Learnable Permutation for Structured Sparsity on Transformer Models

Zekai Li*, Ji Liu*, Guanchen Li, Yixing Xu, Ziqiong Liu, Xuanwu Yin, Dong Li, Emad Barsoum

Advanced Micro Devices, Inc.

{Zekai.Li, Ji.Liu, Guanchen.Li, Yixing.Xu, Ziqiong.Liu, Xuanwu.Yin, d.li, Emad.Barsoum}@amd.com

Abstract

Structured sparsity has emerged as a popular model pruning technique, widely adopted in various architectures, including CNNs, Transformer models, and especially large language models (LLMs) in recent years. A promising direction to further improve post-pruning performance is weight permutation, which reorders model weights into patterns more amenable to pruning. However, the exponential growth of the permutation search space with the scale of Transformer architectures forces most methods to rely on greedy or heuristic algorithms, limiting the effectiveness of reordering.

In this work, we propose a novel **end-to-end learnable** permutation framework. Our method introduces a learnable permutation cost matrix to quantify the cost of swapping any two input channels of a given weight matrix, a differentiable bipartite matching solver to obtain the optimal binary permutation matrix given a cost matrix, and a sparsity optimization loss function to directly optimize the permutation operator. We extensively validate our approach on vision and language Transformers, demonstrating that our method achieves state-of-the-art permutation results for structured sparsity.

Introduction

Transformer architectures (Vaswani et al. 2017) have achieved remarkable success across diverse AI applications, including vision models such as ViT (Yuan et al. 2021), DETR (Carion et al. 2020), DiT (Peebles and Xie 2023), and large language models (LLMs) such as GPT (Floridi and Chiriatti 2020), LLaMA (Touvron et al. 2023), Qwen (Bai et al. 2023), and DeepSeek (Guo et al. 2025). Their strong representational capacity and generalizability have made Transformers the preferred architecture for foundation models. However, deploying these large-scale models on resource-constrained hardware remains challenging, as inference cost grows rapidly with increasing model size. To this end, structured pruning under N:M sparsity constraints, which requires that only N out of every group of M consecutive weights remain nonzero, has emerged as an efficient solution (Bengio, Léonard, and Courville 2013; Han et al. 2015; Sun et al. 2023a; Fang et al. 2024). Its regular structure enables significant parameter reduction while maintaining hardware compatibility, as demonstrated by recent GPUs that accelerate structured sparse patterns such as 2:4 sparsity (Zhou et al. 2021).

Despite their effectiveness, structured pruning methods such as Wanda (Sun et al. 2023b), SparseGPT (Frantar and Alistarh 2023), and PrunerZero (Dong et al. 2024) still degrade accuracy due to a fundamental mismatch between rigid sparsity patterns and inherent weight distributions. Standard N:M pruning preserves only the top-N weights within fixed-size groups, irrespective of actual weight importance. Since Transformer channels are initially ordered arbitrarily, important weights can easily be pruned unintentionally. To mitigate this, channel-wise weight permutation methods reorder weight matrices before pruning to better align weight saliency with sparsity patterns, significantly reducing accuracy loss (Pool and Yu 2021).

However, current permutation approaches mainly rely on heuristic or greedy algorithms (Zhang et al. 2023), which optimize local importance scores rather than directly improving end-to-end task performance. Furthermore, these heuristics neglect global coordination and are computationally expensive, typically employing costly linear sum assignment or searching algorithms (Pool and Yu 2021). Such complexity becomes impractical for large Transformer models with numerous layers and channels, limiting the efficiency and quality of resulting permutations.

To address these shortcomings, we propose a fully learnable permutation framework to jointly optimize channel permutation and structured pruning in an end-to-end manner. However, there are two significant challenges. **First, the permutation operation is inherently discrete and non-differentiable**, complicating its integration with gradient-based training. **Second, existing importance heuristics are insufficient for guiding permutation decisions that affect overall model performance.** This calls for an end-to-end optimization framework that directly links permutation learning with task-level objectives. In response, our framework introduces three key innovations:

- A **learnable permutation cost matrix** that explicitly quantifies the cost of swapping any two input channels of a given weight matrix.
- To address the non-differentiability of discrete permutation, we design a **differentiable approximation of bipartite matching** guided by the learnable cost matrix,

*These authors contributed equally.

enabling efficient and accurate binary permutation matrix learning with minimal computational overhead.

- An **end-to-end sparsity optimization loss** function is proposed to jointly guide the optimization of the permutation operator, achieving a fine balance between task-specific performance and alignment with the dense teacher model through knowledge distillation.

Through end-to-end learning, the proposed framework derives a dedicated permutation matrix for each weight tensor, which is then multiplied with the original weights to produce reordered weights that align more naturally with the target sparsity pattern. We apply this approach to both vision and language models, including ViT, LLM, and VLM backbones, and conduct extensive experiments to validate its effectiveness. Experimental results demonstrate that our framework achieves state-of-the-art structured sparsity with significantly reduced accuracy degradation, outperforming traditional greedy baselines on a variety of benchmarks.

Related Work

Model Pruning. Model pruning compresses a pre-trained model by reducing its parameter count, memory usage, and computational footprint (Li et al. 2025). Contemporary pruning approaches can be broadly categorized into three types. *Unstructured pruning* eliminates individual weight elements, offering fine-grained sparsity control. However, the resulting irregular sparsity patterns pose challenges for hardware acceleration, often requiring extremely high sparsity levels to achieve meaningful speedup (Han, Mao, and Dally 2016; Han et al. 2015; Liao et al. 2025). *Structural pruning* removes entire filters, channels, or layers, producing regular sparsity patterns that are hardware friendly. While it simplifies deployment, this coarse-grained pruning often leads to considerable accuracy degradation and typically demands retraining to recover performance (Ma, Fang, and Wang 2023; Xia et al. 2023; He, Zhang, and Sun 2017). *(Semi-)Structured pruning*, or N:M sparsity, enforces a fixed number of nonzero weights per block, balancing accuracy preservation with hardware efficiency. It retains much of the flexibility of unstructured pruning while producing regular memory layouts suitable for modern accelerators (Pool, Sawarkar, and Rodge 2021; Pool and Yu 2021; Frantar and Alistarh 2023).

