Jackpot: Optimal Budgeted Rejection Sampling for Extreme Actor-Policy Mismatch Reinforcement Learning

Anonymous authors

Reinforcement learning (RL) for large language models (LLMs) remains expensive, particularly because the rollout is expensive. Decoupling rollout generation from policy optimization (e.g., leveraging a more efficient model to rollout) could enable substantial efficiency gains, yet doing so introduces severe distribution mismatch that destabilizes learning. We propose JACKPOT, a framework that leverages Optimal Budget Rejection Sampling (OBRS) to directly reduce the discrepancy between the rollout model and the evolving policy. JACKPOT integrates a principled OBRS procedure, a unified training objective that jointly updates the policy and rollout models, and an efficient system implementation enabled by top-k probability estimation and batch-level bias correction. Our theoretical analysis shows that OBRS consistently moves the rollout distribution closer to the target distribution under a controllable acceptance budget. Empirically, JACKPOT substantially improves training stability compared to importance-sampling baselines, achieving performance comparable to on-policy RL when training Qwen3-8B-Base for up to 300 update steps. Taken together, our results show that OBRS-based alignment brings us a step closer to practical and effective decoupling of rollout generation from policy optimization for RL for LLMs.

1 Introduction

Reinforcement learning (RL) has demonstrated substantial effectiveness in the post-training of large language models (LLMs), yielding significant improvements in domains such as mathematics (Guo et al., 2025; Azerbayev et al., 2023), coding (Jimenez et al., 2023; Ouyang et al., 2025), and other agentic tasks (Liu et al., 2023; Jin et al., 2025). Despite these successes, RL remains expensive (Sheng et al., 2025; Fu et al., 2025; Zheng et al., 2025c), with the majority of the training cost, often 80% (Qin et al., 2025), attributed to rollouts, during which LLMs auto-regressively generate trajectories.

Many approaches have been explored to reduce rollout costs by improving hardware utilization through asynchronous training (Zheng et al., 2025b; Fu et al., 2025; Wu et al., 2025), or through inference optimizations such as 8-bit quantization (Liu et al., 2025b), request balancing (Qin et al., 2025), and speculative decoding (Together-AI). However, these methods all require the target LLM to actively participate in the rollout process to collect trajectories and signals, which ultimately limits their flexibility (Kiran et al., 2021) and efficiency. This leads to a question:

Is it possible to perform rollouts using a completely different model from the one we ultimately want to train?

Ideally, rollout models should be more efficient, such as smaller variants from the same model family. By prior work on test-time compute (Brown et al., 2024; Sadhukhan et al., 2025), even relatively small models can obtain non-zero rewards (i.e., training signals) on challenging problems by leveraging multiple attempts, a behavior that is inherently aligned with the standard RL training paradigm (Guo et al., 2025).

Despite the availability of training signals, the decoupled RL training triggers a severe **distribution mismatch** between the rollout and trained models, which is known to severely affect the stability and convergence of RL (Liu et al., 2025a), primarily due to inaccurate advantage estimates. Existing methods aim to mitigate this issue with various importance-sampling (IS) corrections (Fu et al., 2025; Wu et al., 2025; Liu et al., 2025b; Zheng et al., 2025a; Team et al., 2025). However, we find that the KL divergence between *different* models is often an order of magnitude larger than that observed in asynchronous training or quantised variants, casting

doubt on whether prior correction techniques remain effective under such a large mismatch.

Given the limitations of purely post-hoc corrections, it is desirable also to reduce the mismatch *at the source*. Rejection sampling, which simulates a target distribution from an accessible proposal distribution, offers such a direct mechanism by selectively excluding undesired tokens from contributing to the backward pass and policy updates. It effectively narrows the distribution gap between the rollout and training models, naturally *complementing* existing importance-sampling corrections.

However, applying rejection sampling in RL systems for LLMs poses three key technical challenges. ① **Low Sample Efficiency.** Let $\boldsymbol{p}$ denote the trained model distribution and $\boldsymbol{q}$ denote the rollout model distribution. In standard rejection sampling, a token i proposed from $\boldsymbol{q}$ is accepted with probability $\frac{p_i}{\lambda q_i}$, where the constant λ must satisfy $\lambda \geq \max_i \frac{p_i}{q_i}$. For LLMs, which operate over vocabularies exceeding 100,000 tokens, this requirement leads to an extremely large λ because even small local differences between the two distributions can cause the ratio $\frac{p_i}{q_i}$ to spike on certain rare tokens. Consequently, the acceptance probability becomes vanishingly small for almost all proposed tokens. ② **Widening Gap.** A naïve implementation of decoupled training keeps the rollout model fixed while the trained model continually improves. As learning progresses, the discrepancy between the two models increases, further exacerbating the distribution mismatch and potentially destabilising training. ③ **Efficiency and System Support.** Current RL frameworks (Sheng et al., 2025; Fu et al., 2025; von Werra et al., 2020; Hu et al., 2024) assume that the rollout and training models are identical. It remains unclear how to efficiently support decoupled RL, especially when combined with new algorithms, objectives, or loss functions.

To tackle these challenges, we propose JACKPOT. Our framework leverages optimal budget rejection sampling (OBRS) (Verine et al., 2024) as a relaxed alternative to classical rejection sampling. Although OBRS does not enforce exact equality between the actor and target policy distributions, it provides a provable guarantee that, for any specified rejection budget, the adjusted actor distribution becomes strictly closer to the target distribution than the original proposal. To prevent the distribution gap from widening as training progresses, we further apply a reverse-KL loss to progressively align the rollout model with the trained model. In addition, we develop an end-to-end RL system that efficiently supports decoupled training, incorporating several approximations to reduce the memory overhead introduced by OBRS, which otherwise requires accessing the full vocabulary when reweighting accepted tokens. Taken together, our empirical results show that JACKPOT substantially improves the stability of RL training compared to IS-based baselines, even enabling performance comparable to on-policy training on QWEN3-8B-BASE for 300 training steps (Figure 1).

We organize the remainder of this paper as follows.

- In Section 2, we formalize the notion of distribution mismatch between rollout and trained models, analyze its underlying causes across different training paradigms, and review relevant prior work.
- In Section 3, we start from a brief introduction of optimal budget rejection sampling (Section 3.1) and demonstrate its effectiveness through both numerical simulations and empirical observations (Section 3.2) and present the theoretical optimality (Section 3.3).
- In Section 4, we present the full JACKPOT framework, detailing three key components: (i) the OBRS procedure (Section 4.1); (ii) the formulation of the JACKPOT training objective (Section 4.2); and (iii) an efficient system implementation enabled by Top-k-based probability estimation and batch-level bias correction (Section 4.4).
- In Section 5, we conduct experiments on mathematical reasoning tasks to evaluate JACKPOT. Empirically, we show that JACKPOT significantly improves the stability of RL training over IS-based baselines. We further perform ablations on each module in JACKPOT and analyze its effectiveness in settings with smaller distribution gaps, such as large-batch training and KV-quantized rollout.

2 Background

In this section, we first formalize the distribution mismatch problem that arises in RL for LLMs. We then review several strands of related work of JACKPOT.

