Sparse Steps to Reasoning

Anonymous Authors[1]

Abstract

Recent advances in reinforcement learning (RL) have shown promise for training language models on multi-step reasoning tasks. However, conventional RL methods apply dense gradient updates uniformly across parameters, leading to gradient interference when multiple reasoning steps compete for shared weights. Moreover, large updates can damage language-critical "superweights," causing catastrophic degradation of linguistic capabilities. We propose **Sparse Steps to Reasoning (SSR)**, a step-wise reinforcement learning framework that improves credit assignment and optimization efficiency for multi-step reasoning. SSR combines (1) superweight discovery and freezing to preserve language ability and (2) step-specific parameter allocation so that different reasoning steps primarily update different subsets of parameters. By aligning optimization with the localized structure of RL-induced updates, SSR enables individual reasoning steps to be optimized with reduced interference and more precise gradient signals. Training Qwen3-1.7B and DeepSeek-R1-Distill-Qwen-1.5B on DeepMath-103K, SSR improves AIME 1983-2024 accuracy by +6.0pp on average versus +3.9pp for the strongest GSPO baseline. An ablation further indicates that step-specific parameter allocation provides additional gains beyond step-level credit assignment alone.

1. Introduction

Training language models for multi-step reasoning with reinforcement learning requires addressing two related challenges: *credit assignment* (determining which reasoning steps contributed to success or failure) and *parameter interference* (avoiding conflicting gradient updates when multiple steps share weights).

Recent group-based policy optimization methods (Shao et al., 2024; Zheng et al., 2025) improve training stability using sequence-level importance ratios and group-relative advantages. However, reinforcement learning in general remains sensitive to optimization instabilities, with coarse credit assignment and unstable gradient interactions. **What is a "reasoning step"?** In this paper, a reasoning step is a contiguous segment of the model's generated solution that corresponds to a single transformation in the chain of reasoning (for example, introducing a variable, applying an algebraic rule, simplifying an expression, or checking a constraint). We extract steps by parsing the output into an ordered list using explicit delimiters when present (for example, newlines or enumerated lists), and a lightweight rule-based fallback otherwise. Existing methods treat reasoning chains as single units and assign episode-level rewards uniformly across all steps. This obscures which specific steps require correction and leads to gradient interference when different reasoning steps induce conflicting updates on shared parameters.

We introduce **Sparse Steps to Reasoning (SSR)**, a reinforcement learning framework that addresses gradient interference through disjoint parameter allocation. Specifically, SSR employs an LLM-based critic to evaluate each reasoning step independently, generating fine-grained reward signals. The KronosOpt optimizer implements step-specific parameter allocation across steps so that each step primarily updates its own subset of parameters. The allocation operates sequentially during each training iteration: step 1 claims its top-k parameters, step 2 selects from the remaining pool (excluding step 1's allocation), and so forth. In practice, this encourages low overlap between step-specific update regions, reducing gradient interference while still allowing limited parameter reuse when beneficial.

In summary, we make the following contributions:

- We introduce SSR, a step-wise RL framework that combines step-level credit assignment with explicit interference management.
- We adapt superweight discovery and freezing to protect language-critical parameters from destabilizing RL updates.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

- We propose KronosOpt, a low-overlap masking optimizer that allocates mostly-disjoint parameter subsets to different reasoning steps using a step-dependent budget schedule.
- We demonstrate improved AIME 1983–2024 performance and ablate the contribution of low-overlap allocation beyond step-level credit assignment.

Experiments on mathematical reasoning show promising results. Training Qwen3-1.7B and DeepSeek-R1-Distill-Qwen-1.5B on DeepMath-103K, SSR with Kronos optimizer achieves +6.0pp average gain on AIME 1983-2024 versus +3.9pp for best GSPO baseline. Ablation studies show that disjoint parameter allocation contributes +3.6pp beyond step-level credit assignment alone, demonstrating that explicit management of parameter interference is critical for multi-step reasoning tasks.

2. Proposed Method

2.1. Training Pipeline Overview

SSR operates in two distinct phases. First, *superweight discovery* identifies parameters critical for basic language understanding through gradient analysis. Second, *step-wise reinforcement learning* applies selective updates to develop reasoning capabilities while protecting the identified linguistic foundation. This separation ensures reasoning improvements never corrupt the model's language abilities.

2.2. Phase 1: Superweight Discovery

Before RL training begins, we identify language-critical parameters to protect. Recent work (Yu et al., 2024) demonstrated that a small fraction of parameters ($\sim$0.5%) disproportionately influence language generation; modifying these "superweights" causes catastrophic degradation. We adapt this insight for RL fine-tuning.

Discovery procedure. We identify candidate superweights via gradient magnitude analysis on general language tasks, then validate criticality through KL divergence measurement. Concretely:

1. **Language prompts**: Sample 8–16 non-reasoning prompts (e.g., short QA / completion) and compute gradients under the pretrained model.
2. **Candidate collection**: Take the top 5% parameters by gradient magnitude as candidates.
3. **Union search**: Form multiple candidate unions U across prompt subsets.
4. **Forward-pass ablation (KL test)**: For each union U, create a masked model by setting the selected parameters to zero *in the forward pass* (used only for measurement), and compute

$$\mathrm{KL}(U) = D_{\mathrm{KL}}(\pi_{\text{masked at } U} \| \pi_{\text{reference}}). \quad (1)$$

5. **Selection**: Choose $G = \arg\max_U \mathrm{KL}(U)$.

We mark a candidate set as language-critical if masking causes a large distribution shift (KL ≥ 1). We select KL ≥ 1 because a KL divergence of 1 nat indicates substantial divergence from the reference distribution, corresponding to large probability mass disagreement under Pinsker's inequality (TV $\geq \sqrt{\mathrm{KL}/2} \approx 0.71$). Starting from 4% of parameters, we iteratively reduce this fraction until reaching the minimal set sufficient to preserve language capabilities (typically 0.5%). Sensitivity analysis (Appendix C) confirms this choice: 0.25% protection causes language degradation by step 200, while 1.0% yields similar stability but reduces reasoning capacity by 0.4pp.

How masking is used. During superweight discovery, "masking" means forward-pass ablation (set candidate parameters to zero) used only to measure KL. During RL training, we do not ablate weights in the forward pass; we only block gradients for the protected set G (freeze those parameters), which preserves normal forward behavior.

The identified parameters form a frozen set G excluded from all subsequent RL updates.

2.3. Phase 2: Step-Wise Reinforcement Learning

With language-critical parameters identified and protected, we now apply step-wise reinforcement learning with disjoint parameter updates. This phase combines three key components: (1) multi-sample generation with step-level critic evaluation, (2) hierarchical advantage computation integrating step-level and episode-level signals, and (3) the KronosOpt optimizer implementing disjoint coordinate masking.

