

ELSA: Exploiting Layer-wise N:M Sparsity for Vision Transformer Acceleration

Ning-Chi Huang¹, Chi-Chih Chang¹, Wei-Cheng Lin¹,
Endri Taka², Diana Marculescu², and Kai-Chiang Wu¹

¹National Yang Ming Chiao Tung University, ²University of Texas at Austin

nchuang@cs.nycu.edu.tw, brian1009.en08@nycu.edu.tw, weicheng.lin.cs11@nycu.edu.tw,
endri.taka@utexas.edu, dianam@utexas.edu, kcw@cs.nctu.edu.tw

Abstract

N:M sparsity is an emerging model compression method supported by more and more accelerators to speed up sparse matrix multiplication in deep neural networks. Most existing N:M sparsity methods compress neural networks with a uniform setting for all layers in a network or heuristically determine the layer-wise configuration by considering the number of parameters in each layer. However, very few methods have been designed for obtaining a layer-wise customized N:M sparse configuration for vision transformers (ViTs), which usually consist of transformer blocks involving the same number of parameters.

In this work, to address the challenge of selecting suitable sparse configuration for ViTs on N:M sparsity-supporting accelerators, we propose ELSA, Exploiting Layer-wise N:M Sparsity for ViTs. Considering not only all N:M sparsity levels supported by a given accelerator but also the expected throughput improvement, our methodology can reap the benefits of accelerators supporting mixed sparsity by trading off negligible accuracy loss with both memory usage and inference time reduction for ViT models. For instance, our approach achieves a noteworthy 2.9× reduction in FLOPs to both Swin-B and DeiT-B with only a marginal degradation of accuracy on ImageNet. Our code is publicly available at <https://github.com/ningchihuang/ELSA>.

1. Introduction

In recent years, transformer-based neural networks have been modified for artificial intelligence tasks that include not only natural language processing but also computer vision, such as image classification [5, 18, 32] and object detection [3, 39]. Consisting of a series of transformer blocks that can effectively capture dependencies between patches in a given image, transformer-based neural networks demonstrate outstanding performance on vision tasks and replace convolutional neural networks (CNNs) as the state-of-the-art. For

example, DeiT-B can achieve a Top-1 accuracy of 81.8% on the ImageNet dataset by utilizing 12 transformer blocks and 17.6G parameters. However, it is challenging to deploy such huge models on smartphones or embedded devices, given their limited memory budget and computational resources. Furthermore, most of these devices are power-constrained.

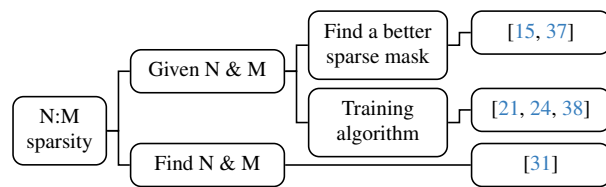


Figure 1. Three categories of methodologies for $N:M$ semi-structured pruning

Various model compression methods have been proposed to reduce the requirements of memory usage and computational cost for model inference, such as quantization [17, 19] and pruning/sparsifying [4, 10, 14, 34, 36]. To maintain the application accuracy, unstructured pruning methods [6, 8, 10, 25] have been presented to remove neurons or connections from a deep neural network (DNN) that do not significantly impact accuracy and only retain important weights for computation. On the other hand, the remaining data in the weight matrix are irregularly distributed and require a high cost of encoding/indexing which induces overhead when these compressed data are sent from memory to processing units. In contrast, structured pruning methods [14, 23, 34] are able to remove rows, columns, or channels that do not significantly impact the accuracy of a DNN model. Although structured pruning methods have little or no memory overhead on data encoding, the application accuracy decreases dramatically when a larger compression ratio is applied for reducing the model size. To overcome the problems mentioned above, fine-grained structured pruning (also called semi-structured pruning) has been introduced for better trade-off among the model size, compression overhead and application accuracy.

Utilizing $N:M$ sparsity is a type of fine-grained structured pruning method which splits every contiguous M data chunks into a group of which only N out of the M in each group are kept for computation (*i.e.*, the other $M-N$ data in the group are pruned) [24, 31, 37, 38]. Resulting sparse data representations have a lower cost on data encoding (each remaining data only needs $\log_2 M$ bits for the index). Compared to unstructured pruning, $N:M$ sparsity is considered to be a hardware-friendly method for model compression.

As illustrated in Fig. 1, methodologies for converting a dense neural network to its $N:M$ sparse counterpart can be classified into three categories: (1) Methodologies focusing on finding a better sparse mask with a given $N:M$, such as CAP [15] and LBC [37], precisely estimate the importance (or the sensitivity to pruning) of each weight and decide the mask based on the importance score; (2) Methods that obtain the target $N:M$ sparse network through sparse training algorithms or strategies, such as ASP [24], SR-STE [38], and STEP [21], are usually proposed to reduce the accuracy loss induced by fine-grained structured pruning by retraining the neural network or updating the weights; (3) Different from the previous two categories, which typically enforce a given/fixed sparsity level across all layers in the whole neural network, *i.e.*, uniform sparsity, methods in the third category find the best $N:M$ sparse configuration for different layers in the network, *i.e.*, layer-wise $N:M$ sparsity. To the best of our knowledge, DominoSearch [31] is the only methodology employing $N:M$ layer-wise sparsity currently.

