# SparseViT: Revisiting Activation Sparsity for Efficient High-Resolution Vision Transformer

Xuanyao Chen[1,2,*] Zhijian Liu[4,*] Haotian Tang[4] Li Yi[1,3] Hang Zhao[1,3] Song Han[4]
[1]Shanghai Qi Zhi Institute [2]Fudan University [3]Tsinghua University [4]MIT

https://sparsevit.mit.edu

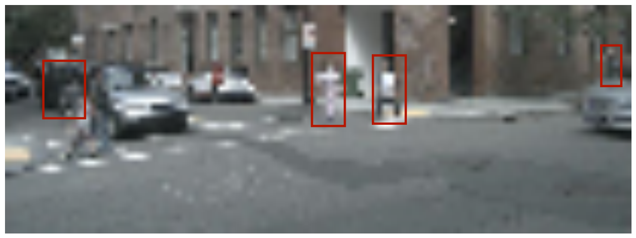
## Abstract

*High-resolution images enable neural networks to learn richer visual representations. However, this improved performance comes at the cost of growing computational complexity, hindering their usage in latency-sensitive applications. As not all pixels are equal, skipping computations for less-important regions offers a simple and effective measure to reduce the computation. This, however, is hard to be translated into actual speedup for CNNs since it breaks the regularity of the dense convolution workload. In this paper, we introduce **SparseViT** that revisits activation sparsity for recent window-based vision transformers (ViTs). As window attentions are naturally batched over blocks, actual speedup with window activation pruning becomes possible:* i.e., *~50% latency reduction with 60% sparsity. Different layers should be assigned with different pruning ratios due to their diverse sensitivities and computational costs. We introduce sparsity-aware adaptation and apply the evolutionary search to efficiently find the optimal layerwise sparsity configuration within the vast search space. SparseViT achieves speedups of **1.5**×, **1.4**×, and **1.3**× compared to its dense counterpart in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with negligible to no loss of accuracy.*

## 1. Introduction

With the advancement of image sensors, high-resolution images become more and more accessible: *e.g.*, recent mobile phones are able to capture 100-megapixel photos. The increased image resolution offers great details and enables neural network models to learn richer visual representations and achieve better recognition quality. This, however, comes at the cost of linearly-growing computational complexity, making them less deployable for resource-constrained applications (*e.g.*, mobile vision, autonomous driving).

∗ indicates equal contributions (listed in alphabetical order).


(a) **Direct Downsample**: Lower Resolution (**0.5x**), Dense (**100%**)

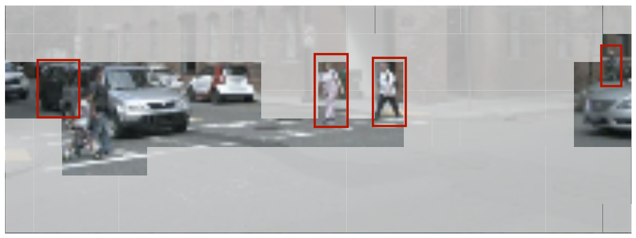

(b) **Window Activation Pruning**: Higher Resolution (**1.0x**), Sparse (**25%**)

Figure 1. *Sparse, high-resolution* features are far more informative than *dense, low-resolution* ones. Compared with direct downsampling, activation pruning can retain important details at a higher resolution, which is essential for most image recognition tasks.

The simplest solution to address this challenge is to downsample the image to a lower resolution. However, this will drop the fine details captured from the high-resolution sensor. What a waste! The missing information will bottleneck the model's performance upper bound, especially for small object detection and dense prediction tasks. For instance, the detection accuracy of a monocular 3D object detector will degrade by more than 5% in mAP by reducing the height and width by 1.6×[*]. Such a large gap cannot be easily recovered by scaling the model capacity up.

Dropping details uniformly at all positions is clearly suboptimal as not all pixels are equally informative (Figure 1a). Within an image, the pixels that contain detailed object features are more important than the background pixels. Moti-

[*]BEVDet [19] (with Swin Transformer [33] as backbone) achieves 31.2 mAP with 256×704 resolution and 25.90 mAP with 160×440 resolution.

vated by this, a very natural idea is to skip computations for less-important regions (*i.e.*, activation pruning). However, activation sparsity cannot be easily translated into the actual speedup on general-purpose hardware (*e.g.*, GPU) for CNNs. This is because sparse activation will introduce randomly distributed and unbalanced zeros during computing and cause computing unit under-utilization [54]. Even with dedicated system support [40], a high sparsity is typically required to realize speedup, which will hurt the model's accuracy.

Recently, 2D vision transformers (ViTs) have achieved tremendous progress. Among them, Swin Transformer [33] is a representative work that generalizes well across different visual perception tasks (such as image classification, object detection, and semantic segmentation). Our paper revisits the activation sparsity in the context of window-based ViTs. Different from convolutions, window attentions are naturally batched over windows, making real speedup possible with window-level activation pruning. We re-implement the other layers in the model (*i.e.*, FFNs and LNs) to also execute at the window level. As a result, we are able to achieve around 50% latency reduction with 60% window activation sparsity.

Within a neural network, different layers have different impacts on efficiency and accuracy, which advocates for a non-uniform layerwise sparsity configuration: *e.g.*, we may prune layers with larger computation and lower sensitivity more, while pruning layers with smaller computation and higher sensitivity less. To this end, we make use of the evolutionary search to explore the best per-layer pruning ratios under a resource constraint. We also propose *sparsity-aware adaptation* by randomly pruning a different subset of the activations at each iteration. This effectively adapts the model to activation sparsity and avoids the expensive re-training of every candidate within the large search space. Our SparseViT achieves speedups of **1.5**×, **1.4**×, and **1.3**× compared to its dense counterpart in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with negligible to no loss of accuracy.

