# LoGRA: Scaling LLM Reinforcement Learning with Low-Rank Gradient Sketches

## Paper discussion: method, evidence, and trade-offs

Paper authors: Shaokun Zhang, Yifan Zhang, Jian Hu, Yueying Li,
Hao Zhang, Binfeng Xu, Jan Kautz, Yi Dong (NVIDIA)

arXiv:2610.06647v3 23 October 2026

# A compact update makes a larger RL run feasible

## One-sentence takeaway

LoGRA accumulates low-rank gradient sketches, merges their updates into full weights, and scales each step with a predicted policy-KL budget.

- At 7B, average training memory drops from **31.82 to 17.29 GiB/GPU** (45.7%) with similar MATH-500 peak Pass@1.
- On the same eight-H100 node, dense Adam OOMs at 27B; LoGRA completes training.
- A separate 27B trajectory improves held-out Reasoning-Gym score beyond 1,100 logged steps.

Paper, Abstract; Table 1, PDF p. 7; Fig. 4, PDF p. 7.

# RL memory includes much more than the weights

## A 7B illustration

- BF16 model weights: about 14 GB.
- Two FP32 Adam moment buffers: about 56 GB.
- Gradients and activations add further cost.

Paper, Sec. 1, PDF pp. 1–2.

## The design question

Can the training signal be represented more compactly while still updating the original weight matrices?

**LoGRA's target:** gradient accumulation, optimizer state, and update representation in the training–rollout loop.

# Sketch a matrix gradient in a smaller column space

$$W, G \in \mathbb{R}^{d\times k}, \qquad A \in \mathbb{R}^{r\times k}, \qquad r \ll \min(d,k)$$

$$\boxed{S = GA^\top \in \mathbb{R}^{d\times r}} \qquad \longrightarrow \qquad \boxed{\widehat{G} = SA = (GA^\top)A}$$

- $A$ is an unlearned random projection; default entries are independent $\pm 1/\sqrt{r}$.
- $S$ stores $dr$ numbers instead of $dk$; the reconstructed update has rank at most $r$.
- $\widehat{G}$ approximates $G$; the full-size weight matrix remains trainable.

Paper, Sec. 2.1, Eqs. (1)–(3), PDF pp. 2–3.

# The implementation accumulates sketches directly

## One linear-layer example

If a sample gives $G_i = g_i x_i^\top$, then

$$S = \sum_i G_i A^\top = \sum_i g_i (A x_i)^\top.$$

Project the inputs before forming the weight-gradient product.

- Hold $A$ fixed over microbatches; accumulate $S$ in FP32 without a dense $d \times k$ gradient buffer.
- For $d = k = 4096$, $r = 64$: the FP32 accumulator falls from **64 MiB to 1 MiB**.
- Weights, projections, activations, and other buffers still occupy memory.

Paper, Secs. 2.1–2.2, PDF p. 3. Linear-layer algebra is an explanatory expansion of Eq. (2).

# Low-rank increments are merged into full weights

$$U = \mathrm{Optimizer}(S), \qquad \boxed{W \leftarrow W - \alpha\eta UA}$$

- For sketch SGD, $U = S$; the main math experiments use RowAdam.
- Refreshing $A$ each step changes the update subspace; the sum of updates need not have rank $r$.
- In the proposed factorized synchronization, rollout regenerates $A$ from a seed and receives $\alpha\eta U$.

## Measurement scope

The main benchmark protocol reports *full-weight policy synchronization*; Table 1 does not measure factorized communication savings.

Paper, Sec. 2.2 and Algorithm 1, PDF pp. 3–4; Appendix D.1, PDF p. 16.

# RowAdam retains inexpensive adaptivity

$$v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2)\frac{1}{r}\sum_{j=1}^{r}(S_t)_{ij}^2, \qquad (\widetilde{U}_t)_{i,:} = \frac{(S_t)_{i,:}}{\sqrt{\widehat{v}_{t,i}} + \epsilon_t}$$

- One second-moment statistic per sketch row; $\beta_2 = 0.95$.
- Rescale $\widetilde{U}_t$ to match the Frobenius norm of $S_t$.
- No first-moment momentum: RowAdam is *not* dense Adam with a compressed storage format.

Paper, Appendix B, Eq. (15), PDF pp. 14–15.

# Predicted KL measures policy sensitivity to an update

$$D_W = -\eta UA, \qquad \overline{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta+\alpha D}) \approx \alpha^2 q(D)$$

$$q(D) = \frac{1}{2|\mathcal{P}|\,T^2} \sum_{s \in \mathcal{P}} \mathrm{Var}_{v \sim p(s)}[\dot{z}_v(s)]$$

- $\mathcal{P}$: sampled next-token contexts; $p(s) = \mathrm{softmax}(z(s)/T)$.
- $\dot{z}_v(s)$: directional logit change along the proposed update $D$.
- The variance ignores a common shift of all token logits, which leaves softmax unchanged.

