The Surprising Effectiveness of Deleting Weights in LLM Post-Training

Jack Lu[1]* **Zhenbang Yang**[1]* **Mike Lasby**[2] **Tejas Pote**[2] **Yani Ioannou**[2] **Mengye Ren**[1]
[1]New York University [2]University of Calgary
https://agenticlearning.ai/deleting-weights/

Abstract

How little of a pretrained large language model (LLM) has to change for it to acquire new capabilities? We find that zeroing fewer than 0.05% of its weights, with no other modification, matches full fine-tuning (FFT) at scale across three LLM post-training regimes: reinforcement learning with verifiable rewards, supervised fine-tuning, and on-policy distillation. Our method, *Deletion Tuning*, learns a binary mask over the gate projection of transformer blocks. Two practical advantages follow: the trained mask is hundreds to thousands of times smaller than the corresponding FFT checkpoint, and Deletion Tuning forgets far less, which addresses a central challenge in continual learning. Under reinforcement learning, its prior-task accuracy stays near the base level while FFT, LoRA, and random sparse fine-tuning lose up to 5.4 points. To understand why masking alone suffices, we analyze the geometry of the learned weight changes under reinforcement learning using stable rank, spectrum drift, and principal-weight overlap. Compared with LoRA and random sparse fine-tuning, Deletion Tuning is closest to FFT across the three measures.

1 Introduction

To acquire new capabilities during LLM post-training, full fine-tuning can alter every parameter, LoRA [1] learns a low-rank update, and the Multiple Ticket Hypothesis (MTH) [2] and SHiRA [3] optimize values at selected weight positions. Weight deletion is typically used differently: pruning removes weights while preserving an already-trained model's capabilities for efficient inference [4, 5], and the Lottery Ticket Hypothesis finds sparse subnetworks that can be trained to match dense networks [6]. Unlike full fine-tuning, LoRA, and sparse updates, deletion writes no new values and only chooses which pretrained weights to keep. Recent work shows that fine-tuned solutions lie close to the pretrained model. RL fine-tuning changes only a small subnetwork [7], and task experts lie densely around pretrained weights [8]. We push this observation to its limit across reinforcement learning with verifiable rewards (RLVR), supervised fine-tuning (SFT), and on-policy distillation (OPD), and ask:

Can a pretrained LLM acquire new capabilities by learning only which weights to set to zero?

The answer is *yes* across all three post-training regimes. We propose *Deletion Tuning*, which learns only a binary mask over the gate-projection weights of transformer blocks. At deployment, each target weight either retains its pretrained value or is set to zero, so the learned binary mask is the entire adapter. We test Deletion Tuning with Group Relative Policy Optimization (GRPO) [9] for RLVR, standard SFT, and Self-Distillation Fine-Tuning (SDFT) [10] for OPD. Our experiments span two model families, eight backbones from 0.5B to 30B, and both in-distribution and out-of-distribution evaluation. At the largest model scale tested in each setting, Deletion Tuning matches full fine-tuning (FFT) across reasoning, domain knowledge, and tool-use tasks.

*Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

The binary-mask form brings two practical advantages: (i) *the adapter is compact*, with the trained mask hundreds to thousands of times smaller than the corresponding FFT checkpoint and smaller than even LoRA $r{=}1$ in our main GRPO comparison; and (ii) *Deletion Tuning forgets less*, with its forward KL remaining low and its prior-task accuracy staying near the base level across all three Llama scales. Under FFT, LoRA, and random sparse fine-tuning, forward KL grows and prior-task accuracy falls (Figure 2), consistent with forward KL as a predictor of forgetting [11]. Learning new tasks while keeping old ones makes Deletion Tuning a promising candidate for continual learning.

Finally, we examine why Deletion Tuning can match FFT with deletion alone. We confirm that the gains come from learning the mask rather than the Wanda prior used in its initialization. We also find that the geometry of Deletion Tuning's learned weight updates tracks FFT's more closely than LoRA or random sparse fine-tuning across stable rank, spectrum drift, and principal-weight overlap.

Our contributions are as follows:

- **Deletion alone matches FFT.** Deletion Tuning learns which gate-projection weights to delete (Section 4). It is on par with FFT at the largest scale we test in GRPO, SFT, and SDFT, up to a 30B mixture-of-experts model (Sections 5.2 and 5.6). Under GRPO, it also solves problems that the base model fails in all 128 sampled attempts (Section 5.2).
- **Deletion Tuning is compact and forgets less.** Its masks are orders of magnitude smaller than FFT checkpoints in all three regimes (Sections 5.2 and 5.6), and it keeps prior-task accuracy throughout training (Section 5.4).
- **Deletion Tuning's learned updates have FFT-like geometry.** Its weight updates resemble FFT's on three geometric measures (Section 5.5), which helps explain its accuracy.

2 Related Work

LLM post-training. We study three post-training regimes that differ in where the training signal comes from. Supervised fine-tuning (SFT) imitates fixed labeled targets [12–14]. Reinforcement learning optimizes a reward on the model's own samples, from PPO-based RLHF [15, 13] to GRPO [9] with verifiable rewards, the recipe behind DeepSeek-R1 [16]. With verifiable rewards, rule-based checkers or LLM verifiers score each sample [17]. DPO [18] instead turns preferences into a closed-form loss. On-policy distillation trains a student on its own rollouts to match a teacher's per-token distribution [19, 10, 20], and SDFT uses it to reduce catastrophic forgetting [21]. Self-distillation can also teach reasoning behaviors [22]. We use all three to test whether deletion works as a general post-training mechanism.

Sparsity and selective parameter updates. One-shot LLM pruning methods like Wanda [4] and SparseGPT [5] extend classical magnitude-based pruning [23, 24] to LLM scale, while continuous relaxations [25–27] make binary masks differentiable. Unstructured pruning can even retain the test-time scaling of reasoning models [28]. Earlier mask learning adapted vision backbones and encoder-only language models to downstream tasks such as image and sentence classification [29–31]. Recent mask methods target semi-structured LLM inference sparsity [32], refinement of a fine-tuned LLM [33], or vision-language backbones [34]. Deletion Tuning instead makes the mask the entire post-training update of a pretrained LLM across RLVR, SFT, and OPD, and it learns how many weights to delete. Sparse weight tuning writes float deltas at selected positions [35, 36, 2, 37, 3], additive adapters introduce trainable float parameters [1, 38–42], and robust fine-tuning methods restrict or regularize movement of pretrained weights under distribution shift [43, 44]. Deletion Tuning instead keeps every retained weight at its pretrained value and learns only which gate-projection weights to delete.

Understanding LLM fine-tuning. The classical Lottery Ticket Hypothesis [6] posits that randomly initialized networks contain sparse subnetworks that train to comparable accuracy in isolation. Recent work characterizes pretrained-model fine-tuning along three axes. *Density:* task-expert solutions densely populate the neighborhood of pretrained weights [8]. *Geometry:* Mukherjee et al. [7] report that RL fine-tuning naturally updates only 5 to 30% of parameters across PPO, GRPO, DPO, and several model families. Zhu et al. [45] show these updates land off the principal weights of W_0 while leaving the spectrum largely intact, where principal weights are the top-magnitude entries of the rank-k SVD reconstruction of W_0 [37]. *Forgetting:* Shenfeld et al. [11] establish that the forward

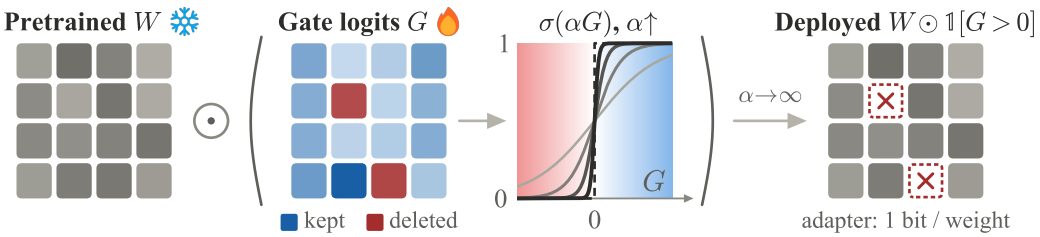

Figure 1: *Deletion Tuning* learns which pretrained weights to delete and writes no new values. The pretrained weights W stay frozen, and each is paired with a trainable logit in G, the only parameters that post-training updates. During training, each weight is multiplied by $\sigma(\alpha G)$, a smooth mask between 0 and 1. The sharpness α rises over training until the sigmoid becomes a hard threshold at $G = 0$. At deployment, weights with positive logits (blue) keep their pretrained values and weights with negative logits (red) are set to zero (crossed out), so the trained adapter is one bit per weight.

