Extended Abstract

Motivation Online RL fine-tuning can improve small-model reasoning when answers are automatically verifiable, but it often allocates compute uniformly. During training, uniform prompt sampling can spend updates on prompts that are already solved or too difficult to provide reward contrast. During inference, Best-of-N sampling carries every candidate to completion, even when some trajectories appear unlikely to succeed. We ask whether a single success-probability signal can allocate compute more efficiently in both phases on Countdown with Qwen2.5-0.5B.

Method We study two uses of this signal, evaluated independently. First, **Frontier-Curriculum RLOO** keeps the RLOO objective fixed and changes only the prompt sampling distribution. It identifies *frontier prompts* whose K rollouts contain both correct and incorrect completions, then replays structurally similar prompts so that training batches contain more within-prompt reward contrast. Second, **ZIP-RC-Lite** adapts ZIP-RC [14] as a frozen-backbone inference signal. A small reserved-logit head predicts final reward and remaining length from unused vocabulary logits in the same forward pass as next-token prediction, enabling adaptive pruning or early stopping without an additional model call.

Implementation For Frontier-Curriculum RLOO, we use a bounded frontier memory, task-specific numeric prompt embeddings, duplicate filtering, and a replay ratio of $\rho = 0.10$, while leaving the verifier reward, optimizer, KL penalty, entropy bonus, SFT initialization, and RLOO update unchanged. For ZIP-RC-Lite, we freeze the Qwen2.5-0.5B backbone and train only a 14–21-logit reserved output head. Because Qwen2.5-0.5B ties input and output embeddings, we untie and clone the output head before training to preserve generation behavior.

Results In a single-run 128-sample Countdown evaluation, Frontier-Curriculum RLOO improves average verifier score from 0.46 to 0.52 and pass@1 from 0.41 to 0.49 over vanilla RLOO, while pass@128 remains at 0.82. This indicates improved low-sample efficiency rather than a higher large-sample pass rate. ZIP-RC-Lite is well calibrated on held-out trajectories, with AUC 0.922 and $K{=}64$ total variation 0.52, lower than the reported ZIP-RC-Lite baseline of 0.63 under the comparison protocol used here. In live adaptive decoding over 3 seeds, pruning reduces active forward passes by approximately 63%, while early stopping reduces latency by approximately 48%, both at near-baseline accuracy. A more conservative held-out blend of adaptive-K and pruning gives a 20%–25% compute saving at neutral accuracy.

Discussion The Frontier-Curriculum result matches the RLOO estimator: mixed-outcome prompts produce positive and negative leave-one-out advantages in the same rollout group, while all-correct or all-failed prompts provide little contrast. **Additional experiments and ablations** include failure-based retrieval, replay-ratio sweeps, Goldilocks frontier selection, taxonomy-weighted replay, Teacher-SFT, the structured three-outcome ZIP-RC-Lite verifier, standalone adaptive-K, and the adaptive-K+pruning blend. These studies support the main interpretation: frontier replay is more useful than replaying only hard failures, and ZIP-RC-Lite is best understood as adaptive compute allocation rather than improved answer selection. The main limitations are the single training seed for Frontier-Curriculum RLOO, task-specific retrieval features, small ZIP-RC-Lite held-out pools, and Countdown's narrow difficulty range.

Conclusion A single success signal can improve efficiency in both phases of verifiable reasoning. Frontier-Curriculum RLOO uses it to focus training on prompts near the model's current learning boundary, while ZIP-RC-Lite uses it to reduce inference compute or latency without extra forward passes. Together, these results suggest a path toward a unified controller for training-time prompt selection and inference-time compute allocation.

Frontier Curriculum and Adaptive Test-Time Compute for Efficient RLOO

Marco Vizcarra
Department of Electrical Engineering
Stanford University
marcoviz@stanford.edu

Andy Kim
Graduate School of Business
Stanford University
andyskim@stanford.edu

Abstract

Online RL can improve language-model reasoning, but uniform prompt sampling can waste updates on prompts that are already solved or too hard to provide useful learning signal. We study RLOO fine-tuning of Qwen2.5-0.5B on Countdown and introduce Frontier-Curriculum RLOO. The method keeps the RLOO update unchanged while replaying prompts where the model sometimes succeeds and sometimes fails. These prompts give stronger leave-one-out reward contrast than prompts that are always solved or always missed. In 128-sample evaluation, Frontier-Curriculum RLOO improves average verifier score from 0.46 to 0.52 and pass@1 from 0.41 to 0.49 over vanilla RLOO. We then study adaptive test-time compute with ZIP-RC-Lite, a frozen-backbone success predictor that reads a small set of unused vocabulary logits without extra model forward passes. Its value estimate is well-calibrated on held-out trajectories (AUC 0.92), and a live decoder that uses this signal cuts roughly 63% of compute through pruning or roughly 48% of latency through early stopping, both at near-baseline accuracy. We frame these results as evidence that success estimates are useful for compute allocation, while noting that Frontier-Curriculum RLOO should be validated with additional seeds.

1 Introduction

Online reinforcement learning is especially useful when model outputs can be scored with verifiable feedback. We study Countdown, an arithmetic reasoning task in which a model must construct an expression that reaches a target number using a fixed set of allowed numbers [15]. Although answers are easy to check, the task remains challenging for small language models because it requires arithmetic search and constraint following. This setting connects to verifier-guided mathematical reasoning, where sampled candidate solutions can be checked after generation [6; 12].

We focus on RLOO fine-tuning [1]. For each prompt x, RLOO samples K completions and compares their verifier rewards using a leave-one-out baseline. Thus, prompt choice matters because prompts where all completions succeed or all completions fail provide little reward contrast, while mixed-outcome prompts provide a stronger learning signal.

This motivates **Frontier-Curriculum RLOO**. We define frontier prompts as prompts whose sampled completions contain both successes and failures, placing them near the model's current learning boundary. Our method keeps the RLOO update, verifier reward, optimizer, and training hyperparameters fixed, and changes only the prompt sampling distribution. Specifically, we replay a small fraction of frontier-similar prompts from a prompt bank while sampling the rest of each batch uniformly.

We also use the same success signal for adaptive test-time compute. Standard Best-of-N sampling carries every candidate to completion, even when some trajectories appear unlikely to succeed. With ZIP-RC-Lite, a frozen Qwen2.5-0.5B reads unused vocabulary logits to estimate trajectory

success without extra forward passes, enabling pruning of low-value samples or early commitment to confident ones.

The paper makes three contributions. First, we introduce Frontier-Curriculum RLOO, a controlled prompt-sampling change that targets mixed-outcome prompts. Second, we connect the frontier criterion to leave-one-out advantage contrast and evaluate it on Countdown. Third, we adapt ZIP-RC-Lite to Qwen2.5-0.5B and show that the same success signal can reduce inference compute or latency without extra forward passes.

2 Related Work

RL fine-tuning and verifier rewards. Modern language-model post-training commonly combines supervised fine-tuning with preference optimization or online reinforcement learning. DPO and IPO learn from paired preferences [18; 3], while RLOO is a REINFORCE-style method that samples multiple completions for the same prompt and uses a leave-one-out baseline [22; 10; 1]. Verifier-based reasoning tasks make this setup especially useful because final answers can be checked automatically [6; 12]. Our work differs by keeping the verifier reward and RLOO estimator fixed, isolating prompt sampling as the only intended training-time change.

Curriculum learning and replay. Curriculum learning, self-paced learning, and automated curricula show that example order can affect learning [4; 11; 8]. Prioritized replay provides a related RL mechanism by sampling transitions according to estimated utility [19; 9]. Recent LLM reasoning curricula adapt prompt sampling using difficulty estimates, category policies, or learning-edge statistics [16; 5; 17; 23]. In contrast, Frontier-Curriculum RLOO does not require a teacher, learned difficulty model, or separate curriculum policy; it uses the rollout outcomes already produced by RLOO.

Frontier examples. Several recent reasoning methods prioritize examples near the model's learning frontier. LILO emphasizes questions with high success variance, where the model sometimes succeeds but not always [7]. Goldilocks RL selects questions that are neither too easy nor too hard for the current model [13]. HIVE tracks a moving learning edge to avoid low-utility rollouts [23]. Our contribution is a controlled RLOO-specific instantiation. We define a frontier prompt as one whose K sampled completions include at least one correct and at least one incorrect answer, then retrieve structurally similar Countdown prompts without changing the objective or reward.

Adaptive test-time compute. A growing body of work spends extra computation at inference rather than during training. Best-of-N sampling draws many candidates and keeps the best; self-consistency aggregates them by majority vote [21], and learned verifiers or value models rerank or select among candidates [6; 12]. Scaling studies show that, for a fixed model, well-allocated test-time compute can rival training a larger one [20]. ZIP-RC removes the usual cost of an allocation signal by predicting reward and remaining length from a frozen policy's own logits, without an extra forward pass [14]. We adapt this idea to Qwen2.5-0.5B on Countdown and use the success signal for concrete allocation decisions such as pruning low-value samples, committing early to confident trajectories, and reallocating samples across prompts.

3 Method

3.1 Frontier-Curriculum RLOO

Figure 1 shows how frontier prompts are identified, stored, retrieved, and mixed into the next training batch.

