Boost Vision Transformer with GPU-Friendly Sparsity and Quantization

Chong Yu[1,2] Tao Chen[3,*] Zhongxue Gan[1,*] Jiayuan Fan[1]
[1]Academy for Engineering and Technology, Fudan University [2]NVIDIA Corporation
[3]School for Information Science and Technology, Fudan University
21110860050@m.fudan.edu.cn; {eetchen, ganzhongxue, jyfan}@fudan.edu.cn *

Abstract

*The transformer extends its success from the language to the vision domain. Because of the stacked self-attention and cross-attention blocks, the acceleration deployment of vision transformer on GPU hardware is challenging and also rarely studied. This paper thoroughly designs a compression scheme to maximally utilize the **GPU-friendly 2:4 fine-grained structured sparsity and quantization**. Specially, an original large model with dense weight parameters is first pruned into a sparse one by 2:4 structured pruning, which considers the GPU's acceleration of 2:4 structured sparse pattern with FP16 data type, then the floating-point sparse model is further quantized into a fixed-point one by sparse-distillation-aware quantization aware training, which considers GPU can provide an extra speedup of 2:4 sparse calculation with integer tensors. A mixed-strategy knowledge distillation is used during the pruning and quantization process. The proposed compression scheme is flexible to support supervised and unsupervised learning styles. Experiment results show **GPUSQ-ViT** scheme achieves state-of-the-art compression by reducing vision transformer models **6.4-12.7**× on model size and **30.3-62**× on FLOPs with negligible accuracy degradation on ImageNet classification, COCO detection and ADE20K segmentation benchmarking tasks. Moreover, **GPUSQ-ViT** can boost actual deployment performance by **1.39-1.79**× and **3.22-3.43**× of latency and throughput on A100 GPU, and **1.57-1.69**× and **2.11-2.51**× improvement of latency and throughput on AGX Orin.*

1. Introduction

Transformer-based neural models [48] have garnered immense interest recently due to their effectiveness and generalization across various applications. Equipped with the attention mechanism [52] as the core of its architecture, transformer-based models specialize in handling long-range dependencies, which are also good at extracting non-local features [9] [5] in the computer vision domain. With comparable and even superior accuracy than the traditional convolution neural networks (CNN) [12] [49], more vision transformer models are invented and gradually replace the CNN with state-of-the-art performance on image classification [27] [26], object detection [70] [59], and segmentation [58] [68] tasks. Due to the vision transformer models having a generally weaker local visual inductive bias [9] inherent in CNN counterparts, many transformer blocks are stacked for compensation. Moreover, the attention module in the transformer block contains several matrix-to-matrix calculations between key, query, and value parts [52]. Such designs give the naive vision transformers more parameters and higher memory and computational resource requirements, causing high latency and energy consuming during the inference stage. *It is challenging for actual acceleration deployment in GPU hardware*.

Model compression techniques to transfer the large-scale vision transformer models to a lightweight version can bring benefits to more efficient computation with less on-device memory and energy consumption. There are some previous studies to inherit CNN compression methods, including pruning [43] [15], quantization [28] [23], distillation [61], and architecture search [6] on vision transformers. However, there are some drawbacks in previous studies:

- Most of these common methods aim to reduce the theoretical model size and Floating Point Operations (FLOPs). But it has been proved [33] [37] that smaller model sizes and FLOPs are not directly proportional to better efficiency on deployed hardware.
- The compression patterns do not match hardware characteristics. For example, pruned [43] or searched [6] vision transformer models have the unstructured sparse pattern in weight parameters, i.e., the distribution of non-zero elements is random. So deployed hardware can not provide actual speedup due to lacking the characteristics support for unstructured sparsity [35].
- How to keep the accuracy to the best with multiple compression methods and how to generalize on multiple vision tasks lack systematical investigation.

[*] Tao Chen and Zhongxue Gan are corresponding authors.

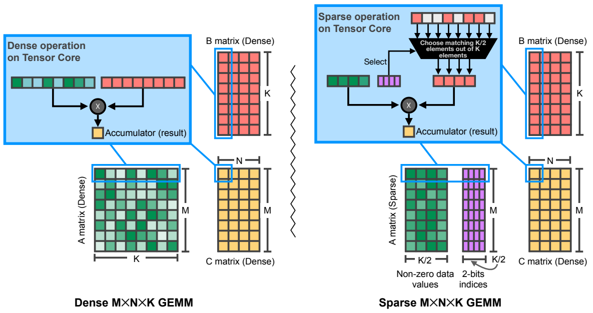

Figure 1. Comparison of computing a $M \times N \times K$ GEMM onto a Tensor Core. Dense matrix A of size $M \times K$ in ***left side*** becomes $M \times \frac{K}{2}$ in ***right side*** after compressing with ***2:4 fine-grained structured sparse pattern***. Sparse Tensor Core automatically picks only the elements from B according to the nonzero elements in A. Comparing the dense and sparse GEMMs, B and C are the same dense $K \times N$ and $M \times N$ matrices, respectively. By skipping the unnecessary multiplications of redundant zeros, sparse GEMM accelerate the dense GEMM with $2\times$.

General Matrix Multiplication (GEMM) is the fundamental implementation inside the common parts of vision transformers, such as convolution, linear projection, and transformer blocks. A specific acceleration unit called Tensor Core [39] is firstly introduced in NVIDIA Volta GPU [34] to accelerate these GEMM instructions and further enhanced to support sparse GEMM in Ampere GPU [35]. To make the GPU hardware efficient for sparse GEMM, a constraint named ***2:4 fine-grained structured sparsity*** [33] is imposed on the allowed sparsity pattern, i.e., two values from every four contiguous elements on rows must be zero. Due to the 2:4 sparsity support on GPU Tensor Core hardware, sparse GEMM can reduce memory storage and bandwidth by almost 2× and provide 2× math throughput compared to dense GEMM by skipping the redundant zero-value computation, as shown in Figure 1. Ampere GPU supports various numeric precision for 2:4 sparsity, including FP32, FP16, INT8, and INT4, etc.

Inspired by GPU's acceleration characteristic for 2:4 fine-grained structured sparse pattern with various low-precision operators, we thoroughly design the compression scheme **GPUSQ-ViT** by utilizing the **GPU**-friendly **S**parsity and **Q**uantization to boost deployment efficacy for **Vi**sion **T**ransformer models, especially on GPU platforms. **GPUSQ-ViT** contains two main workflows. Firstly, 2:4 sparse pruning with knowledge distillation [14] (KD) is proposed to compress the specific structures in vision transformer architecture, e.g., transformer block, patch embedding, to be GPU-friendly. Secondly, we further quantize the sparse model through sparse-distillation-aware Quantization Aware Training [30] (QAT). To measure the influence of quantization errors, we use the feature-based distillation loss in the sparse pruning workflow as the weight factor. The feature-based KD utilizes the scale factor in the quantization compression workflow, which can best compensate for the final compressed model's accuracy. We demonstrate that **GPUSQ-ViT** can generally apply to vision transformer models and benchmarking tasks, with state-of-the-art theoretical metrics on model size and FLOPs. Moreover, as **GPUSQ-ViT** compresses with GPU-friendly patterns, the compressed models can achieve state-of-the-art deployment efficacy on GPU platforms. Our main contributions include:

- Unlike previous compression methods only aiming at reducing theoretical metrics, we propose **GPUSQ-ViT** from the perspective of GPU-friendly 2:4 sparse pattern with low-precision quantization for the first time, achieving GPU acceleration of 4 times than prior arts.
- **GPUSQ-ViT** combines feature-based KD with sparse pruning and QAT, which can best compensate for sparse and quantized models' accuracy.
- **GPUSQ-ViT** can apply to various vision transformer models and benchmarking tasks, with proven state-of-the-art efficacy on model size, FLOPs, and actual deployment performance on multiple GPUs. Moreover, **GPUSQ-ViT** can work without ground truth label annotations in an unsupervised learning style.