However, very few methods have been designed for obtaining a layer-wise customized $N:M$ sparse configuration for vision transformers (ViTs). Most of the methods targeting layer-wise sparsity for CNNs decide the compression ratio or the sparsity setting by considering the number of neurons or parameters in each layer [6, 25, 31]. Those methods compress the layers containing more parameters with higher sparsity; in contrast, the layers with fewer parameters will have lower resulting sparsity. Because ViTs usually consist of transformer blocks involving the same number of parameters, deciding the layer-wise $N:M$ sparse configuration according to the number of parameters is likely to have limited impact on ViTs.

To this end, we propose ELSA, a sparsity exploration framework for exploiting layer-wise $N:M$ sparsity on accelerating ViTs. Considering not only all $N:M$ sparsity levels supported by a given accelerator but also the expected throughput improvement, our methodology can reap the benefits of accelerators supporting mixed sparsity by trading off negligible accuracy loss with both memory usage and inference time reduction for ViT models. Additionally, our approach can obtain multiple target models with varying compression ratios through a single training process.

Our contributions are summarized as follows:

- To the best of our knowledge, this is the first work explor-

ing layer-wise $N:M$ sparse configurations for the linear modules (matrices in linear projection and multi-layer perceptron layers) in vision transformers.

- Our proposed methodology can consider the $N:M$ sparsity levels supported by a given accelerator and search layer-wise $N:M$ sparse neural networks with high accuracy. On the other hand, the sparse configurations obtained by our method can provide an insight about the practicality of different $N:M$ sparsity levels for hardware designers to build accelerators supporting mixed $N:M$ sparsity.
- Considering not only the application accuracy but also the hardware efficiency, our methodology can obtain multiple layer-wise sparse configurations with high accuracy and a significant reduction in FLOPs, which is highly correlated to the throughput improvement on mixed $N:M$ sparsity-supporting accelerators. For instance, our approach achieves a noteworthy $2.9\times$ reduction in FLOPs to DeiT-B with only a marginal degradation of accuracy on ImageNet classification.

2. Preliminaries and Related Work

To support DNN models compressed via $N:M$ sparsity, several accelerators have been presented [1, 7, 12, 20, 28]. For example, the Sparse Tensor Cores in NVIDIA Ampere GPU can support 2:4 sparsity which allows a DNN model to halve its parameter counts and ideally achieve $2\times$ speedup on the operations of compressed 2:4 sparse matrices [28]. In addition, the architectures presented in S2TA [20] and VEGETA [12] support 4:8 weight sparsity and configurable $N:4$ sparsity in their systolic array-based accelerators, respectively, to yield speedup and energy efficiency.

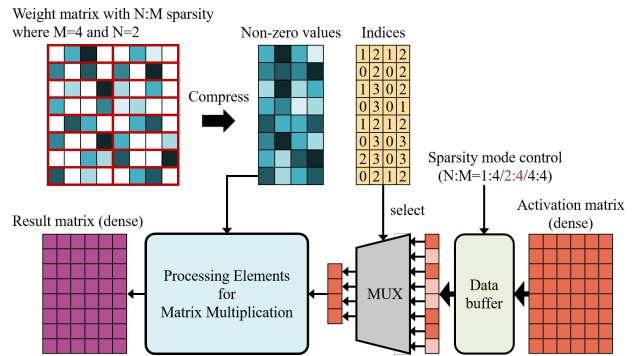


Figure 2. Accelerator for mixed sparsity where weight matrix are pruned by $N:M$ semi-structured pruning

Fig. 2 illustrates a conceptual view of a configurable accelerator with $N:M$ sparsity support. The upper part of Fig. 2 shows a weight matrix compressed by 2:4 sparsity. The white grids in the matrix denote values that are less significant in their 4-element group, and thus are pruned to zero. With 2:4 sparsity being applied, only the most significant two

weights in the group will be retained after compressing the matrix into sparse format, where non-zeros and their 2-bit indices (for distinguishing the original position of the non-zeros from the 4-element group in $N:4$ sparsity) are stored. Therefore, not only the remaining weights but also the overhead of the corresponding data for indexing can be reduced, thereby making $N:M$ sparsity more hardware-friendly than unstructured pruning.

The bottom part of Fig. 2 shows how to produce the accurate result of matrix multiplication from a sparse weight matrix and a dense activation matrix. By decoding the indices of the remaining N non-zeros from each original M -element group in the weight matrix, the corresponding activation/input data for matrix multiplication can be selected and loaded into the processing units. Compared to the corresponding dense matrix multiplication, when the sparsity level of the current weight matrix is 2:4, only 50% of the multiply-and-accumulate operations in the matrix multiplication will be performed. With indices of sparse matrix, control signals, associate MUXs and processing elements for sparse matrix multiplication, accelerators supporting $N:M$ sparsity can effectively accelerate matrix multiplication.

If an accelerator supporting various levels of $N:M$ sparsity (e.g., $N:4$ sparsity) became available, it would be possible to obtain higher throughput by using customized, layer-wise $N:M$ sparsity without significant accuracy loss (compared to aggressively applying uniform sparsity, e.g., 1:4 sparsity). In this scenario, an effective method for finding a layer-wise sparsity setting for different DNNs on the accelerator becomes a necessity.

