A FEW GOOD WEIGHTS: SPARSE SUBNETWORKS IN PREFERENCE-OPTIMIZED LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning fine-tuning of Large Language Models (LLMs) induces sparse, full-rank weight updates concentrated subnetworks containing fewer than 2.5% of parameters. We show the structure of these subnetworks in common settings is determined primarily by the learning objective, rather than the training data. Subnetworks produced by DPO and GRPO differ substantially in parameter selection and internal representation, even on similar data, while subnetworks produced by the same objective on different datasets remain closely aligned. To establish this result rigorously, we develop a theoretical framework for identifying the sparse subnetwork by scoring each parameter's importance. We derive an optimal Oracle scoring function with guarantees on loss retention, prove the existence of an optimal scoring function at any sparsity level, establish a certifiability condition for evaluating practical scoring functions, and develop a scalable threshold estimation algorithm for constructing subnetworks in multi-billion-parameter LLMs. We propose warm-start magnitude scoring, which recovers dense DPO and GRPO training dynamics at or beyond 97.5% sparsity across mathematics, coding, syntax, and semantics. We characterize objective-specific subnetwork divergence through Jaccard similarity, Centered Kernel Alignment, and linear probing. Finally, we introduce a sparsity-aware AdamW optimizer kernel implemented in Triton that reduces optimizer step complexity by the factor $(1-\rho)$, achieving $\sim 1.7\times$ and $\sim 3.4\times$ speedups over PyTorch and HuggingFace 8-bit AdamW respectively at 97.5% sparsity.

1 INTRODUCTION

Preference optimization methods (POMs), including Reinforcement Learning from Human Feedback (RLHF), Direct Preference Optimization (DPO), and Group Relative Policy Optimization (GRPO), have enabled Large Language Models (LLMs) to better align with human intent and improve performance in domains such as mathematics and coding (Christiano et al., 2017; Ouyang et al., 2022; Rafailov et al., 2023; DeepSeek-AI et al., 2025; Dubey et al., 2024). Full-parameter fine-tuning of POMs is prohibitively expensive at billion-parameter scale, motivating Parameter-Efficient Fine-Tuning (PEFT) methods such as Low-Rank Adaptation (LoRA) (Hu et al., 2022), which approximates weight updates with low-rank matrices.

Recent empirical evidence shows that RL fine-tuning weight updates are not low-rank and instead exhibit a sparse, high-rank structure (Mukherjee et al., 2025). These sparse RL updates suggest parameter selection as a more principled lens than low-rank decomposition for RL fine-tuning. Corroborating evidence comes from the Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019; Frankle et al., 2019), which proposes that dense networks contain sparse subnetworks, often referred to as winning tickets, capable of matching full model performance.

In this work, we investigate whether winning tickets exist in modern RL algorithms, specifically DPO and GRPO, and characterize their structure. We show that task-relevant sparse subnetworks can be identified in a single pass, without expensive iterative pruning, from signals derived from early optimization steps. We also find that in common settings a subnetwork's structure is more strongly determined by the *learning objective* than by the training data: DPO and GRPO subnetworks differ substantially in parameter selection and internal representations even on similar data, while subnetworks trained under the same objective on different datasets remain closely aligned.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Our contributions are as follows:

- We develop a theoretical framework for subnetwork discovery in preference optimization. We derive an optimal Oracle scoring function with guarantees on loss retention, prove the existence of an optimal scoring function at any sparsity level, establish a certifiability condition for evaluating practical scoring functions, and develop a scalable threshold estimation algorithm for constructing subnetworks in billion-parameter LLMs.
- We show that subnetworks identified by warm-start magnitudes recover dense training performance at up to 97.5% sparsity in DPO and GRPO, and demonstrate that subnetwork structure is more strongly influenced by the learning objective rather than the dataset content with standard hyperparameter choices, which we characterize via Jaccard similarity, Centered Kernel Alignment (CKA), and linear probing.
- We introduce a training pipeline featuring a Block Sparse Row (BSR) AdamW optimizer, implemented with Triton kernels, which reduces optimizer step times during fine-tuning by exploiting subnetwork sparsity while preserving the rank of weight updates. We report optimizer-step speedups on `Llama-3.1-8B-Instruct` at 97.5% sparsity.

2 Related Work

Preference Optimization of LLMs. Reinforcement Learning from Human Feedback (Christiano et al., 2017; Ouyang et al., 2022) has become the dominant paradigm for aligning LLMs with human preferences, typically using PPO (Schulman et al., 2017) to optimize a learned reward model. Direct Preference Optimization (Rafailov et al., 2023) simplifies the RLHF pipeline by expressing the reward model implicitly through the language model itself, yielding a stable and lightweight alternative. Group Relative Policy Optimization (DeepSeek-AI et al., 2025) reduces memory costs by replacing the value function with group-relative advantage estimates, enabling efficient RL fine-tuning for reasoning tasks.

Lottery Ticket Hypothesis and Sparse Subnetworks. The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) showed that dense networks contain sparse subnetworks that match full model performance when trained from their original initialization. Winning tickets have been found in pre-trained BERT models (Chen et al., 2020) and in RL agents (Yu et al., 2020; Vischer et al., 2021), demonstrating that sparse subnetworks arise across diverse learning settings. Burkholz et al. (2022) proved the existence of universal lottery tickets that generalize across tasks. Panda et al. (2024) showed that weight-delta magnitude scoring on short dense trajectories identifies task-relevant subnetworks in the fine-tuning setting.

Pruning Methods. Identifying which parameters can be removed without degrading model performance has been studied extensively. Iterative magnitude pruning (Han et al., 2015; Frankle & Carbin, 2019) requires multiple rounds of training and pruning, making it computationally demanding at LLM scale. One-shot alternatives instead identify parameters to prune in a single pass without retraining. Panda et al. (2024) identify subnetworks using parameter-update magnitude, motivated by the intuition that important weights tend to be those that change the most during fine-tuning (Vischer et al., 2021). Gradient-based approaches include SNIP (Lee et al., 2019), which scores parameters by their gradient-weight products, and GraSP (Wang et al., 2020), which scores parameters by their contribution to gradient flow preservation. For LLMs specifically, Wanda (Sun et al., 2023) incorporates activation statistics alongside weight magnitudes, and SparseGPT (Frantar & Alistarh, 2023) uses second-order information to update remaining weights after pruning to compensate for the accuracy loss. On the theoretical side, Liebenwein et al. (2020) bound the size and performance of the pruned network as a function of filter sensitivity scores, and Baykal et al. (2019) derive data-dependent compression bounds on how much a network can be pruned while preserving output quality.

Efficient Sparse Parameter Fine Tuning A parallel line of work reduces fine-tuning costs by restricting parameter updates. LoRA Hu et al. (2022) and its variants constrain updates to low-rank subspaces, substantially reducing memory and compute (Dettmers et al., 2023). Mukherjee et al. (2025) showed that RL fine-tuning induces sparse, practically full-rank updates concentrated in small subnetworks, a structure distinct from LoRA's low-rank assumption, motivating hardware-aware sparse implementations over low-rank factorizations. Achieving computational savings from such

sparse models requires specialized kernels. Triton Tillet et al. (2019) provides a framework for writing custom GPU kernels, SparseProp (Nikdan et al., 2023) demonstrated practical speedups from sparse backpropagation of Adam (Kingma & Ba, 2014) on commodity hardware, and Yang et al. (2024) showed that block-structured sparsity yields consistent wall-clock speedups in fine-tuning. In contrast to methods that exploit a fixed sparse structure, MaskLLM (Fang et al., 2024) learns $n : m$ masked blocks end-to-end via Gumbel-Softmax, trading the simplicity of one-shot masking for greater flexibility.

Representational Similarity and Interpretability. Centered Kernel Alignment (Kornblith et al., 2019) has become a standard tool for comparing neural network representations across architectures and training runs. Linear probing (Alain & Bengio, 2016; Tenney et al., 2019) tests what task-relevant information is encoded at specific layers. Merchant et al. (2020) used these tools to study how fine-tuning reshapes representations in BERT, finding that fine-tuning primarily affects upper layers while preserving lower-layer linguistic structure.

3 METHODS

3.1 SUBNETWORK EXTRACTION

We study procedures for identifying task-relevant subnetworks in large language models. Let θ denote the full parameter vector, let $\Delta\theta$ denote the update produced by dense fine-tuning, and let $\mathcal{M} \in \{0,1\}^{|\theta|}$ be a binary mask of the same dimensionality, where $\mathcal{M}_i = 1$ updates parameter θ_i and $\mathcal{M}_i = 0$ retains it. The resulting subnetwork parameters are $\theta_{\mathcal{M}} = \theta + \mathcal{M} \odot \Delta\theta$, where $\odot$ is the elementwise product. Our goal is to construct $\mathcal{M}$ at a target sparsity level ρ, defined as the fraction of parameters excluded from the fine-tuning update. (Han et al., 2015; Frankle & Carbin, 2019; Burkholz et al., 2022; Mukherjee et al., 2025; Lee et al., 2019; Wang et al., 2020).

Our framework has two stages. First, a scoring function $s(\theta)$ assigns each parameter a saliency value $s(\theta_i)$, reflecting the importance of θ_i to the preference optimization objective. Second, a mask construction operator uses the saliency values $\{s(\theta_i)\}_{i=1}^{|\theta|}$ to produce a binary mask $\mathcal{M}_i = \mathbf{1}[s(\theta_i) \geq \tau_\rho]$, where τ_ρ is the score threshold updating exactly the top $(1-\rho)|\theta|$ parameters by $s(\theta_i)$.

3.1.1 SCORING FUNCTIONS

Random (Baseline). The random scoring function assigns scores by sampling uniformly from $\mathcal{U}(0, |\theta|)$, serving as a sanity check baseline against which an effective scoring function should perform strictly better. A uniform distribution is the natural uninformative baseline given the permutation invariance of transformer weights (Ainsworth et al., 2023; Hecht-Nielsen, 1990; Rinaldi et al., 2025), which precludes any persistent index-wise local or global structure in weight space.

Warm-Start Magnitude. Warm-start magnitude (Panda et al., 2024) scoring function measures the absolute displacement of each parameter after k steps of dense fine-tuning, and reads,

$$s(\theta_i) = |\theta_i^k - \theta_i^0| \quad \text{with } k \ll T, \tag{1}$$

where T is the number of training steps and θ^k denotes the parameters after k steps of dense fine-tuning from the pretrained initialization θ^0. As shown in Mukherjee et al. (2025), task-relevant subnetworks solidify early in training. Parameters that move more during the warm-start phase are treated as more sensitive to the preference optimization objective. The warm-start approach is computationally cheaper than a full fine-tuning run and can identify task-relevant structure early in training.

Oracle (Upper Bound). The Oracle is the special case of warm-start magnitude with $k = T$, measuring parameter displacement over a complete dense fine-tuning run, and is given by,

$$s(\theta_i) = |\theta_i^k - \theta_i^0| \quad \text{with } k = T. \tag{2}$$

The Oracle requires a dense fine-tuning run and is computationally impractical as a scoring method. It provides an upper bound on subnetwork quality, capturing the performance achievable if the parameters most relevant to the preference optimization objective were identified before fine-tuning.

3.1.2 MASK CONSTRUCTION

Given saliency values $\{s(\theta_i)\}_{i=1}^{|\theta|}$, we construct a binary mask via a hybrid local-global strategy. A *floor pass* first guarantees a minimum retention fraction r per layer, preventing low-scoring layers

from being entirely excluded. A *global pass* then allocates the remaining budget to the highest-scoring parameters across all layers jointly, with τ_ρ estimated via histogram chunking to avoid materializing the full score vector in GPU memory. Full details are in Appendix 5.

3.2 Subnetwork Properties

3.2.1 Loss Retention

We bound the loss gap between dense fine-tuning and sparse subnetwork fine-tuning, for an arbitrary scoring function for an arbitrary scoring function $s(\theta)$. Let $\mathcal{L}(\theta)$ denote the preference optimization loss, evaluated at the pretrained model parameters θ, and let $H = \nabla^2\mathcal{L}(\theta)$ denote the Hessian of $\mathcal{L}$ with respect to θ. A second-order Taylor approximation of the loss gap shows that withholding the fine-tuning update to parameter θ_i contributes the diagonal term $\frac{H_{ii}}{2}\theta_i^2$ to the loss gap motivating scoring functions of the form $s(\theta_i) = \frac{\kappa(\theta_i)}{2}\theta_i^2$, where $\kappa(\theta_i) \geq 0$ approximates the diagonal curvature H_{ii}. Writing $s(\theta_i)$ in terms of $\kappa(\theta_i)$ reveals that any scoring function implicitly defines a diagonal approximation $D_S = \mathrm{diag}(\kappa(\theta))$ to H. The quality of this approximation controls the tightness of the loss-retention bound (full derivation in Appendix D.1).

Theorem 1 (Loss Retention)**.** *Assume the loss $\mathcal{L}$ is twice continuously differentiable and β-smooth, and let $\|\nabla\mathcal{L}(\theta)\|_\infty \leq \epsilon$. Let $C = \max_i \sum_{j\neq i} |H_{ij}|$. For a mask $\mathcal{M}$ at sparsity ρ with withheld-update vector $\delta = (1-\mathcal{M}) \odot \Delta\theta$:*

$$\mathcal{L}(\theta_{\mathcal{M}}) - \mathcal{L}(\theta + \Delta\theta) \leq \sum_{i\notin\mathcal{M}} s(\theta_i) + \frac{C}{2}\|\delta\|_1^2 + \epsilon\|\delta\|_1 \tag{3}$$

The bound is small when $s(\theta)$ assigns low scores to pruned weights and $\|\delta\|_1$ is small. C is small in the DPO and GRPO settings, as evidenced by the success of diagonal Hessian PEFT and preconditioning methods, indicating that this is a reasonable approximation of the true block-diagonal tendency of LLM Hessians (Zhao et al., 2025; Deng et al., 2026).

3.2.2 Optimal Scoring Function and Certifiability

Setting $\kappa(\theta_i) = H_{ii}$ minimizes the bound in Theorem 1 over all diagonal curvature choices. The resulting optimal scoring function $s^*(\theta_i, B)$ depends on the pruned weight norm $B = \|\delta\|_1$, yielding a fixed-point problem whose solution B^* exists and is unique under a mild magnitude-curvature alignment condition (Appendix D.2). H_{ii} is approximated by the diagonal Fisher $F_i = \mathbb{E}[(\partial_{\theta_i}\mathcal{L})^2]$, tractable from gradients alone.

Before committing to a scoring function, one can certify a priori whether it will recover the same mask as s^*, without running a full fine-tuning experiment.

Theorem 2 (Certifiability)**.** *A scoring function $s(\theta)$ selects the same mask as $s^*(\theta_i, B^*)$ at sparsity ρ if for all i:*

$$|s(\theta_i) - s^*(\theta_i, B^*)| < m_i(s) \tag{4}$$

where $m_i(s) = |s(\theta_i) - \tau_\rho(s)|$ is the margin of parameter i from the mask-selection threshold $\tau_\rho(s)$. At a local minimum, $F_i = H_{ii}$ exactly, making the condition exact. When condition (4) *is violated for weight i, the deviation of s from s^* is large enough to potentially misclassify whether θ_i should receive its update or remain frozen.*

We apply this condition empirically in Section 4.4 to validate warm-start magnitude scoring against the Oracle.

3.2.3 Sparse Training Pipeline

Optimizer steps are memory-bandwidth bound, so updating dense moment buffers for parameters frozen by $\mathcal{M}$ wastes substantial memory traffic proportional to the inactive fraction. We therefore implement a BSR-aware AdamW optimizer that restricts parameter updates, gradient accumulation, and moment buffer writes to active locations only — via gather–update–scatter in the element regime, and block-aware tensor-core-friendly kernels in the block regime. Since dense AdamW touches every parameter each step, all memory traffic and time scale as $\mathcal{O}(P)$; BSR-AdamW restricts to the $A = 1-\rho$ active fraction, reducing both to $\mathcal{O}((1-\rho)P)$ (see Appendix B.4). Gradient clipping norms are computed over active indices only, so clipping cost scales with the active fraction rather than the dense parameter count (Yang et al., 2024; Ainsworth et al., 2023; Rinaldi et al., 2025; Adnan et al., 2025; Fang et al., 2024).