2.3.1. Multi-Sample Generation and Step-Level Critic

For each prompt, we generate multiple response samples (e.g., 8). We parse each response to extract individual reasoning steps, then evaluate each step independently using an LLM-based critic with a task-specific rubric. The critic considers accumulated context from previous steps when scoring each new step.

The critic receives full context (question + previous steps + current step + rubric) and produces both qualitative analysis and a numerical score. We score intermediate reasoning steps on a $[-100, 100]$ scale to allow penalizing harmful steps, while final answers use a $[0, 100]$ scale.

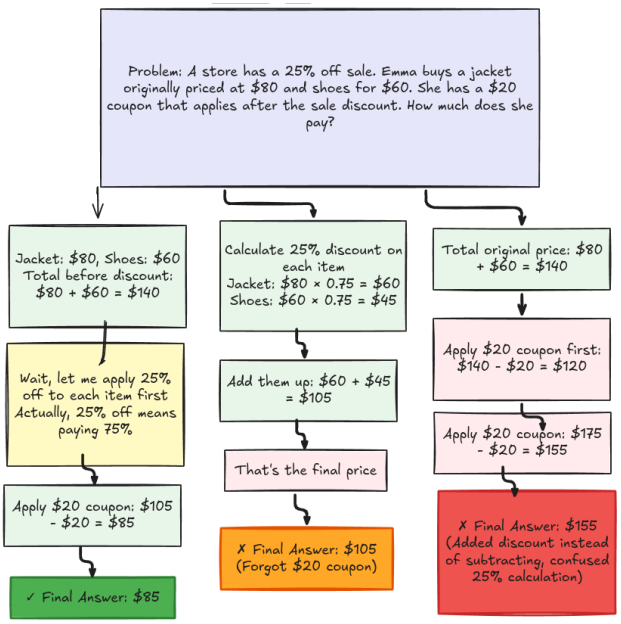

Figure 1. Example reasoning trace and step segmentation used by SSR (illustrative). We parse model outputs into an ordered list of steps (via explicit delimiters when present, with a lightweight fallback otherwise) and score each step conditionally on the previous context.

Reward processing. We normalize raw critic scores in a range-preserving way, using separate normalizations for intermediate steps and final answers:

$$\hat{r}_{\text{step}} = \frac{r}{100}, \qquad r \in [-100, 100] \tag{2}$$

$$\hat{r}_{\text{final}} = \frac{r - 50}{50}, \qquad r \in [0, 100]. \tag{3}$$

Both map to $[-1, 1]$, and we use the normalized scores in advantage computation.

2.3.2. Advantage Computation

We compute advantages at two levels: across samples (group-relative) and within samples (between steps). For each query, we generate G samples (typically $G = 8$). Each sample j contains T_j reasoning steps with critic scores $r_{i,j}$ for step i.

Group-Relative Normalization: Following GSPO (Zheng et al., 2025), we normalize step rewards across all samples responding to the same query. We flatten all step-level rewards within the group and compute group statistics:

$$\mu_{\text{group}} = \frac{1}{\sum_{j=1}^{G} T_j} \sum_{j=1}^{G} \sum_{i=1}^{T_j} r_{i,j} \tag{4}$$

$$\sigma_{\text{group}} = \sqrt{\frac{1}{\sum_{j=1}^{G} T_j} \sum_{j=1}^{G} \sum_{i=1}^{T_j} (r_{i,j} - \mu_{\text{group}})^2} \tag{5}$$

Each step advantage is normalized using group statistics:

$$A_{i,j}^{\text{group}} = \frac{r_{i,j} - \mu_{\text{group}}}{\sigma_{\text{group}} + 10^{-8}} \tag{6}$$

This eliminates question difficulty bias: samples performing better than the group average receive positive advantages.

Within-Sample Normalization: Within each sample, we further normalize advantages to emphasize which steps contributed most. For sample j, we compute:

$$\mu_j = \frac{1}{T_j} \sum_{i=1}^{T_j} A_{i,j}^{\text{group}} \tag{7}$$

$$\sigma_j = \sqrt{\frac{1}{T_j} \sum_{i=1}^{T_j} (A_{i,j}^{\text{group}} - \mu_j)^2} \tag{8}$$

The final advantage for each step is:

$$A_{i,j}^{\text{final}} = \frac{A_{i,j}^{\text{group}} - \mu_j}{\sigma_j + 10^{-8}} \tag{9}$$

This two-level normalization provides both cross-sample comparison (which reasoning chains are better) and within-sample credit assignment (which steps within a chain contributed most).

2.3.3. KronosOpt: Mostly-Disjoint Parameter Updates

The KronosOpt optimizer implements disjoint coordinate masking to prevent gradient interference across reasoning steps. The key architectural insight is creating a three-tier parameter hierarchy: (1) protected language-critical parameters identified via superweight discovery, (2) step-specific reasoning parameters allocated disjointly across steps, and (3) shared parameters receiving no updates. This design prevents the failure mode documented by Yu et al. (Yu et al., 2024): naive optimization would frequently select superweights for large updates (since they naturally have large gradients), corrupting language capabilities.

Claimed Coordinate Tracking: The optimizer maintains a running set of claimed parameter coordinates C, initialized with the protected set G from superweight discovery.

Per-Step Mask Selection: When processing gradients for reasoning step i:

1. Compute eligible coordinates: $E_i = \text{all\_params} \setminus (G \cup S_1 \cup \ldots \cup S_{i-1})$
2. Allocate a total per-problem budget B of available parameters (we use $B = 5\%$) and set a step-dependent per-step budget k_i. Concretely, we run a one-time *budget profiling* pass that measures how gradient magnitudes decay with step index on representative models