RLOO update. We use RLOO as the base online reinforcement learning algorithm. For a prompt x, the policy samples K completions $y_1, \ldots, y_K \sim \pi_\theta(\cdot \mid x)$, and the verifier scores each completion with reward $r_i = R(y_i, x)$. RLOO compares each completion against the other completions from the *same* prompt through a leave-one-out baseline [1; 10]

$$b_i = \frac{1}{K-1} \sum_{j \neq i} r_j, \qquad A_i = r_i - b_i, \qquad \widehat{\nabla_\theta J_{\mathrm{RLOO}}}(x) = \frac{1}{K} \sum_{i=1}^{K} A_i \, \nabla_\theta \log \pi_\theta(y_i \mid x).$$

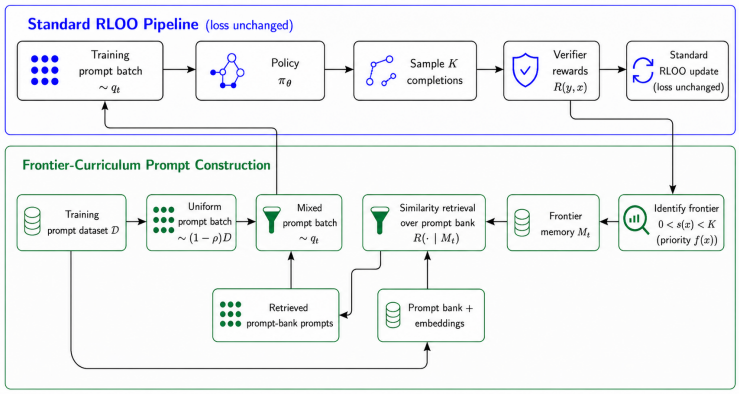

Figure 1: Frontier-Curriculum RLOO. Frontier prompts are stored, retrieved, and mixed with uniform prompts while keeping the RLOO update unchanged.

Our method does not modify this estimator. It changes only the distribution of prompts used to form each training batch.

Frontier signal. The prompt distribution matters because RLOO is most informative when sampled completions for the same prompt produce reward contrast. Let $s(x)$ be the number of sampled completions that are fully correct, and let $p(x) = s(x)/K$ be the prompt-level success rate. We call x a *frontier prompt* when its rollout group contains both successful and unsuccessful completions, $0 < s(x) < K$. To prioritize frontier prompts, we use the variance of the binary success signal. This quantity is zero when all samples fail or all samples succeed, and it is largest when the model succeeds on about half of the samples. We therefore assign each frontier prompt the priority

$$f(x) = 4p(x)\big(1 - p(x)\big).$$

The factor 4 normalizes the score so that its maximum value is one at $p(x) = \frac{1}{2}$. This priority is used only for curriculum memory and retrieval. It is not added to the reward, loss, or advantage estimate. Appendix B.1 derives the connection between this score and leave-one-out advantage contrast.

Selection and replay. At each training step, we first identify frontier prompts in the current rollout batch and store them in a bounded memory buffer with their priorities. We then retrieve structurally similar Countdown prompts from a prompt bank built from the training set. Retrieval uses task-structured numeric embeddings and duplicate filtering based on the target and allowed numbers, with the full procedure described in Appendix B. The retrieved prompts replace a fraction ρ of the uniform batch, giving the sampling distribution

$$q_t = (1 - \rho)D + \rho R(\cdot \mid M_t),$$

where D is the uniform dataset distribution, M_t is the frontier memory, and $R(\cdot \mid M_t)$ is the retrieval distribution. Training then proceeds with the standard RLOO update.

In the main run, $\rho = 0.10$, so a batch of 64 prompts contains approximately 58 uniform prompts and 6 retrieved prompts once memory is active. This preserves broad dataset coverage while allocating a small fraction of training to prompts near the model's current learning boundary. All other training components remain fixed, including the verifier reward, RLOO objective, KL penalty, entropy bonus, optimizer settings, SFT initialization, and reference model.

3.2 Adaptive Test-Time Compute

Our test-time component reuses the same per-prompt/per-prefix success signal at inference. We adopt **ZIP-RC-Lite**, the frozen-backbone variant of Zero-overhead Inference-time Prediction of Reward and Cost (ZIP-RC) [14], and specialize it to Countdown / Qwen2.5-0.5B. The goal is to give the frozen policy near-free introspection: a small head reads a slice of unused vocabulary logits and, in the same forward pass that produces next-token probabilities, predicts a joint distribution over the final reward outcome and the number of tokens remaining to completion. Because this is just

a read of existing logits, it adds no extra model, no extra forward pass, and no architecture change at inference time. This is a stretch extension of the project; it is adapted from the official ZIP-RC repository (the dataset, head-training, and offline-scoring modules are derivatives), while the live adaptive decoder below has no upstream reference and was built from the paper's formulation.

Reserved-logit head. Qwen2.5-0.5B has a vocabulary of $151{,}936$ but its tokenizer only names ids up to $151{,}664$, leaving 271 unused embedding rows (ids $151{,}665$–$151{,}935$). We park the ZIP-RC joint head in the first contiguous slice of these unused ids, starting at id $151{,}665$. (The upstream ZIP-RC default of $151{,}669$ is a *Qwen3* id and is wrong for Qwen2.5; correcting it was one of several model-specific adaptations). The joint is a grid over R reward states and L remaining-length bins, flattened as $\mathrm{idx} = (\text{length bin}) + (\text{reward state}) \times L$, so the head occupies exactly $R \times L$ reserved logits: 14 for the binary verifier and 21 for the structured verifier.

Joint reward–cost distribution. Let the reserved-slice logits at prefix $x_{<t}$ be $z_t \in \mathbb{R}^{R \times L}$. The predicted joint is the softmax over the slice,

$$p_\theta(b, \ell \mid x_{<t}) = \frac{\exp z_t(b, \ell)}{\sum_{b'=1}^{R} \sum_{\ell'=1}^{L} \exp z_t(b', \ell')}.$$

Marginalizing recovers the two control signals: a value/confidence estimate

$$\hat{V}(x_{<t}) = \mathbb{E}[\text{reward}] = \sum_{b=1}^{R} \Big(\textstyle\sum_\ell p_\theta(b, \ell) \Big) r_b, \qquad \mathbb{E}[\ell \mid x_{<t}] = \textstyle\sum_{\ell=1}^{L} (\sum_b p_\theta(b, \ell))\, \bar{\ell},$$

where r_b is the reward value of state b and $\bar{\ell}$ is the midpoint of length bin ℓ. The value $\hat{V}$ is the confidence/selection signal; $\mathbb{E}[\ell]$ is the predicted "thinking time." We use two reward-state sets: a *binary* verifier $\{0, 1\}$ (R=2) and a *structured* 3-outcome verifier $\{0.0, 0.1, 1.0\}$ for {incoherent, coherent, correct} (R=3). Length is discretized into short, Countdown-appropriate bins (Appendix B.5); the upstream 32k-token math bins would place nearly all mass in a single bin here.

Why "Lite". The full ZIP-RC objective trains the head jointly with a KL penalty that keeps the policy close to its original. ZIP-RC-Lite instead *freezes the entire backbone* and trains only the output head, dropping the KL term: a frozen policy cannot drift, so next-token behavior is preserved by construction. A useful side effect is that the plain policy checkpoint (used to generate rollouts) and the head checkpoint share identical hidden states, so offline scoring requires no logit masking. Masking is only needed for the live decoder, which must never emit a reserved token.

Adaptive parallel decoding. The live decoder runs a K-parallel decode loop with a shared KV cache. At every step it reads the joint live, masks the reserved slice to $-\infty$ before sampling, and lets a meta-policy reallocate compute across the K trajectories. We implement three meta-policies plus the paper's utility policy:

- **Best-of-N (none).** Carry all K samples to completion (the fixed-budget baseline).
- **Prune (compute axis).** After a warmup, periodically abandon the lowest-value active samples (keeping at least $k_{\min}$), pruning only predicted losers ($\hat{V} < 0.5$). Cost is measured as total *active* forward passes, so abandoned samples stop accruing compute.
- **Early-stop (latency axis).** Once the best active sample's smoothed value crosses a threshold τ, commit to it and abandon the rest, finishing as soon as that trajectory completes rather than waiting for stragglers.
- **Utility (ZIP-RC sampling).** Keep sample j only if its marginal contribution to the expected best-of-set reward, $v_j \prod_{i \neq j}(1 - v_i)$, exceeds β times its normalized expected remaining cost $\mathbb{E}[\ell_j]/\ell_{\max}$. This is the sampling utility

  $$U(S) = \mathbb{E}\Big[\max_{i \in S} r_i \Big] \; - \; \beta \sum_{i \in S} \mathbb{E}[\ell_i] \; - \; \gamma \max_{i \in S} \mathbb{E}[\ell_i],$$

  where β charges total compute and γ worst-case latency; sweeping β (and τ) traces the Pareto frontier.

Among finished, non-pruned samples we select the one with the highest `value_end` (mean $\hat{V}$ over the last 16 response positions). Cost is the sum of active forward passes (KV-cache accounted) and latency is the number of decode steps (a wall-clock proxy: under data-parallel sampling, latency is governed by the longest active trajectory).