KL between fine-tuned and base policies on the new task accurately predicts forgetting. Deletion Tuning reproduces the off-principal update pattern and low spectrum drift. Deletion Tuning's low forward KL also predicts the strong prior-task retention we measure (Section 5.4).

3 Background

Our experiments use three post-training objectives.

Supervised fine-tuning (SFT). Given a dataset $\mathcal{D}$ of prompts x and target responses y of length $L = |y|$, SFT minimizes the negative log-likelihood of the target tokens, which are fixed in advance and never sampled from the model:

$$\mathcal{L}_{\text{SFT}}(\theta) \;=\; -\,\mathbb{E}_{(x,y)\sim\mathcal{D}} \sum_{j=1}^{L} \log \pi_\theta(y_j \mid x, y_{<j}). \tag{1}$$

Group Relative Policy Optimization (GRPO). For RLVR, we use GRPO [9]. For each prompt x, the policy before the update, $\pi_{\theta_{\text{old}}}$, samples a group of n responses $\{y^i\}_{i=1}^n$, and a verifier assigns each one a reward $r^i = r(x, y^i)$. Rather than learning a value function as in PPO [15], GRPO compares each response with the rest of its group, $A^i = (r^i - \text{mean}(r))/\text{std}(r)$, and gives this advantage to every token of y^i in PPO's clipped objective,

$$\mathcal{L}_{\text{GRPO}}(\theta) \;=\; -\,\mathbb{E}_{x,\,\{y^i\}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|y^i|} \sum_{j=1}^{|y^i|} \min\Big(\rho_j^i A^i,\; \text{clip}\big(\rho_j^i,\, 1{-}\epsilon,\, 1{+}\epsilon\big)\, A^i\Big), \tag{2}$$

where $\rho_j^i = \pi_\theta(y_j^i \mid x, y_{<j}^i) \,/\, \pi_{\theta_{\text{old}}}(y_j^i \mid x, y_{<j}^i)$ is the importance ratio for token j. GRPO optionally adds a KL penalty to the reference model, weighted by β.

Self-Distillation Fine-Tuning (SDFT). For on-policy distillation, we use SDFT [10], in which the model is its own teacher. The teacher's weights θ_T are an exponential moving average of the student's weights θ,

$$\theta_T \;\leftarrow\; (1-\gamma)\,\theta_T + \gamma\,\theta, \tag{3}$$

with $\gamma < 1$, and the teacher's prompt also includes d, a reference response to the same question, which the student never sees. For a response $y \sim \pi_\theta(\cdot \mid x)$ sampled by the student, the loss is the forward KL between the teacher's and the student's next-token distributions at each position:

$$\mathcal{L}_{\text{SDFT}}(\theta) \;=\; \mathbb{E}_{x,\,y\sim\pi_\theta(\cdot|x)} \sum_{j=1}^{|y|} \text{KL}\Big(\pi_{\theta_T}(\cdot \mid d, x, y_{<j}) \;\big\|\; \pi_\theta(\cdot \mid x, y_{<j})\Big). \tag{4}$$

4 Deletion Tuning

Deletion Tuning parameterizes fine-tuning as one trainable scalar logit per pretrained weight at a chosen subset of linear layers, with the base model frozen. During training, a sigmoid of each logit is annealed toward a step function, and at deployment each logit becomes a single bit deciding whether the corresponding weight is kept or deleted (set to zero). In pruning terms, Deletion Tuning performs learned unstructured pruning to fine-tune the model.

Notation. Let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote a target weight matrix in the pretrained model and $G \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ a trainable logit matrix of the same shape, where the sign of G_{ij} at deployment decides whether W_{ij} is kept (positive) or zeroed (negative). We mask one such W per target layer, each with its own independent G. In all our experiments, the target is the gate projection of transformer blocks (a single MLP weight per block). We follow GLU-specific pruning methods by treating the gate/up/down pathway as the main coupled sparsification structure [46, 47]: in the Llama and Qwen2.5 backbones we evaluate, SwiGLU MLPs take the form $\mathrm{down}(\mathrm{SiLU}(\mathrm{gate}(x)) \odot \mathrm{up}(x))$, so the gate projection multiplicatively controls the intermediate channels routed through the down projection.

Forward pass. At training step $t \in \{0, 1, \ldots, T\}$ (with T the total step count), the layer applies a sigmoid relaxation of the binary masking decision per weight:

$$\widetilde{W}(t) \;=\; W \,\odot\, \sigma\big(\alpha(t)\, G\big), \qquad \sigma(x) = \frac{1}{1+e^{-x}}. \tag{5}$$

We call $\sigma(\alpha(t)\, G)$ the *soft mask* and $\mathbf{1}[G > 0]$ the *hard mask*. The sharpness $\alpha(t) > 0$ sets how steep the sigmoid is around $G = 0$. A smaller α widens the band of logits whose soft mask lies strictly between 0 and 1, and raising α pulls the soft mask onto the hard one by the end. We use a linear ramp from $\alpha_{\text{start}} = 10^3$ to $\alpha_{\text{end}} = 10^4$, which we call *mask sharpening*:

$$\alpha(t) \;=\; \alpha_{\text{start}} \;+\; \tfrac{t}{T}(\alpha_{\text{end}} - \alpha_{\text{start}}). \tag{6}$$

At evaluation, the layer applies the hard mask:

$$\widehat{W} \;=\; W \,\odot\, \mathbf{1}\big[G > 0\big]. \tag{7}$$

Gradients are computed through the deterministic soft mask in Eq. (5), while the hard mask is used for evaluation and for generating rollouts in on-policy runs. By the end of training, the two agree: deploying the hard mask instead of the soft one changes accuracy by at most 0.08 points.

Wanda prior initialization. We initialize G from the Wanda one-shot pruning score [4], an importance signal validated as a strong predictor of which base weights are safe to zero. The Wanda score of a weight is its magnitude times the norm of its input activation, $s_{ij} = |W_{ij}| \cdot \|X_j\|_2$, where X_j collects the activations of input channel j in one forward pass over calibration prompts from the task's training distribution. Specifically, we sample each gate logit from one of two Gaussians depending on whether the weight is above or below the η-quantile of s:

$$G_{ij} \;\sim\; \begin{cases} \mathcal{N}(0.03,\; 0.01^2) & \text{if } s_{ij} \text{ is in the top } 1{-}\eta \text{ fraction (high importance),} \\ \mathcal{N}(-0.03,\; 0.01^2) & \text{otherwise (low importance).} \end{cases} \tag{8}$$

At our default $\eta = 1/2048$, used in every main experiment, this zeros approximately $\eta\,\Phi(3) + (1-\eta)\,\Phi(-3) \approx 0.18\%$ of gate-projection weights under the hard mask at initialization, where Φ is the standard normal CDF.

Training objective. We instantiate the policy π_θ of Section 3 by replacing W with $\widetilde{W}(t)$ and keeping pretrained weights elsewhere, so the gate logits G are the only trainable parameters in the model. The training loss $\mathcal{L}_{\text{task}}(\widetilde{W}(t))$ is the regime's own objective from Eqs. (1) to (4), plus the MaskLLM sparse weight regularizer $-\lambda\,\|\widetilde{W}(t)\|_2^2$, which mildly biases the mask toward retaining higher-magnitude weights. We do not constrain G to any fixed sparsity level, leaving the training process to decide it.

- **SFT.** The loss is $\mathcal{L}_{\text{SFT}}$, with $\lambda = 10^{-5}$.

- **GRPO.** The loss is $\mathcal{L}_{\text{GRPO}}$, with $\lambda = 10^{-5}$. Policy rollouts use the hard mask of Eq. (7), while gradients flow into G through the soft mask of Eq. (5).
- **SDFT.** The loss is $\mathcal{L}_{\text{SDFT}}$, with $\lambda = 0$, since the EMA teacher already provides an anchor. Rollouts use the hard mask as in GRPO, and the EMA teacher of Eq. (3) specializes to a teacher gate $G_T \leftarrow (1-\gamma)\, G_T + \gamma\, G$, so the teacher's frozen base weights W stay the same as the student's.