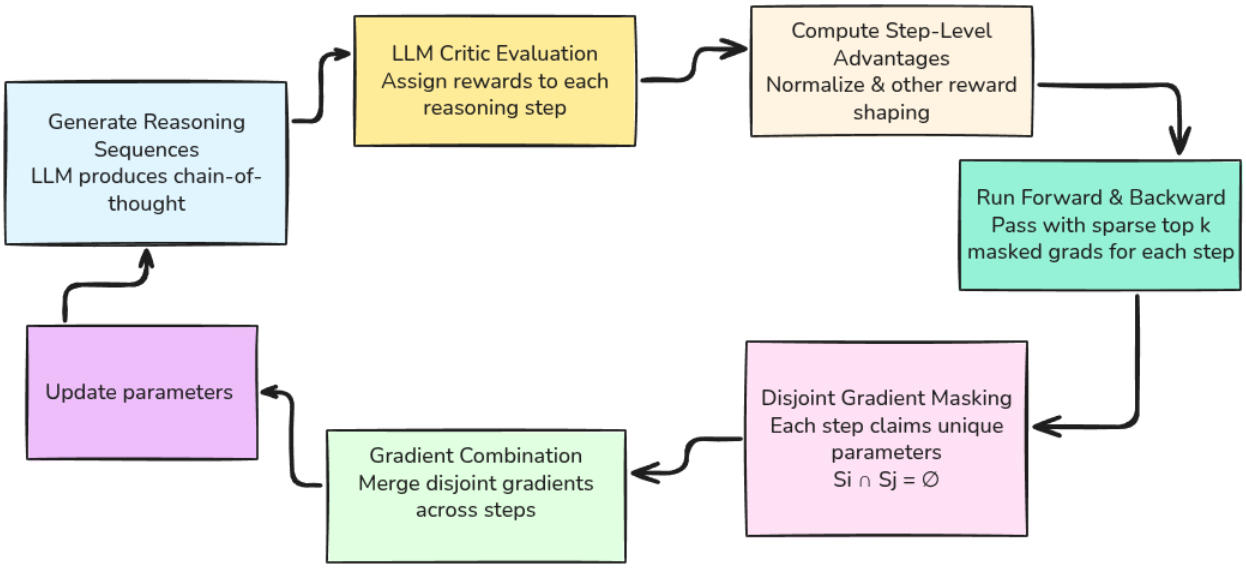

Figure 2. SSR architecture overview showing the three main phases: (Phase 0) Superweight discovery identifies critical parameters to protect, (Phase 1) Step-wise RL with group relative scoring generates advantages per reasoning step, and (Phase 2) Mostly-disjoint (low-overlap) parameter allocation encourages step-specific update regions with low overlap, reducing gradient interference.

(Qwen3-1.7B and Qwen3-8B). Let v_i denote the average (across profiled problems/models) magnitude mass available at step i under the "unclaimed-coordinate" constraint; since v_i typically decays with i, we convert this curve into *compensating* normalized weights via $w_i = \frac{(v_i+10^{-8})^{-1}}{\sum_{t=1}^{T}(v_t+10^{-8})^{-1}}$, so that $\sum_{i=1}^{T} w_i = 1$ and later steps receive larger weights. We then set $k_i = \min(k_{\max}, B \cdot w_i)$, with $k_{\max} = 1\%$ used only as a cap for very short chains.

3. Select top-k_i gradient magnitudes within E_i to form step-specific set S_i
4. Update claimed set: $C \leftarrow C \cup S_i$

Memory savings via block-sparse masks. Although each parameter tensor is large (e.g., 1024×1024 blocks for typical weight matrices), the step-selected masks are extremely sparse at the block level: top-k gradients concentrate in a small number of tiles, and many of those tiles recur across steps/problems due to consistent backprop structure and partial mask overlap. As a result, the vast majority of mask blocks remain zero and never need to be materialized. We therefore store per-step and cumulative masks in Block Sparse Row (BSR) format, which stores only the indices and values of nonzero blocks; in practice, this reduces mask memory substantially compared to dense boolean masks while preserving fast bitwise composition when cached (Appendix A.4).

2.3.4. Accelerated Top-k Selection

Selecting top-k coordinates from eligible sets of size $|\theta| \approx 10^9$ can be a computational bottleneck. Naive sorting requires $O(n \log n)$ comparisons per step, totaling $O(Tn \log n)$ per training problem.

Table 1. Memory and runtime comparison for mask storage strategies on Qwen3-1.7B with $T = 20$ reasoning steps. BSR with caching achieves near-optimal memory while avoiding repeated conversion overhead. T denotes the number of unique parameter tensors tracked.

Storage Strategy	Mask Memory	Conversions/Problem	Overhead
Dense (naive)	18.4 GB	0	—
BSR (no cache)	9.7 GB	$20T$	+41%
BSR + cache (ours)	9.8 GB	T	+3%

Block-level selection. We exploit the observation that high-magnitude gradients cluster spatially due to the block structure of attention and MLP weights. Our algorithm:

1. Reshape scores into 16×16 blocks
2. Compute max score per block: $O(n)$
3. Select top blocks with $2\times$ oversampling: $O(b \log b)$ where $b = n/256$
4. Refine to exact k within selected blocks: $O(2k \log 2k)$

This reduces complexity from $O(n \log n)$ to $O(n + b \log b)$, yielding a $\sim 16\times$ speedup on 7B models. The block size aligns with the BSR storage boundaries (Appendix A.4), avoiding index translation overhead.

Mask storage. Tracking per-step and cumulative masks can be memory-intensive. Table 5 summarizes memory and conversion overheads for alternative mask storage strategies.
Step-Specific Allocation: Our goal is to associate distinct parameter subsets with different reasoning steps so that step-specific gradients primarily target different coordi-

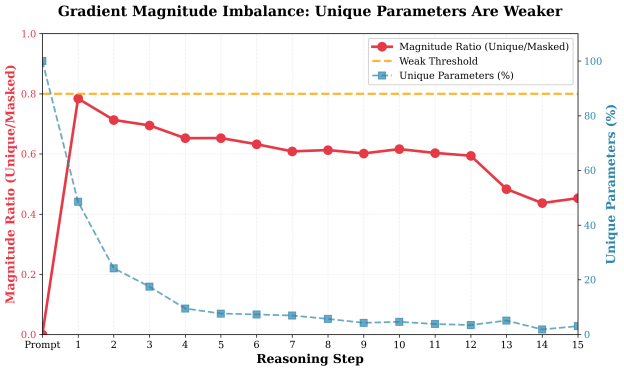

Figure 3. Gradient budget profiling on representative models shows that, as the reasoning trace progresses, the fraction of newly available (unclaimed) high-magnitude coordinates drops sharply. To avoid late-step signal starvation, we use a step-dependent budget schedule: earlier steps use smaller k_i and later steps receive larger k_i in top-k_i selection so that each step contributes a non-negligible update.

nates, which reduces destructive interference while protecting language-critical weights. We enforce that step-selected parameters avoid the protected set G. Across steps, we preferentially select from unclaimed coordinates first, which encourages low overlap between step-specific update regions, but we do not require a strict zero-overlap guarantee to obtain the benefits in practice.

Advantage fusion (implementation detail). For efficiency on long traces, adjacent steps with very similar advantages can be fused into a single update unit (we sum their parameter budgets and treat the fused unit as one selection/update), reducing redundant masking work while preserving the low-overlap selection rule.

