probability velocity $\dot{\pi_\theta} = P_M - \pi_\theta$
2: Map to parameter space $L = \dot{\pi_\theta} \oslash \pi_\theta$
3: **return** L

A practical SFT update. A straight-forward way to match the target is via the forward KL (equivalently, the cross-entropy (CE) or "SFT") objective $D_{\mathrm{KL}}(P_M \| \pi_\theta)$. The benefit of this update is it is straightforward to implement within standard training pipelines. We refer to GEM with this update as GEM+CE. Concretely, the loss is,

$$\hat{l}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_M}\big[-\log \pi_\theta(\mathbf{y}|\mathbf{x})\big]\,. \tag{6}$$

We give a lower bound in Appendix B.1.2 on how far π_{t+1} must travel along the gradient path of $\nabla_\theta D_{\mathrm{KL}}(P_M || \pi_\theta)$ to guarantee improvement over π_t. This guarantee is realizable by running the inner optimization nearly to completion at every t, which is expensive because it requires collecting online samples. If we instead stop the inner loop early, the resulting policy may not improve upon the previous one even if the target has higher expected reward, as illustrated in Figure 1 (details in Appx. C.2).

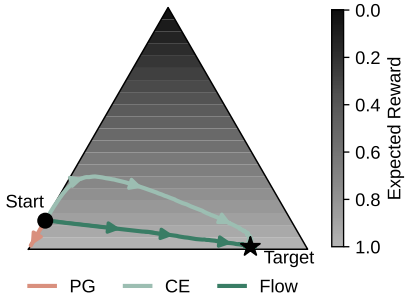


Figure 1: Update geometry on a 3-action task. From a shared start (circle), we trace the trajectory each update follows toward a fixed target distribution (star). PG does not use the target and moves directly toward the nearest reward-maximal vertex. The CE update reaches the target via a reward non-monotonic path, whereas the Flow update monotonically increases in reward.

A probability-space flow enables monotonic improvement. An alternative way to guarantee policy improvement is to instead prescribe the trajectory directly in probability space. As seen in Figure 1, moving along the "straight line path" in probability space (an m-geodesic, Amari [2016]) from the current policy to the target distribution guarantees monotonic improvements in expected rewards. We can therefore construct an update direction such that the distribution moves according to the velocity vector that follows the interpolation of mixture distributions from π_θ (at $\tau = 0$) to P_M (at $\tau = 1$),

$$\dot{\pi_\theta} = \frac{d\pi_\theta}{d\tau} := P_M - \pi_\theta\,. \tag{7}$$

To realize $\dot{\pi_\theta}$ in probability space, we derive the corresponding update to the logits of a softmax which is commonly used to parameterize categorical distribution (over tokens) in transformers. We refer to this as GEM-Flow, and show in Appendix B.3 how this guarantees monotonic policy improvements *at every step*, which rely on the fact that the expected reward is linear in the mixture coordinates.

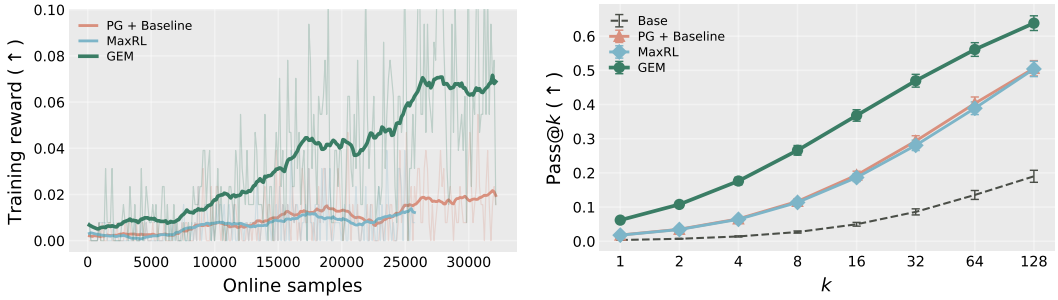
4.3 Relationship to Existing Work

It should be noted that GEM can be viewed generally as a kind of Filtered SFT procedure. While existing F.SFT methods [Dong et al., 2023, Havrilla et al., 2024] follow a recipe of (i) sampling many responses from many prompts, then (ii) taking many updates on the good responses; GEM instead seek a middle ground between fully offline updates and fully online methods [Schulman et al., 2017, Guo et al., 2025], through both saving all sampled good responses, while still continuously sampling online, which is precisly the mechanism required for retention and discovery, respectively.

Additionally, GEM can be viewed as target distribution matching to a specifically chosen target distribution. Indeed, the update direction of policy gradient and more recent methods such as MaxRL [Tajwar et al., 2026] can be recovered as special cases of GEM with specific choices of sampling parameters. We provide additional detailed discussion and theoretical connections in Appendix A.2.