N:M Sparsity. The N:M sparsity constraint enforces at most N nonzero values within each block of M weights, achieving a favorable trade-off between compression and inference efficiency on sparsity-aware hardware (Pool, Sawarkar, and Rodge 2021; Pool and Yu 2021; Fang et al. 2024; Hu, Zhu, and Chen 2024). Earlier methods applied static pruning masks after training, while recent techniques integrate mask learning into the optimization loop using continuous relaxations and gradient-based updates (Zhou et al. 2021; Lu et al. 2023; Fang et al. 2024; Liu et al. 2025). Approaches such as sparse-refined straight-through estimators further promote the retention of important weights while maintaining strict adherence to the N:M sparsity constraint (Bengio, Léonard, and Courville 2013; Han et al. 2015). Beyond mask selection, post pruning weight update

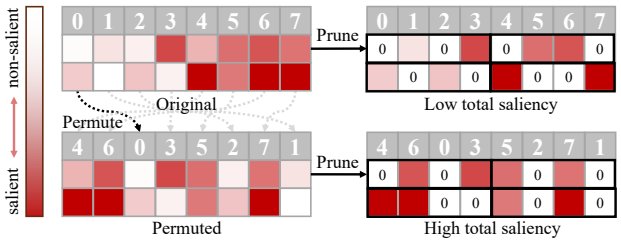

Figure 1: The channel permutation process enhances the friendliness of the 2:4 sparsification, making the overall saliency of the pruning metric more preserved.

methods recover accuracy under the N:M constraint by solving local reconstruction subproblems via second order updates or constrained quadratic optimization (Frantar and Alistarh 2023; Boža 2024). Our work is orthogonal to these pruning and weight update techniques. We focus on learning channel permutations that align saliency with the N:M mask, which can be integrated seamlessly.

Matrix Permutation for Pruning Optimization. Matrix permutation aims to rearrange weights such that salient and non-salient values are distributed more uniformly across pruning groups. This improves alignment with structured sparsity patterns like N:M, enhancing pruning compatibility and preserving model performance. Channel permutation was first introduced in (Pool and Yu 2021), which identifies an optimal reordering via exhaustive greedy search. However, such methods become impractical when applied to large language models due to the computational cost of processing high-dimensional weight matrices. The Plug-and-Play method (Zhang et al. 2024) formulates permutation as a combinatorial optimization problem and solves it efficiently using the Hungarian algorithm. However, as a rule-based method, it does not support end-to-end optimization and incurs high cost in large-scale models due to the linear sum assignment operation over large weight tensors.

In this work, we tackle the large search space of channel permutations by introducing a learnable permutation mechanism for GPT-scale Transformers. Our method enables effective channel reordering that improves model accuracy under N:M sparsity constraints.

Methods

We begin by introducing preliminaries on channel permutation in the context of optimizing structured N:M sparsity for Transformers. We then describe our proposed end-to-end learnable permutation framework, which consists of a learnable cost prediction module, a differentiable bipartite matching solver, and optimization objectives. The unified framework facilitates end-to-end optimization of permutation operators for diverse Transformer-based architectures, including vision, language, and vision-language models.

Preliminaries

Optimize N:M Sparsity via Channel Permutation. Channel permutation enhances the compatibility of weight

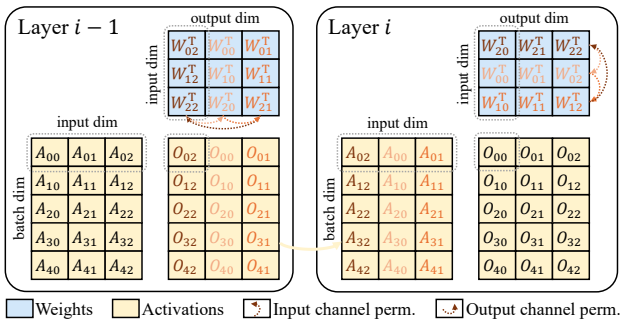

Figure 2: Channel permutation for Linear layers. To guarantee output consistency, after the input channel of the i-th layer weight is permuted, the input activation of that layer should also be permuted accordingly, which can be realized by permuting the output channel of the previous $(i-1)$-th layer weight accordingly.

tensors with structured sparsity patterns such as N:M sparsity. As shown in Figure 1, the weight layout often exhibits uneven saliency, with important weights clustered within certain groups. This skewed distribution lowers the chance of retaining key weights under fixed 2:4 sparsity, resulting in suboptimal pruning with reduced preserved saliency.

By permuting channels before pruning, saliency becomes more evenly distributed across groups. This increases the likelihood that each M-element group contains a mix of important and unimportant weights, allowing structured pruning to retain more informative elements. Channel permutation thus improves N:M pruning by aligning weight layout with sparsity constraints.