Parameter Update Policy: SSR updates only the top-k parameters selected by KronosOpt for each reasoning step, with superweights G strictly excluded from updates. This ensures language-critical parameters remain frozen during reasoning fine-tuning. Future work may explore updating superweights with much smaller learning rates to enable gradual linguistic adaptation alongside reasoning improvements.

3. Experiments

3.1. Experimental Setup

We evaluate SSR on mathematical reasoning using two small language models: Qwen3-1.7B (Team, 2025) and DeepSeek-R1-Distill-Qwen-1.5B. Training runs with 4 questions per batch. For each question, we generate 8 response samples using temperature 0.7. An LLM-based critic evaluates each reasoning step independently using a rubric that assesses correctness, progress toward solution, and clarity.

Algorithm 1 Two-Level Advantage Computation

Input: Question q, group size $G = 8$
Sample G responses: $\{y_1, \ldots, y_G\} \sim \pi_\theta(\cdot|q)$
for each response y_j **do**
  steps$_j \leftarrow$ ParseSteps(y_j)
  for each step $s_{i,j} \in$ steps$_j$ **do**
    $r_{i,j} \leftarrow$ Critic$(q, \text{steps}_{<i,j}, s_{i,j})$
  end for
end for
// Group-relative normalization
$\mu_{\text{group}} \leftarrow \text{mean}(\{r_{i,j} : j \in [1, G], i \in [1, T_j]\})$
$\sigma_{\text{group}} \leftarrow \text{std}(\{r_{i,j} : j \in [1, G], i \in [1, T_j]\})$
for each step (i, j) **do**
  $A^{\text{group}}_{i,j} \leftarrow (r_{i,j} - \mu_{\text{group}})/(\sigma_{\text{group}} + 10^{-8})$
end for
// Within-sample normalization
for each response y_j **do**
  $\mu_j \leftarrow \text{mean}(\{A^{\text{group}}_{i,j} : i \in [1, T_j]\})$
  $\sigma_j \leftarrow \text{std}(\{A^{\text{group}}_{i,j} : i \in [1, T_j]\})$
  for each step $i \in [1, T_j]$ **do**
    $A^{\text{final}}_{i,j} \leftarrow (A^{\text{group}}_{i,j} - \mu_j)/(\sigma_j + 10^{-8})$
  end for
end for
Return: $\{A^{\text{final}}_{i,j}\}$ for all steps in all responses

Table 2. Example training-stability statistics (800 steps). We report a small subset of logged metrics to highlight optimization behavior.

Method	LR	Steps	Final loss	Mean reward	Max reward	Max grad norm
GSPO	5×10^{-6}	800	0.3222	0.1675	1.0055	0.3981
SSR (AdamW)	5×10^{-6}	800	0.0723	0.2005	1.4162	0.4530

Baselines: We compare against GSPO (Zheng et al., 2025) under matched conditions (same model architectures, dataset, and sampling settings). The methods differ in reward granularity and optimization: GSPO uses sequence-level rewards with standard optimization, while SSR uses step-wise critic evaluation and disjoint masking via KronosOpt. We also report SSR (Adam) to isolate the effect of disjoint allocation from step-level supervision.

Training stability and learning-rate robustness. We found that KronosOpt's low-overlap masked updates are substantially more stable than dense-update baselines at higher learning rates. In pilot sweeps, increasing the learning rate by an order of magnitude led to collapse for GSPO and dense SSR variants, while SSR (KronosOpt) remained stable due to its small-grained, safety-constrained updates.

Superweight Configuration: Superweight discovery identified ∼0.5% of parameters as language-critical (8.5M for Qwen3-1.7B, 7.5M for DeepSeek-R1-Distill-Qwen-1.5B). We determined this threshold through iterative refinement:

Algorithm 2 KronosOpt: Low-Overlap Masked Gradient Update

Input: Model parameters θ, superweight mask G, advantages $\{A_i\}$, total budget $B = 5\%$, cap $k_{\max} = 1\%$
Initialize: Claimed set $C \leftarrow G$ (superweights)
// Process each reasoning step sequentially
for step $i = 1, 2, \ldots, n$ **do**
  // Backward pass for step i generates gradients $\nabla_\theta L_i$
  Compute eligible coordinates: $E_i \leftarrow \{\theta\} \setminus C$
  Mask gradients outside eligible region: $g_i \leftarrow \nabla_\theta L_i \odot \mathbf{1}_{E_i}$
  Compute step-balanced budget: $k_i \leftarrow \min(k_{\max}, B \cdot w_i)$, where weights w_i come from budget profiling and satisfy $\sum_{i=1}^{T} w_i = 1$
  $S_i \leftarrow \text{BalancedTopK}(|g_i|, k_i)$ // Indices of top $k_i\%$ gradients in E_i
  Update claimed set: $C \leftarrow C \cup S_i$
  Store masked gradient: $\tilde{g}_i \leftarrow g_i \odot \mathbf{1}_{S_i}$
end for
// Apply advantage-weighted updates
for each parameter $\theta_k \in \theta$ **do**
  $\Delta\theta_k \leftarrow 0$
  for step $i = 1, 2, \ldots, n$ **do**
    if $k \in S_i$ **then**
      $\Delta\theta_k \leftarrow \Delta\theta_k + \eta_{\text{step}} \cdot \tilde{g}_{i,k} \cdot A_i$
    end if
  end for
  $\theta_k \leftarrow \theta_k - \Delta\theta_k$ // Apply accumulated update
end for
Goal: Encourage step-specific update regions with low overlap, while enforcing $S_i \cap G = \emptyset$ for all i

starting from 4%, we progressively reduced the protected fraction while measuring KL divergence, using KL ≥ 1 as our deterioration criterion. At 0.5%, masking causes full deterioration while minimizing frozen capacity. These parameters remain frozen during RL training.

Scalable Parameter Allocation: The remaining 99.5% of parameters are available for reasoning optimization via **dynamic pre-budgeting**. A fixed top-k fraction per step (for example, 1% for every step) is not scalable for long reasoning chains: gradient analysis shows that later steps have systematically smaller gradient magnitudes and fewer newly available (unclaimed) high-magnitude coordinates, which can lead to late-step signal starvation and early exhaustion of the eligible pool.

To address this, we allocate a fixed total budget $B = 5\%$ of available parameters per problem and distribute it across steps using a step-dependent schedule. Concretely, we run a one-time *budget profiling* pass that measures how gradient magnitudes decay with step index on representative models, including Qwen3-1.7B and Qwen3-8B, and use the resulting average curve to define normalized weights w_i with $\sum_{i=1}^{T} w_i = 1$. For a chain with T reasoning steps, we then allocate per-step budgets $k_i = \min(k_{\max}, B \cdot w_i)$, with $k_{\max} = 1\%$ used only as a cap for very short chains. This gives smaller k_i to early steps and larger k_i to later steps so that each step contributes a non-negligible update.

