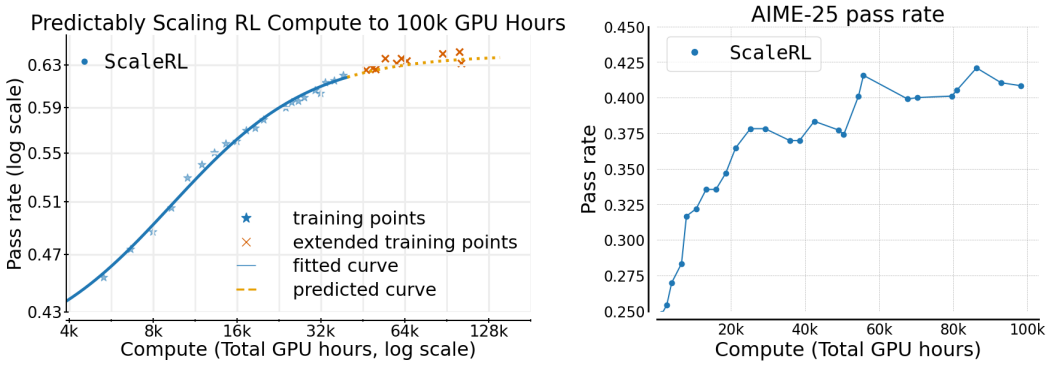
THE ART OF SCALING REINFORCEMENT LEARNING COMPUTE FOR LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

While the training compute for reinforcement learning (RL) for LLMs is massively increasing, the field is still lacking predictive scaling methodologies for RL comparable to those established for pre-training. This gap is increasingly consequential given recent large scale RL efforts for reasoning-centric post-training. We present the first open, large-compute, systematic study of RL scaling for LLMs. We fit sigmoidal compute-performance curves for RL post-training and ablate a wide range of common design choices. We observe: (1) Not all recipes yield similar asymptotic performance; (2) details such as loss aggregation, normalization, curriculum, and precision handling primarily modulate compute efficiency without materially shifting the asymptote; (3) Stable and scalable recipes exhibit predictive performance behavior as a function of compute, akin to established recipes in pre-training. Combining these insights, we propose a "best-practice" recipe, **ScaleRL**, and demonstrate its effectiveness by successfully scaling and predicting RL training performance on up to 100,000 GPU-hours, based on 400,000 GPU-hours of total experiments. Our study provides a principled foundation for predictive RL scaling laws in the LLM era, and a stable, scalable recipe.

1 INTRODUCTION



(a) (b)

Figure 1: (a) ScaleRL scaled to 100k GPU hours on a 8B dense model. We fit a sigmoid curve (eq. (1)) up to 50k GPU hours and extrapolate to 100k. The extrapolated curve closely follows extended training, demonstrating both stability at large compute and predictive fits, establishing ScaleRL as a reliable candidate for future RL scaling laws. (b) Downstream evaluation on AIME-25 shows a consistent scaling trend for ScaleRL, thus generalizing beyond the training data distribution.

Scaling reinforcement learning (RL) compute is emerging as the critical paradigm for advancing large language models (LLMs). While pre-training establishes the foundation of a model, it is the subsequent phase of RL training that unlocks the important capabilities of today's LLM capabilities, from test-time thinking (OpenAI, 2024) to agentic capabilities (Kimi Team et al., 2025a). For instance, Deepseek-R1-Zero used 100,000 H800 GPU hours for RL training – 3.75% of its pre-training

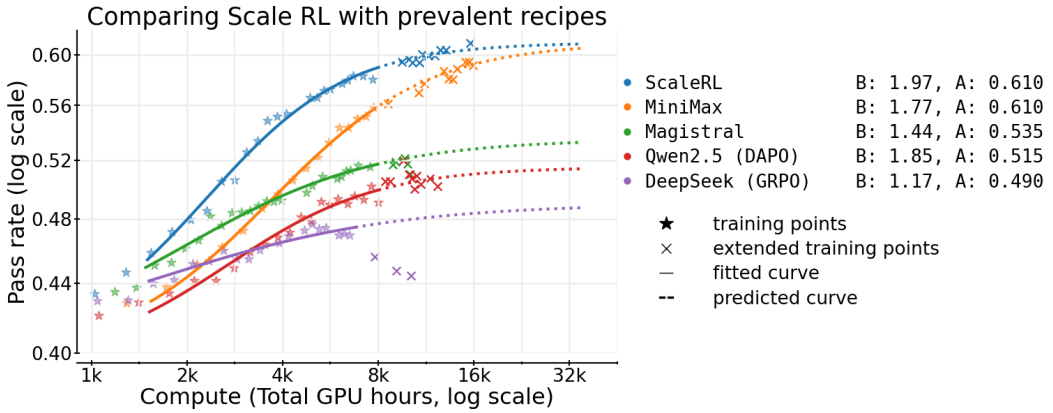


Figure 2: We fit sigmoid curves (Equation (1)) to prevalent training recipes like DeepSeek (GRPO) (Guo et al., 2025), Qwen-2.5 (DAPO) (Yu et al., 2025), Magistral (Rastogi et al., 2025), and Minimax-M1 (MiniMax et al., 2025), and compare them with **ScaleRL**. ScaleRL surpasses all other methods, achieving an asymptotic reward of $A = 0.61$. Stars denote evaluation points; solid curves show the fitted curve on the region of points it's fitted onto; dashed curves extrapolate beyond it. We validate the predictability by running each method for longer ("+" markers), which align closely with the extrapolated curves for stable recipes like ScaleRL and MiniMax.

compute (Guo et al., 2025). This dramatic increase in RL compute is amplified across frontier LLM generations, with more than 10× increase from o1 to o3 (OpenAI, 2025) and a similar leap from Grok-3 to Grok-4 (xAI Team, 2025).

While RL compute for LLMs has scaled massively, our understanding of *how* to scale RL has not kept pace; the methodology remains more *art* than science. Recent breakthroughs in RL are largely driven by isolated studies on novel algorithms (e.g., Yu et al. (DAPO, 2025)) and model-specific training reports, such as, MiniMax et al. (2025) and Magistral (Rastogi et al., 2025). Critically, these studies provide *ad-hoc* solutions tailored to specific contexts, but not *how* to develop RL methods that scale with compute. This lack of scaling methodology stifles research progress: with no reliable way to identify promising RL candidates *a priori*, progress is tied to large-scale experimentation that sidelines most of the academic community.

This work lays the groundwork for science of RL scaling by borrowing from the well-established concept of *scaling laws* from pre-training. While pre-training has converged to algorithmic recipes that scale *predictably* with compute (Kaplan et al., 2020; Hoffmann et al., 2022; Owen, 2024), the RL landscape lacks a clear standard. As a result, RL practitioners face an overwhelming array of design choices, leaving the fundamental questions of *how* to scale and *what* to scale unanswered. To address these questions, we establish a predictive framework for RL performance using a sigmoid-like saturating curve between the expected reward (R_C) and training compute (C):

$$\overbrace{R_C - R_0}^{\text{Reward Gain}} = \overbrace{(A - R_0)}^{\text{Asymptomatic Reward Gain}} \times \underbrace{\frac{1}{1 + (C_{\text{mid}}/C)^B}}_{\text{Compute efficiency}}, \tag{1}$$

where $0 < A \leq 1$ represents the asymptotic pass rate, and $B > 0$ is a scaling exponent that determines compute efficiency, and C_{mid} sets the midpoint of the RL performance curve. This framework in Equation 1 allows researchers to *extrapolate* performance from lower-compute runs to higher compute budgets, enabling them to evaluate scalability of RL methods without incurring the compute cost of running every experiment to its computational limit.

Guided by this framework, we develop **ScaleRL**, an RL recipe that scales *predictably* with compute. In a massive **100,000 GPU-hours training run**, we show that ScaleRL's performance closely matches the scaling curve predicted by our framework (Figure 1). Critically, scaling curves extrapolated from only the initial stages of training closely match the final observed performance, confirming the predictive ability of our framework to extreme compute scales.

The design of ScaleRL is grounded in a comprehensive empirical study of RL scaling that spanned over **400,000 GPU-hours** (GB200). This study explored numerous design choices at an 8B parameter scale, where individual runs use up to 16,000 GPU-hours, making them **6× cheaper** than experimenting at our largest training run scale. This investigation yielded three key principles:

- **RL Performance Ceilings are Not Universal**: As we scale training compute for different methods, they encounter different ceilings on their achievable performance (A). This limit can be shifted by choices such as the off-policy RL loss and batch size.
- **Embracing the Bitter Lesson**: Methods that appear superior at small compute budgets can be worse when extrapolated to large-compute regimes (e.g., Figure 2). We can still identify scalable methods by estimating the scaling parameters (A, B) from the early training dynamics using our framework (Equation 1).
- **Re-evaluating Common Wisdom**: Common interventions thought to improve peak performance (e.g., loss aggregation, data curriculum, length penalty, advantage normalization) mainly adjust compute efficiency (B), not the performance ceiling.

