FLASHMASK: IMPORTANCE AND UNCERTAINTY-GUIDED SELECTIVE MASK LEARNING FOR SEMI-STRUCTURED LLM PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Semi-structured $N:M$ pruning enables hardware-friendly acceleration of large language models (LLMs), but learning high-quality sparse masks remains costly. Learnable-mask methods refine $N:M$ masks end-to-end with frozen LLM weights, yet they maintain dynamic mask distributions for all weight blocks throughout training, which makes the schedule long and expensive. We propose **FlashMask**, an importance- and uncertainty-guided selective mask-learning framework. Starting from a SparseGPT prior, FlashMask uses the prior pruning loss to modulate the prior strength of each block and to construct per-matrix candidate pools. After global exploration, it retains dynamic mask refinement only for blocks with small top-two gate-logit margins within these importance-filtered pools, while freezing all remaining masks into fixed sparse weights. Phase B still optimizes the full-model language-modeling objective, but bypasses Gumbel sampling, candidate-pattern mixing, and direct gate-gradient flow for inactive blocks, which both shortens the viable schedule and lowers the cost of each remaining step. The resulting quality–cost trade-off improves in both directions. On LLaMA-2 7B and LLaMA-3 8B under $2{:}4$ sparsity, FlashMask is consistently cheaper and better than MaskLLM at a matched training budget; most importantly, its 600-step schedule is already competitive with a $2{,}000$-step MaskLLM run while using only about a quarter of the compute ($\approx 4\times$ less). We therefore adopt 600 steps as an efficient default across models.

1 INTRODUCTION

Large language models (LLMs) achieve strong results on a broad range of tasks (Hendrycks et al., 2020; Chiang et al., 2024; Zheng et al., 2023; Gao et al., 2021), but their scale makes deployment expensive (Touvron et al., 2023; Zhang et al., 2022a; Brown et al., 2020; Grattafiori et al., 2024). Pruning reduces model memory and inference cost by removing redundant parameters (Mishra et al., 2021; Pool et al., 2021; Frantar & Alistarh, 2023; Sun et al., 2024). Among sparse formats, semi-structured $N:M$ sparsity retains exactly N weights in every group of M, combining fine-grained flexibility with support from modern sparse hardware and software stacks (Mishra et al., 2021; Pool et al., 2021). However, selecting a high-quality $N:M$ mask for every small weight block creates a large discrete search problem.

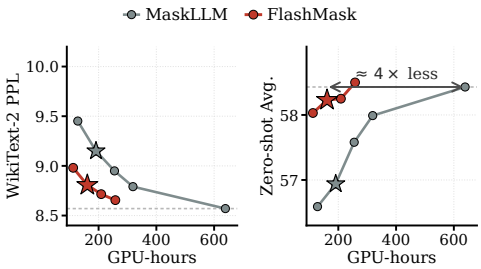

Figure 1: Quality vs. training cost for $2{:}4$ mask learning on LLaMA-3 8B. Each marker is a step budget: FlashMask at $400/600/800/1{,}000$ steps, MaskLLM at these and $2{,}000$. FlashMask (red) dominates MaskLLM (grey) at every matched cost; stars mark the 600-step default, which rivals the $2{,}000$-step MaskLLM quality (dotted) at $\approx 4\times$ less compute (arrow).

One-shot LLM pruning methods, such as SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2024), use a limited calibration set and hand-designed saliency criteria to select masks efficiently. In contrast, MaskLLM (Fang et al., 2024) formulates each $N:M$ mask as a learnable categorical

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.

distribution and optimizes mask logits with Gumbel-Softmax while keeping the pretrained LLM weights frozen. This end-to-end formulation can refine one-shot prior masks using large-scale data, but it applies dynamic mask sampling, candidate-pattern mixing, and gate optimization to every block at every training step.

This uniform treatment is unnecessarily expensive and conceptually mismatched to the pruning problem. Blocks differ in the consequence of their prior pruning decisions, and they do not remain equally ambiguous after initial mask learning: once a gate strongly favors one pattern, further stochastic refinement is unlikely to change it. Learnable mask optimization is therefore better treated as a budget-allocation problem, over which blocks retain dynamic refinement and when that allocation is updated.

We propose **FlashMask**, an importance- and uncertainty-guided selective mask-learning method for frozen-weight $N : M$ pruning. FlashMask reuses the Hessian-aware local reconstruction loss computed during SparseGPT prior pruning as a block-importance signal. Within each prunable weight matrix, blocks near the importance cutoff receive a weaker prior initialization, while the remaining blocks receive a stronger prior. All blocks then participate in a global Phase-A mask-learning stage. At the phase boundary, FlashMask retains, within a per-matrix candidate pool defined by SparseGPT importance, the blocks with the smallest top-two gate-logit margins for Phase-B refinement. The remaining blocks are frozen into fixed sparse weights using the same boundary argmax masks used for final export.

The two signals are complementary: importance decides which blocks are eligible for refinement, gate margin which eligible blocks are still unresolved. Phase B keeps evaluating the complete sparse model under the original language-modeling loss, but bypasses Gumbel sampling, pattern mixing and gate-gradient flow for inactive blocks, so it reduces mask-specific overhead rather than dense Transformer computation.

The effect is a shift in the quality–cost trade-off visible across the whole budget sweep (Fig. 1). At every matched budget FlashMask dominates MaskLLM on both perplexity and accuracy; on LLaMA-3 8B at 600 steps it trains in 161 against 191 GPU-hours while improving perplexity from 9.15 to 8.80 and average accuracy from 56.94 to 58.23. Read as efficiency, the same shift means FlashMask lands within 0.20 accuracy points of the $2{,}000$-step MaskLLM at $\approx 4\times$ less compute, so we adopt 600 steps as the default budget. LLaMA-2 7B shows the same pattern, with a larger gain at a matched budget.

Our contributions are as follows:

- We formulate frozen-weight $N : M$ mask learning as a two-phase selective refinement problem, combining SparseGPT-derived block importance with learned gate-margin information.
- We introduce per-matrix importance-cutoff-margin initialization and importance-constrained gate-margin refresh, which distinguish prior-ranking ambiguity from mask-decision uncertainty.
- We freeze inactive masks into fixed sparse weights at the Phase-A boundary, ensuring consistency between boundary gate decisions, Phase-B execution, and final mask export while bypassing their dynamic mask path.
- Experiments on LLaMA-2 7B and LLaMA-3 8B with $2{:}4$ sparsity, including a training-budget sweep from 400 to $2{,}000$ steps, validate these gains in both quality and training cost.