Adaptive-K allocation. We treat adaptive-K as an auxiliary diagnostic rather than a main result. The head's value *at the prompt itself*, $\hat{V}(x)$ before any sampling, is a per-prompt difficulty estimate. Adaptive-K uses it to spend a fixed *average* sample budget unevenly, assigning more samples to predicted-hard prompts and fewer samples to predicted-easy prompts. This captures the idea of reallocating compute across prompts without modifying the KV cache and can be evaluated fully offline. We allocate by *percentile rank* of $\hat{V}(x)$ rather than by its raw value, since the raw dynamic range is compressed on Countdown. As discussed in Appendix A.3, standalone adaptive-K has little room to help on Countdown because the prompt difficulty range is narrow. We therefore use adaptive-K mainly as a diagnostic and as part of the blended controller with pruning, whose held-out compute-efficiency results are shown in Figure 6.

4 Experimental Setup

4.1 Frontier-Curriculum RLOO

We evaluate Frontier-Curriculum RLOO on the Countdown arithmetic reasoning task using Qwen2.5-0.5B [24; 15]. The training and evaluation prompts are drawn from the verifier-based Countdown dataset `asingh15/countdown_tasks_3to4` [2]. The verifier assigns reward 1.0 to a fully correct expression, reward 0.1 to a formatted but incorrect expression, and reward 0.0 otherwise.

We compare Frontier-Curriculum RLOO against SFT, IPO, and vanilla RLOO. SFT provides the warm-start model used for online reinforcement learning. IPO serves as a pairwise preference-optimization baseline [3]. Vanilla RLOO is identical to our method except that it samples prompts uniformly [1]: both share the same SFT initialization, verifier reward, optimizer settings, and RLOO update, so the only intended experimental change is the prompt sampling distribution, modified through frontier replay. We also evaluate several ablations, including failure-based retrieval, different replay ratios, Goldilocks frontier selection, taxonomy weighting, and Teacher-SFT.

For the main RLOO runs, we train for 80 steps with batch size 64. The frontier runs use rollout group size $K = 8$, learning rate 1×10^{-5}, KL coefficient 0.001, entropy coefficient 0.001, and weight decay 1×10^{-4}. Training-time sampling uses temperature 1.0, top-p value 1.0, top-k value -1, and min-p value 0.0. The main frontier replay ratio is $\rho = 0.10$.

For evaluation, we generate 128 completions per evaluation prompt. We report average verifier score, which includes partial credit, along with pass@1, pass@16, and pass@128, where pass@k measures whether at least one of the first k sampled completions receives full verifier reward. Pass@1 measures single-sample performance, while pass@16 and pass@128 measure how much performance improves when more samples are available. This separates low-sample accuracy from the high-sample search ceiling. Unless otherwise noted, the Frontier-Curriculum RLOO results are single-run evaluations, while the ZIP-RC-Lite results below report multi-seed results on a fixed 50-prompt held-out pool.

4.2 ZIP-RC-Lite Setup and Hyperparameters

The ZIP-RC-Lite experiments use the same Countdown task and base model. We train ZIP-RC-Lite heads on rollouts from the frozen, post-trained Qwen2.5-0.5B policy. The pass@1 values in this subsection are measured on the ZIP-RC-Lite held-out rollout pool and use a separate stronger RLOO checkpoint, so they are not directly comparable to the vanilla RLOO pass@1 in Table 1. The headline adaptive-compute and calibration numbers use the stronger **RLOO** policy (pass@1 ≈ 0.66); the structured-verifier and cross-policy-transfer studies additionally use the weaker **SFT** policy (pass@1 ≈ 0.20). The pipeline has six isolated stages on Modal: rollout generation (vLLM) $\rightarrow$ verifier labeling $\rightarrow$ head-only training (frozen backbone) $\rightarrow$ offline teacher-forced scoring $\rightarrow$ a value-selection viability gate $\rightarrow$ the live adaptive decoder. The held-out set is the test split's full 50 prompts; multi-seed results use 3 seeds. Every hyperparameter that was actually run is listed in Appendix Table 7.

5 Results

5.1 Frontier-Curriculum RLOO

Frontier-Curriculum RLOO changes only the prompt sampling distribution while keeping the RLOO update fixed. We first report quantitative performance on the main evaluation and replay-ratio ablation, then analyze how the method changes the composition of remaining failures.

5.1.1 Quantitative Evaluation

Table 1 reports the main 128-sample evaluation for SFT, IPO, vanilla RLOO, and Frontier-Curriculum RLOO. Relative to vanilla RLOO, Frontier-Curriculum RLOO increases average verifier score from 0.46 to 0.52 and pass@1 from 0.41 to 0.49. Because pass@128 remains around 0.82, the result is best interpreted as improved low-sample efficiency rather than an increase in the model's maximum achievable pass rate under large-sample evaluation.

Table 1: Performance comparison on Countdown with 128 sampled completions per prompt.

Method	Avg. Score	pass@1	pass@16	pass@128
SFT	0.26	0.18	0.74	0.86
IPO	0.33	0.26	0.75	0.80
Vanilla RLOO	0.46	0.41	0.75	0.82
Frontier-Curriculum RLOO	**0.52**	**0.49**	**0.77**	0.82

Table 2: Frontier replay-ratio ablation.

Method	Avg.	pass@1	pass@16	pass@128
Vanilla RLOO	0.46	0.41	0.75	0.82
Frontier $\rho = 0.05$	0.45	0.39	0.72	0.78
Frontier $\rho = 0.10$	**0.52**	**0.49**	**0.77**	0.82
Frontier $\rho = 0.12$	0.52	0.48	0.76	0.82
Frontier $\rho = 0.15$	0.51	0.48	0.74	0.82

Figure 2 shows the pass@k curves up to $k = 16$ for the same four methods. Frontier-Curriculum RLOO is strongest when only a few samples are available and remains competitive as k increases. This pattern is consistent with Table 1, where the method has higher pass@1 and pass@16 while leaving pass@128 roughly unchanged.

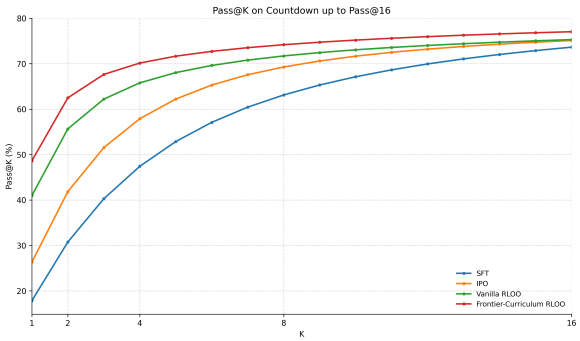

Figure 2: Pass@k on Countdown for SFT, IPO, vanilla RLOO, and Frontier-Curriculum RLOO up to $k = 16$.

The replay-ratio ablation in Table 2 shows that a small amount of frontier replay is most effective. The setting $\rho = 0.10$ gives the best average verifier score, pass@1, and pass@16. A weaker replay ratio, $\rho = 0.05$, does not provide enough frontier exposure. Larger replay ratios, $\rho = 0.12$ and $\rho = 0.15$, do not further improve pass@16 or pass@128. This suggests that frontier prompts provide useful learning signal, but replacing too much of the uniform batch can reduce dataset diversity. The full replay-ratio pass@k curves are provided in Appendix A.1.

5.1.2 Qualitative Analysis

We also analyze failure modes for SFT, IPO, vanilla RLOO, and Frontier-Curriculum RLOO. Figure 3 reports failure categories as a percentage of failed rollouts, while Figure 4 reports the total number of failed rollouts for each method. These two views answer different questions. The normalized distribution shows which error types remain after a method fails. The total-failure plot shows how often failures occur overall.

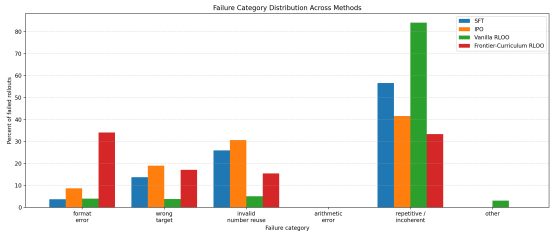

Figure 3: Failure-category distribution across methods, normalized by failed rollouts.

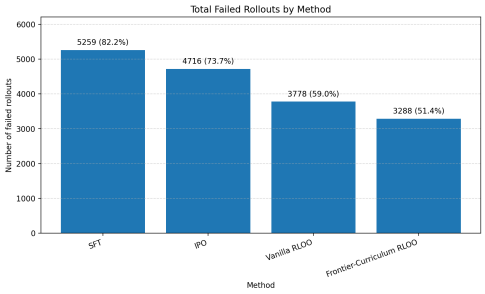

Figure 4: Total failed rollouts by method.

Across the same 6400 evaluated rollouts per method, the failed-sample rate decreases across the main methods. SFT has 5259 failed rollouts, IPO has 4716, vanilla RLOO has 3778, and Frontier-Curriculum RLOO has 3288. Thus, Frontier-Curriculum RLOO reduces the failed-sample rate from 59.03% for vanilla RLOO to 51.38%. This confirms that the result is not only a shift in error types, but also a reduction in the total number of failed generations.

The largest qualitative improvement is the reduction in repetitive or incoherent generations. Vanilla RLOO failures are dominated by long, unstable completions that repeat answer tags, continue reasoning after giving an answer, or contradict their own intermediate steps. Frontier-Curriculum RLOO makes this failure mode much less frequent.