2. Related work

2.1. Sparsity in model compression

Sparsity is a typical pattern [10] in the deep learning paradigm, which can help to save the computational power as well as reduce the memory bandwidth and storage burden [33]. Sparsity has different granularities [29], e.g., we can generate the filter-level, kernel-level, vector-level, and element-level sparsity [29] in a weight tensor from coarse to fine granularity. The coarse-grained sparsity has a regular sparse pattern which can facilitate acceleration with algebra libraries [33]. The fine-grained sparsity leads to a more irregular sparse pattern which is not friendly for acceleration, but it can achieve a higher sparse ratio without harming model accuracy [60] [63]. Many previous efforts [4] [63] [20] have explored the sparse granularity to balance accuracy influence with real performance benefits.

Several efforts explored to compress the vision transformers with sparsity. Inspired by the phenomenon that the vision transformers take effect only according to a subset of most informative tokens [43], we can generate the sparse tokens by pruning the less informative ones. The redundant tokens are pruned based on the inputs, spatial attention mechanism [44], or multi-head interpreter [40] in a dynamical [43] or patch-slimming manner [50].

Other efforts are explored on how to prune the components inside the basic structure in vision transformers, i.e., the multi-head attention block (MHA) [52]. For example, a successful trial [69] is first to learn the importance of each component in MHA by training with sparse regularization, then pruning the less important ones to obtain the sparse

MHA. Other strategies aim to sparsify the attention heads and reduce the sequence length in an MHA structure based on specific numerical metrics [54] or searched optimal policy [15]. A more aggressive approach is pruning the entire MHA blocks to generate a sparse Mixture-of-Experts [16] vision transformer or an extremely compact version [66]. *Most of the prior arts use model sizes and FLOPs as compression targets without considering the characteristics of deployed hardware.* We find low efficiency when deploying these compressed models on GPUs, which inspires us to ***design the compression scheme with a GPU-friendly sparse pattern***. Based on prior arts, weight multiplexing [66] or knowledge distillation [64] [61] are effective to compensate for the accuracy loss.

2.2. Quantization in model compression

Quantization is another orthogonal technique in the model compression area. It refers to the technique [56] of applying alternative formats other than the standard 32-bit single-precision floating-point (FP32) data type for weight parameters, inputs, and activations when executing a neural model. Quantization can significantly speed up the model inference performance because the low-precision formats have higher computational throughput support in many processors [35] [17] [2]. Meanwhile, low-precision representation helps to reduce the memory bandwidth pressure and can save much memory-system operation time with the cache utilization improvement.

Post Training Quantization (PTQ) [18] and Quantization Aware Training (QAT) [30] are two main strategies in quantization. PTQ directly calibrates on limited sample inputs [31] to find the optimal clipping threshold and the scale factor to minimize the quantization noise [3]. PTQ is preferred [47] when without access to the whole training dataset [21]. However, it is a non-trivial effort [28] [65] [25] [23] to ensure the PTQ quantized vision transformer model without an apparent accuracy decrease. And the accuracy degradation is more serious when going below 8 bits formats [47]. QAT inserts the quantization and de-quantization nodes [37] into the float-point model structure, then undergo the fine-tuning process to learn the scale factor adjustment with minimal influence on accuracy [30]. Considering some activation structures like GeLU [13] and Swish [42] are more sensitive [23] than ReLU [1], some efforts are made to design the specific QAT [23] [22] for the vision transformers. Moreover, QAT can provide more quantization robustness for lower-bit formats [23].

Previous efforts to design the PTQ and QAT approaches for vision transformer mainly focused on the accuracy improvement. ***Due to the lack of hardware characters and acceleration library support***, some quantized models using 6 bits [28] or float-point learnable bit-width like 3.7 bits [23] to represent weights and activations ***cannot get the expected speedup on general acceleration hardware***, like GPU [34] [35] and TPU [45]. Moreover, *supporting the specific bit-width quantization*, like 6 bits, *is a non-trivial effort*. End-users need to program the FPGA hardware [22] and develop specific bit-width libraries like Basic Linear Algebra Subprograms (BLAS) [19], which is a heavy burden for actual deployment.

3. Boost vision transformer on GPU

GPUSQ-ViT mainly contains **2:4 structured sparse pruning** and **sparse-distillation-aware QAT** workflows. We further explain the 2:4 sparse pattern in section 3.1, and how to compress each part of a vision transformer model according to the 2:4 sparse pattern in sections 3.2 and 3.3. Section 3.4 describes the **GPUSQ-ViT** design as a whole.

3.1. Fine-grained structured sparsity on GPU

As shown in Figure 1, the sparse GEMM performs the *sparse matrix* × *dense matrix* = *dense matrix* operation by skipping the redundant zero-value computation with sparse Tensor Core acceleration. For example, matrix A of size $M \times K$ follows the ***2:4 fine-grained structured sparse pattern***, and the dense matrix B is of size $K \times N$. If we use the dense GEMM to calculate between matrices A and B, the zero values in A would not be skipped during computation. The entire $M \times N \times K$ dense GEMM will calculate the result matrix C with $M \times N$ size in *T* GPU cycles. If we use the sparse GEMM, only the non-zero elements in each row of matrix A and the corresponding elements from matrix B, which sparse Tensor Core automatically picks out without run-time overhead, are calculated. So the entire $M \times N \times K$ sparse GEMM will also calculate the same result matrix C with $M \times N$ size but only needs *T/2* GPU cycles, leading to $2\times$ math throughput speedup.

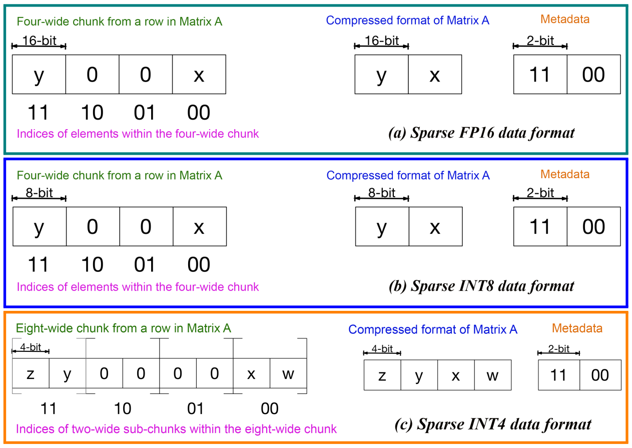
Four-wide chunk from a row in Matrix A
16-bit
y
0
0
x
11
10
01
00
Indices of elements within the four-wide chunk
Compressed format of Matrix A
16-bit
y
x
Metadata
2-bit
11
00
(a) Sparse FP16 data format
Four-wide chunk from a row in Matrix A
8-bit
y
0
0
x
11
10
01
00
Indices of elements within the four-wide chunk
Compressed format of Matrix A
8-bit
y
x
Metadata
2-bit
11
00
(b) Sparse INT8 data format
Eight-wide chunk from a row in Matrix A
4-bit
z
y
0
0
0
0
x
w
11
10
01
00
Indices of two-wide sub-chunks within the eight-wide chunk
Compressed format of Matrix A
4-bit
z
y
x
w
Metadata
2-bit
11
00
(c) Sparse INT4 data format

Figure 2. Storage formats for ***2:4 fine-grained structured sparse pattern*** and metadata with FP16, INT8 and INT4 operators. (w,x,y,z denote the non-zero elements.)

The 2:4 sparsity uses 2-bit metadata per non-zero element to indicate the position of two non-zero elements in every four adjacent elements in a row of matrix Awith FP16 and INT8 data formats. The 2:4 sparsity instruction for the INT4 data format differs from FP16 and INT8. Matrix A is defined as a pair-wise structured sparse at a granularity of 4:8. In other words, each chunk of eight adjacent elements in a row of matrix A has four zero and four non-zero values. Further, the zero and non-zero values are clustered in sub-chunks of two elements each within the eight-wide chunk, i.e., each two-wide sub-chunk within the eight-wide chunk must be all zeroes or all non-zeroes. Only the four non-zero values are stored in the compressed matrix, and two 2-bit indices in the metadata indicate the position of the two two-wide sub-chunks with non-zero values in the eight-wide chunk of a row of matrix A. In conclusion, the sparse format for FP16, INT8, and INT4 lead to 43.75%, 37.5%, and 37.5% savings in storage. **GPUSQ-ViT** will firstly compress model as 2:4 FP16 sparse, then further quantize to 2:4 INT8 or INT4 sparse for best deployment efficiency.