3.3 Generalizability and Interpretability

3.3.1 Structural Overlap

We measure the degree to which two subnetworks select the same parameters using Jaccard similarity. Since our mask construction adopts a hybrid local-global strategy (Section 3.1.2), we report two complementary quantities. At the aggregate level, we use the global and layer-wise Jaccard similarities, defined as:

$$J_\ell(\mathcal{M}_A, \mathcal{M}_B) = \frac{|\mathcal{M}_A^{(\ell)} \wedge \mathcal{M}_B^{(\ell)}|}{|\mathcal{M}_A^{(\ell)} \vee \mathcal{M}_B^{(\ell)}|} \in [0,1], \qquad J_{\text{global}} = \frac{\sum_\ell |\mathcal{M}_A^{(\ell)} \wedge \mathcal{M}_B^{(\ell)}|}{\sum_\ell |\mathcal{M}_A^{(\ell)} \vee \mathcal{M}_B^{(\ell)}|} \in [0,1]. \tag{5}$$

These scores reveal which layers contribute most to structural divergence between subnetworks. J_{global} differs from the unweighted per-layer average $\bar{J} = \frac{1}{L}\sum_\ell J_\ell$, where L is the number of layers, which treats all layers equally regardless of their size. To provide a reference for chance-level overlap, we derive random-mask baselines for both metrics in Appendix A.4.4.

3.3.2 Representational Similarity

Masks with low parameter overlap can produce functionally similar subnetworks, as different sets of parameters can encode similar representations (Kornblith et al., 2019). We measure representational similarity beyond parameter overlap using linear Centered Kernel Alignment (CKA) (Kornblith et al., 2019), which is invariant to isotropic rescaling and orthogonal transformation of the representation basis.

For each sample i, we extract mean-pooled hidden states at layer ℓ weighted by the attention mask, yielding activation matrices $X_A^{(\ell)}$ and $X_B^{(\ell)}$. We define the centered Gram matrices $K_A^{(\ell)} = HX_A^{(\ell)}X_A^{(\ell)\top}H$ and $K_B^{(\ell)} = HX_B^{(\ell)}X_B^{(\ell)\top}H$, where $H = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top$ is the centering matrix. Linear CKA at layer ℓ is given by,

$$\text{CKA}_\ell(X_A, X_B) = \frac{\langle K_A^{(\ell)}, K_B^{(\ell)} \rangle_F}{\sqrt{\langle K_A^{(\ell)}, K_A^{(\ell)} \rangle_F \langle K_B^{(\ell)}, K_B^{(\ell)} \rangle_F}} \in [0,1], \tag{6}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product. CKA is bounded in $[0,1]$, where higher values indicate greater representational similarity. We apply CKA layer-wise to identify where representations diverge across the network and in aggregate to obtain an overall similarity score, both reported in Section 4. To provide a reference for chance-level representational similarity, random-mask baselines for CKA are derived in Appendix A.4.4.

3.3.3 Functional Capability

We test what task-relevant capabilities each subnetwork preserves or loses at each layer. For a given layer ℓ, we extract the activation vector $\mathbf{h}_\ell$ and train a linear classifier to predict specific task-relevant properties. We focus on four categories spanning distinct competencies: syntax (BLiMP, Warstadt et al. 2020), semantics (SST-2, Socher et al. 2013), factual knowledge (AG News, Zhang et al. 2015), and mathematics (GSM8K, Cobbe et al. 2021). We use a pairwise ranking probe (Bradley & Terry, 1952) that trains a direction w_ℓ in representation space maximizing the margin between chosen (h^+) and rejected (h^-) activations, via $\mathcal{L}_{\text{pair}}(w_\ell) = -\sum_{(h^+,h^-)} \log \sigma\left(w_\ell^\top (h^+ - h^-)\right)$, giving a direct readout of whether the subnetwork preserves the preference optimization signal.

For each mask $\mathcal{M}$ we obtain a (property $\times$ layer) accuracy matrix $\mathcal{A}_\mathcal{M}[\,p, \ell\,]$. We report the absolute heatmap $\mathcal{A}_\mathcal{M}$ alongside the dense baseline $\mathcal{A}_{\text{baseline}}$, the delta heatmap $\Delta_\mathcal{M} = \mathcal{A}_\mathcal{M} - \mathcal{A}_{\text{baseline}}$ which separates capability lost by masking from capability the base model never had, and per-property averages across sampled layers in Section 4.

4 Experiments

We investigate our approach across several dimensions: sparse DPO and GRPO subnetworks recover dense training dynamics at high sparsity and preserve downstream metrics; that DPO and GRPO produce distinct subnetworks by Jaccard, CKA, and linear probing, regardless of dataset; and exploiting subnetwork sparsity yields consistent and significant optimizer-step speedups. We show that our findings hold across model families and scales supporting the view that the learning

objective, more than dataset content, shapes preference-optimization subnetworks under common hyperparameter choices. Full experimental details are provided in Appendix E with supplemental results in Appendix A.

4.1 Sparse Subnetworks Recover Dense DPO Training Dynamics

At 97.5% sparsity, warm-start magnitude selection recovers the training dynamics of dense DPO and GRPO, demonstrating that successful sparse preference optimization depends on *which* parameters are updated, not simply on how many. Figure 1 shows the training loss for DPO on Light-R1 and Tülu3. Oracle and warm-start magnitude masks closely track dense training throughout, whereas random masks at the same sparsity fail to reduce the loss. Since every method updates only 2.5% of the model parameters, the observed differences in training dynamics can be attributed entirely to which parameters are updated.

The same relative ordering of Oracle, warm-start, and random masks is also reflected in downstream preference classification accuracy (Appendix A.1), providing an additional evaluation beyond the training loss. This trend is further supported by the linear-probe analysis of Section 4.3, where Oracle and warm-start subnetworks preserve task-relevant capabilities that are largely absent in random subnetworks.

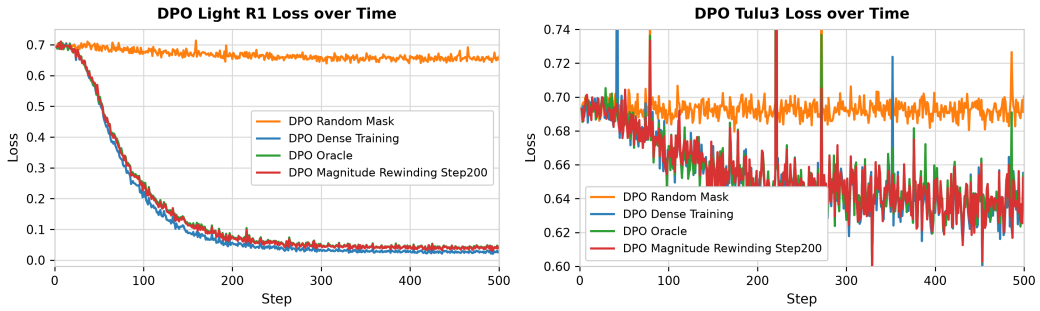


Figure 1: Loss curves for DPO training of `Llama-3.1-8B-Instruct` for 500 steps on Light-R1 (left) and Tülu3 (right) at 97.5% sparsity. Oracle and warm-start magnitude masks closely recover dense training dynamics, while random masks show little to no learning (more details in Appendix A.1). Oracle masks from cross-task settings (Open-R1 GRPO) are included to preview the generalizability results of Section 4.2.

Table 1 reports downstream performance on the Tülu3 evaluation suite. Sparse models preserve dense training performance across all seven benchmarks, demonstrating that restricting updates to 2.5% of parameters does not degrade model performance on downstream tasks relative to full-parameter training. Additional results showing the training choose/reject classification accuracies can be found in Appendix A.1.

Table 1: Downstream benchmark results for `Llama-3.1-8B-Instruct` after 500 steps of preference optimization. All sparse arms use $\rho = 97.5\%$. Metrics are accuracy / EM / pass@1 (higher is better). These evaluations test whether preference optimization under a sparse mask preserves the base model's capabilities, not whether one mask is better than another. Random masks preserves performance trivially due to failing to train at all (Figure 1). We report only a single run per arm.

Method	**MMLU**	**MATH**	**GSM8K**	**HumanEval**	**MBPP**	**IFEval**	**GPQA-D**
Base (no training)	0.687	0.403	0.820	0.683	0.584	0.739	0.263
Dense DPO (Tülu3)	0.687	0.407	0.818	0.671	0.586	0.749	0.273
Oracle DPO (Tülu3)	0.687	0.409	0.817	0.671	0.584	0.749	0.273
Oracle DPO (Light-R1)	0.687	0.407	0.817	0.671	0.584	0.750	0.273
Oracle GRPO (Open-R1 Math)	0.687	0.407	0.816	0.671	0.586	0.741	0.268
Warm-start mag (k=200, Tülu3)	0.687	0.407	0.816	0.671	0.588	0.749	0.268
Random mask	0.687	0.405	0.821	0.683	0.586	0.743	0.263
LoRA (r=64, lr 5×10^{-6})	0.686	0.388	0.831	0.671	0.578	0.763	0.273

4.2 Subnetwork Structure Reflects the Learning Objective

Figure 2 shows layer-wise Jaccard similarity and CKA scores comparing Oracle masks from DPO Light-R1, DPO Tülu3, and GRPO Open-R1. DPO subnetworks trained on Light-R1 and Tülu3 share a global Jaccard similarity of approximately 0.85, indicating that nearly the same parameters are selected despite the datasets covering different domains. Their CKA scores are similarly high, confirming that the two subnetworks produce comparable internal representations. In contrast, DPO and GRPO subnetworks share near-zero Jaccard similarity and substantially lower CKA scores across all layers, despite Light-R1 being derived from Open-R1 — the same dataset used for GRPO training.

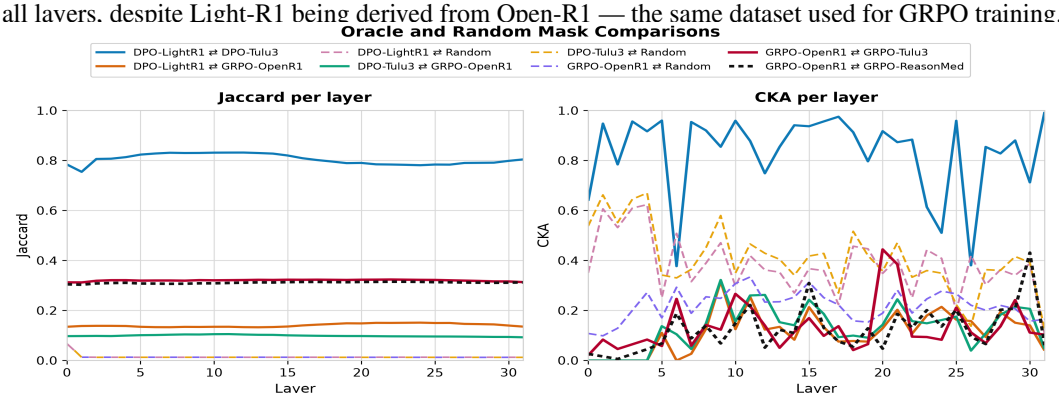


Figure 2: Layer-wise Jaccard similarity (left) and CKA (right) between Oracle mask pairs. DPO Light-R1 and Tülu3 masks share high Jaccard and CKA scores, indicating similar parameter selection and internal representations. Both diverge substantially from the GRPO Open-R1 Oracle mask. Results show that masks produced by GRPO on separate datasets are dissimilar by Jaccard and CKA.

Figure 3 shows that DPO Oracle masks trained on Light-R1 transfer successfully to Tülu3 and vice versa, recovering dense training dynamics under cross-task deployment. GRPO Oracle masks applied to DPO training also partially transfer, though with a larger gap from the in-task Oracle, consistent with the representational divergence observed in Figure 2. Results also show that masks produced by GRPO on separate datasets are dissimilar by Jaccard and CKA.

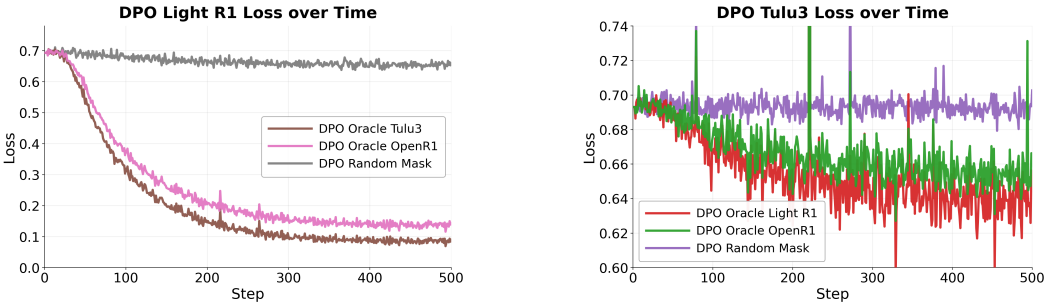


Figure 3: Cross-task transfer loss curves for DPO training of `Llama-3.1-8B-Instruct` for 500 steps on Light-R1 (left) and Tülu3 (right) at 97.5% sparsity using Oracle masks from different tasks and objectives. DPO masks transfer across datasets with minimal performance loss.

Together, Figures 2 and 3 show that subnetwork structure is more strongly determined by the learning objective than by the training data. The divergence between DPO and GRPO subnetworks suggests that these two objectives reorganize the model's parameter landscape in fundamentally different ways, a finding we characterize further through linear probing in Section 4.3.

4.3 Subnetworks Have Interpretable Structure

We test how different subnetworks preserve or lose semantic capabilities using layer-wise paired linear probes (see Section 4.3 and Appendix A.4.3). Specifically, we examine the change in linear probe classification accuracy across syntax, semantics, factual knowledge, and mathematics training on differences between the chosen (h^+) and rejected (h^-) hidden-state representations.

Figure 4 shows the results at $\rho = 97.5\%$ of masks produced at varied k from DPO on Light-R1, and the Oracle mask from GRPO on Open-R1. We find that the capabilities preserved and lost by the two objectives are consistently and significantly distinct across these classes, despite the extraordinary

similarity of the datasets in question. Additional detail of layer-wise probe results on Light-R1 and Tülu3 are provided in Appendix A.4.2.

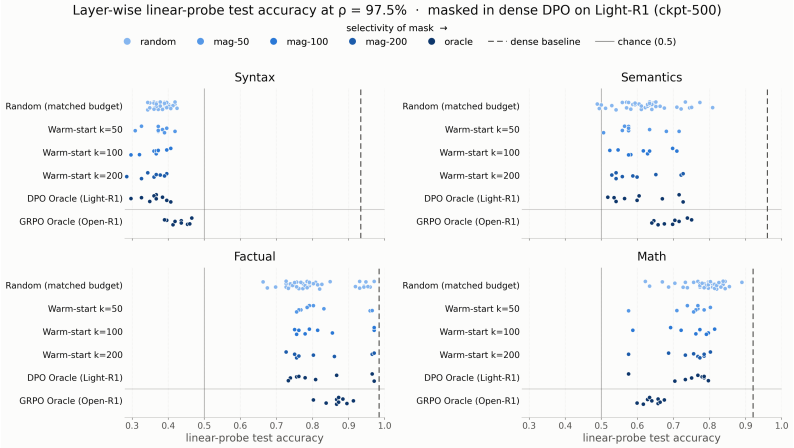


Figure 4: Linear probing results at $\rho = 97.5\%$ of masks produced at varied k from DPO on Light-R1, and the Oracle mask from GRPO on Open-R1. Results shown are from `Llama-3.1-8B-Instruct`.

4.4 Certifiability of Magnitude Rewinding Scoring

To examine the certification margins developed in methods subsections Sec 3.2.1, 3.1.1, and 3.2.2, we measure results for Equation 2 by proxy. Our analysis shows that as k increases, margins w.r.t the Oracle decrease, as expected. We measure the quantities,

$$\delta_i(\theta) = |s_{\text{warm start}}(\theta_i^k, \theta_i^0) - s_{\text{Oracle}}(\theta_i^T, \theta_i^0)| \text{ and } m_i(s) = |s(\theta_i) - \tau_\rho(s)|, \tag{7}$$

for several intermediate step counts, $k = 50, 100, 150, 200$, and a random mask as a baseline. We perform these measurements on masks for DPO training on Light-R1, shown in Figure 5. However, computing the exact global τ_ρ is prohibitively memory intensive. To address this, we leverage the hybrid global selection algorithm used in mask construction, detailed in Algorithm 5 and Algorithm 6, to find an estimate of the true global value, $\hat{\tau}_\rho$.