However, it is challenging to apply a suitable sparse configuration to ViT models so as to achieve speedups from mixed sparsity-supporting accelerators with negligible accuracy loss. For example, when a ViT model including 48 weight matrices is served on an accelerator supporting 1:4, 2:4 sparse matrix multiplication, and 4:4 dense matrix multiplication, there will be 3^{48} combinations for the resulting sparsity configurations. Furthermore, when a higher compression ratio is applied to accelerate operations, some essential values in the matrix are probable to be pruned, which causes non-negligible accuracy loss. Accordingly, mitigating this accuracy loss necessitates extensive fine-tuning over a substantial number of epochs to maintain the accuracy of the compressed model. Therefore, an efficient methodology is required for determining a suitable layer-wise sparse configuration among the many possible combinations without resorting to exhaustive search and fine-tuning.

3. Methodology

Problem Formulation Assuming the availability of an accelerator supporting mixed $N:M$ sparsity levels for matrix multiplication operations, one can apply the $N:M$ semi-structured pruning to the linear modules of ViTs in the pur-

suit of computational efficiency. Consider a pretrained classic ViT comprising B transformer blocks, each including four linear modules, as depicted in Fig. 3. Assuming the accelerator supports a set of K different sparsity levels $\mathcal{S} = \{(N_1:M_1), \dots, (N_K:M_K)\}$, our objective is to determine a sparse configuration (i.e., a sequence of sparsity levels) denoted as $\alpha = (s^1, \dots, s^L)$, where $s^l \in \mathcal{S}$ and $L = 4B$. This configuration α is then used to convert the pretrained weight values, represented as $\mathcal{W} = (W^1, \dots, W^L)$ from a dense to a sparse representation. To harness the acceleration potential of the sparsity-supporting accelerator while minimizing associated performance degradation, it is essential to determine the optimal sparse configuration.

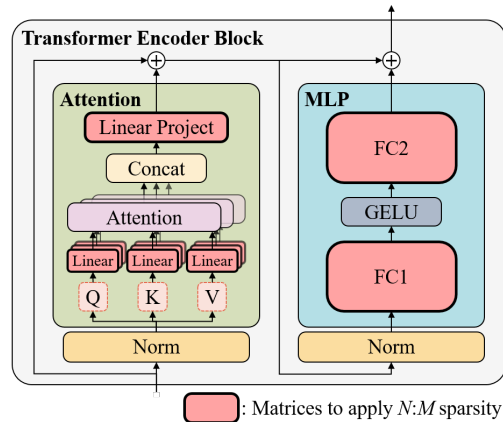


Figure 3. A classic transformer block and the matrices to be sparsified in our methodology

Challenge Deploying a ViT model on accelerators supporting flexible sparse matrix multiplication typically involves two steps: (1) determining the $N:M$ sparsity level for each matrix in the model, followed by (2) fine-tuning the model to mitigate the accuracy loss resulting from sparsification. However, deciding the layer-wise sparsity configurations for ViTs can be more challenging compared to convolution-based models, primarily because ViTs comprise multiple transformer blocks involving the same number of parameters. Consequently, previous heuristic-based approaches to sparsity level selection, as previously employed in the context of convolutional models [6, 16, 25], confront difficulties in selecting configurations that offer both superior accuracy and substantial computational speed-ups on ViTs.

Furthermore, the process of sparsification often results in a significant degradation of accuracy, especially when matrices are pruned in a more structured manner (i.e., through structured or semi-structured pruning). This necessitates extensive fine-tuning over a substantial number of epochs to restore the application accuracy of the given model, enabling a precise evaluation of the chosen sparse configurations.

To address the significant fine-tuning demands, we innovatively adopt the principles of supernet [2, 9, 30]. It was originally designed to avoid training each candidate network from scratch to accurately evaluate the corresponding performance while searching network architecture hyper-parameters, such as the kernel size and the number of channels for CNNs. Through customized encoding strategies, all candidate networks within the search space, referred to as subnets, are encoded into an over-parameterized supernet. These subnets are then trained jointly, effectively amortizing the fine-tuning requirements for evaluating each specific/chosen combination of architecture hyper-parameters. By training only once, we can yield the network corresponding to any specific network configuration by directly inheriting from the supernet without exhaustively training each of them one by one.

In this work, we propose to extend these principles to address the exploration complexities of obtaining suitable layer-wise $N:M$ sparse networks. Specifically, we design a supernet construction scheme aimed at incorporating all candidate $N:M$ sparse networks, each corresponding to an $N:M$ sparse configuration α within the search space $\mathcal{A} = \underbrace{\mathcal{S} \times \mathcal{S} \times \dots \times \mathcal{S}}_L$.

By harnessing this supernet, we can directly derive specific sparse networks without the need for additional fine-tuning. This not only facilitates evaluation of specific sparse configurations α but also expedites their deployment on hardware accelerators to enhance computational throughput.

Detailed description of the algorithms for our supernet construction and training will be provided in the subsequent subsections.

3.1. Supernet Construction – All Sparsity Configurations in One Supernet

We initiate our layer-wise sparsity exploration by first constructing a supernet, which involves the determination of all layers that are utilized to perform matrix multiplication in a ViT model, such as the linear projection and multi-layer perceptron layers. For example, DeiT-S consists of 12 transformer blocks, each containing 4 target layers (*i.e.*, qkv, linear projection, fc1 and fc2); thus, the sparsity configuration of the 48 layers should be determined before deploying a sparse DeiT-S model on an accelerator supporting mixed sparsity. Subsequently, all supported sparsity levels of a given accelerator are encoded as possible choices available to each layer. If an accelerator offers 1:4, 2:4, and 4:4 sparsity options, each target layer within our supernet will contain 3 sparsity choices, where one of them will be chosen at a time, and the corresponding sparsity level will be applied for sparse matrix multiplication when the layer is computed. Moreover, each combination of sparsity choices for a given ViT model is regarded as a sparse configuration,

which can be utilized to derive a sparse subnet within our supernet. After constructing a supernet that contains all sparsity configurations, we train the supernet before searching for a suitable configuration.