Based on these insights, ScaleRL achieves *predictable* scaling by integrating existing methods, rather than inventing novel methods. Specifically, ScaleRL combines asynchronous Pipeline-RL setup, forced length interruptions, truncated importance sampling RL loss (CISPO), prompt-level loss averaging, batch-level advantage normalization, FP32 precision at logits, zero-variance filtering, and No-Positive-Resampling – with each component's contribution validated in a leave-one-out ablation study, consuming 16,000 GPU-hours per run.

ScaleRL not only scales *predictably* but also establishes a new **state-of-the-art** (Figure 2) – it achieves higher asymptotic performance and compute efficiency compared to established RL recipes. Moreover, ScaleRL maintains predictable scaling when increasing compute across multiple training axes – including 2.5× larger batch sizes, longer generation lengths up to 32,000 tokens, multi-task RL using math and code, and larger MoE models (Llama-4 17B×16) – with benefits that consistently transfer to downstream tasks. Overall, this work establishes a foundational methodology for rigorously and cost-effectively predicting the scalability of new RL algorithms.

2 PRELIMINARIES

We consider reinforcement learning with LLMs, where prompts x are sampled from a data distribution D. Our setup follows a generator–trainer split across GPUs: a subset of GPUs (*generators*) use optimized inference kernels for high-throughput rollout generation, while the remaining GPUs (*trainers*) run the training backend (FSDP) and update parameters. We denote by $\pi^{\theta}_{\text{gen}}$ and $\pi^{\theta}_{\text{train}}$ the model with parameters θ on the generator and training backends, respectively. For each prompt, the old policy $\pi^{\theta_{old}}_{gen}$ on the generator GPUs produces candidate completions, which are then assigned scalar rewards. Policy optimization proceeds by maximizing a clipped surrogate objective, taking expectations over $x \sim D$ and rollouts from $\pi^{\theta_{old}}_{gen}$.

Base RL Algorithm As our starting point, we start with a "base" algorithm that resembles GRPO (Shao et al., 2024) without any KL regularization term, in line with large-scale training reports (Rastogi et al., 2025; MiniMax et al., 2025). Additionally, we include the asymmetric DAPO clipping (Yu et al., 2025), because of its widespread adoption as a default approach to avoid entropy collapse and maintain output diversity.

For a given prompt x, the old policy $\pi_{\text{gen}}(\theta_{\text{old}})$ generates G candidate completions $\{y_i\}_{i=1}^{G}$, each assigned a scalar reward r_i. We compute adavantages $\hat{A}_i$ and group-normalized advantages using:

$$\hat{A}_i = r_i - \text{mean}(\{r_j\}_{j=1}^{G}), \quad \hat{A}_i^G = \hat{A}_i / (\text{std}(\{r_j\}_{j=1}^{G}) + \epsilon)$$

Each completion y_i of length $|y_i|$ contributes at the token-level **importance sampling** (IS) ratios $\rho_{i,t}(\theta)$, with asymmetric upper and lower clipping thresholds, akin to DAPO (Yu et al., 2025):

$$\rho_{i,t}(\theta) := \frac{\pi^{\theta}_{train}(y_{i,t} \mid x, y_{i,<t})}{\pi^{\theta_{\text{old}}}_{gen}(y_{i,t} \mid x, y_{i,<t})} = \frac{\pi^{\theta}_{train}(y_{i,t})}{\pi^{\theta_{\text{old}}}_{gen}(y_{i,t})}, \quad \text{clip}_{\text{asym}}(\rho, a) := \text{clip}(\rho,\, 1-\epsilon^{-}, 1+\epsilon^{+}) \quad (2)$$

We aggregate losses at the *sample level*, i.e., averaging per-sample token losses before averaging across samples. The surrogate objective is given by:

$$\mathcal{J}(\theta) = \mathbb{E}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{gen}}^{\theta_{\text{old}}}(\cdot|x)}} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t}(\theta)\hat{A}_i^G,\ \text{clip}_{\text{asym}}(\rho_{i,t}(\theta))\hat{A}_i^G\Big) \right] \tag{3}$$

Controlling Generation Lengths. To prevent reasoning output lengths from exploding during training, which harms training stability and efficiency, we use **interruptions** (GLM-V Team et al., 2025; Yang et al., 2025) that forcibly stop overly long generations by appending an end-of-thinking phrase (e.g., "Okay, time is up. Let me stop thinking and formulate a final answer `</think>`"), signaling the LLM to terminate its reasoning and produce a final answer. We revisit this choice later to compare it with a length-penalty aproach that penalizes overly long generations (Kimi Team et al., 2025b).

Predictive scaling and fitting curves We model pass rate versus $\log(\text{compute})$ with a sigmoidal fit (Equation (1)), consistent with prior work that uses sigmoids to capture bounded metrics such as accuracy (Ruan et al., 2024; Srivastava et al., 2022). Similar to pre-training studies (Hoffmann et al., 2022; Li et al., 2025b; Porian et al., 2025), we find that excluding the very early low-compute regime yields more stable fits, after which training follows a predictable trajectory. Unless noted otherwise, all our fits begin after $\sim$1.5k GPU hours. Full details of the fitting procedure are provided in Appendix A.7.

3 An Empirical Study of RL Scaling

We mainly conduct our RL experiments using an 8B dense model on verifiable math tasks with a batch size of 768 and a maximum output sequence length of 14,336 tokens. To demonstrate predictable scaling with ScaleRL, we also consider training setups with a much larger mixture-of-experts model, multiple tasks (math and code), and longer sequence lengths, in Section 4. More details about training setup, including datasets and hyper-parameters, are provided in Section A.3.

We structure our experiments in two stages: first, we ablate design choices on the baseline at 3.5k-4k GPU hours, since many setups destabilize beyond this scale. In fact, when a design change proved stable, we trained it longer. Second, we combine the best choices into ScaleRL and run leave-one-out (LOO) experiments at 16k GPU-hours. We also testing predictability by fitting on the first 8k GPU-hours and extrapolating the remainder.

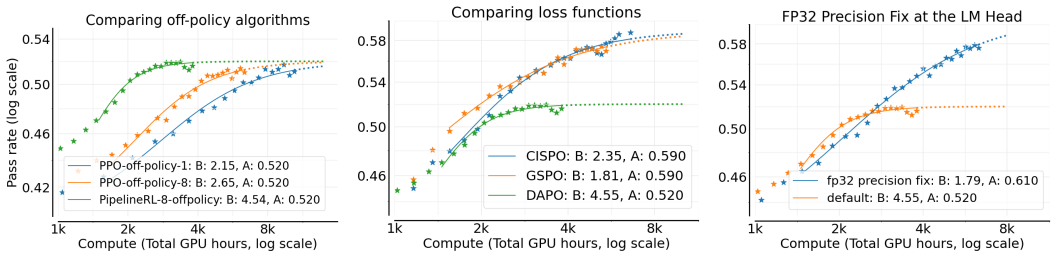
We first investigate the choice of asynchronous off-policy RL setup (Noukhovitch et al., 2024), as it governs training efficiency, independent of all other design choices. Specifically, we consider two approaches to off-policy learning: PPO-off-policy-k and PipelineRL-k. Following this, we study several design axes in terms of their scaling behavior.

3.1 Asynchronous RL Setup

PPO-off-policy-k is the default approach for asynchronous RL and has been used previously by Qwen3 (Yang et al., 2025) and ProRL (Liu et al., 2025a). Here, the old policy $\pi_{gen}^{\theta_{\text{old}}}$ generates reasoning traces for a batch of B prompts. Each gradient update processes a mini-batch of $\hat{B}$ prompts, resulting in $k = B/\hat{B}$ gradient updates per batch. In our experiments, we fix $\hat{B} = 48$ prompts (with 16 generations each), and vary $k \in \{1, 8\}$ by setting $B = k \times 48$.

PipelineRL-k is a recent approach from Piche et al. (2025) and used by Magistral (Rastogi et al., 2025). In this scheme, generators continuously produce reasoning traces in a streaming fashion. Whenever trainers finish a policy update, the new parameters are immediately pushed to the generators, which continue generating with the updated weights but a stale KV cache from the prior policy. Once a full batch of traces is generated, it is passed to the trainers for the next update. In our setup we add a parameter k, whereby the trainers wait if they get k steps ahead of the generators.