5 Experiments

In this section, we empirically validate the effectiveness of GEM, comparing specifically against other online sampling methods including (i) PG with a mean reward baseline, and (ii) MaxRL [Tajwar et al., 2026] as an alternative performant LM post-training method.



(a) Training time accuracy

(b) Evaluation pass@k on MATH500

Figure 2: *GEM* matches or excels performance of PG with mean reward baseline, MaxRL, and the 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). Details in Appendix.

5.1 Sample Efficiency of GEM for Math Reasoning

We first demonstrate how GEM enables greater online sample efficiency compared to standard methods. We construct a challenging mathematical reasoning setting with sparse rewards to test our central motivation that PG training can waste rare successes, and would struggle in cases where successful outputs are rare. We pick a setting where the base LM has a pass@1 rate of nearly 0, specifically using the Llama-3.2-3B base model [Grattafiori et al., 2024], and we train on MATH [Hendrycks et al., 2021] and evaluate using MATH500 [Lightman et al., 2023]. Specifically, we use the GEM+CE update where the sampling distribution over $P_{\mathcal{B}}$ is an recency-biased exponentially decaying distribution with an half-life of 10 iterations.

GEM+CE enables faster training by reusing previous successes. Figure 2 shows that our algorithm outperforms both PG and MaxRL in terms of both training reward and pass@k. These results support that maintaining an explicit memory of past successes is particularly beneficial in such "hard discovery" settings.

5.2 GEM+Flow is more robust to very off-policy data

Given the construction of GEM+Flow, we hypothesize that a primary benefit over GEM+CE is the ability to make use of more off-policy data, such as when we use samples from very old policies.

The FlowUpdate makes GEM more robust to off-policy data. To investigate, we begin by using the same MATH reasoning task as before, but instead of sampling using the recency-weighted exponential decay (as Figure 2 does), we instead sample *uniformly* the entire buffer of all previously found good solutions. This construction means that stale data in the buffer has an equal probability of being sampled.

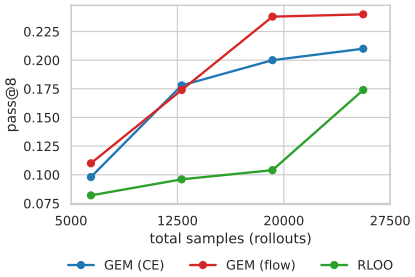


Figure 3: Eval Pass@8 of Llama on MATH. GEM achieves fast improvements, with the flow update providing slightly better learning.

We see in Figure 3 that the pass@8 performance of GEM+CE degrades as compared to when using recency-weighted sampling (which biases toward the current policy). However, when using GEM+Flow, we are able to get improvements in performance. It is worth noting that both GEM variants nevertheless learn more efficiently as compared to regular PG.

The FlowUpdate enables additional updates on stale data. While we have so far focused on the setting of having fresh samples per iteration of Algorithm 1, we further investigate whether updating purely with in-buffer data is possible when using GEM. This is desirable from an sample efficiency perspective. Here, we experiment with a MNIST bandit task, where we frame MNIST [LeCun et al., 1998, Deng, 2012] as a contextual bandit task, where each image $\mathbf{x}$ represents the context and each

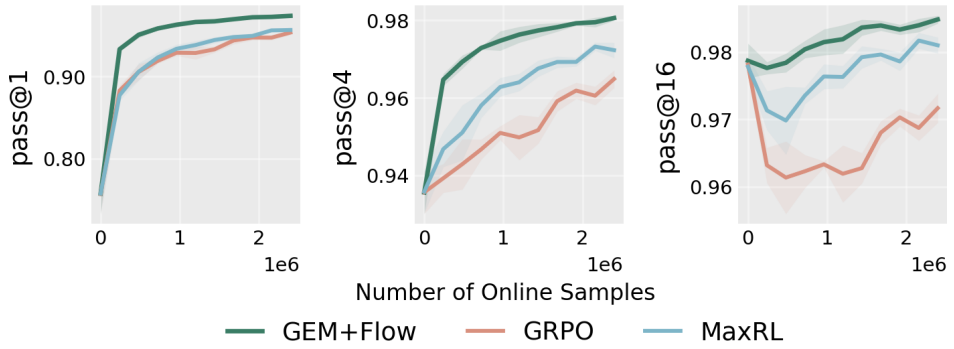


Figure 4: Number of online samples (generations) vs. pass@k for GEM+Flow, GRPO [Shao et al., 2024], and MaxRL [Tajwar et al., 2026] on the MNIST bandit task. Online samples denote newly generated model outputs used for training, excluding reused buffer samples. Curves show mean performance across 3 seeds, with shaded regions indicating standard deviation. GEM+Flow reaches higher pass@k with fewer online generations than GRPO or MaxRL.

answer in the answer space $\mathcal{A} = \{0, \ldots, c-1\}$ (with $c = 10$ by default) represents an arm. We train a ViT [Yuan et al., 2021] architecture.