2 Related Work

2.1 Pruning Large Language Models

LLM pruning methods differ in the sparsity structures they impose. Unstructured methods, including magnitude pruning (Han et al., 2015), SparseGPT (Frantar & Alistarh, 2023), and Wanda (Sun et al., 2024), remove individual weights using saliency criteria. Structured approaches remove larger components such as layers, channels, or hidden dimensions (Ma et al., 2023; Ashkboos et al., 2024; Men et al., 2025). Semi-structured $N : M$ sparsity instead retains a fixed number of weights per local group, providing a hardware-friendly compromise between structured and unstructured sparsity

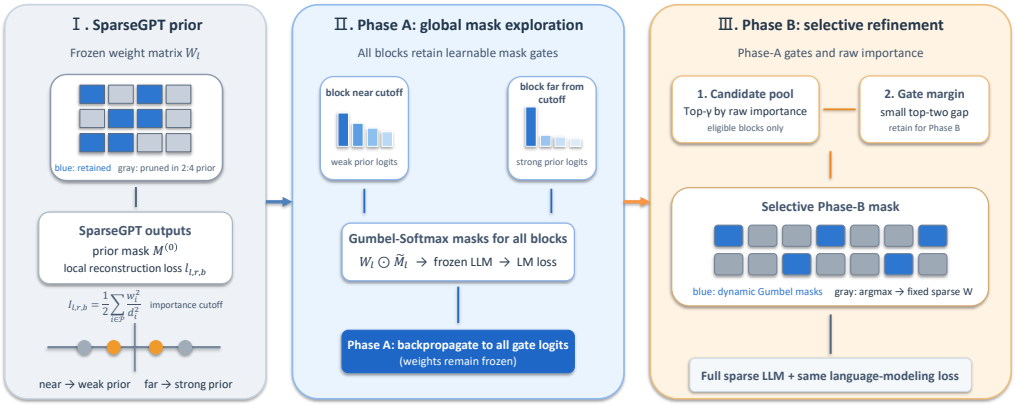

Figure 2: Overview of FlashMask. SparseGPT produces prior $N{:}M$ masks together with block-level Hessian-aware reconstruction losses, which set the per-block prior strength and define a per-matrix importance-filtered candidate pool. After global Phase-A exploration, FlashMask keeps within this pool the blocks with the smallest top-two gate-logit margins for Phase-B refinement, and freezes all remaining masks into fixed sparse weights. The full language-modeling objective is retained in both phases; Phase B bypasses only the dynamic mask path of inactive blocks.

(Mishra et al., 2021; Pool et al., 2021). FlashMask does not introduce a new one-shot pruning criterion; it reuses the local Hessian-aware reconstruction loss produced by SparseGPT prior pruning to guide subsequent mask refinement.

2.2 Learnable Semi-Structured Sparsity

Learnable sparsity methods optimize masks or connectivity using the training objective, for example through straight-through estimators, learnable pattern combinations, or differentiable indexing (Zhou et al., 2021; Lu et al., 2023; Zhang et al., 2022b; Kao et al., 2022). Many methods jointly update weights and sparse connectivity or target training from scratch. MaskLLM (Fang et al., 2024) extends learnable $N:M$ masks to frozen LLMs by optimizing Gumbel-Softmax mask distributions, optionally initialized from one-shot masks. FlashMask retains this formulation but differs in its two-phase allocation: all gates are explored initially, whereas Phase B keeps dynamic mask refinement only for selected blocks and freezes the remaining masks.

2.3 Selective and Uncertainty-Guided Optimization

Uncertainty sampling and margin-based active learning prioritize examples near a model's decision boundary (Lewis, 1995; Settles, 2009; Tong & Koller, 2001). Dynamic sparse training also reallocates connectivity during optimization (Evci et al., 2020; Lasby et al., 2024). FlashMask applies related allocation principles to a different setting: it neither selects data for annotation nor regrows model connections. Instead, it selects frozen-LLM parameter blocks for continued mask refinement. Specifically, SparseGPT importance restricts the eligible candidate pool, and the top-two gate-logit margin identifies unresolved mask decisions within that pool.

3 Methodology

Fig. 2 provides an overview of FlashMask. The method consists of (i) SparseGPT-prior initialization with importance-cutoff-aware prior strength, (ii) global Phase-A mask learning, (iii) importance-constrained gate-margin refresh, and (iv) selective Phase-B refinement that freezes all remaining masks into fixed sparse weights.

3.1 Preliminaries: Frozen-Weight $N{:}M$ Mask Learning

We consider a pretrained language model with frozen weights W. Each prunable linear weight matrix is partitioned into contiguous blocks of M weights along its input-feature dimension. We index a

block by (l, r, b), where l denotes a prunable weight matrix (in tensor-parallel training, its local shard), r an output row, and b an M-weight block within that row, so that $W_{l,r,b} \in \mathbb{R}^M$. Under $N{:}M$ semi-structured sparsity exactly N entries of each block are retained, and the admissible binary patterns form $\mathcal{P}_{N:M} = \{P_k\}_{k=1}^K$ with $K = \binom{M}{N}$; for 2:4 sparsity $K = 6$. Following MaskLLM (Fang et al., 2024), each block carries gate logits $g_{l,r,b} \in \mathbb{R}^K$ that parameterize a categorical distribution over these patterns.

At training step t, a differentiable mask is drawn with Gumbel-Softmax and mixed into an effective weight (block indices omitted for readability):

$$\begin{aligned} \pi_k^{(t)} &= \frac{\exp\Big((a_t g_k^{(t)} + \epsilon_k)/\tau_t\Big)}{\sum_{j=1}^K \exp\Big((a_t g_j^{(t)} + \epsilon_j)/\tau_t\Big)}, \qquad \epsilon_k \sim \mathrm{Gumbel}(0,1), \\ \widetilde{M}^{(t)} &= \sum_{k=1}^K \pi_k^{(t)} P_k, \qquad \widetilde{W}^{(t)} = W \odot \widetilde{M}^{(t)}, \end{aligned} \tag{1}$$

where a_t and τ_t are the logit-scale and temperature schedules. The pretrained weights remain frozen throughout mask learning, and at inference each block takes its most likely pattern, $\widehat{M} = P_{\arg\max_k g_k}$. FlashMask retains this frozen-weight learnable-mask formulation and only changes how prior strength and the Phase-B dynamic mask path are allocated across blocks.

3.2 SparseGPT Prior and Importance-Guided Initialization

We initialize FlashMask from a SparseGPT-pruned model, which provides a prior $N{:}M$ mask $M_{l,r,b}^{(0)}$ for every block and, as a by-product of pruning, the local second-order reconstruction loss that its decision incurs. We reuse the latter as a block-level importance signal.

For one weight matrix W_l with calibration inputs X_l, SparseGPT estimates $H_l = \frac{2}{n} X_l X_l^\top$, applies diagonal damping $\widetilde{H}_l = H_l + \alpha\,\mathrm{mean}(\mathrm{diag}(H_l))I$ with damping ratio α, and forms the upper-triangular Cholesky factor of the inverse, $U_l = \mathrm{chol}(\widetilde{H}_l^{-1})$, with $d_{l,i} = (U_l)_{ii}$. Let $w_{l,r,i}$ be the weight after the preceding SparseGPT error compensation and $q_{l,r,i}$ its value after the current pruning decision. Summing the per-weight losses $\frac{1}{2}(w-q)^2/d^2$ over a block $\mathcal{B}_{l,r,b}$ yields the raw block importance

$$I_{l,r,b} = \frac{1}{2} \sum_{i \in \mathcal{B}_{l,r,b}} \frac{(w_{l,r,i} - q_{l,r,i})^2}{d_{l,i}^2} = \frac{1}{2} \sum_{i \in \mathcal{Z}_{l,r,b}} \frac{w_{l,r,i}^2}{d_{l,i}^2}, \tag{2}$$

where the second equality holds because retained weights satisfy $q = w$, so only the $M - N$ positions $\mathcal{Z}_{l,r,b}$ zeroed by the prior contribute. Thus $I_{l,r,b}$ measures how much local reconstruction error the prior decision already pays on that block.