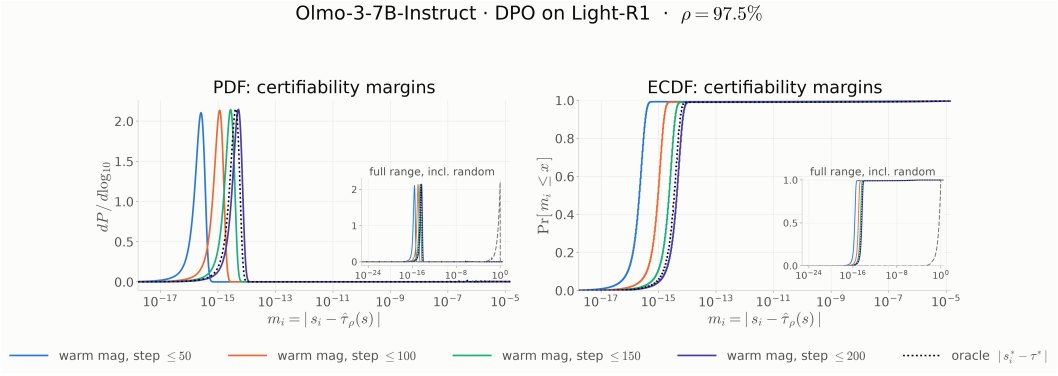


Figure 5: Certifiability margins $m_i(s) = |s_i - \hat{\tau}_\rho(s)|$ at $\rho = 97.5\%$ for warm-magnitude scores $s(\theta_i^{\leq k})$ (see details in Section 3.1.1) at $k \in 50, 100, 150, 200$ training steps. The trend as k increases is that the PDF and ECDFs approach those of the Oracle. However, absolute differences from the Oracle are quite low, indicating that the $\hat{\tau}_\rho$ of the warm start magnitude methods is a good estimate of τ_ρ^* and improves as k increases towards T.

We find that in the DPO case, score analysis indicates that even at low trajectory lengths k the warm start rewinding scoring function is a strong estimator for the Oracle mask. However, there is some potential for overshooting τ_ρ^* in some cases. Supplementary results can be found in Appendix D.4.

4.5 SPARSITY AWARE KERNEL TIMINGS

We measure empirical optimizer step speedups in sparse training with the BSR-AdawmW optimizer described in Algorithms 2 and 3 with the hyperparameter set in Table 6 - although only the prior (1D or element-wise) kernel is used in the sparse training in other experiments (Kingma & Ba, 2014).

Table 2: Optimizer-step time on `Llama-3.1-8B-Instruct` (mean ms/step over the middle 80% of steps; first and last 10% discarded for warmup). Speedup columns report `sparse_adamw` relative to each dense baseline; higher is faster. Each sparsity level was benchmarked on a separate NVIDIA H200, so ratios are computed within-row against that device's own `adamw_torch` measurement and absolute times should not be compared across rows. Note that `adamw_8bit` is itself $\sim 1.98\times$ slower than `adamw_torch` in this setting, trading step time for optimizer-state memory; `adamw_torch` is therefore our primary baseline.

	Mean ms/step			**`sparse_adamw` speedup**	
Sparsity ρ	`adamw_torch`	`adamw_8bit`	`sparse_adamw`	vs. `torch`	vs. `8bit`
75.0%	7.139	14.141	6.207	$1.15\times$	$2.28\times$
97.5%	7.132	14.129	4.214	$1.69\times$	$3.35\times$
99.75%	7.118	14.117	2.171	$3.28\times$	$6.50\times$

Results show that across a broad spectrum of sparsity levels ρ, BSR AdamW makes significantly more steps than PyTorch and HuggingFace dense and 8bit implementations do in the same timespan on the same hardware (Paszke et al., 2019; Dettmers et al., 2022; Kingma & Ba, 2014). The first and last 10% of steps are pruned from the calculation to account for warmup times. Each row is independently ran on different NVIDIA H200 GPUs, as such each row's speedup multipliers are defined with respect to it's own measured dense timing for consistency. A more extensive ablation can be found in Appendix B.1's Table 4.

5 CONCLUSION

We have shown that extremely sparse subnetworks exist in preference-optimized LLMs and that their structure is primarily determined by the learning objective rather than the training data. Warm-start magnitude scoring recovers dense DPO training performance at up to 97.5% sparsity, well above the empirical sparsity estimates of Mukherjee et al. (2025). The structural and representational divergence between DPO and GRPO subnetworks, quantified through Jaccard similarity and CKA, suggests that these objectives reorganize the model's parameter landscape in fundamentally different ways even when trained on related data. Linear probing further reveals that the selected parameters embed semantically interpretable distinctions between objectives on extremely similar data.

Several directions remain open for future work. Probes designed to detect preference-specific representations rather than general linguistic competencies would yield stronger interpretability conclusions (Alain & Bengio, 2016). Our sparse training pipeline currently exploits sparsity in the optimizer step but not in the backward pass; a full BSR backward kernel and structured $n : m$ mask alignment would unlock further wall-clock gains, which we elaborate on in Appendix B.3. Our analysis is confined two objectives, and extending the objective-determines-structure finding to other architectures such as Mixtures of Experts and other RL algorithms such as PPO remains open. Finally, characterizing the natural sparsity threshold of different training objectives and adapting mask construction accordingly would broaden the applicability of our approach.

Beyond efficiency, sparse PEFT carries meaningful practical implications. Training only 2.5% of parameters reduces optimizer memory traffic proportionally, lowering the energy cost and hardware requirements for RL alignment at scale. As preference optimization is deployed more widely, methods that achieve dense performance without dense computation become increasingly relevant, and we hope the framework and findings here provide useful grounding for that development. Our anonymized implementation is available here: A-Few-Good-Weights.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, as well as to implement our methods. We have not used generative AI tools for methodological determinations, narrative structuring of the manuscript, or producing and verifying our theoretical framework. Other required AI use reporting fields are not applicable to this work. Additionally, we have used generative AI tools to help with software implementation of our methods and for rapidly familiarizing authors with the relevant research landscape. We have reviewed all AI-assisted software, text, and artifacts. Authors have thoroughly parsed the bibliography to confirm it does not contain hallucinated or low quality citations. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI, and will respond to queries regarding the provenance of artifacts, data, methods, or portions of the manuscript text itself.

REPRODUCIBILITY STATEMENT

To support the core scientific tenet of peer-reproducibility, we provide all software levered in the implementation and execution of this work. We support this by detailing relevant hyperparameters throughout the Appendix, including Appendix E and Appendix B.3.

REFERENCES

Mohammed Adnan, Rohan Jain, Ekansh Sharma, Rahul Krishnan, and Yani Ioannou. Sparse training from random initialization: Aligning lottery ticket masks using weight symmetry. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=BnfJSwtHLu.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016. URL https://arxiv.org/abs/1610.01644.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*, 2019.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3-4):324–345, 1952. doi: 10.1093/biomet/39.3-4.324.

Rebekka Burkholz, Nilanjana Laha, Rajarshi Mukherjee, and Alkis Gotovos. On the existence of universal lottery tickets. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=SYB4WrJql1n.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained BERT networks. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2007.12223.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03741.

Karl Cobbe, Vineet Kosaraju, Mo Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168, 2021. URL https://api.semanticscholar.org/CorpusID:239998651.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Shenyang Deng, Zhuoli Ouyang, Tianyu Pang, Zihang Liu, Ruochen Jin, Shuhua Yu, and Yaoqing Yang. RMNP: Row-momentum normalized preconditioning for scalable matrix-based optimization. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=BC5aRMdRwp.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv:2305.14314*, 2023.

Abhimanyu Dubey et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. MaskLLM: Learnable semi-structured sparsity for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Llu9nJal7b.

Andres Fernandez, Frank Schneider, Maren Mahsereci, and Philipp Hennig. Connecting parameter magnitudes and hessian eigenspaces at scale using sketched methods. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=yGGoOVpBVP.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *ArXiv*, abs/1903.01611, 2019. URL https://api.semanticscholar.org/CorpusID:67877171.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023. URL https://proceedings.mlr.press/v202/frantar23a.html.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015. URL https://arxiv.org/abs/1506.02626.

Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In Rolf ECKMILLER (ed.), *Advanced Neural Computers*, pp. 129–135. North-Holland, Amsterdam, 1990. ISBN 978-0-444-88400-8. doi: https://doi.org/10.1016/B978-0-444-88400-8.50019-4. URL https://www.sciencedirect.com/science/article/pii/B9780444884008500194.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nuzorkLshJ.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/kornblith19a.html.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1810.02340.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxkOlSYDH.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. What happens to BERT embeddings during fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 33–44. Association for Computational Linguistics, 2020.

Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tür, and Hao Peng. Reinforcement learning finetunes small subnetworks in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=0NdS4xCngO.

Mahdi Nikdan, Tommaso Pegolotti, Eugenia Iofinova, Eldar Kurtic, and Dan Alistarh. SparseProp: Efficient sparse backpropagation for faster training of neural networks. In *International Conference on Machine Learning*, pp. 26215–26227. PMLR, 2023.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Ashwinee Panda, Berivan Isik, Xiangyu Qi, Sanmi Koyejo, Tsachy Weissman, and Prateek Mittal. Lottery ticket adaptation: Mitigating destructive interference in LLMs, 2024. URL https://openreview.net/forum?id=qOqCXEXsX4.

A. Paszke, S. Gross, F. Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 55502–55523, 2023. URL https://arxiv.org/abs/2305.18290.

Filippo Rinaldi, Giacomo Capitani, Lorenzo Bonicelli, Donato Crisostomi, Federico Bolelli, ELISA FICARRA, Emanuele Rodolà, Simone Calderara, and Angelo Porrello. Update your transformer to the latest release: Re-basin of task vectors. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=sHvImzN9pL.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard (eds.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170/.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023. URL https://arxiv.org/abs/2306.11695.

Yu Sun, Xingyu Qian, Weiwen Xu, Hao Zhang, Chenghao Xiao, Long Li, Yu Rong, Wenbing Huang, Qifeng Bai, and Tingyang Xu. Reasonmed: A 370k multi-agent generated dataset for advancing medical reasoning, 2025. URL https://arxiv.org/abs/2506.09513.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://arxiv.org/abs/1905.05950.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019. URL https://doi.org/10.1145/3315508.3329973.

Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. *arXiv preprint arXiv:2105.01648*, 2021. URL https://arxiv.org/abs/2105.01648.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgsACVKPH.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://aclanthology.org/2020.tacl-1.25/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Xinyu Yang, Jixuan Leng, Geyang Guo, Jiawei Zhao, Ryumei Nakada, Linjun Zhang, Huaxiu Yao, and Beidi Chen. S^{2}FT: Efficient, scalable and generalizable LLM fine-tuning by structured sparsity. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=lEUle8S4xQ.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1906.02768.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.

Yanjun Zhao, Sizhe Dang, Haishan Ye, Guang Dai, Yi Qian, and Ivor Tsang. Second-order fine-tuning without pain for LLMs: A hessian informed zeroth-order optimizer. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bEqI61iBue.

A Extended Experimental Evaluations

A.1 Generalization Across Model Families

A.1.1 Llama-3.1-8B-Instruct on Light-R1

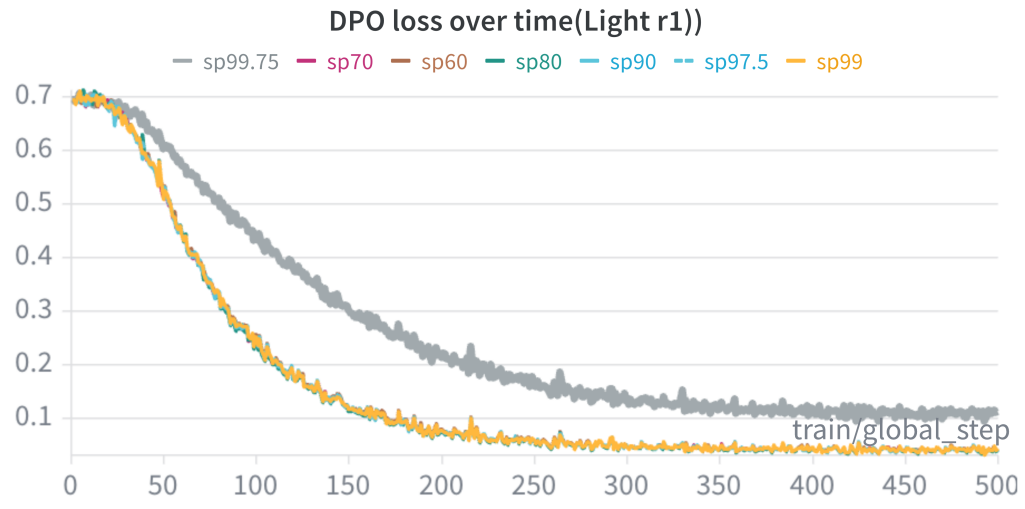


Figure 6: Sweep over several Oracle mask sparsities, $\rho \in \{60, 70, 80, 90, 95, 97.5, 99, 99.75\}$.

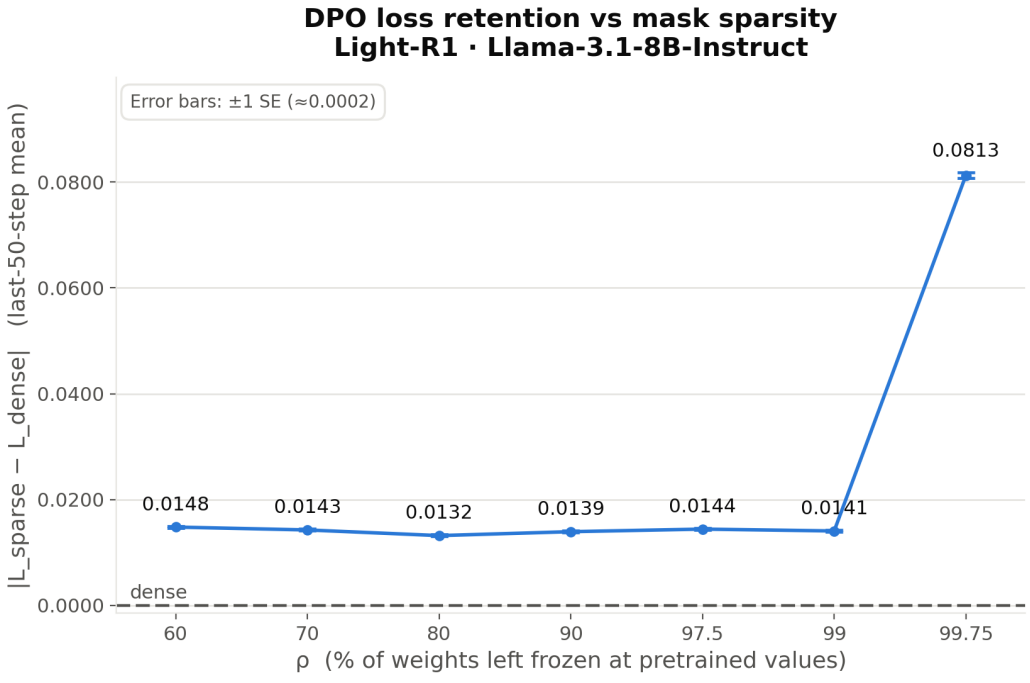


Figure 7: $|R_{sparse} - R_{dense}|$ for several Oracle mask sparsities, $\rho \in \{60, 70, 80, 90, 95, 97.5, 99, 99.75\}$. Loss gaps gaps are measured by the mean and variance of the DPO training loss in the terminal 50 training steps of single training runs.