Channel Permutation for Linear Layers. Applying a channel permutation to a linear layer's input dimension requires aligning the permuted weights with its input activation to preserve output consistency. Let $\mathbf{W}_i^\top \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ be the weight matrix of the i-th linear layer, and let $\mathbf{P} \in \{0,1\}^{d_{\text{in}} \times d_{\text{in}}}$ be a permutation matrix. Applying input channel permutation to the weight yields $\widehat{\mathbf{W}}_i^\top = \mathbf{P}\mathbf{W}_i^\top$. To maintain correct computation, the corresponding activation $\mathbf{A}_i \in \mathbb{R}^{d_{\text{batch}} \times d_{\text{in}}}$ must also be transformed as $\widehat{\mathbf{A}}_i = \mathbf{A}_i\mathbf{P}^\top$, so that the output remains unchanged ($\mathbf{P}^\top = \mathbf{P}^{-1}$):

$$\widehat{\mathbf{A}}_{i+1} = \widehat{\mathbf{A}}_i \widehat{\mathbf{W}}_i^\top = \mathbf{A}_i\mathbf{P}^\top\mathbf{P}\mathbf{W}_i^\top = \mathbf{A}_i\mathbf{W}_i^\top = \mathbf{A}_{i+1}, \quad (1)$$

To avoid runtime activation permutation, we propagate the permutation backward to the output channels of the preceding $(i-1)$-th layer ($\widehat{\mathbf{W}}_{i-1} = \mathbf{W}_{i-1}\mathbf{P}^\top$), as shown in Figure 2, so that:

$$\widehat{\mathbf{A}}_i = \mathbf{A}_{i-1}\widehat{\mathbf{W}}_{i-1} = \mathbf{A}_{i-1}\mathbf{W}_{i-1}\mathbf{P}^\top = \mathbf{A}_i\mathbf{P}^\top. \quad (2)$$

Channel Permutation for Transformer Layers. Applying channel permutations in Transformer architectures is challenging due to the structural coupling within multi-head self-attention (MHA) and feed-forward networks (FFN). Unlike sequential linear layers, Transformer blocks use parallel projections that share inputs and have interdependent

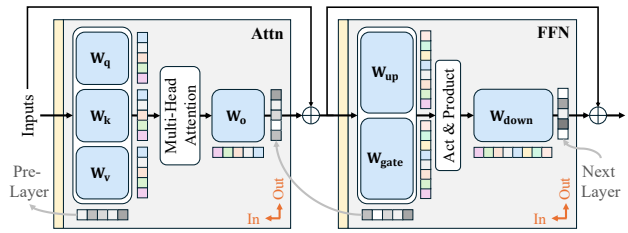

Figure 3: An overview of channel permutation for Transformer layers. The alignment of the input channel permutation of the current layer's weights to the output dimension of the previous layer's weights reflects a structural coupling.

weights, requiring coordinated, structure-aware permutations, as shown in Figure 3.

Based on the rule that the input channel permutation of the current layer's weights will affect the output channel permutation of the previous layer's weights, the input channel permutation of $\mathbf{W}_{\text{o}}$ in the Transformer model will be executed as a binding to $\mathbf{W}_{\text{q}}$, $\mathbf{W}_{\text{k}}$, and $\mathbf{W}_{\text{v}}$. Similarly, the input channel permutation of $\mathbf{W}_{\text{down}}$ will be executed in a binding to $\mathbf{W}_{\text{up}}$ and $\mathbf{W}_{\text{gate}}$. A detailed proof can be found in supplementary materials.

Learnable Channel Permutator

To support semi-structured N:M sparsity constraint in Transformer models, we propose a learnable channel permutation framework that enables end-to-end optimization of permutation operators. As shown in Figure 4, the framework consists of a permutation cost predictor, a differentiable bipartite matching solver, and an optimization training objective. The cost predictor produces layer-wise cost matrices, which are used to generate permutation matrices. These reorder the weights before pruning by an N:M mask generator (actually structured sparsity pruning method, we use Wanda in this paper). During training, pruned weights are inversely permuted for loss computation, preserving gradient flow through the pipeline. Unlike heuristic approaches based on local importance or static ranking, our method learns permutation jointly across gradient-based end-to-end optimization.

Permutation Cost Predictor. The core of our method is a learnable permutation cost predictor, which produces a cost matrix to guide the reordering of channels or features. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ from a weight matrix, the goal is to construct a permutation matrix $\mathbf{P} \in \{0,1\}^{d_{\text{in}} \times d_{\text{in}}}$ that rearranges the input channels such that the pruned model aligns more effectively with structured N:M sparsity constraints.

To enable differentiable learning of $\mathbf{P}$, we introduce a real-valued cost matrix $\mathbf{C} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$, where each element $C_{i,j}$ reflects the cost of assigning the original input channel i to position j. Intuitively, $\mathbf{C}$ encodes the pairwise preference for spatial relocation, integrating both structured sparsity alignment and semantic preservation objectives. We parameterize the cost matrix using learnable parameters. For each input channel, we implement a $d_{\text{in}} \times d_{\text{in}}$ learnable parameter as the cost predictor. The predictor outputs a normal-