Importance-cutoff-margin initial selection. Let $\rho \in (0, 1]$ be the active-block ratio, with $\rho = 1$ recovering full mask learning, and let $\mathcal{I}_l$ be the set of blocks in matrix l. All selections in FlashMask are performed independently within each matrix rather than by ranking blocks globally across the model, and the number of initially active blocks is $K_l^{\mathrm{keep}} = \max\big(1, \min(|\mathcal{I}_l|, \mathrm{round}(\rho|\mathcal{I}_l|))\big)$. We standardize the raw scores inside each matrix, $\widehat{I}_{l,r,b} = (I_{l,r,b} - \mu_l)/\max(\sigma_l, 10^{-6})$, where μ_l and σ_l are the mean and standard deviation over $\mathcal{I}_l$. Since this affine map is order preserving and every comparison is within-matrix, it leaves the selected set unchanged; we keep it because it makes the cutoff rule scale free and accommodates future variants using absolute thresholds. Sorting the standardized scores in descending order, $\widehat{I}_{l,(1)} \geq \cdots \geq \widehat{I}_{l,(|\mathcal{I}_l|)}$, we place the importance cutoff halfway between the last kept and the first dropped block, $c_l^{\mathrm{imp}} = \frac{1}{2}\big(\widehat{I}_{l,(K_l^{\mathrm{keep}})} + \widehat{I}_{l,(K_l^{\mathrm{keep}}+1)}\big)$, and select the blocks closest to it:

$$\mathcal{T}_l^{(0)} = \mathrm{BottomK}_{(r,b)\in\mathcal{I}_l}\left(\left|\widehat{I}_{l,r,b} - c_l^{\mathrm{imp}}\right|, K_l^{\mathrm{keep}}\right). \tag{3}$$

Blocks far above the cutoff are confidently important and blocks far below are confidently prunable; the ambiguity concentrates at the boundary, which is where mask learning can still change the outcome. This importance-boundary margin is distinct from the gate-margin criterion used later for Phase-B refresh.

Hybrid prior initialization. Let $o_{l,r,b,k} = P_k^\top M_{l,r,b}^{(0)}$ count the retained positions that pattern P_k shares with the prior mask, so $o_{l,r,b,k} = N$ identifies the prior pattern itself. Gates are initialized asymmetrically according to the initial set:

$$g_{l,r,b,k}^{(0)} = \begin{cases} \xi_k + s\,\sigma_\xi\,(o_{l,r,b,k} - 1), & (r,b) \in \mathcal{T}_l^{(0)}, \\ \beta\,(2\,\mathbb{I}[o_{l,r,b,k} = N] - 1)\,, & (r,b) \notin \mathcal{T}_l^{(0)}, \end{cases} \tag{4}$$

where $\xi_k \sim \mathcal{N}(0, \sigma_\xi^2)$ is the random gate initialization, s is the prior strength, $\mathbb{I}[\cdot]$ the indicator function, and $\beta = \mathrm{clip}(10s/3, 1, 10)$ the strong-prior logit magnitude. Boundary blocks therefore receive only a weak, overlap-graded bias of order $s\sigma_\xi$, while confident blocks are pinned to the prior pattern with logits two orders of magnitude larger. The initial set controls prior strength, not participation: all blocks retain differentiable gates and receive gate-logit updates during Phase A.

3.3 Global Exploration and Importance-Constrained Refresh

During Phase A, lasting T_A steps, every block in every prunable matrix uses the differentiable mask of Eq. 1, the language-modeling loss is evaluated on the completely masked model, and all gate logits are updated. The weak prior assigned near the importance boundary permits local mask exploration, while the strong prior elsewhere stabilizes confident decisions.

Because Eq. 2 is a static score, the candidate pool is fixed once at initialization and only consumed at the Phase-A boundary. With a pool ratio $\gamma \in [\rho, 1]$ and $K_l^{\mathrm{cand}} = \max\big(K_l^{\mathrm{keep}}, \min(|\mathcal{I}_l|, \mathrm{round}(\gamma|\mathcal{I}_l|))\big)$, the pool collects the most important blocks of each matrix,

$$\mathcal{C}_l = \mathrm{TopK}_{(r,b)\in\mathcal{I}_l}\left(I_{l,r,b}, K_l^{\mathrm{cand}}\right), \tag{5}$$

which is computed from raw scores and, by the order-preserving property above, coincides with the set obtained from standardized scores.

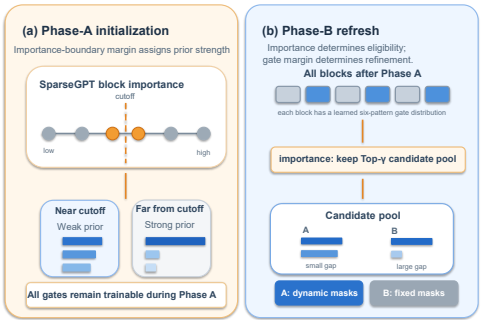

Figure 3: Two margins with distinct roles in FlashMask. The importance-boundary margin sets weak or strong prior logits before Phase A, whereas the top-two gate-logit margin selects unresolved blocks for Phase-B refinement within the importance-filtered candidate pool.

At the boundary we re-rank this pool by the confidence Phase A has reached, as illustrated in Fig. 3. Let $g_{l,r,b,(1)}^{(T_A)}$ and $g_{l,r,b,(2)}^{(T_A)}$ be the largest and second-largest gate logits after T_A steps; a small gate margin $m_{l,r,b}^{\mathrm{gate}} = g_{l,r,b,(1)}^{(T_A)} - g_{l,r,b,(2)}^{(T_A)}$ indicates that several $N{:}M$ patterns remain competitive. Phase B keeps the least decided candidates of each matrix,

$$\mathcal{T}_l^{(B)} = \mathrm{BottomK}_{(r,b)\in\mathcal{C}_l}\left(m_{l,r,b}^{\mathrm{gate}}, K_l^{\mathrm{keep}}\right). \tag{6}$$

SparseGPT importance thus determines which blocks are eligible for refresh, while the gate margin determines which eligible blocks retain dynamic mask refinement; we use the top-two logit gap as a ranking-based proxy for mask-decision confidence at a common phase boundary, not as a calibrated uncertainty estimate. Because Eq. 4 leaves confident blocks with a margin of order 2β while boundary blocks stay near zero, Eq. 6 preserves most of the initial split: it mainly drops initially active blocks that fall outside the candidate pool and back-fills them with the least decided pool members. Selection is the only quantity updated at the boundary; every gate carries its Phase-A state into Phase B, $g^{(T_A^+)} = g^{(T_A)}$.

3.4 Selective Phase-B Refinement via Mask Freezing

Phase B lasts for T_B additional steps. Every block outside $\mathcal{T}_l^{(B)}$ is frozen at its Phase-A decision: we take its boundary gate argmax and freeze it into the stored weight,

$$M_{l,r,b}^{\mathrm{fix}} = P_{\arg\max_k g_{l,r,b,k}^{(T_A)}}, \qquad W_{l,r,b}^{\mathrm{fix}} = W_{l,r,b} \odot M_{l,r,b}^{\mathrm{fix}}, \tag{7}$$

where M^{fix} is recorded at the boundary rather than recomputed afterwards, so an inactive block's boundary decision, baked sparse weight and exported mask are identical by construction. The effective weight during Phase B and the mask exported at the end of training are then

$$W_{l,r,b}^{(t)} = \begin{cases} W_{l,r,b} \odot \widetilde{M}_{l,r,b}^{(t)}, & (r,b) \in \mathcal{T}_l^{(B)}, \\ W_{l,r,b}^{\text{fix}}, & \text{otherwise}, \end{cases}$$
$$M_{l,r,b}^{\text{final}} = \begin{cases} P_{\arg\max_k g_{l,r,b,k}^{(T_A+T_B)}}, & (r,b) \in \mathcal{T}_l^{(B)}, \\ M_{l,r,b}^{\text{fix}}, & \text{otherwise}. \end{cases} \tag{8}$$

Only active blocks perform Gumbel sampling, candidate-pattern mixing and direct gate-gradient flow; inactive blocks bypass the dynamic mask path and reuse their frozen sparse weights. The complete sparse LLM is still evaluated with the original language-modeling objective: FlashMask neither skips backbone computation nor replaces the global loss with a local surrogate.