We compare these approaches in Figure 3a. PipelineRL achieves similar asymptotic performance A, but substantially improves compute efficiency B; thus reaching the ceiling A faster. This is



(a) (b) (c)

Figure 3: (a) Comparing different off-policy algorithms. We report only the B (scaling exponent) and A (asymptotic pass rate) parameters of the fitted sigmoid curve eq. (1). PipelineRL-k is much more efficient and slightly better in the large compute limit. (b) We also compare popular loss functions: DAPO (Yu et al., 2025), GSPO (Zheng et al., 2025a), and CISPO (MiniMax et al., 2025), and find CISPO/GSPO achieve a higher asymptotic reward compared to DAPO. (c) Using FP32 precision in the final layer (LM head) gives a considerable boost in asymptotic reward.

because pipelineRL reduces the amount of idle time in the training process. This choice yields reliable gains with fewer tokens, making larger sweeps at a lower compute budget possible. We also vary the maximum off-policyness for PipelineRL and find $k = 8$ to be optimal, which we discuss in Appendix A.9.

3.2 Algorithmic Design Choices

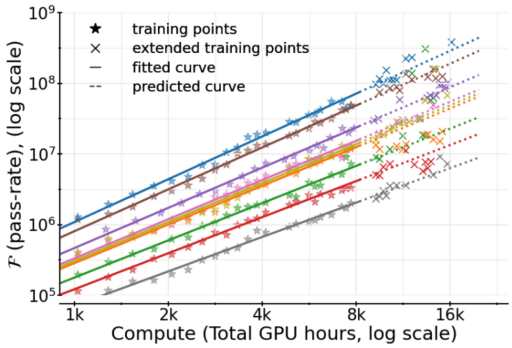
Building on the results above, we adopt PipelineRL-8 as our updated baseline. We now study six additional design axes: (a) loss aggregation, (b) advantage normalization, (c) precision fixes, (d) data curriculum, (e) batch definition, and (f) loss type. In Section 3.3, we combine the best options into a unified recipe, which we call **ScaleRL**, and conduct leave-one-out experiments on a larger scale of $16,000$ GPU-Hours.

Loss Aggregation We evaluate three strategies for aggregating the RL loss: (a) *Sample average* where each rollout contributes equally (as in GRPO, Appendix A.2). (b) *Prompt average* where each prompt contributes equally (as in DAPO, Appendix A.2). (c) *Token average* where all token losses in the batch are averaged directly, without intermediate grouping. The comparison results are shown in Figure 7a. We find prompt-average achieves the highest asymptotic performance and therefore use this choice for ScaleRL.

Advantage Normalization We compare three variants of advantage normalization: (a) *Prompt level* where advantages are normalized by the standard deviation of rewards from the rollouts of the same prompt (as in GRPO, Appendix A.2). (b) *Batch level* where advantages are normalized by the standard deviation across all generations in the batch, as used by Hu et al. (2025a); Rastogi et al. (2025)). (c) *No normalization* where advantages are computed as raw rewards centered by the mean reward of the prompt's generations, without variance scaling (as proposed in Dr. GRPO (Liu et al., 2025b)). Comparison plot is shown in Appendix A.4 (Figure 7b). All three methods yield similar performance. We therefore adopt batch-level normalization as it is theoretically sound and marginally better. This choice is also further corroborated at a larger scale by the LOO experiments in Section 3.3.

Loss type We compare DAPO with two recently proposed alternatives: GSPO (Zheng et al., 2025a) and CISPO (MiniMax et al., 2025; Feng, 2025). GSPO applies importance sampling at the sequence level as opposed to GRPO's token-level formulation. Specifically, GSPO alters the token-level IS ratio (eq. (5)) to sequence-level ratios: $\rho_i(\theta) = {}^{\pi_{train}(y_i|x,\theta)}/_{\pi_{gen}(y_i|x,\theta_{\text{old}})}$. CISPO simply combines truncated IS with vanilla policy gradient (Ionides, 2008), which is given by:

$$\mathcal{J}_{\text{CISPO}}(\theta) = \mathop{\mathbb{E}}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{gen}(\cdot|x,\theta_{\text{old}})}}\left[\frac{1}{T}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\textsf{sg}(\text{clip}_{\text{asym}}(\rho_{i,t}))\hat{A}_i\,\log\left(\pi_{train}(y_{i,t}|x,y_{i<t},\theta)\right)\right]$$



Method	fitted B w/ fixed A = 0.605	original fitted parameters C_{mid} / B / A
ScaleRL	2.01	2542 / 1.92 / 0.610
LOO-prompt-lvl-adv-norm	1.82	2463 / 2.09 / 0.590
LOO-length-penalty	1.75	2660 / 1.68 / 0.610
LOO-batch	1.69	2503 / 1.84 / 0.595
LOO-PPO-off-policy-8	1.89	3409 / 1.89 / 0.605
LOO-no-fp32-precision-fix	1.97	2857 / 1.88 / 0.610
LOO-sample-avg	1.84	2621 / 1.93 / 0.600
LOO-uniform-sampling	1.62	2621 / 1.62 / 0.605
LOO-dapo	1.83	3093 / 1.77 / 0.610

Figure 4: **Leave-One-Out (LOO) Experiments**: Starting from ScaleRL, we revert one design choice at a time to its baseline counterpart and re-train. Most LOO variants reach a similar asymptotic reward, with ScaleRL outperforming slightly overall. The main difference in these methods lies in efficiency. To highlight this, we re-arrange Equation (1) into $\mathcal{F}(R_c) = C^B$, where $\mathcal{F}(R_c) = C_{\text{mid}}^B / \left(\frac{R_c - R_0}{A - R_c} - 1\right)$, and plot $\log \mathcal{F}(R_c)$ vs. $\log C$. This form makes slope B directly visible, showing that ScaleRL achieves the highest compute efficiency.

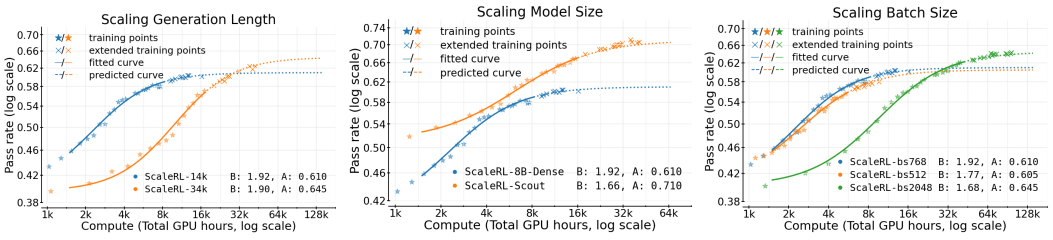
where sg is stop-gradient function.

Figure 3b shows that both GSPO and CISPO substantially outperform DAPO, improving the asymptotic pass-rate A by a large margin. CISPO exhibits a prolonged near-linear reward increase, and is marginally better than GSPO later in training, so we opt for CISPO as our best loss type. Further discussion on GSPO, CISPO, and their comparison is given in Section 3.3, Appendix A.11, and A.10.

FP32 Precision for LLM logits The generator and trainer rely on different kernels for inference and training, leading to small numerical mismatches in their token probabilities (He & Lab, 2025). RL training is highly sensitive to such discrepancies, since they directly affect the IS ratio in the surrogate objective. MiniMax et al. (2025) identified that these mismatches are especially pronounced at the LLM head, and mitigate this by FP32 computations at the head for both generator and trainer. As shown in Figure 3c, the precision fix dramatically improves the asymptotic performance A from 0.52 to 0.61. Given this clear benefit, we include the FP32 precision fix in our ScaleRL recipe.

Zero-Variance Filtering Within each batch, some prompts yield identical rewards across all their generations. These "zero-variance prompts" have zero advantage and therefore contribute zero policy gradient. The default baseline includes such prompts in loss computation, but it is unclear whether they should count toward the effective batch size. To test this, we compare the default setting against an *effective batch* approach, where only prompts with nonzero variance are included in the loss calculation, as done by Seed (Seed et al., 2025). We show in Appendix A.4 (Figure 7c) that using the effective batch performs slightly better, and therefore decide to use it in our ScaleRL recipe.

Adaptive Prompt Filtering A number of data curriculum strategies have been proposed for RL training (An et al., 2025; Zhang et al., 2025; Zheng et al., 2025c). Here, we evaluate a simple yet effective variant introduced in Polaris (An et al., 2025) with the key observation that once a prompt becomes too easy for a policy, it typically remains easy. Since such prompts consume some compute but no longer contribute useful gradient signal (Section 3.2), it is better to exclude them from future training. We implement this by maintaining a history of pass rates and permanently removing any prompt with pass rate ≥ 0.9 from subsequent epochs; we call this **No-Positive-Resampling**. In Figure 7d we compare this curriculum against the default setting where all prompts are resampled uniformly throughout training. We see that the curriculum improves scalability, both in terms of the terminal reward A and compute efficiency B.



(a) (b) (c)

Figure 5: **Large Scale runs:** We scale up the ScaleRL recipe across three axes: (a) *sequence length*: larger sequence is slower but reaches a higher asymptotic reward, (b) *model size*: using the larger 17Bx16 Llama-4 model reaches a much higher terminal reward and shows better generalization, and (c) batch size: larger batch size is slower in training but settles at a higher asymptote. As shown in Section A.13, larger batch size also shows better downstream generalization.