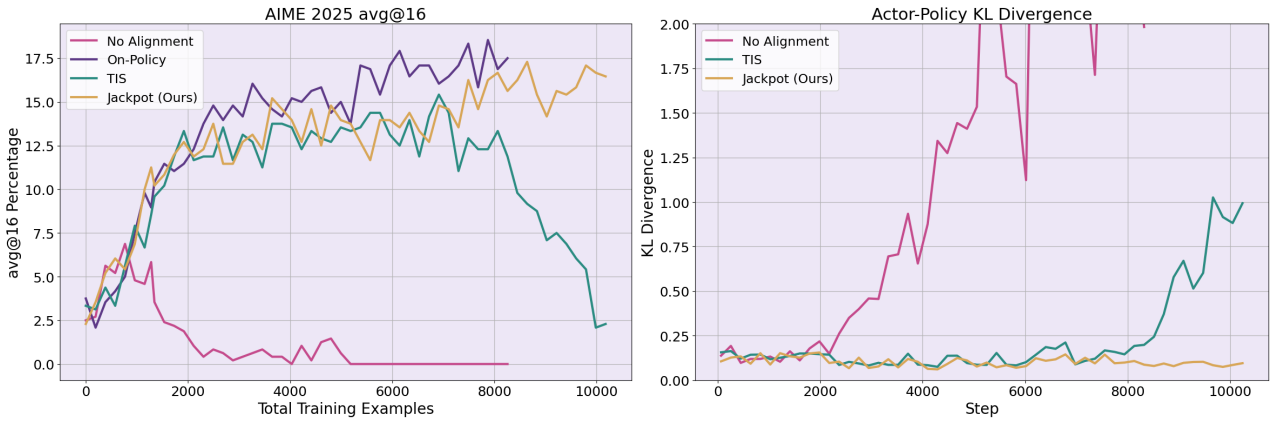

Figure 1 RL training requires actor-policy maintaining strong probability distribution alignment. When actor and policy aren't aligned, they will result in training collapse. Here we show training setting use a QWEN3-1.7B-BASE model training rollout to train a QWEN3-8B-BASE model policy. Without any alignment procedures, training collapses (pink). Prior method TIS (green) also shows a significant gap towards QWEN3-8B-BASE on-policy baseline (purple), while collapsing, using TIS sees KL divergence also violently increasing. Our proposed method, Jackpot (yellow) maintains small KL divergence between actor and policy model probability distribution, while showing stable and competitive training convergence to on-policy setting.

2.1 Problem Setting: PPO Objective and Actor-Policy Distribution Mismatch

We begin with the clipped objective in PPO (Schulman et al., 2017), whose expectation can be written as

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_{x\sim P_{\text{inf}}}\Big[\min\big(r_\theta(x)\hat{A}(x), \text{clip}(r_\theta(x), 1-\epsilon, 1+\epsilon)\hat{A}(x)\big)\Big] \tag{1}$$

where $r_\theta(x) = p_{\boldsymbol{\theta}_{\text{new}}}(x)/p_{\text{ref}}(x)$ is the likelihood ratio between the updated policy $p_{\boldsymbol{\theta}_{\text{new}}}$ and the reference policy p_{ref}, and $\hat{A}(x)$ denotes the estimated advantage at decision x. p_{inf} is the inference distribution used to generate rollouts, p_{ref} is the reference policy distribution assumed in the objective, and $p_{\boldsymbol{\theta}_{\text{new}}}$ is the updated policy distribution. In the standard process, it is assumed that $p_{\text{inf}} = p_{\text{ref}}$, but in practice this assumption is often violated, leading to actor–policy distribution mismatch.

Distribution mismatch is common and arises for several reasons, such as minor discrepancies between the inference engine and the reference policy by FSDP engines, the use of stale or asynchronous data, or rollouts generated by approximated models (e.g., quantized, sparsified, or distilled). Such mismatches can destabilize training and therefore require additional mechanisms to correct or mitigate their impact.

2.2 Related Work

RL for LLM. Reinforcement learning has been widely applied to LLMs to improve human alignment, reasoning, coding, and other complex tasks. Beyond PPO, memory efficient methods have been proposed, including ReMax (Li et al., 2023), RLOO (Ahmadian et al., 2024), and GRPO (Shao et al., 2024). In addition, methods such as SimPO (Meng et al., 2024) and DPO (Rafailov et al., 2023), which are based on offline RL, have also been employed for human alignment. RL training systems for LLMs, such as Verl (Sheng et al., 2025), AReal (Fu et al., 2025), TRL (von Werra et al., 2020), and OpenRLHF (Hu et al., 2024), have been developed to improve training throughput and scalability.

Distribution Mismatch Correction in RL. Actor (i.e., the rollout model) and policy (i.e., the trained model) mismatch is a common problem that has long been studied, e.g., Espeholt et al. (2018). To alleviate the actor-policy distribution gap, prior methods leverage importance sampling to approximate the true PPO objective.

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_{x\sim p_{\text{inf}}}\Big[\text{SG}\Big(\boldsymbol{F}\big(\tfrac{p_{\text{ref}}(x)}{p_{\text{inf}}(x)}\big)\Big)\min\big(r_\theta(x)\hat{A}(x), \text{clip}(r_\theta(x), 1-\epsilon, 1+\epsilon)\hat{A}(x)\big)\Big] \tag{2}$$

SG means stop gradients. The choice of adjustment function $\boldsymbol{F}$ is one of the core focuses of prior work. Areal (Fu et al., 2025) uses an identical function $\boldsymbol{F}(x) = x$; Flash-RL and Llama-RL use $\boldsymbol{F}(x) = \min(x, C)$ (i.e., truncated importance sampling, TIS), where C is a hyper-parameter; IceProp (Team et al., 2025) uses a bi-directional

truncation,

$$\boldsymbol{F}(x) = \begin{cases} x, & \text{if } x \in [\alpha, \beta], \\ 0, & \text{otherwise.} \end{cases}$$

From system perspective, FP32 LM heads (Liu et al., 2025b) and deterministic LLM Inference (He and Lab, 2025) are implemented to mitigate the numerical issue of serving systems when rollout.

In this paper, we proposed JACKPOT. Our method is **orthogonal** to the above prior works. We think about the problem of how to directly close the gap between p_{inf}, and p_{ref}. We will show that through optimal budget rejection sampling and reweighting of the output probabilities so that the divergence between p_{inf} and the target distributions p_{ref} is provably reduced. Moreover, techniques such as TIS can be applied on top of this improved distribution to further correct the remaining mismatch in a complementary way.

3 Rejection Sampling and Optimal Budget Rejection Sampling

In this section, we briefly introduce optimal budget importance sampling (OBRS) and discuss its theoretical guarantee, followed by our validation via numerical simulation.

3.1 Reduce Distribution Gap via Optimal Budget Importance Sampling

We show how to directly modify p_{inf} to close the distribution gap to p_{ref} instead of the post-hoc importance sampling corrections. Let $\boldsymbol{p}$ denote the target distribution we want to align with (e.g., the trained model distribution), $\boldsymbol{q}$ denote the proposed distribution (e.g., the rollout model distribution), and $\tilde{\boldsymbol{q}}$ denote the *post-rejection distribution* (i.e., the distribution of accepted tokens).