3.5 Objective and Computational Implications

Let $W^{(t)}$ denote the complete set of effective block weights in Eq. 8. FlashMask optimizes the same frozen-weight full-model objective as MaskLLM:

$$\min_{\{g\}} \mathbb{E}_{x\sim\mathcal{D}}\Big[\mathcal{L}_{\text{LM}}\left(x; W^{(t)}\right)\Big] - \lambda\left(1 - \frac{t}{T}\right)\sum_l \left\|W_l \odot \widetilde{M}_l^{(t)}\right\|_F^2, \tag{9}$$

subject to $\nabla_W \mathcal{L}_{\text{LM}} = 0$, where the second term is the sparse-weight regularizer shared with the MaskLLM baseline. It rewards masks that retain large-magnitude weights, is annealed linearly to zero over the T training steps, and back-propagates only into the gates because W is frozen. The proposed method changes the allocation of the dynamic mask path, not the frozen-weight language-modeling objective.

For a matrix with $|\mathcal{I}_l|$ blocks, full-mask learning draws Gumbel samples and mixes K candidate patterns for every block. Phase B restricts both operations to the $K_l^{\text{keep}} \approx \rho|\mathcal{I}_l|$ active blocks, so this mask-specific cost drops from $O(|\mathcal{I}_l|K)$ to $O(\rho|\mathcal{I}_l|K)$, whereas writing the assembled mask and applying it to the weight matrix remain $O(|\mathcal{I}_l|M)$ in both phases. The dense Transformer backbone computation and the full-model language-modeling loss are also unchanged, so FlashMask does not claim a proportional reduction in total training FLOPs; its practical benefit is the reduced Phase-B per-step cost of bypassing Gumbel sampling, candidate-pattern mixing and gate-gradient flow for inactive blocks. Writing t_{phaseA} and t_{phaseB} for the measured per-step times, total training time is $T_{\text{IUS}} = T_A t_{\text{phaseA}} + T_B t_{\text{phaseB}} + T_{\text{boundary}}$, where T_{boundary} is the one-time selection and mask-freezing cost; we report Phase-A, Phase-B and end-to-end wall-clock measurements separately.

4 Experiments

4.1 Experimental Setup

Models and setup. We evaluate FlashMask on LLaMA-2 7B (Touvron et al., 2023) and LLaMA-3 8B (Grattafiori et al., 2024) under 2:4 semi-structured sparsity. Following MaskLLM (Fang et al., 2024), all linear layers inside the transformer blocks are pruned while the embedding and output layers are kept dense. Models are held in Megatron-LM format with tensor-parallel size 8, and every run uses 8×NVIDIA H800 GPUs.

Training corpus. MaskLLM learns the masks behind its main table on an in-house blended corpus spanning 69 domains, assembled following the LLaMA-2 recipe, from which 512k samples are drawn (Fang et al., 2024). That corpus is not public. Its authors instead designate C4 (Raffel et al., 2020) as the reproducible configuration: the released code provides only the C4 pipeline, their appendix reports C4-trained results for both LLaMA-2 7B and LLaMA-3 8B, and the corresponding C4-trained checkpoints are released. They also quantify what the substitution costs, reporting a WikiText-2 perplexity difference of 0.07 on LLaMA-2 7B between masks learned on C4 and on the blended corpus. We therefore train every learnable-mask run in this paper on C4, which keeps our comparison inside a setting the baseline's authors published rather than a weakened version of it.

Table 1: Main results under 2:4 sparsity. Every row is our own implementation, trained and evaluated on C4 with an identical data pipeline, tokenizer and evaluation harness. Training-free methods select masks from a calibration set and incur no mask-training cost; learned-mask methods optimize the mask against the language-modeling loss, and GPU-h reports that cost on one node with 8×H800 GPUs. † marks the 2,000-step MaskLLM schedule; the two remaining learned-mask rows share a matched 600-step budget. ∗ marks a cost estimated by rescaling, since those two runs landed on a busier node than every other entry (App. G). Avg. is the mean over seven zero-shot tasks, reported per task in Tab. 3. Our method is shaded; **bold** marks the better of the two 600-step methods and underline the best 2:4 result.

	LLaMA-3 8B			LLaMA-2 7B		
Method	GPU-h	WikiText-2 ↓	Avg. ↑	GPU-h	WikiText-2 ↓	Avg. ↑
Dense (unpruned)	—	5.75	64.03	—	5.12	60.97
Training-free masks						
Magnitude	—	2570.32	41.11	—	54.38	49.51
SparseGPT	—	14.94	52.53	—	10.13	52.15
Wanda	—	22.70	47.28	—	11.39	50.07
Learned masks						
MaskLLM†	639	8.57	58.43	606∗	6.81	56.05
MaskLLM	191	9.15	56.94	182∗	8.27	53.12
FlashMask (ours)	**161**	**8.80**	**58.23**	**153**	**7.04**	**55.51**

Reproduction protocol. Every row of Table 1 is produced by us. The one-shot baselines are re-run through our own harness, and both MaskLLM budgets are trained with the same data pipeline, optimizer and schedule as FlashMask, so that the only remaining difference is how the dynamic mask path is allocated across blocks. As a check on the pipeline, our 2,000-step MaskLLM reproduction reaches 8.57 perplexity on LLaMA-3 8B and 6.81 on LLaMA-2 7B, against the 8.49 and 6.78 reported for the released C4 checkpoints.

Configuration. Prior 2:4 masks come from SparseGPT (Frantar & Alistarh, 2023) with 128 C4 calibration sequences. The block-level reconstruction losses of Eq. 2 are recorded during pruning and cached alongside the prior mask, so importance is obtained at no extra cost. Mask training freezes the LLM weights and optimizes only the gate logits on C4 for 600 steps at global batch size 256 and sequence length 4,096, about 0.63B tokens. Unless stated otherwise FlashMask uses $\rho = 0.3$, $\gamma = 0.5$, $T_A = 200$ and $T_B = 400$. Full hyperparameters are listed in App. C.

Evaluation. We report WikiText-2 (Merity et al., 2016) perplexity with a context length of 4,096, computed directly on the sparse checkpoint. For zero-shot evaluation we export the checkpoint with its learned mask applied and use lm-evaluation-harness (Gao et al., 2021) on seven tasks: HellaSwag, RACE, PIQA, WinoGrande, ARC-e, ARC-c and OpenBookQA. We use length-normalized accuracy where it is defined and plain accuracy otherwise, and Avg. denotes the unweighted mean over the seven tasks.

4.2 Main Results

Table 1 reports perplexity and zero-shot accuracy for both models. One-shot criteria degrade sharply at 2:4 sparsity: magnitude pruning collapses on LLaMA-3 8B, and even SparseGPT, which supplies our prior, loses 9.19 perplexity and 11.50 average accuracy points relative to the dense model. Learnable masks recover a large part of this gap, confirming that the prior masks are far from optimal and that end-to-end refinement is worthwhile.