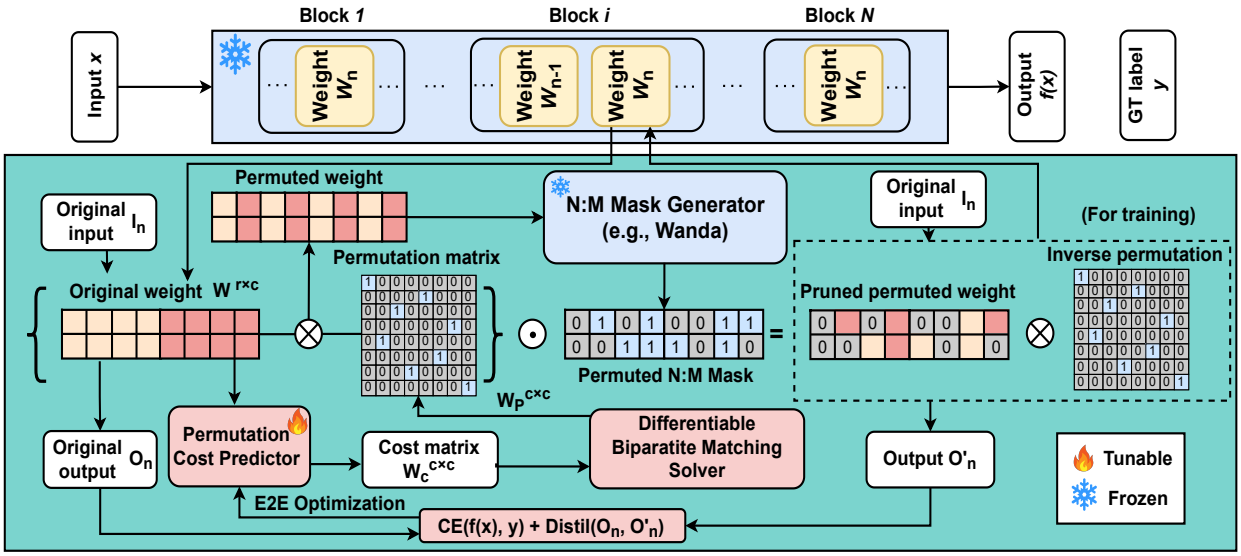

Figure 4: Overview of our learnable permutation framework. A permutation cost predictor generates cost matrices for each linear layer, which are converted into permutation matrices via a differentiable bipartite matching solver. The original weights are permuted accordingly and then sparsified using an N:M mask generator. During training, the pruned weights are inversely permuted and used for loss computation. The entire process supports end-to-end optimization while maintaining gradient flow through the binary permutation matrix generated by the differentiable solver.

ized cost matrix, which quantifies the cost of swapping two input channels. We minimize the cumulative cost of each cost matrix with our proposed bipartite matching solver. Our experimental results (Table 1) demonstrate that the proposed permutation cost predictor achieves strong pruning performance despite its simple design, which is intentionally lightweight to minimize additional training overhead.

Furthermore, directly learning full permutation matrices for large-dimensional weight tensors in Transformer models is computationally expensive and often unnecessary. To improve scalability and reduce overhead, we adopt a group-wise permutation strategy, where the input channels of each layer are partitioned into non-overlapping groups size G, and a separate permutation is learned within each group.

Differentiable Bipartite Matching Solver. To enable gradient-based optimization of permutation matrices, we introduce a differentiable bipartite matching solver. Given a learned cost matrix $\mathbf{C} \in \mathbb{R}^{N\times N}$, where $\mathbf{C}_{i,j}$ indicates the cost of mapping the ith input channel to the jth output, our goal is to find a permutation matrix $\mathbf{P} \in \mathcal{P}$ minimizing:

$$\min_{\mathbf{P}\in\mathcal{P}} \langle \mathbf{C}, \mathbf{P} \rangle, \tag{3}$$

where $\mathcal{P}$ is the set of all $N \times N$ permutation matrices.

Since $\mathcal{P}$ is discrete and non-differentiable, we relax the optimization over the Birkhoff polytope $\mathcal{B}_N$—the convex hull of $\mathcal{P}$—which comprises all doubly stochastic matrices:

$$\mathcal{B}_N = \left\{ \mathbf{P} \in \mathbb{R}^{N\times N} \,\middle|\, \mathbf{P} \geq 0,\ \mathbf{P}\mathbf{1} = \mathbf{1},\ \mathbf{P}^\top \mathbf{1} = \mathbf{1} \right\}. \tag{4}$$

We adopt an entropy-regularized formulation to approximate soft permutations within the Birkhoff polytope $\mathcal{B}_N$, solved via Sinkhorn iterations (Mena et al. 2018):

$$\min_{\mathbf{P}\in\mathcal{B}_N} \langle \mathbf{C}, \mathbf{P} \rangle + \varepsilon \sum_{i,j} P_{ij}(\log P_{ij} - 1), \tag{5}$$

where ε is the temperature parameter controlling the entropy strength. The solution has a closed-form structure $\mathbf{P} = \mathrm{Diag}(u)\,\mathbf{K}\,\mathrm{Diag}(v)$, with $\mathbf{K} = \exp(-\mathbf{C}/\varepsilon)$, and the scaling vectors u and v are iteratively updated (in the log domain) to ensure numerical stability and convergence.

This soft matching mechanism, known as Sinkhorn-Pop (Knight 2008; Mena et al. 2018), provides a differentiable and numerically stable approximation of the optimal permutation. It circumvents the need for non-differentiable alternatives such as the Hungarian algorithm combined with straight-through estimation (STE). During training, the temperature ε is gradually annealed to sharpen the relaxed permutation matrix towards discreteness. At inference time, the final discrete permutation is recovered by solving the original assignment problem using the Hungarian algorithm.

End-to-End Optimization Objectives. To jointly optimize channel permutation for structured pruning, we optimize the framework using a composite loss that combines task-level supervision with intermediate feature alignment. Specifically, the total objective includes two components: a *task-level cross-entropy loss* and a *layer-wise distillation loss*, encouraging both strong downstream performance and internal structural consistency.

The task-level loss directly optimizes the pruned model's predictions. Let $f_{\text{perm+prune}}(x)$ denote the output of the model after applying the learned permutation and N:M structured pruning. The cross-entropy loss is given by:

$$\mathcal{L}_{\text{task}} = \mathrm{CE}(f_{\text{perm+prune}}(x),\ y), \tag{6}$$

where $\mathrm{CE}(\cdot)$ is the cross-entropy loss and y is the ground-truth label. This objective ensures the learned permutation contributes to preserving task-level accuracy.

