2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

HeatViT: Hardware-Efficient Adaptive Token Pruning for Vision Transformers

Peiyan Dong[1], Mengshu Sun[1], Alec Lu[2], Yanyue Xie[1], Kenneth Liu[2], Zhenglun Kong[1], Xin Meng[1], Zhengang Li[1], Xue Lin[1], Zhenman Fang[2], Yanzhi Wang[1]
[1]Northeastern University, [2]Simon Fraser University
{dong.pe, sun.meng, xie.yany, kong.zhe, li.zhen, xue.lin, yanz.wang}@northeastern.edu,
{alec_lu, ksl24, zhenman}@sfu.ca, 1601214372@pku.edu.cn

***Abstract*—While vision transformers (ViTs) have continuously achieved new milestones in the field of computer vision, their sophisticated network architectures with high computation and memory costs have impeded their deployment on resource-limited edge devices. In this paper, we propose a hardware-efficient image-adaptive token pruning framework called HeatViT for efficient yet accurate ViT acceleration on embedded FPGAs. Based on the inherent computational patterns in ViTs, we first adopt an effective, hardware-efficient, and learnable head-evaluation token selector, which can be progressively inserted before transformer blocks to dynamically identify and consolidate the non-informative tokens from input images. Moreover, we implement the token selector on hardware by adding miniature control logic to heavily reuse existing hardware components built for the backbone ViT. To improve the hardware efficiency, we further employ 8-bit fixed-point quantization and propose polynomial approximations with regularization effect on quantization error for the frequently used nonlinear functions in ViTs. Compared to existing ViT pruning studies, under the similar computation cost, HeatViT can achieve 0.7%∼8.9% higher accuracy; while under the similar model accuracy, HeatViT can achieve more than 28.4%∼65.3% computation reduction, for various widely used ViTs, including DeiT-T, DeiT-S, DeiT-B, LV-ViT-S, and LV-ViT-M, on the ImageNet dataset. Compared to the baseline hardware accelerator, our implementations of HeatViT on the Xilinx ZCU102 FPGA achieve $3.46\times\sim4.89\times$ speedup with a trivial resource utilization overhead of 8%∼11% more DSPs and 5%∼8% more LUTs.**

Keywords- Vision Transformer; FPGA Accelerator; Hardware and Software Co-design; Data-level Sparsity.

I. Introduction

Transformers [2], [39] have recently made an attractive resurgence in the form of Vision Transformers (ViTs) [11], showing strong versatility in NLP [3], [26], [48], computer vision (e.g., image classification [11], object detection [5], [66], semantic segmentation [62], image processing [7], and video understanding [64]), and complex scenarios with multi-modal data. Furthermore, ViTs can be used as effective backbone networks [4], [11], [19], [50] with superior transferability to downstream tasks through minor fine-tunings. Seemingly, ViTs and transformers have great potential to unify diverse application domains through common architectures and tackle the reliance on scarce domain data, ultimately addressing the two fundamental problems in deep learning: (i) strong reliance on domain data, and (ii) constant model improvements to serve evolving needs. ViTs and transformers are largely considered as one of the future dominant deep learning techniques.

However, to fully unleash advantages of transformer architectures, we need to address the following *challenges* before ViTs and transformers become an indispensable staple in future AI computing. (i) Although the self-attention mechanism is a key defining feature of transformer architectures, a well-known concern is its quadratic time and memory complexity with respect to the number of input tokens. This hinders scalability in many settings, let alone deployment on resource-constrained edge devices. (ii) Majority of existing works on efficient ViT and transformer techniques followed what has been done for Convolutional Neural Networks (CNNs) by using conventional weight pruning [10], [56], [57], [65], quantization [35], [40], [61], and compact architecture design [6], [8], [9], [16], [18], [31], [43], [49], [51], with limited accuracy and speed performance. (iii) In the efforts of exploring token removal to address the quadratic complexity, the static approaches [10], [13], [33], [42], [45] remove tokens with a fixed ratio in an input-agnostic manner, ignoring input sample dependent redundancy; and most of existing image-adaptive approaches [38], [54], [57] simply discard non-informative tokens and do not fully explore token-level redundancies from different attention heads. Both approaches achieve a relatively low pruning rate (to preserve accuracy) or an undermined accuracy (at a high pruning rate). Moreover, none of these studies support efficient hardware implementation on edge devices. (iv) Transformer architectures tend to use more hardware-unfriendly computations, e.g., more nonlinear operations than CNNs, to improve accuracy. Therefore, we need to address the low hardware efficiency issue of such computations, while enjoying the additional optimization dimension provided by multi-head self-attention.

In this paper, we propose HeatViT, a hardware-efficient image-adaptive token pruning framework, together with 8-bit quantization, for efficient yet accurate ViT inference acceleration on embedded FPGA platforms. To improve pruning rate while preserving model accuracy, we make two observations by analyzing the computation workflow in ViTs, similar to what are reported in [28]: (i) the information redundancy in input tokens differs among attention heads in ViTs; and (ii) non-informative tokens identified in earlier transformer blocks may still encode important information when propagating to

978-1-6654-7652-2/23/$31.00 ©2023 IEEE

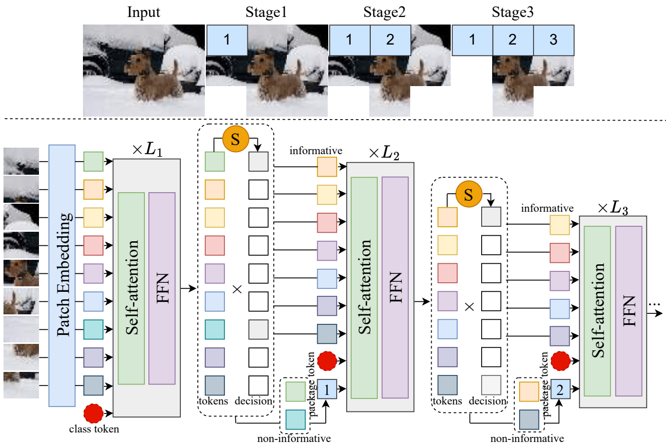


Fig. 1. The workflow of HeatViT on DeiT-S. The token selector S conserves informative tokens conditioned on features from the previous layer. And non-informative tokens are averaged into one package token (blue).

later blocks. Based on these, we adopt an effective token selector module similar to [28] but consider its design with hardware efficiency, which can be inserted before transformer blocks to reduce token number (i.e., number of tokens) with negligible computational overhead. As shown in Fig. 1, we incorporate different token-level redundancies from multiple attention heads to be more accurate in token scoring. Furthermore, instead of completely discarding non-informative ones, similar to [28], we package them together into one informative token to preserve information for later transformer blocks.

For hardware implementation on edge FPGA devices, we design our token selector with linear layers, i.e., fully-connected (FC) layers, instead of convolutional (CONV) layers, to reuse the GEMM (General Matrix Multiply) hardware component built for the backbone ViT (i.e., without token selectors) execution. In addition, we always concatenate the identified (and sparse) informative tokens and the packaged informative token together to form a dense input of tokens to avoid sparse computations on hardware. To improve the hardware efficiency, we further apply 8-bit quantization for weights and activations, and propose polynomial approximations for the frequently used nonlinear functions in ViTs, including GELU [21], Softmax [17], and Sigmoid [37]. Besides, we introduce the regularization effect on quantization error into the design of polynomial approximations to support more ambitious quantization. We design a proof-of-concept FPGA accelerator for ViTs based on our proposed HeatViT. We implement the GEMM engine inspired by [14], [32] to execute the most computation-intensive multi-head self-attention module and feed-forward network in the backbone ViT, and the classification network in our token selector. It is worth noting that only lightweight control logic is added to support our adaptive token pruning by reusing the same hardware components built for the backbone ViT execution.

To reduce inference latency on hardware while preserving model accuracy (typically within 0.5% or 1% accuracy loss), we adopt a similar latency-aware multi-stage training strategy to [28], which (i) determines the transformer blocks for inserting token selectors and (ii) optimizes the desired (average) pruning rates for these token selectors. To reduce the training time, we also adapt the training strategy to insert less number of token selectors and use less number of training epochs: our training effort is roughly 90% of that for training-from-scratch of backbone ViTs without token selectors.

TABLE I. Comparison between representative ViT pruning methods.