Because the ***2:4 fine-grained structured sparse pattern*** is well supported on NVIDIA GPU and corresponding libraries for math acceleration and memory saving, so we are motivated to ***design the compression strategy for vision transformer models to meet such sparse pattern***. Moreover, the 2:4 sparse GEMM supports low-precision formats like INT8 and INT4. So it is natural to ***combine the sparsity and quantization in the proposed strategy jointly*** and further boost the actual deployment performance on GPUs.

3.2. Apply structured sparsity in transformer block

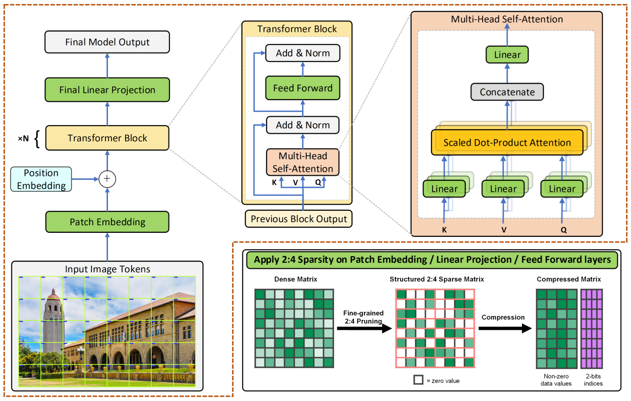

Figure 3. Illustration about applying the ***2:4 fine-grained structured sparsity*** in vision transformer. The target layers include the patch embedding, final linear projection, as well as the feed forward and linear projection inside each transformer block.

The transformer block [52] is the fundamental building structure in various vision transformers. The majority of the weight parameters and the execution time are taken in stacked transformer blocks. For example, about 96% of the weight parameters and 95% of the inference time are from the transformer blocks in Swin Transformer [27]. So we focus on how to apply the ***2:4 fine-grained structured sparsity*** in the transformer block.

Transformer blocks used in vision transformer models are directly borrowed from [9] [51] or made tiny changes [27] [55] on the standard transformer block introduced in the naive attention mechanism [52]. For example, the transformer block in the Swin Transformer model is built by replacing the standard multi-head attention module with a shifted windows attention module [27], with other layers kept the same as the standard transformer block. Without losing the generalization of the proposed method, we explore the utilization of 2:4 sparsity on a standard transformer block. ***2:4 fine-grained structured sparsity*** accelerates GEMM operations, so the Q, K, and V projection layers, the linear projection layer in the multi-head attention module, and the linear projection layers in the feed-forward module are the proper targets to apply, as shown in the zoomed-in parts in Figure 3.

3.3. Apply structured sparsity in patch embedding

The vision transformer paradigm splits each input image into small square patches [9], and each image patch is treated as a token in the same way in the NLP domain. In vision transformer models, the following trainable linear embedding process is handled by a patch embedding layer and is usually implemented as a strided-convolution [9] [27]. Considering the input images are organized as an $N \times C \times H \times W$ batched data format, and each image will be divided into small patches with $P \times P$ square shape, where N refers to batch size, C refers to the number of the input channel, H and W refers to the height and width of an input image, P refers to the size of each patch. So there will be $C \times (H \times W)/(P \times P)$ patches for each image, and each patch will be flattened as a token with shape $1 \times P^2$. Suppose the given embedding dimension is denoted as D_{embed}. In that case, the patch embedding layer can be implemented with a convolution layer with C as the input channel, D_{embed} as the output channel, and kernel size and stride step equal to P. The total Floating Point Operations (FLOPs) of the patch embedding layer is $2 \times N \times C \times H \times W \times D_{embed}$.

The strided-convolution layer is executed as an implicit GEMM [7] [36] on GPUs, which the ***2:4 fine-grained structured sparsity*** can also accelerate, as shown in leftmost of Figure 3. The implicit GEMM transfers the weight matrix of strided-convolution with $C \times P \times P$ as the width of matrix A, which is the target dimension to apply the 2:4 sparsity. It helps to save half of the total FLOPs.

3.4. Overall GPUSQ-ViT compression method

GPUSQ-ViT mainly contains **2:4 structured sparse pruning** and **sparse-distillation-aware QAT** workflows, as shown in Figure 5. KD is applied in each workflow as auxiliary accuracy compensation.

2:4 Structured Sparse Pruning aims to compress the dense floating-point model $\boldsymbol{M}_{DF}$ as the sparse floating-point model $\boldsymbol{M}_{SF}$. Based on Sections 3.2 and 3.3, we can compress each part of a vision transformer model according to the GPU-friendly 2:4 fine-grained structured sparse pattern. To best compensate for the accuracy of $\boldsymbol{M}_{SF}$, we apply KD [14] which can effectively transfer the predicted hard label or soft logits from a teacher model with appealing performance to a student model. If the student model wants to learn more, feature-based KD is applied to mimic the teacher model's feature maps. In 2:4 structured sparse pruning workflow, three KD strategies are jointly used.

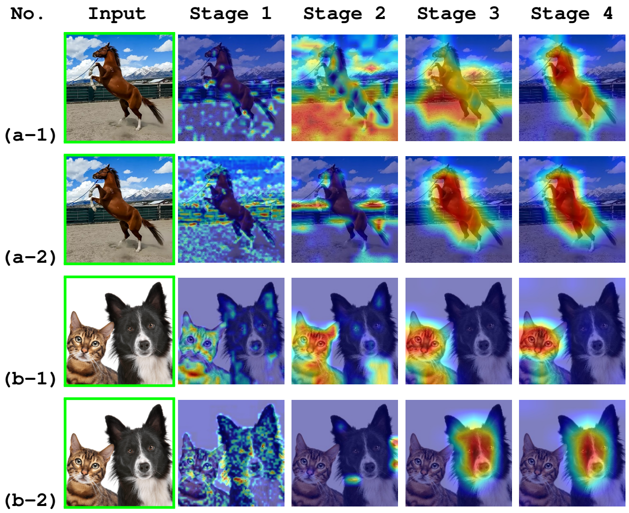

Figure 4. Attention map visualization for Swin Transformer ImageNet-1K pretrained models. **(a-1)** and **(b-1)** Swin-V1-Tiny [2, 2, 6, 2]. **(a-2)** and **(b-2)** Swin-V1-Base [2, 2, 18, 2]. Numbers in square brackets indicate how many Swin Transformer blocks in each stage. We choose the output of last Swin Transformer block in each stage, to generate the CAM visualization results.

To improve the efficiency of feature-based KD, we will not mimic each feature map in the teacher model. *Instead, we find the critical feature maps from the teacher model as learning targets.* We use the tiny- and base-sized Swin Transformer [27] pretrained models as an example to apply the Class Activation Map [46] (CAM) for feature map visualization [53], as shown in Figure 4. The Swin Transformer blocks are organized into four stages with different feature map resolutions. We use the outputs of the last Swin Transformer block in each stage as the representatives. By comparing the CAM results in **(a-1)** and **(a-2)**, we find the attention is focused on local features in the early stages, while focused on global features of the target object in the later stages. Moreover, even though the tiny- and base-sized models provide the same classification result for the horse input image, the CAM from early stages (i.e., stages 1 to 3) are quite different. This phenomenon inspires us that *it is more effective to mimic the feature maps from later stages of the vision transformer models.* By comparing the CAM results in **(b-1)** and **(b-2)**, the tiny-sized model classifies the input as an Egyptian cat, and the base-sized model classifies it as a Border collie. Different classified labels influence the CAM to pay attention to totally different features of a cat and a collie, respectively. *It inspires us to enable mimic feature learning only when the teacher and student models have the same classification labels; otherwise, skip the mimic behavior.*

Denoting distillation losses for the hard label, soft logits and feature maps are $\boldsymbol{L}^{prune}_{hard\_label}$, $\boldsymbol{L}^{prune}_{soft\_logits}$, $\boldsymbol{L}^{prune}_{feature}$, respectively, and their weight factors are: α, β, γ, then the overall sparse pruning loss $\boldsymbol{L}_{prune}$ is calculated as follows:

$$\boldsymbol{L}_{prune} = \alpha * \boldsymbol{L}^{prune}_{hard\_label} + \beta * \boldsymbol{L}^{prune}_{soft\_logits} + \gamma * \boldsymbol{L}^{prune}_{feature} \quad (1)$$

The 2:4 structured sparse pruning workflow minimizes the $\boldsymbol{L}_{prune}$ loss w.r.t weight parameters of $\boldsymbol{M}_{SF}$ model.