To preserve semantic consistency at the feature level, we incorporate a layer-wise distillation loss that aligns intermediate representations between the original and pruned models. Let h_l^{orig} and $h_l^{\text{perm+prune}}$ denote the output features of the l-th layer in the original and permuted-pruned models, respectively. The distillation loss is defined as:

$$\mathcal{L}_{\text{distill}} = \sum_{l=1}^{L} \|h_l^{\text{orig}} - h_l^{\text{perm+prune}}\|_2^2, \tag{7}$$

where L is the number of pruned layers. This loss promotes the retention of task-relevant features despite structural modifications. And the final training objective combines both losses as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \alpha\, \mathcal{L}_{\text{distill}}, \tag{8}$$

where α balances the two losses. This joint objective provides both global supervision and local guidance for effective permutation learning under structured sparsity.

Training and Inference Details

Training. We begin by collecting input feature statistics for each layer of the pretrained Transformer, which are later used by the structured pruning method (Wanda) adopted in our framework. After this preprocessing stage, all Transformer weights are frozen, and only the parameters of the permutation cost predictor remain trainable.

To preserve structural consistency in attention layers, we impose a synchronized permutation constraint across coupled projection matrices, such as $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$. These matrices share the same input representation, and apply consistent permutations. By enforcing a shared permutation across structurally dependent components, our method ensures correctness and preserves the potential for real-world acceleration under structured sparsity.

Inference. At inference time, the learned permutation cost predictor is frozen. For each group of channels, we can obtain the optimal permutation matrix via our bipartite matching solver. These permutations are applied to reorder the weights of each layer, followed by the application of N:M sparsity masks generated by the structured pruning method (Wanda). The resulting pruned model, with permuted and sparsified weights, is then used for standard inference.

Experiments

Experimental Setup

Models. We evaluate our framework for structured sparsity on several Transformer backbones and task domains. For vision domain, we select `ViT-Base/16` and `ViT-Large/14` (Dosovitskiy et al. 2020) for experiments. For language domain, we employ `LLaMA-3.2-1B` (Dubey et al. 2024) and `LLaMA-2-7B` (Touvron et al. 2023) to represent the common small and large language models. For multimodal domain, we chose `Qwen2.5-VL-3B` (Bai et al. 2025) as the object of study.

Model	Sparsity	Method	Top-1 (%)	Top-5 (%)
ViT-Base/16	0%	Dense	79.1	94.1
	2:4	CP	66.2	86.4
		Wanda	65.8	86.4
		RIA	66.6	86.6
		Ours(Wanda)	**67.9**	**87.9**
	4:8	CP	66.8	88.2
		Wanda	66.4	86.6
		RIA	71.4	89.9
		Ours(Wanda)	**71.8**	**90.2**
ViT-Large/14	0%	Dense	85.2	97.9
	2:4	CP	79.5	94.7
		Wanda	79.2	95.2
		RIA	79.6	95.1
		Ours(Wanda)	**80.7**	**95.8**
	4:8	CP	82.0	96.4
		Wanda	82.0	96.4
		RIA	82.3	96.4
		Ours(Wanda)	**82.7**	**96.5**

Table 1: Performance of different approaches on ImageNet after pruning ViT-Base/16 and ViT-Large/14 to the 2:4 and 4:8 structured pattern (50 % non-zero weights).

Datasets. For the vision Transformers, we use the canonical `ImageNet-1K` dataset (Deng et al. 2009), which consists of 1.28M training images and 50K validation images. All models are trained with the official train/validation split with standard input resolution of 224×224. For the language models, permutations are learned on the training set of `C4` dataset (Raffel et al. 2020) and Alpaca-en dataset (Taori et al. 2023), which comprises approximately 806MB of cleaned English text. For the vision-language models, permutations are learned on the training set of Alpaca-en dataset (Taori et al. 2023) and LLaVA-Instruct dataset (Liu et al. 2023) dataset, which is a set of GPT-generated multimodal instruction-following data.

Baselines. Our method is compared with a variety of classic and state-of-the-art pruning baselines, including Magnitude (Han et al. 2015), Wanda (Sun et al. 2023a), SparseGPT (Frantar and Alistarh 2023), PrunerZero (Dong et al. 2024), CP (Pool and Yu 2021), and RIA (Zhang et al. 2023).

Evaluations. For vision evaluations, we use the standard `ImageNet-1K` validation set with top-1/5 accuracy as the main metric. For language models, we measure perplexity on the `WikiText2` test set (Merity et al. 2016). To evaluate compression effects across tasks, we report zero-shot accuracy on `ARC` (Clark et al. 2018), `BoolQ` (Christopher et al. 2019), `HellaSwag` (Zellers et al. 2019), `OpenBookQA` (Mihaylov et al. 2018), and `WinoGrande` (Sakaguchi et al. 2021), along with 5-shot accuracy on `MMLU` (Hendrycks et al. 2020). For multimodal tasks, we report zero-shot accuracy on `MMMU` (Yue et al. 2024), `MMStar` (Chen et al. 2024), and `TextVQA` (Singh et al. 2019).

Implementation Details. We use Wanda as our mask generator by default. Vision models are trained for 20 epochs on 2 AMD MI250 GPUs with AdamW (weight decay 0.01, base learning rate 0.1) under a cosine decay schedule and no warm-up, which takes around 4 hours. Language models and vision-language models are trained for 20 and 10 update