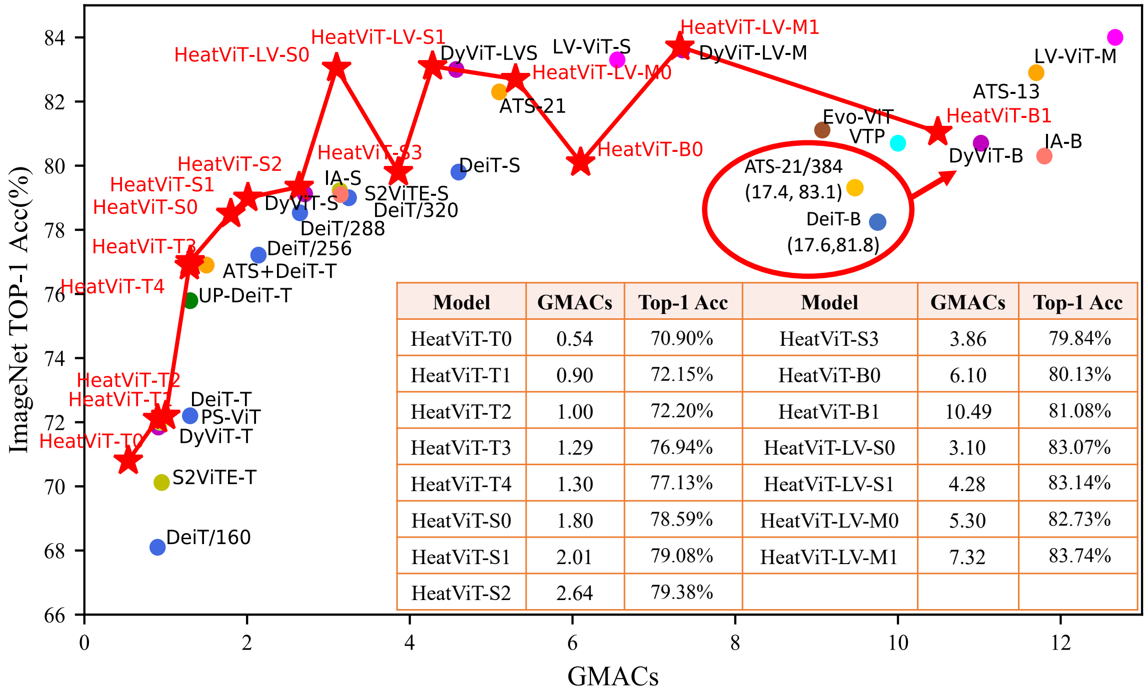
Method	Design Scheme	Method	Design Scheme
DyViT [42]	①	ATS [13]	①
IA [38]	②	PS-ViT [45]	①
VTP [65]	③	Evo-ViT [54]	②
S2ViTE [10]	①③④	UP-DeiT [56]	③④

① – Static Token Pruning; ② – Adaptive Token Pruning; ③ – Head Pruning; ④ – Token Channel Pruning.

As summarized in Fig. 2, we evaluate HeatViT for multiple widely used ViT models on the ImageNet dataset, including DeiT-tiny (DeiT-T, HeatViT-T), DeiT-small (DeiT-S, HeatViT-S), DeiT-base (DeiT-B, HeatViT-B) [46], LV-ViT-small (LV-ViT-S, HeatViT-LV-S), and LV-ViT-medium (LV-ViT-M, HeatViT-LV-M) [25]. These models are already well condensed and thus more challenging to prune compared to larger ViT models. Compared to state-of-the-art ViT pruning studies in Table I (pruning types are explained in detail in Sec II-B), HeatViT achieves better accuracy-computation trade-offs: Under a similar computation cost, HeatViT can achieve 0.7%∼8.9% higher accuracy; while under a similar accuracy, HeatViT can reduce the computation cost by more than 28.4%∼ 65.3%. Compared to the baseline hardware accelerator (16-bit and no token pruning), our implementations of HeatViT on the Xilinx ZCU102 FPGA achieves 3.46×∼4.89×speedup with a trivial resource utilization overhead of 8%∼11% more DSPs and 5%∼8% more LUTs. Compared to the optimized ARM CPU and NVIDIA GPU versions on the NVIDIA Jetson TX2 board with our token pruning, our FPGA implementations achieve 685×∼1695× and 2.68×∼3.79× more speedups, 242.6×∼719.0× and 3.0×∼4.7× higher energy efficiency.

Our contributions are summarized as follows:

- Algorithm and hardware co-design for an effective, hardware-efficient, and learnable token selector to enable efficient image-adaptive token pruning in ViTs.
- A polynomial approximation of nonlinear functions inside ViTs for more ambitious quantization and efficient FPGA-based implementations.
- An end-to-end acceleration framework, with both image-adaptive pruning and 8-bit quantization, for ViT inference on embedded FPGAs.
- Experiments to demonstrate superior pruning rates and inference accuracy of HeatViT over state-of-the-art ViT pruning studies, as well as trivial hardware resource overhead.



Model	GMACs	Top-1 Acc	Model	GMACs	Top-1 Acc
HeatViT-T0	0.54	70.90%	HeatViT-S3	3.86	79.84%
HeatViT-T1	0.90	72.15%	HeatViT-B0	6.10	80.13%
HeatViT-T2	1.00	72.20%	HeatViT-B1	10.49	81.08%
HeatViT-T3	1.29	76.94%	HeatViT-LV-S0	3.10	83.07%
HeatViT-T4	1.30	77.13%	HeatViT-LV-S1	4.28	83.14%
HeatViT-S0	1.80	78.59%	HeatViT-LV-M0	5.30	82.73%
HeatViT-S1	2.01	79.08%	HeatViT-LV-M1	7.32	83.74%
HeatViT-S2	2.64	79.38%			

Fig. 2. Comparison of HeatViT with prior pruning methods: Under a similar computation cost, HeatViT achieves 0.7%∼8.9% higher accuracy; while under a similar accuracy, HeatViT reduces the computation cost by more than 28.4%∼ 65.3%. Final HeatViT models are quantized into 8-bit fixed-point format.

II. Background and Related Work

A. Vision Transformer

Transformers are initially proposed to handle the learning of long sequences in NLP tasks. Dosovitskiy et al. [11] and Carion et al. [5] adapt the transformer architecture to image classification and detection, respectively, and achieve competitive performance against CNN counterparts with stronger training techniques and larger-scale datasets. DeiT [46] further improves the training pipeline with the aid of distillation, eliminating the need for large-scale pretraining [58]. Inspired by the competitive performance and global receptive field of transformer models, follow-up works design ViTs for different computer vision tasks including object detection, semantic segmentation, 3D object animation, and image retrieval.

Fig. 3 illustrates three components in ViTs: *patch embedding*, *transformer encoders*, and *classification output head*. For **patch embedding**, an image $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$ is reshaped into a sequence of 2D patches $\mathbf{X}_p \in \mathbb{R}^{N\times(P^2\cdot C)}$, where (H, W) is original image resolution, C is channel number, (P, P) is the resolution of patches, and $N = HW/P^2$ is the number of patches and also the effective input sequence length (i.e., number of tokens) to transformer encoders. Patch embeddings are obtained by mapping $\mathbf{X}_p$ into D dimensions via a trainable linear projection $\mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D}$: $\mathbf{X}_0 = [\mathbf{X}_{class}; \mathbf{X}_p^1\mathbf{E}; \mathbf{X}_p^2\mathbf{E}; ...; \mathbf{X}_p^N\mathbf{E}]+\mathbf{E}_{pos}$, where D is the constant latent vector size throughout transformer encoders, and $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}$ is learnable position embeddings. A learnable embedding $\mathbf{X}_0^0 = \mathbf{X}_{class}$ is prepended in patch embeddings $\mathbf{X}_0$. Then $\mathbf{X}$ goes through a stack of L transformer encoders for processing. The output $\mathbf{X}_L^0$ (class token) of the final (L-th) transformer encoder is fed to **a classification output head** implemented by an MLP to obtain the *final result*. The MLP module uses two linear layers with a Gaussian Error Linear Unit (GELU) activation layer in between.

The l-th **transformer encoder** receives the patch embeddings $\mathbf{X}_{l-1}$ as input. Fig. 3 illustrates a ViT encoder block, consisting of a multi-head self-attention (MSA) module and an MLP (FFN) module, both with LN (layer normalization) applied before input. The encoder block operations are then:

$$\begin{aligned} \mathbf{X'}_{l-1} &= \text{MSA}(\text{LN}(\mathbf{X}_{l-1})) + \mathbf{X}_{l-1}, \\ \mathbf{X}_l &= \text{FFN}(\text{LN}(\mathbf{X'}_{l-1})) + \mathbf{X'}_{l-1}. \end{aligned} \tag{1}$$

For h heads in MSA, $\text{LN}(\mathbf{X}_{l-1})$ is split into h parts. For each head, the corresponding part in $\text{LN}(\mathbf{X}_{l-1})$ is transferred into query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ through three linear projections W_Q, W_K, and W_V, respectively. Then self-attention function [48] in each head is performed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{D})\mathbf{V}. \tag{2}$$

Attention outputs from all the heads are concatenated and linearly transformed.

B. Model Pruning on ViT

State-of-the-art ViT pruning studies exploring the redundancy from three dimensions in ViTs are summarized below. **Redundancy of the attention head.** Attention head pruning [10], [56], [57], [65] reduces weight redundancy on the transformation matrices (W_Q, W_K, W_V) before MSA operation. Due to the parallel computing nature of the transformer heads, these works directly prune some heads entirely and

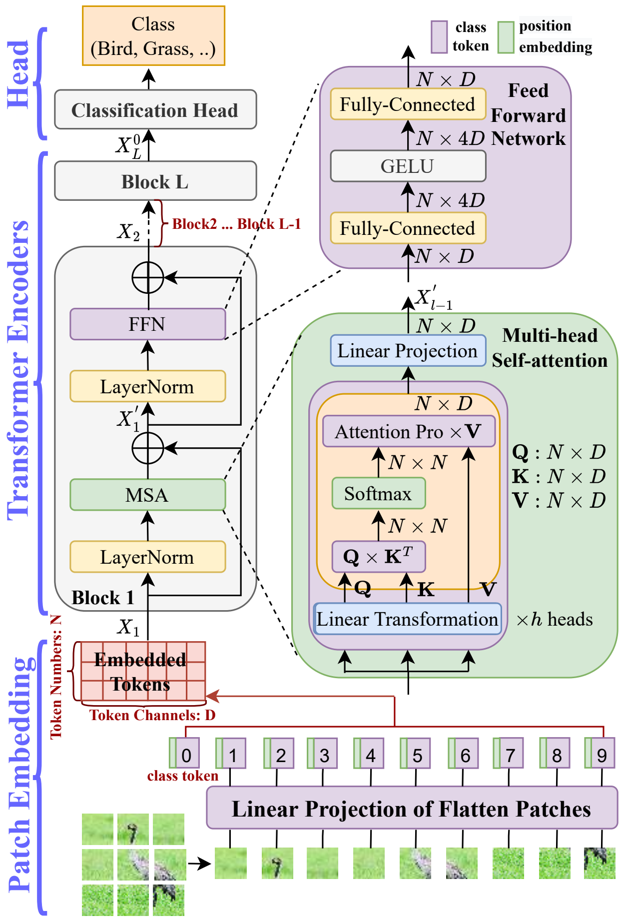


Fig. 3. Overview of ViT

lead to significant accuracy drop, for example, 27% GMACs reduction, but with 2.2% accuracy drop on DeiT-T in [10]. It is an inefficient way of computation reduction because the attention head usually contributes less than 43% of the total computation in several ViT architectures [28].