3.3 ScaleRL Recipe and Leave One Out Experiments

From the eight design axes studied above, we consolidate the best-performing settings into a single recipe, which we term **ScaleRL** (**Scale**-able **RL**). ScaleRL is an asynchronous RL recipe that uses **PipelineRL with** 8 **steps off-policyness**, **interruption-based length control** for truncation, **FP32 computation for logits**, and optimizes the **loss** $\mathcal{J}_{\text{ScaleRL}}(\theta)$. This loss combines prompt-level loss aggregation, batch-level advantage normalization, truncated-IS REINFORCE loss, zero-variance filtering, and no-positive resampling:

$$\mathcal{J}_{\text{ScaleRL}}(\theta) = \mathop{\mathbb{E}}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{gen}^{\theta_{old}}(\cdot|x)}}\left[\frac{1}{\sum_{i=1}^{G}|y_i|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\mathsf{sg}(\min(\rho_{i,t},\epsilon))\hat{A}_i^{\text{norm}}\log\pi_{train}^{\theta}(y_{i,t})\right],$$

$$\rho_{i,t}=\frac{\pi_{train}^{\theta}(y_{i,t})}{\pi_{gen}^{\theta_{old}}(y_{i,t})},\ \hat{A}_i^{\text{norm}}=\hat{A}_i/\hat{A}_{\text{std}},\ 0<\text{mean}(\{r_j\}_{j=1}^{G})<1,\ \text{pass\_rate}(x)<0.9,$$

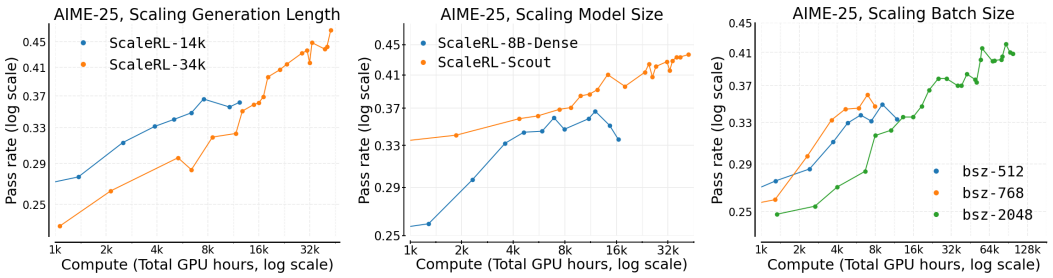
where sg is stop-gradient function, $\hat{A}_{\text{std}}$ is the standard deviation of all advantages $\hat{A}_i$ in the mini-batch and pass_rate(x) denotes historical pass rate of a prompt x. To validate that these choices remain optimal when combined, we conduct *leave-one-out* (LOO) experiments: starting from ScaleRL, we revert one axis at a time to its baseline counterpart from Section 2. This ensures that each design decision contributes positively even in the presence of all others.

Figure 4 reports these experiments, each scaled to 16k GPU hours. Across all axes, ScaleRL consistently remains the most effective configuration, slightly outperforming LOO variants either in asymptotic reward or in compute efficiency. Since most LOO variants reach similar asymptotic pass rates, we transform the sigmoidal fit to a power-law fit, to highlight efficiency differences via the slope B (details in Figure 4). Concretely, we average the asymptotic reward A across all runs, re-fit the curves with this fixed A, and then compare slopes. The corresponding non-transformed pass-rate vs. compute curves are provided in Appendix A.5.

We fit the sigmoidal curve up to 8k GPU hours and extrapolate to 16k, observing that the predicted curves align closely with both training and extended points; demonstrating the stability and predictability of ScaleRL under large-scale training.

4 Scaling Training Compute Across Different Axes

Given a fixed or growing compute budget, which scaling knob—context length, batch size, generations per prompt, model size, or task mix—buys the most reliable performance gain, and how early can we predict that return? We answer this by (i) fitting the saturating power-law in Eq. (1) early in training for each setting (precisely, half the target budget), (ii) extrapolating to the target budget, and (iii) extending training to verify the forecast. Across all axes below we observe clean, predictive fits whose extrapolated curves align with the extended trajectories, mirroring the behavior seen



(a) (b) (c)

Figure 6: **AIME 2025:** Even though (a) sequence length and (c) batch size show an inverse trend initially where smaller values seem better at lower compute budget; but because of their higher asymptotic performance, larger sequence length and batch size reach a higher value for the downstream evals. (b) larger models generalize better from the start as evident with the 17Bx16 vs a 8B dense model.

in our 100k GPU-hour run (Fig. 1a) and the cross-recipe comparison in Fig. 2. As in Appendix A.4, asymptote estimates vary within ±0.02 pass-rate, which we use as the natural error bar for all extrapolation claims.

Generation length (context budget). Increasing the generation length from 14k to 32k tokens slows early progress (lower B) but consistently lifts the fitted asymptote (A), yielding higher final performance once sufficient compute is provided (Figure 5a). This validates long-context RL as a ceiling-raising knob rather than a mere efficiency trade-off. Extrapolations made from the 14k-token fit correctly forecast the higher 32k-token trajectory within the ±0.02 error band when training is extended.

Global batch size. Larger batches reliably improve the asymptote and avoid the downstream stagnation we observe in smaller-batch runs (Appendix A.14). In our largest math run, moving to batch 2048 both stabilized training and yielded a fit that extrapolated from up to 50k GPU hours to the final 100k point, also consistent with a full-trajectory refit (Fig. 1a). Figure 5c shows the same qualitative pattern at mid-scale: small batches may appear better early but are overtaken as compute grows.

Generations per prompt (fixed total batch) For a fixed total batch, is it better to allocate more prompts or more generations per prompt? Sweeping generations per prompt 8,16,24,32 (and adjusting prompts to keep total batch fixed) leaves fitted curves essentially unchanged (Appendix A.13), suggesting that—at moderate batch—this allocation is a second-order choice for both A and B. Clearer differences may emerge at much larger batches (e.g., 2k+), which we leave for future work.

Model scale (MoE) Does ScaleRL remain predictive and stable on larger models? Training the 17B×16 Llama-4 Scout MoE with ScaleRL exhibits the same predictable scaling behavior as the 8B model, with low truncation rates and no instability pathologies noted for alternative losses (Appendix A.12). Figure 5b shows the training curve. The extended points align with the fitted curve, supporting the model-scale invariance of our recipe.

Multi-task RL (math + code). Joint training on math and code produces clean, parallel power-law trends per domain, with extended points staying on the extrapolated trajectories. This indicates that the same compute-performance law applies under multi-task mixtures, and ScaleRL's benefits transfer across domains. We show the plots in Appendix A.13.

5 Related Work

We detail two most relevant works to our study in this section. ProRL (Liu et al., 2025a) demonstrates that prolonged RL fine-tuning on LLMs ($\sim$ 2000 optimization steps, $16k$ GPU-hours) using a mix of reasoning tasks (math, code, logic puzzles, etc.) uncovers novel solution strategies beyond a model's base capabilities. This longer training regimen delivered significant gains on a 1.5B model, rivaling the performance of larger models on some benchmarks. ProRL's contributions lie in specific

heuristics for stability (KL-regularization, policy resetting, entropy controls, etc.) to achieve the best performance out of a 1.5B model.

Liu et al. (2025c) offer a complementary perspective and ablates various design choices under consistent conditions on Qwen-3 4B/8B (Yang et al., 2025), and presents a minimalist combination, LitePPO, that outperforms more complex methods like GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025). This yields valuable algorithmic insights, but the focus is on comparative empirical findings, rather than on scaling behaviour. Moreover, RL training in their study is done at a smaller scale (800 - 1200 steps), and downstream improvements are smaller.

None of these work study "scaling" properties of these methods. In fact, the main comparisons are done on downstream evals, which are not the right metric to study scaling. Rather, as done in pre-training and in our work here, one needs to study performance on in-distribution held out eval set. In contrast to the mentioned related works, our work develops and validates a compute-performance framework with predictive fits, while operating at a much larger compute budget and model scale compared to the above studies. Additionally, our findings yield a compositional recipe that can scale to over 100,000 GPU-hours without any stability issues. The rest of the related work is deferred to Appendix Section A.1.