The reduction is not uniform across all categories. Under Frontier-Curriculum RLOO, format errors, wrong-target errors, and invalid-number-reuse errors become a larger share of the remaining failures. Some of these categories also increase in absolute count relative to vanilla RLOO. This should not be read as a contradiction of the main result, since the total number of failed rollouts still decreases. Instead, it suggests that frontier replay removes many degenerate completions, leaving a smaller but more structured set of failures. These residual failures often reflect strict answer formatting, arithmetic accuracy, or number-use constraints rather than complete generation instability.

Overall, frontier replay improves low-sample correctness and reduces the total number of failures, especially repetitive or incoherent outputs. The remaining errors are more interpretable and more localized, which points to possible follow-up improvements such as stronger answer-format supervision, constrained decoding over allowed numbers, or additional training on structured arithmetic mistakes.

The failure-category definitions used in Figures 3 and 4 are described in Appendix B.4.

5.2 ZIP-RC-Lite Adaptive Test-Time Compute

ZIP-RC-Lite uses the frozen policy's success signal to allocate inference compute more selectively. We first report calibration, live-decoder operating points, selection behavior, and cross-policy transfer. We then discuss what these results imply about pruning, early stopping, answer selection, and the limits of the signal.

5.2.1 Quantitative Evaluation

The ZIP-RC-Lite head drives the live decoder, and which lever wins depends on whether the objective is compute or latency. Table 3 reports calibration for the binary head, and Table 4 reports adaptive-decoding results with 3 seeds, 50 held-out prompts, $K = 8$, and the scaled head. The calibration results show that the reward estimate is close to the paper baseline. The live-decoder results reproduce the compute and latency tradeoff on Countdown. Pruning cuts roughly 63% of active forward passes for a small accuracy tradeoff, while early stopping at $\tau = 0.8$ cuts roughly 48% of latency with no clear accuracy loss within the reported variance.

Table 3: K=64 ground-truth calibration (binary head, n=32 prompts with $\geq$ 64 rollouts), compared to the paper's reported ZIP-RC-Lite. Lower TV is better; higher F1/accuracy is better.

Metric	Ours (ZIP-RC-Lite)	Paper ZIP-RC-Lite
Mean Total Variation (↓)	**0.52**	0.63
End-of-gen reward F1 (↑)	0.81	**0.82**
End-of-gen reward accuracy (↑)	0.67	**0.71**

Table 4: Adaptive-compute operating points vs. full Best-of-N (3 seeds, mean $\pm$ std). Cost is active forward passes; latency is decode steps. Percentages are relative to "none".

Policy	Accuracy	Compute (fwd passes)	Latency (decode steps)
None (full BoN)	0.708 ± 0.012	3944 ± 106	692 ± 11
Prune	0.683 ± 0.012	$\mathbf{1468 \pm 16}$ (-63%)	508 ± 16 (-27%)
Early-stop τ=0.8	$\mathbf{0.725 \pm 0.020}$	2306 ± 147 (-42%)	$\mathbf{356 \pm 12}$ (-48%)
Utility β=0.02	0.650 ± 0.020	1192 ± 36 (-70%)	443 ± 17 (-36%)

Figure 5 shows the same live-decoder runs as paired compute–accuracy and latency–accuracy Pareto plots. On the compute axis, pruning and the utility policy move down and left from full Best-of-N, trading a small accuracy change for a large compute reduction; on this task, the simple absolute-threshold prune is stronger than the redundancy-aware utility policy. On the latency axis, early stopping at $\tau = 0.8$ commits to a confident trajectory, cutting roughly 48% of decode steps while keeping accuracy within the reported variance of full Best-of-N. The compute-axis panel alone, with its single-lever operating points, is reproduced in Appendix B.6.

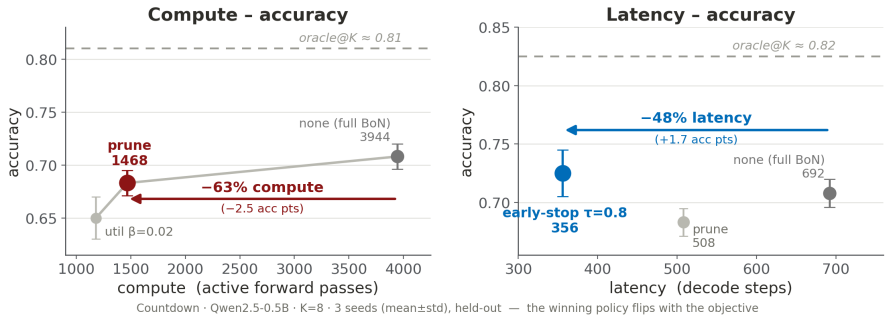

Figure 5: Compute–accuracy (left) and latency–accuracy (right) Pareto for the live decoder (Countdown, Qwen2.5-0.5B, K=8, 3 seeds, mean $\pm$ std, held-out; same runs as Table 4). The winning policy flips with the objective: pruning is best on the compute axis (-63% compute for -2.5 accuracy points), while early stopping at τ=0.8 is best on the latency axis (-48% latency, with accuracy within the reported variance of the full Best-of-N baseline).

We also include two diagnostic tables that clarify how the success signal should be used. Table 5 shows that value-based answer selection does not beat majority voting on Countdown. Table 6 shows that the head transfers across SFT and RLOO rollouts with little loss, especially from the stronger RLOO policy to the weaker SFT policy.

Table 5: Offline selection at K=8 on the held-out set. `random` is the chance floor (mean correctness); `oracle` is the ceiling (any-correct).

Method	Accuracy @ K=8
random@8 (chance floor)	0.70
value@8 (ZIP-RC selection)	0.74
majority@8 (self-consistency)	0.78
oracle@8 (ceiling)	0.78

Table 6: Cross-policy transfer, AUC(value $\rightarrow$ correct), same base (Qwen2.5-0.5B). Diagonal is matched (head and rollouts from the same policy); off-diagonal is transfer.

Head trained on ↓ / scored on →	RLOO rollouts	SFT rollouts
RLOO head	**0.922** (matched)	0.865 (transfer)
SFT head	0.890 (transfer)	**0.867** (matched)

The cross-policy result is useful for the broader system design. It suggests that the introspection head does not need to be retrained after every policy update. A single frozen head may be able to support both the curriculum signal and the live decoder across nearby checkpoints, which reduces the cost of maintaining the adaptive-compute component.

Finally, as an auxiliary adaptive-K diagnostic, blending adaptive-K with pruning gives an additional compute-efficiency result. Figure 6 shows that the blended controller achieves a held-out 20% to 25%

compute saving at neutral accuracy across evaluation pools. This result is smaller than the headline single-lever pruning gain, but it is more conservative because it uses held-out operating points and broader difficulty pools.

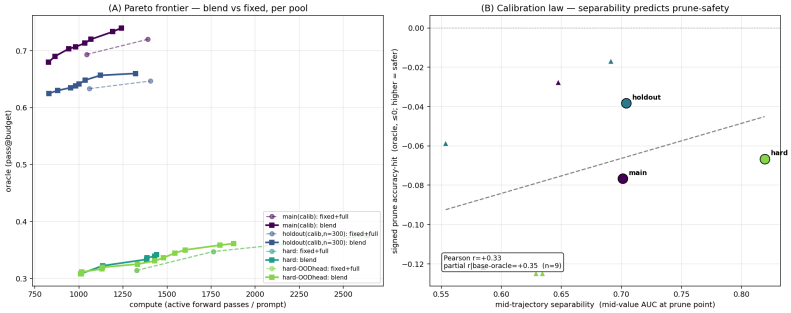

Figure 6: Blending adaptive-K with pruning. **(A)** The blend frontier (squares) sits left of fixed-allocation Best-of-N (circles) on every pool, a held-out $\sim$20–25% compute saving at neutral accuracy. **(B)** Prune-safety (signed accuracy hit; higher is safer) rises with mid-trajectory separability (value AUC at the prune point), but only weakly across pools (difficulty-controlled partial r=0.35, n=9).

5.2.2 Qualitative Analysis

The adaptive-compute results are best interpreted as compute allocation rather than answer selection. Value-based selection reaches 0.74 accuracy at $K = 8$, while majority voting reaches 0.78, matching the oracle on this held-out set. This means ZIP-RC-Lite is not primarily improving which completed answer is selected. Instead, it improves how compute is spent before all samples finish.

The qualitative behavior of the decoder explains why different policies help different objectives. Pruning is useful when the goal is total compute, because abandoned trajectories stop accumulating active forward passes. Early stopping is useful when the goal is latency, because the decoder can commit to a confident trajectory instead of waiting for all parallel samples to finish. On Countdown, early stopping may avoid some failures because long, looping continuations are often wrong, so committing to a stable high-value trajectory avoids some degenerate completions.

The results also clarify the limits of the signal. The head reads whether a trajectory is likely to succeed better than it predicts exact remaining length. As a result, reward-based pruning and early stopping are reliable, while more ambitious allocation policies depend on how separable successful and failed trajectories are before generation finishes. The blend study supports this interpretation. When mid-trajectory separability is high, pruning is safer. When successful and failed trajectories remain hard to distinguish until late in the response, the compute-saving ceiling is set more by the problem than by the head.