We keep the same total budget across problems to maintain parameter efficiency; the $B = 5\%$ choice was selected empirically for typical AIME chain lengths (8-25 steps): 10% yielded marginal gains (+0.3pp) at significantly higher computational cost, while 2.5% showed slight degradation (-0.5pp).

Variable-Length Chains: Within each batch, problems have different reasoning chain lengths. Gradients are computed per-step within each sample, then averaged across samples at corresponding step indices. The dynamic pre-budgeting uses a step-dependent schedule $\{k_i\}_{i=1}^{T}$ with $k_i = \min(k_{\max}, B \cdot w_i)$ and $\sum_i w_i = 1$, so each problem instantiates its own per-step budgets based on its chain length T.

Computational Overhead: KronosOpt's disjoint masking adds minimal overhead (<5% wall-clock time) over standard optimizers. The masking cost is dominated by set operations for tracking claimed coordinates. The larger overhead (∼30%) comes from step-level critic evaluation, which is shared with SSR (Adam) and comparable to other modern RL methods using LLM judges (Zheng et al., 2023; Dubois et al., 2023).

Critic Model and Reliability: Step-level grading benefits from a reasonably capable model; very small critics were unreliable at scoring intermediate steps in our pilot experiments. We use Grok-4.1-Fast (non-reasoning mode) as a fast, cost-effective step-level critic, and observed comparable behavior from critics in a similar capability tier (e.g., GPT-OSS 120B, Gemini Flash).

Evaluation Benchmark: We evaluate on AIME 1983-2024 (933 competition-level mathematics problems spanning 42 years). Results are averaged over 3 runs; standard deviations are ±0.3pp for SSR variants and ±0.4pp for GSPO baselines.

3.2. Results

Table 1 presents our main results, including a critical ablation isolating the contribution of disjoint parameter allocation.

Isolating reward granularity from disjoint allocation. To disentangle the effect of step-level supervision from disjoint parameter allocation, SSR (Adam) uses the *identical* step-level critic, two-level advantage normalization, and

Table 3. Performance on AIME 1983-2024 (933 problems), averaged over 3 runs (± standard deviation). SSR (Kronos) achieves +6.0pp average gain compared to +3.9pp for best GSPO baseline (CoT rewards).

METHOD	DEEPSEEK-R1-DISTILL	QWEN3-1.7B
	Pretrained Baseline	
BASE MODEL	22.4%	23.47%
	After Training	
GSPO (FINAL ANSWER)	25.29±0.4% (+2.89PP)	25.93±0.4% (+2.46PP)
GSPO (COT REWARDS)	26.36±0.4% (+3.96PP)	27.33±0.4% (+3.86PP)
SSR (ADAM)	24.54±0.3% (+2.14PP)	26.15±0.3% (+2.68PP)
SSR (KRONOS)	**28.40±0.3% (+6.00PP)**	**29.58±0.3% (+6.11PP)**
	Kronos vs Best GSPO	
IMPROVEMENT	**+2.04PP**	**+2.25PP**

Table 4. Comparison of training variants (average across both models), reporting gain over the pretrained baseline. Rows are not a sequential "add one component" progression; in particular, SSR (Adam) underperforms GSPO (CoT) despite using step-level credit, suggesting that the *structure* of parameter updates matters.

CONFIGURATION	GAIN OVER PRETRAINED
GSPO (FINAL ANSWER)	+2.9PP
GSPO (COT REWARDS)	+3.9PP
SSR (ADAM)	+2.4PP
SSR (KRONOS)	+6.0PP

training hyperparameters as SSR (Kronos), differing *only* in the optimizer. SSR (Adam) thus directly answers the question: "What if step-level rewards are applied with a standard optimizer and no disjoint allocation?" This controlled comparison isolates KronosOpt's contribution from step-level supervision.

Ablation: Disjoint Parameter Allocation. Comparing SSR (Adam) vs SSR (Kronos) isolates the effect of disjoint parameter masking while holding step-level credit assignment constant. SSR (Adam) applies standard dense gradient updates to all non-superweight parameters, while SSR (Kronos) enforces disjoint parameter allocation via coordinate masking.

Results show SSR (Kronos) outperforms SSR (Adam):

- DeepSeek-R1-Distill: +3.86pp improvement (28.40% vs 24.54%)
- Qwen3-1.7B: +3.43pp improvement (29.58% vs 26.15%)

This ablation shows that step-level credit assignment alone (SSR Adam) provides modest gains over GSPO baselines (+2.4pp average), while disjoint parameter allocation (SSR Kronos) adds further improvement (+3.6pp beyond SSR Adam, +6.0pp total). We hypothesize that, under standard dense updates, step-level signals can be partially blended in shared parameters (especially when early steps dominate the available high-magnitude coordinates), whereas GSPO (CoT) aggregates process information into a single sequence-level reward without attempting to enforce per-step separation.

By protecting language-critical parameters identified through superweight discovery (Yu et al., 2024), SSR maintains linguistic coherence during reasoning training.

3.3. Why Disjoint Parameter Allocation Improves Learning

The SSR (Adam) vs SSR (Kronos) ablation isolates the effect of disjoint allocation while holding step-level credit assignment fixed, showing a +3.6pp gain attributable to disjoint masking.

Mechanism diagnostics (lightweight). As lightweight evidence to make the interference story more direct, future work should report simple diagnostics alongside performance: (i) overlap/uniqueness statistics of step-selected coordinates across steps (e.g., $|S_i \cap S_j|$ and Jaccard overlap), and (ii) gradient conflict metrics across steps (e.g., cosine similarity between per-step gradients before masking), and relate these measures to the observed SSR (Adam) vs SSR (Kronos) gap.

Independent validation of gradient conflict. Concurrent work provides complementary evidence for the same interference mechanism. DaGRPO (TODO: fill authors, 2025) analyzes how counteracting updates on shared parameters can create a "tug-of-war" dynamic, leading to unstable, zigzag optimization trajectories rather than a consistent descent direction. They further argue that this conflict can effectively wash out learning signals for persistent structure (e.g., reusable reasoning templates or grammatical regularities), since correct patterns can be penalized when they co-occur with incorrect reasoning in negative samples.

DaGRPO mitigates this at the *sample level* by selecting more homogeneous comparison pairs, whereas SSR targets it at the *parameter level*: by encouraging different reasoning steps to update mostly-disjoint coordinates, SSR reduces within-trajectory tug-of-war between steps.