Among the learned-mask methods the comparison runs along two axes, quality and training cost, and FlashMask is favourable on both. Against MaskLLM at the same 600-step budget it is simultaneously cheaper and clearly better: on LLaMA-3 8B it uses 161 GPU-hours against 191, a factor of 1.19, and still improves perplexity from 9.15 to 8.80 and average accuracy from 56.94 to 58.23. Against MaskLLM at 2,000 steps the trade-off reverses into an efficiency claim: FlashMask spends 4.0 times less compute yet closes 60% of the perplexity gap and 87% of the accuracy gap, ending within 0.20 points of the long-schedule average accuracy. The pattern is more pronounced on LLaMA-2 7B, where the shortened schedule hurts MaskLLM considerably more: at 153 against 182 GPU-hours

FlashMask improves perplexity from 8.27 to 7.04 and average accuracy from 53.12 to 55.51, winning on all seven tasks, and it recovers 84% and 82% of the respective gaps to a baseline that costs 4.0 times as much.

Training-budget sweep. The advantage holds across the whole schedule (Fig. 1, full numbers in Tab. 7): FlashMask is better on both metrics at 400, 800 and 1,000 steps as well, and by 1,000 steps it overtakes the 2,000-step MaskLLM (58.50 against 58.43) at 2.5× less compute. Because 600 steps already recover most of that quality at roughly a quarter of the cost, we adopt it as the default budget and expect the choice to transfer to other models.

4.3 Training Cost

Fig. 4 splits the LLaMA-3 8B totals of Tab. 1 by stage. The saving has two sources: the shorter schedule accounts for most of it, and Phase B removes a further 16% on top, which is why FlashMask is faster than MaskLLM even at an identical step count. The per-step numbers isolate the second source. Phase A costs 0.320 GPU-hours per step, indistinguishable from MaskLLM, as expected since it is ordinary full-mask learning, while Phase B costs 0.242, a 24% reduction obtained purely by removing Gumbel sampling, candidate-pattern mixing and gate-gradient flow for inactive blocks. With $\rho = 0.3$ these operations are skipped for 70% of all blocks, yet the speedup is much smaller because the dense backbone, the language-modeling loss and the mask-application cost are unchanged; FlashMask therefore makes no claim of proportional FLOP reduction.

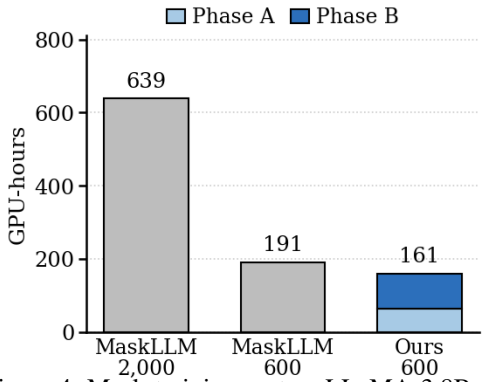

Figure 4: Mask-training cost on LLaMA-3 8B, on one node with 8×H800 GPUs.

4.4 Ablation Studies

Table 2: Initial selection rule and boundary refresh on LLaMA-3 8B. Top-K keeps the most important blocks of each matrix, whereas Margin keeps those closest to the importance cutoff of Eq. 3. The shaded row is our default configuration. Perplexity is given with four decimals because the differences are small relative to the main-result scale.

Initial selection		Boundary	LLaMA-3 8B	
Top-K	Margin	refresh	WikiText-2 ↓	Avg. ↑
✓			8.9088	57.79
✓		✓	8.9022	57.85
	✓		8.8175	58.11
	✓	✓	**8.8071**	**58.23**

All ablations use LLaMA-3 8B with the 600-step budget and vary one factor at a time around the default configuration. Since the differences are an order of magnitude smaller than those in Table 1, we report perplexity with four decimals; the corresponding zero-shot averages are listed in App. D.

Selection rule and boundary refresh. Table 2 separates the two mechanisms that decide which blocks stay active. Replacing the importance-cutoff margin of Eq. 3 by plain Top-K importance, which simply keeps the $\rho|\mathcal{I}_l|$ most important blocks, costs 0.09–0.10 perplexity and 0.3–0.4 accuracy points regardless of whether refresh is enabled. This is the clearest ablation signal in our study and supports the motivation behind Eq. 3: blocks far above the cutoff are already decided by the prior, so allocating the exploration budget to them is wasteful, whereas cutoff-adjacent blocks are the ones whose masks can still change.

Boundary refresh helps consistently but by a smaller margin, 0.0104 perplexity and 0.12 accuracy points for margin selection and 0.0066 and 0.06 for Top-K. This is expected rather than disappointing. As noted after Eq. 6, hybrid initialization leaves confident blocks with gate margins of order 2β while

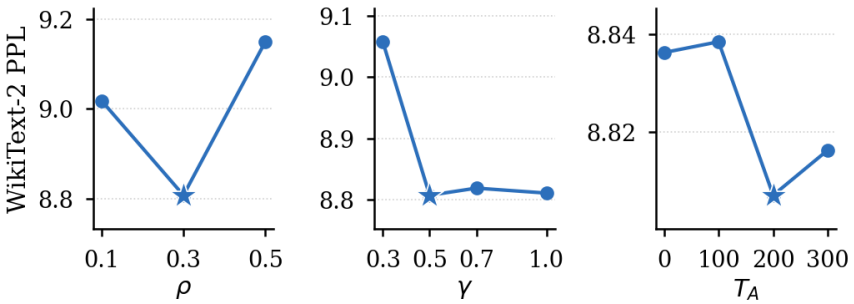

Figure 5: Ablation on the active-block ratio ρ, the candidate-pool ratio γ and the number of Phase-A steps T_A, on LLaMA-3 8B under the 600-step budget. Stars mark the default configuration. Note the narrow vertical range of each panel.

boundary blocks stay near zero, so the refresh is conservative by construction and mainly re-assigns the initially active blocks that fall outside the candidate pool. Its role is to correct a bounded number of selection mistakes at negligible cost, not to reshuffle the allocation. Measured directly, the refresh leaves 78.9% of the weight matrices untouched and replaces 0.42% of the active blocks on average (App. F).

Active-block ratio ρ. Fig. 5 (left) shows a clear optimum at $\rho = 0.3$, with $\rho = 0.5$ the weakest setting overall at 9.1497. The non-monotone behaviour follows from the dual role of ρ in Eq. 4: it sets both how many blocks remain trainable in Phase B and how many blocks receive the weak prior at initialization. Increasing ρ therefore removes the strong prior from blocks whose decision was already reliable and makes Phase B more expensive, while $\rho = 0.1$ leaves too few blocks able to correct the prior; a moderate ratio is thus preferable on both quality and cost.

Candidate-pool ratio γ. Fig. 5 (middle) shows that $\gamma = 0.3$ is markedly worse at 9.0576, whereas $\gamma \in \{0.5, 0.7, 1.0\}$ lie within 0.012 perplexity of one another. The failure case is structural: with $\gamma = \rho$ the pool has exactly the size of the active set, so Eq. 6 has no freedom left and selection reduces to importance ranking alone, discarding the gate-margin signal. Once the pool is large enough for re-ranking, performance is flat, and the unconstrained setting $\gamma = 1.0$ is not better than $\gamma = 0.5$. The importance constraint thus costs nothing while keeping boundary re-ranking cheap, so we adopt $\gamma = 0.5$ as the default.

Phase-A steps T_A. Fig. 5 (right) shows that the split between the two phases is the least sensitive hyperparameter, with all settings within 0.032 perplexity and 0.4 accuracy points. Still, $T_A = 200$ is best on both metrics. Skipping global exploration entirely ($T_A = 0$) is 0.029 worse, since the refresh of Eq. 6 then ranks gate margins still fixed by initialization. Beyond that, extra Phase-A steps mainly displace cheaper Phase-B steps, so $T_A = 200$ gives the best quality at the lowest cost.