A.1.2 QWEN3-8B ON LIGHT-R1

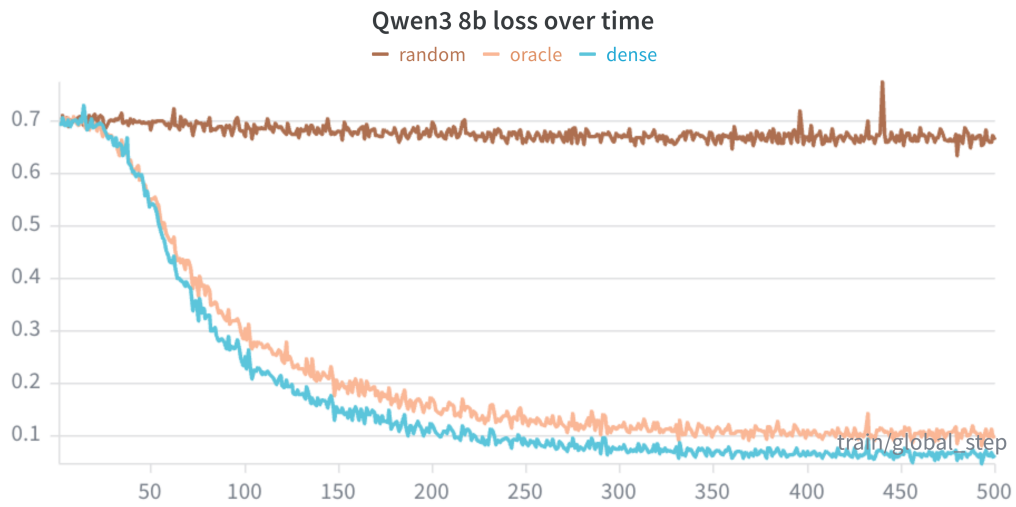


Figure 8: `Qwen3-8B` DPO training loss curves on Light-R1, with random and Oracle masks at $\rho = 97.5\%$

A.1.3 OLMO-3-7B-INSTRUCT ON LIGHT-R1

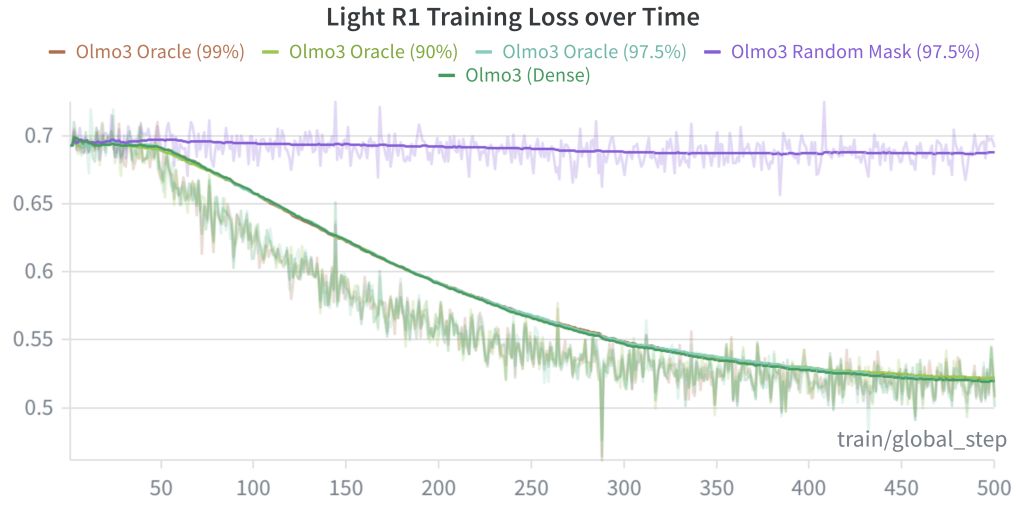


Figure 9: Sweep over several Oracle mask sparsities, $\rho \in \{90, 97.5, 99\}$.

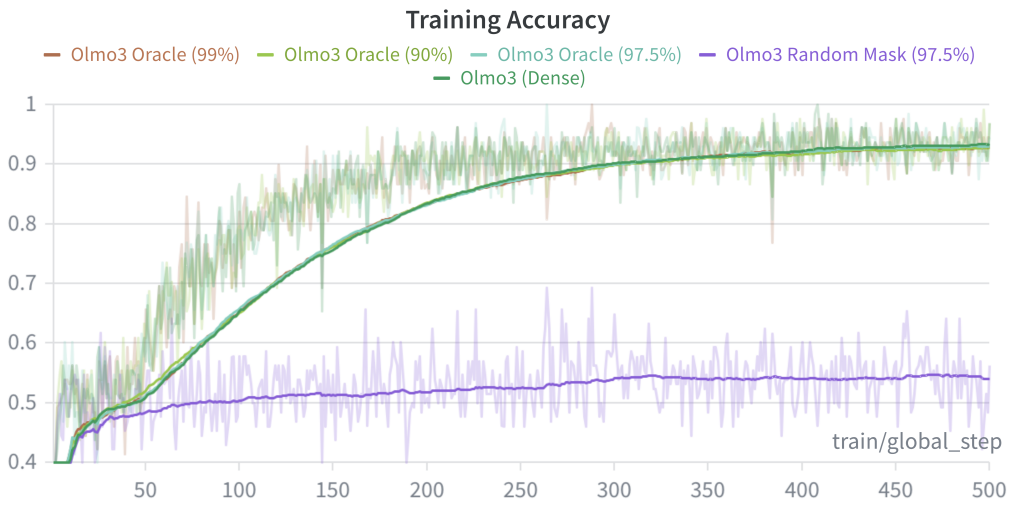


Figure 10: Sweep over several Oracle mask sparsities, $\rho \in \{90, 97.5, 99\}$.

A.1.4 QWEN3-32B ON LIGHT-R1

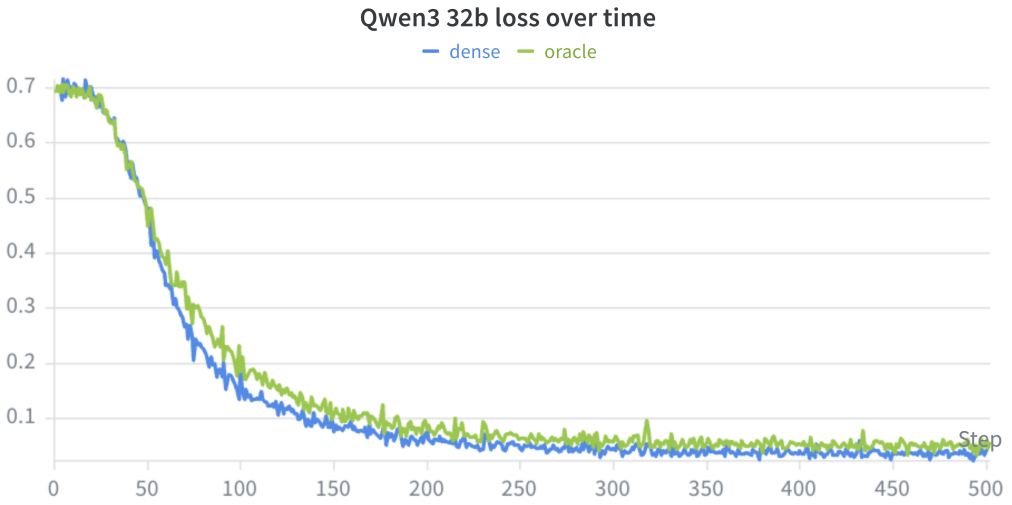


Figure 11: `Qwen3-32B` DPO training loss curves on Light-R1, with Oracle mask at $\rho = 97.5\%$

A.2 SPARSE FULL-RANK TRAINING VS LORA

Existing parameter-efficient fine-tuning (PEFT) methods are typically evaluated along two dimensions: adaptation quality and computational efficiency. In this work, we argue that a third dimension, interpretability of the optimization process, is equally important for understanding how preference optimization modifies pretrained language models. While downstream accuracy evaluates the outcome of fine-tuning, interpretability of the optimization process reveals the mechanism by which that outcome is achieved. Understanding these optimization mechanisms can, in turn, inform the design of future fine-tuning algorithms by identifying which parameters and representations are most important for preference optimization.

Sparse full-rank training and LoRA are both PEFT methods that optimize only a small fraction of the model parameters, but they make fundamentally different assumptions about how preference optimization updates should be represented. Like LoRA, our framework seeks an efficient parameterization of preference optimization. Rather than restricting updates to a low-rank subspace, we preserve the model's original parameter space by directly optimizing a sparse subset of the pretrained weights. This not only produces an efficient sparse parameterization, but also enables parameter-level interpretability of the learned subnetworks. The accompanying theoretical framework also explains why these subnetworks recover dense preference optimization. Table 3 summarizes these complementary perspectives.

Table 3: Comparison between sparse full-rank training and LoRA. The two approaches make different structural assumptions and emphasize complementary properties of preference optimization.

Criterion	**Ours**	**LoRA** ($r = 64$)
Structural Assumptions		
Optimizes pretrained parameters directly	✓	✗
Update parameterization	**Sparse**	Low-rank
Median stable rank	**178**	≤ 64
Downstream Adaptation		
Held-out preference accuracy	0.542	**0.618**
Reference-policy drift	**Lower**	Higher
Interpretability		
Enables interpretable parameter-level analyses	✓	✗

LoRA achieves higher held-out preference accuracy in our experimental setting, demonstrating the effectiveness of low-rank adaptation for downstream fine-tuning. Downstream accuracy alone does not explain how preference optimization modifies a pretrained language model or which pretrained parameters are responsible for the learned behavior. Understanding preference optimization also requires identifying which pretrained parameters are consistently reused across learning objectives, datasets, and functional capabilities. Identifying parameter-level patterns provides scientific insight into transfer, objective-specific specialization, continual learning, and the mechanistic interpretability of preference optimization.

We seek an efficient sparse parameterization while preserving direct optimization of the pretrained model parameters, enabling parameter-level interpretability of the optimization process. In contrast, LoRA performs optimization through auxiliary low-rank adapter parameters rather than the pretrained weights themselves. While the resulting update can be projected back into the original parameter space after training, the optimization trajectory is defined over the adapter parameters rather than the pretrained weights. This obscures the relationship between optimization dynamics and the pretrained model parameters, making it difficult to directly attribute representational or functional changes to specific pretrained weights.

We measured the stable rank of preference optimization updates across 48 weight matrices to better understand the structure of the learned updates. Across three independent runs, the median stable rank of the per-matrix updates ranges from 132 to 229, with 30–42 of the 48 matrices exceeding the maximum stable rank representable by LoRA with $r = 64$.These observations indicate that preference optimization updates are substantially higher-rank than those representable by the LoRA parameterization, supporting the use of sparse full-rank updates for characterizing preference optimization.

Section 4 illustrates the analyses enabled by sparse full-rank training. Because optimization is performed directly over the pretrained model parameters, we can relate parameter-level changes to representational and functional behavior. This enables us to analyze parameter overlap using Jaccard similarity (Section 4.2), compare learned representations using CKA (Section 4.2), and characterize functional specialization through linear probes (Section 4.3). Together, these analyses provide a unified characterization of how preference optimization modifies pretrained language models by connecting optimization dynamics with representational and functional changes.

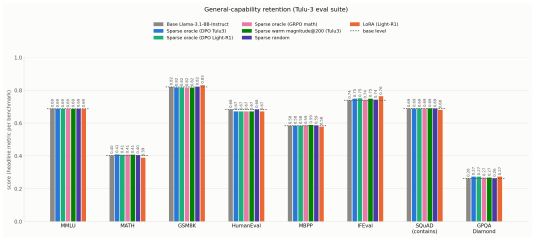


Figure 12: Benchmark performance scores of dense, sparse masked, and LoRA training on Light-R1. Results indicate no meaningful performance gap on these downstream applications between these methods. Also shown in Table 1

A.3 Additional GRPO Experiments

A.3.1 GRPO Oracle masking

To further investigate the extension of our magnitude and Oracle masking methods to the GRPO objective, a thorough sweep of sparse Oracle training runs was run across sparsity levels ρ. Specifically, $\rho \in \{50, 60, 70, 80, 90, 95, 97.5, 99, 99.75\}$. The results of this sweep are shown below in Figure 13, clearly indicating that Oracle masking is effective in application to GRPO, and that some sparsity levels actually achieve superior results against the dense baseline. This stands in partial contrast to the measurement of Mukherjee et al. (2025), whose closest point of comparison is `deepseek-math-7B-instruct`, which showed 68.5% sparsity in GRPO training. Analysis continued in Appendix A.4.1.

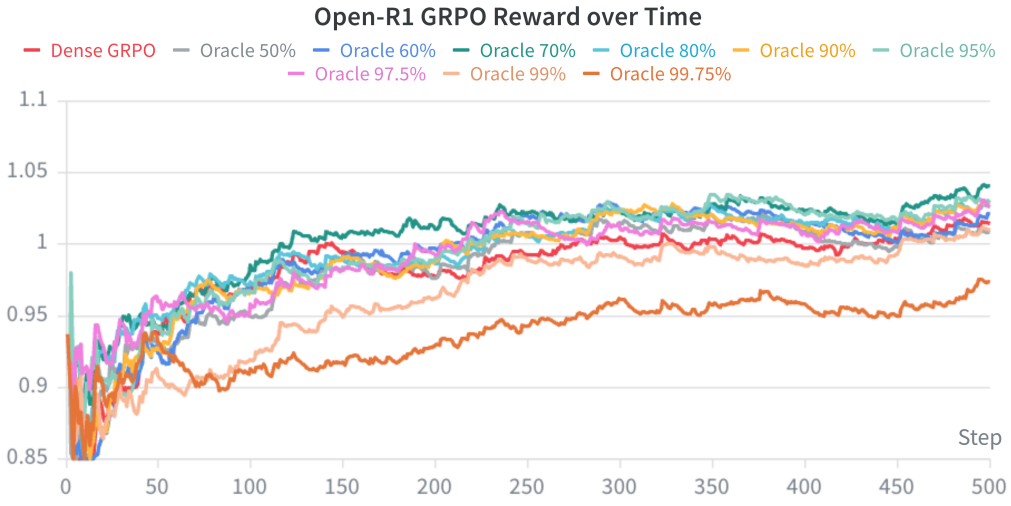


Figure 13: Aggregate GRPO rewards over training of Dense and Oracle masked training at $\rho \in \{0, 60, 70, 80, 90, 95, 97.5, 99, 99.75\}$ in `Llama-3.1-8B-Instruct` applied to CoT style GRPO training on Open-R1 (Dubey et al., 2024; DeepSeek-AI et al., 2025; Wei et al., 2022).

A.3.2 Cross-Task GRPO

To examine the capacity of GRPO Oracle masks to transfer across tasks, we trained `Llama-3.1-8B-Instruct` on Reason-Med and examined the capability of the resulting Oracle mask to transfer to Open-R1 (Sun et al., 2025).

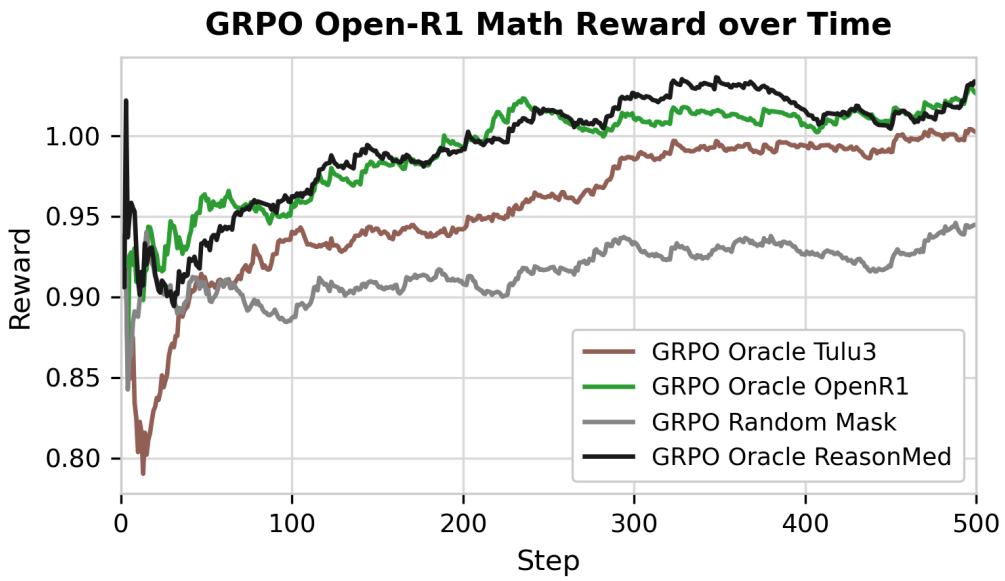


Figure 14: Results of cross-dataset transfer of GRPO Oracle masks at $\rho = 97.5\%$ indicate that GRPO masks transfer well compared to masks from DPO datasets and random masks.

A.4 Sparsity Ablations

A.4.1 GRPO

To further investigate this phenomena, we examine the aggregate reward gaps across sparsity levels, shown in Figure 15. Reward gaps are measured by the mean and variance of the aggregate GRPO reward in the terminal 50 training steps. We find that the reward gap is maximized by $\rho = 70\%$, roughly in-line with the measurement of Mukherjee et al. (2025).