Gradient interference elimination. Standard optimizers (Adam) allow multiple reasoning steps to compete for the same parameters. When Step 3 has advantage +0.8 and Step 5 has advantage -0.6, their conflicting signals on shared parameter θ_i produce:

$$\Delta\theta_i = \eta \cdot (g_3 \cdot 0.8 - g_5 \cdot 0.6) \tag{10}$$

The resulting update is a noisy compromise. KronosOpt mitigates this interference by encouraging low overlap between step-specific update regions, giving each step more unambiguous credit attribution on its allocated parameters.

Dense updates without disjointness fail. SSR (Adam) applies standard dense updates to all non-superweight parameters. Its underperformance relative to SSR (Kronos) despite using the same step-level credit suggests that dense accumulation can still blur step-specific learning signals when multiple steps compete in shared parameters.

Protected superweights prevent collapse. Both SSR variants protect language-critical parameters identified via superweight discovery. The fact that SSR (Adam) achieves reasonable (though suboptimal) performance validates that superweight protection successfully preserves linguistic capabilities during reasoning training.

4. Related Work

4.1. Gradient Interference and Multi-Task Learning

Gradient interference occurs when updates for different objectives conflict, degrading performance. Recent work validates this problem for LLMs: Zhang et al. (Zhang et al., 2024) found sparse training reduces conflicts by limiting parameter influence, while TSSR (Wang et al., 2025) showed that different tasks utilize distinct subnetworks and that "simultaneous parameter updates cause gradients from one task to disrupt specialized subnetworks." A 2025 study (Chen et al., 2025) demonstrated RL fine-tuning naturally produces sparse updates (5-30% of parameters), validating SSR's sparse approach.

SSR encourages low overlap between step-specific update regions through sequential allocation, reducing parameter-level interference in practice. Alternative architectural approaches include Mixture of Experts (MoE) architectures (Si et al., 2023) with learned routing per reasoning step, which provide implicit parameter separation. Future work should explore these complementary approaches and hybrid strategies combining explicit allocation with learned routing.

4.2. Group-Based Policy Optimization and Process Rewards

GRPO (Shao et al., 2024) introduced group-relative advantages for variance reduction. GSPO (Zheng et al., 2025) improved stability with sequence-level importance ratios. SSR builds on these foundations by adding step-level credit assignment with disjoint parameter isolation. Process reward models (Lightman et al., 2023) evaluate intermediate steps but require human annotation; SSR's LLM-based critic provides automated step-level evaluation.

5. Conclusion

We presented Sparse Steps to Reasoning (SSR), a reinforcement learning framework that addresses gradient interference in multi-step reasoning through mostly-disjoint (low-overlap) parameter allocation. SSR contributions include: (1) a conservative superweight protection strategy motivated by preliminary experiments showing catastrophic collapse with naive top-K selection, (2) a low-overlap parameter allocation mechanism that reduces gradient interference in practice, and (3) a simplified two-level advantage normalization approach for step-wise credit assignment. Our experimental results indicate that low-overlap allocation adds gains beyond step-level credit assignment alone. Training Qwen3-1.7B and DeepSeek-R1-Distill-Qwen-1.5B on DeepMath-103K, SSR (Kronos) achieves +6.0pp average improvement on AIME 1983-2024 compared to +3.9pp for GSPO with CoT rewards. The performance gap between SSR (Adam) and SSR (Kronos) isolates the architectural contribution of low-overlap parameter masking from the algorithmic contribution of step-level evaluation.

6. Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Chen, A., Kim, S., and Patel, R. Rl finetunes sparse subnetworks in large language models. *arXiv preprint*, 2025. Under review.

Daneshmand, H., Joudaki, A., and Bach, F. Batch normalization is blind to the first and second order statistics of the data. *arXiv preprint arXiv:2510.26788*, 2025.

Dubois, Y., Li, X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P., and Hashimoto, T. B. AlpacaFarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Si, C., Chen, W., Zhao, Z., et al. Mixture of reasoning experts for multi-hop question answering. *arXiv preprint arXiv:2305.14628*, 2023.

Team, Q. Qwen3-1.7b. https://huggingface.co/Qwen/Qwen3-1.7B, 2025.

TODO: fill authors. DaGRPO: Debiased and aligned group relative policy optimization. *arXiv preprint arXiv:TODO*, 2025. TODO: replace this placeholder entry with the official BibTeX (authors/venue/arXiv id).

Wang, J., Li, H., and Zhou, T. Task-specific subnetwork refinement for large language models. *Machine Learning*, 114:1–25, 2025.

Yu, M., Wang, D., Shan, Q., Reed, C. J., and Wan, A. The super weight in large language models. *arXiv preprint arXiv:2411.07191*, 2024.

Zhang, W., Liu, Y., and Chen, M. Proactive gradient conflict mitigation via sparse training. *arXiv preprint*, 2024. Under review.

Zheng, C., Liu, S., Li, M., Chen, X.-H., Yu, B., Gao, C., Dang, K., Liu, Y., Men, R., Yang, A., Zhou, J., and Lin, J. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36, 2023.

A. Implementation Details

This appendix provides comprehensive implementation details for reproducing SSR experiments, including training procedures, hyperparameters, and system requirements.

A.1. KronosOpt: Design Rationale and Architecture

A.1.1. The Problem: Gradient Interference in Multi-Step RL

In standard reinforcement learning for multi-step reasoning, each reasoning step receives its own credit signal (advantage) from the critic. However, when gradients from multiple steps are applied to the same parameters, their signals interfere:

Example: Consider parameter θ_i that influences both Step 3 (advantage $A_3 = +0.8$) and Step 5 (advantage $A_5 = -0.6$). Standard optimizers apply:

$$\Delta\theta_i = \eta \cdot (g_3 \cdot A_3 + g_5 \cdot A_5) = \eta \cdot (0.8g_3 - 0.6g_5) \tag{11}$$

The resulting update is a conflicting mixture where the positive signal from Step 3 is partially canceled by the negative signal from Step 5, even though these steps performed different reasoning operations. This interference degrades credit assignment precision.

A.1.2. The Solution: Low-Overlap Parameter Allocation

KronosOpt reduces interference by encouraging low overlap between step-specific update regions. The core principle is maintaining a **claimed coordinate set** that tracks which parameters have been allocated to which steps.

Low-overlap objective: For steps $i \neq j$, KronosOpt encourages low overlap between their claimed parameter sets (while always enforcing $S_i \cap G = \emptyset$). This ensures:

$$\Delta\theta_k = \begin{cases} \eta_{\text{step}} \cdot g_3 \cdot A_3 & \text{if } k \in S_3 \\ \eta_{\text{step}} \cdot g_5 \cdot A_5 & \text{if } k \in S_5 \\ 0 & \text{otherwise (or no update if } k \in G) \end{cases} \tag{12}$$

Each parameter receives an unambiguous credit signal from exactly one step, enabling precise attribution of learning to specific reasoning operations.