Sparse-distillation-aware QAT aims to further compress the sparse floating-point model $\boldsymbol{M}_{SF}$ as the sparse quantized model $\boldsymbol{M}_{SQ}$ on data format, i.e., quantize from the floating-point formats to INT8 or INT4. We mainly discuss the QAT strategy for the following reasons. *From the performance perspective*, QAT can achieve the same deployment efficiency with the toolkit [37]. *From the accuracy perspective*, QAT learns the scale factor adjustment during training, so the learned scale factor leads to less quantization noise and a better accuracy compensation effect. Moreover, compression by 2:4 fine-grained structured sparsity needs the ***premise*** [33] to access the training set and undergo a fine-tuning process. *So we can fully utilize the training set and fine-tuning process to calibrate the quantization scale factor and boost the accuracy of quantized model.*

We borrow the KD idea and jointly learn to calibrate the quantization scale factor from the teacher model's hard label prediction, soft logits, and feature maps from critical layers. Unlike the sparse pruning workflow in which $\boldsymbol{M}_{DF}$ model serves as the teacher and $\boldsymbol{M}_{SF}$ model serves as the student, in the QAT process, $\boldsymbol{M}_{SF}$ model serves as the teacher, and $\boldsymbol{M}_{SQ}$ model serves as the student.[1] *Another difference between the KD strategies in two workflows is a weight factor to multiply the feature-based calibration result from each critical layer.* The value of each weight factor is determined by the feature-based distillation loss between the corresponding layers from $\boldsymbol{M}_{DF}$ and $\boldsymbol{M}_{SF}$ models.

Usually, after the 2:4 structured sparse pruning workflow, $\boldsymbol{M}_{DF}$ and $\boldsymbol{M}_{SF}$ models have similar accuracy. So intuitively, if the distillation loss for the feature map of a specific layer between $\boldsymbol{M}_{DF}$ and $\boldsymbol{M}_{SF}$ models is still significant, *it indicates this layer has little influence on the model's final accuracy* and vice versa. So if the distillation

[1] Using the dense floating-point model serves as the teacher in the QAT process is not recommended, even though it usually has better accuracy than the 2:4 sparse floating-point model. Because based on the previous study [32] [62], the distillation effectiveness will drop if the teacher and student models have a noticeable gap in scale or data format.

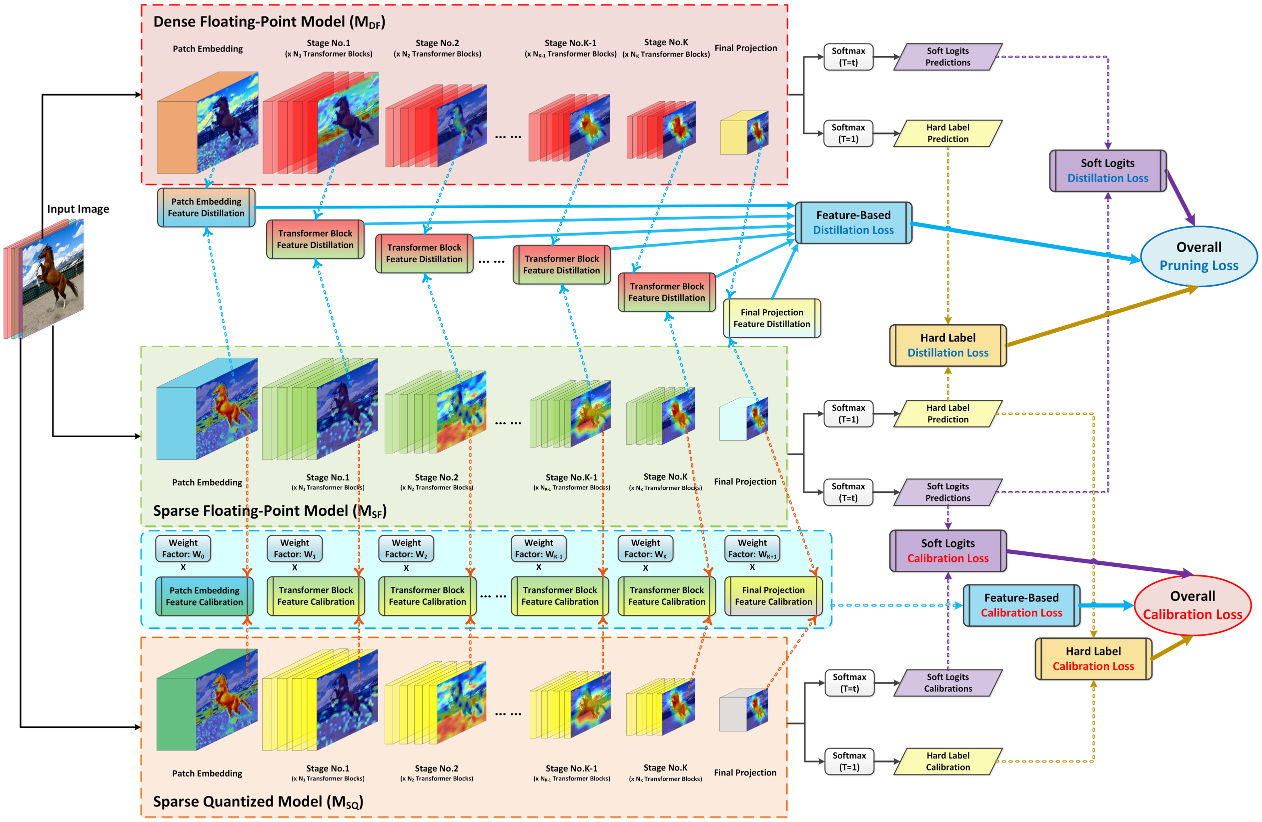
Dense Floating-Point Model (M_DF)
Patch Embedding
Stage No.1 (x N1 Transformer Blocks)
Stage No.2 (x N2 Transformer Blocks)
Stage No.K-1 (x NK-1 Transformer Blocks)
Stage No.K (x NK Transformer Blocks)
Final Projection
… …
Input Image
Softmax (T=t)
Soft Logits Predictions
Softmax (T=1)
Hard Label Prediction
Soft Logits Distillation Loss
Patch Embedding Feature Distillation
Transformer Block Feature Distillation
Final Projection Feature Distillation
Feature-Based Distillation Loss
Overall Pruning Loss
Hard Label Distillation Loss
Sparse Floating-Point Model (M_SF)
Weight Factor: W0
Weight Factor: W1
Weight Factor: W2
Weight Factor: WK-1
Weight Factor: WK
Weight Factor: WK+1
X
Patch Embedding Feature Calibration
Transformer Block Feature Calibration
Final Projection Feature Calibration
Soft Logits Calibration Loss
Feature-Based Calibration Loss
Overall Calibration Loss
Hard Label Calibration Loss
Soft Logits Calibrations
Hard Label Calibration
Sparse Quantized Model (M_SQ)

Figure 5. **GPUSQ-ViT** scheme with two sub-workflows. For the **2:4 structured sparse pruning workflow**, the dense floating-point model $\boldsymbol{M}_{DF}$ is compressed as the sparse floating-point model $\boldsymbol{M}_{SF}$. Hard label, soft logits and feature-based distillation losses are accumulated as the overall sparse pruning loss. The sparse floating-point model $\boldsymbol{M}_{SF}$ is quantized as the sparse quantized model $\boldsymbol{M}_{SQ}$ for the **sparse-distillation-aware QAT workflow**. Hard label and soft logits calibration losses are obtained in a similar manner. Each feature maps calibration result is multiplied with a weight factor to indicate this layer's probability of having a real influence on $\boldsymbol{M}_{SQ}$ model's final accuracy. Three calibration losses are accumulated as the overall quantization calibration loss.