## 2. Related Work

**Vision Transformers.** Transformers [46] have revolutionized natural language processing (NLP) and are now the backbone of many large language models (LLMs) [9]. Inspired by their success, researchers have explored the use of transformers in a range of visual recognition tasks [22]. ViT [24] was the first work in this direction, demonstrating that an image can be divided into 16×16 words and processed using multi-head self-attention. DeiT [45] improves on ViT's data efficiency. T2T-ViT [57], Pyramid ViT [50,51], and CrossFormer [52] introduce hierarchical modeling capability to ViTs. Later, Swin Transformer [31,33] applies self-attention to non-overlapping windows and proposes window shifting to enable cross-window feature communication. There have also been extensive studies on task-specific ViTs, such as ViTDet [26] for object detection, and SETR [59] and SegFormer [53] for semantic segmentation.

**Model Compression.** As the computational cost of neural networks continues to escalate, researchers are actively investigating techniques for model compression and acceleration [12,16]. One approach is to design more efficient neural network architectures, either manually [18,20,35,41,58] or using automated search [2,3,11,34,43,60]. These methods are able to achieve comparable performance to ResNet [15] with lower computational cost and latency. Another active direction is neural network pruning, which involves removing redundant weights at different levels of granularity, ranging from unstructured [12,13] to structured [17,32]. Although unstructured pruning can achieve higher compression ratios, the lower computational cost may not easily translate into measured speedup on general-purpose hardware and requires specialized hardware support. Low-bit weight and activation quantization is another approach that has been explored to reduce redundancy and speed up inference [15,21,48,49].

**Activation Pruning.** Activation pruning differs from static weight pruning as it is dynamic and input-dependent. While existing activation pruning methods typically focus on reducing memory cost during training [29,30,38], few of them aim to improve inference latency as activation sparsity does not always lead to speedup on hardware. To overcome this, researchers have explored adding system support for activation sparsity [40,42,55]. However, these libraries often require extensive engineering efforts and high sparsity rates to achieve measurable speedup over dense convolutions.

**Efficient ViTs.** Several recent works have explored different approaches to improve the efficiency of ViTs. For instance, MobileViT [36] combines CNN and ViT by replacing local processing in convolutions with global processing using transformers. MobileFormer [5] parallelizes MobileNet and Transformer with a two-way bridge for feature fusing, while NASViT [10] leverages neural architecture search to find efficient ViT architectures. Other works have focused on token pruning for ViTs [4,23,25,37,39,44,47,56]. However, these approaches mainly focus on token-level pruning, which is finer-grained than window pruning.

## 3. SparseViT

In this section, we first briefly revisit Swin Transformer and modify its implementation so that all layers are applied to windows. We then introduce how to incorporate the window activation sparsity into the model. Finally, we describe an efficient algorithm (based on sparsity-aware adaptation and evolutionary search) to find the layerwise sparsity ratio.

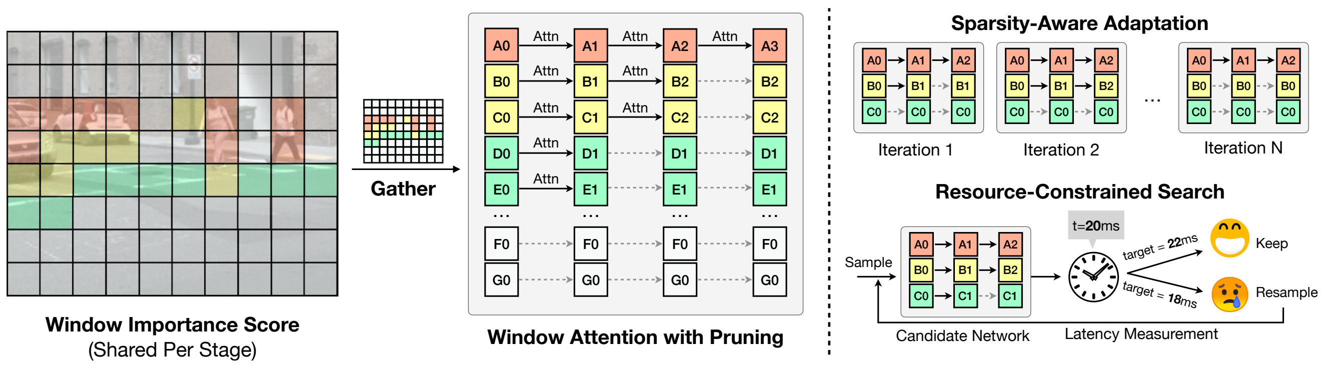


Figure 2. Overview of SparseViT. **(Left)** SparseViT first computes the L2 norm of each window activation as its importance score. After that, it first gathers the features from the windows with the highest importance scores, then runs self-attention on selected windows, and finally scatter the results back. **(Right upper)** SparseViT leverages sparsity-aware adaptation that samples a different layerwise activation sparsity at each training iteration to accommodate the activation sparsity. **(Right lower)** SparseViT makes use of evoluationary search to explore the best layerwise sparsity configuration given a latency constraint.

## 3.1. Swin Transformer Revisited

Swin Transformer [33] applies multi-head self-attention (MHSA) to extract local features within non-overlapping image windows (*e.g.*, 7×7). The transformer design follows a standard approach, involving layer normalization (LN), MHSA, and a feed-forward layer (FFN) applied to each window. Swin Transformer uses a shifted window approach that alternates between two different window partitioning configurations to introduce cross-window connections efficiently.