Redundancy of the token channel. Token channel [10], [56], [57] pruning executes feature-level pruning on the embedding of the token feature to diminish the redundancy of the feature representation. Since this pruning type is equivalent to structured pruning on the token embedding, i.e., removing the same embedding dimension for different tokens, it is difficult to guarantee an ideal pruning rate without significant accuracy deterioration.

Redundancy of the token number. Token pruning [10], [13], [33], [38], [42], [45], [54], [57] aims at removing the non-informative input data. According to [55], the accuracy of neural networks is more related to the object information, but not the background information, implying the input-level redundancy. There is often much higher redundancy along the token number dimension.

C. Computational Complexity

As summarized in Table II, the computational complexity of the ViT can be written as $[4ND_{ch}(hD_{attn_s}) + 2N^2(hD_{attn_s}) + 8ND_{ch}D_{fc}]$. Compared with other prunable dimensions, directly pruning the number of tokens (N) contributes to the reduction of all operations linearly or even quadratically (N^2 in MSA layers), leading to more compression benefits. Please note that since token pruning is equivalent to removing information redundancy within the image data, it can be more freely integrated into the compression process on other model dimensions (e.g., model quantization in our case).

TABLE II. Computational complexity of one ViT block.

Notation: D_{ch} – channel size of a token; D_{attn_s} – sub-channel size of one head; D_{fc} – channel size of FFN; Query ($\mathbf{Q}$), Key ($\mathbf{K}$), Value ($\mathbf{V}$); h – the number of heads; N – the number of tokens.

Module	Input Size	Computation	Output Size	#MACs
MSA	$N \times D_{ch}$	Linear Transformation	$N \times hD_{attn_s}$	$ND_{ch}hD_{attn_s}$
	$N \times hD_{attn_s}$	$\mathbf{Q} \times \mathbf{K}^{\mathrm{T}}$	$N \times N$	$N^2hD_{attn_s}$
	$N \times N$	$\mathbf{QK}^{\mathrm{T}} \times \mathbf{V}$	$N \times hD_{attn_s}$	$N^2hD_{attn_s}$
	$N \times hD_{attn_s}$	Projection	$N \times D_{ch}$	$NhD_{attn_s}D_{ch}$
FFN	$N \times D_{ch}$	FC Layer	$N \times 4D_{fc}$	$4ND_{ch}D_{fc}$
	$N \times 4D_{fc}$	FC Layer	$N \times D_{ch}$	$4ND_{fc}D_{ch}$
Total MACs: $4ND_{ch}hD_{attn_s} + 2N^2hD_{attn_s} + 8ND_{ch}D_{fc}$				

D. Static vs. Image-Adaptive Token Pruning

There are two sub-branches in the token pruning of the ViT: static token pruning and image-adaptive token pruning. Static token pruning [10], [13], [33], [42], [45] limits the compression ratio by reducing the input tokens of each image with a fixed ratio while ignoring the fact that each image's information varies in terms of both region size and location. In contrast, to achieve a per-image adaptive pruning rate, image-adaptive token pruning [28], [38], [54], [57] eliminates redundant tokens depending on the inherent image features. Generally, smaller pruning rates are applied for complex and information-rich images while larger pruning rates for simple and information-less ones. Ideally, it can achieve a larger overall pruning rate than the static one as Fig 4.

However, most of existing adaptive token pruning studies [38], [54], [57] do not consider the visual characteristics inside the ViT (token redundancy in ViT, Sec III) –visual receptive field of different heads, and delete the less informative tokens completely without the opportunity to correct evaluation errors. Moreover, there are no more details on how to determine the location, number, and pruning rate of the pruning evaluation module specifically. Hence, [38], [54], [57] often lead to a limited overall pruning rate (e.g., 35% GMACs reduction on DeiT-S [38]).

Most recently, SPViT [28] analyzes the token redundancy in each ViT head and proposes soft pruning techniques to identify and package pruned tokens to preserve the information of less-informative tokens. Also, it designs an automatic search algorithm to determine the location and settings of each pruning module. However, SPViT only considers the algorithm aspect and lacks the optimization of hardware design and resource utilization on edge devices. It does not take into account of the subsequent model quantization, either, which is often required for hardware implementation and efficiency.

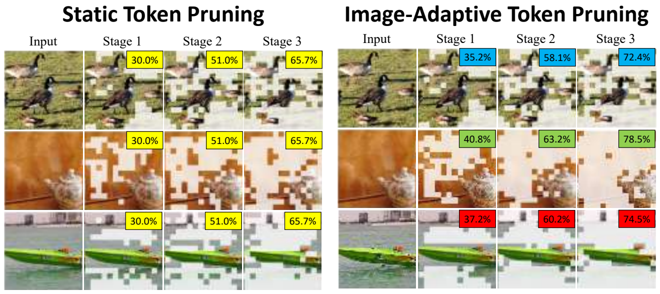


Fig. 4. Comparison between static token pruning and HeatViT image-adaptive token pruning.

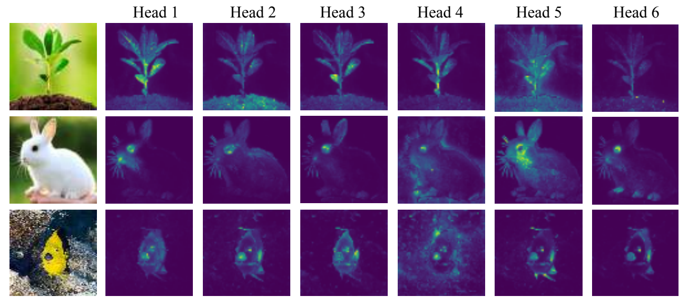


Fig. 5. The information region detected by each head in DeiT-S.

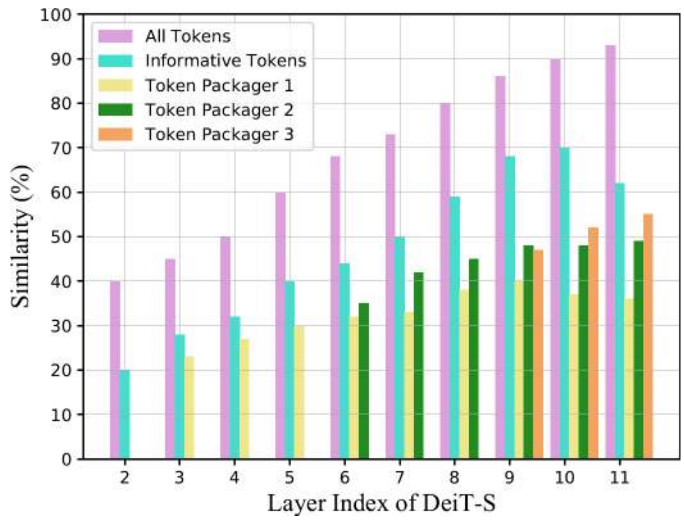


Fig. 6. The similarity between the CLS (class token) and other tokens measured after each ViT block (2 to 11).

Our HeatViT design is inspired by SPViT [28] but is optimized for the actual deployment on embedded FPGA hardware. It differs to SPViT in the following ways. First, we explore different accuracy and hardware trade-offs for the token selector module design, including the comparison between fully-connected layer and convolutional layer, and comparison between different activation functions such as ReLU [1], GELU [21], and Hardswish [23]. Second, we apply more ambitious 8-bit quantization and a polynomial approximation of nonlinear functions inside ViTs—such as GELU [21], Softmax [17], and Sigmoid [37]—to support quantization and efficient FPGA-based implementations, without losing accuracy. In the training process, we also stop the training earlier by skipping early ViT blocks that are more accuracy-sensitive to the pruning and stopping the pruning module insertion once we meet the target latency. After the pruning stage, the algorithm enters the quantization process. Lastly, we also implement an end-to-end ViT hardware accelerator on embedded FPGAs by heavily reusing the hardware components built for the backbone ViT to support the adaptive token pruning module and 8-bit data precision.

E. Transformer Accelerators on FPGAs

Previous state-of-the-art FPGA accelerators on Transformer-based models [30], [41] and [60] also leverage sparsity. However, they have three main limitations. (i) [30] depends on block-circulant matrix-based weight representation in order to replace the matrix-vector multiplication in FC layers with FFT/IFFT-based processing elements. However, [30] accelerates only the MSA part of the model and omits the FFN part (about 65% computation of the whole model). (ii) [41] utilizes the block-wise weight pruning and [60] implements the structure pruning (row-wise or column-wise) to compress weights to similar sizes (lower memory footprint). However, the pruning granularity of both is coarse, resulting in severe accuracy degradation at limited compression rates. For example, [41] shows 2% accuracy drops under the 36% sparsity. (iii) These FPGA-based accelerators are mainly for transformers in the Natural Language Processing (NLP) tasks, not for computer vision ones. In addition, they all compress the model weights, which means that new substructures or additional overhead is needed to support the specific sparse matrix multiplication. According to our Section III analysis, the difference in input data type will introduce a unique redundancy that motivates an effective compression space. Experiments show that adaptive token pruning at the data level can speed up the model inference with negligible accuracy losses and little overhead for hardware implementations.