loss value is larger, then we give a smaller weight factor for the corresponding feature-based calibration loss, to indicate even the quantization compression leads to the difference between $\boldsymbol{M}_{SF}$ and $\boldsymbol{M}_{SQ}$ models; however, ***this difference has a low probability of having the real influence on the quantized model's final accuracy***. That's the reason why we named **GPUSQ-ViT** quantization workflow as **sparse-distillation-aware QAT**. Denoting calibration losses for the hard label, soft logits and feature maps are $\boldsymbol{L}_{hard\_label}^{calibrate}$, $\boldsymbol{L}_{soft\_logits}^{calibrate}$, $\boldsymbol{L}_{feature}^{calibrate}$, respectively, and their weight factors are still: α, β, γ, then the overall quantization calibration loss $\boldsymbol{L}_{calibrate}$ is calculated as follows:

$$\boldsymbol{L}_{calibrate} = \alpha * \boldsymbol{L}_{hard\_label}^{calibrate} + \beta * \boldsymbol{L}_{soft\_logits}^{calibrate} + \gamma * \boldsymbol{L}_{feature}^{calibrate} \quad (2)$$

The sparse-distillation-aware QAT workflow minimizes the $\boldsymbol{L}_{calibrate}$ loss w.r.t weight parameters of $\boldsymbol{M}_{SQ}$ model. The details about each loss items in **GPUSQ-ViT** are provided in **Algorithm 1** in **Appendix**.

4. Experiments

For the experiments in this paper, we choose PyTorch [41] with version 1.12.0 as the framework to implement all algorithms. The results of the dense model training, sparse compression, and QAT experiments are obtained with A100 [35] GPU clusters. The acceleration performance results for deployment are obtained with A100 GPU and AGX Orin chip [38] to represent the server and edge device scenarios, respectively. Both A100 and Orin have the Tensor Core [39] support for 2:4 structured sparsity and mixed-precision calculation among FP16, INT8, and INT4. All the reference algorithms use the default data type provided in public repositories.

4.1. Compression efficacy for classification task

To evaluate the compression efficacy of **GPUSQ-ViT** and make the comparison with prior arts on the image classification task, DeiT [51][2] and Swin Transformer [27][3] are chosen as the experiment target models. For the state-of-the-art vision transformer compression methods, we choose the Dyn-ViT [43], MiniViT [66], UVC [64], PS-ViT [50], IA-RED^2 [40], MultiViT [15], SViTE [8] and S^{2}ViTE [8] as the reference methods from sparse pruning category, and

[2] https://github.com/facebookresearch/deit
[3] https://github.com/microsoft/Swin-Transformer

Model	Method	Input	Format	Params (M)	FLOPs (G)	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-Tiny	*Baseline*	224^2	FP32	5.72	1.30	72.2	91.1
	S^2ViTE		FP32	4.21	0.99	70.1	90.1
	SViTE		FP32	3.46	0.86	71.8	90.6
	MiniViT		FP32	3.09	1.30	72.8	91.6
	PS-ViT		FP32	3.08	0.70	72.0	91.0
	UVC		FP32	3.08	0.69	71.8	90.6
	FQ-ViT		INT8	1.43	1.27	71.6	90.6
	GPUSQ-ViT		INT8	0.90 (6.4×)	0.04 (31×)	72.4 (+0.2)	90.9 (-0.2)
	Q-ViT		INT4	0.72	0.34	71.6	90.5
	GPUSQ-ViT		INT4	0.45 (12.7×)	0.02 (62×)	71.7 (-0.5)	90.6 (-0.5)
DeiT-Small	*Baseline*	224^2	FP32	22.05	4.60	79.9	95.0
	DyViT		FP32	26.90	3.70	82.0	95.5
	MultiViT		FP32	16.76	2.90	79.9	94.9
	IA-RED^2		FP32	14.99	3.10	79.1	94.5
	S^2ViTE		FP32	14.60	2.12	79.2	94.6
	MiniViT		FP32	11.45	4.70	80.7	95.6
	PS-ViT		FP32	12.46	2.59	79.4	94.7
	UVC		FP32	12.70	2.65	79.4	94.7
	SViTE		FP32	8.90	1.38	79.4	94.7
	PTQ-ViT		INT8	5.51	5.67	78.1	94.2
	PTQ4ViT		INT8	5.51	3.45	79.5	94.7
	FQ-ViT		INT8	5.51	4.61	79.2	94.6
	GPUSQ-ViT		INT8	3.46 (6.4×)	0.14 (31×)	80.3 (+0.4)	95.1 +0.1)
	Q-ViT		INT4	2.76	1.22	80.1	94.9
	GPUSQ-ViT		INT4	1.73 (12.7×)	0.07 (62×)	79.3 (-0.6)	94.8 (-0.2)
DeiT-Base	*Baseline*	224^2	FP32	86.57	17.60	81.8	95.6
	MultiViT		FP32	64.93	11.20	82.3	96.0
	IA-RED^2		FP32	58.01	11.80	80.9	95.0
	S^2ViTE		FP32	56.80	11.77	82.2	95.8
	MiniViT		FP32	44.10	17.70	83.2	96.5
	PS-ViT		FP32	48.22	9.80	81.5	95.4
	UVC		FP32	39.40	8.01	80.6	94.5
	SViTE		FP32	34.80	7.48	81.3	95.3
	PTQ-ViT		INT8	21.64	20.10	81.3	95.2
	FQ-ViT		INT8	21.64	17.48	81.2	95.2
	PTQ4ViT		INT8	21.64	13.10	81.5	95.3
	GPUSQ-ViT		INT8	13.55 (6.4×)	0.55 (31×)	82.9 (+1.1)	96.4 (+0.8)
	PTQ4ViT		INT4	10.82	6.94	75.9	95.3
	GPUSQ-ViT		INT4	6.78 (12.7×)	0.28 (62×)	81.6 (-0.2)	95.5 (-0.1)
DeiT-Base	*Baseline*	384^2	FP32	86.86	55.60	82.9	96.2
	IA-RED		FP32	54.31	34.70	81.9	95.7
	MiniViT		FP32	44.39	56.90	84.7	97.2
	PTQ4ViT		INT8	21.71	41.70	82.9	96.3
	GPUSQ-ViT		INT8	13.62 (6.4×)	1.74 (31×)	82.9 (+0.0)	96.3 (+0.1)
	GPUSQ-ViT		INT4	6.81 (12.7×)	0.87 (62×)	82.4 (-0.5)	96.1 (-0.1)
Swin-Tiny	*Baseline*	224^2	FP32	28.29	4.49	81.2	95.5
	Dyn-ViT		FP32	19.80	4.00	80.9	95.4
	MiniViT		FP32	12.00	4.60	81.4	95.7
	FQ-ViT		INT8	7.07	4.39	80.5	95.2
	PTQ4ViT		INT8	7.07	3.37	81.2	95.4
	GPUSQ-ViT		INT8	4.43 (6.4×)	0.14 (31×)	81.2 (+0.0)	95.5 (+0.0)
	Q-ViT		INT4	3.54	1.10	80.6	95.2
	GPUSQ-ViT		INT4	2.21 (12.7×)	0.07 (62×)	80.7 (-0.5)	95.3 (-0.2)
Swin-Small	*Baseline*	224^2	FP32	49.61	8.75	83.2	96.2
	Dyn-ViT		FP32	34.73	6.90	83.2	96.3
	MiniViT		FP32	26.46	8.93	83.6	97.0
	FQ-ViT		INT8	12.40	8.77	82.7	96.1
	PTQ4ViT		INT8	12.40	6.56	83.1	96.2
	GPUSQ-ViT		INT8	7.77 (6.4×)	0.27 (31×)	83.1 (-0.1)	96.3 (+0.1)
	GPUSQ-ViT		INT4	3.88 (12.7×)	0.14 (62×)	82.8 (-0.4)	96.2 (+0.0)
Swin-Base	*Baseline*	224^2	FP32	87.77	15.44	83.5	96.5
	Dyn-ViT		FP32	61.44	12.10	83.4	96.4
	MiniViT		FP32	46.44	15.71	84.3	97.3
	FQ-ViT		INT8	21.94	15.33	83.0	96.3
	PTQ4ViT		INT8	21.94	11.58	83.2	96.3
	GPUSQ-ViT		INT8	13.73 (6.4×)	0.48 (31×)	83.4 (-0.1)	96.4 (-0.1)
	GPUSQ-ViT		INT4	6.87 (12.7×)	0.24 (62×)	83.2 (-0.3)	96.3 (-0.2)
Swin-Base	*Baseline*	384^2	FP32	87.90	47.11	84.5	97.0
	MiniViT		FP32	47.00	49.40	85.5	97.6
	PTQ4ViT		INT8	21.98	35.33	84.3	96.8
	GPUSQ-ViT		INT8	13.77 (6.4×)	1.47 (31×)	84.4 (-0.1)	97.0 (0.0)
	GPUSQ-ViT		INT4	6.88 (12.7×)	0.74 (62×)	84.4 (-0.1)	96.9 (-0.1)