Definition 3.1 (Rejection Sampling (RS))**.** RS stochastically rejects tokens in the trajectories sampled with p_{inf} based on the difference between the two distributions. In standard rejection sampling, a token i proposed from $\boldsymbol{q}$ is accepted with probability $\frac{p_i}{\lambda q_i}$, where the constant λ must satisfy $\lambda \geq \max_i \frac{p_i}{q_i}$. Therefore, we have $\tilde{q}_i \propto q_i \cdot \frac{p_i}{\lambda q_i}$. Then $\tilde{q}_i = p_i$, after normalization. Therefore, rejection sampling transforms samples from $\boldsymbol{q}$ into exact samples from the target distribution $\boldsymbol{p}$.

While standard rejection sampling can, in principle, perfectly align two distributions, its direct application in our setting is impractical. In high-dimensional discrete spaces such as LLM vocabularies—often exceeding 100,000 tokens —even minor local discrepancies between the rollout and target distributions can cause the likelihood ratio $\frac{p_i}{q_i}$ to spike for rare tokens. This forces the normalizing constant λ to become extremely large, which, in turn, drives the acceptance rate to near zero, resulting in almost all proposed tokens being rejected.

To overcome this, we adopt the principled approach of **Optimal Budgeted Rejection Sampling (OBRS)** (Verine et al., 2024). This technique reframes the problem: instead of demanding perfect adherence to the target distribution at the cost of sample efficiency, it seeks the optimal rejection rule that, for a given target acceptance rate (a "budget"), produces a distribution as close as possible to the target. This is precisely the trade-off our problem requires.

Definition 3.2 (Optimal Budget Rejection Sampling (OBRS) (Verine et al., 2024))**.** Instead of using the dominating constant $\lambda = \max_i \frac{p_i}{q_i}$, OBRS selects a smaller user-specified parameter $\lambda > 0$ that reflects the desired rejection budget. For a proposed token $i \sim \boldsymbol{q}$, OBRS accepts it with probability $a_i = \min\left(1, \frac{p_i}{\lambda q_i}\right)$. The resulting post-rejection distribution is therefore $\tilde{q}_i \propto q_i \cdot a_i$

While the resulting distribution of OBRS does not generally equal $\boldsymbol{p}$ but is guaranteed to be *closer* to $\boldsymbol{p}$ than the original proposal distribution for any λ. Thus, OBRS provides a controllable trade-off between acceptance rate and distribution alignment, avoiding the vanishing acceptance rates that arise in high-dimensional settings.

3.2 Numerical Simulation

We validate the effectiveness of OBRS via numerical simulation in Figure 2. Crucially, this calibration is highly efficient; the acceptance rate remains high even when there is a large initial KL divergence. The impact on distributional alignment is dramatic: a significant reduction in KL divergence is observed with high acceptance rates. By

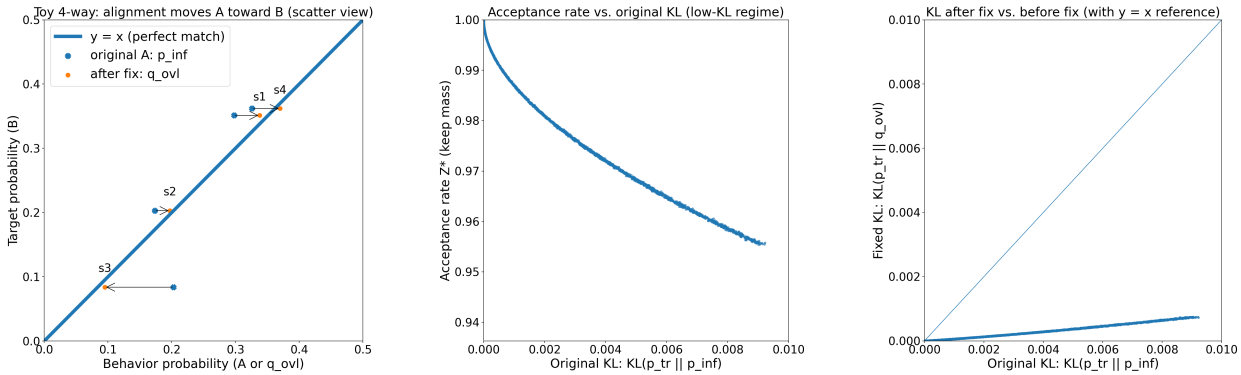

Figure 2 OBRS calibration results across three views: (a) per-token probability-ratio clipping pulls the model distribution toward the target, (b) acceptance remains high ($\approx 95\%$) even at large initial KL, and (c) overall KL is reduced by roughly an order of magnitude.

systematically damping the most extreme probability ratios, OBRS produces a distribution that is not only provably closer to the on-policy target but also primed to yield more stable and effective PPO/GRPO policy updates.

3.3 Theoretical Guarantees

To justify the use of OBRS within our framework, we establish its fundamental theoretical properties, which formally characterize its ability to reduce distribution mismatch. OBRS possesses proven optimality.

Theorem 3.3 (OBRS Improves Distribution Alignment)**.** *Let $\boldsymbol{p}$ be the target distribution and $\boldsymbol{q}$ be the proposal distribution. For any $\lambda > 0$, define the OBRS acceptance rule*

$$a_i = \min\left(1, \frac{p_i}{\lambda q_i}\right), \qquad \tilde{q}_i \propto q_i a_i.$$

Then the post-rejection distribution $\tilde{\boldsymbol{q}}$ is strictly closer to $\boldsymbol{p}$ than the original proposal $\boldsymbol{q}$ in the sense that

$$D_{\mathrm{KL}}(\boldsymbol{p} \| \tilde{\boldsymbol{q}}) \leq D_{\mathrm{KL}}(\boldsymbol{p} \| \boldsymbol{q}),$$

whenever $\lambda < \max_i \frac{p_i}{q_i}$.

In other words, OBRS always moves the proposal distribution toward the target distribution under any nontrivial rejection budget.

Theorem 3.4 (Optimality of OBRS under a Fixed Acceptance Budget)**.** *For any desired average acceptance rate $\bar{a} \in (0,1]$, there exists a unique scaling factor $\lambda > 0$ such that the OBRS acceptance rule*

$$a_i = \min\left(1, \frac{p_i}{\lambda q_i}\right)$$

achieves the exact acceptance budget:

$$\sum_i q_i a_i = \bar{a}.$$

Moreover, among all acceptance rules $a_i \in [0,1]$ that satisfy this constraint, the OBRS rule is the unique minimizer of the divergence to the target distribution:

$$\tilde{\boldsymbol{q}} = \operatorname*{argmin}_{\hat{\boldsymbol{q}}} D_{\mathrm{KL}}(\boldsymbol{p} \| \hat{\boldsymbol{q}}), \quad s.t. \quad \hat{q}_i \propto q_i a_i, \sum_i q_i a_i = \bar{a}.$$

Thus, OBRS is provably optimal for aligning $\boldsymbol{q}$ toward $\boldsymbol{p}$ under any specified rejection budget.