**Window-Wise Execution.** The original implementation of Swin Transformer applies MHSA at the window level, while FFN and LN are applied to the entire feature map. This mismatch between the two operations requires additional structuring before and after each MHSA, making window pruning more complicated as the sparsity mask must be mapped from the window level to the feature map. To simplify this process, we modify the execution of FFN and LN to be window-wise as well. This means that all operations will be applied at the window level, making the mapping of the sparsity mask very easy. In practice, this modification incurs a minimal accuracy drop of less than 0.1% due to padding, even without re-training. By making the execution of all operations window-wise, our method simplifies the pruning process.

## 3.2. Window Activation Pruning

Not all windows are equally important. In this paper, we define the importance of each window as its L2 activation magnitude. This is much simpler than other learning-based measures since it introduces smaller computational overhead and is fairly effective in practice.

Given an activation sparsity ratio (which will be detailed in the next section), we first gather those windows with the highest importance scores, then apply MHSA, FFN and LN only on these selected windows, and finally scatter outputs back. Figure 2 shows the workflow of window activation pruning.

To mitigate information loss due to coarse-grained window pruning, we simply duplicate the features of the unselected windows. This approach incurs no additional computation, yet proves highly effective in preserving information, which is critical for dense prediction tasks such as object detection and semantic segmentation.

**Shared Scoring.** Unlike conventional weight pruning, importance scores are input-dependent and need to be computed during inference, which can introduce significant overhead to the latency. To mitigate this, we compute the window importance score only once per stage and reuse it across all the blocks within the stage, amortizing the overhead. This also ensures that the window ordering remains consistent within a stage. We simplify the gathering operation using slicing, which does not require any feature copying.

## 3.3. Mixed-Sparsity Configuration Search

Using a uniform sparsity level throughout a model may not be the best strategy because different layers have varying impacts on both accuracy and efficiency. For example, early layers typically require more computation due to their larger feature map sizes, while later layers are more amenable to pruning as they are closer to the output. Thus, it is more beneficial to apply more pruning to layers with lower sensitivity and higher costs. However, manually exploring layerwise sparsity can be a time-consuming and error-prone task. To overcome this challenge, we propose a workflow that efficiently searches for the optimal mixed-sparsity pruning.

**Search Space.** We first design the search space for mixed-sparsity activation pruning. For each Swin block, we allow the sparsity ratio to be chosen from $\{0\%, 10\%, \ldots, 80\%\}$. Note that each Swin block contains two MHSAs, one with shifted window and one without. We will assign them with the same sparsity ratio. Also, we enforce the sparsity ratio

| Backbone | Resolution | Width | #MACs (G) | Latency (ms) | mAP$_\uparrow$ | mATE$_\downarrow$ | mASE$_\downarrow$ | mAOE$_\downarrow$ | mAVE$_\downarrow$ | mAAE$_\downarrow$ | NDS$_\uparrow$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Swin-T | 256×704 | 1× | 140.8 | 36.4 | 31.2 | 69.1 | 27.2 | 52.3 | 90.9 | 24.7 | 39.2 |
| **SparseViT** (Ours) | 288×792 | 1× | 113.9 | 34.5 | **32.0** | 72.8 | 27.2 | 53.8 | 79.4 | 25.7 | **40.1** |
| Swin-T (R224) | 224×616 | 1× | 78.5 | 23.0 | 29.9 | 71.8 | 27.4 | 60.9 | 79.0 | 26.0 | 38.4 |
| Swin-T (W0.6×) | 256×704 | 0.6× | 56.0 | 22.6 | 29.9 | 69.9 | 27.5 | 59.9 | 81.4 | 25.8 | 38.5 |
| **SparseViT** (Ours) | 256×704 | 1× | 78.4 | 23.8 | **31.2** | 70.9 | 27.5 | 58.7 | 83.1 | 27.2 | **38.9** |
| Swin-T (R192) | 192×528 | 1× | 67.1 | 18.7 | 28.7 | 74.3 | 27.9 | 59.5 | 76.7 | 27.8 | 37.7 |
| Swin-T (W0.4×) | 256×704 | 0.4× | 20.4 | 17.6 | 27.6 | 74.2 | 27.9 | 63.4 | 91.0 | 26.2 | 35.5 |
| **SparseViT** (Ours) | 256×704 | 1× | 58.6 | 18.7 | **30.0** | 72.0 | 27.5 | 59.7 | 81.7 | 26.6 | **38.3** |

Table 1. Results of monocular 3D object detection on nuScenes.

to be non-descending within each stage. This ensures that a pruned window will not engage in the computation again.

**Sparsity-Aware Adaptation.** To identify the best mixed-sparsity configuration for a model, it is crucial to evaluate its accuracy under different sparsity settings. However, directly assessing the original model's accuracy with sparsity might produce unreliable results (see Section 4.2). On the other hand, retraining the model with all possible sparsity configurations before evaluating its accuracy is impractical due to the significant time and computational costs involved. We therefore propose *sparsity-aware adaptation* as a more practical solution to address this challenge. Our approach involves adapting the original model, which was trained with only dense activations, by randomly sampling layerwise activation sparsity and updating the model accordingly at each iteration. After adaptation, we can obtain a more accurate estimate of the performance of different sparsity configurations without the need for full retraining. This enables us to efficiently and effectively evaluate different mixed-sparsity configurations and identify the optimal one for the model. Notably, our approach differs from super network training (used in NAS) as we only randomly sample activations, without changing the number of parameters.

**Resource-Constrained Search.** With an accurate estimate of the model's performance through sparsity-aware adaptation, we can proceed to search for the best sparsity configurations within specified resource constraints. In this paper, we consider two types of resource constraints: hardware-agnostic theoretical computational cost, represented by the number of multiply-accumulate operations (#MACs), and hardware-dependent measured latency. To perform the sparsity search, we adopt the evolutionary algorithm [11]. We first initialize the population with $n$ randomly sampled networks within the search space and using rejection sampling (*i.e.*, repeated resampling until satisfaction) to ensure every candidate meets the specified resource constraint. In each generation, we evaluate all individuals in the population and select the top $k$ candidates with the highest accuracy. We then generate the population for the next generation through $(n/2)$ mutations and $(n/2)$ crossovers using rejection sampling to satisfy the hard resource constraints. We repeat this process to obtain the best configuration from the population in the last generation.