Model	Method	Wikitext2	Arc-Easy	Arc-Challenge	BoolQ	HellaSwag	OpenBookQA	WinoGrande	MMLU	Average
LLaMA-3.2-1B (2:4)	Dense	9.06	65.36	31.40	63.82	47.73	26.60	60.69	31.19	46.68
	Magnitude	4808.42	27.95	19.45	38.50	26.13	11.80	51.78	23.80	28.49
	SparseGPT	**32.20**	**45.29**	20.65	62.11	**31.99**	15.20	54.54	24.68	**36.35**
	Wanda	75.76	37.16	18.17	62.05	28.57	12.00	50.20	24.45	33.23
	PrunerZero	141.40	36.70	19.28	57.43	27.72	13.40	50.12	25.72	32.91
	CP	68.17	38.35	18.14	62.08	28.56	12.40	53.31	23.82	33.81
	RIA	72.56	38.12	20.33	61.34	27.12	12.80	52.76	24.31	33.83
	Ours(Wanda)	45.32	42.26	**21.08**	**62.11**	29.12	**15.60**	**54.85**	**26.27**	35.90
LLaMA-2-7B (2:4)	Dense	5.12	76.30	43.43	77.68	57.14	31.40	69.06	45.84	57.26
	Magnitude	52.00	61.87	30.20	59.79	45.42	21.80	60.93	26.87	43.84
	SparseGPT	10.30	64.10	**32.51**	**71.25**	**43.35**	**25.00**	**67.25**	**28.56**	**47.43**
	Wanda	11.38	62.75	30.38	67.65	41.18	23.60	62.59	27.82	45.14
	PrunerZero	12.91	61.20	27.47	66.15	39.43	24.40	61.01	27.41	43.87
	CP	10.68	63.32	30.96	66.92	41.32	23.80	63.56	26.51	45.20
	RIA	10.52	63.67	31.82	67.13	42.03	23.00	64.13	27.56	45.62
	Ours(Wanda)	**10.17**	**64.23**	32.00	68.17	43.31	23.60	63.77	28.13	46.17

Table 2: Comparisons of different pruning methods on the LLaMA3.2-1B and LLaMA2-7B language models. Performance across more sparsity patterns can be found in the supplementary materials.

Sparsity	Method	MMMU	MMStar	TextVQA	Average
0%	Dense	53.1	55.8	79.3	62.7
2:4	Magnitude	34.1	48.7	76.5	53.1
	Wanda	37.2	51.2	77.2	55.2
	RIA	37.3	51.4	77.1	55.3
	Ours(Wanda)	**38.1**	**51.9**	**77.8**	**55.9**

Table 3: Performance of Qwen2.5-VL-3B under different pruning methods.

Epochs	Wikitext2	Arc-Easy	Arc-Chall.	MMLU	Average
0	11.38	62.75	30.38	27.82	40.32
1	10.56	62.88	30.89	27.61	40.46
2	10.31	63.13	31.06	27.96	40.72
5	10.27	64.10	31.91	28.05	41.36
10	10.21	**64.23**	31.88	28.12	41.42
20	**10.17**	64.23	**32.00**	**28.13**	**41.46**

Table 4: Performance over training epochs on LLaMA-2-7B.

epochs, respectively, with AdamW (lr $= 10^{-4}$). Training takes approximately 10 hours for a 1B model and 40 hours for a 7B model. The default permutation group number G is 4. All runs use native AMP with FP16 precision, gradient accumulation set to one, and a small Smooth-L1 distillation term weighted 10^{-5}. The default N:M sparsity is 2:4.

Comparison with State-of-the-Art Methods

Results on Vision Transformers. On the vision side, we apply structured pruning to `ViT-Base/16` and `ViT-Large/14` under 2:4 and 4:8 sparsity, retaining 50% of weights. Our method consistently achieves the highest top-1 and top-5 accuracy across all settings. For `ViT-Base/16`, it reaches 67.9% / 87.9% (top-1 / top-5) at 2:4 sparsity, outperforming the strong permutation baseline (RIA) by 1.3 points. At 4:8 sparsity, accuracy improves to 71.8% / 90.2%, ahead of RIA by 0.4 / 0.3 points. Similar gains are observed on `ViT-Large/14`, achieving 80.7% / 95.8% at 2:4 and 82.7% / 96.5% at 4:8 sparsity, both surpassing prior methods. These results demonstrate the accuracy-preserving strength of our approach on Vision Transformers.

Results on Language Transformers. On the language side, we evaluate pruning performance on both `LLaMA-3.2-1B` and `LLaMA-2-7B` models across a range of context modeling and commonsense benchmarks. Our method—using Wanda pruning followed by a learned channel permutation—consistently improves downstream performance over baseline methods without requiring weight updates. On `LLaMA-3.2-1B`, our approach improves the average accuracy to 35.90%, outperforming Wanda (33.23%), CP (33.81%), and RIA (33.83%) by 2.1 to 2.7 points. Similar trends are observed on `LLaMA-2-7B`, where our method reaches 46.17% average accuracy, compared to 45.14% (Wanda), 45.20% (CP), and 45.62% (RIA). Notably, our method improves performance on challenging tasks such as `ARC-Challenge` (e.g., 3.79 points over Wanda) and maintains strong results on `BoolQ` and `WinoGrande`. While methods like SparseGPT achieve higher accuracy due to weight updates, our approach operates in the same constrained setting as Wanda, offering a fair and efficient comparison. Moreover, on `WikiText2`, our method yields better perplexity than baselines, demonstrating its ability to preserve language modeling quality.

Results on Multimodal Transformers. As shown in Table 3, our method achieves the highest performance on the `MMMU` benchmark using the `Qwen2.5-VL-3B` model. While baseline methods such as Magnitude, Wanda, and RIA yield scores of 34.1, 37.2, and 37.3 respectively, our approach outperforms with a score of 38.1. For Transformer-based models of various types, the learnable permutation consistently lifts performance beyond magnitude-only or rule-based permutation strategies, and does so while preserving the full latency advantage of 2:4 sparsity.