Formal proofs of the theorems are provided in Verine et al. (2024). This guarantee ensures we are using the provably best method for trading sample efficiency for distributional accuracy.

4 Jackpot: Design and Methodology

In this section, we present details on the design considerations of JACKPOT. We show how we apply OBRS in RL, i.e., the token rejection criteria and reweighting procedures in Section 4.1, the formulation of JACKPOT objective in Sections 4.2 and 4.3, and our memory optimization and efficient implementation in Section 4.4.

4.1 OBRS Procedure

To reduce the mismatch between the inference distribution p_{inf} and the target distribution p_{target} (which may refer to either the reference policy p_{ref} or the updated policy $p_{\boldsymbol{\theta}_{\text{new}}}$; see Section 4.3 for details), we adopt a rejection rule analogous to Leviathan et al. (2023). For a token x sampled from p_{inf}, we accept it with probability

$$a(x) = \min\left(1, \frac{p_{\text{target}}(x)}{\lambda p_{\text{inf}}(x)}\right), \tag{3}$$

where $\lambda > 0$ controls the overall rejection budget.

Equation (3) specifies the conditional probability

$$P(x \text{ accepted} \mid x \sim p_{\text{inf}}).$$

Tokens that are rejected are masked out and excluded from the loss computation and gradient propagation. The acceptance rule induces a new (post-rejection) distribution over tokens:

$$P_{\text{OBRS}}(x) = \frac{p_{\text{inf}}(x)a(x)}{\sum_{x'} p_{\text{inf}}(x')a(x')} = \frac{\min\left(p_{\text{inf}}(x), \frac{p_{\text{target}}(x)}{\lambda}\right)}{Z}. \tag{4}$$

where $Z = \sum_{x'} \min\left(p_{\text{inf}}(x'), \frac{p_{\text{target}}(x')}{\lambda}\right)$.

Importantly, P_{OBRS} represents the *true* probability distribution governing the accepted samples. Therefore, in subsequent training, we reweight all accepted tokens according to P_{OBRS}, ensuring that their contributions to the loss and gradient faithfully reflect their probabilities under the adjusted distribution. We visualize the process in Figure 3.

4.2 Formulations of the Jackpot Objective

JACKPOT optimizes three objectives during training: (i) an OBRS-adjusted RL loss for the trained model (i.e., policy) model θ, (ii) a standard PPO loss for the rollout model (i.e., actor) ω, and (iii) an on-policy distillation loss that keeps the rollout model aligned with the improving policy. We now describe each component.

(1) OBRS-adjusted RL loss for the policy model. Following Section 4.1, tokens are sampled from the rollout inference distribution p_{inf}, but each sampled token x is accepted or rejected via the OBRS acceptance probability

$$a(x) = \min\left(1, \frac{p_{\text{target}}(x)}{\lambda p_{\text{inf}}(x)}\right), \qquad \text{Mask}(x) \sim \text{Bernoulli}(a(x)).$$

Accepted tokens ($\text{Mask}(x) = 1$) define the OBRS-adjusted distribution

$$p'_{\text{inf}}(x) = \frac{p_{\text{inf}}(x)a(x)}{\sum_{x'} p_{\text{inf}}(x')a(x')}.$$

The original PPO objective in Section 2,

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_{x \sim p_{\text{inf}}}\left[\text{SG}\left(\boldsymbol{F}\left(\frac{p_{\text{ref}}(x)}{p_{\text{inf}}(x)}\right)\right)\min\left(r_\theta(x)\hat{A}(x), \text{clip}(r_\theta(x), 1-\epsilon, 1+\epsilon)\hat{A}(x)\right)\right],$$

becomes the OBRS-aware PPO objective:

$$\mathcal{L}^{\text{PPO-OBRS}}(\theta) = \mathbb{E}_{x \sim p_{\text{inf}}}\Bigg[\text{Mask}(x)\,\text{SG}\left(\boldsymbol{F}\left(\frac{p_{\text{target}}(x)}{p'_{\text{inf}}(x)}\right)\frac{p_{\text{ref}}(x)}{p_{\text{target}}(x)}\right) \min\left(r_\theta(x)\hat{A}(x), \text{clip}(r_\theta(x), 1-\epsilon, 1+\epsilon)\hat{A}(x)\right)\Bigg], \tag{5}$$

where rejected samples are removed by $\text{Mask}(x)$, and accepted samples are reweighted according to $p'_{\text{inf}}(x)$. The function $\boldsymbol{F}$ is the truncated IS correction (Section 2), and the choice of p_{target} is discussed in Section 4.3 (Empirically, we can use reference policy p_{ref} or the updated policy $p_{\boldsymbol{\theta}_{\text{new}}}$ as the $p_{\text{target}}(x)$ for distribution alignment.) We present the details of this part in Algorithm 1.

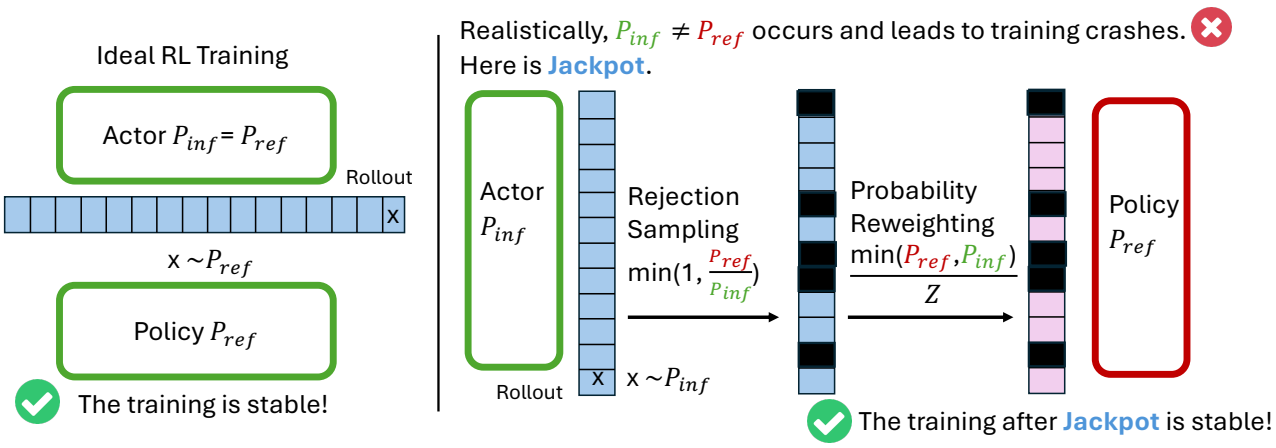

Figure 3 Illustration of JACKPOT Pipeline focusing on Optimal Budgeted Rejection Sampling (OBRS) and Reweighting Procedures

(2) Standard PPO loss for the rollout model. The rollout model ω is also optimized using PPO, but without OBRS:

$$\mathcal{L}^{\text{PPO}}(\omega) = \mathbb{E}_{x\sim p_{\text{inf}}}\left[\text{SG}\left(\boldsymbol{F}\left(\frac{p_{\omega,\text{ref}}(x)}{p_{\text{inf}}(x)}\right)\right)\min\left(r_\omega(x)\hat{A}(x), \text{clip}(r_\omega(x), 1-\epsilon, 1+\epsilon)\hat{A}(x)\right)\right], \tag{6}$$

where $p_{\omega,\text{ref}}$ and r_ω explicitly denote quantities computed from the rollout model.