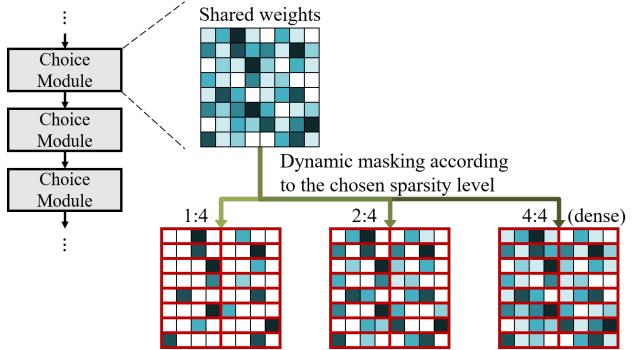


Figure 4. Shared weight values and dynamic masking for each layer in a transformer-based model (including all linear projection and multi-layer perceptron layers)

Subset-Superset Relationship We notice the subset-superset relationship among different $N:M$ sparsity levels and utilize it to construct the supernet. Specifically, the non-zeros selected under higher $N:M$ sparsity level form a subset of those selected under lower ones when the same criteria for pruning is applied. For the $N:4$ sparsity levels illustrated in Fig. 4, the cells of darker color mean their values are more significant; both 1:4 and 2:4 sparsities select their non-zeros according to the significance of those weights. It can be seen that the weight selected when 1:4 sparsity is applied must be also selected when 2:4 sparsity is applied. Furthermore, those weights are all included in the dense matrix (*i.e.*, using 4:4 sparsity).

Leveraging this relationship, instead of building up a supernet that has different instances of sparse weights for each sparsity level in each matrix and thus causing huge memory overhead for the supernet training, we only maintain a stack of weight matrices $\mathcal{W}_{\mathcal{A}} = (W_{sup}^1, \dots, W_{sup}^L)$, where W_{sup}^i denotes the shared weight matrix for the i^{th} linear module. That is, all $N:M$ sparsity choices share the same weight matrix in each layer within our supernet.

Dynamic Masking During the process of both training and evaluation, given a sparse configuration $\alpha = (s^1, \dots, s^L)$, we generate the sparse networks $\mathcal{W}_{\mathcal{A}}(\alpha) = (W_{sup}^1(s^1), \dots, W_{sup}^L(s^L))$ by dynamically projecting the shared parameters W_{sup}^l of l^{th} linear modules into corresponding sparse matrix $W_{sup}^l(s^l)$ according to the specific $N:M$ sparsity level s^l . To achieve this, we treat M consecutive parameters as a group $G = \{w_1, \dots, w_M\}$, where $G \subset W_{sup}^l$. We prune the $M - N$ parameters that have

the least significant saliency score $\rho(\cdot)$, *i.e.*, the importance score for each weight. As Fig. 4 illustrates, when 1:4 sparsity is chosen, three parameters in each 4-element group will be pruned. Mathematically, we define ζ as the N^{th} smallest saliency score among G , and the mask value m_i for weight elements in a group can be calculated as:

$$m_i = \begin{cases} 1, & \text{if } \rho(w_i) \geq \zeta \\ 0, & \text{if } \rho(w_i) < \zeta \end{cases}, \text{ for } i \in \{1, \dots, M\} \quad (1)$$

With binary mask m , we can yield the sparse weights by pruning the parameter $w_i \in G$ to zero if the corresponding mask value m_i is equal to 0. In our framework, the saliency score function $\rho(\cdot)$ can be various kind of metrics, including the weight magnitude and Taylor score [26, 27]. In simplification, we utilize weight magnitude as the default saliency score estimator.

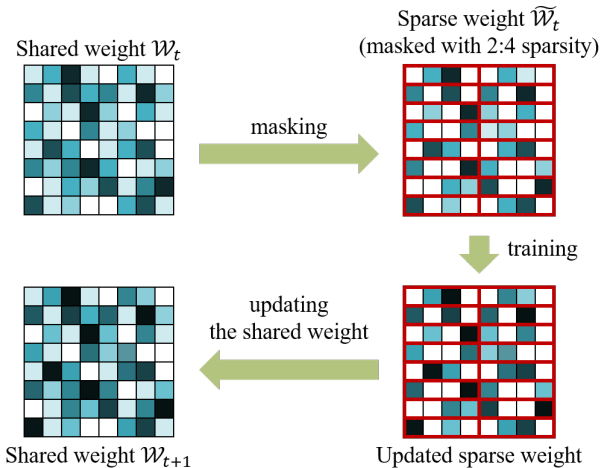


Figure 5. Example of updating the shared weight with 2:4 sparsity being applied

In general, by leveraging the subset-superset relationship, all given sparsity levels can be encoded into our supernet. This approach ensures that the same number of weights in the given neural network is maintained, eliminating the need for multiple weight replicas across different sparsity levels. This not only reduces memory cost but also significantly reduces training overhead, streamlining the computational process. Furthermore, this shared weight approach enhances the convergence of sparse subnetworks, as selecting a particular sparsity level for optimization during supernet training simultaneously trains its subsets and supersets. As illustrated in Fig. 5, if 2:4 sparsity is applied, the corresponding sparse weight matrix will be trained and the shared weight will also be updated.