**Finetuning with Optimal Sparsity.** The resulting model from our resource-constrained search has been trained under a variety of sparsity configurations during the adaptation stage. To further optimize its performance, we finetune the model with the fixed sparsity configurations identified in the search process until convergence.

## 4. Experiments

In this section, we evaluate our method on three diverse tasks, including monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation.

**Latency Measurement.** We report the latency of the backbone in all our results as our method is only applied to the backbone. The latency measurements are obtained using a single NVIDIA RTX A6000 GPU. To ensure accurate measurements, we perform 500 inference steps as a warm-up and subsequently measure the latency for another 500 steps. To minimize the variance, we report the average latency of the middle 250 measurements out of the 500.

### 4.1. Main Results

#### 4.1.1 Monocular 3D Object Detection

**Dataset and Metrics.** We use nuScenes [1] as the benchmark dataset for monocular 3D object detection, which includes 1k scenes with multi-modal inputs from six surrounding cameras, one LiDAR, and five radars. We only employ camera inputs in our experiments. We report official metrics, including mean average precision (mAP), average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE), and average attribute error (AAE). We also report the nuScenes detection score (NDS), which is a weighted average of the six metrics.

| Backbone | Resolution | Width | #MACs (G) | Latency (ms) | $AP^{bbox}$ | $AP^{bbox}_{50}$ | $AP^{bbox}_{75}$ | $AP^{mask}$ | $AP^{mask}_{50}$ | $AP^{mask}_{75}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Swin-T | 640×640 | 1× | 161.8 | 46.6 | 42.0 | 63.3 | 45.7 | 38.3 | 60.3 | 40.9 |
| Swin-T (R576) | 576×576 | 1× | 149.5 | 41.3 | 41.0 | 62.1 | 44.9 | 37.2 | 59.0 | 39.6 |
| Swin-T (W0.9×) | 640×640 | 0.9× | 122.3 | 41.8 | 40.4 | 61.9 | 43.8 | 37.1 | 58.9 | 39.8 |
| **SparseViT** (Ours) | 672×672 | 1× | 139.5 | 41.3 | **42.4** | 63.3 | 46.4 | **38.5** | 60.3 | 41.3 |
| Swin-T (R544) | 544×544 | 1× | 119.8 | 34.8 | 40.5 | 61.2 | 43.8 | 36.8 | 58.2 | 39.1 |
| Swin-T (W0.8×) | 640×640 | 0.8× | 90.5 | 35.9 | 39.4 | 60.7 | 42.8 | 36.4 | 57.9 | 38.8 |
| **SparseViT** (Ours) | 672×672 | 1× | 116.5 | 34.1 | **41.6** | 62.5 | 45.5 | **37.7** | 59.4 | 40.2 |
| Swin-T (R512) | 512×512 | 1× | 117.5 | 32.9 | 39.6 | 60.1 | 43.4 | 36.0 | 57.0 | 38.2 |
| Swin-T (W0.6×) | 640×640 | 0.6× | 63.4 | 31.7 | 38.7 | 60.2 | 41.6 | 35.7 | 57.0 | 38.0 |
| **SparseViT** (Ours) | 672×672 | 1× | 105.9 | 32.9 | **41.3** | 62.2 | 44.9 | **37.4** | 59.1 | 39.7 |

Table 2. Results of 2D instance segmentation on COCO.

**Model and Baselines.** We use BEVDet [19] as the base model for monocular 3D object detection. It adopts Swin-T [33] as the baseline and employs FPN [27] to fuse information from multi-scale features. Following BEVDet [19], we resize the input images to 256×704 and train the model for 20 epochs. We compare our SparseViT against two common model compression strategies: reducing resolution and width. For reduced resolution, we re-train the model with different resolutions. For reduced width, we uniformly scale down the model to 0.4× and 0.6×, then pre-train it on ImageNet [8] and finally finetune it on nuScenes [1].

**Compared with Reduced Resolution.** The accuracy of monocular 3D object detection is highly influenced by resolution scaling. With fine-grained features in higher-resolution images, our SparseViT outperforms the baseline with smaller resolutions, with comparable or faster latency. The results in Table 1 show that SparseViT achieves the same accuracy as Swin-T with **1.8**× lower #MACs and **1.5**× faster inference latency. Furthermore, when compared to the baseline with 192×528 resolution, SparseViT achieves 30.0 mAP and 38.3 NDS at 50% latency budget of the full Swin-T backbone, which is **1.3** mAP and 0.6 NDS better, respectively.

**Comparison with Reduced Width.** Reducing the model's width considerably lowers #MACs. However, this decrease in computation cost might not necessarily translate into a measured speedup due to low device utilization. SparseViT outperforms the baseline with 0.6× width by 1.3 mAP and the one with 0.4× width by 2.4 mAP at similar latency. This indicates that activation pruning is more effective than model pruning in latency-oriented compression.

#### 4.1.2 2D Instance Segmentation

**Dataset and Metrics.** We use COCO [28] as our benchmark dataset for 2D instance segmentation, which contains 118k/5k training/validation images. We report the box/mask average precision (AP) over 50% to 95% IoU thresholds.

**Model and Baselines.** We use Mask R-CNN [14] as the base model. The model uses Swin-T [33] as its backbone. We adopt 640×640 as the default input resolution and train the model for 36 epochs. We compare our SparseViT against baselines with reduced resolutions and widths. For reduced resolution, we train the model using random scaling augmentation [14, 33] and evaluate the model under different resolutions. For reduced width, we uniformly scale down the model to 0.6×, 0.8× and 0.9×, pre-train it on ImageNet [8] and finetune it on COCO [28].