A.1.3. Two-Tier Parameter Architecture

KronosOpt maintains two disjoint parameter tiers:

1. **Superweights (G, 0.5%)**: Critical parameters for language understanding, identified via reverse gradient search. These are frozen (excluded from gradient updates) to prevent degradation of linguistic capabilities during reasoning optimization.
2. **Step-Specific Parameters ($S_1, S_2, \ldots, S_n$)**: Each reasoning step sequentially claims its top-k_i parameters from the remaining unclaimed space. We use dynamic pre-budgeting with a step-dependent schedule: for a chain with T steps, we allocate a total budget $B = 5\%$ of the non-superweight parameters and set per-step budgets using normalized weights w_i obtained from a one-time budget profiling pass (gradient magnitude decay vs. step index), with $\sum_{i=1}^{T} w_i = 1$ and $k_i = \min(k_{\max}, B \cdot w_i)$. Step 1 selects from all non-superweight parameters, Step 2 selects from what Step 1 did not claim, Step 3 selects from what Steps 1-2 did not claim, and so on. This sequential allocation ensures each step optimizes distinct parameters without interference.

Sequential Selection Rationale: By allowing each reasoning step to claim its top-K most relevant parameters from the unclaimed pool, we ensure that:

- Each step identifies and updates the parameters most critical for its specific reasoning operation
- Later steps have access to a distinct parameter pool, preventing interference with earlier steps

- The model develops specialized parameter subsets for different types of reasoning steps

This sequential top-K mechanism helps reduce gradient interference by encouraging low overlap between step-specific update regions. In particular, by prioritizing selection from unclaimed coordinates, we reduce the extent to which different steps compete in shared parameters.

A.1.4. Why Not Use Standard Optimizers?

Standard optimizers like AdamW apply gradients uniformly:

$$\theta_{t+1} = \theta_t - \eta \cdot \text{AdamW}(\nabla_\theta) \tag{13}$$

For multi-step RL, this means:

$$\nabla_\theta = \sum_{i=1}^{n} A_i \cdot g_i \tag{14}$$

When advantages $\{A_i\}$ have mixed signs and the same parameters appear in multiple g_i, the resulting updates are noisy and imprecise. KronosOpt resolves this by ensuring each g_i targets distinct parameters, making the advantage signals orthogonal in parameter space.

Sparse optimization and parameter efficiency. Our ablation results align with recent findings on sparse training dynamics. Chen et al. (2025) observed that RL fine-tuning naturally produces sparse updates (5-30% of parameters), validating SSR's sparse approach. However, their work did not investigate structured sparsity or disjoint allocation.

Our ablation suggests that the *structure* of sparsity matters as much as sparsity itself. Unstructured sparse updates (Adam with top-K selection) may cause parameter starvation when multiple steps compete for limited budget, while structured disjoint allocation ensures each step receives dedicated capacity.

Following the findings of (Yu et al., 2024), we observe that a small fraction of parameters (0.5%) are disproportionately critical for basic language modeling. These parameters, when modified even slightly, cause dramatic degradation in the model's ability to generate coherent text. For reasoning-specific RL training, these parameters should be either:

- **Frozen entirely**: No updates, preserving linguistic capabilities exactly

Exclusion from Top-K Selection: Critically, superweights are **excluded from the eligible pool** when selecting step-specific parameters. When Step i selects its top-K parameters, it chooses from:

$$E_i = \text{all parameters} \setminus (G \cup S_1 \cup \ldots \cup S_{i-1}) \tag{15}$$

This ensures that superweights are never claimed by any reasoning step, preventing aggressive reasoning-specific updates from corrupting the language foundation.

A.1.5. Discovery Procedure

We include the core superweight discovery procedure in the main paper (Section 2.1) to keep the key method self-contained; this appendix provides complementary context and rationale.

A.1.6. Relationship to Apple's Superweight Work

Our approach is inspired by (Yu et al., 2024), which identified that single parameters can have outsized influence on LLM behavior. While their work focused on identifying individual critical weights for quantization, we extend this concept to identify a *set* of critical parameters (0.5% rather than single weights) for protection during RL training.

Key difference: Apple's method identifies parameters that cause model collapse when *removed*. In our discovery phase, we measure distribution shift by ablating candidate parameters in the forward pass (used only for the KL test). During RL training, we protect the discovered set by blocking gradients (freezing), rather than removing parameters.

A.1.7. Why Small Iterative Changes Matter

RL training naturally produces noisy gradient signals, especially in multi-step reasoning, where:

- Credit assignment can be imprecise across long chains
- Advantages have high variance even with normalization
- Early training steps explore suboptimal policies

Applying these noisy signals with large learning rates to superweights would quickly degrade language capabilities. By protecting superweights, we ensure that RL makes **targeted reasoning improvements** rather than destabilizing the entire model. The model learns to reason *on top of* its linguistic foundation rather than *at the expense of* it.

A.2. Training Infrastructure

Both SSR and GSPO baseline experiments were run with batch size 4 on an NVIDIA H200 GPU. Complete implementation details, hyperparameters, and instructions for reproducing both SSR training and GSPO baseline experiments are available in the code repository's `README.md`.

A.3. LLM-Based Critic Configuration

A.3.1. Critic Model

We use Grok-4.1-Fast (non-reasoning mode) via API for step-level evaluation, selected for its balance of scoring quality and inference throughput. In pilot experiments, very small/weak critics often failed to score intermediate steps meaningfully; we found that critics in a similar capability tier (e.g., GPT-OSS 120B, Gemini Flash) also provide reliable step-level signals.

A.3.2. Critic Prompt and Scoring

Each reasoning step is scored conditionally on the question, the previous steps, the current step, and a task-specific rubric.

A.3.3. Reward Processing

We describe reward normalization in the main paper (Section 2.2); this appendix focuses on additional implementation details.

A.4. Efficient Mask Storage via Block Sparse Row Tensors

A practical challenge in SSR is memory overhead: tracking disjoint parameter allocations across T reasoning steps requires storing $T + 1$ boolean masks (one per step plus the superweight mask G), each of size $|\theta|$. For a 7B parameter model with 20 reasoning steps, naive dense storage would require $21 \times 7\text{B} \times 1$ bit ≈ 18GB—nearly doubling memory consumption.