(3) On-policy distillation loss for the rollout model. To ensure the rollout model tracks the improving policy, we apply a forward KL distillation loss using the same sampled trajectories:

$$\mathcal{L}^{\text{distill}}(\omega) = \mathbb{E}_{x\sim p_{\text{inf}}}\left[D_{\text{KL}}\left(\text{SG}(p_{\theta_{\text{new}}}(x)) \| p_\omega(x)\right)\right]. \tag{7}$$

After optimization, the updated rollout distribution p_ω is used as the inference distribution p_{inf} for the next training iteration, if we do not consider the numerical precision differences between the rollout engine and the training engine (He and Lab, 2025).

Joint objective. Combining the three components above, the overall JACKPOT training objective is

$$\mathcal{L}^{\text{JACKPOT}}(\theta,\omega) = \underbrace{\mathcal{L}^{\text{PPO-OBRS}}(\theta)}_{\text{policy RL}} + \underbrace{\mathcal{L}^{\text{PPO}}(\omega)}_{\text{rollout RL}} + \lambda_{\text{distill}} \underbrace{\mathcal{L}^{\text{distill}}(\omega)}_{\text{on-policy distillation}} \tag{8}$$

where $\lambda_{\text{distill}} \geq 0$ are hyperparameters that balance the contributions of the rollout RL loss and the distillation loss.

4.3 Which policy to approximate?

OBRS allows us to align the inference distribution p_{inf} toward *any* chosen target distribution p_{target}. This flexibility raises a natural question: *which policy distribution should we approximate during training?*

A straightforward choice is the reference policy p_{ref}, which is consistent with the formulation of standard PPO and provides a stable trust-region anchor. However, we empirically find that in training regimes where policy staleness becomes a major issue, such as large-batch RL or asynchronous rollout-update pipelines, the reference policy may drift too far from the current updated policy. In such cases, aligning p_{inf} toward the latest policy p_{new} yields better performance, as the trust region defined by p_{ref} is too outdated to offer meaningful guidance.

Therefore, JACKPOT offers users the flexibility to choose $p_{\text{target}} = p_{\text{ref}}$ or $p_{\text{target}} = p_{\text{new}}$, depending on the requirement in their training setup.

4.4 Memory Optimization and Efficiency

Implementing JACKPOT directly faces a huge challenge of computational feasibility. Note that the weight's normalization constant, Z Section 4.1, requires a sum over the entire vocabulary ($|\mathcal{V}| > 100{,}000$), creating a crippling memory bottleneck from storing full logit vectors (`batch_size` $\times$ `seq_len` $\times$ `vocab_size`). This severely restricts batch sizes, directly undermining the efficiency OBRS is intended to provide. Therefore, transforming this principled approach into a large-scale RL system requires non-trivial engineering: we must introduce

mechanisms to both bound the importance weights for stability and develop a computationally efficient, low-bias estimator for the normalization constant. To overcome the computational bottleneck of calculating Z, we employ a top-k approximation and then empirically debias it. We present the details in Algorithm 1.

4.4.1 Top-k Approximation

The probability mass of language models is typically concentrated in a small subset of the vocabulary. We leverage this property by approximating the sum over $\mathcal{V}$ with a sum over a much smaller set, $\mathcal{V}_k$, which contains the most likely tokens from both the inference and current policies. Specifically, let top-k(p) be the set of k tokens with the highest probability under distribution p. We define our approximation set as the union:

$$\mathcal{V}_k = \text{top-k}(p_{\text{inf}}) \cup \text{top-k}(p_{\boldsymbol{\theta}_{\text{new}}})$$

The union is crucial because a token might be highly probable under one distribution but not the other, and the min function makes these overlapping regions important. The approximate normalization constant, Z_{approx}, is then: $Z_{\text{approx}} = \sum_{a' \in \mathcal{V}_k} \min\left(p_{\text{inf}}(a'), \frac{p_{\text{target}}(a')}{\lambda}\right)$

4.4.2 Bias Correction

While efficient, this top-k approximation introduces a systematic bias. Since the terms in the sum are non-negative, omitting tokens from the full vocabulary $\mathcal{V}$ can only decrease the total sum. Therefore, our approximation is a consistent underestimation of the true value:

$$\mathbb{E}[Z_{\text{approx}}] \leq Z$$

For k=20, . This bias could systematically alter the scale of the gradients during training. Fortunately, there is an elegant way to correct this. A key property of the framework is that the true normalization constant Z is exactly equal to the expected acceptance rate, $\bar{\alpha}$:

$$\bar{\alpha} = \sum_{a \in \mathcal{V}} p_{\text{inf}}(a) \cdot \min\left(1, \frac{p_{\text{target}}(a)}{\lambda \cdot p_{\text{inf}}(a)}\right) = \sum_{a \in \mathcal{V}} \min\left(p_{\text{inf}}(a), \frac{p_{\text{target}}(a)}{\lambda}\right) = Z.$$

During the data collection phase (Algorithm 1, Phase 1), we can compute an unbiased empirical estimate of $\bar{\alpha}$ from the observed samples:

$$\hat{\bar{\alpha}} = \frac{\text{Number of accepted samples}}{\text{Total number of proposed samples}}$$

This gives us two estimators for Z: the low-variance but biased Z_{approx}, and the unbiased but higher-variance $\hat{\bar{\alpha}}$. We can combine them to create a de-biased, low-variance estimator. We compute a batch-wide calibration factor, κ, by dividing the empirical acceptance rate by the batch-averaged Z_{approx}:

$$\kappa = \frac{\hat{\bar{\alpha}}}{\frac{1}{B}\sum_{i=1}^{B} Z_{\text{approx}}^{(i)}}$$

where B is the number of samples in the batch. We then apply this scalar correction to each per-token Z_{approx} value used in the loss calculation. This procedure scales our efficient top-k estimate to match the true expected value observed in practice, effectively removing the bias while retaining the computational benefits and lower variance of the top-k approach.

4.4.3 Efficiency Analysis

Despite introducing two additional objectives, the rollout-model RL loss and the on-policy distillation loss, JACKPOT remains highly efficient in practice. We summarize the key reasons below.

(1) Extra losses do not introduce extra rollouts. Rollout generation is the dominant computational bottleneck in RL training for LLMs, typically contributing over 80% of total runtime. Both the rollout-model PPO loss and the distillation loss operate entirely on the *same trajectories* collected for training the policy model; hence, no additional rollouts are required. Moreover, the rollout model is intentionally chosen to be more efficient than the policy model (e.g., a smaller variant), further reducing the marginal cost of updating ω. As a result, the added losses introduce only a small overhead while keeping the overall training throughput dominated by rollout efficiency.

(2) Minimal tensor overhead and no extra probability computation. JACKPOT is lightweight in implementation: it reuses tensors that are already materialized within the PPO computation graph. No additional `log_prob` evaluations are needed because both p_{ref} and p_{new} are produced as part of the standard objective (5).