Training Convergence and Efficiency

To assess convergence, we report performance from epoch 1 to 20 in Table 4. The initial performance is limited (average 33.23; `Wikitext2` perplexity 75.76), but improves markedly after one epoch, reaching 40.47 and 10.56 respectively, reflecting an 86% reduction in perplexity and indicating fast optimization of the permutator. Subsequent gains taper off: 40.71 at epoch 2, 41.36 at epoch 5, 41.41 at

Number of Groups	Trainable Parameters	Wikitext2	Arc-Easy	Arc-Chall.	BoolQ	HellaSwag	OpenBookQA	WinoGrande	MMLU	Average
1	1.0×	10.11	64.90	32.00	68.01	43.56	23.80	63.93	28.07	46.33
2	0.68×	10.12	64.35	31.83	68.20	43.82	23.60	64.01	28.25	46.30
4	0.41×	10.17	64.23	32.00	68.17	43.31	23.60	63.77	28.13	46.17
8	0.23×	10.25	64.14	31.83	68.20	43.15	23.60	63.85	28.15	46.13
16	0.12×	10.63	63.59	31.14	67.92	42.89	23.00	63.69	27.67	45.71

Table 5: Performance of our approach on LLaMA-2-7B with different permutation group numbers.

Model	Pruning Methods	Wikitext2(Ours/Baseline)	Arc-Easy	Arc-Chall.	BoolQ	HellaSwag	OpenBookQA	WinoGrande	MMLU	Average(Ours/Baseline)
LLaMA-2-7B	Magnitude(Ours)	45.82 /52.00	62.88	30.89	61.68	**45.47**	22.20	62.12	27.12	44.63 /43.84
	Wanda(Ours)	10.17/11.38	64.23	32.00	68.17	43.31	23.60	63.77	28.13	46.17/45.14
	RIA(Ours)	10.02/10.52	63.89	32.51	**68.84**	42.59	24.00	64.33	29.02	46.45/45.62
	AdmmPruner(Ours)	**9.56**/9.68	**69.02**	**32.68**	63.39	45.12	**26.00**	**65.98**	**29.62**	**47.40**/47.05

Table 6: Performance of different pruning methods when integrated with our approach on LLaMA-2-7B.

Loss	Wikitext2	Arc-Easy	Arc-Chall.	MMLU	Average
CE Loss	46.15	42.01	20.53	26.12	29.55
Distillation Loss	52.58	40.51	20.12	25.97	28.87
CE+Distillation loss	**45.32**	**42.27**	**21.08**	**26.27**	**29.87**

Table 7: Ablation of optimization loss on LLaMA3.2-1B.

epoch 10, and 41.44 at epoch 20. Task-level improvements show a similar pattern. `Arc-Easy` improves from 37.16 to 64.22, `Arc-Challenge` from 18.17 to 31.96, and `MMLU` from 24.45 to 28.13, with most gains observed after the first epoch, which takes approximately 3 to 4 hours.

We also evaluate other heuristic permutation methods under the same settings, such as the LSA algorithm in RIA and the search-based approach in CP. These methods typically require 5 to 10 times longer to converge compared to our approach. These results suggest that the method converges efficiently, with nearly all achievable performance obtained within five epochs, only 0.08 below the final result at epoch 20, while maintaining low per-epoch computational cost.

Ablation Study

Impact of Permutation Group Number. We investigate how the number of permutation groups G, defined as the granularity of partitioning the weight matrix before learning permutations, affects performance and efficiency. A smaller G enables more global reordering, potentially leading to better permutation masks. A larger G reduces the number of parameters but constrains the search space. As shown in Table 5, we evaluate $G \in \{1, 2, 4, 8, 16\}$ on LLaMA-2-7B. The average score decreases only marginally from 46.33 at $G = 1$ to 46.30, 46.17, 46.13, and 45.71 as the number of groups increases. `Wikitext2` perplexity rises slightly from 10.11 to 10.63. Meanwhile, the number of trainable parameters is significantly reduced. Considering the reduced number of trainable parameters, along with the preservation of `Wikitext2` perplexity and task accuracy, we adopt $G = 4$ as the default setting in this paper. Notably, $G = 8$ is also a viable option when a bigger accuracy drop is acceptable and computational resources are limited.

Performance over Different Pruning Methods. To assess the generality of our approach, we integrate the learnable permutator into four representative pruning baselines: Magnitude, Wanda, RIA, and AdmmPruner. As shown in Table 6, our method enhances performance across all settings. Specifically, the average score improves from 43.84 to 44.63 for Magnitude, from 45.14 to 46.17 for Wanda, and from 45.62 to 46.45 for RIA. The perplexity on `Wikitext2` also decreases in every case—for example, from 52.00 to 45.82 for Magnitude, from 11.38 to 10.17 for Wanda, and from 10.52 to 10.02 for RIA. On `MMLU`, the scores rise from 26.87, 27.82, and 27.56 to 27.12, 28.13, and 29.02, respectively. Furthermore, the permutator is compatible with post-pruning weight updates. When applied to AdmmPruner, it achieves the highest overall average score (47.40), the best `MMLU` accuracy (29.62), and the lowest `Wikitext2` perplexity (9.56). These results demonstrate that the permutator is broadly applicable to both pruning baselines and post-pruning optimization techniques.

Effect of Our Optimization Loss. To evaluate the contributions of our optimization components end-to-end cross entropy (CE) loss and layer wise distillation we conduct an ablation study. As shown in Table 7, CE loss alone achieves an average score of 29.55 and perplexity of 46.15, while distillation alone yields 28.87. Their combination leads to the best results: an average score of 29.87 with top performance on Arc Easy (42.27), Arc Challenge (21.08), and MMLU (26.27), and a low perplexity of 45.32. These findings confirm the complementary benefits of both loss terms.