Figure 4 shows how GEM can make use of additional updates on this MNIST bandit task. Specifically, GEM performs 5 buffer-samples-only gradient updates for each policy sampling. Empirically in this setting, 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 online samples, indicating improved sample efficiency.

6 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 provide proofs of generalized conditions for policy improvement which allow us to use more flexible classes of target policies for update that naturally make use of off-policy data. We also 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]. 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), suggesting another interesting avenue for future work.

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.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. *arXiv preprint arXiv:2510.18874*, 2025.

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

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

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 Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning*, pages 267–274, 2002.

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.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yuetai Li, Zhangchen Xu, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Bill Yuchen Lin, Xiang Yue, and Radha Poovendran. Temporal sampling for forgotten reasoning in llms. *arXiv preprint arXiv:2505.20196*, 2025.

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.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750, 2007.

Doina Precup, Richard S Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. *International Conference on Machine Learning*, 2000.

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.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

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.

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

Algorithm	C	$P_{\mathcal{B}}$
REINFORCE [Williams, 1992]	BK	$\mathcal{U}$
MaxRL [Tajwar et al., 2026]	BK	$1/V^{\pi_t}(\mathbf{x})$

Table 2: Algorithmic choices of GEM that recover various methods.

A A Unified Target-Matching View of Post-Training

This section expands on the claim that standard families of post-training algorithms are all instances of matching a reward-tilted target. We then use this perspective to show how different choices of GEM correspond to existing algorithms.

Definitions. Fix an input $\mathbf{x}$. Let $\pi(\mathbf{y} \mid \mathbf{x})$ denote the current policy and let $\pi'(\mathbf{y} \mid \mathbf{x})$ denote an updated policy. Define the expected reward,

$$V^{\pi}(\mathbf{x}) := \mathbb{E}_{\mathbf{y}\sim\pi(\cdot \mid \mathbf{x})}\left[R(\mathbf{x},\mathbf{y})\right]. \tag{8}$$

Unless stated otherwise, assume that $R(\mathbf{x},\mathbf{y}) > 0$ and that all distributions have compatible support. Define the reward-tilted distributions q, q' for π and π', respectively:

$$q(\mathbf{y} \mid \mathbf{x}) := \frac{\pi(\mathbf{y} \mid \mathbf{x})R(\mathbf{x},\mathbf{y})}{V^{\pi}(\mathbf{x})}, \qquad q'(\mathbf{y} \mid \mathbf{x}) := \frac{\pi'(\mathbf{y} \mid \mathbf{x})R(\mathbf{x},\mathbf{y})}{V^{\pi'}(\mathbf{x})}. \tag{9}$$

A.1 Connection to PG and EM

Proposition A.1 (Forward-KL matching of q recovers PG and EM)**.** *Treat the target q as fixed at the current iterate (a stop-gradient, as in the E-step of EM). Then the descent direction of the forward KL $D_{\mathrm{KL}}(q \,\|\, \pi_\theta)$ equals the policy-gradient ascent direction up to a per-input scaling $1/V^{\pi}(x)$:*

$$\nabla_\theta D_{\mathrm{KL}}\big(q \,\big\|\, \pi_\theta\big) = -\,\mathbb{E}_{\mu(x)}\,\mathbb{E}_{q(y \mid x)}\big[\nabla_\theta \log \pi_\theta(y \mid x)\big] \tag{10}$$

$$= -\,\mathbb{E}_{\mu(x)}\left[\frac{1}{V^{\pi}(x)}\,\mathbb{E}_{\pi(y \mid x)}\big[R(x,y)\,\nabla_\theta \log \pi_\theta(y \mid x)\big]\right]. \tag{11}$$

Proof. Since the target is fixed, $\nabla_\theta D_{\mathrm{KL}}(q\|\pi_\theta) = \nabla_\theta \mathbb{E}_q[\log q - \log \pi_\theta] = -\mathbb{E}_q[\nabla_\theta \log \pi_\theta]$, which is Equation (10). Substituting Equation (9) gives $\mathbb{E}_q[\,\cdot\,] = \frac{1}{V^{\pi}(x)}\mathbb{E}_\pi[R(x,y)\cdot\,]$, which is Equation (11). □

A.2 Recovery of Existing Methods

This perspective makes several post-training methods special cases of GEM, obtained by specific algorithmic choices. Table 2 summarizes the choices.