Table 1. Compare the model size and FLOPs of **GPUSQ-ViT** with state-of-the-art compression methods on classification task.

we choose the FQ-ViT [25], Q-ViT [23], PTQ-ViT [28] and PTQ4ViT [65] as the reference methods from quantization category. For **GPUSQ-ViT**, the loss adjustment factors for hard label, soft logits and feature-based losses apply $\alpha = 1$, $\beta = 10$, and $\gamma = 5$), respectively. The model size and FLOPs comparison results are shown in Table 1.

We can apply **GPUSQ-ViT** to compress each vision model as INT8 and INT4 versions. For INT8 compressed models, **GPUSQ-ViT** can bring 6.4× reduction for model size and 31× reduction for FLOPs with negligible accuracy drop. For INT4 compressed models, **GPUSQ-ViT** can get 12.7× and 62× reduction for model size and FLOPs with a small accuracy drop. Compared with both sparse pruning and quantization prior arts, **GPUSQ-ViT** can steadily provide more reduction for model size and FLOPs.

Model	Method	Input	Format	NVIDIA A100 GPU		NVIDIA AGX Orin	
				FPS (BS=1)	FPS (BS=256)	FPS (BS=1)	FPS (BS=64)
DeiT-Tiny	*Baseline*	224^2	FP32	3067	14934	2671	4005
	GPUSQ-ViT		INT8	3864 (1.26×)	38978 (2.60×)	3232 (1.21×)	7329 (1.83×)
	GPUSQ-ViT		INT4	4263 (1.39×)	51224 (3.43×)	4193 (1.57×)	8531 (2.13×)
DeiT-Small	*Baseline*	224^2	FP32	1256	5277	877	1280
	GPUSQ-ViT		INT8	1629 (1.30×)	13359 (2.53×)	1096 (1.25×)	2291 (1.79×)
	GPUSQ-ViT		INT4	1809 (1.44×)	17775 (3.37×)	1447 (1.65×)	2701 (2.11×)
DeiT-Base	*Baseline*	224^2	FP32	485	1682	351	513
	GPUSQ-ViT		INT8	645 (1.33×)	4136 (2.46×)	453 (1.29×)	939 (1.83×)
	GPUSQ-ViT		INT4	714 (1.47×)	5643 (3.35×)	569 (1.62×)	1206 (2.35×)
DeiT-Base	*Baseline*	384^2	FP32	256	689	233	303
	GPUSQ-ViT		INT8	350 (1.37×)	1730 (2.51×)	308 (1.32×)	561 (1.85×)
	GPUSQ-ViT		INT4	394 (1.54×)	2315 (3.36×)	371 (1.59×)	761 (2.51×)
Swin-Tiny	*Baseline*	224^2	FP32	621	2907	544	968
	GPUSQ-ViT		INT8	807 (1.30×)	6975 (2.40×)	675 (1.24×)	1946 (2.01×)
	GPUSQ-ViT		INT4	910 (1.46×)	9911 (3.41×)	892 (1.64×)	2275 (2.35×)
Swin-Small	*Baseline*	224^2	FP32	330	1802	309	631
	GPUSQ-ViT		INT8	426 (1.29×)	4411 (2.45×)	392 (1.27×)	1306 (2.07×)
	GPUSQ-ViT		INT4	510 (1.55×)	5942 (3.30×)	516 (1.67×)	1521 (2.41×)
Swin-Base	*Baseline*	224^2	FP32	282	1261	247	433
	GPUSQ-ViT		INT8	388 (1.37×)	3226 (2.56×)	309 (1.25×)	842 (1.94×)
	GPUSQ-ViT		INT4	485 (1.72×)	4071 (3.22×)	410 (1.66×)	1063 (2.45×)
Swin-Base	*Baseline*	384^2	FP32	154	531	140	226
	GPUSQ-ViT		INT8	226 (1.47×)	1310 (2.47×)	180 (1.28×)	414 (1.83×)
	GPUSQ-ViT		INT4	369 (1.79×)	1747 (3.29×)	238 (1.69×)	562 (2.48×)

Table 2. Deployment efficiency of **GPUSQ-ViT** compressed DeiT and Swin Transformer models on NVIDIA GPUs. The latency is measured with batch size 1 on a single A100 GPU and AGX Orin. The throughput is measured with batch size fixed to 256 on a single A100 GPU and with batch size fixed to 64 on a single AGX Orin.

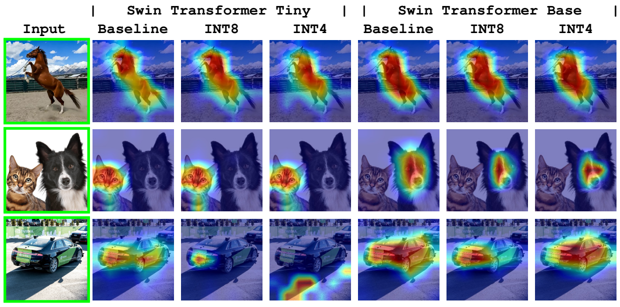

Figure 6. CAM visualization for Swin Transformer baseline dense models and **GPUSQ-ViT** compressed INT8 and INT4 models.

Moreover, **GPUSQ-ViT** can greatly boost the compressed models' deployment efficiency on GPUs with TensorRT toolkit [37] support of 2:4 sparsity. For INT8 compressed models, **GPUSQ-ViT** can bring 1.26-1.47× and 2.4-2.6× improvement for various DeiT and Swin Transformer models of latency and throughput on A100 GPU, and 1.21-1.32× and 1.79-2.07× improvement of latency and throughput on AGX Orin. For INT4 compressed models, **GPUSQ-ViT** can bring 1.39-1.79× and 3.22-3.43× improvement of latency and throughput on A100 GPU, and 1.57-1.69× and 2.11-2.51× improvement of latency and throughput on AGX Orin, as shown in Table 2.

To compare between dense and **GPUSQ-ViT** compressed models in visualization, we apply CAM for tiny- and base-sized Swin Transformer models' attention on final norm layer. The results are shown in Figure 6.

4.2. Compression efficacy for detection task

To evaluate the compression efficacy of **GPUSQ-ViT** on the object detection task, Mask R-CNN [11][4], DETR [5][5] and Deformable-DETR [70] [6] are chosen as the target mod-

[4] https://github.com/SwinTransformer/Swin-Transformer-Object-Detection

[5] https://github.com/facebookresearch/detr

[6] https://github.com/fundamentalvision/Deformable-DETR

els. **GPUSQ-ViT** compression results on COCO dataset [24] are shown in Table 3.