Additional experiments and ablations. Appendix A groups the auxiliary studies: frontier replay-ratio curves, failure-based retrieval, Goldilocks frontier selection, taxonomy-weighted replay, Teacher-SFT, the structured ZIP-RC-Lite verifier, standalone adaptive-K, single-lever Pareto analysis, and the held-out adaptive-K+pruning blend protocol. These results are diagnostic or negative results rather than the main claims.

6 Discussion

The Frontier-Curriculum result is consistent with the structure of the RLOO estimator. Because RLOO forms advantages within rollout groups for the same prompt, mixed-outcome prompts provide direct reward contrast: successful completions receive positive relative advantages, while failed completions receive negative relative advantages. Prompts for which all sampled completions are correct or all sampled completions fail provide little contrast and therefore weaker learning signal.

This perspective also explains why failure-only retrieval is less effective. Although replaying total failures may seem useful, these prompts are often outside the current policy's reachable region. When every sampled completion fails, the verifier provides little information about which completion should

be preferred. Frontier prompts are more useful because the same policy sometimes succeeds and sometimes fails on them.

The main limitation is that Frontier-Curriculum RLOO uses a single training seed, which is common in LLM fine-tuning because repeated online RL runs are expensive. To make the evaluation more stable, we report a large pass@k evaluation with 128 completions per prompt, including pass@1 and pass@128; however, multi-seed training remains important future work. The gains are concentrated at low k, while pass@128 is nearly unchanged. This suggests that the method improves low-sample efficiency rather than performance when many samples are available. Other limitations are that the retrieval features are task-specific, the ZIP-RC-Lite held-out pools are small, and Countdown's narrow difficulty range limits cross-prompt adaptive allocation. Future work should test multi-seed robustness, adaptive replay schedules, broader tasks, and learned success predictors that estimate frontier status without requiring K rollouts.

The ZIP-RC-Lite results support the same success-signal perspective at inference time. Its main benefit is adaptive compute allocation rather than answer selection. Value-based selection matches but does not exceed majority voting on Countdown, yet the same zero-overhead signal reduces compute by approximately 63% through pruning or latency by approximately 48% through early stopping at near-baseline accuracy. The head predicts trajectory success reliably, with mid-generation and end-of-generation AUC around 0.90 to 0.92, but its remaining-length estimates are weaker. Cross-policy transfer with off-diagonal AUC at least 0.865 suggests that the success signal is partly policy-agnostic within the same base model. This suggests, but does not yet establish, a possible unified controller in which a frozen introspection head supports both curriculum selection and inference-time allocation.

7 Conclusion

We introduced Frontier-Curriculum RLOO, a prompt-sampling curriculum for verifier-based online RL. The method replays prompts where the current policy has mixed success and failure, giving RLOO stronger within-prompt reward contrast while leaving the reward function and policy-gradient objective unchanged. In a single-run 128-sample Countdown evaluation, Frontier-Curriculum RLOO improves average verifier score from 0.46 to 0.52 and pass@1 from 0.41 to 0.49 over vanilla RLOO, while pass@128 remains at 0.82. Thus, the result is best interpreted as improved low-sample efficiency: correct solutions appear earlier in the sampling process, but the large-sample pass rate does not increase.

ZIP-RC-Lite shows that a frozen Qwen2.5-0.5B carries a useful zero-overhead success signal in its unused vocabulary logits. Under the comparison protocol used here, the head is well calibrated on held-out trajectories, with AUC 0.92 and lower total variation than the reported ZIP-RC-Lite baseline. A live decoder using this signal can reduce compute by approximately 63% through pruning or latency by approximately 48% through early stopping at near-baseline accuracy. These results support the interpretation of ZIP-RC-Lite as adaptive compute allocation rather than improved answer selection.

The compute-saving numbers describe different operating regimes: the 63% compute reduction and 48% latency reduction are single-lever in-distribution operating points, while the 20% to 25% blend result is a more conservative held-out estimate across broader difficulty pools. Cross-policy transfer with off-diagonal AUC at least 0.865 further suggests that the success signal can transfer across nearby policies derived from the same base model. This points toward, but does not yet establish, a unified controller in which a frozen introspection head could replace rollout-based frontier counting during fine-tuning and allocate compute during inference.

Team Contributions

- **Marco Vizcarra:** Implemented the SFT and IPO baselines, led the failure-aware/frontier curriculum experiments, implemented the main curriculum ablations, analyzed failure categories, and collaborated to the poster and final report.
- **Andy Kim:** Implemented the RLOO, led the ZIP-RC-Lite adaptive-compute extension, ran multi-seed evaluations, and collaborated to the poster and final report.

Changes from Proposal. The original proposal focused on failure-aware retrieval, taxonomy-based prompt selection, and a structured ZIP-RC-Lite verifier. Early training experiments showed that retrieving total failures was not sufficient, because prompts for which all sampled completions fail often provide weak RLOO advantage contrast. We therefore shifted the main RL extension to frontier replay, where prompts contain both successful and failed completions under the current policy. Taxonomy weighting, Goldilocks frontier selection, and Teacher-SFT were retained as ablations rather than main methods.

The ZIP-RC-Lite extension also changed after empirical validation. The planned structured three-outcome verifier did not pass the required viability gate because failure-mode diversity depended strongly on policy capability. The converged RLOO policy failed mostly coherently, while the weaker SFT policy failed mostly incoherently. We therefore used the binary verifier as the main setting and report the structured verifier as a negative result. The final head is also smaller than originally proposed, using a 14–21 logit reserved slice starting at id $151{,}665$ with seven Countdown-specific length bins.

Finally, adaptive-K allocation had little room to help on Countdown because prompt difficulty was too narrow. The main inference-time gains instead came from pruning and early stopping, and we further evaluated a held-out blend of adaptive-K with pruning. Beyond the proposal, we added the untie-and-clone fix needed for Qwen2.5-0.5B's tied embeddings, a cross-policy transfer study, and a more conservative statistical protocol with held-out operating-point selection and prompt-level bootstrap analysis.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. https://aclanthology.org/2024.acl-long.662/

[2] asingh15. [n. d.]. countdown_tasks_3to4. https://huggingface.co/datasets/asingh15/countdown_tasks_3to4. Hugging Face dataset, accessed 2026-06-06.

[3] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Rémi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A General Theoretical Paradigm to Understand Learning from Human Preferences. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. https://arxiv.org/abs/2310.12036

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 41–48. doi:10.1145/1553374.1553380

[5] Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025. Self-Evolving Curriculum for LLM Reasoning. *arXiv preprint arXiv:2505.14970* (2025). https://arxiv.org/abs/2505.14970

[6] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021). https://arxiv.org/abs/2110.14168

[7] Thomas Foster and Jakob Foerster. 2025. Learning to Reason at the Frontier of Learnability. *arXiv preprint arXiv:2502.12272* (2025). https://arxiv.org/abs/2502.12272

[8] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. 2017. Automated Curriculum Learning for Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 1311–1320. https://proceedings.mlr.press/v70/graves17a.html

[9] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. 2021. Prioritized Level Replay. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 4940–4950. https://proceedings.mlr.press/v139/jiang21b.html

[10] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Buy 4 REINFORCE Samples, Get a Baseline for Free. In *Deep Reinforcement Learning Meets Structured Prediction Workshop at ICLR*. https://openreview.net/forum?id=r1lgTGL5DE

[11] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems*, Vol. 23. https://papers.nips.cc/paper_files/paper/2010/hash/e57c6b956a6521b28495f2886ca0977a-Abstract.html

[12] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050* (2023). https://arxiv.org/abs/2305.20050

[13] Ilia Mahrooghi, Aryo Lotfi, and Emmanuel Abbe. 2026. Goldilocks RL: Tuning Task Difficulty to Escape Sparse Rewards for Reasoning. *arXiv preprint arXiv:2602.14868* (2026). https://arxiv.org/abs/2602.14868

[14] Rohin Manvi, Joey Hong, Tim Seyde, Maxime Labonne, Mathias Lechner, and Sergey Levine. 2025. Zero-Overhead Introspection for Adaptive Test-Time Compute. *arXiv preprint arXiv:2512.01457* (2025). https://arxiv.org/abs/2512.01457

[15] Jiayi Pan, Jun Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. TinyZero. https://github.com/Jiayi-Pan/TinyZero. GitHub repository.

[16] Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2025. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. *arXiv preprint arXiv:2506.06632* (2025). https://arxiv.org/abs/2506.06632

[17] Yun Qu, Qi Wang, Yixiu Mao, Vincent Tao Hu, Björn Ommer, and Xiangyang Ji. 2025. Can Prompt Difficulty be Online Predicted for Accelerating RL Finetuning of Reasoning Models? *arXiv preprint arXiv:2507.04632* (2025). https://arxiv.org/abs/2507.04632

[18] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems*, Vol. 36. https://arxiv.org/abs/2305.18290

[19] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952* (2015). https://arxiv.org/abs/1511.05952

[20] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314* (2024). https://arxiv.org/abs/2408.03314

[21] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171* (2022). https://arxiv.org/abs/2203.11171

[22] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8 (1992), 229–256. doi:10.1007/BF00992696

[23] Jiahao Wu, Ning Lu, Shengcai Liu, Kun Wang, Yanting Yang, Li Qing, and Ke Tang. 2026. Train at Moving Edge: Online-Verified Prompt Selection for Efficient RL Training of Large Reasoning Model. *arXiv preprint arXiv:2603.25184* (2026). https://arxiv.org/abs/2603.25184

[24] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024). https://arxiv.org/abs/2412.15115

A Additional Experiments

We tested several alternatives to plain frontier replay. These experiments are useful for understanding why the main method works, but we treat them as ablations rather than primary results.