Recovery of REINFORCE. We recover REINFORCE [Williams, 1992] by restricting GEM to retain no historical data. Specifically, we set the buffer capacity to match the current iteration's sample size, $C = BK$, so the buffer distribution is the current policy's rollout distribution, π_t. By setting the target distribution $m(a|s)$ to be the reward-weighted empirical distribution of the current batch, the expected gradient update becomes Equation (2), confirming that REINFORCE is a special case of GEM.

Recovery of MaxRL. MaxRL [Tajwar et al., 2026]is a PG variant that maximizes the marginal likelihood of generating a correct solution. The exact gradient of the log-likelihood objective is:

$$\nabla_\theta \log V^{\pi_t}(\mathbf{x}) = \mathbb{E}_{\mathbf{y}\sim\pi_t(\cdot|\mathbf{x})}\left[\frac{R(\mathbf{x},\mathbf{y})}{V^{\pi_t}(\mathbf{x})}\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\right]. \tag{12}$$

We recover this objective within GEM by setting our per-sample weights to $1/V^{\pi_t}(\mathbf{x})$. Applying these weights to the on-policy samples constructs the target distribution:

$$m(\mathbf{y} \mid \mathbf{x}) = \frac{\pi_t(\mathbf{y} \mid \mathbf{x})R(\mathbf{x},\mathbf{y})}{V^{\pi_t}(\mathbf{x})}. \tag{13}$$

When GEM updates the policy toward this specific m using the forward KL gradient, the resulting expected update is identical to the MaxRL gradient.

A.3 Policy Update Equivalence

In this section we expand on the discussing that all existing work around LLM policy optimization, which operate in the contextual bandit setting, can be more simply viewed through the lens of iterative minimization of a forward KL divergence (i.e. "SFT update") towards the target distribution $q_t \propto \pi_t(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})$, by noting that for any positive reward setting, the forward KL update,

$$\nabla_\theta D_{\mathrm{KL}}\big(q_\theta||\pi_\theta\big) = \mathbb{E}_{\mu(\mathbf{x})}\left[\mathbb{E}_{q_\theta(\mathbf{y}|\mathbf{x})}\Big[\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\Big]\right], \quad q_\theta(\mathbf{y}|\mathbf{x}) = \frac{\pi_\theta(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_\theta)} \tag{14}$$

$$= \mathbb{E}_{\mu(\mathbf{x})}\left[\frac{1}{v(\mathbf{x},\pi_\theta)}\,\mathbb{E}_{\pi_\theta(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\nabla_\theta \log \pi_\theta(\mathbf{y}|\mathbf{x})\big]\right], \tag{15}$$

correspond to the standard policy gradient update [Williams, 1992, Schulman et al., 2017, Guo et al., 2025] up to a per-question scaling constant of $1/v(\mathbf{x},\pi_\theta)$. Alternatively, it corresponds exactly to the policy optimization methods based on expectation maximization which aim to directly increase the likelihood of success [Dayan and Hinton, 1997, Abdolmaleki et al., 2018b, 2024, Tajwar et al., 2026]. Further, in the binary reward case, q_θ is also precisely the distribution from filtering. In contrast, we explore along two alternative dimensions: (i) alternative target distributions beyond q_t by considering a mixture of historical policy distributions, and (ii) alternative update directions beyond the forward KL through using a Flow update toward a target.

B Policy Improvement

GEM instantiates the target-matching perspective. We train the policy by moving it toward a target P_M. We now show how GEM can provably yield policy improvement.

B.1 The GEM Target Improves on the Current Policy

As stated in Section 4.1, we require a target distribution P_M for π_θ to match. We now show how P_M has higher expected reward than the current policy.

Lemma. *For any state $\mathbf{x}$ in the buffer distribution ($P_{\mathcal{B}}(\mathbf{x}) > 0$), if buffer has a strictly higher expected reward, $\mathbb{E}_{P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\pi_t}[R(\mathbf{y},\mathbf{x})]$, then the conditional mixture distribution at $\mathbf{x}$ has a strictly higher expected reward than π_t for any $\alpha \in (0,1)$:*

$$\mathbb{E}_{P_M(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})]\,.$$

Proof of Lemma 4.1. We first note the conditional distribution $P_M(\mathbf{y}|\mathbf{x})$ is itself a mixture,