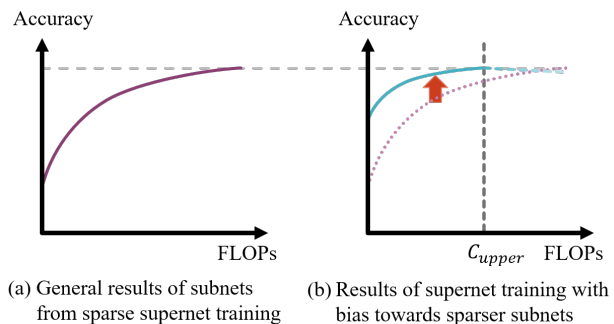
3.2. Supernet Training and Subnetwork Sampling

Supernet training aims to comprehensively train all sparse networks within a designated search space simultaneously. The essence of this process is captured by the following objective function:

$$\mathcal{W}_{\mathcal{A}}^* = \underset{\mathcal{W}_{\mathcal{A}}}{\operatorname{argmin}} \mathbb{E}_{\alpha \sim \Gamma(\mathcal{A})} [\mathcal{L}(\alpha, \mathcal{W}_{\mathcal{A}}(\alpha))], \quad (2)$$

Here, $\mathcal{W}_{\mathcal{A}}$ represents the weights of the supernet, \mathcal{L} denotes the training loss. Each sparse configuration is represented by $\alpha \in \mathcal{A}$, where $\mathcal{W}_{\mathcal{A}}(\alpha)$ denotes the sparse weights corresponding to each configuration α . The objective corresponds to minimizing the expected loss across all existing sparse subnetworks within the search space, \mathcal{A} . In each optimization step, a sparse architecture is sampled from a prior distribution $\Gamma(\mathcal{A})$. Such sparse architectures are trained to minimize the training loss \mathcal{L} and the gradient will be updated back to the shared supernet weights.

Prioritizing sparser subnets In supernet training, managing the complexity of the search space is important. We streamline this process by strategically focusing on (i) configurations that can be converted into sparser networks for obtaining further throughput improvement or (ii) configurations within a user-defined threshold of computational cost, C_{upper} . In practical terms, this means that only configurations with a computation cost $\mathcal{F}(\alpha)$ less than or equal to C_{upper} are included in the sampling process. This tailored adjustment enables the search space to adapt to various practical constraints, such as hardware limitations, making the search more aligned with available computational resources and real-world applicability. In addition, by considering a given C_{upper} value, our supernet training can focus on sparser subnets, which usually require more effort to recover model accuracy. As shown in Fig. 6, the performance of sparser subnets will be higher with C_{upper} being considered due to the prioritization during the supernet training.



(a) General results of subnets from sparse supernet training (b) Results of supernet training with bias towards sparser subnets

Figure 6. Supernet training with a bias towards sparser subnets

Sampling Strategy Before we delve into our proposed prior distribution for candidate configuration, we first review the vanilla approach. In the conventional method [9], all candidate configuration are treated equally. Specifically, sparse configuration are sampled by uniformly selecting the sparsity level s_i from \mathcal{S} for each i^{th} linear module independently.

Leveraging insights from the central limit theorem [22], it becomes evident that under this vanilla sampling paradigm, the computational cost distribution $\mathcal{F}(\alpha)$ will approximately converge to a normal distribution. Given the architecture of our weight-sharing supernet, when a particular sparse configuration α is sampled, similar configurations with nearly identical computational costs are also indirectly trained. This biases the training towards architectures with computational costs clustering around the mean, leaving architectures with extremal computational costs, especially those with higher sparsity, subject to undertraining.

To motivate and ensure a more balanced exploration across the architectural spectrum, we employ a two-step sampling strategy. This process starts by discretizing the computational costs into distinct levels, represented as C_1, C_2, \dots, C_K , where C_1 corresponds to the minimum computational cost (C_{lower}), and C_K corresponds to the user-specified maximum computational cost (C_{upper}). Mathematically, this discretization can be defined by denoting a set of intervals $\mathcal{I} = \{[C_i, C_{i+1}]_{1 \leq i \leq K-1}\}$. With these intervals, the sampling unfolds in two stages: first, a computational cost interval $I = [C_i, C_{i+1}]$ is chosen uniformly from \mathcal{I} , and then, within this selected interval, an architecture is uniformly sampled. Based on this strategic approach, we can ensure the balanced training for architectures of different computational costs, thereby alleviating the undertraining issue for the extremal architectures of high sparsity.

Enhanced Training Recipe Training a supernet is challenging due to the concurrent training of multiple architectures. This complexity necessitates a simplification of the training process to facilitate convergence. Regularization methods (*e.g.*, dropPath, dropout), commonly included in the training process of ViTs, are disabled in our approach. In the dynamic environment of a supernet, where parameters are continuously masked and varied, additional regularization could create training difficulties, obstructing effective learning across the various networks encapsulated within the supernet.