JACKPOT simply reuses these probabilities for OBRS reweighting and alignment. Furthermore, JACKPOT requires no changes to vLLM—no custom operators, kernels, or special numerical precision assumptions. Our implementation runs directly on standard vLLM for rollout without any system-level modification.

(3) No trajectory resampling is required (contrast with speculative decoding or sequence-wise rejection sampling). A critical distinction from speculative decoding Leviathan et al. (2023) is that JACKPOT does *not* resample the remainder of a trajectory once a token is rejected. Speculative decoding discards the entire suffix of a trajectory after the first mismatch between the draft and target models, requiring additional sampling to regenerate the remaining tokens. In contrast, JACKPOT simply masks individual rejected tokens according to the OBRS criterion, while keeping the rest of the trajectory intact. This design avoids any resampling of trajectories and significantly reduces computational overhead.

Algorithm 1 The Jackpot Algorithm

Require: Policies: current p_{new}, reference p_{ref}, inference p_{inf}.
Require: Hyperparameters: OBRS threshold λ, PPO clip ϵ, Jackpot clips c_1,c_2, top-k count.

Convention: $\text{SG}(\cdot)$ denotes the stop-gradient operation.
Implementation note: Jackpot only reweights quantities from the *standard* rollout and PPO/GRPO forward passes; it does *not* perform extra model forward passes or trajectory recomputation.
Phase 1: Efficient Rollout (Standard Generation)
Initialize experience buffer $\mathcal{D} \leftarrow \emptyset$.
for each trajectory sampling step t **do**
  Single forward pass of $p_{\text{inf}}(\cdot \mid s_t)$, sample $a_t \sim p_{\text{inf}}(\cdot \mid s_t)$.
  From the same forward, compute and store top-k log-probabilities of p_{inf}: $\text{TopK}_{\text{inf}}(s_t)$.
  Store $(s_t, a_t, p_{\text{inf}}(a_t \mid s_t), \text{TopK}_{\text{inf}}(s_t))$ (plus rewards, values, etc.) in buffer $\mathcal{D}$.
end for
Compute advantages $\hat{A}_t$ using collected trajectories.
Phase 2: PPO Update with Jackpot Reweighting
for each mini-batch sampled from $\mathcal{D}$ **do**
  // 1. Standard PPO Computation (reused by Jackpot)
  Forward pass p_{new} and p_{ref} on the mini-batch to get logits, $p_{\text{new}}(a_t \mid s_t)$, $p_{\text{ref}}(a_t \mid s_t)$, and $\text{TopK}_{\text{new}}(s_t)$.
  Compute policy ratio: $r_t(\boldsymbol{\theta}) = \dfrac{p_{\text{new}}(a_t \mid s_t)}{p_{\text{ref}}(a_t \mid s_t)}$.
  Compute vanilla PPO objective: $\mathcal{L}_{\text{PPO}} = \min\big(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)$.
  // 2. Efficient Z-Approximation and Bias Correction (no extra forward passes)
  Construct approximation set $\mathcal{V}_k = \text{TopK}_{\text{inf}}(s_t) \cup \text{TopK}_{\text{new}}(s_t)$.
  Compute

$$Z_{\text{approx}} = \sum_{x \in \mathcal{V}_k} \min\left(p_{\text{inf}}(x \mid s_t), \frac{p_{\text{new}}(x \mid s_t)}{\lambda}\right).$$

  Estimate correction factor κ using the OBRS-based bias-correction procedure described in Sec. 4.4.2 (e.g., from batch-level OBRS statistics).
  Set corrected normalizer $Z_t \leftarrow \kappa \cdot Z_{\text{approx}}$.
  // 3. Jackpot Weight Calculation
  OBRS weight: $w_{\text{OBRS}} = Z_t \cdot \max\left(\lambda, \frac{p_{\text{new}}(a_t \mid s_t)}{p_{\text{inf}}(a_t \mid s_t)}\right)$.
  $\rho_{\text{jackpot}} = \min(w_{\text{OBRS}}, c_1) \cdot \min\left(\frac{p_{\text{ref}}(a_t \mid s_t)}{p_{\text{new}}(a_t \mid s_t)}, c_2\right)$.
  // 4. Apply Weight to Loss
  $\mathcal{L}_{\text{final}} = \text{SG}(\rho_{\text{jackpot}}) \cdot \mathcal{L}_{\text{PPO}}$.
  Update policy parameters new using gradient of $-\mathcal{L}_{\text{final}}$.
end for

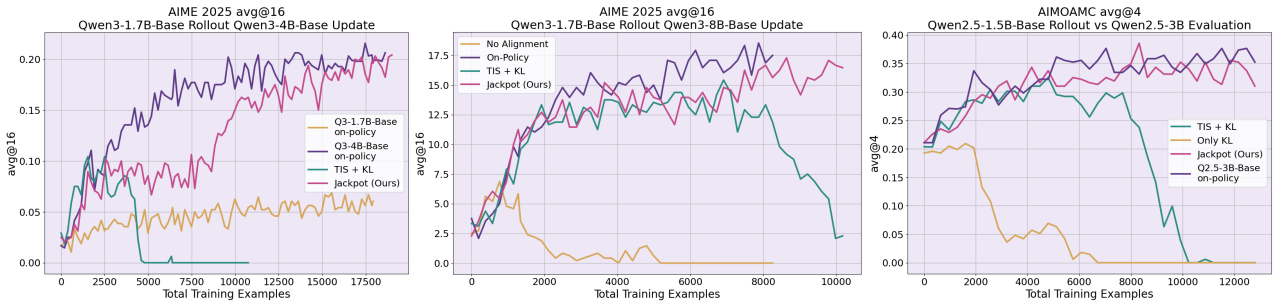

Figure 4 Jackpot enables probability distribution alignment beyond existing methods. On the extreme two model joint training setting, with Jackpot, the smaller and weaker model is able to rollout trajectories which are used by the bigger stronger models for computing its training. We show that prior TIS methods, even added the KL, consistently suffers from unstable training across three different settings. Qwen2.5 series 1.5B and 3B, Qwen3 1.7B and 4B, and Qwen3 1.7B and 8B base models. In contrast, Jackpot leads to comparable performance with the large model on-policy performance.

Table 1 Evaluation results across datasets for different actor–policy training configurations. Mean@4 and Pass@4 are computed using four independent samples.