Paper, Sec. 2.3, Eqs. (5)–(6), PDF pp. 4–5; Appendix A, PDF pp. 13–14.

# A KL budget determines the global step multiplier

$$\alpha = \min\left\{\alpha_{\max}, \sqrt{\delta / q(D)}\right\}$$

- $\delta$ is the allowed *predicted average* next-token KL for a training step.
- If $q(D) = 4\delta$, choose at most half the nominal update size.
- A cosine-annealed budget reduces allowed policy movement later in training.
- The approximation is local; realized KL can differ from the prediction.

## Implementation detail

The long 27B run uses a finite-difference probe based on direct KL evaluations rather than the JVP expression above.

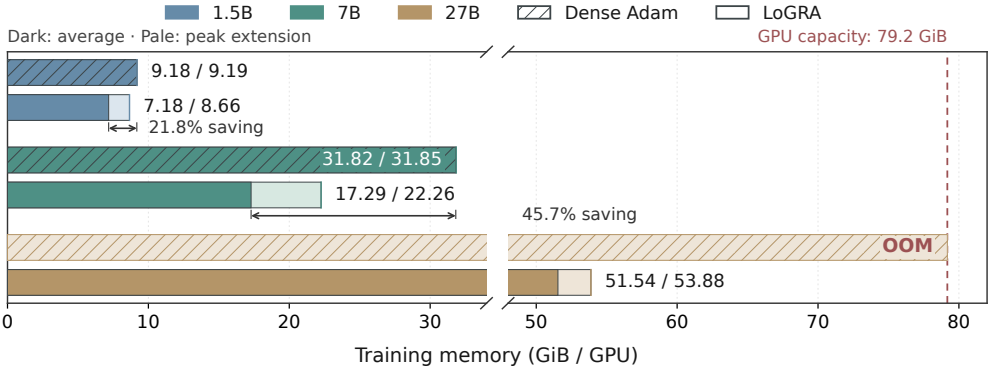
Paper, Sec. 2.3, Eq. (7), PDF p. 5; Sec. 3.3, PDF p. 7.

# The main comparison uses a fixed eight-GPU allocation

| Training protocol | Reported metrics |
| --- | --- |
| • Qwen2.5-Math-1.5B/7B; Qwen3.8-27B.<br>• DAPO-Math-7.5K, 128 prompts/update.<br>• Four FSDP training H100s; four generation H100s.<br>• LoGRA: rank 256 + RowAdam + annealed KL budget.<br>• Dense: Adam + annealed learning rate. | • Training-GPU memory: temporal mean of per-update peak allocation.<br>• Peak accuracy: best MATH-500 evaluation in updates 0–200.<br>• Throughput: completed updates per hour.<br>• Three seeds for completed configurations. |

Paper, Secs. 3.1–3.2, PDF p. 5; Table 1, PDF p. 7; Appendix C–D, PDF pp. 15–16.

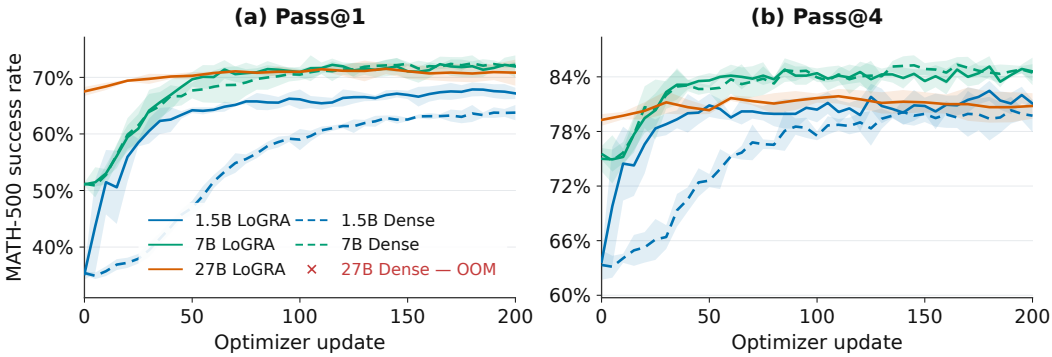
# The memory advantage grows with model scale




- Average memory: **21.8% less at 1.5B**; **45.7% less at 7B**.
- 27B dense Adam OOM is a capacity mark, not a measured dense-memory value.

Original paper Fig. 1, PDF p. 1; numerical values in Table 1, PDF p. 7. Broken axis: 34–48 GiB.