Inspired by prior work in model compression [29, 35], we incorporate knowledge distillation [11], a technique premised on a teacher-student paradigm to further augment and stabilize our supernet training. Specifically, the uncompressed model is adopt to be the teacher in our implementation. During training, a mini-batch of data X is initially forwarded through this uncompressed model to produce prediction logits. Subsequently, the sparse subnetworks sam-

Algorithm 1 Sparse Supernet Training

- Input:** pretrained parameter \mathcal{W} , max iteration T , training data D , learning rate η , computational budget C_{upper}
- Output:** trained supernet weights \mathcal{W}_A^*
- 1: Init supernet’s weight \mathcal{W}_A with pretrained weight \mathcal{W}
 - 2: Define computational cost intervals \mathcal{I} by C_{upper}
 - 3: **for** $t = 1, \dots, T$ **do**
 - 4: Sample a interval $I = [C_i, C_{i+1}]$ w.r.t $I \sim U(\mathcal{I})$
 - 5: Choose an configuration α w.r.t $\alpha \sim U(\{\alpha \mid C_i \leq F(\alpha) < C_{i+1}\})$
 - 6: Derive the sparse weights from $\mathcal{W}_{A,t}$ $\mathcal{W}_{A,t}(\alpha) = (W_{sup,t}^l(s^1), \dots, W_{sup,t}^l(s^L))$
 - 7: Get a training batch X from D
 - 8: Compute gradient via knowledge distillation $g_t = \nabla \mathcal{L}(X, \mathcal{W}_{A,t}(\alpha), \mathcal{W})$
 - 9: Update the weights \mathcal{W}_A using gradient descent $\mathcal{W}_{A,t+1} = \mathcal{W}_{A,t}(\alpha) - \eta g_t$
 - 10: **end for**
 - 11: return trained supernet $\mathcal{W}_{A,T}$
-

pled from the supernet are optimized to mimic the uncompressed model’s behavior by minimizing the cross-entropy loss relative to the teacher-generated logits. This strategy aims to leverage the guidance from the uncompressed model to facilitate the training of the multiple sparse architectures within our supernet.

We summarize the procedure of our proposed supernet training paradigm in Algorithm 1.

3.3. Searching for the Pareto-optimal Solutions

To obtain a suitable sparse configuration from the trained supernet, we rely on evolutionary search to determine a suitable trade-off between accuracy and acceleration. Evolutionary algorithms contain four steps: initialization, mutation, crossover, and selection. First, we initialize the population of subnetworks by randomly sampling several sparse configurations that consist of a sequence of sparsity choices for the ViT. Second, the chosen sparsity level for each layer in a subnetwork is mutated (*e.g.*, the sparsity level is changed from 2:4 to 1:4) with a mutation probability. Third, we perform crossover on multiple pairs of subnetworks in the population by changing partial sparsity settings to generate new sparsity combinations. After that, according to a fitness function, only the sparsity setting with the top- k best fitness score is retained in the population for the next iteration (those with lower scores are removed). After several iterations of mutation, crossover and selection, suitable sparse configurations will be obtained, and the corresponding high-accuracy subnetworks can be extracted from our supernet without further fine-tuning.

4. Experiments

Table 1. Experimental results of the proposed ELSA methodology on various vision transformers. Accuracy denotes the Top-1 accuracy measure on the ImageNet-1K validation set.

Model	Sparsity Pattern	FLOPs	Accuracy
DeiT-S	Dense	4.7G	79.8%
ELSA-DeiT-S-2:4	Uniform 2:4	2.5G (1.00×)	79.1%
ELSA-DeiT-S-N:4	Layer-wise N:4	2.2G (1.14×)	79.0%
		2.0G (1.25×)	78.3%
DeiT-B	Dense	17.6G	81.8%
ELSA-DeiT-B-2:4	Uniform 2:4	9.2G (1.00×)	81.6%
ELSA-DeiT-B-N:4	Layer-wise N:4	7.0G (1.30×)	81.6%
		6.0G (1.53×)	81.4%
Swin-S	Dense	8.7G	83.2%
ELSA-Swin-S-2:4	Uniform 2:4	4.6G (1.00×)	82.8%
ELSA-Swin-S-N:4	Layer-wise N:4	4.0G (1.15×)	82.8%
		3.5G (1.31×)	82.5%
Swin-B	Dense	15.4G	83.5%
ELSA-Swin-B-2:4	Uniform 2:4	8.0G (1.00×)	83.1%
ELSA-Swin-B-N:4	Layer-wise N:4	6.0G (1.33×)	83.0%
		5.3G (1.51×)	82.8%

Dataset and Benchmarks Our experiments are conducted on the ImageNet-1k dataset for image classification tasks. We apply our methodology to DeiT, which includes multiple transformer encoder blocks, and to the improved Swin-Transformer, which employs a hierarchical design and shifted window approach.

Settings The sparse supernet is trained starting from the pretrained weights of the uncompressed models. The training lasts for 150 epochs. For the supernet training, we mostly keep the original hyper-parameters of each compression target. We employ an AdamW [13] optimizer with an initial learning rate of $5e - 5$, a weight decay of 0.005 and a batch size of 1024. We define the computation cost $\mathcal{F}(\cdot)$ as the FLOPs, and user-defined threshold C_{upper} as 50% FLOPs of the dense models in all supernet training. Also, dropout and dropPath [33] are both disabled while training the supernet.

4.1. Pruning Results on ImageNet-1K

In our methodology, we employ N as a power-of-two to define our search space and engage in a cohesive training process of a supernet. This supernet houses various sparse subnetworks. Using evolutionary search, we strategically

navigate the search space to identify the best layer-wise sparsity configurations. Our results are presented in Table 1. All models are directly inherited from the supernet, necessitating no additional adjustments, fine-tuning or post-training.