Compact storage and ease of deployment. The trained adapter is the hard mask $\mathbf{1}[G > 0]$ at each masked layer, which takes one bit per masked pretrained weight before compression. We bit-pack the booleans with NumPy and compress the bit stream with zstd [48]. Because fewer than 0.05% of all weights are deleted in every GRPO run of Table 1, the bit stream is highly redundant, and zstd compresses it to 1.0 MB at Llama1B and 8.4 MB at Llama8B. This gives Deletion Tuning a practical deployment advantage in compact storage and cheap adapter switching: changing tasks only loads a compact bitmask and scatters zeros into the base weights, rather than loading a separate full model checkpoint.

Connection to MaskLLM. MaskLLM [32] learns $2{:}4$ masks for inference speedup with a Gumbel-Softmax selector over $N{:}M$ candidate patterns. Deletion Tuning has a different goal, expanding model capability, and a different form, with one deterministic logit per weight and no fixed sparsity pattern (Appendix F relates the two formally). Deletion Tuning also adopts MaskLLM's Wanda prior initialization (Eq. (8)) and its sparse weight regularizer.

5 Experiments

5.1 Experimental Setup

Tasks, benchmarks, and models. We study three post-training regimes on two model families, referring to a family as Llama or Qwen and to a size by its suffix (e.g., Llama1B, Qwen7B). Evaluation uses greedy decoding, and we report the best validation accuracy across saved checkpoints.

- **GRPO, in distribution.** We train and evaluate on GSM8K [49] with Llama1B, Llama3B, and Llama8B (Llama3.2-1B/3B-Instruct and Llama3-8B-Instruct) [50], validating every 25 optimizer steps.
- **GRPO, out of distribution.** Following SimpleRL-Zoo [51], we train on its math corpus and report the mean over six held-out benchmarks (GSM8K, MATH500, Minerva Math, OlympiadBench, AIME24, AMC23; per-benchmark results in Appendix C) with Qwen0.5B, Qwen3B, and Qwen7B (Qwen2.5 base) [52]. At larger scale, we add Qwen14B (Qwen2.5-14B) and the Qwen3-30B-A3B mixture-of-experts model (30B total and 3B active parameters). We validate every 10 optimizer steps.
- **SFT.** We train Llama1B and Llama3B (Llama3.2-1B/3B-Instruct) separately on each of the 14 MMLU-Pro [53] subjects and 23 BBEH [54] subtasks, validate at the end of training, and average accuracy within each suite.
- **SDFT.** We follow the ToolAlpaca [55] tool-use and Science settings of Shenfeld et al. [10] with Qwen7B (Qwen2.5-7B-Instruct), reporting exact-match accuracy under each task's evaluator. Science is the Chemistry L-3 subset of SciKnowEval [56]. We validate every 100 optimizer steps on ToolAlpaca and every 25 on Science.

Training pipelines. All runs use AdamW with fp32 master weights, bf16 mixed precision, and gradient checkpointing.

- **GRPO.** We use verl [57] with a constant learning rate. GSM8K uses prompt batch 64, $n{=}4$ rollouts, response length 4096, 3 epochs, and no KL penalty, while SimpleRL-Zoo uses prompt batch 256, $n{=}8$ rollouts, response length 4096, 150 steps, and KL penalty 10^{-4}.
- **SFT.** We minimize cross-entropy with a constant learning rate for $T{=}400$ steps per task at effective batch 4, with a $75/25$ train/eval split (per-task counts in Appendix D).
- **SDFT.** We follow Shenfeld et al. [10]: cosine schedule with 0.1 warmup ratio, prompt batch 32, 2 epochs, EMA mixup $\gamma = 0.01$, and one demonstration prepended to the teacher's input.

Baselines. All three baselines use the same data and number of steps as Deletion Tuning.

Table 1: Deletion Tuning has the smallest adapter at every scale and matches FFT at the largest backbone of each family. Best-checkpoint validation accuracy (%) and adapter size (MB) under GRPO, on GSM8K for Llama and on the six-benchmark SimpleRL-Zoo mean for Qwen. The baselines are full fine-tuning (FFT), low-rank adapters (LoRA), and random sparse fine-tuning (MTH).

	Llama, GSM8K						Qwen, SimpleRL-Zoo					
	1B		3B		8B		0.5B		3B		7B	
Method	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**
Base	1.21	0	53.53	0	60.27	0	10.44	0	37.73	0	40.18	0
FFT	60.50	4714	**82.49**	12256	83.24	30633	19.01	1885	**44.35**	11772	51.89	29051
LoRA r=1	59.29	2.7	82.39	5.8	83.38	10.0	20.30	2.1	43.55	7.1	52.13	9.6
LoRA r=4	60.58	10.8	82.44	23.2	82.87	40.0	20.30	8.4	43.59	28.6	51.70	38.5
MTH 99%	**60.73**	68.2	80.53	177.4	83.31	443.6	**21.84**	27.3	44.16	170.4	52.03	420.7
MTH 99.95%	58.61	4.1	82.17	10.7	83.07	26.6	20.16	1.7	43.87	13.7	51.24	25.3
Deletion Tuning	54.44	**1.0**	81.49	**3.1**	**83.55**	**8.4**	17.64	**0.5**	41.98	**2.7**	**52.32**	**6.9**

- **FFT.** Full fine-tuning trains all weights and is the reference we aim to match. Its learning rate is 10^{-6} on GSM8K, 5×10^{-7} on SimpleRL-Zoo, 10^{-6} to 3×10^{-6} on SFT (Appendix H), and 5×10^{-5} on SDFT as in Shenfeld et al. [10].
- **LoRA.** We train rank-1 and rank-4 adapters [1] on all seven projections in each block (q/k/v/o and gate/up/down) with learning rate 10^{-4}.
- **MTH.** The closest sparse-update method [2] trains a random 1% or 0.05% of all parameters (99% and 99.95% sparsity), with the learning rates recommended by Adewuyi et al. [2] (10^{-4} and 10^{-3}).

Deletion Tuning implementation. We compute the Wanda scores once per task, on 2048 training prompts for GRPO and SFT and on 32 for SDFT.

- **GRPO.** Every main run uses $\eta = 1/2048$, learning rate 3×10^{-4}, $(\alpha_{\text{start}}, \alpha_{\text{end}}) = (10^3, 10^4)$, and $\lambda = 10^{-5}$, with no tuning per benchmark or model size (Section 5.7 varies η, learning rate, and λ).
- **SFT.** We keep the same η and λ, with learning rate 5×10^{-4} and $(\alpha_{\text{start}}, \alpha_{\text{end}}) = (10^2, 10^3)$ for every subject and subtask.
- **SDFT.** We set $\eta = 1/2048$ and $\lambda = 0$ and use a hard teacher mask, with learning rate 5×10^{-3} and $(10^2, 10^4)$ on ToolAlpaca and 3×10^{-3} and $(10^2, 10^3)$ on Science.

Hardware. We train on H200 GPUs, using one GPU for backbones up to 3B and multiple GPUs with FSDP [58] for larger models. SFT and the post-hoc analyses use L40S GPUs.

5.2 Results on GRPO Post-Training

We train Deletion Tuning and three baselines (FFT, LoRA at ranks 1 and 4, MTH at 99% and 99.95%) on the two GRPO settings of Section 5.1. Table 1 reports best-checkpoint validation accuracy and bytes-on-disk adapter size for six backbones from 0.5B to 8B across two model families. Every Deletion Tuning run deletes fewer than 0.05% of all weights (Appendix B). Appendix A defines adapter size per method.

Deletion Tuning matches FFT at 7B and 8B. At the largest backbone in each family, Deletion Tuning scores 83.55% against FFT's 83.24% on Llama8B GSM8K and 52.32% against 51.89% on Qwen7B SimpleRL-Zoo. Tuning FFT's learning rate and KL coefficient does not change this (Appendix H). At these two backbones, Deletion Tuning is also competitive with both LoRA ranks and both MTH sparsity levels (Table 1). Since Deletion Tuning can only delete pretrained weights, this restriction may act as a mild regularizer, which could explain why Deletion Tuning matches FFT here. The same restriction pays off more clearly for retention, where Deletion Tuning keeps the prior-task accuracy that FFT loses (Section 5.4).