# Compressed updates preserve competitive learning




- At 7B: peak Pass@1 **72.33% LoGRA vs 72.48% dense** over updates 0–200.
- At 1.5B: 67.87% vs 63.77%; dense 27B does not complete an update.

Original paper Fig. 2, PDF p. 6; Table 1, PDF p. 7. Curves show three-seed mean ± sample SD.

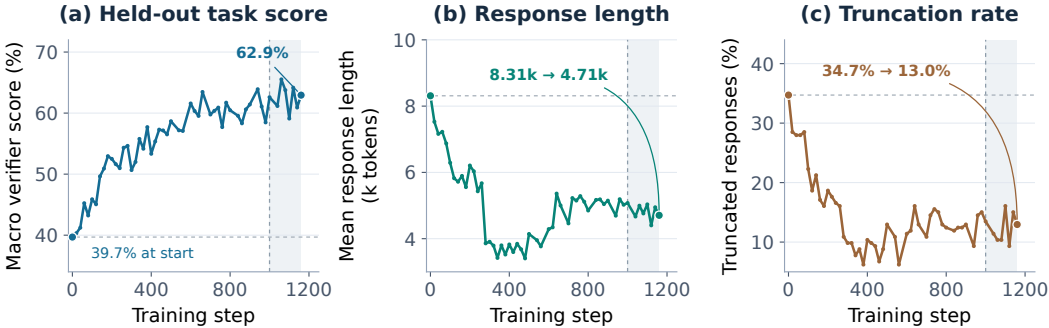
# Table 1 shows memory savings, not a large speedup

| Model | Method | Avg. GiB/GPU ↓ | Updates/h ↑ | Peak P@1 ↑ |
|---|---|---|---|---|
| 1.5B | Dense | 9.18 | 54.28 | 63.77% |
| 1.5B | LoGRA | **7.18** | 55.44 | **67.87%** |
| 7B | Dense | 31.82 | 25.28 | **72.48%** |
| 7B | LoGRA | **17.29** | 25.73 | 72.33% |
| 27B | Dense | OOM at initial Adam-state allocation | | |
| 27B | LoGRA | 51.54 | 6.99 | 71.52% |

- The 27B case has no measured dense throughput or memory ratio.
- Accuracy peaks use updates 0–200; memory and throughput summarize the full runs.

Adapted from paper Table 1, PDF p. 7; protocol in Sec. 3.2, PDF p. 5.

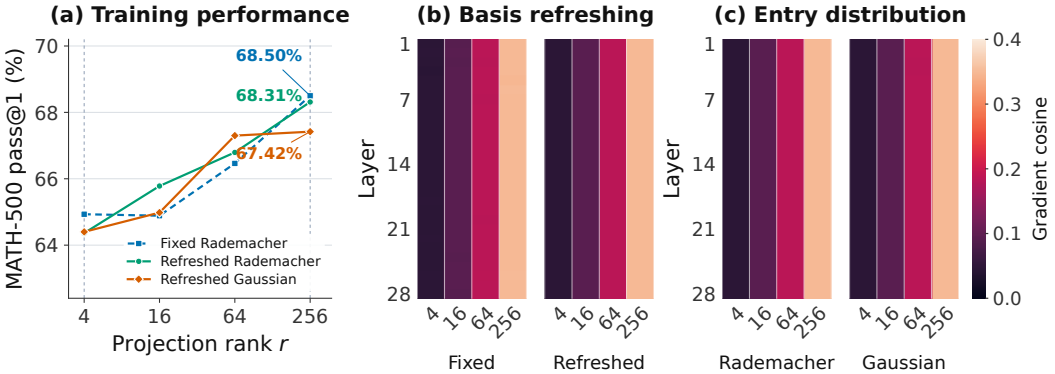
# A separate 27B run sustains learning beyond 1,100 steps




- Eight-task held-out macro score: **39.69%** → **62.94%**; peak 65.52% at step 1,060.
- Mean response length: 8,312 → 4,705 tokens; truncation: 34.72% → 12.95%.

Original paper Fig. 4 and Sec. 3.3, PDF p. 7. One trajectory; 193 held-out problems.

# Projection rank matters more than basis details here




- Rank 4 → 256 improves Pass@1 by 3.02–3.95 percentage points.
- At rank 256, fixed and refreshed Rademacher score 68.50% and 68.31%: no consistent short-horizon refresh benefit.

Original paper Fig. 5, PDF p. 8. (b,c) show gradient-reconstruction cosine, not task accuracy.