6 Conclusion & Discussion

In this work, we study the scaling properties of different techniques used in RL for LLMs in pursuit of a scalable recipe. With this mission, we derive a method for fitting predictive scaling laws for accuracy on the validation set that allows us to quantify the asymptotic performance and compute efficiency of an RL method. Using this methodology, our primary contribution is to conduct a careful series of ablations of several algorithmic options that go into the RL recipe. For each ablation, we choose the option with higher asymptotic performance when possible and improved efficiency otherwise. Combining these choices yields the ScaleRL recipe which scales better than all existing recipes in our experiments. A few observations are in order:

- Compute efficiency. The important insight of our scaling methodology is that we can use smaller-scale ablations in a systematic way to predict performance at larger scales. This allows us to create our final scalable recipe.
- Most important decisions. The off-policy algorithm, loss function, and model precision are the most important decisions from our ablations. Each of the other decisions does not have a large individual effect, but as we see from the leave-one-out experiments, they still do have some cumulative impact when all combined.
- Asymptotic performance vs. efficiency. For many of our ablations, we found the better option to improve both efficiency and asymptotic performance, but this is not always the case (e.g. for FP32). When doing the "forward" ablations starting from the baseline method, we opt for asymptotic performance first and foremost. Interestingly, when doing the "backward" leave-one-out ablations from the ScaleRL recipe, we find very little impact on asymptotic performance from each decision, but each component of the algorithm seems to help efficiency. This shows that the cumulative effect of the changes is quite robust.
- Generalization. Throughout we focus on validation accuracy on a held-out dataset from the training distribution of prompts. This still leaves an issue of generalization where the model must generalize from the training prompt distribution to held out test sets, which could be especially tricky in the RL setting where we take many epochs over the training prompt set. While a full characterization of generalization behavior is beyond the scope of our work, we do find generalization to work well most of the time. However, there are some algorithmic choices that seem to help generalization that we want to note here including: larger batch size, reducing truncations, and larger model scale.

There are of course many more design choices in the RL algorithm, so we don't think that our ScaleRL recipe is the end of the story. We hope that our focus on scalable RL and methodology for predicting scalability can inspire future work to push the frontier of RL for LLMs even further.

REFERENCES

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

AoPS. Aime problem set 1983-2025, 2025. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Yao Feng. Your efficient rl framework secretly brings you off-policy rl training, 2025. URL https://fengyao.notion.site/off-policy-rl. Accessed through a social media reference.

GLM-V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Bin Chen, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiale Zhu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingdao Liu, Mingde Xu, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Tianyu Tong, Wenkai Li, Wei Jia, Xiao Liu, Xiaohan Zhang, Xin Lyu, Xinyue Fan, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yanzi Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuting Wang, Yu Wang, Yuxuan Zhang, Zhao Xue, Zhenyu Hou, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL https://arxiv.org/abs/2507.01006.

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.

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. *arXiv preprint arXiv:2103.03874*, 2021.

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

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: An efficient rlhf algorithm with robustness to both prompt and reward models, 2025a. URL https://arxiv.org/abs/2501.03262.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025b.

Edward L Ionides. Truncated importance sampling. *Journal of Computational and Graphical Statistics*, 17(2):295–311, 2008.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025b.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025a.

Margaret Li, Sneha Kudugunta, and Luke Zettlemoyer. (mis)fitting: A survey of scaling laws, 2025b. URL https://arxiv.org/abs/2502.18969.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models, 2025a. URL https://arxiv.org/abs/2505.24864.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025b. URL https://arxiv.org/abs/2503.20783.

Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, Shengyi Huang, Siran Yang, Jiamang Wang, Wenbo Su, and Bo Zheng. Part i: Tricks or traps? a deep dive into rl for llm reasoning, 2025c. URL https://arxiv.org/abs/2508.08221.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

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

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

MiniMax, :, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du, Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui Du, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Enwei Jiao, Haigang Zhou, Haimo Zhang, Han Ding, Haohai Sun, Haoyu Feng, Huaiguang Cai, Haichao Zhu, Jian Sun, Jiaqi Zhuang, Jiaren Cai, Jiayuan Song, Jin Zhu, Jingyang Li, Jinhao Tian, Jinli Liu, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kaiyi Feng, Ke Yang, Kecheng Xiao, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Li, Lin Zheng, Linge Du, Lingyu Yang, Lunbin Zeng, Minghui Yu, Mingliang Tao, Mingyuan Chi, Mozhi Zhang, Mujie Lin, Nan Hu, Nongyu Di, Peng Gao, Pengfei Li, Pengyu Zhao, Qibing Ren, Qidi Xu, Qile Li, Qin Wang, Rong Tian, Ruitao Leng, Shaoxiang Chen, Shaoyu Chen, Shengmin Shi, Shitong Weng, Shuchang Guan, Shuqi Yu, Sichen Li, Songquan Zhu, Tengfei Li, Tianchi Cai, Tianrun Liang, Weiyu Cheng, Weize Kong, Wenkai Li, Xiancai Chen, Xiangjun Song, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xinzhu Hou, Xuan Lu, Xun Zou, Xuyang Shen, Yan Gong, Yan Ma, Yang Wang, Yiqi Shi, Yiran Zhong, Yonghong Duan, Yongxiang Fu, Yongyi Hu, Yu Gao, Yuanxiang Fan, Yufeng Yang, Yuhao Li, Yulin Hu, Yunan Huang, Yunji Li, Yunzhi Xu, Yuxin Mao, Yuxuan Shi, Yuze Wenren, Zehan Li, Zelin Li, Zhanxu Tian, Zhengmao Zhu, Zhenhua Fan, Zhenzhen Wu, Zhichao Xu, Zhihang Yu, Zhiheng Lyu, Zhuo Jiang, Zibo Gao, Zijia Wu, Zijian Song, and Zijun Sun. Minimax-m1: Scaling test-time compute efficiently with lightning attention, 2025. URL https://arxiv.org/abs/2506.13585.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. *arXiv preprint arXiv:2410.18252*, 2024.

OpenAI. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

OpenAI. Introducing OpenAI o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025. Accessed: 22 September 2025.

David Owen. How predictable is language model benchmark performance? *arXiv preprint arXiv:2401.04757*, 2024.

Alex Piche, Rafael Pardinas, Ehsan Kamalloo, and Dzmitry Bahdanau. Pipelinerl. 2025. URL https://huggingface.co/blog/ServiceNow/pipelinerl.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models, 2025. URL https://arxiv.org/abs/2406.19146.

Abhinav Rastogi, Albert Q Jiang, Andy Lo, Gabrielle Berrada, Guillaume Lample, Jason Rute, Joep Barmentlo, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, et al. Magistral. *arXiv preprint arXiv:2506.10910*, 2025.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance, 2024. URL https://arxiv.org/abs/2405.10938.

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

ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

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.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

xAI Team. Grok 4. 2025. URL https://x.ai/news/grok-4.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. What's behind ppo's collapse in long-cot? value optimization holds the secret. *arXiv preprint arXiv:2503.01491*, 2025.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Ruiqi Zhang, Daman Arora, Song Mei, and Andrea Zanette. Speed-rl: Faster training of reasoning models via online curriculum learning, 2025. URL https://arxiv.org/abs/2506.09016.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025a. URL https://arxiv.org/abs/2507.18071.

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*, 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, 2025c. URL https://arxiv.org/abs/2506.02177.

A Appendix

A.1 Related Work

A wave of recent work has applied Reinforcement Learning (RL) to improve the reasoning abilities of large language models (LLMs); often achieving state-of-the-art results on challenging tasks (OpenAI, 2024; Guo et al., 2025; Seed et al., 2025). OpenAI's o1 series of models established that large-scale RL can substantially enhance long-horizon reasoning, but did not release any details on how these models were trained. Deepseek R1 (and R1-Zero) (Guo et al., 2025) provided the first comprehensive study on training high-performing and long Chain-of-Thought (CoT) models primarily via RL, documenting emergent behaviours under extended RL without any reliance on reward models (Lightman et al., 2023) or Monte Carlo Tree Search (MCTS) (Xie et al., 2024).

The earliest widely referenced RLVR (verifiable-reward) algorithm underlying this wave of reasoning development is Group Relative Policy Optimization (GRPO), introduced in Shao et al. (2024). GRPO is a critic-free, group-relative policy gradient with PPO-style clipping that replaces a learned value baseline with group baselines to reduce computational cost and stabilize credit assignment for long CoTs. While GRPO catalyzed rapid progress, subsequent work document its limitations (token-level clipping, model collapse risks) and motivate different group- or sequence- level variants (Yu et al., 2025; Yue et al., 2025; Hu et al., 2025b; Zheng et al., 2025b).

Yu et al. (2025) propose the Decoupled clip and Dynamic Sampling Policy Optimization (DAPO), where they decouple $\epsilon_{\texttt{low}}$ and $\epsilon_{\texttt{high}}$ clipping in the GRPO objective and do *Clip-Higher* for $\epsilon_{\texttt{high}}$ to avoid entropy collapse. Furthermore, they do dynamic sampling of prompts in a given batch to avoid samples with zero variance (or advantage) which contribute zero policy gradients. Finally, they employ token-level loss aggregation unlike GRPO, which uses sample-level loss averaging. With these modifications, they are able to surpass the vanilla GRPO baseline while avoiding entropy collapse in the RL training. In parallel, Yue et al. (2025) develop VAPO; a value-augmented PPO tailored for long CoTs with strong stability and outperforming value-free baselines like GRPO and DAPO. They combine value pre-training and decoupled Generalized Advantage Estimation (GAE) from VC-PPO (Yuan et al., 2025), loss objective modifications from DAPO, and propose length-adaptive GAE to come up with an open recipe, VAPO, that has been used to train large MoE models in Seed et al. (2025). Similarly, other technical report like Magistral (Rastogi et al., 2025), Kimi-k1.5 (Kimi Team et al., 2025b), Minimax-01 (Li et al., 2025a) detail various details on their RL training recipes, but don't share extensive experiments on why their design choices are better than the baselines.