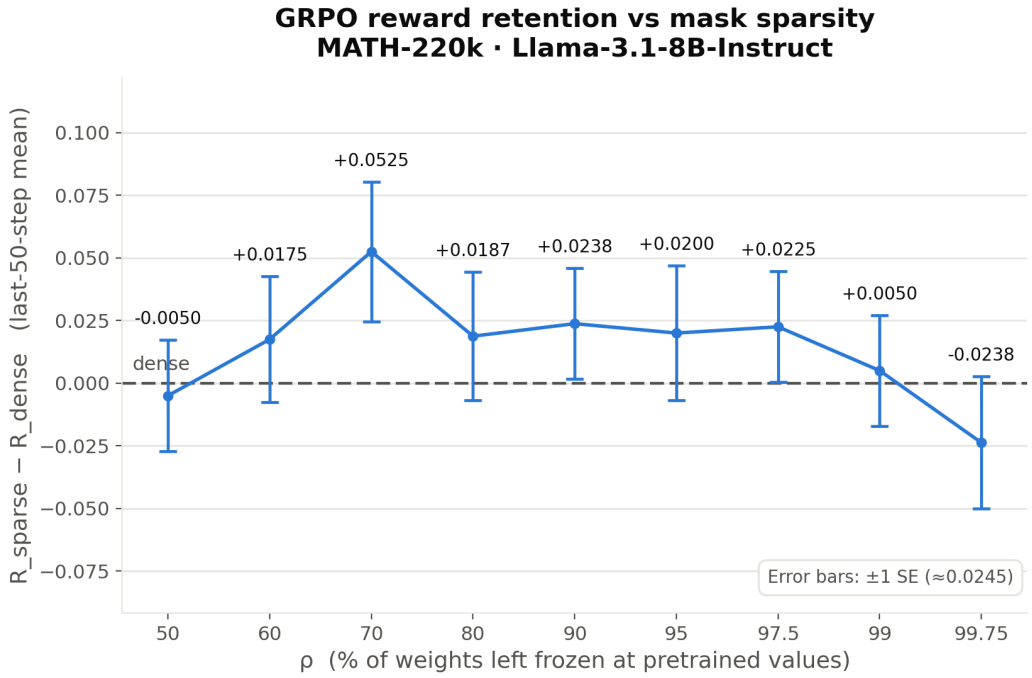


Figure 15: $R_{sparse} - R_{dense}$ for several Oracle mask sparsities, $\rho \in \{50, 60, 70, 80, 90, 95, 97.5, 99, 99.75\}$. Reward gaps are measured by the mean and variance of the aggregate GRPO reward in the terminal 50 training steps of single training runs.

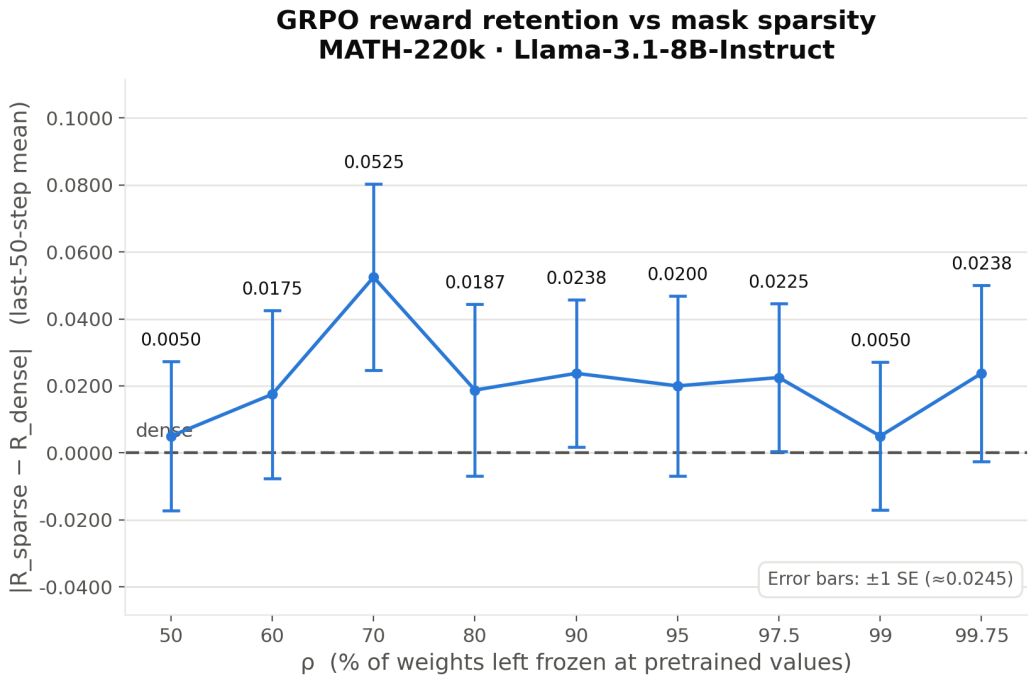


Figure 16: $|R_{sparse} - R_{dense}|$ for several Oracle mask sparsities, $\rho \in \{50, 60, 70, 80, 90, 95, 97.5, 99, 99.75\}$. Reward gaps are measured by the mean and variance of the aggregate GRPO reward in the terminal 50 training steps of single training runs.

Two expected behaviors emerge:

1. The dense rewards exceed the sparse rewards at high sparsity ($\rho \to 100\%$), indicating that there is no longer adequate masked model capacity for learning.

2. The sparse rewards match the dense rewards at low sparsity ($\rho \leq 50\%$), indicating that the masked model has approximately equal capacity as the dense model.

This behavior connects our loss retention analysis to the empirical measurements of GRPO sparsity by Mukherjee et al. (2025) by showing that the retention gap is maximized at roughly the natural sparsity measured.

The precise mechanism as to *why* sparse masked Oracles outperform the dense baseline, and why that gap is largest around $\rho \approx 70\%$, remain promising directions for more deeply investigating the structure of GRPO gradients and weight updates in connection with model and dataset attributes.

A.4.2 Linear Probes

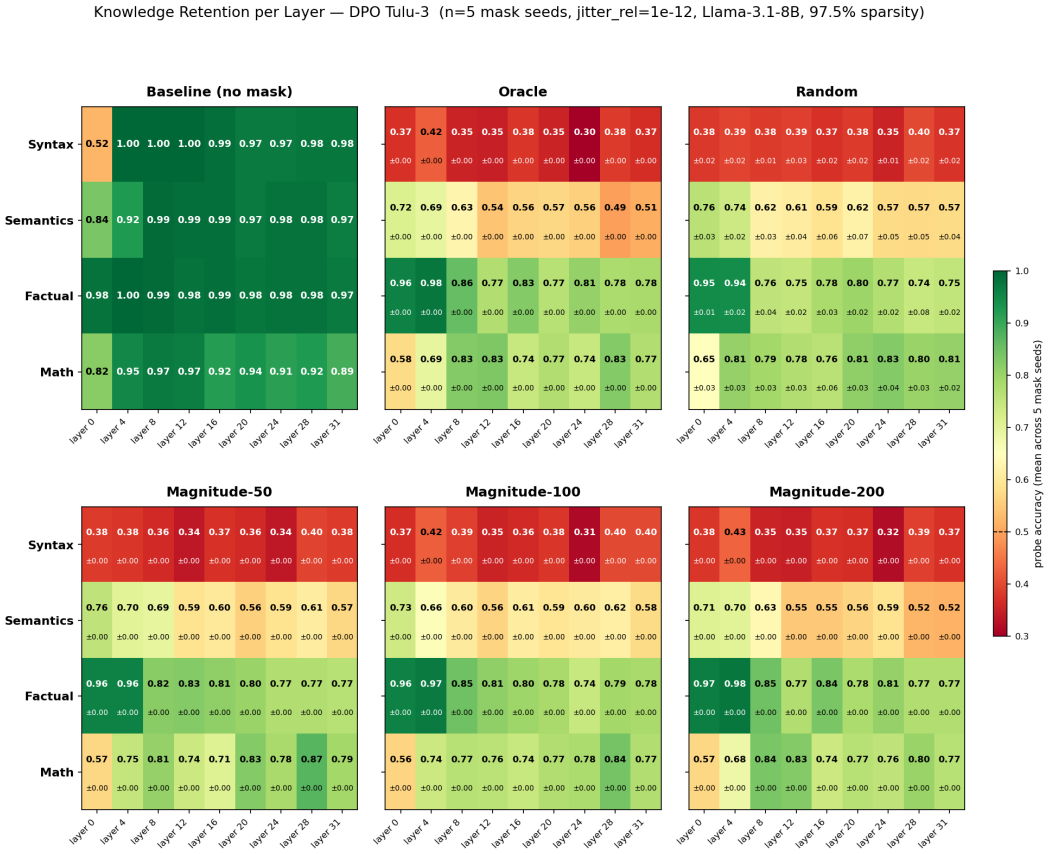


Figure 17: Linear probes test-set classification accuracy across Tülu3 DPO masks (from left to right): dense (unmodified), Oracle, Random, Magnitude 50, 100, and 200 masks at 97.5% sparsity. In sparse masks, masked out elements are zero-padded for shape consistency in probe input.

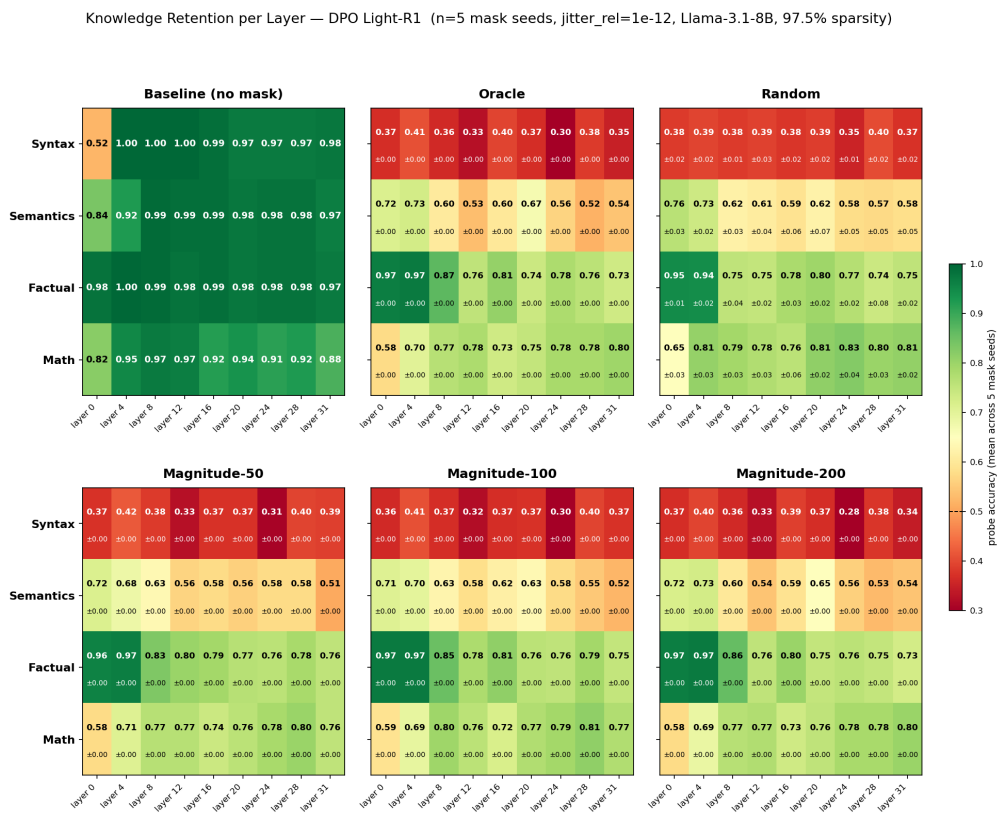


Figure 18: Linear probe test-set classification accuracy across Light-R1 DPO masks at 97.5% sparsity: dense, Oracle, random, and warm-start magnitude masks at 50, 100, and 200 steps (all with $n = 5$). Oracle and warm-start magnitude masks show greater degradation on mathematics and factual knowledge than random masks, indicating that the selected parameters encode these capabilities. Results for Tülu3 are provided in Appendix A.4.2.

A.4.3 Linear Probe Procedure

This section details how the probe accuracies $\mathcal{A}_{\mathcal{M}}[p, \ell]$ reported in Figure 18 and Figure 17 are computed. In short, the mask is first applied to the model weights, the masked model then runs a new forward pass over the probe texts, and a linear logistic probe is fitted on the resulting pooled MLP activations:

$$\mathcal{M} \;\rightarrow\; \tilde{W} = W \odot \mathcal{M} \;\rightarrow\; \text{forward pass} \;\rightarrow\; \text{capture } \texttt{down_proj} \text{ input} \\ \rightarrow\; \text{token mean pooling} \;\rightarrow\; \text{logistic probe.}$$

Weight masking. For each mask configuration, every parameter tensor W covered by the mask is temporarily replaced by $\tilde{W} = W \odot \mathcal{M}$: entries with $\mathcal{M}_{ij} = 1$ keep W_{ij}, and entries with $\mathcal{M}_{ij} = 0$ are set to zero. The original weights are restored after activation collection, and the dense baseline uses the unmodified model. Because the mask is applied to all layers at once, the input H_ℓ to layer ℓ already carries the accumulated effect of every preceding masked layer. Activation changes at layer ℓ are therefore network-level effects, not only local effects of masking that layer's matrices.

Captured activations. Let $H_\ell \in \mathbb{R}^{B\times T\times d}$ be the input to the MLP of transformer layer ℓ, where d is the hidden width and m the MLP intermediate width. The SwiGLU MLP computes

$$G_\ell = H_\ell \tilde{W}_{\text{gate}}^\top \in \mathbb{R}^{B\times T\times m}, \qquad U_\ell = H_\ell \tilde{W}_{\text{up}}^\top \in \mathbb{R}^{B\times T\times m},$$
$$A_\ell = \text{SiLU}(G_\ell) \odot U_\ell \in \mathbb{R}^{B\times T\times m}, \qquad O_\ell = A_\ell \tilde{W}_{\text{down}}^\top \in \mathbb{R}^{B\times T\times d}.$$

A forward hook registered on each `down_proj` module reads the module's input, i.e. it captures A_ℓ, the intermediate activation after the SwiGLU product. It does not capture logits, attention outputs,

the residual stream, or the output of `down_proj`. Since masked weights are zeroed rather than removed, A_ℓ has the same width m under every mask configuration.

Token pooling. Texts are tokenized with right truncation at 256 tokens. Let $q_{it} \in \{0, 1\}$ be the attention mask of token t in text i, which removes padding. Each text is reduced to one vector per layer by an attention-mask-aware mean over its real tokens,

$$v_i^{(\ell)} = \frac{\sum_t q_{it}\, A_\ell[i, t, :]}{\sum_t q_{it}} \in \mathbb{R}^m, \qquad X_\ell = \left[\, v_1^{(\ell)}, \ldots, v_N^{(\ell)} \,\right]^\top \in \mathbb{R}^{N \times m}.$$

Rows of X_ℓ are text examples, columns are MLP intermediate neurons, and a separate matrix X_ℓ is built for each hooked layer. Probes are trained on a subset of layers sampled at a fixed stride, always including the final layer.

Probe training. For each property p and each sampled layer ℓ, the rows of X_ℓ belonging to the (class-balanced) texts of p are paired with binary labels $y \in \{0, 1\}$. Each feature is standardized, $\hat{X}_{\cdot j} = (X_{\cdot j} - \mu_j)/\sigma_j$, and an ℓ_2-regularized logistic regression is fitted,

$$p(y = 1 \mid \hat{v}) = \sigma\big(\beta_\ell^\top \hat{v} + b_\ell\big), \qquad \mathcal{L}(\beta_\ell, b_\ell) = \mathrm{BCE}\big(y,\, \sigma(\beta_\ell^\top \hat{v} + b_\ell)\big) + \lambda \|\beta_\ell\|_2^2,$$

with inverse regularization strength $C = 1.0$ and the L-BFGS solver (at most 2000 iterations). We use stratified 5-fold cross-validation with shuffling and a fixed seed. The standardization is fitted inside each training fold only, which prevents test-fold leakage. For every (property, layer) cell we record the mean test accuracy across folds, which is the entry $\mathcal{A}_{\mathcal{M}}[\, p, \ell \,]$, together with the standard deviation of the test accuracy, the mean train accuracy, and the train–test gap. Activation matrices that contain non-finite values or are (near-)constant are marked as degenerate and reported as missing instead of being silently scored.

Interpretation. A high probe accuracy means that the target label is linearly decodable from the pooled MLP representation at that layer (Alain & Bengio, 2016). It does not establish that the language model causally uses the decoded feature during generation.

Relation to CKA. The CKA analysis of Section 3.3.2 shares the activation-extraction stage with the probes: both use the same feature extractor, hook the same tensor (the input of every `down_proj`), and apply the same attention-mask-aware mean pooling, so the matrices $X_A^{(\ell)}$ and $X_B^{(\ell)}$ used for CKA are the pooled matrices X_ℓ defined above, computed under two different masks. Both analyses temporarily apply and then restore the mask in the same way. They differ only in the downstream statistic: the probe standardizes X_ℓ, fits a logistic regression and reports classification accuracy, whereas CKA centers two activation matrices and reports their representational similarity.