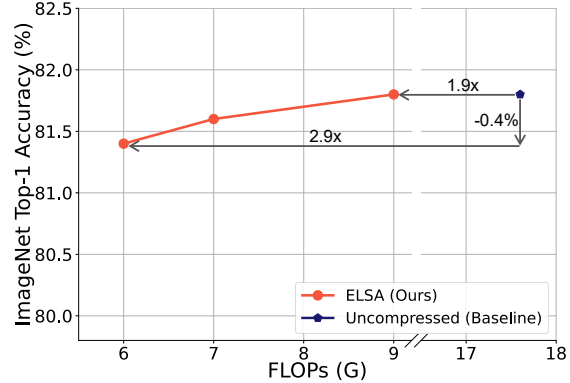


Figure 7. Result of sparsity exploration on DeiT-B

Comparison with the Dense Model (uniform 4:4) As shown in Table 1, our approach demonstrates the capability to significantly reduce FLOPs through exploring $N:M$ sparsity while maintaining model performance. For instance, when applying our pruning strategy to DeiT-B, we achieve a noteworthy 2.9× reduction in computational costs, with only a marginal decline in accuracy as depicted in Fig. 7. Pursuing a FLOP reduction of approximately 50%, our employment of an evolutionary search approach has directed us towards the uniform 2:4 sparsity setting from our well-trained supernet. Furthermore, the sparse network with uniform 2:4 sparsity setting can facilitate an almost twofold reduction in computational costs with only 0.2% reduction in accuracy. Similar efficiency is observed in the case of the Swin-Transformer models, thus confirming the robustness of our methodology.

Benefits of Layer-wise Sparsity The adoption of layer-wise sparsity serves as a foundational aspect of our methodology, presenting notable advantages in comparison to the conventional 2:4 sparse configuration. As demonstrated by the DeiT-B model, our approach, through the exploration of layer-wise sparsity, has achieved improved efficiency, manifesting a 1.53× reduction in FLOPs (6.0G vs. 9.2G). In addition, the reduction in FLOPs is highly correlated to the speedup obtained by deploying the sparsified network on $N:M$ sparsity-supporting accelerators. This result indicates the capacity of layer-wise sparsity to harness heightened rates of hardware acceleration, a trend that is also evident in the Swin-B model. Notably, the Swin-B model reported a 1.51× enhancement in computational cost efficiency (5.3G vs. 8G) as a direct consequence of our method.

It is worth mentioning that the FLOPs reduction yielded by our semi-structured sparsity exploration can theoretically be translated into runtime speed-ups when paired with emerging $N:M$ sparsity-supporting hardware accelerators. Assuming optimal hardware utilization, a network with 2:4 sparsity can attain nearly a $2\times$ speed enhancement on platforms like the NVIDIA Ampere GPU [28]. Likewise, sparse networks employing layer-wise $N:4$ sparsity, such as ELSA-DeiT-B-N:4, could expect speed-ups of up to $2.9\times$ when deployed on VEGETA [12]. Moreover, it is crucial to emphasize the adaptability of our methodology, which allows for the exploration of various semi-structured sparsity patterns. This adaptability facilitates seamless integration with different hardware designs, such as S2TA [20], which targets $N:8$ sparsity, aiding the identification of efficient configurations for developing well-trained sparse networks without compromising accuracy.

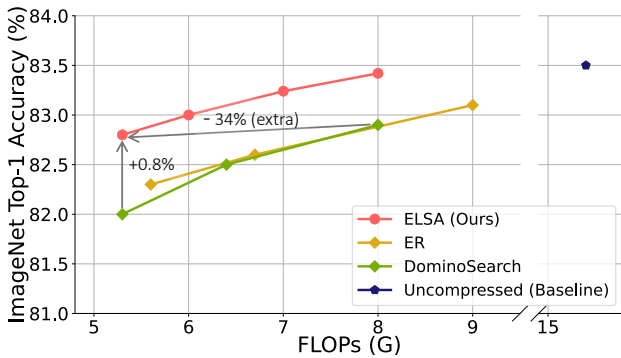


Figure 8. Comparisons of ELSA with other search methods on Swin-B

4.2. Comparison to Other Sparse Configuration Search Methods

To demonstrate the efficacy of our methodology in identifying an optimal layer-wise sparse configuration, we compare our approach to other prevailing sparsity selection paradigms. The methods under comparison are:

- **Erdős-Rényi (ER)** [25]: Originally designed to determine layer-wise unstructured sparsity, ER employs a heuristic based on the sum of input and output channels. We modified ER for semi-structured sparsity by initially deciding the unstructured sparsity level and subsequently rounding it to the closest $N:M$ sparsity level. For instance, within an $N:4$ search space, if the unstructured sparsity level is 70%, we would select 1:4 (approximately 75% zero parameters) as the final semi-structured sparse configuration.
- **DominoSearch** [31]: Tailored specifically for CNNs, DominoSearch is an $N:M$ sparsity level selection approach. During its operation, each cluster of M consecutive weights in a pruning target (*e.g.*, linear layers) is allo-

cated a threshold, which is determined analytically based on weight magnitude. To delve into layer-wise redundancy, DominoSearch adds an extra regularization penalty, which incrementally pushes the preserved weights towards the predetermined threshold until the desired sparsity is attained.

For our experiments, the aforementioned strategies were employed on pretrained ViTs to decide the sparsity level for each targeted linear module. Once the sparsity levels are determined, we derive the sparse sub-networks from the fully-trained supernet and evaluate their performance on the ImageNet-1k validation dataset.