5 Conclusion

We presented FlashMask, which casts learnable-mask optimization for frozen-weight $N{:}M$ pruning as a budget-allocation problem: it reuses the Hessian-aware reconstruction loss that SparseGPT already computes as a block-importance signal, concentrates initial exploration near the per-matrix importance cutoff, and after a short global phase keeps dynamic refinement only for importance-eligible blocks whose top-two gate logits remain close, freezing all other masks into fixed sparse weights so that the boundary decision, the Phase-B execution and the exported mask coincide by construction. On LLaMA-2 7B and LLaMA-3 8B with $2{:}4$ sparsity, FlashMask outperforms MaskLLM at an equal 600-step budget and recovers most of the 2,000-step quality at about a quarter of its cost. The allocation is decided at a single boundary; extending it to multiple boundaries or a continuously adapted budget is left to future work.

Use of Large Language Models

In this work, we used generative AI tools only to aid and polish the writing of the paper: improving the readability, grammar and phrasing of text that the authors had already written, and lightly editing table and figure captions. We did not use generative AI tools for any other purpose. In particular,

we did not use them for retrieval or discovery (e.g., finding related work), for research ideation or execution, for drafting sections of the paper, for generating synthetic datasets, or for proving mathematical claims; all of these were carried out by the authors. We have reviewed all AI-assisted text and take full responsibility for the final content of this work, including all claims, figures and artifacts.

REFERENCES

Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, volume 2024, pp. 11682–11701, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference, 2024. *URL https://arxiv. org/abs/2403.04132*, 2 (10), 2024.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pp. 2943–2952. PMLR, 2020.

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. *Advances in Neural Information Processing Systems*, 37:7736–7758, 2024.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Sheng-Chun Kao, Amir Yazdanbakhsh, Suvinay Subramanian, Shivani Agrawal, Utku Evci, and Tushar Krishna. Training recipe for n: M structured sparsity with decaying pruning mask. *arXiv preprint arXiv:2209.07617*, 2022.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. In *International Conference on Learning Representations*, volume 2024, pp. 32982–33010, 2024.

David D Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *Acm sigir forum*, volume 29, pp. 13–19. ACM New York, NY, USA, 1995.

Yucheng Lu, Shivani Agrawal, Suvinay Subramanian, Oleg Rybakov, Christopher De Sa, and Amir Yazdanbakhsh. Step: learning n: M structured sparsity masks from scratch with precondition. In *International Conference on Machine Learning*, pp. 22812–22824. PMLR, 2023.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. *NVIDIA Developer Technical Blog, https://developer. nvidia. com/blog/accelerating-inference-with-sparsityusing-ampere-and-tensorrt*, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Burr Settles. Active learning literature survey. 2009.

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022a.

Yuxin Zhang, Mingbao Lin, Zhihang Lin, Yiting Luo, Ke Li, Fei Chao, Yongjian Wu, and Rongrong Ji. Learning best combination for efficient n: M sparsity. *Advances in Neural Information Processing Systems*, 35:941–953, 2022b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

Algorithm 1 FlashMask: importance- and margin-guided selective $N{:}M$ mask learning

Require: frozen weights W; SparseGPT prior masks $M^{(0)}$ and block importances I (Eq. 2); active ratio ρ, pool ratio γ; phase lengths T_A, T_B; logit-scale/temperature schedules a_t, τ_t
Ensure: final $N{:}M$ mask M^{final}
// Initialization (independently per prunable matrix l)
for each prunable matrix l **do**
  $K_l^{\text{keep}} \leftarrow \text{round}(\rho|\mathcal{I}_l|)$, $K_l^{\text{cand}} \leftarrow \text{round}(\gamma|\mathcal{I}_l|)$ (both clipped to $[1, |\mathcal{I}_l|]$, $K_l^{\text{cand}} \geq K_l^{\text{keep}}$)
  standardize scores, set cutoff c_l^{imp}; $\mathcal{T}_l^{(0)} \leftarrow \text{BottomK}(|\widehat{I} - c_l^{\text{imp}}|, K_l^{\text{keep}})$ ▷ Eq. 3
  $\mathcal{C}_l \leftarrow \text{TopK}(I, K_l^{\text{cand}})$ ▷ candidate pool, Eq. 5
  init gates $g^{(0)}$: weak prior bias on $\mathcal{T}_l^{(0)}$, strong prior elsewhere ▷ Eq. 4
end for
// Phase A: global exploration
for $t = 1$ to T_A **do**
  every block draws $\widetilde{M}^{(t)}$ by Gumbel-softmax (Eq. 1); update **all** gates on $\mathcal{L}_{\text{LM}}$+reg (W frozen)
end for
// Boundary: select and freeze (per matrix l)
for each prunable matrix l **do**
  margins $m^{\text{gate}} \leftarrow g_{(1)}^{(T_A)} - g_{(2)}^{(T_A)}$; $\mathcal{T}_l^{(B)} \leftarrow \text{BottomK}_{\mathcal{C}_l}(m^{\text{gate}}, K_l^{\text{keep}})$ ▷ Eq. 6
  for all $(r, b) \notin \mathcal{T}_l^{(B)}$ **do**
    $M_{l,r,b}^{\text{fix}} \leftarrow P_{\arg\max_k g_{l,r,b,k}^{(T_A)}}$; bake $W_{l,r,b}^{\text{fix}}$ ▷ Eq. 7
  end for
end for
// Phase B: selective refinement
for $t = T_A + 1$ to $T_A + T_B$ **do**
  blocks in $\mathcal{T}^{(B)}$ draw $\widetilde{M}^{(t)}$ and update gates; all other blocks reuse W^{fix}
end for
$M^{\text{final}} \leftarrow P_{\arg\max_k g^{(T_A+T_B)}}$ on $\mathcal{T}^{(B)}$, else M^{fix} ▷ Eq. 8

A The FlashMask Algorithm

Algorithm 1 collects the four stages of Sec. 3 into a single procedure. Everything before the two training loops runs once and per weight matrix: it turns the SparseGPT prior into an initial active set, a candidate pool and the asymmetric gate initialization. The two loops share the same frozen-weight full-model objective and differ only in which blocks keep the dynamic mask path. The boundary step is the sole point at which selection changes and inactive masks are frozen; after it, no block ever leaves or re-enters the trainable set.

B Search Space of 2:4 Mask Learning

Size of the space. Under 2:4 sparsity each block of $M{=}4$ weights keeps exactly $N{=}2$ of them, so it admits $\binom{4}{2} = 6$ binary patterns, the candidate set $\mathcal{P}_{2:4} = \{P_1, \ldots, P_6\}$ drawn in Fig. 6(a). Choices in different blocks are independent, so a model with B prunable blocks has 6^B admissible masks. For LLaMA-3 8B, $B = 1{,}744{,}830{,}464$ (Table 5), which puts the number of valid 2:4 masks above $10^{1.35\times10^9}$. Enumerating this space, or scoring even a vanishing fraction of it, is hopeless; the point of a learnable mask is to search it by gradient descent instead.