Table 2: Deletion Tuning matches FFT on dense and mixture-of-experts models from 7B to 30B with an adapter thousands of times smaller. Best-checkpoint validation accuracy (%) and adapter size (MB) under GRPO. Qwen14B and Qwen3-30B-A3B use the SimpleRL-Zoo setup of Qwen7B.

	Llama8B GSM8K		**Qwen7B** SimpleRL-Zoo		**Qwen14B** SimpleRL-Zoo		**Qwen3-30B-A3B** SimpleRL-Zoo	
Method	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**
Base	60.27	0	40.18	0	44.37	0	49.41	0
FFT	83.24	30633	51.89	29051	54.87	56343	58.43	116471
Deletion Tuning	**83.55**	**8.4**	**52.32**	**6.9**	**55.43**	**11.4**	**58.54**	**29.6**

Table 3: Deletion Tuning acquires new capabilities. Pass@k (%), the share of problems solved within k attempts, estimated from 128 samples per problem at temperature 1.0. A lead at large k means solving problems the base model misses even with many attempts.

	Llama8B, GSM8K					**Qwen7B, SimpleRL-Zoo**				
Method	k=1	2	8	32	128	k=1	2	8	32	128
Base	53.20	66.75	83.64	92.40	95.29	27.35	38.75	56.18	66.26	74.01
Deletion Tuning	**81.75**	**88.51**	**95.75**	**98.44**	**99.62**	**49.39**	**53.22**	**64.64**	**73.23**	**80.08**

The gap to FFT closes with scale. At smaller backbones, Deletion Tuning recovers 90% to 99% of FFT's accuracy. At larger scale, Deletion Tuning matches FFT on Qwen14B and on the Qwen3-30B-A3B mixture-of-experts model (Table 2). For the mixture-of-experts model, we mask the gate projection inside each of its 128 experts in every layer and leave the router unchanged. These results suggest that deletion becomes more effective as models grow.

Deletion Tuning acquires new capabilities. Prior sparsity methods aim to preserve base performance under aggressive pruning [32, 4, 5]. Deletion Tuning instead uses sparsity as the learning mechanism itself. On pass@k (Table 3), Deletion Tuning leads the base model at every k. Even at k=128, the base model's pass@k stays 4.3 and 6.1 points below Deletion Tuning's on Llama8B and Qwen7B.

Smallest adapter at every scale. Deletion Tuning's adapter is 0.5 MB at Qwen0.5B and 8.4 MB at Llama8B, 3,600 to 4,700× smaller than the FFT checkpoints and the smallest of any adapter family at every scale. The closest baselines by size are LoRA r=1 and MTH at 99.95% sparsity. They are still often several times larger than Deletion Tuning, and MTH at 99% is more than 50× larger in every run. After compression, the mask takes about 0.03 bits per target weight, while every other method stores floating-point values for the weights it changes.

5.3 Learning the Mask Drives the Gains

Do Deletion Tuning's gains over the base model come from learning the mask itself, beyond the Wanda prior that initializes the gates? We test this directly. For each run in Table 1, we apply Wanda one-shot pruning without GRPO training at 8 sparsity levels from $1/2$ to $1/8192$. We evaluate each level on the same validation set and record the *best* accuracy (Appendix E reports every level). Table 4 compares this best-Wanda score with the base model and the trained Deletion Tuning mask.

The best Wanda mask stays within 2.4 points of the base on all six backbones, while Deletion Tuning, starting from the same Wanda scores, gains up to 53.2 points.

5.4 Deletion Tuning Forgets Less

Forward KL on the new task predicts how much a fine-tuned model forgets [11]. We hypothesize that deletion keeps the perturbation to the base policy small, because each edit can only set a weight to zero and training chooses which few weights to edit, yielding low forward KL and therefore little forgetting. Following Shenfeld et al. [11]'s evaluation, Figure 2 tracks forward KL and prior-task accuracy over

Table 4: Wanda alone stays near the base model. Validation accuracy (%) of the base model, of Wanda one-shot pruning at the best of eight sparsity levels, and of the best checkpoint of the Deletion Tuning mask trained with GRPO.

	Llama, GSM8K			Qwen, SimpleRL-Zoo		
Method	1B	3B	8B	0.5B	3B	7B
Base	1.21	53.53	60.27	10.44	37.73	40.18
Wanda (best η)	1.59	52.46	60.65	11.42	38.70	42.49
Deletion Tuning	**54.44**	**81.49**	**83.55**	**17.64**	**41.98**	**52.32**

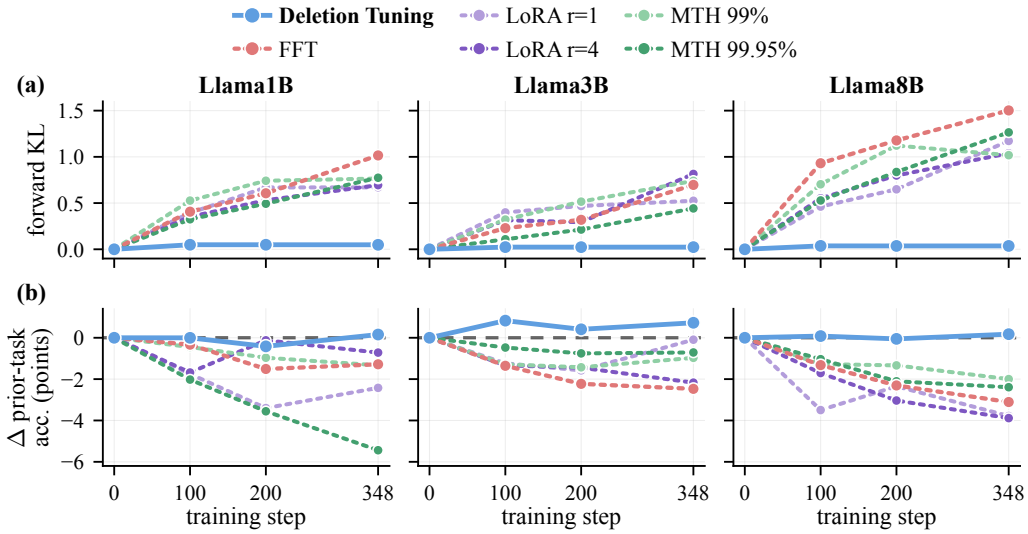

Figure 2: Deletion Tuning keeps prior-task accuracy at every Llama scale while the baselines drift. (a) Forward KL between the trained and base policies on GSM8K during GRPO training, the forgetting predictor of Shenfeld et al. [11]. (b) Change in mean accuracy on a six-task prior-task suite from the base model (dashed line).

training for FFT, LoRA, MTH, and Deletion Tuning at the three Llama scales. The prior-task suite has six benchmarks: HellaSwag, MMLU, TruthfulQA, WinoGrande, IFEval, and HumanEval.

By step 348, FFT's forward KL reaches 1.0 on Llama1B and 1.5 on Llama8B, and its prior-task accuracy ends 1.3 to 3.1 points below base. Deletion Tuning's forward KL stays at or below 0.05 at all three scales, and its prior-task accuracy stays within one point of base throughout training. LoRA and MTH train only a small fraction of parameters, yet their forward KL reaches 0.4 to 1.3 and their prior-task accuracy falls up to 5.4 points below base. MTH at 99.95% sparsity trains 0.05% of all parameters, comparable to the 0.035% to 0.046% that Deletion Tuning deletes at these scales (Table 7), suggesting that the small number of edited weights does not explain Deletion Tuning's retention, and that deletion itself keeps forgetting low.

5.5 Why Deletion Tuning Works

How can binary masking alone (Section 5.2) match FFT under GRPO at scale? We compare the weight updates of Deletion Tuning and the three baselines at a saved checkpoint on three geometric measures.