From the visual representation in Fig. 8, it is evident that our approach consistently outperforms the baseline methodologies on the Swin-B architecture. Our methodology exhibits a more favorable trade-off between FLOPs and accuracy. Notably, our method manages to identify a sparsity configuration that achieves a significant extra 34% reduction in FLOPs compared to the DominoSearch strategy, while still maintaining comparable, if not superior, accuracy. Additionally, an improvement of 0.8% in Top-1 Accuracy is observed when aligning the FLOPs. Lastly, the plots emphasize that our results consistently reside on the Pareto frontier, indicating a better balance between computational efficiency and performance. This visual evidence underscores the effectiveness of our proposed layer-wise sparsity exploration technique in ViTs. For a detailed visualization of the sparse configurations searched by the discussed algorithms, please see the corresponding section in the Appendix.

5. Conclusion

This paper presents ELSA, a first-of-its-kind layer-wise $N:M$ sparsity exploration methodology for accelerating ViTs. We address the challenge of selecting suitable layer-wise sparse configurations for ViTs on $N:M$ sparsity-supporting accelerators. Leveraging the subset-superset relationship among $N:M$ sparsity levels, we construct the supernet with all $N:M$ sparsity choices sharing their weights and applying dynamic masking to extract the sparse matrix, resulting in both reduction of training overhead and improvement of convergence of sparse subnets. With our proposed ELSA methodology, we can yield not only sparsified ViT models with high accuracy but also sparse configurations with effective reduction in FLOPs, which is highly correlated to the speedups yielded by mixed sparsity-supporting accelerators. We anticipate that our work will establish a robust baseline for future research in sparse ViT models with $N:M$ sparsity and offer valuable insights for upcoming hardware and software co-design studies.

References

- [1] Abhimanyu Rajesh Bambhaniya, Amir Yazdanbakhsh, Suvinay Subramanian, and Tushar Krishna. Accelerating attention based models via HW-SW co-design using fine-grained sparsification. In *Architecture and System Support for Transformer Models (ASSYST @ISCA 2023)*, 2023. [2](#)
- [2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [4](#)
- [3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. [1](#)
- [4] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, pages 19974–19988, 2021. [1](#)
- [5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. [1](#)
- [6] 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*, pages 2943–2952. PMLR, 2020. [1](#), [2](#), [3](#)
- [7] Chao Fang, Aojun Zhou, and Zhongfeng Wang. An algorithm–hardware co-optimized framework for accelerating n:m sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022. [2](#)
- [8] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. [1](#)
- [9] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, pages 544–560, 2020. [4](#), [6](#)
- [10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. [1](#)
- [11] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. [6](#)
- [12] Geonhwa Jeong, Sana Damani, Abhimanyu Rajeshkumar Bambhaniya, Eric Qin, Christopher J. Hughes, Sreenivas Subramoney, Hyesoon Kim, and Tushar Krishna. Vegeta: Vertically-integrated extensions for sparse/dense gemm tile acceleration on cpus. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 259–272, 2023. [2](#), [8](#)
- [13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. [7](#)
- [14] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Xuan Shen, Geng Yuan, Bin Ren, Hao Tang, Minghai Qin, and Yanzhi Wang. Spvit: Enabling faster vision transformers via latency-aware soft token pruning. In *Computer Vision – ECCV 2022*, pages 620–640, Cham, 2022. Springer Nature Switzerland. [1](#)
- [15] Denis Kuznedelev, Eldar Kurtic, Elias Frantar, and Dan Alistarh. CAP: correlation-aware pruning for highly-accurate sparse vision models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. [1](#), [2](#)
- [16] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020. [3](#)
- [17] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. FQ-ViT: Fully quantized vision transformer without retraining. *CoRR*, 2021. [1](#)
- [18] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, 2021. [1](#)
- [19] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. In *NeurIPS*, pages 28092–28103, 2021. [1](#)
- [20] Zhi-Gang Liu, Paul N. Whatmough, Yuhao Zhu, and Matthew Mattina. S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 573–586, 2022. [2](#), [8](#)
- [21] 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, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, pages 22812–22824. PMLR, 2023. [1](#), [2](#)
- [22] James E. Marengo, David L. Farnsworth, and Lucas Stefanic. A geometric derivation of the irwin-hall distribution. *Int. J. Math. Math. Sci.*, 2017:3571419:1–3571419:6, 2017. [6](#)
- [23] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. [1](#)
- [24] Asit K. Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *CoRR*, abs/2104.08378, 2021. [1](#), [2](#)
- [25] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018. [1](#), [2](#), [3](#), [8](#)

- [26] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 5
- [27] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 11264–11272. Computer Vision Foundation / IEEE, 2019. 5
- [28] Nvidia. A100 tensor core gpu architecture. Technical report, 2020. 2, 8
- [29] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, pages 13937–13949, 2021. 6
- [30] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: designing hardware-efficient convnets in less than 4 hours. In *ECML PKDD*, 2019. 4
- [31] Wei Sun, Aojun Zhou, Sander Stuijk, Rob G. J. Wijnhoven, Andrew Nelson, Hongsheng Li, and Henk Corporaal. Dominosearch: Find layer-wise fine-grained n:m sparse schemes from dense neural networks. In *Advances in Neural Information Processing Systems*, 2021. 1, 2, 8
- [32] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers and distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021. 1
- [33] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1058–1066, Atlanta, Georgia, USA, 2013. PMLR. 7
- [34] Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width & depth pruning for vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3143–3151, 2022. 1
- [35] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *ICLR*, 2022. 6
- [36] Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. *CoRR*, abs/2204.07154, 2022. 1
- [37] 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. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 1, 2
- [38] 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. 1, 2
- [39] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable {detr}: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021. 1