Models	GSM8K Mean@4	MATH-500 Mean@4	AMC22/23 Mean@4	AMC12 Mean@4	AIME24 Mean@4	AIME24 Pass@4	AIME25 Mean@4	AIME25 Pass@4
Qwen2.5-1.5B→ Qwen2.5-3B (14k, 64, MATH-8K)								
Q2.5-3B On-policy	0.8500	0.6390	0.3765	0.2611	–	–	–	–
Only Reverse KL	0.7417	0.4535	0.2093	0.1407	–	–	–	–
TIS + Reverse KL	0.8250	0.6045	0.3253	0.2444	–	–	–	–
Jackpot (Ours)	0.8428	0.6275	0.3855	0.2778	–	–	–	–
Qwen3-1.7B→ Qwen3-4B (20k, 64, DeepScaleR)								
Q3-1.7B On-policy	0.8380	0.6590	0.3524	0.2500	0.1000	0.1741	0.0680	0.1336
Q3-4B On-policy	0.9256	0.8082	0.5813	0.5166	0.2500	0.3514	0.2156	0.2966
TIS + Reverse KL	0.9121	0.7365	0.4639	0.3277	0.1333	0.2031	0.1041	0.1912
Jackpot (Ours)	0.9215	0.8052	0.5949	0.5388	0.2350	0.3364	0.2083	0.2778
Qwen3-1.7B→ Qwen3-8B (15k, 64, DeepScaleR)								
Q3-8B On-policy	0.9329	0.7950	0.6114	0.5333	0.2437	0.3385	0.1687	0.2616
TIS + Reverse KL	0.9361	0.7645	0.5662	0.3722	0.1770	0.2741	0.1541	0.2223
Jackpot (Ours)	0.9357	0.8265	0.6204	0.5444	0.2500	0.3657	0.1916	0.2769

5 Empirical Analysis

JACKPOT can be layered on top of truncated importance sampling (TIS) and enables training in regimes where actor–policy mismatch is extremely large. In this section, we consider the most challenging configuration: training a strong, expensive policy model using a completely separate, smaller, and more efficient actor model for rollouts. Unlike scenarios where the actor model is merely stale or an approximate quantized version of the policy, employing a fully separate model introduces KL divergences that are orders of magnitude larger than those observed in conventional asynchronous-training setups. At the same time, such decoupling offers the potential for substantial rollout cost reductions, making this setting both practically important and algorithmically demanding.

Setup. We evaluate JACKPOT in the context of training LLMs on mathematical reasoning tasks using the GRPO algorithm Guo et al. (2025). Our experiments span three strong–weak actor–policy model pairs across two model families and two datasets: (1) Qwen2.5-1.5B-Base (actor) with Qwen2.5-3B-Base (policy) trained on the MATH dataset Hendrycks et al. (2021); (2) Qwen3-1.7B-Base with Qwen3-4B-Base trained on the DeepScaleR dataset Luo et al. (2025); and (3) Qwen3-1.7B-Base with Qwen3-8B-Base also trained on DeepScaleR. The maximum generation length is set to 8192 tokens for all experiments.

Results. Naively applying existing methods to this decoupled-actor setting leads to highly unstable training, even when truncated importance sampling (TIS) is used to constrain the distribution gap between the reference

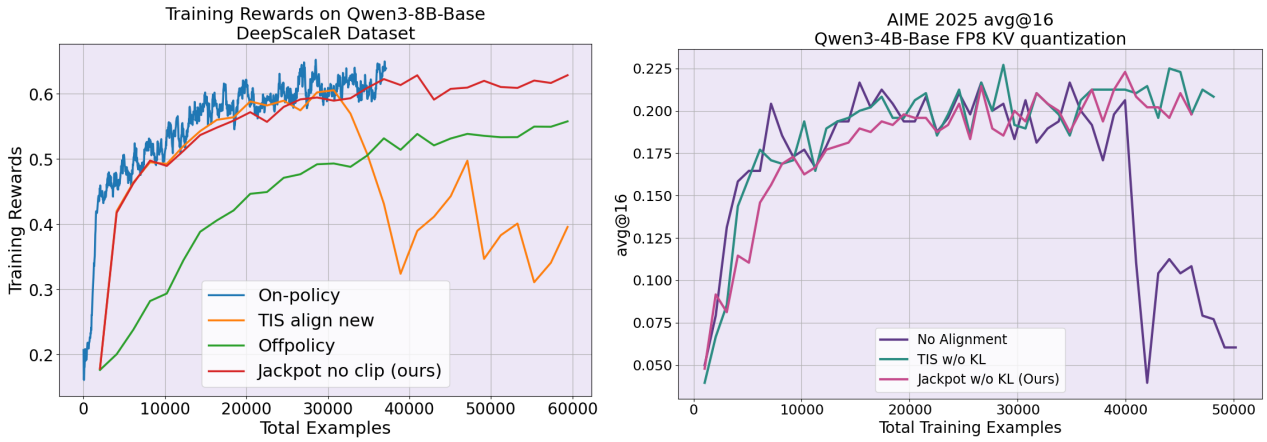

Figure 5 Jackpot enables removal of clipping from stale RL training. Jackpot isn't showing improvement nor harm when actor-policy distributions are relatively close and can be sufficiently corrected by TIS.

and inference models—training still collapses. In contrast, JACKPOT maintains stable learning for a substantially extended number of update steps, demonstrating stability up to 300 steps across the evaluated model pairs. These results highlight the effectiveness of OBRS-based alignment in mitigating distribution mismatch and improving robustness in decoupled RL training.

We show the results in the Figure. We show that the large model performance can indeed be stably trained for a large number of steps and cover a large number of examples. Surprisingly, for the Qwen3-4B-Base model, our joint training even matches the 4B model performance when training itself. JACKPOT consistently outperforms TIS even with reverse KL added. The above results marks the potential for significant RL training cost savings.

6 Ablation Studies

6.1 When Jackpot isn't Effective

We find that JACKPOT offers little improvement over standard methods when the distribution shift is inherently small or already well-controlled by existing techniques.

RL algorithms like PPO and GRPO employ clipping mechanisms to enforce trust regions, explicitly constraining the deviation of the current policy from the old (rollout) policy. While this design enhances stability, it inherently limits the magnitude of the update per step. Consequently, the distribution shift between the rollout policy and the current policy remains small. In these regimes, the distribution correction provided by JACKPOT becomes less critical, resulting in performance on par with, rather than superior to, standard baselines.

Similarly, while FP8 KV quantization during rollout introduces a larger distribution gap between the training and rollout frameworks by increasing the "off-policy" nature of the task, standard techniques like TIS effectively mitigate this instability. Because TIS is sufficient to restore performance levels comparable to unquantized baselines, the additional distribution correction offered by JACKPOT yields diminishing returns in this context. Detailed evaluations are demonstrated in Table 2.

6.2 What Jackpot enables

In contrast, JACKPOT provides substantial benefits in regimes with large policy shifts, enabling both stable training and faster convergence.

To evaluate the benefits of JACKPOT, we investigate a training regime where the protective constraints of PPO clipping are removed. While clipping ensures stability by limiting the policy update magnitude, it simultaneously throttles the learning speed. By relaxing these constraints, we simulate a scenario with significant distribution shifts: a setting where robust distribution correction is extremely necessary. We compare the performance

Table 2 JACKPOT provides little additional benefit when distribution shifts are already controlled by PPO clipping or TIS under FP8 KV quantization.