Update geometry. Let W_0 denote the pretrained value of a weight matrix, W_+ its effective deployed value at the saved checkpoint (Llama8B step 348, Qwen7B step 100), and $\Delta W = W_+ - W_0$ the weight delta. For FFT, W_+ is the trained matrix, and for the adapters it is the merged or masked matrix that runs at inference. We report three quantities, each averaged over all attention and MLP linear layers:

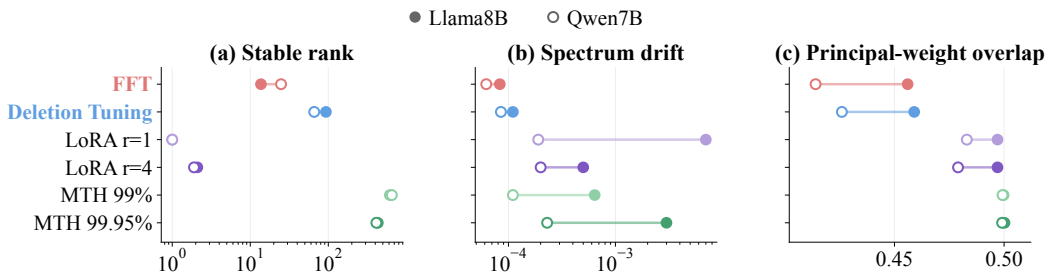

Figure 3: Deletion Tuning (blue) tracks FFT (red) most closely across three update-geometry metrics. Values at the saved GRPO checkpoint for Llama8B (filled) and Qwen7B (open): (a) stable rank of ΔW, (b) spectrum drift, and (c) principal-weight overlap. Overlap below the random baseline of 0.5 is the RLVR signature reported by Zhu et al. [45].

- **Stable rank** of ΔW is $\|\Delta W\|_F^2/\|\Delta W\|_2^2 = \sum_{m=1}^{q} \sigma_m^2/\sigma_1^2$, where $q = \min(d_{\text{out}}, d_{\text{in}})$ and $\sigma_1 \geq \cdots \geq \sigma_q$ are the singular values of ΔW. It counts how many directions the update uses, independent of its scale: it is close to 1 when one direction dominates and close to k when k directions carry similar energy.
- **Spectrum drift** is the normalized singular-value shift $\|\sigma(W_+) - \sigma(W_0)\|_2/\|\sigma(W_0)\|_2$ [45], where $\sigma(W)$ lists the singular values of W. It is small when training leaves the singular values close to where they started.
- **Principal-weight overlap** measures the fraction of changed weight positions that fall on what Zhu et al. [45] call the base model's *principal weights*. The *principal-weight mask* $M_{\text{princ}} = \text{Top}_{50\%}\big(|W_0^{(128)}|\big)$ marks the top 50% entries by magnitude of $W_0^{(128)}$, the best rank-128 approximation of W_0. The *update mask* M_{update} marks positions where the deployed matrix differs from the base by $|W_+(i,j) - W_0(i,j)| > 10^{-3} \cdot \max(|W_0(i,j)|, |W_+(i,j)|)$, the relative tolerance of Zhu et al. [45], which ignores changes too small to register in bf16. The overlap is $|M_{\text{princ}} \cap M_{\text{update}}|/|M_{\text{update}}|$. Update positions spread uniformly at random would score 0.5, the density of M_{princ}.

Figure 3 shows the three metrics for Deletion Tuning and every baseline setting, and Deletion Tuning is the closest to FFT on the three together. Its stable rank of 66 to 93 sits far below MTH's 410 to 650 and above LoRA's 1 to 2. Its spectrum drift stays within $1.4\times$ of FFT's, while LoRA r=1 drifts up to $84\times$ more than FFT on Llama8B. Its principal-weight overlap of 0.43 to 0.46 matches FFT's sub-random 0.41 to 0.46. LoRA stays near the random baseline at 0.48 to 0.50, and MTH sits at 0.50.

Interpretation. Low spectrum drift and sub-random principal-weight overlap are the RLVR signature reported by Zhu et al. [45]: RLVR leaves the singular values of each weight matrix nearly unchanged and steers its updates away from the principal weights. LoRA caps the rank of its update at r, and MTH trains weights chosen at random before training. Deletion Tuning has neither restriction, since it learns which weights to change and has no rank cap. These two freedoms help explain how Deletion Tuning reaches FFT-like geometry even though it never sets a weight to a new non-zero value.

5.6 Generalizing Beyond GRPO: Results on SFT and SDFT

Does deletion also work when the training signal is denser or comes from outside the model? GRPO samples from the model itself and rewards whole responses, and RL updates of this kind are known to be sparse [7]. SFT instead imitates fixed external targets token by token and produces dense updates [7], which makes it a harder test for an adapter that can only delete. SDFT sits in between: the student samples its own responses, and a teacher supplies a target distribution at every token. We test Deletion Tuning with SFT on knowledge and reasoning benchmarks (MMLU-Pro and BBEH) and with SDFT on tool use (ToolAlpaca) and science questions (Science), using the setups of Section 5.1.

Under SFT, Deletion Tuning matches FFT at 3B on both MMLU-Pro and BBEH (Table 5). Under SDFT, Deletion Tuning matches FFT on both Qwen7B tasks (Table 6). In both regimes, its adapters are 600 to $4{,}700\times$ smaller than FFT checkpoints.

Table 5: Under SFT, Deletion Tuning matches FFT at 3B. Best-checkpoint evaluation accuracy (%), averaged over the 14 MMLU-Pro subjects and the 23 BBEH subtasks, and adapter size (MB) for Llama1B and Llama3B.

	MMLU-Pro				BBEH			
	1B		3B		1B		3B	
Method	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**	**Acc.**	**Size**
Base	10.37	0	12.63	0	2.17	0	8.87	0
FFT	16.97	4714	21.19	12256	**8.35**	4714	**11.39**	12256
Deletion Tuning	**18.62**	**1.0**	**22.28**	**3.1**	5.65	**1.0**	**11.39**	**3.1**

Table 6: Under SDFT, Deletion Tuning matches FFT on Qwen7B. Best-checkpoint exact-match accuracy (%) and adapter size (MB) on ToolAlpaca tool use and Science.

	ToolAlpaca		Science	
Method	**Acc.**	**Size**	**Acc.**	**Size**
Base	41.24	0	32.35	0
FFT	68.04	29051	**71.20**	29051
Deletion Tuning	**69.07**	**27.1**	70.61	**47.1**

5.7 Deletion Tuning Is Robust to Hyperparameter Changes

Deletion Tuning works across a range of values for its hyperparameters: initial sparsity η, learning rate lr, and sparse-weight regularization strength λ. We sweep $\eta \in \{1/2048, 1/4096, 1/8192\}$, $lr \in \{10^{-4}, 3\times10^{-4}\}$, and $\lambda \in \{0, 10^{-5}\}$ on Llama8B GSM8K and Qwen7B SimpleRL-Zoo (Tables 13 and 14 in Appendix G). Every setting lands within 1.6 points of the best run on Llama8B and within 3.2 points on Qwen7B, and the single configuration we use for all main runs is at or near the best setting at both backbones. Varying the initial sparsity η by $4\times$ changes accuracy by at most 0.8 points on Llama8B and 1.7 points on Qwen7B at fixed lr and λ, so the default configuration is a reasonable starting point for new tasks.

6 Conclusion

We introduced *Deletion Tuning*, which fine-tunes a model by learning only a binary mask over its gate projections. At the largest scale we test in each regime (Qwen14B and the Qwen3-30B-A3B mixture-of-experts model under GRPO, Qwen7B under SDFT, and Llama3B under SFT), it matches full fine-tuning. The mask is smaller than a rank-1 LoRA and hundreds to thousands of times smaller than an FFT checkpoint. In our GRPO retention runs, Deletion Tuning keeps prior-task accuracy near the base level throughout training, while FFT, LoRA, and MTH lose up to 5.4 points. Its weight updates also resemble FFT's more closely than those of LoRA or MTH across stable rank, spectrum drift, and principal-weight overlap, which helps explain how deletion alone can match FFT. These results suggest that post-training can work by selection alone, keeping nearly every pretrained weight and choosing the few to delete.

Limitations and Future Work. Because Deletion Tuning forgets little, a natural next test is continual learning over a sequence of tasks. Training keeps one logit per masked weight, so Deletion Tuning uses less training memory than FFT but more than LoRA. Restricting the logits to a small set of candidate weights could close this gap. The masks are also small and discrete, which aids interpretability: comparing which weights different tasks delete could localize the changes that post-training makes to a model. Masks for different tasks might also be combined, for example by taking their union, to merge skills without further training. Finally, restricting the mask to $N{:}M$, channel-wise, or block patterns could let one mask both adapt and compress the model.