$$\begin{aligned}
P_M(\mathbf{y}|\mathbf{x}) = \frac{P_M(\mathbf{y},\mathbf{x})}{P_M(\mathbf{x})} &= \frac{\alpha\, P_{\mathcal{B}}(\mathbf{x},\mathbf{y}) + (1-\alpha)\, P_{\pi_t}(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{y}}\Big(\alpha\, P_{\mathcal{B}}(\mathbf{x},\mathbf{y}) + (1-\alpha)\, P_{\pi_t}(\mathbf{x},\mathbf{y})\Big)}\,,\\
&= \frac{\alpha\, P_{\mathcal{B}}(\mathbf{x})P_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + (1-\alpha)\,\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})}{\alpha P_{\mathcal{B}}(\mathbf{x})\sum_{\mathbf{y}} P_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + (1-\alpha)\mu(\mathbf{x})\sum_{\mathbf{y}}\pi_t(\mathbf{y}|\mathbf{x})}\,,\\
&= \frac{\alpha\, P_{\mathcal{B}}(\mathbf{x})P_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + (1-\alpha)\,\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})}{\alpha P_{\mathcal{B}}(\mathbf{x}) + (1-\alpha)\mu(\mathbf{x})}\\
&= w_\alpha(\mathbf{x})\, P_{\mathcal{B}}(\mathbf{y}|\mathbf{x}) + \big(1 - w_\alpha(\mathbf{x})\big)\pi_t(\mathbf{y}|\mathbf{x})\,,
\end{aligned}$$

with state-dependent mixture weights $w_\alpha(\mathbf{x}) = \alpha P_{\mathcal{B}}(\mathbf{x})/\big(\alpha P_{\mathcal{B}}(\mathbf{x}) + (1-\alpha)\mu(\mathbf{x})\big)$. By assumption, we have $P_{\mathcal{B}}(\mathbf{x}) > 0$ and $\alpha > 0$, which implies $w_\alpha(\mathbf{x}) > 0$. Taken together, the expected reward at state $\mathbf{x}$ is,

$$\begin{aligned}
\mathop{\mathbb{E}}_{P_M(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big] &= \sum_{\mathbf{y}\in\mathcal{Y}}\left[w_\alpha(\mathbf{x})\, P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y}) + \big(1 - w_\alpha(\mathbf{x})\big)\pi_t(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})\right],\\
&= w_\alpha(\mathbf{x}) \mathop{\mathbb{E}}_{P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big] + \big(1 - w_\alpha(\mathbf{x})\big) \mathop{\mathbb{E}}_{\pi_t(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big],\\
&> \mathop{\mathbb{E}}_{\pi_t(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big],
\end{aligned}$$

where the strict inequality in the final step follows directly from our assumption that $\mathbb{E}_{P_{\mathcal{B}}(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\pi_t}[R(\mathbf{y},\mathbf{x})]$. □

Proposition. *Assume a buffer distribution sharing non-zero support with μ, $\sum_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x})P_{\mathcal{B}}(\mathbf{x}) > 0$. Evaluating the conditional mixture distribution $P_M(\mathbf{y}|\mathbf{x})$ over μ yields a strict improvement over the current policy π_t for any $\alpha \in (0,1)$:*

$$\mathop{\mathbb{E}}_{\mu(\mathbf{x})P_M(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big] > \mathop{\mathbb{E}}_{\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big]. \tag{16}$$

Proof of Proposition 4.2. The expected reward of the joint distribution is,

$$\mathbb{E}_{\mu(\mathbf{x})P_M(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big] = \sum_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x}) \mathop{\mathbb{E}}_{P_M(\mathbf{y}|\mathbf{x})}\Big[R(\mathbf{x},\mathbf{y})\Big].$$

We evaluate this across two cases:

Case 1: For a state $\mathbf{x}$ no in the buffer distribution, $P_{\mathcal{B}}(\mathbf{x}) = 0$, we have $w_\alpha(\mathbf{x}) = 0$, leading to $P_M(\mathbf{y}|\mathbf{x}) = \pi_t(\mathbf{y}|\mathbf{x})$. Thus, the expected rewards are equal, $\mathbb{E}_{P_M(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})] = \mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})]$.

Case 2: For a state $\mathbf{x}$ in the buffer, $P_{\mathcal{B}}(\mathbf{x}) > 0$, we established from Lemma 4.1 the strict inequality: $\mathbb{E}_{P_M(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})] > \mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x},\mathbf{y})]$.

Taken together, as long as there exist at least one state where $\mu(\mathbf{x})P_{\mathcal{B}}(\mathbf{x}) > 0$, the global score over state distribution μ will have an expected reward that is strictly better for P_M,

$$\begin{aligned}
\mathbb{E}_{\mu(\mathbf{x})P_M(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big] &= \sum_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x})\Bigg[w_\alpha(\mathbf{x}) \mathop{\mathbb{E}}_{P_B(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big] + \big(1 - w_\alpha(\mathbf{x})\big) \mathop{\mathbb{E}}_{\pi_t(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big]\Bigg], \\
&> \sum_{\mathbf{x}\in\mathcal{X}} \mu(\mathbf{x})\Big[\mathop{\mathbb{E}}_{\pi_t(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big]\Big], \\
&= \mathbb{E}_{\mu(\mathbf{x})\pi_t(\mathbf{y}|\mathbf{x})}\big[R(\mathbf{x},\mathbf{y})\big].
\end{aligned}$$

□

B.1.1 ELBO Identity (used by EM and REINFORCE)

The target improving on π_t does not by itself imply that an update toward it will improve π_t (see Figure 1 for a concrete example of when reward can decrease). We now state an identity that characterizes the policy's one-step change in expected reward as the progress that the update makes toward P_M. We measure this progress in KL divergence, minus a correction that measures the distance from the target to the on-policy reference q (defined in Equation (9)). We first provide some useful definitions, then state the identity.