A.1 Frontier Replay-Ratio Curves

Figure 7 shows the full pass@k curves for the frontier replay-ratio ablation. The table in the main Results section summarizes the key metrics, while this appendix figure shows how the replay ratios compare across the low-sample regime.

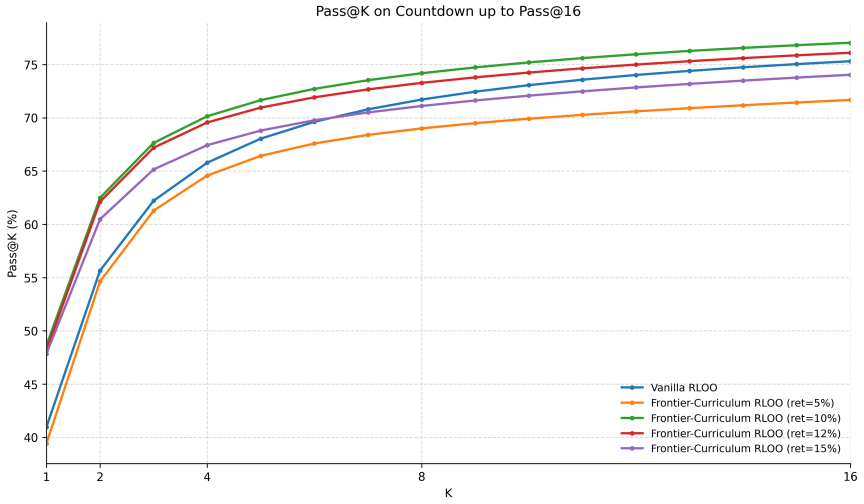

Figure 7: Pass@k replay-ratio ablation for Frontier-Curriculum RLOO up to $k = 16$.

A.2 Other Frontier Variants

We also tested several variants of frontier replay to understand whether the gains came from adaptive replay in general or from the specific mixed-outcome frontier signal. These variants were useful as ablations, but none provided a cleaner improvement than plain Frontier-Curriculum RLOO.

Failure-based retrieval. Failure-based retrieval used the same structured prompt embeddings and retrieval pipeline as Frontier-Curriculum RLOO, but changed the memory criterion. Instead of storing prompts with mixed outcomes, it stored prompts for which all sampled completions failed. The goal was to test whether replaying hard prompts alone was sufficient. This did not consistently improve over vanilla RLOO. Retrieval from total failures with $\rho = 0.10$ reached average score 0.4692 and pass@1 0.4228, while $\rho = 0.30$ dropped to average score 0.4345 and pass@1 0.3753. These results support the motivation for frontier selection. Total failures are often difficult, but they do not necessarily provide useful RLOO signal because the sampled completions can all receive similar rewards. In that case, the leave-one-out baseline has little contrast to exploit. Frontier prompts are more informative because they contain both successful and unsuccessful completions within the same rollout group.

Goldilocks frontier selection. Goldilocks frontier selection used a stricter version of the frontier criterion. Instead of storing every prompt satisfying $0 < s(x) < K$, it emphasized prompts whose empirical success rate $p(x) = s(x)/K$ was closer to the middle of the range. The intuition was that prompts near $p(x) = 0.5$ should provide the strongest within-group contrast, since the rollout group contains a balanced mixture of successful and failed completions. This helped pass@1 in some settings, but it was not clearly better overall. The stricter filter also reduced the number of prompts eligible for memory, especially early in training. The result suggests that intermediate-difficulty prompts are useful, but the simpler frontier condition $0 < s(x) < K$ was sufficient and more robust for the main experiment.

Taxonomy-weighted replay. Taxonomy-weighted replay added hand-designed weights for failure categories such as wrong target, invalid number reuse, and format errors. The motivation was to prioritize failure modes that seemed more actionable than generic incorrect answers. This made the curriculum more interpretable, since retrieved prompts could be associated with specific error types. However, it did not outperform the simpler frontier signal. A likely reason is that fixed taxonomy weights are not directly aligned with the RLOO advantage structure. The frontier criterion is determined by the model's current rollout outcomes and therefore tracks where the policy already has partial ability. In contrast, a failure category can be informative for analysis without necessarily producing useful leave-one-out reward contrast.

Teacher-SFT. Teacher-SFT used exact-solver demonstrations for selected frontier prompts. This gave the model a stronger supervised signal on prompts near its current learning boundary. However, this variant changes the training setup more substantially than frontier replay because it introduces new supervised targets rather than only changing the prompt sampling distribution. It is therefore less controlled as a comparison to vanilla RLOO. We treat Teacher-SFT as an ablation rather than the main method, since the main contribution is to improve RLOO by modifying prompt selection while keeping the reward function, optimizer, initialization, and RLOO update fixed.

A.3 Additional ZIP-RC-Lite Experiments

This appendix collects ZIP-RC-Lite experiments that did not produce a clear positive result in this setting. These experiments help scope the limits of the success signal on Countdown.

Structured three-outcome verifier. The structured {incoherent, coherent, correct} verifier produced a negative result with a useful finding behind it. Failure-mode diversity is policy-capability-dependent. The converged RLOO policy fails almost only *coherently*, with only $\sim 3\%$ incoherent failures. The weaker SFT policy fails almost only *incoherently*, with only $\sim 10\%$ coherent failures. No single 0.5B checkpoint populates all three classes above the 15% gate we required. Trained on SFT rollouts, where all three classes are present, the structured head collapsed to the binary head, with AUC 0.867 versus 0.865. The structured signal may be more useful for curriculum analysis than for inference on this setting.

Adaptive-K allocation. Adaptive-K spends a fixed average sample budget unevenly, drawing more samples for prompts the head predicts are hard and fewer for easy ones. On Countdown this neither helps nor hurts. A small gain on the 256-prompt head, with mean-K 2–6 and oracle changes of $+0.006$ to $+0.019$, did not replicate on the scaled head, where the sign was mixed within ± 0.007. The reason is a property of the benchmark, not of the mechanism. Countdown's 3–4-number prompts are nearly uniform in difficulty. Even ranked by percentile, the head's prompt-level value $\hat{V}(x)$ is compressed to roughly $[0.52, 0.54]$, so there is almost nothing to reallocate. Allocation can only pay off when prompts differ enough in difficulty for the saved samples to matter elsewhere, and Countdown does not provide that spread.

B Implementation Details

B.1 Frontier Score and Advantage Contrast

The frontier score is a curriculum priority, not part of the RLOO objective. It is meant to capture how much useful contrast a prompt produces under the current policy. The idea can be seen directly from the leave-one-out advantages in the binary correctness case. This analysis is only used to motivate the curriculum priority. The training update still uses the verifier rewards produced by the task checker.

Suppose a prompt has $s = s(x)$ correct completions among K samples. A correct completion has advantage

$$A_i^+ = 1 - \frac{s-1}{K-1} = \frac{K-s}{K-1},$$

because the leave-one-out baseline averages the other $s-1$ correct completions and $K-s$ incorrect completions. An incorrect completion has advantage

$$A_i^- = 0 - \frac{s}{K-1} = -\frac{s}{K-1}.$$

If $s = 0$, all completions fail and every advantage is zero. If $s = K$, all completions are correct and every advantage is also zero. Only the mixed case $0 < s < K$ creates both positive and negative advantages.

A compact way to summarize the amount of contrast in the group is the average squared advantage

$$\frac{1}{K}\sum_{i=1}^{K} A_i^2 = \frac{1}{K}\left[s\left(\frac{K-s}{K-1}\right)^2 + (K-s)\left(\frac{s}{K-1}\right)^2\right] = \frac{s(K-s)}{(K-1)^2}.$$

We use this expression only for analysis. It is not optimized directly. Its shape is proportional to $p(x)(1-p(x))$, where $p(x) = s(x)/K$. This is why the implementation uses

$$f(x) = 4p(x)(1-p(x)).$$

The factor 4 normalizes the maximum value to 1. This score is highest when the model is around half correct on a prompt and lowest when the prompt is always failed or always solved. For finite K, the largest attainable value is exactly 1 when K permits $p(x) = \frac{1}{2}$, and is otherwise the closest attainable value to this maximum. In our implementation, $s(x)$ counts only fully correct completions with reward 1.0. Partial-credit completions, such as formatted but incorrect answers with reward 0.1, still contribute to the RLOO baseline and advantage, but they do not count as successful completions for the frontier score. Thus, the frontier score identifies prompts with mixed binary success outcomes, while the policy update itself continues to use the raw verifier reward.