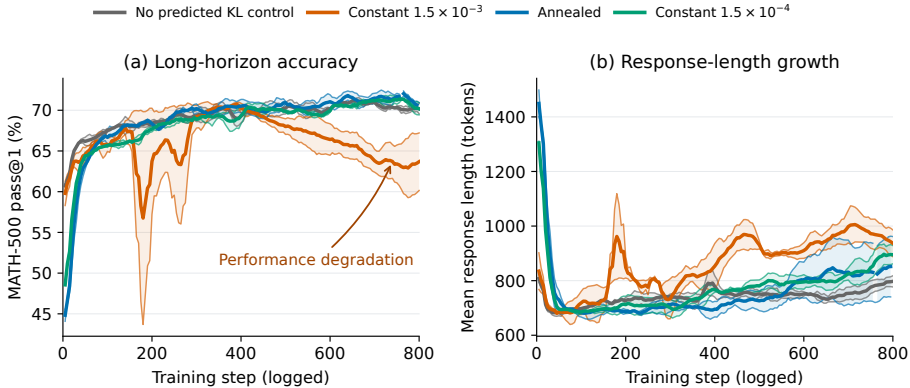
# LoGRA trades some speed and Pass@1 for lower memory vs LoRA

| Method | Avg. GiB ↓ | Peak GiB ↓ | Pass@1 ↑ | s/update ↓ |
|---|---|---|---|---|
| LoRA | 13.21 | 13.38 | **70.45%** | **10.63** |
| LoGRA | **7.18** | **8.66** | 68.47% | 11.31 |

- Rank-256 Qwen2.5-Math-1.5B; three seeds per method.
- LoGRA Pass@4 is 81.80% vs 81.40% for LoRA.
- The time metric excludes generation and synchronization; optimizers and schedules differ.

Adapted from paper Table 2 and Sec. 3.4.2, PDF p. 9; Appendix D.4, PDF p. 17.

# KL control helps only with a well-chosen budget




- Tail Pass@1, steps 700–800: 70.30% (none), 63.43% (large), 71.03% (small), 71.47% (annealed).
- No-control also remains stable; two-seed runs do not isolate annealing alone.

Original paper Fig. 6 and Sec. 3.4.3, PDF pp. 9–10. Curves smoothed; tail means use raw evaluations.

# Read the evidence at its measured scope

1. **Strongest result:** substantial training-GPU memory savings and 27B feasibility under the reported eight-H100 allocation.
2. **Comparable learning:** 7B MATH-500 peak accuracy is close to dense Adam; 1.5B is higher under the reported recipes.
3. **Protocol differences:** RowAdam vs dense Adam, and LoGRA vs LoRA uses different optimizers and schedules.
4. **Unmeasured here:** the main system comparison uses full-weight synchronization, so its numbers do not quantify factorized communication savings.

Paper, Tables 1–2, PDF pp. 7, 9; Appendix B/D.1/D.4, PDF pp. 14–17.

# Take-home messages

## Method

Directly accumulated $S = GA^\top$ supports a compact optimizer state and a low-rank increment to full weights.

## Control

Predicted KL scales the increment in policy space; conservative budgets perform best in the reported long-horizon ablation.

## Result

Up to 45.7% lower average training memory in the main math experiments, plus a successful single-node 27B run where dense Adam OOMs.

Paper, Secs. 2–3 and Conclusion, PDF pp. 2–10.

# Backup: why predicted KL is quadratic

For fixed context $s$, define $K_s(a) = D_{\mathrm{KL}}(p(0) \| p(a))$ with $p(a) = \pi_{\theta + aD}(\cdot \mid s)$.

$$K_s(0) = 0, \qquad K_s'(0) = 0, \qquad K_s(a) = \tfrac{1}{2} a^2 K_s''(0) + O(a^3).$$

For $p_v(a) = \mathrm{softmax}(z_v(a)/T)$,

$$\left. \frac{d}{da} \log p_v(a) \right|_0 = \frac{\dot{z}_v - \mathbb{E}_{u \sim p} \dot{z}_u}{T}, \qquad K_s''(0) = T^{-2} \, \mathrm{Var}_{v \sim p}[\dot{z}_v].$$

Average over sampled contexts to obtain $q(D)$ in Eq. (6).

Paper, Appendix A.1, Eqs. (8)–(13), PDF pp. 13–14.

# Backup: experimental settings that affect interpretation

- Main runs: three seeds $\{1, 2, 42\}$; BF16, activation checkpointing, no CPU offload.
- LoGRA rank 256, projection refreshed every update; predicted-KL budget $2 \times 10^{-4} \rightarrow 2 \times 10^{-5}$ over 300 actual updates.
- Main 27B math runs end at 200 updates without shortening the budget schedule.
- Long 27B Reasoning-Gym run: seed 42, SGD sketch, 16,384-token context; budget annealed over 1,000 merges.
- LoRA comparison uses 300 logged steps for LoRA and 300 actual updates for LoGRA.

Paper, Appendix C–D, PDF pp. 15–17.