Definitions. Let $P_M(\mathbf{y} \mid \mathbf{x})$ be an arbitrary target distribution, e.g., a distribution induced by filtered samples, search, expert demonstrations, or replay. Define

$$\delta(\pi', \pi; \mathbf{x}) = \log \frac{V^{\pi'}(\mathbf{x})}{V^{\pi}(\mathbf{x})}. \tag{17}$$

A new policy π' is an improvement over π in state $\mathbf{x}$ if $\delta(\pi', \pi; \mathbf{x}) > 0$.

Proposition B.1. *We can get per-state policy improvement by improving the lowerbound,*

$$\log \frac{v(\mathbf{x}, \pi')}{v(\mathbf{x}, \pi_t)} \geq \mathbb{E}_{P_M}\left[\log \frac{R(\mathbf{x},\mathbf{y})}{v(\mathbf{x}, \pi_t)}\right] - D_{\mathrm{KL}}(P_M \,\|\, \pi'), \tag{18}$$

with Jensen gap $D_{\mathrm{KL}}(P_M \,\|\, q')$.

Proof. Just some algebra.

$$\log \frac{v(\mathbf{x}, \pi')}{v(\mathbf{x}, \pi_t)} = \mathbb{E}_{P_M}\left[\log \frac{v(\mathbf{x}, \pi')}{v(\mathbf{x}, \pi_t)}\right] = \mathbb{E}_{P_M}\left[\log \frac{\pi'(\mathbf{y}|\mathbf{x})\, R(\mathbf{x},\mathbf{y})}{v(\mathbf{x}, \pi_t)\, q'(\mathbf{y}|\mathbf{x})}\right], \tag{19}$$

$$= \mathbb{E}_{P_M}\left[\log \frac{\pi'(\mathbf{y}|\mathbf{x})\, R(\mathbf{x},\mathbf{y})\, P_M(\mathbf{y}|\mathbf{x})}{v(\mathbf{x}, \pi_t)\, P_M(\mathbf{y}|\mathbf{x})\, q'(\mathbf{y}|\mathbf{x})}\right], \tag{20}$$

$$= \mathbb{E}_{P_M}\left[\log \frac{\pi'(\mathbf{y}|\mathbf{x})\, R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)\, P_M(\mathbf{y}|\mathbf{x})}\right] + \mathbb{E}_{P_M}\left[\log \frac{P_M(\mathbf{y}|\mathbf{x})}{q'(\mathbf{y}|\mathbf{x})}\right], \tag{21}$$

$$= \mathbb{E}_{P_M}\left[\log \frac{R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)}\right] - \mathbb{E}_{P_M}\left[\log \frac{P_M(\mathbf{y}|\mathbf{x})}{\pi'(\mathbf{y}|\mathbf{x})}\right] + \mathbb{E}_{P_M}\left[\log \frac{P_M(\mathbf{y}|\mathbf{x})}{q'(\mathbf{y}|\mathbf{x})}\right], \tag{22}$$

$$= \underbrace{\mathbb{E}_{P_M}\Big[\log R(\mathbf{x},\mathbf{y})\Big] - \log v(\mathbf{x},\pi_t)}_{\text{Reward improvement}} - \underbrace{D_{\mathrm{KL}}\Big(P_M||\pi'\Big)}_{\text{Approx error}} + \underbrace{D_{\mathrm{KL}}\Big(P_M||q'\Big)}_{\text{Jensen gap}}. \tag{23}$$

□

The lowerbound is tight (Jensen = 0) when the target is the reward-tilted distribution,

$$P_M(\mathbf{y}|\mathbf{x}) = q'(\mathbf{y}|\mathbf{x}) = \frac{\pi'(\mathbf{y} \mid \mathbf{x}) R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi')}, \tag{24}$$

and we can show why this is a nice target.

Corollary B.2. *When the target distribution is $P_M = q_t \propto \pi_t(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})$, we get policy improvement when,*

$$\log \frac{v(\mathbf{x},\pi')}{v(\mathbf{x},\pi_t)} \geq D_{\mathrm{KL}}\big(q_t||\pi_t\big) - D_{\mathrm{KL}}\big(q_t||\pi'\big), \tag{25}$$

which is tight at $\pi' = \pi_t$ (zero Jensen gap).