Model	Backbone	Method	Format	Params (M)	FLOPs (G)	bbox mAP	segm mAP
Mask R-CNN	Swin-Tiny	*Baseline*	FP32	48	267	46.0	41.6
		GPUSQ-ViT	INT8	7.5 (6.4×)	8.8 (30.5×)	46.0 (+0.0)	41.6 (+0.0)
		GPUSQ-ViT	INT4	3.8 (12.7×)	4.4 (61.0×)	45.7 (-0.3)	41.4 (-0.2)
	Swin-Small	*Baseline*	FP32	69	359	48.5	43.3
		GPUSQ-ViT	INT8	10.8 (6.4×)	11.8 (30.5×)	48.6 (+0.1)	43.4 (+0.1)
		GPUSQ-ViT	INT4	5.4 (12.7×)	5.9 (61.0×)	48.3 (-0.2)	43.2 (-0.1)
Cascade Mask R-CNN	Swin-Tiny	*Baseline*	FP32	86	745	48.1	41.7
		GPUSQ-ViT	INT8	13.4 (6.4×)	24.4 (30.5×)	48.1 (+0.0)	41.8 (+0.1)
		GPUSQ-ViT	INT4	6.8 (12.7×)	12.2 (61.0×)	47.8 (-0.3)	41.5 (-0.2)
	Swin-Small	*Baseline*	FP32	107	838	51.9	45.0
		GPUSQ-ViT	INT8	16.7 (6.4×)	27.5 (30.5×)	52.0 (+0.1)	45.2 (+0.2)
		GPUSQ-ViT	INT4	8.4 (12.7×)	13.7 (61.0×)	51.7 (-0.2)	44.9 (-0.1)
	Swin-Base	*Baseline*	FP32	145	982	51.9	45.0
		GPUSQ-ViT	***INT8***	22.7 (6.4×)	32.2 (30.5×)	52.1 (+0.2)	45.3 (+0.3)
		GPUSQ-ViT	***INT4***	11.4 (12.7×)	16.1 (61.0×)	51.8 (-0.1)	44.9 (-0.1)
DETR	ResNet50	*Baseline*	FP32	41	86	42.0	N/A
		GPUSQ-ViT	INT8	6.4 (6.4×)	2.8 (30.5×)	42.0 (+0.0)	N/A
		GPUSQ-ViT	INT4	3.2 (12.7×)	1.4 (61.0×)	41.7 (-0.3)	N/A
Deformable DETR	ResNet50	*Baseline*	FP32	40	173	44.5	N/A
		GPUSQ-ViT	INT8	6.3 (6.4×)	5.7 (30.5×)	44.5 (+0.0)	N/A
		GPUSQ-ViT	INT4	3.1 (12.7×)	2.8 (61.0×)	44.1 (-0.4)	N/A

Table 3. Effectiveness of **GPUSQ-ViT** on object detection task.

4.3. Compression efficacy for segmentation task

To evaluate the compression efficacy of **GPUSQ-ViT** on the semantic segmentation task, UPerNet [57][7] is chosen as the target model. **GPUSQ-ViT** compression results on ADE20K dataset [67] are shown in Table 4.

Model	Backbone	Method	Format	Params (M)	FLOPs (G)	Mean IoU (%)	Pixel Acc. (%)
UPerNet	Swin-Tiny	*Baseline*	FP32	60	945	44.51	81.09
		GPUSQ-ViT	INT8	9.4 (6.4×)	31.2 (30.3×)	44.47 (-0.04)	81.01 (-0.08)
		GPUSQ-ViT	INT4	4.7 (12.7×)	15.6 (60.6×)	43.93 (-0.58)	80.89 (-0.20)
	Swin-Small	*Baseline*	FP32	81	1038	47.64	82.45
		GPUSQ-ViT	INT8	12.7 (6.4×)	34.3 (30.3×)	47.66 (+0.02)	82.41 (-0.04)
		GPUSQ-ViT	INT4	6.4 (12.7×)	17.1 (60.6×)	47.15 (-0.49)	82.30 (-0.15)
	Swin-Base	*Baseline*	FP32	121	1188	48.13	82.37
		GPUSQ-ViT	***INT8***	18.9 (6.4×)	39.2 (30.3×)	48.18 (+0.05)	82.43 (+0.06)
		GPUSQ-ViT	***INT4***	9.5 (12.7×)	19.6 (60.6×)	47.86 (-0.27)	82.19 (-0.18)

Table 4. Effectiveness of **GPUSQ-ViT** on semantic segmentation.

GPUSQ-ViT provides good compression effects on detection and segmentation tasks in Table 3 and 4 with small accuracy gap to the dense baseline models.

4.4. GPUSQ-ViT with unsupervised learning

Because the compressed model can learn the representation of target from dense model's prediction when lacking ground-truth label annotations, so **GPUSQ-ViT** can still work well in unsupervised training, as shown in Table 5.

Model	Input	GPUSQ-ViT (INT8)		GPUSQ-ViT (INT4)	
		Top-1 Acc(%)	Top-5 Acc(%)	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-Tiny	224^2	72.0 (-0.2)	90.8 (-0.3)	71.4 (-0.8)	90.2 (-0.9)
DeiT-Small	224^2	79.8 (-0.1)	94.9 (-0.1)	79.2 (-0.7)	94.2 (-0.8)
DeiT-Base	224^2	82.0 (+0.2)	95.7 (+0.1)	81.1 (-0.7)	95.0 (-0.6)
DeiT-Base	384^2	82.5 (-0.4)	95.9 (-0.3)	82.0 (-0.9)	95.7 (-0.5)
Swin-Tiny	224^2	80.8 (-0.4)	95.2 (-0.3)	80.3 (-0.9)	94.9 (-0.6)
Swin-Small	224^2	82.7 (-0.5)	95.9 (-0.3)	82.3 (-0.9)	95.7 (-0.5)
Swin-Base	224^2	82.9 (-0.6)	96.1 (-0.4)	82.5 (-1.0)	95.7 (-0.8)
Swin-Base	384^2	83.9 (-0.6)	96.6 (-0.4)	83.7 (-0.8)	96.4 (-0.6)

Table 5. Effectiveness of **GPUSQ-ViT** in unsupervised learning.

4.5. Ablation study of GPUSQ-ViT

The ablation study to measure the influence of the different adjustment factors for the hard label, soft logits, and feature-based losses (α, β, γ) and enabling sparse-distillation-aware weight factor on **GPUSQ-ViT** compressed model accuracy is shown in Table 6. From the ablation results, we can find enabling sparse-distillation-aware weight factor has an apparent boost for the compressed models' accuracy. Such a boost effect is more influential on INT4 than INT8 model, because disabling this weight factor will see a more significant drop in INT4 compressed model. The potential reason is sparse-distillation-aware weight factor indicates how much influence the quantization error from each critical layer has on the final accuracy. So the distillation process can focus on mimicking the layers with more accuracy influence, which is more effective for limited quantized bits. Then we can find disabling the feature-based distillation will lead to a more severe influence than disabling the soft logits distillation. It indicates that mimicking feature maps is very helpful for accuracy compensation in **GPUSQ-ViT** compression. Finally, we can find **GPUSQ-ViT** is relatively robust to the soft logits and feature-based loss adjustment factors, i.e., within the close range of $\beta = 10$ and $\gamma = 5$ the accuracy of compressed models are stable.

Model	Factor α	Factor β	Factor γ	Enable QAT Weight Factor	GPUSQ-ViT (INT8)		GPUSQ-ViT (INT4)	
					Top-1 Acc(%)	Top-5 Acc(%)	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-Base (224^2)	1	10	5	✓	82.9 (+1.1)	96.4 (+0.8)	81.6 (-0.2)	95.5 (-0.1)
	1	10	5	✗	82.4 (+0.6)	96.1 (+0.5)	80.1 (-1.7)	94.3 (-1.3)
	1	0	5	✓	82.7 (+0.9)	96.2 (+0.6)	81.3 (-0.5)	95.2 (-0.4)
	1	10	0	✓	82.2 (+0.4)	95.8 (+0.2)	80.8 (-1.0)	94.8 (-0.8)
	1	20	5	✓	82.9 (+1.1)	96.4 (+0.8)	81.6 (-0.2)	95.6 (+0.0)
	1	30	5	✓	82.9 (+1.1)	96.5 (+0.9)	81.6 (-0.2)	95.6 (+0.0)
	1	10	10	✓	82.8 (+1.0)	96.5 (+0.9)	81.5 (-0.3)	95.5 (-0.1)
	1	10	2.5	✓	82.8 (+1.0)	96.5 (+0.9)	81.5 (-0.3)	95.6 (+0.0)
Swin-Base (224^2)	1	10	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.3 (-0.2)
	1	10	5	✗	82.9 (-0.6)	96.0 (-0.5)	81.5 (-2.0)	94.9 (-1.6)
	1	0	5	✓	83.2 (-0.3)	96.2 (-0.3)	82.9 (-0.6)	96.0 (-0.5)
	1	10	0	✓	82.7 (-0.8)	95.7 (-0.8)	82.4 (-1.1)	95.5 (-1.0)
	1	20	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.3 (-0.2)
	1	30	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.4 (-0.1)
	1	10	10	✓	83.3 (-0.2)	96.4 (-0.1)	83.1 (-0.4)	96.3 (-0.2)
	1	10	2.5	✓	83.3 (-0.2)	96.4 (-0.1)	83.1 (-0.4)	96.4 (-0.1)

Table 6. Ablation study of the loss adjustment factors and sparse-distillation-aware weight factors of **GPUSQ-ViT** method.