Acknowledgments and Disclosure of Funding

We thank members of the NYU Agentic Learning AI Lab and the Calgary ML Lab for helpful discussions. JL is supported by an NSERC PGS-D Scholarship. This work was supported in part by Visko Platform, Pionex, a Google TPU Award, a Google ML and Systems Junior Faculty Award, and the Institute of Information & Communications Technology Planning & Evaluation (IITP) with a grant funded by the Ministry of Science and ICT (MSIT) of the Republic of Korea in connection with the Global AI Frontier Lab International Collaborative Research Agreement (No. RS-2024-00469482 & RS-2024-00509279). Computational support was provided by NYU High Performance Computing through its resources, services, and staff expertise. This research was enabled in part by support provided by the Digital Research Alliance of Canada (alliancecan.ca).

ML, TP, and YI gratefully acknowledge the support of Alberta Innovates (ALLRP 577350-22, ALLRP 600038-24), the Natural Sciences and Engineering Research Council of Canada (NSERC) (RGPIN-2022-03120, DGECR-2022-00358), Defence Research and Development Canada (DGDND-2022-03120), and NSERC/Agence Nationale de la Recherche (ANR) (ALLRP 602719-24). This project was undertaken thanks to funding from IVADO and the Canada First Research Excellence Fund. ML additionally acknowledges the support of RBC Borealis through the RBC Borealis AI Global Fellowship Award.

Funding transparency statement. *Funding in direct support of this work:* Gift from Visko Platform, Pionex, a Google TPU Award, a Google ML and Systems Junior Faculty Award, and grants from IITP and MSIT of the Republic of Korea through the Global AI Frontier Lab (RS-2024-00469482, RS-2024-00509279) (MR); NSERC PGS-D Scholarship (JL); compute from NYU High Performance Computing and the Digital Research Alliance of Canada. ML, TP, and YI were directly supported by Alberta Innovates, the Natural Sciences and Engineering Research Council of Canada (NSERC), Agence Nationale de la Recherche (ANR), Defence Research and Development Canada, and IVADO. *Additional revenues:* Other funding received by the authors: MR received the NYU-KAIST Award A25-0081-002, student support from the Meta AI Mentorship Program, Azure credits from the Microsoft Accelerating Foundation Models Research program, Google Cloud credits, Anthropic API credits, and compute from the Center for AI Safety. JL was a research intern at NVIDIA in 2026 and has been a visiting researcher at Meta since September 2026. ML held a research internship at Cerebras in 2025 and has been employed full time at Cerebras since August 2026.

References

[1] Edward 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 *ICLR*, 2022.

[2] Israel Adewuyi, Solomon Okibe, and Vladmir Ivanov. The multiple ticket hypothesis: Random sparse subnetworks suffice for RLVR. In *ICML Workshop on High-dimensional Learning Dynamics*, 2026.

[3] Kartikeya Bhardwaj, Nilesh Prasad Pandey, Sweta Priyadarshi, Viswanath Ganapathy, Shreya Kadambi, Rafael Esteves, Shubhankar Borse, Paul Whatmough, Risheek Garrepalli, Mart Van Baalen, Harris Teague, and Markus Nagel. Sparse high rank adapters. In *NeurIPS*, 2024.

[4] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *ICLR*, 2024.

[5] Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *ICML*, 2023.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[7] Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tür, and Hao Peng. Reinforcement learning finetunes small subnetworks in large language models. In *NeurIPS*, 2025.

[8] Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. In *ICML*, 2026.

[9] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[10] Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. In *ICML*, 2026.

[11] Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's razor: Why online reinforcement learning forgets less. In *ICLR*, 2026.

[12] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *ICLR*, 2022.

[13] 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 *NeurIPS*, 2022.

[14] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A strong, replicable instruction-following model, 2023. Stanford CRFM tech report.

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

[16] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025.

[17] Jack Lu, Ryan Teehan, Jinran Jin, and Mengye Ren. When does verification pay off? A closer look at LLMs as solution verifiers. In *ICLR Workshop on AI with Recursive Self-Improvement*, 2026.

[18] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

[19] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *ICLR*, 2024.

[20] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *ICML*, 2026.

[21] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

[22] Zayne Sprague, Jack Lu, Manya Wadhwa, Sedrick Keh, Mengye Ren, and Greg Durrett. SkillFactory: Self-distillation for learning cognitive behaviors. In *ICLR*, 2026.

[23] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015.

[24] Michael H. Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *ICLR Workshop*, 2018.

[25] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *ICLR*, 2018.

[26] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[27] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *ICLR*, 2017.

[28] Ocean Monjur, Shahriar Kabir Nahin, and Anshuman Chhabra. Revisiting the effectiveness of LLM pruning for test-time scaling. In *EMNLP Findings*, 2026.

[29] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, 2018.

[30] Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. In *EMNLP*, 2020.

[31] Evani Radiya-Dixit and Xin Wang. How fine can fine-tuning be? Learning efficient language models. In *AISTATS*, 2020.

[32] 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 *NeurIPS*, 2024.

[33] Mingyuan Zhang, Yue Bai, Huan Wang, Yizhou Wang, Qihua Dong, Yitian Zhang, and Yun Fu. Boosting large language models with mask fine-tuning. *arXiv preprint arXiv:2503.22764*, 2025.

[34] Kecheng Zheng, Wei Wu, Ruili Feng, Kai Zhu, Jiawei Liu, Deli Zhao, Zheng-Jun Zha, Wei Chen, and Yujun Shen. Regularized mask tuning: Uncovering hidden knowledge in pre-trained vision-language models. In *ICCV*, 2023.

[35] Demi Guo, Alexander M. Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *ACL*, 2021.

[36] Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. In *NeurIPS*, 2021.

[37] Zihang Liu, Tianyu Pang, Oleg Balabanov, Chaoqun Yang, Tianjin Huang, Lu Yin, Yaoqing Yang, and Shiwei Liu. LIFT the veil for the truth: Principal weights emerge after rank reduction for reasoning-focused supervised fine-tuning. In *ICML*, 2025.

[38] John X. Morris, Niloofar Mireshghallah, Mark Ibrahim, and Saeed Mahloujifar. Learning to reason in 13 parameters. *arXiv preprint arXiv:2602.04118*, 2026.

[39] Qi Sun, Edoardo Cetin, and Yujin Tang. Transformer-Squared: Self-adaptive LLMs. In *ICLR*, 2025.

[40] Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzębski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, 2019.

[41] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.

[42] Jack Lu, Ryan Teehan, Zhenbang Yang, and Mengye Ren. Context tuning for in-context optimization. In *ICML*, 2026.

[43] Yoonho Lee, Annie S. Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. In *ICLR*, 2023.

[44] Junjiao Tian, Chengyue Huang, and Zsolt Kira. Rethinking weight decay for robust fine-tuning of foundation models. In *NeurIPS*, 2024.

[45] Hanqing Zhu, Zhenyu Zhang, Hanxian Huang, DiJia Su, Zechun Liu, Jiawei Zhao, Igor Fedorov, Hamed Pirsiavash, Jinwon Lee, David Z. Pan, Zhangyang Wang, Yuandong Tian, and Kai Sheng Tai. The path not taken: RLVR learns off the principals. In *NeurIPS*, 2026.

[46] Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *NeurIPS*, 2023.

[47] Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Dependency-aware semi-structured sparsity of GLU variants in large language models. *Transactions on Machine Learning Research*, 2025.

[48] Yann Collet and Murray Kucherawy. Zstandard compression and the `application/zstd` media type, 2021. IETF RFC 8878.

[49] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[50] Llama Team, AI @ Meta. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[51] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. SimpleRL-Zoo: Investigating and taming zero reinforcement learning for open base models in the wild. In *COLM*, 2025.

[52] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[53] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *NeurIPS Datasets and Benchmarks*, 2024.

[54] Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-bench extra hard. In *ACL*, 2025.

[55] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

[56] Kehua Feng, Keyan Ding, Weijie Wang, Xiang Zhuang, Zeyuan Wang, Ming Qin, Yu Zhao, Jianhua Yao, Qiang Zhang, and Huajun Chen. SciKnowEval: Evaluating multi-level scientific knowledge of large language models. *arXiv preprint arXiv:2406.09098v1*, 2024.