III. Analysis of Token Redundancy in ViT and Overview of HeatViT

A. Token Redundancy in ViT

Token redundancy viewed by different attention heads. Assuming that the class (CLS) token is significantly correlated with the final prediction [63], we visualize the information regions detected by the CLS token in different attention heads of DeiT-S, as shown in Fig. 5. It can be seen that each head extracts the features of the image (multi-head visual receptive area of the image) independently and differently [22], [36], [38], indicating that the heads contain distinct token-level redundancy. This inspires the token selector design with multiple heads, which is elaborated in Section IV-A.

Token redundancy viewed by different transformer blocks. Fig. 6 gives the centered kernel alignment (CKA) [29] that represents the similarity between the tokens in each transformer block and the final CLS token, showing a tendency from weak

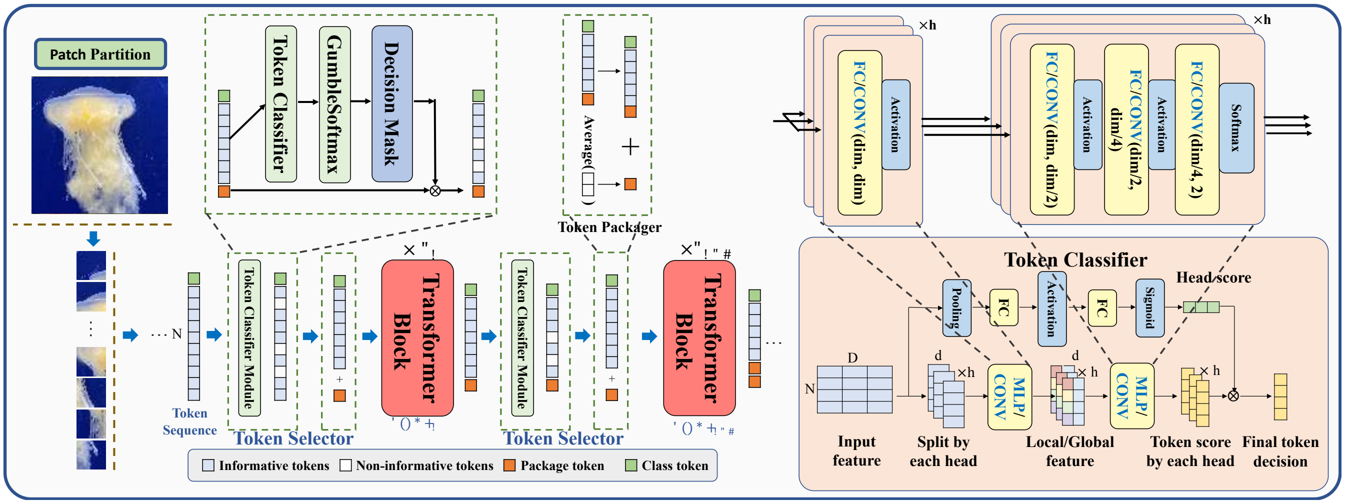


Fig. 7. HeatViT overview. Left: Transformer blocks split into multiple stages with token selectors inserted between them (One token selector includes one token classifier and one token packager). Right: Multi-head token classifier to identify informative tokens.

to strong. It tells us that it is inaccurate for each transformer block to encode or evaluate the image tokens, especially in the front blocks. Thus, pruning rates for these front blocks should be lower to avoid ruling the informative tokens out. This also inspires the token packager technique to provide chances to make up for pruning mistakes (Section IV-B).

These two observations are consistent with what have been reported in [28].

B. Overview of HeatViT

To develop a hardware-efficient image-adaptive token pruning framework, we first adopt an effective and hardware-efficient adaptive token pruning module (also known as **token selector**, based on [28]) according to the vision redundancy in ViTs (Section IV). This module includes an *head-evaluation multi-head token classifier* and a *token packager*, as shown in Fig. 7. To improve the pruning rate and accuracy, the token classifier incorporates different token-level redundancies in multiple heads to more accurately classify tokens and the token packager consolidates (instead of discarding) non-informative tokens to one informative token to reserve information for later transformer blocks. To improve the hardware efficiency, we choose linear layers for the token selector to reuse the GEMM hardware component for the backbone ViT, and always concatenate the classified sparse tokens into dense ones. Please note that it is challenging for CNN-based architecture to implement data-level pruning, because the kernel size of the convolution operation is fixed so that the irregular input features cannot be directly concatenated into dense ones to speed up the model inference. Moreover, we deploy *8-bit fixed-point quantization* to further compress the model, and propose a *polynomial approximation of nonlinear functions* inside ViTs for FPGA-based efficient implementations and regularize the quantization errors. A *proof-of-concept hardware accelerator* design is presented in Section V. To meet the target ViT inference speed on hardware while reserving the accuracy (typically within 1% accuracy loss), one also needs to carefully decide the number of token selectors to insert in the backbone ViT, the location and pruning rate of each token selector, as well as the quantization influence on the accuracy performance. To address this challenge, in Section VI, based on [28], we adapt a more efficient *latency-aware multi-stage training strategy* to learn all these parameters.

IV. Adaptive Token Pruning Module

Shown in Fig. 7, the adaptive token selector is based on observations in Section III and inspired by SPViT [28], including the head-evaluation multi-head token classifier and the token packager. Note that, different to SPViT, nonlinear functions involved in our token pruning module and the ViT backbone are the implementation of polynomial approximation as described in Section V-D. Next, we give a brief introduction to the token classifier and token packager design (for interested audience, please refer to [28] for more details), and focus more on the trade-off of accuracy and hardware efficiency.

A. Head-Evaluation Multi-Head Token Classifier

Multi-Head Token Classifier. As shown in Fig. 7 (right side), each head focuses on extracting different content and position of an image by using a simple learnable neural network, demonstrating the different importance of each token towards each head. Different to SPViT that uses a default fully-connect layer, in Section IV-C, we will explore different neural network units for the consideration of accuracy and hardware efficiency trade-offs. At the end, our multi-head classifier generates a score-map vector S for each input token, which separately marks the information amount of each token in each head.

Head-Evaluation Branch. To synthesize the importance of each head, A, we also add a head-evaluation branch along the

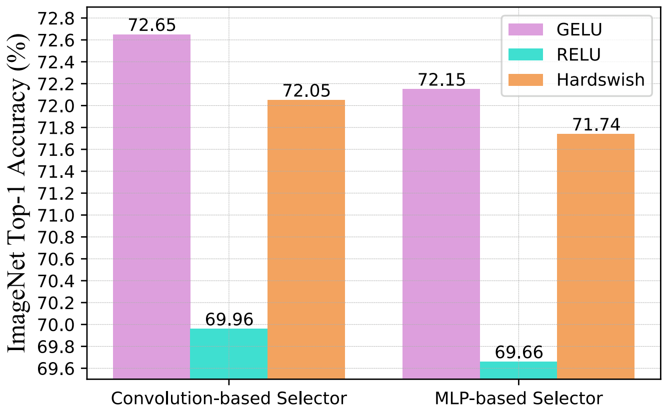


Fig. 8. Accuracy comparison of different token selector structures under the same computational cost. We use DeiT-T as the backbone network, where its accuracy is 72.2% before the pruning.

classifier backbone based on the attention mechanism [24]. Then the token score S is weighted averaged by A:

$$\tilde{S} = \frac{\sum_{i=1}^{h} s_i * a_i}{\sum_{i=1}^{h} a_i} \in \mathbb{R}^{N\times 2} \tag{3}$$

Then we apply the Gumbel-Softmax on $\tilde{S}$ for the token keep/prune decision mask, M. Since deleted image tokens cannot appear in subsequent blocks, M passes on to the following blocks and will be updated by the next pruning step.

B. Token Packager

To avoid pruning useful token information in earlier ViT blocks, we apply a token packaging step that summarizes non-informative tokens (predicted by the classifier) into a package token instead of completely discarding them. Assuming there are T (evaluated by the classifier) non-informative tokens $\{\hat{x}_t\}_{t=1}^{T}$ along with their token scores $\{\tilde{s}_t\}_{t=1}^{T}$, these tokens are compressed into one token through weighted averaging:

$$P = \frac{\sum_{t=1}^{T} \hat{x}_t * \tilde{s}_t[0]}{\sum_{t=1}^{T} \tilde{s}_t[0]} \in \mathbb{R}^{1\times D} \tag{4}$$

P continues the subsequent calculations along with the informative ones, enabling the model to correct scoring mistakes.

C. Exploration for Hardware Efficiency

Hardware Consideration for Token Classifier. Different from SPViT [28] that uses fixed MLP and GELU to extract image features in the token classifier by default, we explore different alternatives for the design, considering the trade-off between model accuracy and hardware efficiency. For the major operation units to extract feature information, we consider both fully-connected layers (i.e., FC, or MLP) and convolutional layers (i.e., conv), where the prior favors better hardware efficiency and the latter favors better model accuracy. For the activation function, we consider three alternatives: more hardware-friendly ReLU [1] and ReLU-based Hardswish [23], and more accurate GELU [21].

In Fig. 8, we compare these different alternatives, where we set their computation cost as the same: 0.9 GMACs after pruning for the DeiT-T model; more detailed experimental setup will be presented in Section VII-A. Compared to MLP layers, the final accuracy of using convolutional layers is further improved by $0.3\% \sim 0.5\%$ with the same computational complexity. However, to maximize the hardware resource utilization, we choose the MLP layer as our main computational unit, since it has already been there in the backbone ViT blocks and the accuracy is acceptable. For the activation function, although ReLU is much more hardware friendly, it would lead to a $2.49\% \sim 2.69\%$ accuracy drop. Even though Hardswish is more hardware friendly than GELU, we still choose GELU as the activation function, since 1) it achieves $0.3\% \sim 0.6\%$ better accuracy than Hardswish, 2) it already exists in the backbone ViT blocks, and 3) we can apply the polynomial approximation of GELU to optimize its hardware efficiency (Section V-D).