Block Sparse Row Format. We observe that step-specific parameter masks exhibit high spatial sparsity: each step claims only 0.1–0.5% of parameters, and gradient magnitudes cluster in contiguous regions due to the block structure of attention and MLP weights. This motivates storing masks in PyTorch's Block Sparse Row (BSR) format with 16×16 blocks, matching the natural blocking of matrix operations on modern GPUs.

BSR stores only non-zero blocks plus index arrays, reducing storage from $O(|\theta|)$ to $O(nnz + n_{\text{blocks}})$ where nnz is the number of non-zero elements. For masks with 0.25% density, this yields approximately 50% memory reduction compared to dense boolean tensors, as the block overhead is amortized across many zero regions.

Hybrid Caching Strategy. While BSR provides compact storage, repeated sparse↔dense conversions during top-k selection would dominate runtime. We employ a hybrid strategy:

- **Long-term storage:** Superweight masks and cumulative claimed masks persist in BSR format
- **Active computation:** Dense masks are cached on first access and updated in-place via bitwise OR as steps claim new coordinates
- **Lazy conversion:** BSR→dense conversion occurs only once per training problem; the cache persists across all T steps

Block Size Alignment. Critically, the BSR block size (16×16) matches the block size used in our accelerated top-k selection (Section 2.3.4). This alignment ensures that when we aggregate gradient magnitudes into blocks for coarse selection, the resulting mask naturally aligns with BSR storage boundaries, avoiding index translation overhead.

Table 5 reports memory usage across configurations. The BSR+caching strategy reduces mask storage by 47% while adding only 3% runtime overhead from the initial conversion.

Storage Strategy	Mask Memory	Conversions/Problem	Overhead
Dense (naive)	18.4 GB	0	—
BSR (no cache)	9.7 GB	$20T$	+41%
BSR + cache (ours)	9.8 GB	T	+3%

Table 5. Memory and runtime comparison for mask storage strategies on Qwen3-1.7B with $T = 20$ reasoning steps. BSR with caching achieves near-optimal memory while avoiding repeated conversion overhead. T denotes the number of unique parameter tensors tracked.

B. Theoretical Motivation for Sparse Gradient Updates in Reinforcement Learning

This section presents the theoretical foundations motivating SSR's sparse gradient approach, drawing on empirical observations from recent work on parameter-efficient fine-tuning and numerical precision in reinforcement learning.

B.1. The Delicate Nature of RL Weight Updates

Reinforcement learning for language model fine-tuning exhibits fundamentally different characteristics from supervised fine-tuning (SFT) or pretraining. Multiple lines of evidence suggest that RL induces substantially smaller magnitude weight updates:

Low-Rank Adaptation in RL vs. SFT. LoRA (Hu et al., 2021) shows that parameter-efficient adaptation is often possible in fine-tuning. While this is not direct evidence about RL sparsity, it motivates investigating whether RL fine-tuning also admits structured, low-dimensional (or sparse) updates.

Numerical Precision Sensitivity. In practice, RL training can be more sensitive than supervised fine-tuning to small numerical differences between BF16 and FP16 representations, which is consistent with RL operating in a regime of finer-grained parameter adjustments.

Non-Determinism from Reduction Operations. Distributed training introduces non-deterministic behavior through tensor parallel reductions and batch-invariant operations (Daneshmand et al., 2025). While SFT tolerates minor numerical variations from these operations, RL training can exhibit divergent behavior. This further supports the view that RL can be sensitive to small perturbations.

Sparse Circuit Hypothesis. Work in mechanistic interpretability suggests that complex behaviors can be localized to sparse subnetworks. Combined with the observations above, this motivates the hypothesis that RL may primarily modify a sparse subset of parameters rather than requiring dense updates across all weights.

B.2. Implications for Sparse Optimization

The preceding observations converge on a central insight: *RL fine-tuning makes small, localized parameter changes rather than large, dense modifications*. This naturally motivates sparse gradient approaches:

1. **Sparsity sufficiency**: If RL's effective updates are low-rank and localized, dense gradient computation is wasteful. Sparse updates targeting high-impact parameters should suffice.

2. **Precision preservation**: Sparse updates reduce numerical accumulation errors by limiting the number of parameters receiving small gradient contributions that may be corrupted by floating-point imprecision.

3. **Targeted modification**: By explicitly selecting which parameters to update, sparse methods avoid inadvertently modifying regions of weight space that are critical for model stability.

B.3. Why Naïve Top-K Gradient Selection Fails

A natural implementation of sparse gradient optimization would compute full gradients, select the top-K coordinates by magnitude, and apply updates only to those positions. However, this naïve approach leads to immediate model collapse in practice.

The failure mechanism relates directly to superweights. Naïve top-K selection operates on instantaneous gradient magnitudes without considering parameter criticality. Since superweights govern fundamental language understanding capabilities, they frequently receive large gradient signals during RL training. A naïve top-K selector would:

1. Identify superweights as having large gradients
2. Apply aggressive learning rate updates ($\eta \sim 10^{-4}$–10^{-5}) to these coordinates
3. Rapidly degrade the model's linguistic foundation
4. Produce incoherent text output within dozens of training steps

B.4. SSR's Resolution: Protected Superweights with Sequential Step Allocation

SSR resolves this failure mode through two mechanisms:

Superweight Protection: By identifying critical parameters a priori (Phase 1; Appendix A) and excluding them from the eligible top-K pool, SSR ensures aggressive reasoning-specific updates never corrupt the language foundation. Superweights are frozen during RL training, preserving linguistic capabilities.

Sequential Step Allocation: Rather than allowing multiple reasoning steps to compete for the same parameters (causing gradient interference), KronosOpt sequentially allocates low-overlap parameter sets. Step 1 claims its top-K coordinates, Step 2 claims from the remaining pool, and so forth. This reduces conflicts in shared parameters while still allowing limited parameter reuse when beneficial.

The combination enables SSR to exploit RL's sparse update structure while avoiding the failure modes of naïve sparse optimization. Reasoning improvements occur in dedicated parameter regions, language understanding remains protected, and gradient interference is reduced through mostly-disjoint (low-overlap) allocation.

B.5. Empirical Validation

The observed improvements over GSPO (Table 3) provide empirical support for this theoretical framework. GSPO applies dense gradient updates uniformly, suffering from both superweight degradation and gradient interference. SSR's sparse, protected approach achieves better performance with fewer effective parameters updated per step, suggesting that RL's true learning signal may be sparse and that explicit management of this sparsity may provide benefits.

C. Additional Discussion and Future Work

Future work includes: (1) evaluating SSR on larger models and additional reasoning domains, (2) comparing strict disjointness against overlap-permitting variants, and (3) stress-testing on longer reasoning chains to quantify any budget-related effects.