[57] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *EuroSys*, 2025.

[58] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 2023.

A Adapter Size

The **Size** entries in Tables 1, 2, 5, and 6 (in binary units, 1 GB $= 1024$ MB) are lower bounds on the bytes on disk needed to rebuild the trained model from the public base checkpoint, computed the same way for every method. We leave out training overhead that deployment discards (file headers, the Adam moments m and v, and learning-rate scheduler and RNG state), which would otherwise dominate the smallest adapters.

Full fine-tuning (FFT). A fine-tuned LLM checkpoint at fp32:

$$\text{size}(\text{FFT}) = N_{\text{total}} \cdot 4 \,\text{bytes},$$

where N_{total} is the total parameter count. We use fp32 to match the trained master weights, which we store in fp32 for every adapter family.

LoRA. The B and A low-rank factor matrices, no bias terms:

$$\text{size}(\text{LoRA-}r) = \sum_{\ell \in \mathcal{T}_{\text{LoRA}}} \left(r \cdot d_{\text{in}}^{(\ell)} + d_{\text{out}}^{(\ell)} \cdot r \right) \cdot 4 \,\text{bytes}.$$

MTH. The active fp32 values plus a compressed boolean mask of their positions:

$$\text{size}(\text{MTH}) = N_{\text{active}} \cdot 4 \,\text{bytes} + \text{zstd}\big(\text{packbits}(\mathbf{1}[\text{active}])\big),$$

where $N_{\text{active}} = (1 - s) \cdot N_{\text{total}}$ for sparsity $s \in \{0.99, 0.9995\}$.

Deletion Tuning. A compressed binary mask over the target weights:

$$\text{size}(\text{Deletion Tuning}) = \text{zstd}\big(\text{packbits}(\mathbf{1}[G > 0])\big).$$

No floating-point values are stored: kept weights are read from the public checkpoint, and zeroed weights are zero. At the learned sparsity below 0.05% across the six GRPO runs in Table 1, zstd compresses to roughly 0.03 bits per masked weight.

B Sparsity Learned by Deletion Tuning per Run

Table 7 reports the fraction of all model weights that Deletion Tuning deletes in the GRPO runs of Table 1 ($\eta = 1/2048$, $\lambda = 10^{-5}$, $lr = 3{\times}10^{-4}$), at the earliest and final saved checkpoints. Every value stays below 0.05%.

Table 7: Fraction of all weights deleted by Deletion Tuning in the six GRPO runs. Fraction of all model parameters set to zero by the trained mask, at the earliest and final saved checkpoint. Every value is below 0.05%.

	Llama, GSM8K			**Qwen, SimpleRL-Zoo**		
	1B	**3B**	**8B**	**0.5B**	**3B**	**7B**
Earliest saved	0.0342%	0.0420%	0.0456%	0.0393%	0.0379%	0.0381%
Final saved	0.0347%	0.0422%	0.0459%	0.0405%	0.0418%	0.0415%

C Per-Benchmark Qwen2.5 Generalization Results

Table 8 reports the full per-benchmark breakdown that underlies the SimpleRL-Zoo (Qwen) accuracy columns of Table 1.

Table 8: Qwen2.5 on SimpleRL-Zoo: per-benchmark validation accuracy (%) at the best checkpoint. Six benchmarks: GSM8K, MATH500, Minerva Math, OlympiadBench, AIME24, AMC23.

Method	GSM8K	MATH500	Minerva	OlympiadBench	AIME24	AMC23
Qwen0.5B						
FFT	48.07	36.00	4.78	7.73	0.00	17.50
LoRA r=1	49.20	33.80	7.35	8.92	0.00	22.50
LoRA r=4	52.46	36.60	8.82	10.55	3.33	10.00
MTH 99%	50.34	35.20	10.29	9.36	3.33	22.50
MTH 99.95%	49.73	35.20	6.25	7.28	0.00	22.50
Deletion Tuning	44.12	33.40	5.51	6.98	3.33	12.50
Qwen3B						
FFT	83.62	66.20	23.16	26.45	16.67	50.00
LoRA r=1	86.81	69.80	21.69	31.35	6.67	45.00
LoRA r=4	85.60	67.00	22.79	28.68	10.00	47.50
MTH 99%	86.28	67.40	22.43	29.68	6.67	52.50
MTH 99.95%	85.37	63.80	22.79	27.93	13.33	50.00
Deletion Tuning	83.78	66.20	22.79	25.78	13.33	40.00
Qwen7B						
FFT	91.43	78.20	33.46	39.92	13.33	55.00
LoRA r=1	92.65	77.60	27.21	39.52	13.33	62.50
LoRA r=4	91.66	77.00	29.04	39.97	20.00	52.50
MTH 99%	91.21	77.20	29.78	39.82	16.67	57.50
MTH 99.95%	92.04	75.00	31.62	38.78	20.00	50.00
Deletion Tuning	91.28	78.80	32.72	40.29	23.33	47.50

D SFT Data Splits

For both MMLU-Pro and BBEH, we split each task's released set $75\%/25\%$ into train and held-out evaluation sets. Tables 9 and 10 list per-task counts.

Table 9: Per-task train/test split for MMLU-Pro under the $75/25$ split. Total: $9{,}018$ train / $3{,}014$ test across 14 tasks.

Category	Train	Test	Category	Train	Test
biology	537	180	law	825	276
business	591	198	math	1013	338
chemistry	849	283	other	693	231
computer science	307	103	philosophy	374	125
economics	633	211	physics	974	325
engineering	726	243	psychology	598	200
health	613	205	history	285	96

Table 10: Per-task train/test split for BBEH under the $75/25$ split. Twenty-two tasks share the 200-example pattern. The only outlier is `disambiguation_qa` with 120 examples. Total: $3{,}390$ train / $1{,}130$ test across 23 tasks.

Task group	Train	Test
22 tasks (200 examples each)	150 each	50 each
`disambiguation_qa` (120 examples)	90	30

E Full Wanda Sparsity Sweep

Tables 11 and 12 report the per-sparsity breakdown underlying Section 5.3's best-Wanda picks. η is the fraction of gate-projection weights deleted (same convention as Eq. (8)), and the sweep runs from $\eta = 1/2$ (half deleted) to $\eta = 1/8192$ ($\approx 0.012\%$ deleted). Wanda uses noise-free gate init ($G_{ij} = \pm 0.03$, hard-mask sparsity exactly η), while the trained Deletion Tuning runs of Table 1 use the noisy init of Eq. (8). Qwen values are the six-benchmark mean across GSM8K, MATH500, Minerva Math, OlympiadBench, AIME24, and AMC23. "Base" is the model before fine-tuning. Bold marks the best Wanda accuracy per backbone.

Table 11: Wanda sweep on Llama (%).

η	Llama1B	Llama3B	Llama8B
$1/2$	0.00	48.07	**60.65**
$1/8$	1.36	51.71	59.14
$1/32$	1.52	51.55	58.91
$1/128$	1.52	52.08	59.67
$1/512$	1.44	52.16	60.05
$1/2048$	**1.59**	52.16	59.74
$1/4096$	**1.59**	51.63	60.05
$1/8192$	1.36	**52.46**	59.67
Base	1.21	53.53	60.27

Table 12: Wanda sweep on Qwen (%).

η	Qwen0.5B	Qwen3B	Qwen7B
$1/2$	1.92	21.06	39.39
$1/8$	10.45	36.09	**42.49**
$1/32$	11.24	38.27	41.81
$1/128$	11.21	37.71	41.90
$1/512$	10.56	37.27	40.56
$1/2048$	**11.42**	38.24	40.14
$1/4096$	11.09	38.60	40.82
$1/8192$	10.74	**38.70**	41.43
Base	10.44	37.73	40.18

F Algebraic Connection to MaskLLM

Deletion Tuning and MaskLLM [32] arrive at their forward passes from different starting points: Deletion Tuning from a continuous relaxation of a per-weight binary masking decision (Eq. (5)), and MaskLLM from a Gumbel-Softmax over $N{:}M$ candidate patterns (Eq. (9)). After three structural modifications to Eq. (9) (binary candidates, no per-group cardinality, no Gumbel noise), the modified MaskLLM forward matches Eq. (5) algebraically.