A.2 Background

Group Relative Policy Optimization (GRPO) GRPO (Shao et al., 2024) adapts PPO Schulman et al. (2017) for LLM fine-tuning with verifiable rewards. For a given prompt x, the old policy

$\pi_{\text{gen}}(\theta_{\text{old}})$ generates G candidate completions $\{y_i\}_{i=1}^{G}$, each assigned a scalar reward r_i. To emphasize relative quality within the group, rewards are normalized as

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^{G})}{\text{std}(\{r_j\}_{j=1}^{G}) + \epsilon}. \tag{4}$$

Each completion y_i of length $|y_i|$ contributes at the token level through ratios

$$\rho_{i,t}(\theta) = \frac{\pi_{train}(y_{i,t} \mid x, y_{i,<t}, \theta)}{\pi_{gen}(y_{i,t} \mid x, y_{i,<t}, \theta_{\text{old}})}. \tag{5}$$

The GRPO objective averages across both completions and tokens:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{gen}}(\cdot|x,\theta_{\text{old}})}} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}(\rho_{i,t}(\theta), 1\pm\epsilon)\hat{A}_i\Big) \right] \tag{6}$$

Thus GRPO preserves token-level policy ratios as in PPO, while using sequence-level, group-normalized advantages to stabilize learning under sparse rewards.

Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) DAPO (Yu et al., 2025) extends GRPO with two key modifications. First, it replaces symmetric clipping with *asymmetric clipping*, using distinct thresholds for upward and downward deviations: $\text{clip}_{\text{asym}}(\rho, a) = \text{clip}(\rho,\ 1-\epsilon^-, 1+\epsilon^+)$, where ϵ^- and ϵ^+ are hyper-parameters.

Second, DAPO changes the aggregation scheme to operate at the *prompt level*. For a given prompt $x \sim D$, the old policy produces G completions $\{y_i\}_{i=1}^{G}$ with advantages $\{\hat{A}_i\}$ (Equation (4)). Let $T = \sum_{i=1}^{G} |y_i|$ denote the total number of tokens across all completions. With token-level ratios as in Equation (5). The DAPO surrogate objective is

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{gen}}(\cdot|x,\theta_{\text{old}})}} \left[\frac{1}{T}\sum_{i=1}^{G} \sum_{t=1}^{|y_i|} \min\Big(\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}_{\text{asym}}(\rho_{i,t}(\theta), \hat{A}_i)\hat{A}_i\Big) \right] \tag{7}$$

This prompt-level normalization ensures that each token contributes equally to the prompt's loss, regardless of the number or length of its sampled completions. DAPO also introduces dynamically dropping 0-variance prompts from the batch during training and filling the batch with more prompts until the batch is full. We skip that change here since its effect is similar to having a larger batch size.

A.3 Training Setup

Datasets For the small-scale SFT, we use a curated datamix of reasoning traces. We filter this dataset by removing trivial prompts, discarding solution traces exceeding $12k$ tokens, and decontaminating with AIME 2024/2025 (AoPS, 2025) and MATH-500 (Hendrycks et al., 2021) benchmarks. For the RL stage, we use the Polaris-53K dataset (An et al., 2025) for most of our runs; additionally using the Deepcoder dataset (Luo et al., 2025) for runs with both math and code.

Supervised Fine-tuning We run SFT using a batch size of 2M tokens, max sequence length of 32768, and a learning rate of 3×10^{-5} using the AdamW optimizer (Loshchilov & Hutter, 2019) on 32 H100 GPU nodes for approximately 4 epochs and 32B tokens in total.

Reinforcement Learning We allocate 14k generation budget during RL training, where 12k tokens are allocated to the intermediate reasoning ("thinking"), followed by 2k tokens for the final solution and answer. We sample 48 prompts in each batch, each with 16 generations per prompt. Thus, we get the total batch size as 768 completions per gradient update step. The rewards are given as ± 1 to correct and incorrect traces respectively.

We use automated checkers like Sympy (Meurer et al., 2017) or Math-Verify[1] for assessing the correctness of the final answer for math problems after stripping out the thinking trace

[1] https://github.com/huggingface/Math-Verify

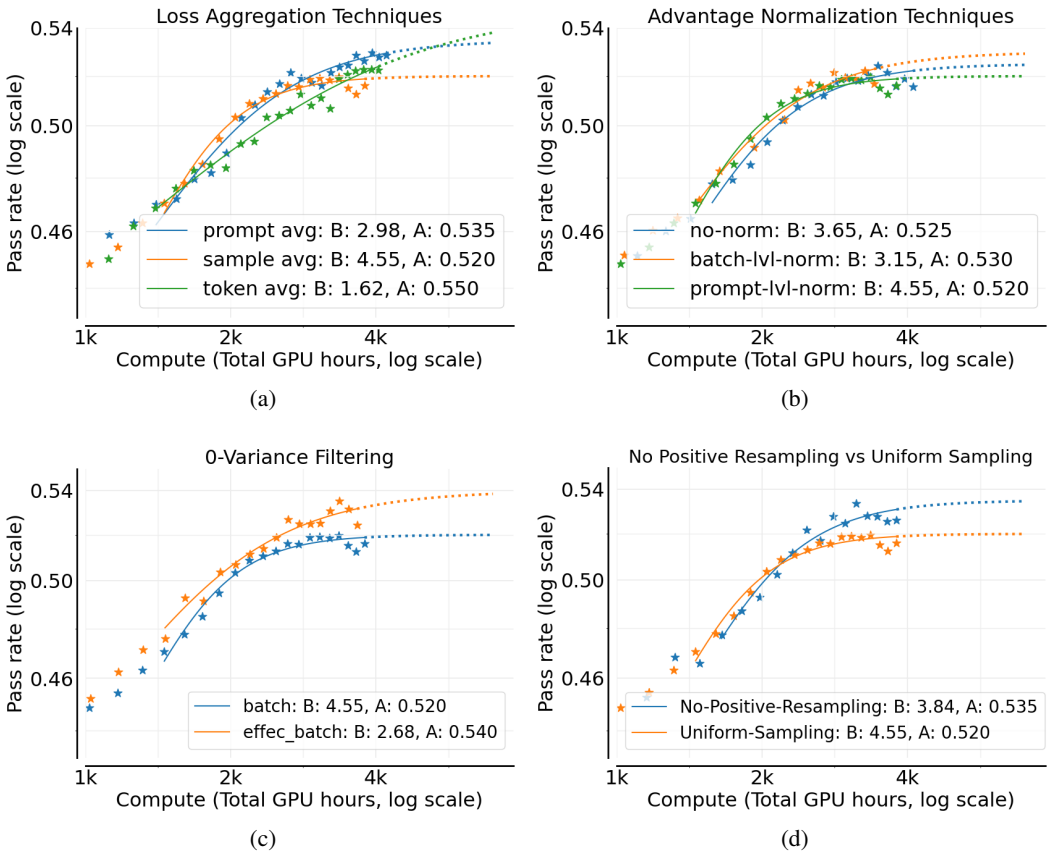


Figure 7: Comparing (a) loss aggregation, (b) different advantage normalization techniques, (c) "zero" variance filtering, and (d) adaptive prompt sampling.

(<think>···</think>). We use a custom code execution environment for coding problems involving unit tests and desired outputs.

We use a constant learning rate of 5×10^{-7}, AdamW optimizer (Loshchilov & Hutter, 2019) with $\epsilon = 10^{-15}$, weight decay of 0.01, and a linear warmup of 100 steps. We use 80 Nvidia GB200 GPUs for a single run, with a compute budget ranging from 3.5-4K GPU hours for ablating different design choices, 16K for the leave-one-out experiments, and finally 30k-100K GPU hours for our larger scale runs.

A.4 Forward Ablations

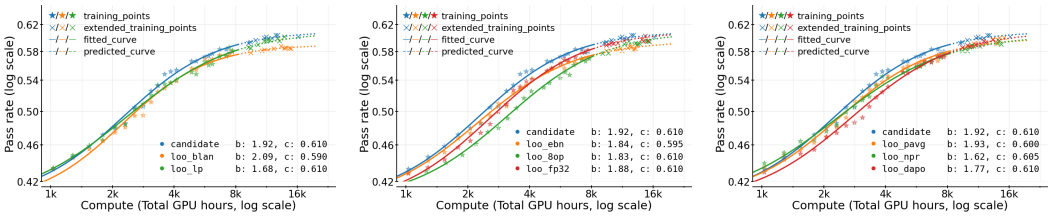
We show additional results for Section 3.2 in Figures 7a-7d.