B.2 Prompt Embeddings

We use a task-structured embedding rather than a sentence retriever because Countdown prompts are mostly the same natural-language template. The important variation is the arithmetic structure of the instance, not the surface wording. In `asingh15/countdown_tasks_3to4`, the prompt text repeatedly follows the same instruction pattern and mainly changes the target value and the allowed numbers [2]. This makes sentence-level semantic retrieval less useful for our setting, since two prompts can look almost identical in text while requiring very different arithmetic behavior. The embedding is therefore designed to represent the numerical structure of the Countdown instance directly. It captures the target scale, the number of available operands, summary statistics of the allowed numbers, the gap between the available-number sum and the target, parity information, and the sorted list of allowed numbers.

For a prompt with target t, allowed numbers $a_1, \ldots, a_n$, sorted and padded values $\tilde{a}_1, \ldots, \tilde{a}_4$, sum S, mean μ, standard deviation σ, and average parity $\bar{p}$, the embedding starts from the 14-dimensional feature vector

$$\phi(x) = \left[\frac{t}{100}, \frac{n}{4}, \frac{\min_\ell a_\ell}{100}, \frac{\max_\ell a_\ell}{100}, \frac{\mu}{100}, \frac{\sigma}{100}, \frac{S}{300}, \frac{S-t}{300}, \frac{|S-t|}{300}, \bar{p}, \frac{\tilde{a}_1}{100}, \frac{\tilde{a}_2}{100}, \frac{\tilde{a}_3}{100}, \frac{\tilde{a}_4}{100}\right].$$

Here, $S = \sum_{\ell=1}^{n} a_\ell$, and μ and σ describe the scale and spread of the allowed numbers. The parity feature is

$$\bar{p} = \frac{1}{n}\sum_{\ell=1}^{n} \mathbb{1}\{a_\ell \equiv 1 \pmod 2\}.$$

The sorted padded values preserve instance-specific information that is not fully captured by the aggregate statistics. The division constants are scale normalizers rather than learned parameters. We divide n by 4 because the dataset contains prompts with at most four allowed numbers, so this feature lies on a comparable scale to the other normalized quantities. We divide the target, individual allowed numbers, and their summary statistics by 100 because these values are on the order of 10^2 in Countdown. We divide the sum-based features by 300 because sums and target gaps can be several times larger than individual numbers. These constants keep all coordinates in a similar numerical range before ℓ_2 normalization, so retrieval is not dominated by whichever feature has the largest raw magnitude. The exact constants are not intended to encode additional task knowledge beyond this scale matching.

We then normalize the vector as

$$e(x) = \frac{\phi(x)}{\|\phi(x)\|_2}.$$

After normalization, dot product is equivalent to cosine similarity. This makes retrieval depend on the arithmetic shape of the prompt instead of the repeated sentence template. This embedding is intentionally simple and fixed. It does not require training a separate retriever, and it keeps retrieval tied to the task structure used by the verifier.

B.3 Retrieval Mechanism

The retrieval mechanism uses a prompt bank built from the training set. Let $B = \{(x_m, e_m, k_m)\}_{m=1}^{N}$ be the prompt bank, where k_m is a canonical Countdown key, and let $M_t = \{(z_j, e_j, u_j, k_j)\}_{j=1}^{L_t}$ be the current frontier memory, where u_j is the memory priority. The key is

$$k(x) = (t, \mathrm{sort}(a_1, \ldots, a_n)),$$

where t is the target and $a_1, \ldots, a_n$ are the allowed numbers. This key treats two prompts with the same target and same allowed numbers as the same arithmetic instance, even if their text formatting differs. The embedding is still used for similarity, while the key is used only for duplicate filtering.

In the plain frontier setting, the memory priority is the frontier score $f(z_j) = 4p(z_j)(1 - p(z_j))$. We compute the similarity matrix

$$S_{mj} = e_m^\top e_j.$$

Each prompt-bank item receives its best priority-weighted frontier match

$$c_m = \max_{1 \le j \le L_t} S_{mj} u_j.$$

This score is large when a candidate prompt is arithmetically similar to at least one high-priority frontier prompt. The maximum over memory entries allows each candidate to match the most relevant stored frontier prompt instead of being diluted by unrelated memory items. The priority factor u_j biases retrieval toward prompts that previously produced strong within-group contrast.

Before sampling retrieved prompts, we remove candidates whose key appears in the frontier memory or in the current uniform batch. We also keep a local set of retrieved keys so that the same arithmetic instance is not selected twice in one mixed batch. After this filtering step, we select a top candidate pool using the largest remaining c_m values and sample the required number of retrieved prompts from that pool. Thus, embeddings guarantee similarity only in the ranking sense, while the canonical-key filter is what prevents exact prompt reuse. If the frontier memory is not yet large enough, or if filtering leaves too few valid candidates, the batch falls back to uniform prompts for the missing positions. This makes early training identical to vanilla RLOO until the memory contains enough frontier examples to support retrieval. The memory is bounded to recent frontier prompts, so retrieval follows the current policy's learning boundary rather than frontier examples that may be stale later in training.

The main run uses a frontier memory window of recent frontier prompts, a replay ratio of 0.10, and a minimum memory size before retrieval starts. With replay ratio $\rho = 0.10$, a batch of 64 prompts contains approximately 58 uniformly sampled prompts and 6 retrieved prompts once memory is active. This keeps most of each batch drawn from the original dataset distribution while allocating a small fraction of training to prompts near the current frontier. The retrieval mechanism changes only the prompt sampling distribution. The verifier reward, RLOO objective, KL penalty, entropy bonus, optimizer settings, SFT initialization, and reference model remain fixed.

B.4 Failure-Mode Taxonomy

We categorize failed rollouts using the generated response, the extracted answer expression, and the verifier outcome. A rollout is counted as failed when its verifier score is less than 1.0. The taxonomy is descriptive and is used only for analysis. It is not used for training, reward computation, or model selection.

Some failed rollouts contain more than one issue. For example, a completion may repeat answer tags and also contain an expression that evaluates to the wrong value. In these cases, we assign the category that best describes the dominant failure mode. Global generation failures, such as missing answer tags, repeated outputs, or corrupted text, are assigned before more local arithmetic labels.

Format error. A format error occurs when the model does not produce a usable final answer in the required format. This includes missing `<answer>` tags, malformed answer tags, empty answer fields, non-arithmetic text inside the answer field, code-like output instead of a mathematical expression, or an expression that cannot be parsed by the verifier. This category is important because Countdown is automatically verified. Even if the reasoning contains useful intermediate steps, the verifier cannot assign full credit unless the final answer is cleanly extractable and parseable.

Wrong target. A wrong-target error occurs when the final answer is a parseable arithmetic expression and uses the allowed numbers correctly, but the expression evaluates to a value different from the target. This category captures arithmetic search failures. The model follows the output format and respects the number-use constraint, but it solves the wrong numerical problem. For example, an expression may use each number exactly once and evaluate to 60 when the target is 98.

Invalid number use. An invalid-number-use error occurs when the final expression is parseable but violates the Countdown constraint on allowed numbers. This includes reusing a number more times than allowed, omitting a required number, introducing an extra constant, or using a modified number that is not produced by valid arithmetic from the allowed set. This category reflects constraint-following failures rather than formatting failures. It is especially relevant for Countdown because a numerically correct expression is still invalid if it uses the wrong multiset of numbers.

Arithmetic error. An arithmetic error occurs when the expression has a valid-looking structure but cannot be evaluated as a valid arithmetic computation. The most common example is division by zero. This category is rare in our runs because most problematic arithmetic outputs either fail parsing and become format errors, or evaluate successfully but reach the wrong target.

Repetitive or incoherent. A repetitive or incoherent error occurs when the model produces an unstable completion rather than a single clean solution. This includes repeated answer tags, repeated verification loops, multiple inconsistent final answers, continued reasoning after a final answer, irrelevant code or markup, corrupted multilingual fragments, or text that no longer follows the task. These errors indicate generation instability. They are different from wrong-target or invalid-number-use errors because the main issue is not a single arithmetic mistake, but a failure to produce a coherent final response.

Other. The other category contains rare failures that do not fit cleanly into the categories above. This includes unusual parser edge cases or outputs whose failure source is ambiguous. Because this category is small, we do not interpret it as a major behavioral pattern.

B.5 ZIP-RC-Lite Implementation Details

This appendix records the non-obvious implementation choices behind the ZIP-RC-Lite results (Section 5.2); the run hyperparameters are in Table 7.

Reserved slice and bin layout. Qwen2.5-0.5B has `vocab_size`=151,936, but the tokenizer only names ids up to 151,664, leaving 271 unused embedding rows. The joint head starts at `DISTRIBUTION_TOKEN_ID`=151,665 and occupies $R \times L$ contiguous ids. Length uses logarithmic-style bins $\{0, 16, 32, 64, 128, 256, 512, 1024\}$ (L=7); a startup bin plus powers of two keeps resolution where Countdown actually lives (responses are $<\sim 300$ tokens), whereas the upstream $\{0, 256, \ldots, 32768\}$ bins target 32k-token math traces and would collapse all mass into bin 0. The bin index is $\mathrm{idx} = \text{(length bin)} + \text{(reward state)} \times L$, with length the fastest-varying axis.