Proof. We start from Equation (18) and substitute $P_M = q_t$,

$$\begin{aligned}
\log \frac{v(\mathbf{x},\pi')}{v(\mathbf{x},\pi_t)} &\geq \mathbb{E}_{q_t}\left[\log \frac{R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)}\right] - D_{\mathrm{KL}}(q_t \,\|\, \pi'), \\
&= \mathbb{E}_{q_t}\left[\log \frac{R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)}\right] - \mathbb{E}_{q_t}\left[\log \frac{\pi_t(\mathbf{y}|\mathbf{x})R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)\pi'(\mathbf{y}|\mathbf{x})}\right], \\
&= \mathbb{E}_{q_t}\left[\log \pi'(\mathbf{y}|\mathbf{x}) - \log \pi_t(\mathbf{y}|\mathbf{x})\right], \\
&= D_{\mathrm{KL}}\big(q_t||\pi_t\big) - D_{\mathrm{KL}}\big(q_t||\pi'\big).
\end{aligned}$$

□

Corollary B.2 implies that if we start at π_t, *any* change in π' that takes it closer to the target q_t will guarantee policy improvement. Indeed, it is worth noting that *all* reinforcement learning methods including variants of the score function / REINFORCE gradient [Williams, 1992, Schulman et al., 2017, Guo et al., 2025], expectation maximization [Dayan and Hinton, 1997, Peters and Schaal, 2007, Abdolmaleki et al., 2018a,b, 2024, Tajwar et al., 2026], or fully on-policy correctness filtered SFT optimize toward the target q_t.

B.1.2 A more general target and improvement bounds

We now consider a more general setting and analyze arbitrary target distributions instead of q_t. The hope here is that this will allow us to view "off-policy" learning in a fundamentally new way: not by how to better approximate updates towards target q_t, but by asking which targets (and/or update) can still guarantee improvement, without as much reliance for on π_t.

Differences in KL Differences We will try to expand out Equation 22 in a few other ways that could be more useful to intuit about. We start by noting that from the definition of q_t we have,

$$\frac{R(\mathbf{x},\mathbf{y})}{v(\mathbf{x},\pi_t)} = \frac{q_t(\mathbf{y}|\mathbf{x})}{\pi_t(\mathbf{y}|\mathbf{x})}. \tag{26}$$

Plugging this into the first term in Equation 22 gives us a difference-of-KLs,

$$\log \frac{v(\mathbf{x},\pi')}{v(\mathbf{x},\pi_t)} = \Big[D_{\mathrm{KL}}(P_M \,\|\, \pi_t) - D_{\mathrm{KL}}(P_M \,\|\, \pi')\Big] - \Big[D_{\mathrm{KL}}(P_M \,\|\, q_t) - D_{\mathrm{KL}}(P_M \,\|\, q')\Big], \tag{27}$$

which gives us a condition for guaranteeing policy improvement,

$$\underbrace{D_{\mathrm{KL}}(P_M \,\|\, \pi_t) - D_{\mathrm{KL}}(P_M \,\|\, \pi')}_{\text{how much is } \pi' \text{ closer to } P_M \text{ vs. } \pi_t} > \underbrace{D_{\mathrm{KL}}(P_M \,\|\, q_t) - D_{\mathrm{KL}}(P_M \,\|\, q')}_{\text{how much is } q' \text{ closer to } P_M \text{ vs. } q_t}. \tag{28}$$

Or, alternatively,

$$\underbrace{D_{\mathrm{KL}}(P_M \,\|\, \pi_t) - D_{\mathrm{KL}}(P_M \,\|\, \pi')}_{\text{Contraction toward target from } \pi_t} > \underbrace{D_{\mathrm{KL}}(P_M \,\|\, q_t)}_{\text{Penalty away from } q_t} - \underbrace{D_{\mathrm{KL}}(P_M \,\|\, q')}_{\geq 0} \geq D_{\mathrm{KL}}(P_M \,\|\, q_t) \ . \tag{29}$$

Equation 29 is saying something like we need to find a π' that is sufficiently close to P_M, to overcome any difference between not choosing q_t as P_M. Note that for the choice of $P_M = q_t$ the R.H.S. is just 0, and we recover the results in Corollary B.2.

B.2 A Sufficient Condition for the Cross-Entropy Update

We provide an additional theoretical result that lower-bounds *how far* an updated training policy π_{t+1} needs to move along the path provided by the forward KL (i.e., SFT) gradient $\nabla_\theta D_{\mathrm{KL}}(P_M || \pi_\theta)$ until policy improvement over the previous policy π_t can be guaranteed.