MaskLLM's forward. The base weight matrix is partitioned into groups of M contiguous weights, and each group is handled independently. Within a single group, MaskLLM enumerates $C(M,N) = \binom{M}{N}$ candidate binary patterns $\widehat{M}_c$ with $c \in \{0, \dots, C(M,N)-1\}$, each keeping exactly N of M entries. It places a learnable logit vector $\pi \in \mathbb{R}^{C(M,N)}$ over the candidates and draws a Gumbel-Softmax sample [27, 26]:

$$\widetilde{y}_c \;=\; \frac{\exp\big((\kappa\,\pi_c + g_c)/\tau\big)}{\sum_{c'} \exp\big((\kappa\,\pi_{c'} + g_{c'})/\tau\big)}, \qquad g_c \overset{\text{iid}}{\sim} \mathrm{Gumbel}(0,1). \tag{9}$$

The group's soft mask is the candidate-pattern-weighted average $\widetilde{M} = \sum_c \widetilde{y}_c \widehat{M}_c$, and the layer forward applies $W \odot \widetilde{M}$ within the group. At evaluation, the hard mask is $\widehat{M}_{c^\star}$ with $c^\star = \arg\max_c \pi_c$. The temperature τ and logit-scale κ both follow linear schedules over training.

Step 1: $C(M,N){=}2$ collapses the two-candidate softmax to a sigmoid. Set $M{=}2, N{=}1$, so each group has exactly two candidates: $\widehat{M}_0 = (1,0)$ keeps position 0 and $\widehat{M}_1 = (0,1)$ keeps position 1. With only two candidates, the Gumbel-Softmax probability on candidate 0 simplifies to a sigmoid of the logit difference:

$$\widetilde{y}_0 = \frac{e^{(\kappa\pi_0+g_0)/\tau}}{e^{(\kappa\pi_0+g_0)/\tau} + e^{(\kappa\pi_1+g_1)/\tau}} = \frac{1}{1+e^{-\frac{1}{\tau}[\kappa(\pi_0-\pi_1)+(g_0-g_1)]}} = \sigma\Big(\tfrac{1}{\tau}\big[\kappa\,(\pi_0-\pi_1)+(g_0-g_1)\big]\Big), \tag{10}$$

where the second equality divides numerator and denominator by the first exponential, and the third uses $\sigma(x) = 1/(1+e^{-x})$. The difference of two independent $\mathrm{Gumbel}(0,1)$ variables is $\mathrm{Logistic}(0,1)$, so

$$\widetilde{y}_0 \;=\; \sigma\big(\tfrac{1}{\tau}[\kappa\,\Delta\pi + \xi]\big), \qquad \Delta\pi := \pi_0 - \pi_1,\;\; \xi \sim \mathrm{Logistic}(0,1). \tag{11}$$

Eq. (11) is the Binary-Concrete distribution [26].

Step 2: drop the per-group cardinality constraint. MaskLLM forces each group of M weights to keep exactly N of them. Letting each weight choose its own binary mask value replaces this group categorical with an independent binary masking choice per weight, parameterized by a single logit G_{ij}. Each weight then has its own Binary-Concrete (Eq. (11) applied per-weight, with G_{ij} in place of $\Delta\pi$):

$$\widetilde{M}_{ij} \;=\; \sigma\big(\tfrac{1}{\tau}(\kappa\, G_{ij} + \xi_{ij})\big), \qquad \xi_{ij} \sim \mathrm{Logistic}(0,1). \tag{12}$$

Step 3: drop the noise. Setting the noise ξ_{ij} to zero in Step 2's per-weight Binary-Concrete gives a deterministic soft mask:

$$\widetilde{M}_{ij} \;=\; \sigma\big(\tfrac{\kappa}{\tau}\, G_{ij}\big). \tag{13}$$

This matches Eq. (5) with the scalar $\alpha(t) := \kappa(t)/\tau(t)$; in our implementation, we schedule this scalar directly and linearly.

G Deletion Tuning Hyperparameter Sweeps

Tables 13 and 14 give every setting of the sweep in Section 5.7.

Table 13: Robustness to hyperparameters on Llama8B GSM8K. Best-checkpoint accuracy (%) of Deletion Tuning for each (η, λ, lr).

	$\lambda = 0$		$\lambda = 10^{-5}$	
$\eta \setminus lr$	10^{-4}	$3{\times}10^{-4}$	10^{-4}	$3{\times}10^{-4}$
$1/2048$	83.40	82.49	82.34	**83.55**
$1/4096$	83.02	82.03	83.09	83.09
$1/8192$	83.40	82.49	83.09	83.02

Table 14: Robustness to hyperparameters on Qwen7B SimpleRL-Zoo. Best-checkpoint accuracy (%) of Deletion Tuning for each (η, λ, lr).

	$\lambda = 0$		$\lambda = 10^{-5}$	
$\eta \setminus lr$	10^{-4}	$3{\times}10^{-4}$	10^{-4}	$3{\times}10^{-4}$
$1/2048$	51.79	51.92	49.60	52.32
$1/4096$	52.23	**52.80**	50.29	51.91
$1/8192$	51.14	52.10	51.29	52.34

H FFT Hyperparameter Sweeps

To make sure the FFT baseline is not under-tuned, we sweep its learning rate under GRPO and SFT and its KL coefficient under GRPO. All other settings follow the main runs. None of these settings changes the comparison with Deletion Tuning (Tables 15 and 16).

Table 15: FFT sweep under GRPO. Best-checkpoint validation accuracy (%), as in Table 1.

Method	**Learning rate**	**KL coefficient**	**Llama8B, GSM8K**	**Qwen7B, SimpleRL-Zoo**
FFT	$5{\times}10^{-7}$	0	83.23	50.55
FFT	$5{\times}10^{-7}$	10^{-4}	82.55	51.89
FFT	10^{-6}	0	83.24	51.93
FFT	10^{-6}	10^{-4}	81.89	50.79
Deletion Tuning	$3{\times}10^{-4}$	as in Table 1	**83.55**	**52.32**

Table 16: FFT learning-rate sweep under SFT, Llama3B MMLU-Pro. Best-checkpoint accuracy (%), as in Table 5.

Method	Learning rate	Accuracy
Base	–	12.63
FFT	10^{-5}	19.35
FFT	3×10^{-6}	21.19
FFT	10^{-6}	18.35
Deletion Tuning	5×10^{-4}	**22.28**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims are directly supported by Sections 4 and 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Limitations paragraph in Section 6 discusses limitations of this work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Appendix F states the assumptions (binary candidates $M{=}2$, $N{=}1$; no per-group cardinality; no Gumbel noise) and gives an algebraic derivation in three steps showing that

MaskLLM's structured forward (Eq. (9)) reduces to Deletion Tuning's soft-mask forward (Eq. (5)) under those assumptions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 4 and 5.1 fully specify the parameterization, training pipelines, and hyperparameters used to produce the reported numbers.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We release the Deletion Tuning implementation on the project page, with our GRPO trainer fork, pinned dependencies, and a README with end-to-end setup and run instructions for GRPO with Deletion Tuning across every Llama and Qwen backbone used in the paper. All datasets and base models are publicly available and fully specified in the main text.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Section 5.1 gives extensive details on data splits, optimizer, batch sizes, learning rates, schedules, and the checkpoint selection rule. Appendices G and H report the hyperparameter sweeps.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Each reported result comes from a single training run, as is common for LLM post-training at these scales given the compute cost. Section 5.7 shows that the results hold across the hyperparameters of our method (initial sparsity, learning rate, and regularization strength).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5.1 reports the GPUs used for each regime (H200 and L40S) and FSDP sharding. All experiments together used about 3,500 GPU-hours.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public datasets and pretrained models, involves no human subjects, and conforms with the Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work has no direct path to deployment-specific harms beyond those of the underlying base models and its baseline methodologies.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released artifact would be small binary masks over already-public base models and poses no incremental misuse risk.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets, base models, and code packages used are cited at first mention and used under their published terms.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not introduce any new datasets, benchmarks, or pre-trained model assets; its primary contribution lies in proposing a parameterization scheme for fine-tuning. The released code comes with its license and usage instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method does not use LLMs as an important, original, or non-standard component, and the Llama and Qwen models in the paper are the subject of fine-tuning. We used LLM tools to provide final polish to our own writing. The idea, the experimental design, and the claims are ours, and we checked every sentence and every line of code.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.