Hardware Consideration for Token Packager. In the token packaging step, the sparse input matrices—i.e., all the informative tokens and package token—will be concatenated into a new smaller-size dense matrix to complete the computation in the following blocks, which can speed up the model inference directly.

Since most of the component operations (FC layer, Softmax, GELU) in the token selector already exist in the backbone ViT blocks, we can maximize the resource utilization and leverage our unified operation-level optimization scheme of the ViT deployment on FPGA.

V. ViT Hardware Accelerator Design with Adaptive Token Pruning

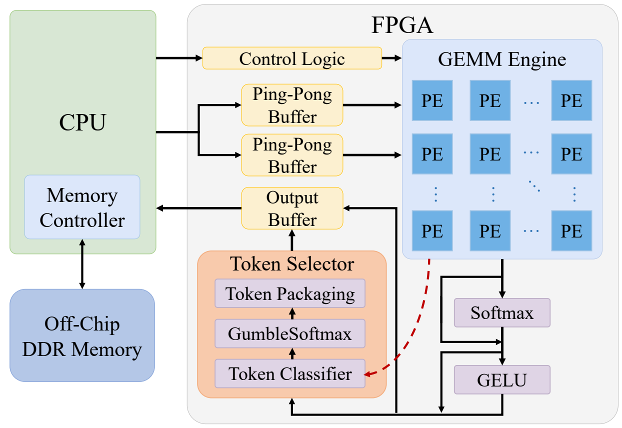
The hardware architecture of ViT accelerators in HeatViT is illustrated in Fig. 9, including the accelerator design for pruned ViTs with token selector and the computation flow in the General Matrix Multiply (GEMM) engine. Besides the LayerNorm layer (less time consuming but more complex to implement on the FPGA) that is left on the ARM CPU, all other components in HeatViT are implemented on the FPGA.

A. Challenges

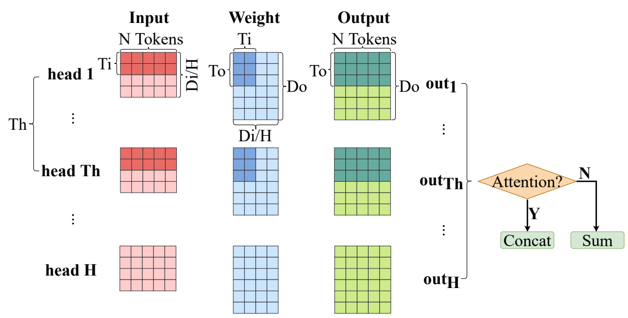
To implement ViT FPGA accelerator with our dynamic token pruning, we address the following challenges. (i) The token selector module should be implemented with minimal hardware overhead by adding miniature control logic and reusing existing hardware for the backbone ViT. (ii) The GEMM loop tiling should accommodate an additional tiling dimension due to multi-head parallelism. (iii) ViTs use more nonlinear operations than CNNs, and we need to refine these operations for more aggressive quantization and efficient hardware implementations without losing accuracy.

B. ViT Accelerator with Dynamic Token Selection

As displayed in Fig. 9(a), the input (tokens) and weight of each ViT layer are loaded from the off-chip DDR memory to the on-chip buffers, and processed by the GEMM engine which we will implement motivated by [15], [32]. Outputs are further processed with the activation function (Softmax or



(a) ViT hardware accelerator arch. supporting dynamic token selection.



(b) GEMM loop tiling with one extra tiling from multi-head mechanism.

Fig. 9. Hardware architecture of ViT accelerator in HeatViT.

GELU), and then stored from the on-chip output buffer back to the off-chip memory. Double buffering will be applied to overlap data transfer time with computation. The token selector for pruning consists of FC layers and GELU activations, which also exist in the original ViTs, and thus can be managed by the same computation engine with negligible hardware overhead (Section V-C). Note LayerNorm is executed on the CPU.

1) Loop Tiling in GEMM: We generalize loop tiling [59] for GEMM engine to deal with relatively large ViT layers. The concurrency of MAC operations in matrix multiplications requires pipelining and loop unrolling, as well as array partitioning of buffers. Fig. 9(b) shows detailed computation flow in GEMM loop tiling for ViTs accommodating an additional tiling dimension from multi-head mechanism. Given a ViT layer j from ①∼⑥ in Table II that performs one or h matrix multiplications with its N_j tokens after pruning, we represent input size as $N_j \times D_i$, weight matrix size as $D_i \times D_o$, and output size as $N_j \times D_o$, where D_i denotes D_{ch}, D_{attn}, or N in the MHSA module, and D_{ch} or $4D_{fc}$ in the MLP module. Tiling is applied to D_i and D_o dimensions, with tiling sizes T_i and T_o, respectively. For attention computations ($\mathbf{Q} \times \mathbf{K}^{\mathrm{T}}$ in ② and $\mathbf{QK}^{\mathrm{T}} \times \mathbf{V}$ in ③), both input and output are split into h groups. A control signal is used to indicate whether current computations are attention-related. Results from ② and ③ are kept in h groups, while results from other layers are accumulated. To improve throughput, we optimize parallelism factors including T_i, T_o, and T_h (an additional tiling dimension for h heads). Therefore, we will conduct comprehensive FPGA resource modeling for available computing and on-chip memory resources.

2) Throughput and Resource Utilization Analysis: The inference throughput (FPS) of ViT inference is determined by the parallelism factors T_i, T_o and T_h, which are bounded by the number of available computation (mainly DSPs) and on-chip memory (BRAMs) resources. The inference is executed layer-by-layer, and the FPS can be inferred through dividing the total number of clock cycles required to process all the ViT layers with the working frequency on FPGAs. The throughput and resource utilization analysis of ViT accelerators is similar to that of CNN accelerators [44], except that the head dimension in ViTs needs to be additionally considered for parallelism. Since the layers in token selectors can be managed by the same GEMM engine as for the existing ViT layers, this analysis also applies to token selectors.

C. Token Selection Flow

Token selection contains token classification to determine informative and non-informative tokens using GumbelSoftmax with a threshold value (usually 0.5), and token packaging to average the non-informative tokens to one that is then consolidated into the informative tokens. The pruned token (input) sequences with sparsity will be reorganized as dense ones, eliminating hardware overhead for indexing. Fig. 10 describes the three steps to implement token selection in our ViT hardware accelerator: (1) calculating the exponent for each token x_i and the summation Sum of all these exponents; (2) dividing each exponent by the sum and classifying the corresponding token as informative or not according to a threshold; and (3) if the token is informative, concatenating it to the informative token sequence, otherwise adding it to a temporary token Tmp. Finally, Tmp is averaged and concatenated to the informative token sequence.

D. Polynomial Approximation of Nonlinear Functions

ViT models contain nonlinear functions including GELU, Softmax, and Sigmoid. (i) Some nonlinear operations inside those functions, e.g., exponential function $\exp(x)$ and error function $\mathrm{erf}(x)$ consume large amounts of computing resources when implemented with the built-in Xilinx Vitis HLS math library [52], and thus incurring difficulty for hardware acceleration (Table III). (ii) To apply more aggressive quantization than CNN/RNN models, we need to add the regularization effect on quantization error to these approximate operations. Inspired by [27], we propose to explore algorithm-level polynomial approximation to implement GELU and Softmax functions, through which we introduce δ_1 and δ_2 (both <1) to control the regularization effect. Since the Sigmoid function is only present inside token selectors (a small number), we do not introduce a regularization effect for it.

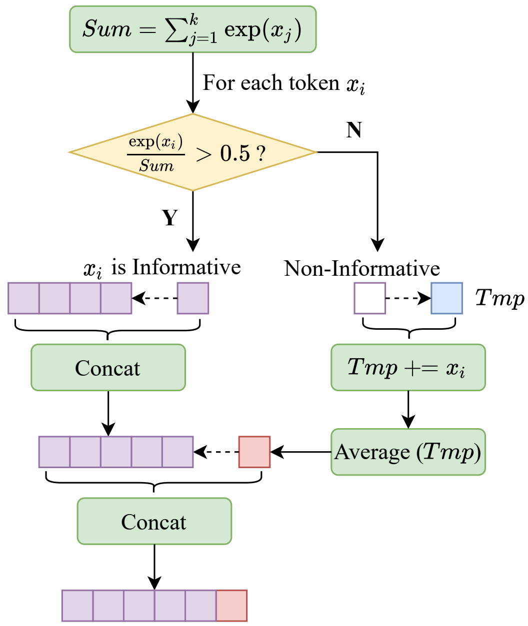


Fig. 10. Token selection flow.

The $\mathrm{erf}(x)$ function is approximated using a second-order polynomial as,

$$L_{\mathrm{erf}}(x) = \mathrm{sign}(x) \cdot \delta_1 \cdot [a(\mathrm{clip}(|x|, \max = -b) + b)^2 + 1], \quad (5)$$

with the constants $a = -0.2888$, $b = -1.769$ and $\delta_1 < 1$.