5. Conclusion and limitation

This paper is inspired by GPU's acceleration characteristic for 2:4 sparse pattern with various low-precision operators to design the **GPUSQ-ViT** compression method, which can boost deployment efficiency for various vision transformer models of benchmarking tasks on NVIDIA GPUs.

We should notice a potential *limitation*. If the structured sparse support is changed or extended to support other patterns like 1:4 or 2:16, **GPUSQ-ViT** needs to make the according adjustments to fit the new or more sparse patterns.

Acknowledgements This work is supported by National Natural Science Foundation of China (No. 62071127, U1909207 and 62101137), Shanghai Municipal Science and Technology Major Project (No.2021SHZDZX0103), Shanghai Natural Science Foundation (No. 23ZR1402900), Zhejiang Lab Project (No. 2021KH0AB05).

[7] https://github.com/SwinTransformer/Swin-Transformer-Semantic-Segmentation

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018. 3

[2] Mohamed Arafa, Bahaa Fahim, Sailesh Kottapalli, Akhilesh Kumar, Lily P Looi, Sreenivas Mandava, Andy Rudoff, Ian M Steiner, Bob Valentine, Geetha Vedaraman, et al. Cascade lake: Next generation intel xeon scalable processor. *IEEE Micro*, 39(2):29–36, 2019. 3

[3] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019. 3

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. 2020. 2

[5] 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, 7

[6] Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric P Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4931–4941, 2022. 1

[7] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006. 4

[8] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in Neural Information Processing Systems*, 34:19974–19988, 2021. 6

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. 2020. 1, 4

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 2

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017. 7

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 5

[15] Zejiang Hou and Sun-Yuan Kung. Multi-dimensional vision transformer compression via dependency guided gaussian process search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3669–3678, 2022. 1, 3, 6

[16] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *arXiv preprint arXiv:2206.03382*, 2022. 3

[17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017. 3

[18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 3

[19] Feng Li, Yunming Ye, Zhaoyang Tian, and Xiaofeng Zhang. Cpu versus gpu: which can perform matrix computation faster—performance comparison for basic linear algebra subprograms. *Neural Computing and Applications*, 31(8):4353–4365, 2019. 3

[20] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5284–5294, 2020. 2

[21] Zhikai Li, Liping Ma, Mengjuan Chen, Junrui Xiao, and Qingyi Gu. Patch similarity aware data-free quantization for vision transformers. pages 154–170, 2022. 3

[22] Zhengang Li, Mengshu Sun, Alec Lu, Haoyu Ma, Geng Yuan, Yanyue Xie, Hao Tang, Yanyu Li, Miriam Leeser, Zhangyang Wang, et al. Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization. pages 109–116, 2022. 3

[23] Zhexin Li, Tong Yang, Peisong Wang, and Jian Cheng. Q-vit: Fully differentiable quantization for vision transformer. *arXiv preprint arXiv:2201.07703*, 2022. 1, 3, 7

[24] 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 *European Conference on Computer Vision*, pages 740–755. Springer, 2014. 8

[25] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. Fq-vit: Post-training quantization for fully quantized vision transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1173–1179, 2022. 3, 7

[26] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12009–12019, 2022. 1

[27] 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*, pages 10012–10022, 2021. 1, 4, 5, 6

[28] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103, 2021. 1, 3, 7

[29] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the granularity of sparsity in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 13–20, 2017. 2

[30] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191*, 2018. 2, 3

[31] Szymon Migacz. NVIDIA 8-bit Inference with TensorRT. *GPU Technology Conference*, 2017. 3

[32] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5191–5198, 2020. 5

[33] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021. 1, 2, 5

[34] NVIDIA. NVIDIA Tesla V100 GPU Architecture, 2017. 2, 3

[35] NVIDIA. NVIDIA A100 Tensor Core GPU Architecture, 2020. 1, 2, 3, 6

[36] NVIDIA. NVIDIA CUTLASS, 2022. 4

[37] NVIDIA. NVIDIA TensorRT, 2022. 1, 3, 5, 7

[38] NVIDIA-Orin. NVIDIA Jetson Agx Orin series technical brief, 2021. 6

[39] NVIDIA-TC. NVIDIA Tensor Core, 2020. 2, 6

[40] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. IA-RED2: Interpretability-aware redundancy reduction for vision transformers. *Advances in Neural Information Processing Systems*, 34:24898–24911, 2021. 2, 6

[41] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems-Autodiff Workshop*, 2017. 6

[42] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 3

[43] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in Neural Information Processing Systems*, 34:13937–13949, 2021. 1, 2, 6

[44] Michael Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: Adaptive space-time tokenization for videos. *Advances in Neural Information Processing Systems*, 34:12786–12797, 2021. 2

[45] K Sato. An in-depth look at google's first tensor processing unit (tpu). *Google Cloud Platform*, 2017. 3

[46] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017. 5

[47] Gil Shomron, Freddy Gabbay, Samer Kurzum, and Uri Weiser. Post-training sparsity-aware quantization. *Advances in Neural Information Processing Systems*, 34:17737–17748, 2021. 3

[48] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 2014. 1

[49] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 1

[50] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12165–12174, 2022. 2, 6

[51] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021. 4, 6

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. 1, 2, 4

[53] Haofan Wang, Zifan Wang, Mengnan Du, Fan Yang, Zijian Zhang, Sirui Ding, Piotr Mardziel, and Xia Hu. Score-cam: Score-weighted visual explanations for convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 24–25, 2020. 5

[54] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021. 3

[55] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021. 4

[56] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020. 3

[57] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision*, pages 418–434. Springer, 2018. 8

[58] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021. 1

[59] Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu. End-to-end semi-supervised object detection with soft teacher. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3060–3069, 2021. 1

[60] Tao Yang, Yunkun Liao, Jianping Shi, Yun Liang, Naifeng Jing, and Li Jiang. A winograd-based cnn accelerator with a fine-grained regular sparsity pattern. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 254–261. IEEE, 2020. 2

[61] Zhendong Yang, Zhe Li, Ailing Zeng, Zexian Li, Chun Yuan, and Yu Li. Vitkd: Practical guidelines for vit feature knowledge distillation. *arXiv preprint arXiv:2209.02432*, 2022. 1, 3

[62] Chong Yu. Minimally invasive surgery for sparse neural networks in contrastive manner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3589–3598, 2021. 5

[63] Chong Yu and Jeff Pool. Self-supervised generative adversarial compression. *Advances in Neural Information Processing Systems*, 33:8235–8246, 2020. 2

[64] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. 2022. 3, 6

[65] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization framework for vision transformers. *arXiv preprint arXiv:2111.12293*, 2021. 3, 7

[66] Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12145–12154, 2022. 3, 6

[67] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019. 8

[68] Daquan Zhou, Zhiding Yu, Enze Xie, Chaowei Xiao, Animashree Anandkumar, Jiashi Feng, and Jose M Alvarez. Understanding the robustness in vision transformers. In *International Conference on Machine Learning*, pages 27378–27394. PMLR, 2022. 1

[69] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021. 2

[70] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. 2021. 1, 7