Boost Vision Transformer with GPU-Friendly Sparsity and Quantization

Chong Yu\textsuperscript{1,2}, Tao Chen\textsuperscript{3,*}, Zhongxue Gan\textsuperscript{1,*}, Jiayuan Fan\textsuperscript{1}
\textsuperscript{1}Academy for Engineering and Technology, Fudan University \textsuperscript{2}NVIDIA Corporation \textsuperscript{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 \textit{6.4-12.7} on model size and \textit{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 \textit{1.39-1.79} on latency and throughput on A100 GPU, and \textit{1.57-1.69} and \textit{2.11-2.51} improvements 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. \textit{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:

\begin{itemize}
\item 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.
\item 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].
\item How to keep the accuracy to the best with multiple compression methods and how to generalize on multiple vision tasks lack systematical investigation.
\end{itemize}
Sparse M×N×K GEMMDense M×N×K GEMM
K
A matrix (Dense)
☓
Accumulator (result)
N
Dense operation on Tensor Core
M
B on Tensor Core
Select
B matrix (Dense)
C matrix (Dense)
N
Choose matching K/2 elements out of K elements
M M
K
storage and bandwidth by almost 2
Tensor Core hardware, sparse GEMM can reduce memory rows must be zero. Due to the 2:4 sparsity support on GPU i.e., two values from every four contiguous elements on [33] is imposed on the allowed sparsity pattern, structured sparse pattern
GEMM, a constraint named 2:4 fine-grained structured sparse pattern
2:4 fine-grained structured sparsity
Tensor Core. Dense matrix A of size M × K in left side becomes M × 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 × N and M × N matrices, respectively. By skipping the unnecessary multiplications of redundant zeros, sparse GEMM accelerate the dense GEMM with 2×.

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 Sparsity and Quantization to boost deployment efficacy for Vision Transformer 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.

![Figure 2. Storage formats for 2:4 fine-grained structured sparse pattern and metadata with FP16, INT8 and INT4 operators.](image)

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 A with 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

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 feedforward 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 \( C \times \frac{(H \times W)}{(P \times P)} \) 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 \( D_{\text{embed}} \). In that case, the patch embedding layer can be implemented with a convolution layer with \( C \) as the input channel, \( D_{\text{embed}} \) 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 \( 2 \times N \times C \times H \times W \times D_{\text{embed}} \).

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 $M_{DF}$ as the sparse floating-point model $M_{SF}$. 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 $M_{SF}$, 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](image.png)

**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 $L_{prune \_hard \_label}$, $L_{prune \_soft \_logits}$, $L_{prune \_feature}$, respectively, and their weight factors are: $\alpha$, $\beta$, $\gamma$, then the overall sparse pruning loss $L_{prune}$ is calculated as follows:

$$L_{prune} = \alpha \ast L_{prune \_hard \_label} + \beta \ast L_{prune \_soft \_logits} + \gamma \ast L_{prune \_feature}$$  (1)

The 2:4 structured sparse pruning workflow minimizes the $L_{prune}$ loss w.r.t weight parameters of $M_{SF}$ model.

Sparse-distillation-aware QAT aims to further compress the sparse floating-point model $M_{SF}$ as the sparse quantized model $M_{SQ}$ 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 $M_{DF}$ model serves as the teacher and $M_{SF}$ model serves as the student, in the QAT process, $M_{SF}$ model serves as the teacher, and $M_{SQ}$ 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 $M_{DF}$ and $M_{SF}$ models.

Usually, after the 2:4 structured sparse pruning workflow, $M_{DF}$ and $M_{SF}$ models have similar accuracy. So intuitively, if the distillation loss for the feature map of a specific layer between $M_{DF}$ and $M_{SF}$ 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.
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 \( M_{SF} \) and \( M_{SQ} \) 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 \( L_{\text{calibrate hard label}} \), \( L_{\text{calibrate soft logits}} \), \( L_{\text{calibrate feature}} \), respectively, and their weight factors are still: \( \alpha, \beta, \gamma \), then the overall quantization calibration loss \( L_{\text{calibrate}} \) is calculated as follows:

\[
L_{\text{calibrate}} = \alpha \times L_{\text{calibrate hard label}} + \beta \times L_{\text{calibrate soft logits}} + \gamma \times L_{\text{calibrate feature}}
\] (2)

The sparse-distillation-aware QAT workflow minimizes the \( L_{\text{calibrate}} \) loss w.r.t weight parameters of \( M_{SQ} \) 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] and Swin Transformer [27] 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
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 any 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.

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.

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], DETR [5] and Deformable-DETR [70] are chosen as the target models.
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] is chosen as the target model. **GPUSQ-ViT** compression results on ADE20K dataset [67] are shown in Table 4.

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

**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.

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,
References


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


[34] NVIDIA. NVIDIA Tesla V100 GPU Architecture, 2017. 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 Orin series technical brief, 2021. 6


[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