The GELU function is then expressed as

$$\mathrm{GELU_{aprx}}(x) = \frac{x}{2}\left[1 + L_{\mathrm{erf}}\left(\frac{x}{\sqrt{2}}\right)\right], \quad (6)$$

The Softmax function is approximated as

$$\mathrm{Softmax_{aprx}}(\mathbf{x}_i) = \frac{\delta_2 \exp(\tilde{x}_i)}{\sum_{j=1}^{N} \exp(\tilde{x}_j)}, \quad (7)$$

with $\tilde{x}_i = x_i - x_{\max}$, $x_{\max} = \max_i(x_i)$ and $\delta_2 < 1$. This subtraction ensures the numerical stability during the approximation calculation, and all inputs can be decomposed as $\tilde{x} = (-\ln 2)z + p$, where z is a non-negative integer and p is a real number in $(-\ln 2, 0]$. $\exp(\tilde{x})$ can then be calculated as $\exp(p) >> z$, where $z = \lfloor -\tilde{x}/\ln 2 \rfloor$, $p = \tilde{x} + z \ln 2$, and $\exp(p)$ is approximated as

$$\exp(p) = 0.3585(p + 1.353)^2 + 0.344. \quad (8)$$

Both δ_1 and δ_2 are regularization value (< 1) on quantization error, which can be constant (δ_1=0.5, δ_2=0.5 in our case).

For the Sigmoid function, we adopt the piece-wise linear approximation (PLAN) from [47].

Table III compares the resource utilization for these non-linear functions between the original implementations using the built-in Xilinx Vitis HLS math library and our proposed implementations. Our methods are more resource efficient than those using the HLS math library, with 1.5×∼572× resource improvement. Furthermore, for each model, we try multiple sets of token pruning ratios and there is no accuracy drops between the approximate model and the original one.

TABLE III. Resource utilization for nonlinear functions between original (Orig.) and approximation (Aprx.) implementations.

	GELU		Sigmoid		Softmax	
	Aprx.	Orig.	Aprx.	Orig.	Aprx.	Orig.
FF	334	191116	1015	2334	1939	2464
LUT	438	160909	1512	2333	2364	2476
DSP	4	139	0	3	2	3

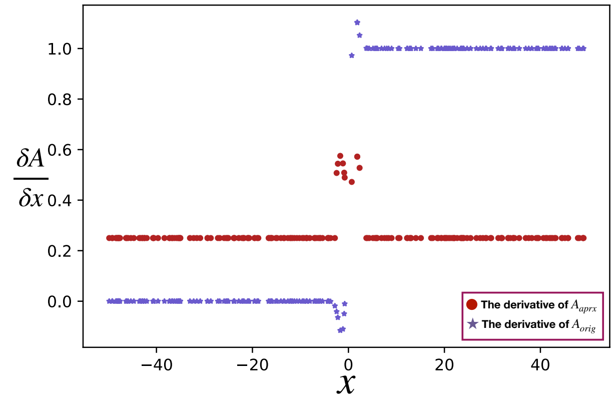


Fig. 11. Regularization effect on quantization error of approximated GELU.

E. Regularization Effect on Quantization Error

GELU and Softmax functions are abundant in transformer blocks, which inspires us to introduce the regularization effect of quantization error into the approximated functions for more aggressive quantization (e.g., 8-bit fixed-point quantization in our case). Here we proof that our regularization works.

GELU for activation data is: A=GELU(x). Quantization on x can be considered as adding a small error Δe to x. We examine the influence of Δe on output A by computing the GELU derivative $\frac{\partial A}{\partial x}$. Assuming that A changes by Δe>0, we can obtain the absolute error of output A:

$$\mathrm{Error_{gelu}}(x) = \frac{\partial A}{\partial x} \cdot \Delta e. \quad (9)$$

Since $\frac{\partial A_{aprx}}{\partial x}$ is always < 1 (Fig. 11) for GELU, the total quantization error is reduced after approximation.

Softmax for activation data is: $A_i = \frac{\delta_2 \exp(\tilde{x}_i)}{\sum_{j=1}^{N} \exp(\tilde{x}_j)}$. As a similar process as GELU, we compute the $Softmax_{aprx}$ derivative $\frac{\partial A}{\partial x}$:

$$\frac{\partial A}{\partial x} = \begin{cases} \frac{\partial \frac{\delta_2 \exp(\tilde{x}_i)}{\sum_{j=1}^{N} \exp(\tilde{x}_j)}}{\partial \tilde{x}_j} = \delta_2 \cdot A_i \cdot (1 - A_i), & i = j \\ \frac{\partial \frac{\delta_2 \exp(\tilde{x}_i)}{\sum_{j=1}^{N} \exp(\tilde{x}_j)}}{\partial \tilde{x}_i} = -\delta_2 \cdot A_i \cdot A_j, & i \neq j \end{cases}. \quad (10)$$

Assuming A_0 changes by Δe_0, the absolute error of all outputs with Equation (10) is:

$$\begin{aligned} Error_{\mathrm{softmax}} &= |\delta_2 \Delta e_0 A_0(1 - A_0)| + \sum_{i=1}^{N-1} |-\delta_2 \Delta e_0 A_0 A_i| \\ &= 2\delta_2 |\Delta e_0|\ A_0(1 - A_0) < \Delta e_0, \end{aligned} \quad (11)$$

since $0 \leq A_0 \leq 1$, $2A_0(1 - A_0)$ is always smaller than 1 and $2\delta_2 A_0(1-A_0)$ is further reduced ($\delta_2 < 1$). So, the total quantization error after Softmax approximation is $< \Delta e_0$.

VI. Latency-Aware Multi-Stage Training Strategy

Our training strategy is as follows: (1) We leverage the same latency-sparsity loss concept in [28] to take into account the latency characteristics of the hardware side during the training process. (2) We design a block-to-stage training pipeline to learn the number of token selectors to insert in the backbone ViT, their locations and pruning rates. Different from the brute-force token selector insertion in [28] (i.e., insert a token selector after every ViT block), we reduce the number of token selectors to insert and reduce the training epochs. Note that (i) the block-to-stage training is based on token pruning, and 8-bit quantization on weight and activation will lead to 2×∼2.4× speedup without accuracy drops; and (ii) our training strategy uses finetuning for each selector, and the training effort of our pipeline is equivalent to the effort of training-from-scratch of the backbone ViT.

Relationship between Latency and Keep Ratio. We build the latency-sparsity table for the target FPGA as Table IV. In this paper, we measure the actual numbers from our FPGA implementation. Each time we only input the $KeepRatio$ tokens into one ViT block with a selector and test the latency.

TABLE IV. Tested latency of one DeiT block with different token keeping ratios on ZCU102 FPGA.

Keep Ratio		1.0	0.9	0.8	0.7	0.6	0.5
Latency (ms)	DeiT-T	1.034	0.945	0.881	0.764	0.702	0.636
	DeiT-S	3.161	2.837	2.565	2.255	1.973	1.682

Latency-Sparsity Loss. ℓ_{ratio} is built as follows:

$$\mathrm{t_i}(\rho_i) = latency\_sparsity\_table(1-\rho_i) \tag{12}$$

$$\sum_{i=1}^{L} \mathrm{t}_i(\rho_i) \leq L_{target} \tag{13}$$

$$\ell_{ratio} = \sum_{i=1}^{L}(1-\rho_i - \frac{1}{B}\sum_{b=1}^{B}\sum_{j=1}^{N} M_j^{i,b})^2 \tag{14}$$

where Eq. (12) shows the look-up-table function (in Table IV) for the latency t_i of $\mathrm{Layer_i}$, under the pruning rate ρ_i. Eq. (13) constraints the inference speed of the whole model satisfies the target hardware latency requirement (only for the pruning part, and the subsequent 8-bit quantization can lead to $1.8\times \sim 2.1\times$ speedups). Based on Eq. (12) and (13), we can derive proper ρ_i and then feed ρ_i to Eq. (14) to calculate the latency-sparsity loss ℓ_{ratio} that participates in the back propagation process. In Eq. (14), M denotes the token keep decision, N denotes the number of tokens, and B denotes the batch size. The average pruning rate of all images in a batch is set as the convergence target in Eq. (14).

Block-to-Stage Training. Algorithm 1 presents our training strategy to find the optimal accuracy and pruning rate trade-offs, a proper number of token selectors and their locations to insert, based on the token redundancy (Fig. 6). Inspired by SPViT [28], we adopt finetuning to insert the token selector from later blocks to front ones progressively. Each time when we insert a token selector, we train the current selector and fine-tune the other parts by decreasing the latency of the current block until accuracy drops noticeably ($> 0.5\%$).

Algorithm 1: Latency-Aware Multi-Stage Training with Image-Adaptive Token Pruning

Input : ViT blocks $\{\mathrm{Layer}_i\}_{i=1}^{L}$;
Accuracy drop constraint a_{drop};
Initial pruning rate ρ_{init};
Target latency L_{target}.
Output : Token selectors with pruning rates $\rho_{s_1}, ..., \rho_{s_k}$.

```
// Step1: Insert a token selector between each two adjacent blocks and adjust the pruning rate ρi.
```

foreach $i \in [L, L-1, \ldots, 4]$ **do**
  $\rho_i = \rho_{\mathrm{init}}$;
  $a, t \leftarrow$ Evaluate($\{\mathrm{Layer}_j(\rho_j)\}_{j=1}^{L}$);
  `// a and t represent accuracy drop and latency of the whole model.`
  while $a < a_{\mathrm{drop}}$ **do**
    if $t < L_{target}$ **then**
      Return the finalized token pruned ViT;
    else
      Decrease t_i;
      ρ_i = latency_sparsity_table(t_i);
      $a, t \leftarrow$ Evaluate($\{\mathrm{Layer}_j(\rho_j)\}_{j=1}^{L}$);

```
// Step2: Combine sequential selectors with similar pruning rates as one stage, keep the first selector and retrain ViT.
```

$\rho_{s_1}, ..., \rho_{s_k} \leftarrow$ Combine $\rho_1, ..., \rho_L$;
Retrain ViT[$\mathrm{Layer}_1(\rho_1), .., \mathrm{Layer}_i(\rho_{s_1}), .., \mathrm{Layer}_L(\rho_{s_k})$];
if $t < L_{target}$ **then**
  Go to the quantization process;
else
  Increase a_{drop} or L_{target};
  Initialize the model and selectors from the end of the last Step1;
  Go to Step 1 and repeat the training process.