	GSM8K	MATH-500	AMC22 & 23	AMC12 2024	AIME24		AIME25	
Models / Methods	Mean@4	Mean@4	Mean@4	Mean@4	Mean@16	Pass@16	Mean@16	Pass@16
Qwen3-4B-Base on DeepScaleR-Preview Dataset (rollout batch size = 2048; train batch size = 32; 64×)								
Off Policy	92.10	81.55	60.54	56.11	27.50	38.36	23.12	31.36
Jackpot (Ours)	92.53	81.95	59.94	56.11	27.71	37.77	22.70	31.22
Qwen3-4B-Base on DeepScaleR-Preview Dataset (rollout batch size = 128; train batch size = 64; FP8 KV)								
TIS	92.68	83.65	60.84	53.89	25.83	36.95	22.70	29.2
Jackpot (Ours)	91.83	81.30	62.35	50.56	24.79	35.13	22.29	29.51

Table 3 Ablation under large distribution shift (no PPO clipping): JACKPOT enables stable and fast convergence across benchmarks. We report the best test accuracy for the first 30K training examples on Qwen3-4B-Base and 50K on Qwen3-8B-Base.

	GSM8K	MATH-500	AMC22 & 23	AMC12 2024	AIME24		AIME25	
Models / Methods	Mean@4	Mean@4	Mean@4	Mean@4	Mean@16	Pass@16	Mean@16	Pass@16
Qwen3-4B-Base on DeepScaleR-Preview Dataset (rollout batch size = 2048; train batch size = 32; 64×)								
On Policy	92.19	81.55	58.43	51.11	23.12	33.13	22.91	30.95
Off Policy	88.04	71.15	39.15	29.44	13.96	23.03	11.04	18.61
No-Clip	**92.76**	79.50	**57.22**	43.33	18.75	26.03	17.71	24.61
Jackpot (Ours)	92.24	**80.05**	53.92	**50.00**	**20.63**	**29.48**	**18.13**	**23.63**
Qwen3-4B-Base on DeepScaleR-Preview Dataset (rollout batch size = 4096; train batch size = 32; 128×)								
Off Policy	79.70	60.20	33.00	24.44	8.00	15.73	5.00	11.00
No-Clip	20.70	19.10	7.80	5.00	1.00	4.00	1.00	2.00
Jackpot (Ours)	**92.00**	**80.00**	**51.20**	**47.22**	**19.16**	**24.58**	**18.52**	**25.08**
Qwen3-8B-Base on DeepScaleR-Preview Dataset (rollout batch size = 2048; train batch size = 32; 64×)								
On Policy	94.24	93.99	68.95	54.44	28.95	37.89	22.50	28.54
Off Policy	91.05	77.15	50.60	40.00	18.54	28.67	14.16	21.98
No-Clip	93.85	82.55	60.54	48.33	24.58	35.06	20.00	22.90
Jackpot (Ours)	**94.01**	**83.05**	**63.55**	**54.44**	**26.87**	**36.23**	**20.41**	**26.57**

of three configurations over the first 30,000 training examples: a **no-clip** baseline (off-policy without PPO clipping), a **vanilla off-policy** baseline, and our proposed **Jackpot** method (remove PPO clipping).

Our empirical results of Qwen3-4B-Base with rollout batch size 4096 (128× staleness) in Table 3 highlight a critical trade-off between stability and convergence speed that standard methods fail to navigate: 1. In the absence of trust-region clipping, the **no-clip** run crashes mid-training. 2. The **vanilla off-policy** baseline maintains stability throughout the training run. While it demonstrates constant, monotonic improvement in performance, it suffers from slow convergence speeds. Without an effective mechanism to correct for the distribution lag between the behavior and target policies, the model learns inefficiently. While the training does not crash with rollout batch size 2048 (64× staleness) for either model, we still observe clear benefits of JACKPOT in both convergence speed and final accuracy. Because all methods are evaluated under the same fixed budgets of 30K and 50K training examples, these higher accuracies directly indicate faster convergence in the large-batch regime.

JACKPOT successfully combines the benefits of the conflicting baselines without their respective drawbacks. Unlike the **no-clip** setting, JACKPOT maintains robust stability and completes the training without crashing, effectively managing the large distribution shifts introduced by unclipped updates. Furthermore, it significantly outperforms the vanilla off-policy baseline in terms of convergence speed. By providing a more accurate distribution correction, JACKPOT enables the model to take larger, stable optimization steps, accelerating learning in regimes where standard trust-region constraints are significantly loosened or removed.

7 Conclusion

We presented JACKPOT, a framework that leverages Optimal Budget Rejection Sampling to reduce the distribution discrepancy between the rollout model and the policy model in reinforcement learning for large language models. Through a principled OBRS procedure, a unified training objective that couples policy and rollout updates, and an efficient system implementation enabled by Top-k-based probability estimation and batch-level bias correction, JACKPOT moves the RL training pipeline a step closer toward fully decoupling rollout generation from policy optimization. This work highlights a practical direction for enabling more flexible and efficient training regimes for large language models.

8 Limitations

As shown in Section 6, when the distribution shift is already small or adequately controlled by existing techniques, JACKPOT yields only minor improvements over standard baselines. Moreover, although JACKPOT reduces the mismatch between the rollout model and the policy model, using a fully separate, smaller actor model to train a large and expensive policy does not completely eliminate the distribution gap or the resulting training instability. In our experiments, we observe that training may still diverge after extended optimization (e.g., beyond 500 update steps), and JACKPOT has not yet been validated on larger-scale models such as 32B variants.

Achieving a fully robust decoupling between rollout generation and policy optimization likely requires additional mechanisms beyond OBRS. A promising direction is to introduce a closed-loop control scheme that adapts the relative strength of the distillation loss and RL loss based on real-time measurements of the KL divergence between the rollout and policy models.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. https://arxiv.org/abs/1802.01561.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Horace He and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

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, 2021. https://arxiv.org/abs/2103.03874.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Yu Shen. When speed kills stability: Demystifying RL collapse from the training-inference mismatch, September 2025a. https://richardli.xyz/rl-collapse.

Liyuan Liu, Feng Yao, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Flashrl: 8bit rollouts, full power rl, August 2025b. https://fengyao.notion.site/flash-rl.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*, 2023.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025. Notion Blog.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235, 2024.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. Seer: Online context learning for fast synchronous llm reinforcement learning. *arXiv preprint arXiv:2511.14617*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws. *arXiv preprint arXiv:2506.05333*, 2025.

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

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.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

Ling Team, Anqi Shen, Baihui Li, Bin Hu, Bin Jing, Cai Chen, Chao Huang, Chao Zhang, Chaokun Yang, Cheng Lin, et al. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *arXiv preprint arXiv:2510.18855*, 2025.

Together-AI. Adaptive-learning speculator system (atlas): A new paradigm in llm inference via runtime-learning accelerators. https://www.together.ai/blog/adaptive-learning-speculator-system-atlas.

Alexandre Verine, Muni Sreenivas Pydi, Benjamin Negrevergne, and Yann Chevaleyre. Optimal budgeted rejection sampling for generative models. In *International Conference on Artificial Intelligence and Statistics*, pages 3367–3375. PMLR, 2024.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, et al. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm trainin. *arXiv preprint arXiv:2505.24034*, 2025.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025a.

Haizhong Zheng, Jiawei Zhao, and Beidi Chen. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? *arXiv preprint arXiv:2510.01161*, 2025b.

Haizhong Zheng, Yang Zhou, Brian R Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *arXiv preprint arXiv:2506.02177*, 2025c.