Proposition B.3. *Let $R(\mathbf{x}, \mathbf{y}) > 0$ be a strictly positive reward function. An updated policy π_{t+1} is a strict pointwise improvement over π_t ($\mathbb{E}_{\pi_{t+1}(\mathbf{y}|\mathbf{x})}[R(\mathbf{x}, \mathbf{y})] > \mathbb{E}_{\pi_t(\mathbf{y}|\mathbf{x})}[R(\mathbf{x}, \mathbf{y})]$) if the SFT objective's improvement satisfies the following inequality (omitting conditioning on $\mathbf{x}$ for brevity):*

$$D_{\mathrm{KL}}\big(P_M \,\|\, \pi_t\big) - D_{\mathrm{KL}}\big(P_M \,\|\, \pi_{t+1}\big) > D_{\mathrm{KL}}\big(P_M \,\|\, q_t\big) \ ,$$

where q_t is a reward-tilted version of π_t, defined as:

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

The above states that when optimizing with SFT using samples from arbitrary buffer target P_M, we must reduce the loss by at least $D_{\mathrm{KL}}(P_M \,\|\, q_t)$ to guarantee that the updated policy outperforms π_t. This is a generalization of a classical policy improvement guarantee from Dayan and Hinton [1997]. Indeed, it can be shown that vanilla policy gradient reduces to setting the target $P_M = q_t$, which has the benefit that *any* reduction in loss trivially yields policy improvement, but enforces a restrictive target distribution that only allows using samples drawn from the most recent policy π_t. Relaxing this sampling restriction is precisely what our buffer-based algorithm aims to achieve.

We provide the full derivations and an extensive discussion in Appendix **??**. As demonstrated in our experiments, employing a simple recency-weighted sampling scheme over the replay buffer (which has the effect of concentrating mass toward q_t) already results in a highly performant algorithm.

B.3 The Flow Update Provides Monotonic Improvement

The Flow update specifies the trajectory in probability space.

Proposition B.4. *Fix an iteration t of the outer loop of Algorithm 1. Now, $\pi_t(\cdot \mid \mathbf{x})$ denotes the policy at iteration t. Consider the flow in probability space: $\dot{p} := m - \pi_t$. Let $\tau \geq 0$ be progress along the update from π_t to m. Given a softmax parameterization, we show the following. If m yields expected reward at least as high as π_t, such that $\mathbb{E}_{\mathbf{y} \sim m(\cdot \mid \mathbf{x})}[R(\mathbf{x}, \mathbf{y}])$ $\geq$ $\mathbb{E}_{\mathbf{y} \sim \pi_t^\tau(\cdot \mid \mathbf{x})}[R(\mathbf{x}, \mathbf{y})]$, then $\frac{dV^{\pi_t^\tau}}{d\tau} \geq 0, \forall \tau$.*

Proof of Proposition B.4. We start by explicitly mapping the probability simplex to the parameter space θ. 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 V using the chain rule over the policy parameters θ as:

$$\frac{dV^{\pi_t^\tau}}{d\tau} = \sum_{i=1}^{K-1} \frac{\partial V^{\pi_t^\tau}}{\partial \theta_i} \dot{\theta}_i.$$

We now plug in the trajectory of the probability distribution, $\dot{p} := m - \pi_t$.

$$\frac{dV^{\pi_t^\tau}}{d\tau} = \sum_{i=1}^{K} m_i \big(R(\mathbf{x}, \mathbf{y}_i) - V^{\pi_t^\tau}(\mathbf{x})\big).$$

By the assumption that $\sum_j m_j \big(R(\mathbf{x}, \mathbf{y}_i) - V^{\pi_t^\tau}(\mathbf{x})\big) \geq 0$, it follows 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 3: Default training hyperparameters for all MNIST experiments.

C Experiment Details

C.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:

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

C.2 Didactic Bandit Problem

We use a minimal contextual bandit problem to visualize how PG (REINFORCE with a mean-reward baseline), CE, and Flow updates move a policy toward a target, and why their reward profiles differ. This didactic task is a simple 3-action, binary reward bandit problem with a single context, in which 2 actions induce non-zero reward. The policy $\pi_\theta = \text{softmax}(z)$ is parameterized by three logits $z \in \mathbb{R}^3$. The configuration in Figure 1 uses initial logits $z_0 = (2.0, -5.0, 0.0)$, target $m = (0.2, 0.79, 0.01)$, learning rate 0.05, and 600 steps.

C.3 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 4 were generated with the hyperparameters in Table 3. 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.

C.4 Math Reasoning

Hyperparameters. Table 4 lists the hyperparameters for the mathematical reasoning tasks.

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 4: Setup and data for math reasoning.