Our training algorithm differs from SPViT in three ways. First, since token pruning in the front three ViT blocks leads to more severe accuracy drops, we do not insert token selectors for these three blocks. Second, to ensure that the number of token selectors is minimal, once the pruned model has satisfied the target latency, we end the training and finalize the pruned model. After the progressive training, we combine the neighbor token selectors with a similar pruning rate into one. Third, if the final latency of the whole model is lower than the target latency, we move to the quantization stage; otherwise, we will relax the accuracy or latency constraints and repeat the training.

VII. Experimental Results

A. Experimental Setup

Our experiments include adaptive token pruning and hardware implementation for ViTs with different pruning settings. Note that after token pruning we will apply 8-bit quantization on weight and activation, and all the quantization processes do not lose accuracy, except for 1.2% on DeiT-T.

1) Training Setup for ViT Pruning: The baseline models with 32-bit floating-point precision are from the TorchVision library [12]. Our experiments are conducted on the ImageNet-1K dataset with various transformers backbones, including DeiT-T, DeiT-S, DeiT-B, LV-ViT-S, and LV-ViT-M, as shown in Table V. We follow the training settings in DeiT. Training on one selector insertion costs 30 epochs on 8 NVIDIA A100-SXM4-40GB GPUs. The training effort on block-to-stage training is listed in Table V which illustrates that the training effort of the entire block-to-stage pipeline is equivalent to the train-from-scratch of the backbone ViT. Through our training pipeline, we observe that 3$\sim$4 token selectors are suitable for most of the models.

TABLE V. Training effort for ViTs with different backbones.

Model	#Heads	Embed. Dim.	Depth	#Epochs for Training	
				Baseline	Ours
DeiT-T	3	192	12	300	270
DeiT-S	6	384	12	300	270
DeiT-B	12	768	12	300	270
LV-ViT-S	6	384	16	400	390
LV-ViT-M	8	512	20	400	390

2) Hardware Platform: Our hardware accelerator designs are evaluated on the Xilinx ZCU102 platform [53] with Zynq UltraScale+ MPSoC, containing 2520 DSPs, 912 BRAM blocks, and 274.1k LUTs. We use Vitis HLS and Vitis 2020.1 tool [52] to generate, synthesis, and implement FPGA accelerator design, with 150 MHz operating frequency. The data in all models are represented in an 8-bit fixed-point format. The hardware design for HeatViT is built based on state-of-the-art FPGA design for ViT [32] with the GEMM engine described in Section V-B1. Additionally, the HeatViT hardware design incorporates a token selector and polynomial approximation of nonlinear functions explained in Section V-C and V-D.

B. Accuracy and GMACs Results

Our models outperform other pruned models in terms of accuracy-computation trade-offs, as shown in Fig. 2. Our HeatViT reduces the computation cost by 16.1%$\sim$42.6% for various backbones with negligible $\leq$ 0.75% accuracy degradation, which surpasses existing methods on both accuracy and efficiency. Also, we train more DeiT models with the embedding dimension of 160/256/288/320 as our baselines. The accuracy improvement of HeatViT is 4.05% (72.15% vs. 68.1% with 0.9 GMACs) over DeiT-T-160, 4.74% (76.94% vs. 72.20% with 1.3 GMACs) over DeiT-T, and 0.85% (79.38% vs. 78.53% with 2.64 GMACs) over DeiT-S-288.

TABLE VI. Results on COCO object detection and instance segmentation. FLOPs are computed on 1280 $\times$ 800 image, and backbone FLOPs are reported. AP^{box} denotes box mAP and AP^{mask} denotes mask mAP, both of which are common metrics for the accuracy of object detection.

Backbone	GFLOPs	AP^{box} (%)	AP^{mask} (%)
Swin-T	267	45.9	41.4
HeatSwin-T	227	45.6	41
Swin-S	359	48.5	43.3
HeatSwin-S	306	48.3	42.9

We further evaluate our adaptive pruning method on object detection and instance segmentation. We conduct our experiments on the COCO 2017 dataset [34] with the widely used Mask-RCNN [20]. The Swin-transformer [4] is used as the backbone following the training receipt on semantic segmentation. As shown in Table VI, our HeatViT model is capable of the object detection and instance segmentation tasks, with only a negligible degradation on the final performance after reducing 15% of the FLOPs. The results also indicate that our technique accommodates well to much larger image resolution for the detection task.

C. Hardware Results

Multiple hardware accelerators are designed according to the number of heads in a specific ViT. As shown in Table VII, with the same total degree of computation parallelism, the resource utilization and power of DeiT-S and LV-ViT-S designs are higher than those of DeiT-T ones, since DeiT-T has 3 heads while DeiT-S and LV-ViT-S all have 6 heads, requiring more BRAM space to accommodate data of all the attention heads. This trend is similar for DeiT-B (12 heads).

Compared with the baseline hardware designs (16-bit and no token pruning), the accelerators with token selector in HeatViT framework utilize 9% more DSPs and 8% more LUTs for DeiT-T, 8% more DSPs and 5% more LUTs for DeiT-S and LV-ViT-S, 11% more DSPs and 6% more LUTs for DeiT-B. This demonstrates that the control flow to support adaptive token pruning introduces negligible resource utilization overhead. After the token pruning, the frame rate increases from 78.3 FPS to 142.7 FPS (1.82$\times$) for DeiT-T, from 25.9 FPS to 57.6 FPS (2.22$\times$) for DeiT-S, from 19.4 FPS to 46.9 FPS (2.42$\times$) for LV-ViT-S, and from 11.2 FPS to 28.9 FPS (2.58$\times$) for DeiT-B. Furthermore, we deploy the 8-bit fixed-point quantization on models to achieve another 1.90$\times$ speedup, ending up the final speedup with 3.46$\times$ (271.2 FPS) for DeiT-T, 4.22$\times$ (109.2 FPS) for DeiT-S, 4.59$\times$ (89.1 FPS) for LV-ViT-S, and 4.89$\times$ (54.8 FPS) for the DeiT-B.

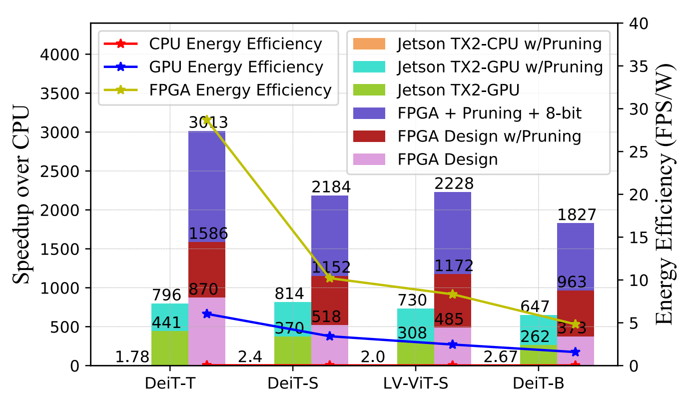


Fig. 12. Comparison of energy efficiency between HeatViT and TX2 CPU/GPU with the improvement breakdown of different techniques.

TABLE VII. Hardware results under different pruning settings for various ViTs on ImageNet dataset.

Design	Model	Keep Ratio (Stage 1/2/3)	#GMACs (Pruning Rate)	Bitwidth	Resource Utilization kLUT	kFF	BRAM36	DSP	Power (W)	FPS (Accl. Rate)	Energy Effi. (FPS/W)
Baseline	DeiT-T	1/1/1	1.30 (1×)	16	115.6 (42%)	101.5 (19%)	288.5 (32%)	1739 (69%)	8.012	78.3 (1×)	9.77
	DeiT-S	1/1/1	4.60 (1×)	16	130.3 (48%)	102.8 (19%)	492.5 (54%)	1754 (70%)	10.095	25.9 (1×)	2.57
	LV-ViT-S	1/1/1	6.55 (1×)	16						19.4 (1×)	1.92
	DeiT-B	1/1/1	17.60 (1×)	16	144.5 (53%)	103.9 (19%)	664.3 (73%)	1786 (71%)	11.041	11.2 (1×)	1.01
HeatViT with Token Selector	DeiT-T	0.85/0.79/0.51	1.00 (1.30×)	8	137.6 (50%)	126 (23%)	355.5 (39%)	1968 (78%)	9.453	183.4 (2.34×)	19.4
		0.76/0.70/0.41	0.90 (1.44×)	8						198.8 (2.54×)	21.0
		0.70/0.39/0.21	0.75 (1.74×)	8						271.2 (3.46×)	28.7
	DeiT-S	0.90/0.84/0.61	3.86 (1.19×)	8	145 (53%)	100.4 (18%)	338.5 (37%)	1955 (78%)	10.697	57.0 (2.20×)	5.33
		0.70/0.39/0.21	2.64 (1.74×)	8						97.1 (3.75×)	9.08
		0.42/0.21/0.13	2.02 (2.27×)	8						109.2 (4.22×)	10.2
	LV-ViT-S	0.90/0.84/0.61	5.49 (1.19×)	8						62.8 (3.24×)	5.87
		0.70/0.39/0.21	3.77 (1.74×)	8						72.8 (3.75×)	6.81
		0.42/0.21/0.13	2.88 (2.27×)	8						89.1 (4.59×)	8.33
	DeiT-B	0.90/0.84/0.61	14.79 (1.19×)	8	161.4 (59%)	101.8 (19%)	528.6 (58%)	2066 (82%)	11.352	36.1 (3.22×)	3.18
		0.70/0.39/0.21	10.11 (1.74×)	8						43.3 (3.87×)	3.81
		0.42/0.21/0.13	7.75 (2.27×)	8						54.8 (4.89×)	4.83