**Comparison with Reduced Resolution.** As in Table 2, SparseViT consistently outperforms the baseline with less computation across various input resolutions from 512×512 to 640×640. Our key insight is that starting with a high resolution of 672×672 and aggressively pruning the activation is more efficient than directly scaling down the input resolution. This observation aligns with the visualizations in Figure 1, where fine-grained details become indistinguishable under low resolution. Despite using a higher resolution, SparseViT achieves 1.2× smaller #MACs than the baseline while delivering 0.4% higher $AP^{bbox}$ and 0.2% higher $AP^{mask}$. With similar accuracy, SparseViT has **1.4**× lower #MACs, resulting in a perfect **1.4**× speedup. This is because our SparseViT performs *window-level* activation pruning, which is equivalent to reducing the batch size in `MHSA` computation and is easy to accelerate on hardware. Similarly, to match the accuracy of the baseline with 90% resolution, SparseViT is **1.3**× faster and consumes **1.4**× less computation. Remarkably, despite using 30% larger resolution (*i.e.*, 1.7× larger #MACs to begin with!), SparseViT is more efficient than the baseline at 512×512 resolution, while providing significantly better accuracy (**+1.7** $AP^{bbox}$ and **+1.4** $AP^{mask}$).

**Comparison with Reduced Width.** In Table 2, we also compare SparseViT with the baseline with reduced channel width. Although reducing channel width leads to a significant reduction in #MACs, we do not observe a proportional increase in speed. For example, the baseline with 0.6× chan-

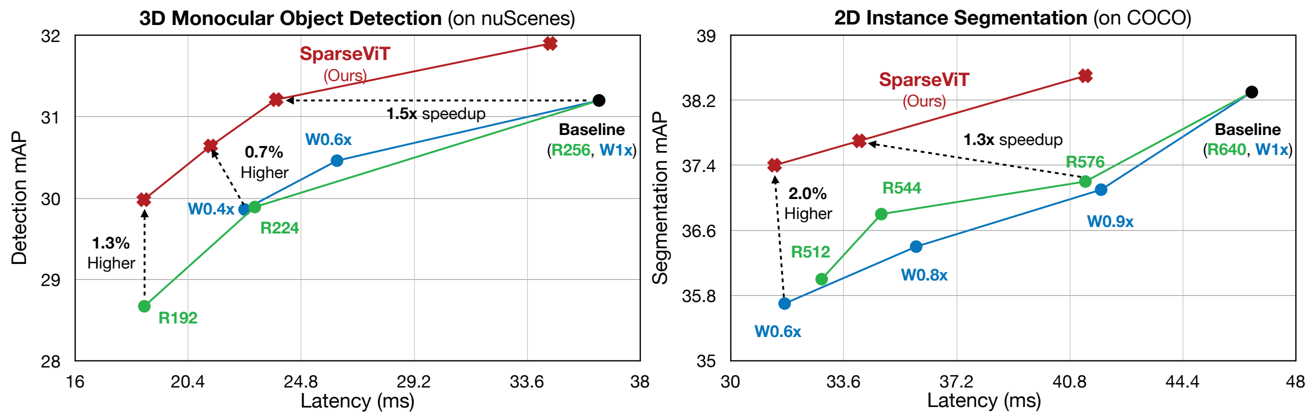


Figure 3. SparseViT delivers a significantly better accuracy-efficiency trade-off than the baselines with reduced resolutions and widths on monocular 3D object detection (**left**) and 2D instance segmentation (**right**).

| Backbone | Resolution | Latency (ms) | mIoU |
|---|---|---|---|
| Swin-L | 1024×2048 | 329.5 | 83.3 |
| Swin-L (R896) | 896×1792 | 256.5 | 82.8 |
| **SparseViT** (Ours) | 1024×2048 | 250.6 | **83.2** |

Table 3. Results of 2D semantic segmentation on Cityscapes.

nel width on 640×640 inputs consumes only 63.4G MACs, yet it runs slightly slower than SparseViT on 672×672 inputs with 105.9G MACs (which is actually 1.7× heavier!). GPUs prefer wide and shallow models to fully utilize computation resources. Pruning channels will decrease device utilization and is not as effective as reducing the number of windows (which is equivalent to directly reducing the batch size in `MHSA` computation) for model acceleration.

### 4.1.3 2D Semantic Segmentation

**Dataset and Metrics.** Our benchmark dataset for 2D semantic segmentation is Cityscapes [7], which consists of over 5k high-quality, fully annotated images with pixel-level semantic labels for 30 object classes, including road, building, car, and pedestrian. We report mean intersection-over-union (mIoU) as the primary evaluation metric on this dataset.

**Model and Baselines.** We use Mask2Former [6] as our base model, which uses Swin-L [33] as its backbone. We train the model for 90k iterations with an input resolution of 1024×2048. Here, we only compare to the baseline with reduced resolution.

**Results.** Based on Table 3, SparseViT model attains comparable segmentation accuracy to the baseline while delivering a speedup of 1.3×. In contrast, reducing the resolution results in a more substantial decrease in accuracy. By utilizing spatial redundancy, SparseViT delivers competitive results while being more efficient than direct downsampling.

## 4.2. Analysis

In this section, we present analyses to validate the effectiveness of our design choices. All studies are conducted on monocular 3D object detection (on nuScenes), except the evolutionary search one in Figure 4(c), which is conducted on 2D instance segmentation (on COCO).