A.4.4 Random-Mask Baselines for Jaccard and CKA

For two independent uniformly sampled binary masks $\mathcal{M}_1, \mathcal{M}_2 \in \{0, 1\}^n$ with keep rate $A = 1 - \rho$, the Jaccard similarity and linear CKA have the following expectations and variances:

$$\mathbb{E}[J] = \frac{A}{2 - A}, \qquad \mathrm{Var}[J] = \frac{2(1 - A)}{n(2 - A)^3} \tag{8}$$

$$\mathbb{E}[\mathrm{CKA}] = \frac{1}{n - 1}, \qquad \mathrm{Var}[\mathrm{CKA}] = \frac{2}{(n - 1)^2} \tag{9}$$

At LLM scale ($n = |\theta| \approx 8 \times 10^9$ for `Llama-3.1-8B-Instruct`), both variances practically vanish, so the expectations serve as sharp chance-level baselines. At the sparsities used in our experiments these evaluate to $\mathbb{E}[J] \approx 1.27 \times 10^{-2}$ at $\rho = 0.975$ and $\mathbb{E}[J] \approx 1.27 \times 10^{-3}$ at $\rho = 0.9975$; these are the horizontal references against which empirical Jaccard scores in Figure 2 should be read.

Masks are materialized as `bool` tensors on common parameter names; intersection and union are computed via PyTorch bit-ops, emitting per-layer J_ℓ alongside global and summary statistics. The computation is purely combinatorial, requires no model forward pass, and completes in seconds on CPU at 8B-parameter scale.

B SPARSE OPTIMIZATION

B.1 FULL SPARSE OPTIMIZER STEP SPEEDUP TABLE

Table 4: Optimizer-step times across the full sparsity grid on `Llama-3.1-8B-Instruct`. We report the mean ms/step over the intermediate 80% of steps, discarding the opening and closing 10% to account for GPU warmup. Speedup columns report `sparse_adamw` relative to each dense baseline (higher is faster). Each row was benchmarked on a separate NVIDIA H200, so we report ratios computed against that device's own `adamw_torch` measurement. Absolute times should not be compared across rows.

	Mean ms/step			**`sparse_adamw` speedup**	
Sparsity ρ	`adamw_torch`	`adamw_8bit`	`sparse_adamw`	vs. `torch`	vs. `8bit`
0.25%	7.120	14.108	6.208	$1.15\times$	$2.27\times$
50.0%	7.131	14.120	6.180	$1.15\times$	$2.28\times$
75.0%	7.139	14.141	6.207	$1.15\times$	$2.28\times$
90.0%	7.148	14.118	6.168	$1.16\times$	$2.29\times$
97.5%	7.132	14.129	4.214	$1.69\times$	$3.35\times$
99.75%	7.118	14.117	2.171	$3.28\times$	$6.50\times$

B.2 HARDWARE EFFICIENCY AND COMPLEXITY ANALYSIS

To realize the theoretical benefits of sparsity, our training pipeline optimizes for both compute complexity (FLOPs) and memory bandwidth (Bytes moved). We model the performance gains of our sparse training stack compared to dense baselines.

B.2.1 MEMORY COMPLEXITY (BANDWIDTH)

Element-wise optimizer steps (e.g., AdamW) are memory-bandwidth bound. For a model with P parameters, standard AdamW moves $7 \times 4P = 28P$ bytes of data (reading/writing parameters, gradients, and two momentum buffers).

Our indexed sparse optimizer gather and scatter only from the $A \cdot P$ non-zero coordinates. The memory movement is reduced to $28AP$. At 97.5% sparsity, this represents a 97.5% reduction in VRAM traffic. The Practical throughput is slightly lower due to non-coalesced memory access patterns, but remains significantly faster than dense counterparts as P scales.

Table 5: Theoretical complexity comparison per training step.

Method	Backprop FLOPs	Optimizer Memory I/O
Dense AdamW	$1.0\times$	$28P$ bytes
Indexed AdamW	$1.0\times$	$28AP$ bytes

B.3 BLOCK SPARSE ROW (BSR) BACKWARD KERNEL

A BSR backwards kernel for MLP layers. Not included in sparse training results or in practical speed measurement phases, but developed as a future application/direction.

Algorithm 1 Triton BSR Backward Kernel for Weight Gradients

Require: $\delta \in \mathbb{R}^{B\times D_{out}}$, $\mathbf{x} \in \mathbb{R}^{B\times D_{in}}$, $\mathcal{M} \in \{0,1\}^{D_{out}\times D_{in}}$; Block size $N = 16$
Ensure: $\nabla_w \mathcal{L} \in \mathbb{R}^{D_{out}\times D_{in}}$

Parallel for each block (i, j) in grid $(D_{out}/N, D_{in}/N)$:
  Load mask block $\mathcal{M}_{i,j} \in \mathbb{R}^{N\times N}$
  if $\max(\mathcal{M}_{i,j}) == 0$ **then**
    Store $\nabla_w \mathcal{L}_{i,j} = \mathbf{0}_{N\times N}$; **return**
  end if
  Initialize accumulator $\mathbf{acc} \in \mathbb{R}^{N\times N} = \mathbf{0}$
  for $b_{\text{start}} = 0$ to B step K **do**:
    Load $\delta[b_{\text{start}} : b_{\text{start}} + K, i \cdot N : (i + 1) \cdot N]$
    Load $\mathbf{x}[b_{\text{start}} : b_{\text{start}} + K, j \cdot N : (j + 1) \cdot N]$
    $\mathbf{acc} \leftarrow \mathbf{acc} + \delta_{\text{block}}^{\top} \mathbf{x}_{\text{block}}$
  end for
  $\nabla_w \mathcal{L}_{i,j} \leftarrow \mathbf{acc} \odot \mathcal{M}_{i,j}$
  Store $\nabla_w \mathcal{L}_{i,j}$

B.4 Triton BSR AdamW Kernel

For a model with P trainable parameters and sparsity $S \in [0, 1)$ (active fraction $A = 1 - S$):

$$\begin{aligned} \text{Dense AdamW:} \quad & \text{Time, Memory} = \mathcal{O}(P) \\ \text{BSR-AdamW:} \quad & \text{Time, Memory} = \mathcal{O}(A \cdot P) = \mathcal{O}((1 - S) \cdot P) \end{aligned}$$

Dense AdamW touches every parameter each step (read p, g, m, v; write m', v', p'), giving $\mathcal{O}(P)$ memory traffic and time. BSR-AdamW restricts updates to the AP active parameters, scaling all three as $\mathcal{O}((1-S)P)$.

Time and memory complexity. We analyze the 1D variant (Algorithm 4); the 2D BSR variant (Algorithm 5) is not exercised in our experiments. Let $P_{\mathcal{M}}$ be the parameter count covered by the mask, L the number of masked tensors, $A = 1 - \rho$ the active fraction, and d_p the per-element dtype size. The deployed pipeline runs in `bfloat16` throughout (parameters, gradients, and both Adam moments), so $d_p = 2$.

Time complexity (per optimizer step). Each lane issues one `int64` index gather, four element-sized gathers (param, grad, m, v) and three element-sized scatters (param, m, v); the arithmetic is $\Theta(1)$ AdamW math per active element. Aggregate cost is therefore $\Theta(A\, P_{\mathcal{M}})$ memory transactions plus $\Theta(L)$ kernel-launch overhead. Concretely the optimizer moves

$$\text{IO}_{\text{sparse}} \;=\; \big(8 + 7\, d_p\big)\, A\, P_{\mathcal{M}} \;=\; 22\, A\, P_{\mathcal{M}} \text{ bytes per step}, \tag{10}$$

versus $14\, P_{\mathcal{M}}$ bytes for dense AdamW, a bandwidth-bound speedup ratio of $0.636/A$ ($\sim$25$\times$ at $\rho = 0.975$). Compute is $\approx 14\, A\, P_{\mathcal{M}}$ FLOPs, dominated by bandwidth on H200 (4.8 TB/s HBM3, $\sim$67 TFLOPs/s) by roughly 30$\times$.

Memory complexity (resident). The deployed implementation holds dense `exp_avg` and `exp_avg_sq` buffers, so moment storage is $\Theta(P_{\mathcal{M}})$ — identical to dense AdamW. The Boolean mask adds $\Theta(P_{\mathcal{M}})$ bytes (one byte per parameter; `torch.bool`) and the sorted `int64` active-index list adds $\Theta(A\, P_{\mathcal{M}})$. Resident footprint is therefore $\Theta(P_{\mathcal{M}}) + \Theta(A\, P_{\mathcal{M}})$, strictly larger than the dense baseline by $(1 + 8A)\, P_{\mathcal{M}}$ bytes; the kernel reduces optimizer *I/O* by $\Theta(1/A)$, not VRAM. Asymptotic VRAM compaction would require the `USE_SPARSE_STATES` packed-state path, not deployed here.

Realised vs. theoretical. Two effects separate the $\Theta(1/A)$ I/O ratio from wall-clock. Gather/scatter coalescing on unstructured masks at $A = 0.025$ has expected index gap $1/A = 40$, so each 32-byte HBM transaction carries $\sim$0.2 useful `bf16` elements ($\eta \lesssim 1/4$). Per-tensor launch overhead $\Theta(L) \approx 1-2$ ms ($L \sim 200$ on `Llama-3.1-8B-Instruct`, $5-10\,\mu$s each) is comparable to the bandwidth term at high sparsity. Realised optimizer-step speedups at $\rho = 0.975$ are shown in Table 2 and expanded in Table 4.

The 1D or element-wise variant, where $\mathcal{M}$ is assumed to be sampled from $\mathcal{U}(\prime, |\theta|)$ 2 (Kingma & Ba, 2014). This is the general case we implement. Assume all implementations use this approach unless specifically stated otherwise.

Algorithm 2 BSR-aware AdamW update — `block_1d` (1D grid, `bsr_sparse_adamw_kernel`)

Require: Weight matrix $P \in \mathbb{R}^{M \times N}$, gradient G (same shape); first/second moments m, v (dense $M \times N$ **or** packed length KB^2 if sparse storage); active block list `active_blocks` $\in \mathbb{Z}^K$; block size B (`BLOCK_SIZE` $= 16$); $(\texttt{lr}, \beta_1, \beta_2, \varepsilon, \texttt{wd})$; bias corrections $\texttt{bc}_1, \texttt{bc}_2$; flag `USE_SPARSE_STATES`

Ensure: In-place updates to P, m, v on active tiles only

Launch one program per active block: `pid` $\in \{0, \ldots, K-1\}$
if `pid` $\geq K$ **then return**
end if
$b \leftarrow$ `active_blocks[pid]` ▷ flat block id
$N_b \leftarrow \lceil N/B \rceil$, $b_m \leftarrow \lfloor b/N_b \rfloor$, $b_n \leftarrow b \bmod N_b$
Form row indices $r_s \leftarrow b_m B + s$ and column indices $c_t \leftarrow b_n B + t$ for $s, t \in \{0, \ldots, B-1\}$
Validity mask $\mathcal{V}_{s,t} \leftarrow (r_s < M) \wedge (c_t < N)$
Load $B \times B$ tiles $\mathbf{p}, \mathbf{g}$ from P, G at (r_s, c_t), treating out-of-bounds as 0
if `USE_SPARSE_STATES` **then**
    Base offset $\ell_0 \leftarrow$ `pid` $\cdot B^2$; load $\mathbf{m}, \mathbf{v}$ from contiguous slices $m[\ell_0{:}\ell_0{+}B^2)$, $v[\ell_0{:}\ell_0{+}B^2)$ reshaped in row-major order aligned with (s, t)
else
    Load $\mathbf{m}, \mathbf{v}$ from m, v at the same (r_s, c_t) coordinates as $\mathbf{p}$
end if
AdamW: $\mathbf{p}' \leftarrow \mathbf{p} \odot (1 - \texttt{lr} \cdot \texttt{wd})$ ▷ elementwise
$\mathbf{m}' \leftarrow \beta_1 \mathbf{m} + (1-\beta_1)\mathbf{g}$, $\mathbf{v}' \leftarrow \beta_2 \mathbf{v} + (1-\beta_2)\mathbf{g} \odot \mathbf{g}$
$\widehat{\mathbf{m}} \leftarrow \mathbf{m}'/\texttt{bc}_1$, $\widehat{\mathbf{v}} \leftarrow \mathbf{v}'/\texttt{bc}_2$
$\mathbf{p}_{\text{new}} \leftarrow \mathbf{p}' - \texttt{lr} \cdot (\widehat{\mathbf{m}}/(\sqrt{\widehat{\mathbf{v}}} + \varepsilon))$ ▷ "/" is elementwise (Hadamard) division
Store $\mathbf{p}_{\text{new}}$ into P at valid (r_s, c_t); store $\mathbf{m}', \mathbf{v}'$ to the same storage layout used for loads

And the 2D or $n : m$ variant, which assumes that $\mathcal{M}$ is inherently $n : m$ block-sparse 3.

Algorithm 3 BSR-aware AdamW update — `block_2d` (2D grid, `bsr_sparse_adamw_kernel_2d`; `block_1d` is the $T{=}B$ special case)

Require: Same tensors and hyperparameters as Algorithm 2; row tile size $T{=}$`ROW_TILE` with $B/T \in \mathbb{Z}_{>0}$ (default $T{=}8$ when $B{=}16$)

Ensure: In-place updates to P, m, v on active row-tiles only

Launch 2D grid: $\texttt{pid}_b \in \{0, \ldots, K-1\}$, $\texttt{pid}_t \in \{0, \ldots, B/T - 1\}$
if $\texttt{pid}_b \geq K$ **then return**
end if
$b \leftarrow$ `active_blocks`$[\texttt{pid}_b]$; $N_b \leftarrow \lceil N/B \rceil$; decode (b_m, b_n) as in Algorithm 2
Row slice $r_s \leftarrow b_m B + \texttt{pid}_t \cdot T + s$ for $s \in \{0, \ldots, T-1\}$; $c_t \leftarrow b_n B + t$ for $t \in \{0, \ldots, B-1\}$
$\mathcal{V}_{s,t} \leftarrow (r_s < M) \wedge (c_t < N)$; load $T \times B$ tiles $\mathbf{p}, \mathbf{g}$ from P, G
if `USE_SPARSE_STATES` **then**
    $\ell_0 \leftarrow \texttt{pid}_b \cdot B^2$; $\ell_1 \leftarrow \texttt{pid}_t \cdot T \cdot B$; load $\mathbf{m}, \mathbf{v}$ from m, v at offsets $\ell_0 + \ell_1$ in row-major layout over the full $B \times B$ block
else
    Load $\mathbf{m}, \mathbf{v}$ at coordinates (r_s, c_t)
end if
$\mathbf{p}' \leftarrow \mathbf{p} \odot (1 - \texttt{lr} \cdot \texttt{wd})$; $\mathbf{m}' \leftarrow \beta_1 \mathbf{m} + (1-\beta_1)\mathbf{g}$; $\mathbf{v}' \leftarrow \beta_2 \mathbf{v} + (1-\beta_2)\mathbf{g} \odot \mathbf{g}$
$\widehat{\mathbf{m}} \leftarrow \mathbf{m}'/\texttt{bc}_1$; $\widehat{\mathbf{v}} \leftarrow \mathbf{v}'/\texttt{bc}_2$; $\mathbf{p}_{\text{new}} \leftarrow \mathbf{p}' - \texttt{lr} \cdot (\widehat{\mathbf{m}}/(\sqrt{\widehat{\mathbf{v}}} + \varepsilon))$ ▷ "/" elementwise
Scatter results back to P, m, v with the same indexing as the loads

B.5 ACCELERATION ABLATION DETAILS

Random global masks target sparsities $\{99.75, 97.5, 95, 90\}\%$. Tokenizer and DPO preference dataset loaded once (`tulu3` registry / Allen AI mixture); each phase reloads model weights so dense and sparse graphs are not fused across phases. Per-device batch size, gradient accumulation, learning rate, LR warmup fraction, sequence caps, optimizer choice for the dense baseline, gradient checkpointing, BSR block size, and SparseAdamW block size match the deployed Slurm driver defaults (replace every *TBD* with the value from your Slurm env / job log).

Table 6: DPO throughput benchmark — H200 sparse BSR / SparseAdamW grid (L`Llama-3.1-8B-Instruct`, randomized global masks).