What the gates parameterize. The learnable-mask search framework used here is inherited from MaskLLM (Fang et al., 2024): the 6-way gate logits, their Gumbel-softmax relaxation and the argmax export are all part of that formulation, and FlashMask reuses them unchanged, altering only how the search is initialized and how much of it is kept live in Phase B. Within this framework a mask is never represented as a point in the combinatorial set during training. Each block instead carries a 6-way categorical distribution, encoded by the gate logits $g_{l,r,b} \in \mathbb{R}^6$, whose discrete choice is relaxed with Gumbel-softmax (Eq. 1). The joint over the whole model is the product of these per-block categoricals, so the intractable 6^B enumeration is replaced by $6B$ continuous parameters

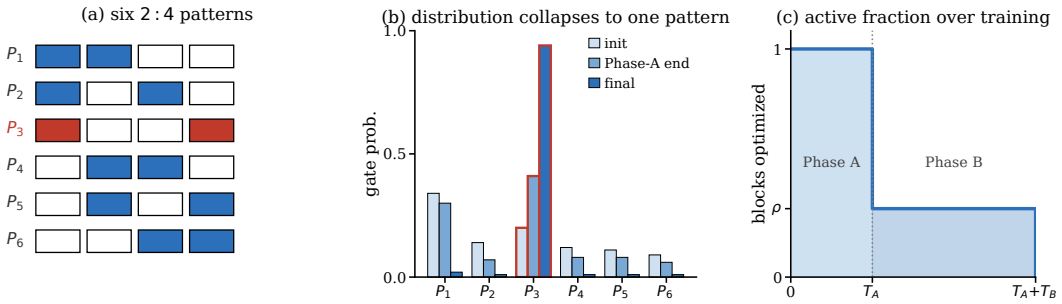

Figure 6: Search space of 2:4 mask learning. (a) Each block admits $\binom{4}{2} = 6$ patterns; filled cells are the two retained weights and P_3 is the pattern learned in this example. (b) The gate categorical distribution over the six patterns collapses from a prior-biased spread at initialization to a near one-hot decision, and can move away from the prior choice (P_1) to the learned one (P_3). (c) Fraction of blocks whose mask is still being optimized: the whole model during Phase A, only an ρ fraction during Phase B, and none after export.

that are optimized jointly with one full-model language-modeling loss. The weights W stay frozen throughout: the *only* thing being learned is, for each block, which two of its four weights to keep.

How the choice is resolved. Fig. 6(b) traces one block's distribution through training. At initialization the categorical is biased toward the SparseGPT prior pattern (here P_1), but not committed to it. During Phase A every block is free to move, and gradients from the full-model loss can shift the mass onto a different pattern than the prior chose—in the example the distribution reconcentrates on P_3. As the logit scale a_t grows and the temperature τ_t falls, the relaxation tightens and each distribution sharpens toward one vertex of its 6-way simplex. By the phase boundary most blocks have a large top-two gate margin and are effectively decided; FlashMask reads off their argmax, freezes them, and bakes the pattern into the stored weights (Eq. 7). Only the $K_l^{\text{keep}} \approx \rho|\mathcal{I}_l|$ least-decided blocks of each matrix keep a live distribution in Phase B, where the remaining ambiguity is resolved. Fig. 6(c) shows the corresponding narrowing of the actively searched space: the full model in Phase A, an ρ fraction in Phase B, and a single pattern per block once training ends.

What is ultimately learned. At convergence every block's distribution is essentially one-hot, so the trained object collapses back to a single point of the original 6^B space: one pattern index per block, equivalently the deployed 2:4 mask $M^{\text{final}} = P_{\arg\max_k g_k}$. The continuous relaxation is only a vehicle for searching the combinatorial space; the exported artifact is a discrete per-block assignment, which is exactly why the freezing step of App. E must store the pattern the model was optimized with rather than recompute it.

C Training Hyperparameters

Prior pruning. The SparseGPT prior is computed with 128 C4 calibration sequences of length 4,096, Hessian damping and a column block size of 128, pruning every linear layer inside the transformer blocks to 2:4. The one-shot baselines in Table 1 follow the standard published setting for those methods, 128 C4 sequences of length 2,048 with a fixed seed, and are evaluated through the same harness as all other rows.

Mask training. All models are held in Megatron-LM format with tensor-parallel size 8 and no pipeline parallelism, and train in bfloat16 on one node with 8×H800 GPUs. The LLM weights stay frozen and only the gate logits are optimized. We use micro batch size 1, global batch size 256 and sequence length 4,096, giving about 0.63B tokens over the 600-step schedule. The optimizer is Adam with $\beta = (0.9, 0.95)$, weight decay 0.1 and gradient clipping 1.0, on a cosine schedule from 5×10^{-5} to 5×10^{-6} without warmup, with a gate learning-rate multiplier of 10. The Gumbel logit scale a_t is annealed from 10^2 to 5×10^2 and the temperature τ_t from 4 to 0.05. The prior strength is $s = 3.0$, which gives a strong-prior logit magnitude $\beta = 10$, and the sparse-weight regularizer uses $\lambda = 10^{-5}$.

Table 3: Per-task zero-shot accuracy behind the averages of Tab. 1. All pruned models use 2:4 sparsity and every row is our own implementation. † marks the 2,000-step MaskLLM schedule. Our method is shaded; **bold** marks the better of the two 600-step methods and underline the best 2:4 result in each column.

Method	WikiText-2 ↓	HellaSwag	RACE	PIQA	WinoG.	ARC-e	ARC-c	OBQA	Avg. ↑
LLaMA-3 8B									
Dense	5.75	79.18	39.14	80.58	73.32	77.61	53.75	44.60	64.03
Magnitude	2570.32	41.87	26.22	63.00	53.51	42.63	29.52	31.00	41.11
SparseGPT	14.94	57.92	37.70	72.36	67.01	60.86	35.67	36.20	52.53
Wanda	22.70	49.15	32.54	70.08	61.25	54.17	31.40	32.40	47.28
MaskLLM†	8.57	72.79	38.95	77.75	68.11	69.15	41.47	40.80	58.43
MaskLLM	9.15	70.22	**39.43**	76.28	67.40	63.76	40.87	**40.60**	56.94
FlashMask (ours)	**8.80**	**70.90**	39.33	**77.20**	**67.88**	**68.39**	**43.52**	40.40	**58.23**
LLaMA-2 7B									
Dense	5.12	76.16	39.62	78.73	69.30	73.78	44.97	44.20	60.97
Magnitude	54.38	58.17	31.20	71.27	61.17	55.85	34.30	34.60	49.51
SparseGPT	10.13	59.17	38.09	72.91	66.54	58.33	33.02	37.00	52.15
Wanda	11.39	55.22	36.46	70.89	61.96	57.87	32.51	35.60	50.07
MaskLLM†	6.81	68.45	40.10	76.22	65.82	64.14	38.40	39.20	56.05
MaskLLM	8.27	65.05	37.51	73.94	64.01	60.44	35.67	35.20	53.12
FlashMask (ours)	**7.04**	**67.64**	**39.04**	**75.46**	**65.90**	**63.64**	**37.12**	**39.80**	**55.51**

Baselines. The MaskLLM baseline shares every setting above and differs only in that all blocks keep the dynamic mask path for the whole schedule. We report it at $T = 600$, matching our budget, and at $T = 2{,}000$, its default schedule.

Table 4: Full ablation numbers on LLaMA-3 8B under the 600-step budget, corresponding to Fig. 5. One factor is varied at a time around the default configuration.

Factor	Value	WikiText-2 ↓	Avg. ↑
Active-block ratio ρ	0.1	9.0180	57.78
	0.3 (default)	**8.8071**	**58.23**
	0.5	9.1497	57.31
Candidate-pool ratio γ	0.3	9.0576	57.68
	0.5 (default)	**8.8071**	**58.23**
	0.7	8.8187	58.00
	1.0	8.8106	58.08
Phase-A steps T_A	0	8.8362	58.08
	100	8.8384	57.83
	200 (default)	**8.8071**	**58.23**
	300	8.8163	58.04

D Full Ablation Numbers

Table 4 lists the exact perplexity and zero-shot average behind Fig. 5. The zero-shot averages track the perplexity ordering in every group, so the conclusions drawn in Sec. 4.4 do not depend on which of the two metrics is used.