**Window pruning is more effective than token pruning.** Table 4 demonstrates that SparseViT achieves a low computational cost and latency without any loss of accuracy, whereas DynamicViT [39], a learnable token pruning method, experiences a substantial decrease in accuracy of 0.4 mAP with only a minor reduction in computational cost. These findings offer valuable insights into the comparative performance of these pruning methods. Furthermore, it is worth noting that token pruning requires more fine-grained *token-level* gathering, which has inferior memory access locality and tends to be slower on GPUs, unlike window pruning in our SparseViT that only necessitates *window-level* gathering.

| | #MACs (G) | Latency (ms) | mAP Drop |
|---|---|---|---|
| DynamicViT | 98.6 | 33.3 | -0.1 |
| DynamicViT | 91.7 | 32.6 | -0.4 |
| SparseViT | **78.4** | **23.8** | **0.0** |

Table 4. Window pruning (SparseViT) is more efficient and effective than learnable token pruning (DynamicViT).

**Pruning from higher resolution matters.** A key design insight in SparseViT is that it is more advantageous to start with a *high-resolution* input and *prune more*, rather than beginning with a *low-resolution* input and *pruning less*. While counterintuitive, starting with a high-resolution input allows us to retain fine-grained information in the image. The abundance of uninformative background windows provides us with ample room for activation pruning. Quantitatively, as in

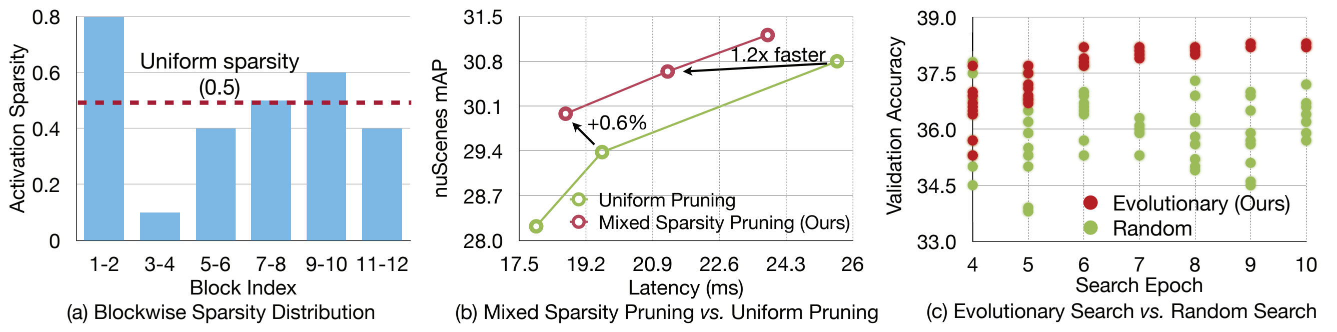


Figure 4. Non-uniform sparsity is better than uniform sparsity (a, b). Evolutionary search is more sample-efficient than random search (c).

Table 5, starting from the highest resolution (*i.e.*, 256×704) produces the best accuracy under the same latency constraint.

| Input Resolution | #MACs (G) | Latency (ms) | mAP |
|---|---|---|---|
| 192×528 | 67.1 | **18.7** | 28.7 |
| 224×616 | 64.9 | 19.1 | 29.7 |
| 256×704 | **58.6** | **18.7** | **30.0** |

Table 5. Starting from a high-resolution input and pruning more is better than starting from a low-resolution input and pruning less.

**Mixed-sparsity pruning is better than uniform pruning.** In Figure 4(a), we show the pruning strategy used by SparseViT to achieve 50% overall sparsity. Unlike uniform sparsity ratios applied to all layers, SparseViT favors non-uniform sparsity ratios for different layers based on their proximity to the input. Specifically, the smaller window sizes in the first and second blocks allow for more aggressive pruning, while larger window sizes in later layers result in less aggressive pruning. This non-uniform sparsity selection leads to better accuracy, as in Figure 4(b). Compared to uniform pruning, SparseViT achieves similar accuracy but is up to 1.2× faster. Alternatively, when compared at similar speeds, SparseViT achieves 0.6% higher accuracy than uniform pruning.

**Evolutionary search is better than random search.** We demonstrate the efficacy of evolutionary search in selecting the sparsity ratios. Figure 4(c) compares the results obtained by evolutionary search and random search, by visualizing the validation mAP of the best-performing models found in the last seven epochs. The accuracy of the models discovered by evolutionary search converges to a high value of 37.5 after the sixth epoch, whereas the models found by random search still exhibit high variance until the final epoch.

**Sparsity-aware adaptation offers a better accuracy proxy.** Conventional pruning methods [12, 13] typically scan different sparsity ratios at each layer, zero out corresponding weights, and evaluate accuracy directly on a holdout validation set. While this approach works well for weight pruning in CNNs, it falls short when generalizing to activation pruning in ViTs. For example, as in Table 6, the naive accuracy sensitivity analysis approach asserts that pruning away 50% of the windows in stage 2 is much better than doing so in stage 1 (*i.e.*, 30.93 mAP *vs.* 28.31 mAP). However, if both models are finetuned, the accuracy difference is almost zero. In contrast, sparsity-aware adaptation provides a much better accuracy proxy (*i.e.*, 30.83 mAP *vs.* 30.63 mAP), which can serve as a better feedback signal for evolutionary search.

| SAA | Sparsity Ratios | mAP w/o FT | mAP w/ FT |
|---|---|---|---|
| | (0.5, 0, 0, 0) | 28.31 | 31.40 |
| | (0, 0.5, 0, 0) | 30.93 | 31.37 |
| ✓ | (0.5, 0, 0, 0) | 30.83 | 31.45 |
| ✓ | (0, 0.5, 0, 0) | 30.63 | 31.44 |

Table 6. Sparsity-aware adaptation (SAA) improves the correlation between the accuracy before and after finetuning (FT).