A.5 Leave One Out (LOO) Ablations

We plot the remaining leave one out plots in Figure 8

A.6 Comparing algorithms

Consistent with observations in large-scale pre-training, where the loss exhibits a sharp initial drop before settling into a predictable power-law decay (Li et al., 2025b), we observe a similar two-phase behavior in RL. The mean reward increases rapidly, almost linearly, during the first ∼epoch (∼ 1.1k steps), after which the curve follows power-law behavior. Our sigmoidal-law fits are applied to this latter portion of the training curve.



(a) (b) (c)

Figure 8: Comparison of different leave-one-out strategies using 16k GPU-hours compute budget.

Unlike pre-training, our goal is not to predict the performance of a fixed recipe, but to identify which algorithms and design choices scale reliably. Achieving highly robust fits typically requires very large runs with hundreds or thousands of evaluation points, which is impractical in our setting for two reasons. First, running all ablations at such scale would be computationally prohibitive. Second, many RL algorithms we compare are themselves not scalable to such extreme budgets: they often saturate much earlier or even degrade with more compute due to instability. For example, our baseline method (Section 3.2) destabilizes beyond $\sim$ 3500 GPU-hours, since overlong generation truncations exceed 10% of generations - reducing the effective batch size. See A.15 for the extended training curve.

As we ablate across different axes in Section 3.2, we discover design choices that improve stability at higher compute. Some ablated variants can scale further, e.g., $\sim$ 5k GPU hours for $\epsilon = 0.26$ in DAPO, $\sim$ 6k GPU hours with the FP32 precision fix (Section 3.2), and $\sim$ 7k GPU hours for CISPO. Once we combine the best design choices, we obtain a stable and scalable recipe, which allows us to run leave-one-out (LOO) experiments for $\sim$ 1200 GPU hours per run.

A.7 Fitting curves

We fit the power-law equation in Equation (1) to the mean reward on our held-out evaluation set. This set consists of 1k prompts held out from the POLARIS (An et al., 2025) math dataset, with 16 generations sampled every evaluation step performed at 100 steps intervals.

Directly fitting all three parameters $\{a, b, c\}$ is challenging. Instead, we perform a grid search over $c \in \{0.45, 0.46, \ldots, 0.8\}$, and for each candidate c fit a and b. The best fit (measured by sum of squared residuals) across this grid is selected as the final curve. We use SciPy's `curve_fit` with default initialization; varying initialization strategies produced identical results.

To estimate the error margin of our fits, we trained three independent runs of ScaleRL with batch size 768 and generation length 14k (as used in Section 3.3). For these runs, we refined the grid search with c increments of 0.005 (i.e., $c \in \{0.455, 0.460, 0.465, \ldots, 0.800\}$). We found that the fitted c values varied by at most ± 0.02, suggesting this as a reasonable error margin on asymptotic performance estimates. Estimating the error margin for fitted b value is tough, as different algrithms with different c values can have different error margin for b. However, for the purpose of comparing algorithms, we can safely deduce that if two methods achieve similar c values (within 0.02), the one with higher b is at least as good in terms of scalability (Section A.6).

A.8 Controlling generation length

One common concern in reasoning RL is to control exploding generation lengths, which harms both training efficiency and stability (Appendix A.15). We consider two approaches: (a) *interruptions*, used in works like GLM-4.1V (GLM-V Team et al., 2025), and Qwen3 (Yang et al., 2025) and (b) *length penalties*, used in works like DAPO (Yu et al., 2025), Kimi (Kimi Team et al., 2025b), Magistral (Rastogi et al., 2025), and Minimax-M1 (MiniMax et al., 2025).

Interruptions forcibly stop generation by appending a marker phrase such as "Okay, time is up. Let me stop thinking and formulate a final answer `</think>`", signaling the model to terminate its

reasoning and produce a final answer. In our setup, the interruptions tokens are placed randomly in between $[10k, 12k]$ token length, to induce generalization to different generation lengths.

Length penalties instead reshape the reward. Following DAPO (Yu et al., 2025), we penalize overly long completions with a tolerance interval L_{cache}:

$$R_{\text{length}}(y) = clip\left(\frac{L_{\max} - |y|}{L_{\text{cache}}} - 1, -1, 0\right) \tag{8}$$

This penalty is added only to the correct traces, discouraging excessively long generations. In the length penalty experiment, we set $L_{\max} = 14$k tokens and $L_{\text{cache}} = 2$k tokens.

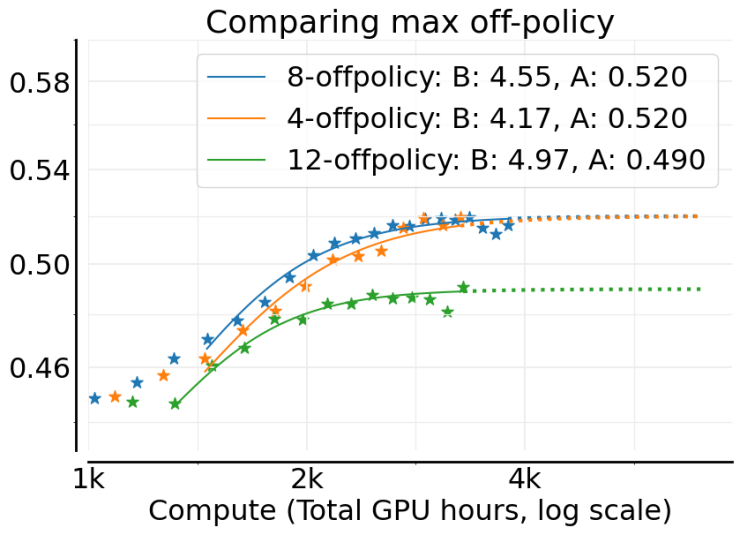
In Section 3.3, we compare length penalty and interruption at a scale of 12k GPU-Hours. We find that replacing interruption with length penalty in our final ScaleRL recipe does not improve performance.

A.9 PipelineRL

Using the baseline setup, we ablated the off-policy parameter in PipelineRL (Figure 9a). Both 4 and 8 off-policyness performed equally well, and we adopt 8 as the default setting when updating the baseline in Section 3.1.

Why does PipelineRL consistently outperform the classic PPO-off-policy approach (Sections 3.1 and 3.3)? We attribute this to its closer alignment with on-policy training. In PPO-off-policy, generation and training proceed in alternating phases: the trainer operates strictly on batches that are as off-policy as the chosen parameter k, making updates based on stale rollouts. In contrast, PipelineRL operates in a streaming fashion. As soon as a batch is available, it is passed to the trainer; likewise, as soon as a model update is ready, it is shared back to the generators, who immediately use it—including in the continuation of partially generated traces. This tight feedback loop keeps training closer to the on-policy regime, reducing the mismatch between generator and trainer distributions.

Importantly, this distinction affects the asymptotic performance c of the scaling curve, not just the efficiency exponent b. Very few axes shift the asymptote in this way, making the choice of off-policy algorithm one of the most consequential design decisions in RL post-training.



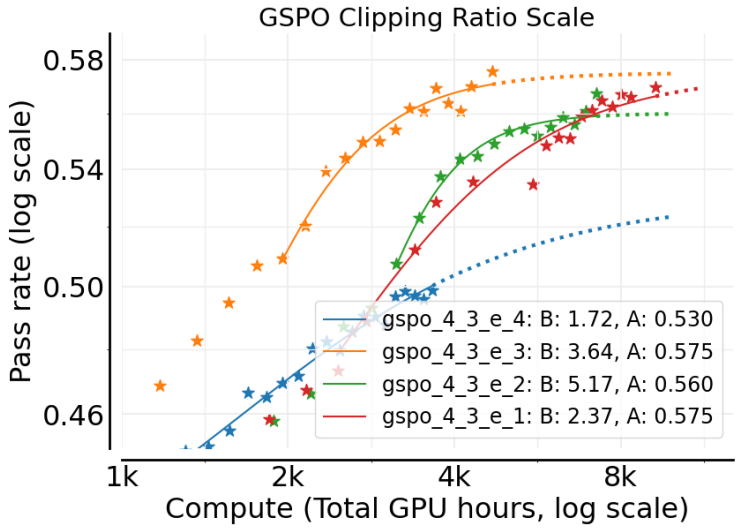
(a)

Figure 9: Different off-policy runs with PipelineRL

A.10 GSPO ABLATIONS

We ablate the clipping-ratio scale used in GSPO, as shown in Figure 10a. The 10^{-3} scale consistently performs as well as, or better than, alternatives. Given this scale, we further varied the upper clipping ratio in $\{4\times10^{-3}, 5\times10^{-3}, 7\times10^{-3}\}$ and found $\{5\times10^{-3}\}$ yielded the at least as good fit as the other choices.