1) Comparisons with CPUs and GPUs: We also test DeiT-T, DeiT-S, LV-ViT-S, and DeiT-B on Jetson TX2 with 4-core ARM CPU and NVIDIA Pascal GPU, and compared them with our FPGA (ZCU102) implementation. Since MSA and FFN computations are reduced by token pruning, CPUs and GPUs can also be accelerated. And TX2 CPU/GPU does not support low-bit computation, so we only present the full precision model for them with adaptive token pruning as shown in Fig. 12. All the results are normalized against the original model on TX2 CPU without token pruning. First, our final FPGA implementation achieves the highest $1827\times \sim 3013\times$ speedup with 9.453W, 10.697W, and 11.352W power for different designs. While the baseline FPGA design [32] (16-bit and no token pruning) achieves $373\times\sim870\times$ speedup, token pruning can bring $1.82\times\sim2.58\times$ speedup and ambitious 8-bit quantization can contribute another $1.90\times$ speedup. Second, with token pruning, TX2 GPU achieves $647\times\sim814\times$ speedup with 12W power and TX2 CPU achieves $1.78\times\sim2.67\times$ speedup with 4W power. For the energy efficiency, our FPGA implementation achieves 4.8 FPS/W$\sim$28.7 FPS/W, which is $242.6\times\sim719.0\times$ higher than TX2 CPU and $3.0\times\sim4.7\times$ higher than TX2 GPU (with token pruning).

VIII. Conclusion

In this paper, we have proposed a hardware-efficient image-adaptive token pruning framework called HeatViT for ViT inference acceleration on resource-constraint edge devices. To improve the pruning rate and accuracy, we first adopted an effective and hardware-efficient token selector that can more accurately classify tokens and consolidates non-informative tokens. We also implemented a proof-of-concept ViT hardware accelerator on FPGAs by heavily reusing the hardware components built for the backbone ViT to support the adaptive token pruning module. Besides, we propose a polynomial approximation of nonlinear functions for ambitious (8-bit) quantization and efficient hardware implementation. Finally, to meet both the target inference latency and model accuracy, we applied a latency-aware multi-stage training strategy to learn the number of token selectors to insert into the backbone ViT, and the location and pruning rate of each token selector. Experimental results show that HeatViT achieves superior pruning rate and accuracy compared to state-of-the-art pruning studies while incurring a trivial amount of hardware resource overhead.

Acknowledgements

This work is partly supported by NSF CCF-1901378; NSERC Discovery Grant RGPIN-2019-04613, DGECR-2019-00120, Alliance Grant ALLRP-552042-2020; CFI John R. Evans Leaders Fund.

References

[1] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[2] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations (ICLR)*, 2015.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[4] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, "Swin-unet: Unet-like pure transformer for medical image segmentation," *arXiv preprint arXiv:2105.05537*, 2021.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 213–229.

[6] B. Chen, P. Li, C. Li, B. Li, L. Bai, C. Lin, M. Sun, J. Yan, and W. Ouyang, "Glit: Neural architecture search for global and local image transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12–21.

[7] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao, "Pre-trained image processing transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 12 299–12 310.

[8] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 12 270–12 280.

[9] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao, and H. Ling, "Searching the search space of vision transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 8714–8726.

[10] T. Chen, Y. Cheng, Z. Gan, L. Yuan, L. Zhang, and Z. Wang, "Chasing sparsity in vision transformers: An end-to-end exploration," in *Advances in Neural Information Processing Systems*, 2021.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[12] Facebook, "Torchvision," 2021, last accessed Sept 12, 2021. [Online]. Available: https://pytorch.org/vision/stable/models.html

[13] M. Fayyaz, S. A. Kouhpayegani, F. R. Jafari, E. Sommerlade, H. R. V. Joze, H. Pirsiavash, and J. Gall, "Ats: Adaptive token sampling for efficient vision transformers," *arXiv preprint arXiv:2111.15667*, 2021.

[14] S. Fox, J. Faraone, D. Boland, K. Vissers, and P. H. Leong, "Training deep neural networks in low-precision with high accuracy using fpgas," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 1–9.

[15] S. Fox, J. Faraone, D. Boland, K. Vissers, and P. H. Leong, "Training deep neural networks in low-precision with high accuracy using fpgas," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 1–9.

[16] C. Gong, D. Wang, M. Li, X. Chen, Z. Yan, Y. Tian, and V. Chandra, "Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training," in *International Conference on Learning Representations (ICLR)*, 2021.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[18] Y. Guo, Y. Zheng, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang, "Nat: Neural architecture transformer for accurate and compact architectures," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[19] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," in *Advances in Neural Information Processing Systems*, 2021.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[21] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[22] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[23] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[24] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.

[25] Z. Jiang, Q. Hou, L. Yuan, D. Zhou, Y. Shi, X. Jin, A. Wang, and J. Feng, "All tokens matter: Token labeling for training better vision transformers," *arXiv preprint arXiv:2104.10858*, 2021.

[26] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[27] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," *arXiv preprint arXiv:2101.01321*, 2021.

[28] Z. Kong, P. Dong, X. Ma, X. Meng, W. Niu, M. Sun, X. Shen, G. Yuan, B. Ren, H. Tang, M. Qin, and Y. Wang, "Spvit: Enabling faster vision transformers via latency-aware soft token pruning," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, 2022, p. 620–640.

[29] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 3519–3529.

[30] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 175–180.

[31] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, "Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 12 281–12 291.

[32] Z. Li, M. Sun, A. Lu, H. Ma, G. Yuan, Y. Xie, H. Tang, Y. Li, M. Leeser, Z. Wang, X. Lin, and Z. Fang, "Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization," in *International Conference on Field Programmable Logic and Applications*. Springer, 2022, pp. 289–300.

[33] Y. Liang, C. GE, Z. Tong, Y. Song, J. Wang, and P. Xie, "EVit: Expediting vision transformers via token reorganizations," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=BjyvwnXXVn_

[34] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[35] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 28 092–28 103.

[36] M. Mao, R. Zhang, H. Zheng, P. Gao, T. Ma, Y. Peng, E. Ding, and S. Han, "Dual-stream network for visual recognition," in *Advances in Neural Information Processing Systems*, 2021.

[37] T. M. Mitchell, "Machine learning and data mining," *Communications of the ACM*, vol. 42, no. 11, pp. 30–36, 1999.

[38] B. Pan, Y. Jiang, R. Panda, Z. Wang, R. Feris, and A. Oliva, "Ia-red^2: Interpretability-aware redundancy reduction for vision transformers," in *Advances in Neural Information Processing Systems*, 2021.

[39] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2249–2255.

[40] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully quantized transformer for machine translation," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1–14.

[41] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, "Accommodating transformer onto fpga: Coupling the balanced model compression and fpga-implementation optimization," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 163–168.

[42] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "Dynamicvit: Efficient vision transformers with dynamic token sparsification," in *Advances in Neural Information Processing Systems*, 2021.

[43] D. So, Q. Le, and C. Liang, "The evolved transformer," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 5877–5886.

[44] M. Sun, Z. Li, A. Lu, Y. Li, S.-E. Chang, X. Ma, X. Lin, and Z. Fang, "Film-qnn: Efficient fpga acceleration of deep neural networks with intra-layer, mixed-precision quantization," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022, p. 134–145.

[45] Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, and D. Tao, "Patch slimming for efficient vision transformers," 2021.

[46] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, 2021, pp. 10 347–10 357.

[47] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware implementation of sigmoid activation functions using fpga," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019, pp. 34–38.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[49] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020, pp. 7675–7688.

[50] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for

dense prediction without convolutions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[51] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "Cvt: Introducing convolutions to vision transformers," *arXiv preprint arXiv:2103.15808*, 2021.

[52] Xilinx, "Vitis unified software platform," 2022, last accessed April 21, 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html#development

[53] Xilinx, "Zcu102 evaluation board - user guide," 2022, last accessed April 21, 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf

[54] Y. Xu, Z. Zhang, M. Zhang, K. Sheng, K. Li, W. Dong, L. Zhang, C. Xu, and X. Sun, "Evo-vit: Slow-fast token evolution for dynamic vision transformer," *arXiv preprint arXiv:2108.01390*, 2021.

[55] C. Yang, Z. Wu, B. Zhou, and S. Lin, "Instance localization for self-supervised detection pretraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 3987–3996.

[56] H. Yu and J. Wu, "A unified pruning framework for vision transformers," *arXiv preprint arXiv:2111.15127*, 2021.

[57] S. Yu, T. Chen, J. Shen, H. Yuan, J. Tan, S. Yang, J. Liu, and Z. Wang, "Unified visual transformer compression," in *International Conference on Learning Representations*, 2022.

[58] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 558–567.

[59] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.

[60] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on fpga devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[61] Z. Zhao, Y. Liu, L. Chen, Q. Liu, R. Ma, and K. Yu, "An investigation on different underlying quantization schemes for pre-trained language models," in *CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, 2020, pp. 359–371.

[62] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. Torr *et al.*, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6881–6890.

[63] D. Zhou, Y. Shi, B. Kang, W. Yu, Z. Jiang, Y. Li, X. Jin, Q. Hou, and J. Feng, "Refiner: Refining self-attention for vision transformers," 2021.

[64] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong, "End-to-end dense video captioning with masked transformer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8739–8748.

[65] M. Zhu, K. Han, Y. Tang, and Y. Wang, "Visual transformer pruning," in *KDD 2021 Workshop on Model Mining*, 2021.

[66] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," in *International Conference on Learning Representations (ICLR)*, 2020.