E Mask Freezing and Boundary Gates

Once an inactive block is frozen at the phase boundary, its mask can be exported in two ways: recomputed from the final gate argmax, or read back from the boundary mask M^{fix} that Eq. 7 records. The two are not equivalent. Inactive gates receive no loss gradient in Phase B, but Adam momentum accumulated during Phase A and decoupled weight decay keep updating them, so a block whose top two logits were nearly tied at the boundary can flip after it has already been baked into the weights.

Table 5 quantifies this drift, which is numerically tiny: 8,874 of 1.22 billion inactive blocks change pattern, and the two export modes are indistinguishable in quality, 8.8109 against 8.8111, well inside

Table 5: Effect of the mask export mode on LLaMA-3 8B with the default configuration. *Recomputed* derives the exported mask of an inactive block from its final gate argmax; *stored* exports the boundary mask that was baked into the weights. Block counts are summed over all eight tensor-parallel shards.

Prunable 2:4 blocks	1,744,830,464
Blocks inactive in Phase B	1,221,381,376
Blocks whose exported mask differs	8,874
share of inactive blocks	0.0007%
Weight matrices affected	214/1024
WikiText-2, recomputed export	8.8111
WikiText-2, stored export	8.8109

Table 6: Per-step training cost on one node with 8×H800 GPUs. Phase A of FlashMask is ordinary full-mask learning and matches the MaskLLM per-step cost; the saving comes entirely from Phase B. Multiplying by the step counts gives the GPU-h totals of Tab. 1. $*$ marks an estimate, obtained as described below.

			GPU-h/step	
Method	Stage	Steps	LLaMA-3 8B	LLaMA-2 7B
MaskLLM†	—	2,000	0.320	0.303*
MaskLLM	—	600	0.319	0.303*
FlashMask	Phase A	200	0.320	0.303
FlashMask	Phase B	400	**0.242**	**0.230**
FlashMask	total	600	0.268	0.255

the evaluation noise of App. I. We nevertheless keep the stored mask, because the argument is about deployment rather than accuracy. A 2:4 model is shipped by permanently compressing the surviving weights into the pattern that the exported mask specifies, so that pattern has to be exactly the one Phase B computed with and the one the reported numbers were measured on. Under recomputed export it is not: the drift touches 214 of 1024 weight matrices, the deployed pattern no longer matches the sparse forward pass that produced the validation numbers, and it is a function of optimizer state rather than of the trained masks, so it is not reproducible from the checkpoint alone. Storing M^{fix} removes that failure mode at no cost.

F How Much Does the Boundary Refresh Change?

Sec. 4.4 shows that the boundary refresh helps, but it is worth recording how small an intervention it is. We logged, for each of the 128 prunable weight matrices of LLaMA-3 8B, the symmetric difference between the active set chosen at initialization and the one in force after the refresh at $T_A = 200$.

The refresh is extremely conservative. In 101 of the 128 matrices (78.9%) it changes nothing at all, and averaged over all matrices it replaces only 0.42% of the active blocks; even the most affected matrix exchanges 10.9% of its active set. The candidate pool is the reason: a block can only be promoted if the prior already ranked it in the top γ fraction by importance, so the refresh can correct the importance-based selection where Phase A disagrees with it, but cannot overturn it.

This is consistent with the size of the effect in Table 2. Reassigning well under one percent of the active blocks is worth 0.0085 perplexity, which indicates that the blocks the refresh moves are exactly the ones whose masks were still genuinely undecided, rather than that the initial selection was poor. It also explains why the refresh is essentially free: it runs once, and it neither adds nor removes trainable blocks.

G Per-Step Training Cost

Table 6 breaks Fig. 4 into per-step costs. We measure the median steady-state per-iteration time and multiply by the eight GPUs of the node. This excludes process startup and checkpoint writing, which scale with the checkpointing interval rather than with the method. Phase A is indistinguishable

Table 7: Training-budget sweep on LLaMA-3 8B under 2:4 sparsity, the numbers behind Fig. 1. All runs use the identical C4 pipeline and evaluation harness of Tab. 1; GPU-h is the mask-training cost on one node with 8×H800 GPUs. At each shared budget the better perplexity and accuracy are in **bold**.

Steps	GPU-h	WikiText-2 ↓	Avg. ↑
MaskLLM			
400	128	9.45	56.59
600	191	9.15	56.94
800	255	8.95	57.58
1,000	319	8.79	57.99
2,000	639	8.57	58.43
FlashMask (ours)			
400	112	**8.98**	**58.03**
600	161	**8.80**	**58.23**
800	209	**8.72**	**58.25**
1,000	258	**8.66**	**58.50**

from the MaskLLM baseline, as it should be, since it runs the same full-mask computation on every block, and Phase B is about 24% cheaper per step. The one-time boundary cost, which comprises the gate-margin ranking and the freezing of the inactive masks into the stored weights, is included in the Phase-B figure and is below the resolution of a single training step.

One caveat deserves stating, because it decides which measurement we report. The cluster is shared, and its nodes fall into two clearly separated speed regimes: on LLaMA-3 8B the identical MaskLLM computation took 143.8, 143.6 and 144.0 seconds per step in three runs, but 175.2 seconds, a factor of 1.22 slower, in a fourth. Comparing a run from one regime against a run from the other measures the cluster rather than the method, so we report every number in the faster regime, in which all three LLaMA-3 8B runs and our LLaMA-2 7B run were trained.

The two LLaMA-2 7B MaskLLM budgets are the only entries that were trained in the slower regime, at 164.6 seconds per step, and we rescale them. Two independent routes give the same answer. Phase A is by construction the same computation as MaskLLM, so our LLaMA-2 7B Phase A measurement, 136.3 seconds per step, is directly the quantity we need; on LLaMA-3 8B this identity holds to within 0.3%. Alternatively, dividing the slow measurement by the 1.22 regime ratio observed on LLaMA-3 8B gives 135.1 seconds per step. The two estimates agree to 0.9%, and we adopt the former. Both are far below the raw slow measurement, which would have inflated our reported speedup on LLaMA-2 7B from 1.19× to 1.29×.

H Training-budget sweep

Table 7 lists the LLaMA-3 8B numbers behind Fig. 1, sweeping the mask-training budget for both methods under the otherwise identical configuration of Sec. 3. FlashMask is trained for 400, 600, 800 and 1,000 steps and MaskLLM additionally for 2,000; every point keeps $\rho = 0.3$, $\gamma = 0.5$ and $T_A = 200$, so only the number of Phase-B steps grows with the budget. At every matched budget FlashMask is better on both metrics, and its 1,000-step run (8.66 perplexity, 58.50 average accuracy) already surpasses the 2,000-step MaskLLM in average accuracy while using 2.5 times less compute. This is what motivates the default 600-step budget used elsewhere: it captures most of the achievable quality at a fraction of the long-schedule cost, and we expect the same trade-off to transfer to other models and model families.

I Reproducibility

The ablation differences in App. D are small in absolute terms, so their scale should be read against measurement noise. Re-evaluating one fixed checkpoint of the default configuration three times gives 8.8071, 8.8074 and 8.8081, a spread of 0.0010 perplexity that comes from non-deterministic reduction order in bfloat16. The margin-versus-Top-K effect of 0.09–0.10 perplexity is therefore two orders of magnitude above evaluation noise, whereas the boundary-refresh and candidate-pool effects are smaller and should be read as consistent trends rather than as precisely resolved gaps.