**Sparsity-aware adaptation improves the accuracy.** In Table 7, we explore the impact of sparsity-aware adaptation on model convergence during the finetuning stage. Using the same window sparsity ratios, we compare three approaches: **(a)** training from scratch, **(b)** finetuning from a pre-trained Swin-T, and **(c)** finetuning from a pre-trained Swin-T with sparsity-aware adaptation (SAA). Our results indicate that approach **(c)** achieves the best performance. We speculate that sparsity-aware adaptation could serve as a form of implicit distillation during training. Models with higher window sparsity can benefit from being co-trained with models with lower window sparsity (and therefore higher #MACs and higher capacity), resulting in improved overall performance.

| | mAP |
|---|---|
| **(a)** Training from scratch | 29.67 |
| **(b)** Finetuning from a pre-trained Swin-T | 30.83 |
| **(c)** Finetuning from a pre-trained Swin-T with SAA | **31.21** |

Table 7. Sparsity-aware adaptation improves the convergence.

**SparseViT learns to keep important foreground windows.** In Figure 5, we visualize the window pruning strategy dis-

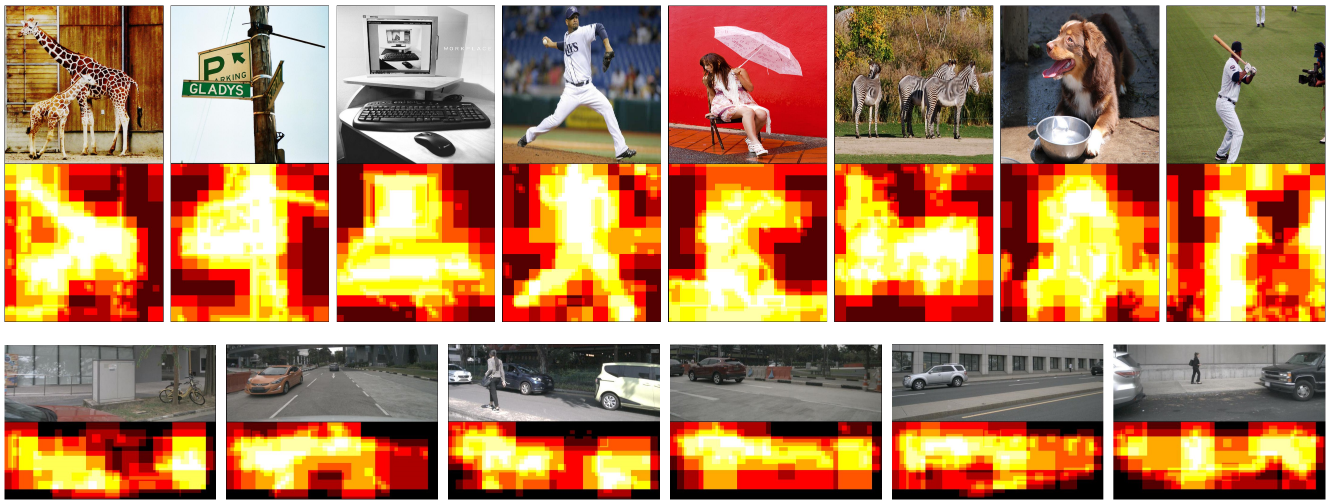


Figure 5. SparseViT effectively prunes irrelevant background windows while retaining informative foreground windows. Each window's color corresponds to the number of layers it is executed. Brighter colors indicate that the model has executed the window in more layers.

covered by SparseViT, where the color represents the number of layers each window is executed. Notably, on the first row, SparseViT automatically learns the contour of the objects, as demonstrated in the third and fourth figures, where the computer and sportsman are respectively outlined. Furthermore, on the second row, the windows corresponding to foreground objects are not pruned away. Despite being a small object, the pedestrian in the last figure is retained throughout the entire execution, illustrating the effectiveness of SparseViT.

**L2 magnitude-based scoring is simple and effective.** Table 8 demonstrates that the L2 magnitude-based scoring is simple and effective, outperforming the learnable window scoring that utilizes MLP and Gumbel-Softmax for predicting window scores. We also include the regularization loss on pruning ratios, following Rao *et al*. [39], to restrict the proportion of preserved windows to a predetermined value in the learnable window scoring baseline. However, the added complexity of the learnable scoring results in higher latency and #MACs compared to the L2 magnitude-based scoring. Achieving an optimal balance between pruning regularization loss and detection loss is not an easy task, as evidenced by a 0.4 mAP drop observed in the learnable scoring method.

| Scoring | #MACs (G) | Latency (ms) | mAP Drop |
|---|---|---|---|
| Learnable | 89.6 | 33.1 | -0.6 |
| L2 Magnitude | **78.4** | **23.8** | **0.0** |

Table 8. L2 magnitude-based scoring is simple and effective, achieving a better accuracy-efficiency trade-off than learnable scoring.

**Shared scoring is better than independent scoring.** We compare our proposed shared scoring strategy with the independent scoring. Despite being an approximation, sharing window scores per stage, as in Table 9, does not negatively impact the performance. This strategy amortizes the cost of score calculation, which allows for more effective computation and results in a better accuracy-latency trade-off.

| Scoring | Sparsity | #MACs (G) | Latency (ms) | mAP |
|---|---|---|---|---|
| Independent | 0.5 | 70.68 | 23.3 | 30.08 |
| Shared | 0.5 | 70.68 | **23.0** | **30.17** |

Table 9. Shared scoring per stage reduces the cost of score calculation, leaving room for effective computation and offering a better accuracy-latency trade-off than independent scoring per block.