Conclusion

In this work, we present a novel end-to-end learnable permutation framework to enhance structured sparsity in large-scale transformer-based models. By introducing a differentiable permutation cost predictor and a bipartite matching solver, our approach learns optimal weight reorderings that align better with N:M sparsity constraints. Extensive experiments on vision and language backbones demonstrate that our method consistently outperforms state-of-the-art baselines, offering a powerful and generalizable strategy for model compression with minimal performance loss.

References

Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Boža, V. 2024. Fast and optimal weight update for pruned large language models. *arXiv preprint arXiv:2401.02938*.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *European conference on computer vision*, 213–229. Springer.

Chen, L.; Li, J.; Dong, X.; Zhang, P.; Zang, Y.; Chen, Z.; Duan, H.; Wang, J.; Qiao, Y.; Lin, D.; et al. 2024. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37: 27056–27087.

Christopher, C.; Kenton, L.; Ming-Wei, C.; Tom, K.; Michael, C.; and Kristina, T. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457v1*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dong, P.; Li, L.; Tang, Z.; Liu, X.; Pan, X.; Wang, Q.; and Chu, X. 2024. Pruner-Zero: Evolving Symbolic Pruning Metric From Scratch for Large Language Models. In *International Conference on Machine Learning*, 11346–11374. PMLR.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv e-prints*, arXiv–2407.

Fang, G.; Yin, H.; Muralidharan, S.; Heinrich, G.; Pool, J.; Kautz, J.; Molchanov, P.; and Wang, X. 2024. Maskllm: Learnable semi-structured sparsity for large language models. *arXiv preprint arXiv:2409.17481*.

Floridi, L.; and Chiriatti, M. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30: 681–694.

Frantar, E.; and Alistarh, D. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, 10323–10337. PMLR.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *International Conference on Computer Vision (ICCV)*.

Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; and Steinhardt, J. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Hu, Y.; Zhu, J.; and Chen, J. 2024. S-ste: Continuous pruning function for efficient 2: 4 sparse pre-training. *Advances in Neural Information Processing Systems*, 37: 33756–33778.

Knight, P. A. 2008. The Sinkhorn–Knopp algorithm: convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1): 261–275.

Li, J.; Xu, Y.; Huang, H.; Yin, X.; Li, D.; Ngai, E. C.; and Barsoum, E. 2025. Gumiho: A Hybrid Architecture to Prioritize Early Tokens in Speculative Decoding. *arXiv preprint arXiv:2503.10135*.

Liao, H.; Xu, Y.; He, S.; Li, G.; Yin, X.; Li, D.; Barsoum, E.; Zhao, J.; and Liu, K. 2025. SparK: Query-Aware Unstructured Sparsity with Recoverable KV Cache Channel Pruning. *arXiv preprint arXiv:2508.15212*.

Liu, H.; Li, C.; Wu, Q.; and Lee, Y. J. 2023. Visual instruction tuning. *Advances in neural information processing systems*, 36: 34892–34916.

Liu, H.; Saha, R.; Jia, Z.; Park, Y.; Huang, J.; Sabach, S.; Wang, Y.-X.; and Karypis, G. 2025. PROXSPARSE: REGULARIZED LEARNING OF SEMI-STRUCTURED SPARSITY MASKS FOR PRETRAINED LLMS. In *Forty-second International Conference on Machine Learning*.

Lu, Y.; Agrawal, S.; Subramanian, S.; Rybakov, O.; De Sa, C.; and Yazdanbakhsh, A. 2023. Step: Learning n: M structured sparsity masks from scratch with precondition. In *International Conference on Machine Learning*, 22812–22824. PMLR.

Ma, X.; Fang, G.; and Wang, X. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Advances in Neural Information Processing Systems*.

Mena, G.; Belanger, D.; Linderman, S.; and Snoek, J. 2018. Learning Latent Permutations with Gumbel-Sinkhorn Networks. In *International Conference on Learning Representations*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843.

Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *EMNLP*.

Peebles, W.; and Xie, S. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, 4195–4205.

Pool, J.; Sawarkar, A.; and Rodge, J. 2021. 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*.

Pool, J.; and Yu, C. 2021. Channel permutations for n: m sparsity. *Advances in neural information processing systems*, 34: 13316–13327.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9): 99–106.

Singh, A.; Natarajan, V.; Shah, M.; Jiang, Y.; Chen, X.; Batra, D.; Parikh, D.; and Rohrbach, M. 2019. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8317–8326.

Sun, M.; Liu, Z.; Bair, A.; and Kolter, J. Z. 2023a. A Simple and Effective Pruning Approach for Large Language Models. *arXiv preprint arXiv:2306.11695*.

Sun, M.; Liu, Z.; Bair, A.; and Kolter, J. Z. 2023b. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. arXiv:1706.03762.

Xia, M.; Gao, T.; Zeng, Z.; and Chen, D. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Yuan, L.; Chen, Y.; Wang, T.; Yu, W.; Shi, Y.; Jiang, Z.-H.; Tay, F. E.; Feng, J.; and Yan, S. 2021. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, 558–567.

Yue, X.; Ni, Y.; Zhang, K.; Zheng, T.; Liu, R.; Zhang, G.; Stevens, S.; Jiang, D.; Ren, W.; Sun, Y.; et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9556–9567.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhang, Y.; Bai, H.; Lin, H.; Zhao, J.; Hou, L.; and Cannistraci, C. V. 2023. Plug-and-Play: An Efficient Post-training Pruning Method for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Zhang, Y.; Bai, H.; Lin, H.; Zhao, J.; Hou, L.; and Cannistraci, C. V. 2024. Plug-and-Play: An Efficient Post-training Pruning Method for Large Language Models. In *12th International Conference on Learning Representations (ICLR 2024)*.

Zhou, A.; Ma, Y.; Zhu, J.; Liu, J.; Zhang, Z.; Yuan, K.; Sun, W.; and Li, H. 2021. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*.