Hyperparameter	Value
Hardware	$1 \times$ NVIDIA H200
Base model	`meta-llama/Llama-3.1-8B-Instruct`
Preference dataset	Tulu-3
Device placement	Single-GPU (no model sharding or offload)
Floating-point precision	BF16 weights / activations; DPO Trainer `bf16=True`
TF32 (cuBLAS)	Enabled (PyTorch default; not explicitly disabled)
Gradient checkpointing	Enabled
Epochs per phase	1
Optimizer updates per phase (max steps)	25
Learning rate	5×10^{-7}
LR schedule	Linear decay; warmup ratio 0.1
Weight decay	0.0
Adam $\beta_1, \beta_2, \epsilon$	0.9, 0.999, 10^{-8}
Max gradient norm	1.0
DPO β	0.1
Per-device batch size	2
Gradient accumulation steps	64
Effective batch (pairs per optimizer step)	$2 \times 64 = 128$ chosen/rejected pairs
Max prompt length / Max sequence length	1024 / 1024 tokens
Mask construction	Random global scores, deterministic per (sparsity, mask type) seed; per-layer keep floor of 2.5×10^{-3}
Mask scope	Full model (all masked 2-D linear weights)
Targeted global sparsity grid (%)	$\{90,\ 95,\ 97.5,\ 99.75\}$
Mask types	Element-wise (unstructured) and block (BSR-aligned)
Sparse factorial grid per job	4 sparsities $\times$ 2 mask types $\times$ 2 SparseAdamW kernels $= 16$ timed sparse phases (plus 1 dense baseline phase when included)
Dense baseline phase	Standard `nn.Linear` stack; HF Trainer optimizer AdamW 8-bit (bitsandbytes) (Dettmers et al., 2022)
Sparse phases	SparseLinear with BSR backward on weights; dense forward and dense grad-input w.r.t. activations
Sparse optimizer	SparseAdamW (this work); kernel block layout switched between BLOCK_1D and BLOCK_2D per phase
BSR tile / SparseAdamW block tile	16 / 128
Throughput instrumentation	Per-step Trainer logging; micro-batch CUDA profiler timings disabled
Stable BSR kernel settings	Atomic accumulation disabled; batch-axis launch-tile floor $= 1$
Checkpoint / weight saves	Disabled

C Scoring and Masking Algorithms

C.1 Scoring Algorithms

Algorithm 4 Warm-start magnitude scoring over streamed checkpoints

Require: Ordered checkpoint paths $\{\Delta\theta_1, \ldots, \Delta\theta_T\}$; optional filter on parameter names (e.g. MLP-only)
Ensure: Score tensor S aligned with each parameter in θ
1: Initialize $A \leftarrow 0$ for every scored parameter shape
2: **for** each checkpoint $t = 1, \ldots, T$ **do**
3: Load $\Delta\theta_t$; optionally drop non-MLP tensors
4: **for** each parameter tensor p **do**
5: $A[p] \leftarrow A[p] + |\Delta\theta_t[p]|$ ▷ elementwise
6: **end for**
7: **end for**
8: $S \leftarrow A$

C.2 Mask Construction Algorithms

Algorithm 5 Default hybrid mask: global budget with per-tensor keep floor (`local_pool`= false, `min_layer_keep_ratio`= r)

Require: Score tensors $\{S_p\}$; target exclusion fraction $\rho \in [0, 1]$; floor ratio $r \in [0, 1]$; optional tie-break noise on scores (default on in API)
Ensure: Binary inclusion masks $\{m_p\}$ with total kept count $k_{\text{keep}} = \lfloor (1 - \rho)\, N \rfloor$, $N = \sum_p \text{numel}(S_p)$
1: Sanitize S_p (replace NaN/Inf; dtype float32)
2: **if** tie-break enabled **then**
3: add i.i.d. Gaussian noise to each S_p with scale $\propto$ global max $|S|$ (fixed RNG seed in code)
4: **end if**
5: Initialize all $m_p \leftarrow 0$
6: **Floor pass:** for each p, set $f_p \leftarrow \lfloor r \cdot \text{numel}(S_p) \rfloor$; if $\sum_p f_p > k_{\text{keep}}$, multiply all f_p by $k_{\text{keep}} / \sum_q f_q$ and round down within each layer's size
7: **for** each p with $f_p > 0$ **do**
8: $I_p^{\text{floor}} \leftarrow$ indices of the top-f_p elements of $\text{vec}(S_p)$
9: set $m_p[i] = 1$ for $i \in I_p^{\text{floor}}$
10: **end for**
11: $k_{\text{rem}} \leftarrow k_{\text{keep}} - \sum_p |I_p^{\text{floor}}|$
12: **Global pass:** among positions not fixed by the floor, select the k_{rem} largest scores (implementation: flat topk on masked concatenated scores if N is below an internal threshold; otherwise chunked threshold search on CPU with histogram refinement, then topk within the final score bin)

C.3 HYBRID τ ESTIMATION ALGORITHM

Algorithm 6 Certification threshold τ under `hybrid_global_phase` (global phase after per-layer floors; matches mask construction in Algorithm 5)

Require: Layer-partitioned selection scores $\{\tilde{S}_p\}_{p=1}^{L}$ in fixed iteration order; same sparsity / exclusion hyperparameters as Algorithm 5: global keep budget k_{keep} on $N = \sum_p \text{numel}(\tilde{S}_p)$, floor ratio r; optional tie-break (same rule as Algorithm 5)

Ensure: Scalar τ (global-phase cutoff) or undefined if the global remainder is empty

1: **Floors:** compute per-layer floor counts $\{f_p\}_{p=1}^{L}$ *with the same scaling rule* as the floor pass of Algorithm 5 (i.e. $f_p \approx \lfloor r \cdot \text{numel}(\tilde{S}_p) \rfloor$, then scale down uniformly if $\sum_p f_p > k_{\text{keep}}$, clipping each f_p to $[0, \text{numel}(\tilde{S}_p)]$)

2: **Sanitize and tie-break:** for each p, form $\text{vec}(\tilde{S}_p)$; replace NaN/Inf; if tie-break is on, add i.i.d. Gaussian noise with scale $\propto$ global $\max|\tilde{S}|$ (fixed seed), matching Algorithm 5

3: Flatten layers in order into one vector $x \in \mathbb{R}^N$ (logical concatenation $\tilde{S}_1 \| \cdots \| \tilde{S}_L$)

4: **Floor reservation (score space):** initialize offset $o \leftarrow 0$. For each layer p, let $x[o : o + n_p)$ be the slice of length $n_p = \text{numel}(\tilde{S}_p)$. If $f_p > 0$, let I_p be the indices of the top-f_p entries in that slice; set $x_i \leftarrow -\infty$ for $i \in I_p$. Update $o \leftarrow o + n_p$

5: Let $F \leftarrow \sum_p f_p$ and $R \leftarrow k_{\text{keep}} - F$ *// global-phase quota after floors*

6: **if** $R \leq 0$ **then**

7: **return** τ undefined (NaN in code): no global-phase slots remain

8: **end if**

9: Let $\mathcal{T}$ be the set of the R largest *finite* entries of x (ties broken by the noise in step 2)

10: **return** $\tau \leftarrow \min \mathcal{T}$ *// boundary of the global-pass keep set; if F=0, R=k_{keep} recovers pure global τ*

Streaming variant (same τ). When N is too large to materialize x, process layers in order: for each p, build the sanitized/tie-broken slice, mark its top-f_p entries as $-\infty$, and feed slices sequentially into a buffer that maintains only the best R values seen so far over the logical concatenation; then τ is the minimum of those R values (equivalently: $\tau = \min \text{top-}R(x)$ over finite entries).

Relation to Algorithm 5. Algorithm 5 assigns binary masks: floors take F weights, then the global pass keeps the top-$k_{\text{rem}} = R$ scores among non-floor positions. Algorithm 6 computes the *score cutoff* τ at the bottom of that global-pass keep set. The selection scores $\{\tilde{S}_p\}$ are whatever tensor defines the ranking for the certification track (e.g. magnitude of weight movement $|\Delta w|$ for an Oracle reference, or warm-start scores for magnitude); sanitation and tie-break match the mask pipeline so τ is comparable to the hybrid mask's ranking semantics.

D THEORETICAL DETAILS

D.1 PROOF OF THEOREM 1: LOSS RETENTION

We derive the bound in Theorem 1. By the mean value theorem there exists ξ on the segment between $\theta_{\mathcal{M}}$ and $\theta + \Delta\theta$ such that:

$$\mathcal{L}(\theta_{\mathcal{M}}) - \mathcal{L}(\theta + \Delta\theta) = -\nabla\mathcal{L}(\theta + \Delta\theta)^\top \delta + \frac{1}{2}\delta^\top H(\xi)\delta$$

This is exact by the mean value theorem; no approximation has been made. For the first term, by Hölder's inequality:

$$-\nabla\mathcal{L}(\theta + \Delta\theta)^\top \delta \leq \|\nabla\mathcal{L}(\theta + \Delta\theta)\|_\infty \|\delta\|_1 \leq \epsilon\|\delta\|_1$$

where $\epsilon = \|\nabla\mathcal{L}(\theta + \Delta\theta)\|_\infty = \max_i |\nabla_i\mathcal{L}(\theta)|$. For the second term, substituting $H(\xi) = D_S + R_S(\xi)$ where $D_S = \text{diag}(\kappa(\theta))$ with $\kappa(\theta_i) \geq 0$ satisfying $s(\theta_i) = \frac{\kappa(\theta_i)}{2}\Delta\theta_i^2$:

$$\frac{1}{2}\delta^\top H(\xi)\delta = \frac{1}{2}\sum_{i\notin\mathcal{M}} \kappa(\theta_i)\Delta\theta_i^2 + \frac{1}{2}\delta^\top R_S(\xi)\delta = \sum_{i\notin\mathcal{M}} s(\theta_i) + \frac{1}{2}\delta^\top R_S(\xi)\delta$$

For the residual term, we use the matrix ∞-norm bound: for any matrix A and vector v,

$$|v^\top A v| \leq \|v\|_1^2 \cdot \|A\|_\infty$$

Applying this with $A = R_S(\xi)$ and $v = \delta$:

$$\left|\frac{1}{2}\delta^\top R_S(\xi)\delta\right| \leq \frac{\|R_S\|_\infty}{2}\|\delta\|_1^2$$

where $\|R_S\|_\infty = \|H - D_S\|_\infty = \max_i \sum_{j\neq i} |H_{ij}| + |H_{ii} - \kappa(\theta_i)|$. Setting $\kappa(\theta_i) = H_{ii}$ minimizes $\|R_S\|_\infty$ exactly, yielding the irreducible residual $C = \max_i \sum_{j\neq i} |H_{ij}|$. Combining all terms:

$$\mathcal{L}(\theta_{\mathcal{M}}) - \mathcal{L}(\theta + \Delta\theta) \leq \sum_{i\notin\mathcal{M}} s(\theta_i) + \frac{C}{2}\|\delta\|_1^2 + \epsilon\|\delta\|_1$$

where we write ϵ for ϵ_∞ for brevity. □

General norm pairs. The bound generalizes to any norm pair (p, q) satisfying the compatibility condition $|v^\top A v| \leq \|A\|_{(p)}\|v\|_q^2$, where $\|\cdot\|_{(p)}$ is the matrix norm induced by $\|\cdot\|_q$. The ℓ_1 choice is adopted in the main text because it admits the closed-form minimizer $\kappa(\theta_i) = H_{ii}$ for $\|R_S\|_\infty$ over all diagonal matrices, which is not available for other norm choices such as the spectral norm.

D.2 OPTIMAL SCORING FUNCTION

Theorem 3 (Optimal Scoring Function)**.** *There exists a fixed point B^* satisfying $\Phi(B^*) = B^*$, where $\Phi(B) = \sum_{i\notin\mathcal{M}(B)} |\Delta\theta_i|$ and $\mathcal{M}(B)$ is the top-$(1-\rho)$ mask by $s^*(\theta_i, B) = \frac{H_{ii}}{2}\Delta\theta_i^2 + (CB + \epsilon)|\Delta\theta_i|$. The scoring function $s^*(\theta_i, B^*)$ minimizes the bound in Theorem 1 over all masks at sparsity ρ. B^* is unique under the magnitude-curvature alignment condition: parameters with smaller $H_{ii}\Delta\theta_i^2$ also have smaller $|\Delta\theta_i|$, empirically supported by Fernandez et al. (2025).*

Derivation. The optimal mask minimizes:

$$f(\mathcal{M}) = \sum_{i\notin\mathcal{M}} s(\theta_i) + \frac{C}{2}\|\delta\|_1^2 + \epsilon\|\delta\|_1$$

subject to $|\mathcal{M}| = (1-\rho)|\theta|$. Writing $B = \|\delta\|_1 = \sum_{i\notin\mathcal{M}} |\Delta\theta_i|$ and $g(B) = \frac{C}{2}B^2 + \epsilon B$, this becomes:

$$\min_{\mathcal{M}} \sum_{i\notin\mathcal{M}} \frac{H_{ii}}{2}\Delta\theta_i^2 + g(B(\mathcal{M}))$$

For fixed B, the optimal mask minimizes $\sum_{i\notin\mathcal{M}} \left[\frac{H_{ii}}{2}\Delta\theta_i^2 + g'(B)|\Delta\theta_i|\right] = \sum_{i\notin\mathcal{M}} \left[\frac{H_{ii}}{2}\Delta\theta_i^2 + (CB + \epsilon)|\Delta\theta_i|\right]$, which is solved by the top-$(1-\rho)$ mask under $s^*(\theta_i, B)$ via a Lagrangian relaxation with multiplier $\lambda = g'(B) = CB + \epsilon$. Since B depends on the mask and the mask depends on B, the solution is the fixed point of $\Phi(B) = \sum_{i\notin\mathcal{M}(B)} |\Delta\theta_i|$.

Existence. Φ maps the compact set $[0, \|\theta\|_1]$ to itself and is non-increasing: larger B increases scores proportionally to $|\Delta\theta_i|$, shifting pruning toward smaller weights and decreasing $\Phi(B)$. Since Φ takes values in a finite set (there are finitely many masks at sparsity ρ), a fixed point exists.

Uniqueness. Under the magnitude-curvature alignment condition, the rankings induced by $\frac{H_{ii}}{2}\Delta\theta_i^2$ and $|\Delta\theta_i|$ agree, so the ranking induced by $s^*(\theta_i, B)$ is invariant to B. This means Φ is strictly decreasing, guaranteeing a unique fixed point.

Iterative algorithm. Initialize $B^{(0)} = 0$. At each step t: (1) compute $s^*(\theta_i, B^{(t)})$ for all i; (2) construct the top-$(1-\rho)$ mask $\mathcal{M}^{(t)}$; (3) set $B^{(t+1)} = \sum_{i \notin \mathcal{M}^{(t)}} |\Delta\theta_i|$. Terminate when $|B^{(t+1)} - B^{(t)}| < \eta$. Convergence is guaranteed since Φ is non-increasing and bounded, and is fast in practice (typically < 10 iterations) since Φ is contractive near B^* under the alignment condition.

D.3 Certifiability: Analysis of Scoring Functions

Theorem 2 provides a per-weight condition for certifying that a practical scoring function $s(\theta)$ recovers the same mask as $s^*(\theta_i, B^*)$. The condition depends on two quantities: the deviation $|s(\theta_i) - s^*(\theta_i, B^*)|$ and the margin $m_i(s) = |s(\theta_i) - \tau_\rho(s)|$.

Warm-start magnitude. For $s(\theta_i) = |\theta_i^k - \theta_i^0|$, the deviation from the optimal scoring function $s^* = \frac{H_{ii}}{2}\Delta\theta_i^2 + (CB^* + \epsilon)|\Delta\theta_i|$ is controlled by how well early parameter movement approximates s^*. Parameters with larger H_{ii} and large $|\theta_i|$ receive higher scores under s^*, since both curvature and parameter magnitude increase the bound associated with pruning them. Section 4.4 demonstrates empirically on DPO with Light-R1 that warm-start parameter displacement closely follows this optimal importance score, explaining why warm-start magnitude induces a ranking similar to s^*.

Random scoring. For $s(\theta_i) \sim \mathcal{U}(0, |\Delta\theta|)$, the deviation from s^* is $\mathcal{O}(|\Delta\theta|)$ for all i, while the margin $m_i(s)$ concentrates near zero at high sparsity. The certifiability condition therefore fails for a constant fraction of weights, explaining the consistent performance gap between random and Oracle masks observed in Figure 1.

D.4 Certifiability Margins & Gradient Energy Capture

We continue the theoretical analysis of our theoretical framework described in Section 3.2.2 and Section 4.4 across a variety of models, datasets, learning objectives, and mask sparsities, ρ. An additional metric introduced here is *gradient energy capture*, ϕ, as a quantitative measure of training signal captured by masks vs. random baselines.