An important observation is that GSPO is quite robust to the choice of clipping ratio. Once the correct scale is identified, most nearby values or even larger scale perform similarly. This robustness contrasts sharply with DAPO-style losses, which are highly sensitive to the exact value of the higher clipping ratio, as noted in Section 3.2.



(a)

Figure 10: GSPO Scale comparison. gspo_x_y_e_z in the legend means an upper and lower threshold of $\{x \times 10^{-z}$ and $y \times 10^{-z}\}$ respectively.

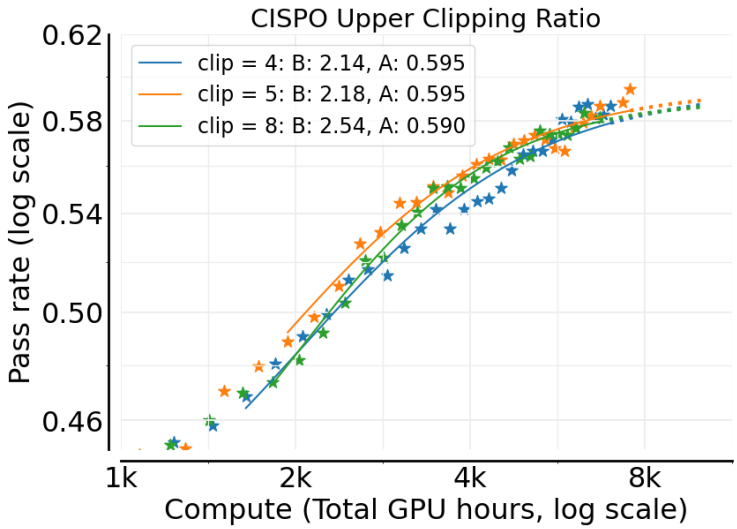
A.11 CISPO CLIPPING RATIOS

We ablate the higher clipping ratio for CISPO, keeping the lower clipping ratio fixed at 0 (Figure 11a). Across a wide range of values, we find little difference in performance, indicating that CISPO is largely insensitive to this hyperparameter. This robustness mirrors our findings for GSPO (Section A.10), and stands in contrast to DAPO/GRPO-style objectives, which are highly sensitive to the exact choice of clipping threshold. Such stability under hyperparameter variation makes CISPO a strong candidate for default use in large-scale training.

A.12 ENTROPY

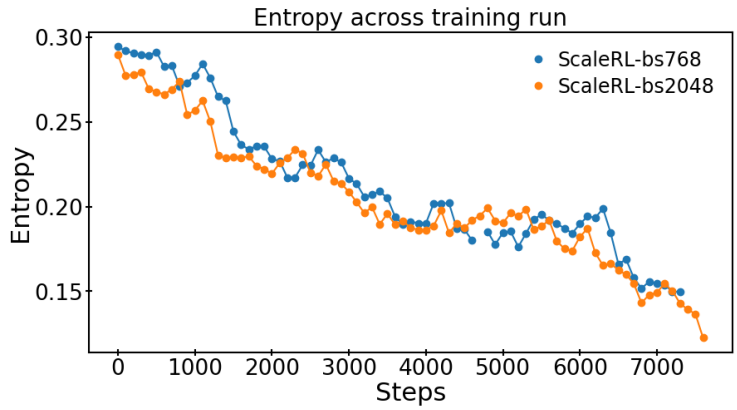
We tracked entropy on the held-out in-distribution evaluation set throughout training. Across all experiments—spanning variations in batch size, number of tasks, generation length, and model scale—we observed a consistent overall decrease in entropy. We show this in Figure Figure 12a For brevity, we report only the most representative and insightful plots here.

In Figure 12a, we plot entropy for ScaleRL runs with batch sizes 768 and 2048. Despite the 2048-batch size run achieving much stronger downstream performance at every stage (Figure 6c), both runs followed nearly identical entropy trajectories per step. This highlights an important point - although entropy is sometimes used as a proxy for exploration, simply maintaining higher entropy does not translate into better generalization. Instead, larger batches reduced effective exploration yet still yielded substantially better performance, underscoring batch size as an important decisive factor.



(a)

Figure 11: CISPO clipping ratio ablations



(a)

Figure 12: (a) Comparing entropy of large and smaller batch size runs

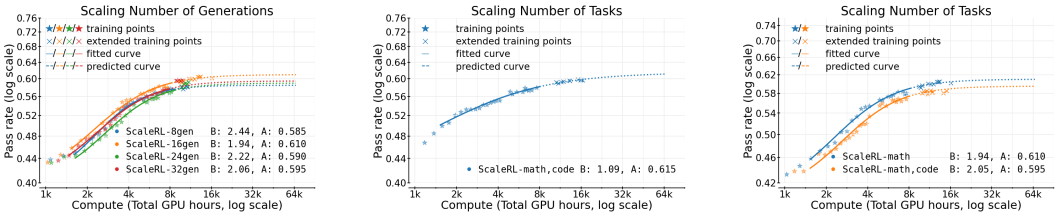
Overall, our findings suggest that while entropy decreases consistently during training, it is not by itself a reliable predictor of downstream performance. This observation reinforces the need to focus on algorithmic and scaling choices (e.g., batch size, off-policy method) rather than entropy dynamics when aiming for improved performance, both on training distribution as well as downstream task distribution.

A.13 Scaling on multiple axes

We provide the remaining scaling to different axes figure here in Figure 13

A.14 Downstream performance

In Figures 6a to 6c, we report a representative set of downstream evaluation curves. These include ScaleRL runs with batch sizes $\{512, 768, 2048\}$, long-context training run with 32k generation length, the large-model (Scout) training run, and a multi-task run (math + code). For each setting we plot performance against compute, and in some cases also against training steps for comparabil-



(a) (b) (c)

Figure 13: Scaling to (a) different number of generationes per prompt, (b) Code performance on math+code run, (c) Math performance on math+code run

ity across runs of different scales (to avoid smaller runs being visually compressed on the compute axis).

Two main patterns emerge. First, smaller-batch runs show early stagnation on downstream benchmarks even as in-distribution performance continues to improve. Second, larger-batch runs avoid this stagnation and instead exhibit downstream scaling curves that mirror their in-distribution power-law behavior. This supports the conclusion that larger batch sizes should be preferred for stable and scalable downstream performance.

A.15 Truncations and training instabilities

Across our experiments we found that training instabilities were often linked to truncations. As generation length grew, many RL runs exhibited fluctuating truncation rates that sometimes increased over training. At batch size 768, we observed that truncations in the range of 10–15% typically destabilized training, with performance degrading and not recovering without intervention. Examples include the extended GRPO run in Figure 2, where instability correlated with rising truncation rates, and the updated baseline used in Section 3.2.

By contrast, ScaleRL runs were more stable. On the 8B model, truncations remained below 5% for over 90% of training. At batch size 2048, truncations were slightly higher, occasionally approaching $\sim 7\%$. This increase was largely attributable to longer average generation lengths observed during training, which naturally raise the chance of exceeding the budget. Nevertheless, because the effective batch size (after excluding truncated samples) remained large, training stability was preserved. Intuitively, larger generation length budget should help reduce truncations. Training with 34k generation length (batch 768) remained stable - truncations briefly spiked to $\sim 4\%$ but quickly fell below 2%.

Larger models were even more robust. On the Scout run, truncations remained consistently below 2%, and for $> 90\%$ of training steps were under 1%. This likely reflects both the inherent ability of larger models to regulate generation length and their stronger instruction-following ability, which made interruption signals more effective.

Overall, we suggest practitioners monitor truncation rates closely. Our findings indicate that high truncation rates are a reliable warning signal of instability, while larger models, higher generation budgets, and careful design choices (as in ScaleRL) substantially mitigate this risk.

A.16 Loss Type - Stability and Robustness

As discussed in Section 3.2, GRPO/DAPO-style losses are highly sensitive to the choice of clipping ratio hyperparameter $\epsilon_{\max}$. In contrast, CISPO and GSPO show far greater robustness. For example, in Appendix Section A.11, varying $\epsilon_{\max}$ for CISPO between $\{4, 5, 8\}$ produced no significant differences in performance. For GSPO, the 10^{-4} clipping scale used in the original paper (Zheng et al., 2025a) did not work well in our setting. We therefore ablated across broader scales and found that once the correct order of magnitude was identified (e.g., 4×10^{-3} and higher), performance was stable and largely insensitive to fine-grained changes (e.g., $\{4\times10^{-3}, 5\times10^{-3}, 7\times10^{-3}\}$).

That said, we encountered additional stability issues with GSPO. On multiple occasions, GSPO runs diverged mid-training, leading to sudden drops in performance. For 8B models, restarting from a stable checkpoint allowed recovery, but this strategy failed on larger models such as Scout, where instability persisted despite repeated resetting to a stable checkpoint. While we checked to the best of our ability for any implementation bugs, we did not find one.

Overall, while all three loss families can be competitive under tuned settings, CISPO offers the best balance of stability and robustness to hyperparameters, making it our recommended choice.