Untie and clone the LM head (the critical bug). Qwen2.5-0.5B sets `tie_word_embeddings=True`, so the output head *is* the input embedding matrix. Training the head while tied leaks gradient into the input embeddings and silently corrupts the policy's generation: offline AUC can look fine ($\sim$0.99) while the decoder emits coherent-but-degenerate tokens (e.g. `<loom>` for `<think>`). The fix is to untie and clone the head into a separate `Linear` (initialized from the input embedding), freeze the input embeddings, and train only the output head. Upstream's larger 1.5B/1.7B models are untied and never hit this. The recurring lesson, validated in the harness, is that offline metrics on clean rollouts can look perfect while generation is broken, so we validate the head's *generation quality* against the raw policy, not just its scoring.

Table 7: ZIP-RC-Lite hyperparameters and run configuration (Countdown / Qwen2.5-0.5B). "Effective batch" is batch size × gradient accumulation. The head occupies $R \times L$ reserved logits (R=2 binary, R=3 structured; L=7 length bins).

Setting	Value
Rollout generation (vLLM, plain policy checkpoint)	
Dataset	`countdown_tasks_3to4` (3–4 numbers)
Samples per prompt	4 (head training) / 64 (calibration set)
Temperature / top-p / top-k / min-p	1.0 / 1.0 / -1 (off) / 0.0
Max new tokens / max model len	1024 / 2048
Stop string	`</answer>`
Seed	42
Verifier / labeling	
Correct (r=1.0)	rule-based `compute_score`=1.0
Structured failure split	coherent (0.1) / incoherent (0.0)
Coherence judge (structured)	Claude Haiku, failures-only, cached, max_tokens 8
Head-only training (backbone frozen, head untied + cloned)	
Trainable params	output head only
Reserved-slice start id	151,665
Length bins (remaining tokens)	$\{0, 16, 32, 64, 128, 256, 512, 1024\}$ (7 bins)
Reward states	binary $\{0, 1\}$ / structured $\{0, 0.1, 1.0\}$
Loss	per-position cross-entropy on joint bin; **no KL**
Optimizer	AdamW, β=$(0.9, 0.95)$, weight decay 0.0
Learning rate / schedule	1×10^{-3} / cosine, warmup ratio 0.05
Batch size / grad accumulation / effective	8 / 4 / 32
Epochs / grad clip / max length	2 / 1.0 / 2048
Precision / attention	bfloat16 / FlashAttention-2 (fallback default)
Gradient checkpointing	off (backbone frozen)
Training set size	256 prompts (main) / 512 prompts (scaled)
Seed	42
Live adaptive decoding	
Parallel samples K / max new tokens	8 / 1024
Temperature / top-p / top-k	1.0 / 1.0 / -1
Warmup / prune interval / $k_{\min}$	128 steps / 32 steps / 2
Value smoothing window	4
`value_end` (selection signal)	mean $\hat{V}$ over last 16 positions
Prune rule	abandon active samples with smoothed $\hat{V} < 0.5$
Utility β sweep	$\{0.002, \ldots, 0.1\}$ (matched-cost near $\{.005, .01, .02, .05\}$)
Early-stop τ sweep	$\{0.7, 0.8, 0.9, 0.95\}$
Cost / latency metric	active forward passes / decode steps
Stop tokens	`<|im_end|>`, `<|endoftext|>`, `</answer>`
Held-out prompts / seeds	50 / 3
Compute	
Modal budget used	≈$15–25 of $400; A10G smoke, single-GPU training

Stop token. The chat model ends a turn with `<|im_end|>`, not `<|endoftext|>`. Stopping only on the latter let every sample run to the 1024-token cap into degenerate text (and made the latency metric meaningless); the decoder stops on either special token or on `</answer>`.

No gradient checkpointing; teacher-forced head loss. The backbone is frozen, so no gradient flows through the transformer body and checkpointing would only waste compute. The loss is computed by a manual `F.linear` over the reserved slice of `output_hidden_states[-1]`, with cross-entropy against the per-position joint bin label.

β normalization and saturation. With K=8 redundancy, the marginal value of an additional sample is small (≈ 0.004), so an initial β range was $\sim$100$\times$ too high and the Pareto frontier looked flat. We swept $\beta \in \{0.002, \ldots, 0.1\}$ and, following the paper, normalize the cost coefficient per prompt by the predicted typical total token count so the reward–cost trade-off is stable across short and long generations.

Cost and latency accounting. Cost is the sum of *active* forward passes (pruned samples stop counting; KV caching is accounted for), matching the paper's normalized-compute metric. Latency is the number of decode steps, a wall-clock proxy: under unconstrained data-parallel sampling, latency is governed by the longest active trajectory. Pruning is implemented by masking abandoned samples and counting only active ones, which yields a faithful cost–accuracy curve without literal KV-row deletion (a later FLOP optimization that gives the same Pareto numbers).

Validation harness. Every live-decode run asserts two invariants: (i) *zero* reserved-token ids appear in any generated sequence (masking works), and (ii) the live per-step AUC(`value_end` $\rightarrow$ correct) matches the offline scorer (the readout is correct in-flight). A value-selection viability gate (`value@`$K \gg$ `random@`K) was required before building the live decoder at all.

B.6 Single-Lever Compute–Accuracy Pareto

Figure 8 isolates the compute axis of the live-decoder operating points in Table 4, i.e. the left panel of the main-text Figure 5 shown on its own. Pruning and the utility policy both move down and left from full Best-of-N, trading a small accuracy change for a large compute reduction, and the simple absolute-threshold prune dominates the redundancy-aware utility policy on Countdown. This is the single-lever, compute-axis result that the blend study below extends by stacking adaptive-K allocation on top of pruning.

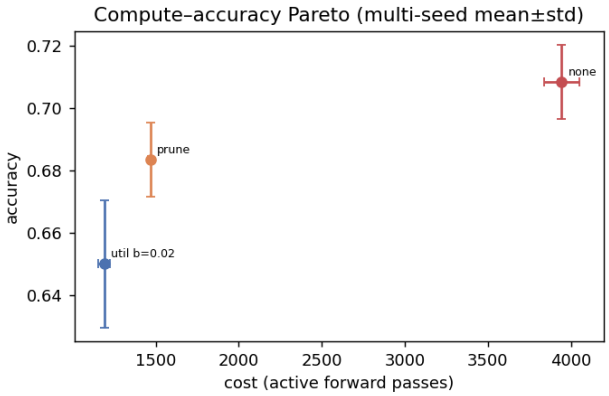

Figure 8: Multi-seed compute–accuracy Pareto for the live decoder (3 seeds, mean $\pm$ std, K=8, scaled head; same runs as Table 4). Pruning and the utility policy move down-left from full Best-of-N ("none"), trading a small accuracy change for a large compute cut.

B.7 Blend Study: Protocol and Detailed Results

This appendix records the protocol and the per-pool numbers behind the blend study shown in Figure 6.

Controller. The blend runs adaptive-K and pruning together. Adaptive-K allocates a fixed *average* sample budget across prompts using a mid-trajectory difficulty probe (the head's `value_q25`, its value read about a quarter of the way through a generation); pruning then abandons predicted-loser samples within each prompt as in the headline decoder.

Evaluation protocol. An earlier single-run result reported a $+0.02$ oracle accuracy gain at -22% compute. That gain was winner's-curse from selecting the best grid point on the same data used to score it, and it does not survive a held-out analysis, so we report only the conservative numbers below. To remove the bias we use (i) prompt-level paired-bootstrap confidence intervals, (ii) held-out 2-fold selection of operating points, so a configuration is chosen on one split and measured on the other, and (iii) four evaluation pools of increasing difficulty (Table 8).

Table 8: Blend-study pools and headline outcomes. "Compute saving" is the held-out reduction in active forward passes at neutral accuracy; the oracle-delta confidence interval includes 0 on every pool. The last row is a control that re-scores the hard pool's generations with an out-of-distribution head.

Pool	Difficulty	Compute saving	Accuracy Δ
Main (n=50, 6 seeds)	standard 3–4 operand	$\sim$20–25%	CI includes 0
Holdout (n=300)	standard, head-disjoint	$\sim$20–25%	CI includes 0
Hard (n=120)	3–6 operand	$\sim$20–25%	CI includes 0
Hard, OOD-head	3–6 operand (control)	n/a	prune cost unchanged

Compounding. On every pool the blend's compute reduction is at least the product of the two levers applied alone, and on the calibrated pools it *super*-compounds, by $+16$ to $+19$ points beyond the product. The mechanism is visible in the keep rate: under adaptive allocation, pruning keeps only $\sim$58% of samples versus $\sim$72% without it, because allocation concentrates samples on hard prompts where the most doomed samples live.

What limits pruning. Prune safety tracks *mid-trajectory separability*, the value AUC at the prune point ($\sim$0.6–0.7, well below the $\sim$0.91 value-end AUC). Within a calibrated pool the higher-separability difficulty tier prunes more safely (2/2 consistent), but as a cross-pool law this is weak: a difficulty-controlled partial correlation gives r=0.35 at permutation p=0.36 (n=9), and an earlier p=0.03 did not survive adding the best-powered pool and dropping a non-independent one. The head-swap control supports this interpretation: re-scoring the same hard generations with an out-of-distribution head leaves the prune accuracy cost unchanged (-0.125 at both difficulty tiers), so the ceiling is the problem's intrinsic predictability, not the head. A practical gate follows: use more aggressive pruning when prune-point separability is above $\sim$0.65 and back off below $\sim$0.6. The latency lever (early-stopping) should be gated on the head's weaker length prediction before being folded into the same utility.