D.4.1 Additional Certifiability Margins

We report additional certifiablility margin results alongside those in Section 3.2.2, arrayed across k and ρ combinations of interest. These present alternatives to Figure 5 across ρ and model family. We see that the behavior that $m_i(k) \to m_i(T)$ as $k \to T$, with some potential for overshooting $m_i(T)$, persists across $\rho \in \{97.5, 99, 99.5, 99.9\}$.

Margins in `Olmo-3-7B-IT` DPO on Light-R1

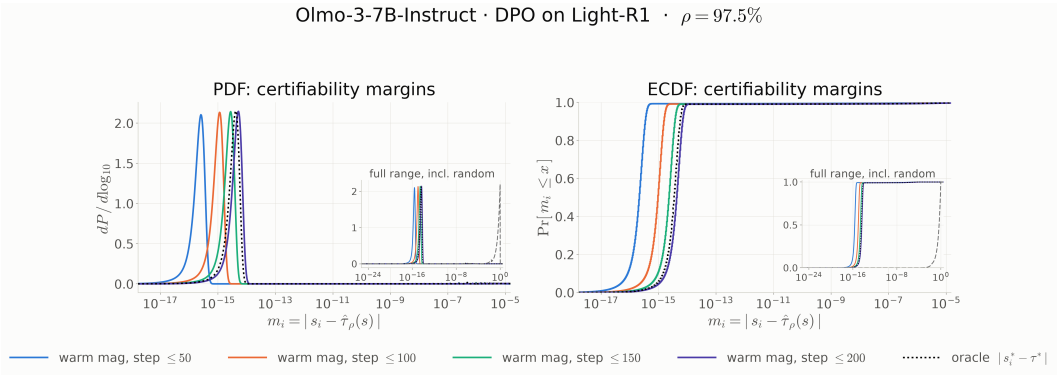


Figure 19: See caption in Figure 5

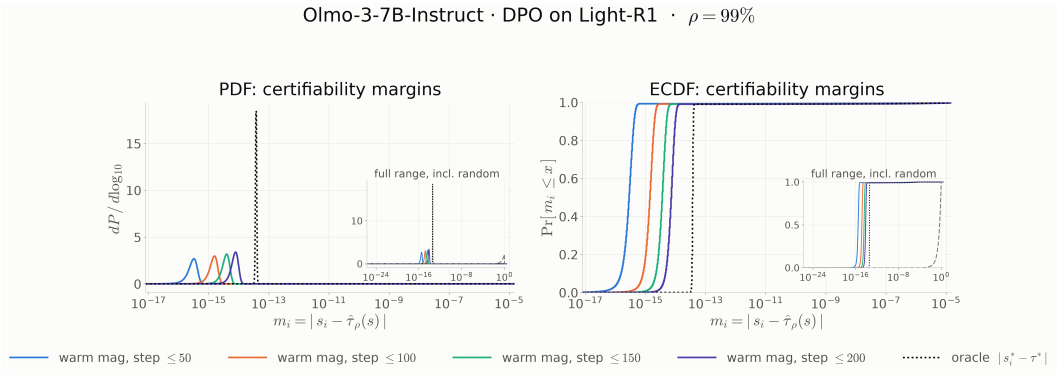


Figure 20: See caption in Figure 5

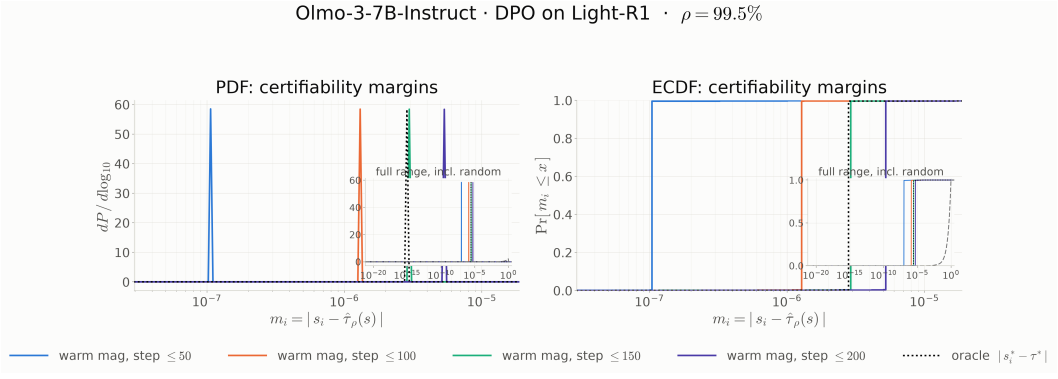


Figure 21: See caption in Figure 5

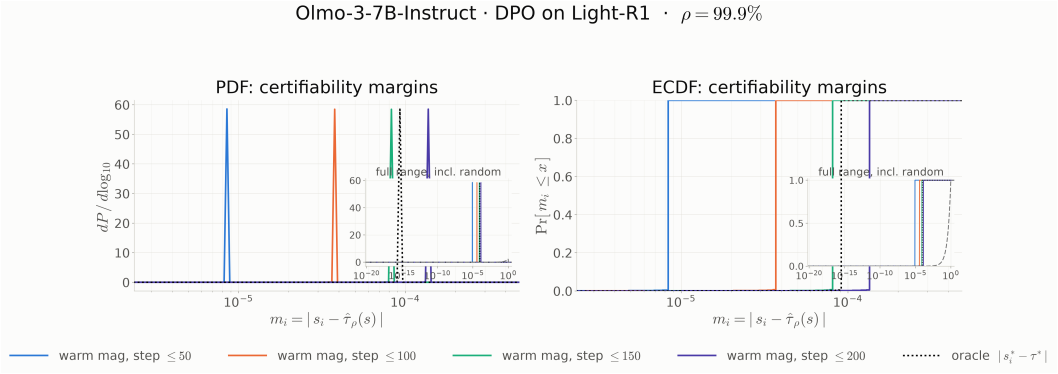


Figure 22: See caption in Figure 5

Margins in `Qwen3-8B` DPO on Light-R1

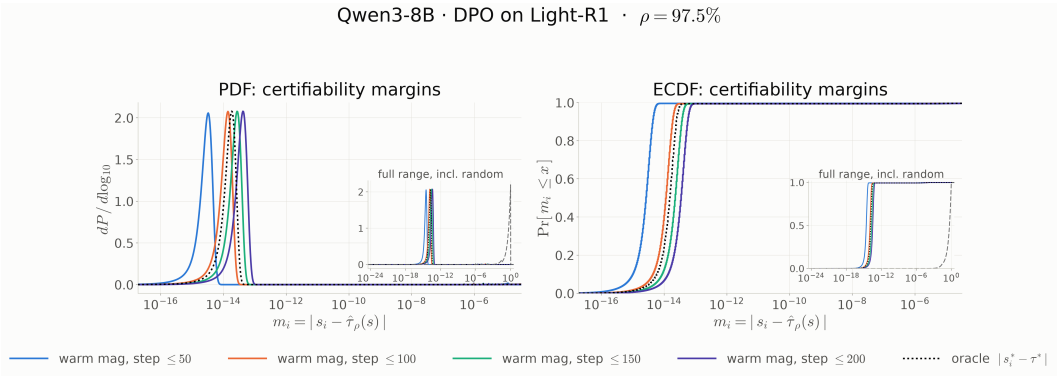


Figure 23: See caption in Figure 5

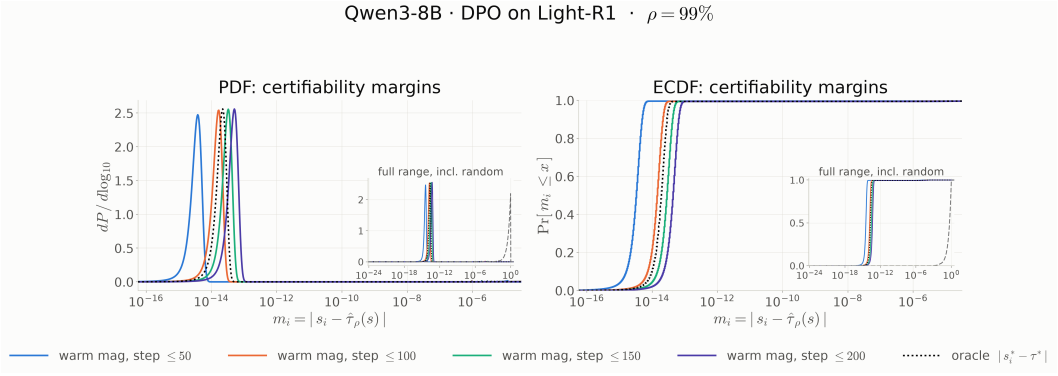


Figure 24: See caption in Figure 5

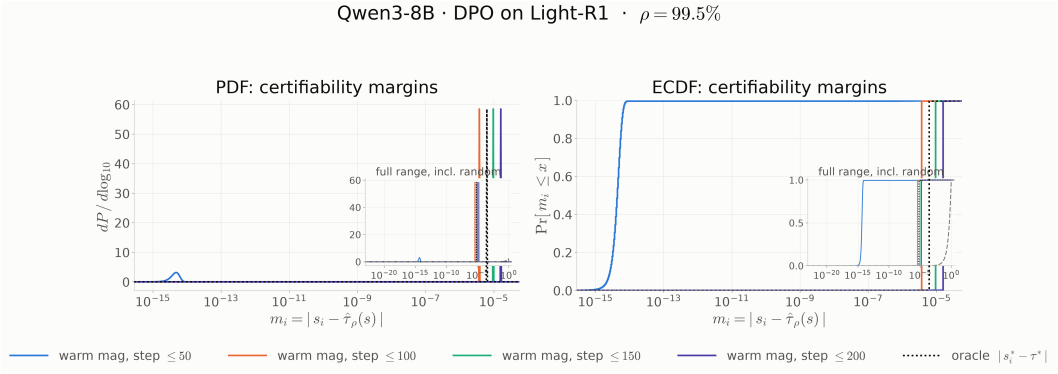


Figure 25: See caption in Figure 5

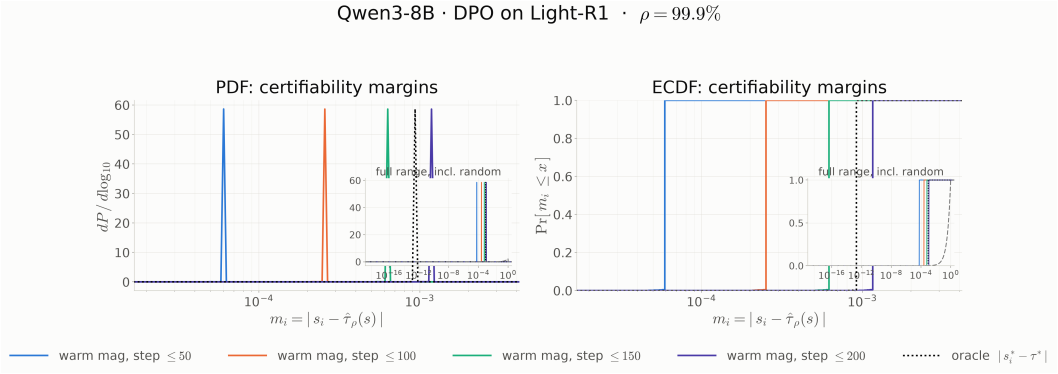


Figure 26: See caption in Figure 5

D.4.2 Gradient Energy Capture

While gradient energy capture is an appealing metric from the perspective of training dynamics, a mask built from the earliest displacement is best aligned with the *initial* gradient - and drifts over time. Therefore, ϕ is not a simple proxy for mask quality, as proximity to the initial gradients and proximity to the gradient capture of the Oracle or dense training are different objectives.

In this data, `Llama-3.1-8B-Instruct` masks produced from Open-R1 training were used to collect ϕ on Open-R1 and on Light-R1 to examine cross-task application. All DPO runs shown in this subsection are on Light-R1.

Here we benchmark measures of ϕ against 'chance', or the ϕ_{random} captured by a random mask drawn at equal sparsity.

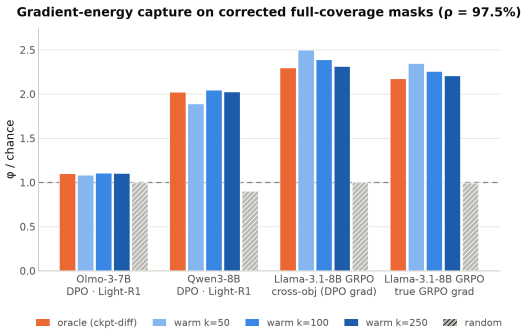


Figure 27: Gradient energy capture across $k \in 50, 100, 250$ and the Oracle against a random baseline. The dashed line indicates the capture of the random baseline.

Below is an additional view of the same underlying data collection, which clarifies the evolution of ϕ/chance over k.

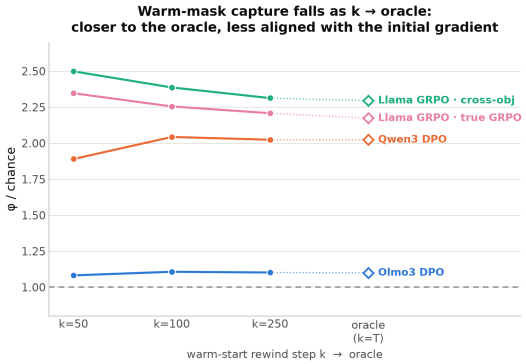


Figure 28: Gradient energy capture falls as $k \to T$. For both `Llama-3.1-8B-Instruct` GRPO gradients capture falls monotonically as k grows toward the Oracle. `Qwen3-8B` DPO is roughly flat, and `Olmo-3-7B-IT` DPO is flat just above chance. The dashed line indicates the capture of the random baseline.

E HYPERPARAMETERS

E.1 GRPO ON OPEN-R1

Table 7: GRPO (Open-R1 Math) training hyperparameters — `Llama-3.1-8B-Instruct`, locked defaults. These hyperparameters are also used in all sparse GRPO runs described in our research.

Hyperparameter	Value
Base model	`meta-llama/Llama-3.1-8B-Instruct`
Preference / prompt dataset	`math-220k` (`open-r1/OpenR1-Math-220k`)
Training steps	500
Learning rate	5×10^{-6}
LR schedule	Cosine decay; warmup ratio 0.1 (dense) / 50 warmup steps (sparse)
Weight decay	0.0
Max gradient norm	0.1
GRPO β (KL-style coeff.)	0.025
Per-device train batch size	2
Gradient accumulation steps	4
Effective batch (1 GPU, per optimizer step)	8
`num_generations`	8
`generation_batch_size`	8
Max prompt length (tokens)	512
Max completion length (tokens)	2048
Reward / parsing profile	`llama_cot`
Precision	bf16
Optimizer (dense)	AdamW 8-bit (`adamw_8bit`)
Optimizer (sparse BSR)	SparseAdamW (`sparse_adamw`)
Checkpoint saves (HF)	Every 50 steps; retain 3
Activation gradient checkpointing	Enabled

E.2 `LLAMA-3.1-8B-INSTRUCT` DPO ON LIGHT-R1 AND TULU3

Table 8: DPO training hyperparameters — `Llama-3.1-8B-Instruct` on Light-R1 and Tülu-3. "Dense" is the reference Phase-1 run that sources the Oracle / warm-magnitude masks; "Sparse" is the Phase-3 training of the same model under a constructed mask.

Hyperparameter	**Dense (Phase 1)**	**Sparse (Phase 3)**
Base model	`meta-llama/Llama-3.1-8B-Instruct`	
Preference dataset	Light-R1 / Tülu-3 mixture (`allenai/llama-3.1-tulu-3-8b-preference-mixture`)	
Optimisation steps	500	500
Learning rate	5×10^{-7}	5×10^{-7}
LR schedule	Linear; warmup ratio 0.1	Linear; warmup ratio 0.1
Weight decay	0.0	0.0
DPO β	0.1	0.1
Max gradient norm	1.0	1.0
Per-device batch size	2	2
Gradient accumulation	16	64
GPUs	4 × NVIDIA H200	1 × NVIDIA H200
Effective batch	128	128
Max prompt length	1024	1024
Max sequence length	1024	1024
Precision	bf16	bf16
Optimiser	AdamW 8-bit (`adamw_8bit`)	SparseAdamW (mask-aware Triton)
Mask-aware decay	—	frozen weights are never decayed
Gradient checkpointing	enabled	enabled
Checkpoint cadence	every 25 steps (Light-R1) / 50 (Tülu-3)	every 50 steps
Save total limit	30 / 8	default
Auto-resume	`-resume_from_checkpoint auto`	`-resume_from_checkpoint auto`
Walltime cap	03:00:00	08:00:00
Random seed	42	42