## 5. Conclusion

Although activation pruning is a very powerful technique for preserving high-resolution information, it does not offer actual speedup for CNNs. In this paper, we revisit activation sparsity for recent window-based ViTs and propose a novel approach to leverage it. We introduce sparsity-aware adaptation and employ evolutionary search to efficiently find the optimal layerwise sparsity configuration. As a result, SparseViT achieves 1.5×, 1.4×, and 1.3× measured speedups in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with minimal to no loss in accuracy. We hope that our work inspires future research to explore the use of activation pruning for achieving better efficiency while retaining high-resolution information.

**Acknowledgement.** This work was supported by National Science Foundation (NSF), MIT-IBM Watson AI Lab, MIT AI Hardware Program, Amazon-MIT Science Hub, NVIDIA Academic Partnership Award, and Hyundai. Zhijian Liu was partially supported by Qualcomm Innovation Fellowship.

## References

[1] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *CVPR*, 2020. 4, 5

[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 *ICLR*, 2020. 2

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*, 2019. 2

[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*, 2021. 2

[5] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging Mobilenet and Transformer. In *CVPR*, 2022. 2

[6] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-Attention Mask Transformer for Universal Image Segmentation. In *CVPR*, 2022. 6

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *CVPR*, 2016. 6

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 5

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019. 2

[10] Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, qiang liu, and Vikas Chandra. NASViT: Neural Architecture Search for Efficient Vision Transformers with Gradient Conflict aware Supernet Training. In *ICLR*, 2022. 2

[11] 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*, 2020. 2, 4

[12] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016. 2, 7

[13] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. In *NeurIPS*, 2015. 2, 7

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 5

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[16] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *ECCV*, 2018. 2

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *ICCV*, 2017. 2

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dimitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv*, 2017. 2

[19] Junjie Huang, Guan Huang, Zheng Zhu, and Dalong Du. BEVDet: High-performance Multi-camera 3D Object Detection in Bird-Eye-View. *arXiv*, 2021. 1, 5

[20] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and $<$ 0.5MB Model Size. *arXiv*, 2016. 2

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*, 2018. 2

[22] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Comput. Surv.*, 2022. 2

[23] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned Token Pruning for Transformers. In *KDD*, 2022. 2

[24] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. 2

[25] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Bin Ren, Minghai Qin, Hao Tang, and Yanzhi Wang. SPViT: Enabling Faster Vision Transformers via Soft Token Pruning. In *ECCV*, 2022. 2

[26] Yanghao Li, Hanzi Mao, Girshick, and Kaiming He. Exploring Plain Vision Transformer Backbones for Object Detection. In *ECCV*, 2022. 2

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *ICCV*, 2017. 5

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014. 5

[29] Jianhui Liu, Yukang Chen, Xiaoqing Ye, Zhuotao Tian, Xiao Tian, and Xiaojuan Qi. Spatial Pruned Sparse Convolution for Efficient 3D Object Detection. In *NeurIPS*, 2022. 2

[30] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. Dynamic Sparse Graph for Efficient Deep Learning. In *ICLR*, 2019. 2

[31] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin Transformer V2: Scaling Up Capacity and Resolution. In *CVPR*, 2022. 2

[32] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV*, 2017. 2

[33] 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 *ICCV*, 2021. 1, 2, 3, 5, 6

[34] Zhijian Liu, Haotian Tang, Shengyu Zhao, Kevin Shao, and Song Han. PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution. *TPAMI*, 2021. 2

[35] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*, 2018. 2

[36] Sachin Mehta and Mohammad Rastegari. MobileViT: Light-Weight, General-Purpose, and Mobile-Friendly Vision Transformer. In *ICLR*, 2022. 2

[37] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. IA-RED$^2$: Interpretability-Aware Redundancy Reduction for Vision Transformers. In *NeurIPS*, 2021. 2

[38] Md Aamir Raihan and Tor Aamodt. Sparse Weight Activation Training. In *NeurIPS*, 2020. 2

[39] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification. In *NeurIPS*, 2021. 2, 6, 8

[40] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. SBNet: Sparse Blocks Network for Fast Inference. In *CVPR*, 2018. 2

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 2

[42] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. TorchSparse: Efficient Point Cloud Inference Engine. In *MLSys*, 2022. 2

[43] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In *ECCV*, 2020. 2

[44] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch Slimming for Efficient Vision Transformers. In *CVPR*, 2022. 2

[45] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training Data-Efficient Image Transformers & Distillation through Attention. In *ICML*, 2021. 2

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017. 2

[47] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *HPCA*, 2021. 2

[48] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *CVPR*, 2019. 2

[49] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Hardware-Centric AutoML for Mixed-Precision Quantization. *IJCV*, 2020. 2

[50] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. In *ICCV*, 2021. 2

[51] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. PVTv2: Improved Baselines with Pyramid Vision Transformer. *CVM*, 2022. 2

[52] Wenxiao Wang, Lu Yao, Long Chen, Binbin Lin, Deng Cai, Xiaofei He, and Wei Liu. CrossFormer: A Versatile Vision Transformer Hinging on Cross-scale Attention. In *ICLR*, 2022. 2

[53] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In *NeurIPS*, 2021. 2

[54] Zirui Xu, Fuxun Yu, Chenxi Liu, Zhe Wu, Hongcheng Wang, and Xiang Chen. FalCon: Fine-grained Feature Map Sparsity Computing with Decomposed Convolutions for Inference Optimization. In *WACV*, 2022. 2

[55] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 2018. 2

[56] Hongxu Yin, Arash Vahdat, Jose Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. AdaViT: Adaptive Tokens for Efficient Vision Transformer. In *CVPR*, 2021. 2

[57] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-Token ViT: Training Vision Transformers From Scratch on ImageNet. In *ICCV*, 2021. 2

[58] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*, 2018. 2

[59] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers. In *CVPR*, 2021. 2

[60] Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *ICLR*, 2017. 2