UWYN (USE ONLY WHAT YOU NEED): EFFICIENT INFERENCING IN 2D AND 3D VISION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient computation for Vision Transformers (ViTs) is critical for latency-sensitive applications. However, early-exit schemes rely on auxiliary controllers that introduce non-trivial overhead. We propose UWYN, an end-to-end framework for image classification and shape classification tasks that embeds exit decisions directly within the transformer by reusing the classification head at each residual block. UWYN first partitions inputs via a lightweight feature-threshold into "simple" and "complex" samples: simple samples are routed to a shallow ResNet branch, while complex samples traverse the ViT and terminate as soon as their per-block confidence exceeds a preset confidence level. During the ViT pass, UWYN also dynamically prunes redundant patch embeddings and attention heads to further cut computation. We implement and evaluate this strategy on both 2D (ImageNet,CIFAR-10, CIFAR-100, SVHN, BloodMNIST) and 3D (ModelNet-40, Scan Object NN) benchmarks. UWYN reduces Multiply-Accumulate operations (MACs) by over 75% compared to SOTA models, e.g., LGViT [ACM MM '23], achieving 83.29% accuracy on CIFAR-100 and 84.39% on ImageNet. We also show faster inference with minimal accuracy loss.

1 INTRODUCTION

Vision Transformer (ViT) (23) architectures have emerged as powerful tools for a wide range of computer vision tasks. However, their high computational demands present challenges, especially in resource-constrained environments like mobile and embedded systems. These systems, equipped with powerful SoCs (Systems-on-Chips) and heterogeneous processing units, require models that balance accuracy with efficiency to meet real-time processing needs. Despite significant advances in accuracy, traditional ViT models often overlook the resource constraints, such as limited processing power, memory, and energy, faced by such devices. For instance, mobile GPUs typically have limited memory, ranging from 2-8 GB, which is far below the requirements of large-scale transformer models. To illustrate, let us consider a state-of-the-art transformer model like MegaViT (11), which has 22 billion (22B) parameters. Since each parameter is typically stored as a 4 byte floating-point value, storing the model's parameters alone requires approximately 88 GB of memory. This large memory requirement renders models like MegaViT impractical for most GPUs or TPUs. Even high-end GPUs with 40-80 GB struggle to fit such models without optimizations like model parallelism, sharding, or offloading model segments to disk. Smaller network architectures like ResNet (15) rely on local convolutions, which limits their ability to capture long-range dependencies. In contrast, transformers address this by using self-attention mechanisms that model global relationships. While both ViTs and ResNets have demonstrated impressive capabilities, there is increasing interest in using early exits to improve efficiency and model interpretability. Recent work has focused on training smaller networks and applying algorithmic approximations to tailor transformer-based models for embedded devices (8; 66). While these approaches have made strides in reducing model size and computational demands, they often compromise accuracy or limit

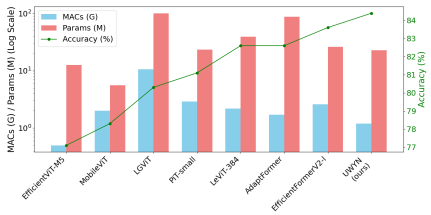


Figure 1: Computational complexity (MACs), model size (parameters), and ImageNet accuracy across different ViT methods (36; 40; 66; 16; 14; 8; 30). UWYN shows a substantial reduction in MACs with only a minimal dip in accuracy.

the model's full potential. This trade-off highlights the need for solutions that optimize ViTs for deployment in real-world, resource-constrained environments — solutions that can significantly reduce computational requirements without drastically reducing performance.

This forms our problem statement: *How can a ViT dynamically allocate computation based on input complexity?* Our broad solution approach is to approximate the execution based on content complexity, allowing the model to dynamically adapt its processing to the demands of the data. Consider the ViT (23) architecture, a widely adopted transformer model comprising 12 blocks, applied here in the context of image classification task on the ImageNet dataset (12; 49; 23). To quantify the depth–efficiency trade-off, we systematically drop the final k blocks from a 12-layer ViT and record end-to-end inference time versus classification accuracy, as shown in Table 1. Similar trends have been observed in NLP problems (Appendix A.8). These findings suggest a trade-off between accuracy and efficiency, a balance that could be leveraged for applications where rapid inference is critical or resources are constrained. This approach raises the intriguing possibility of leveraging intermediate outputs for early exits, enabling the model to make informed decisions on when to terminate processing based on content complexity.

Table 1: **Motivation:** trade-off between inference time and accuracy on ImageNet when dropping the final k of the 12 ViT blocks.

Number of Blocks Dropped	Inference Time (s)	Accuracy (%)
0	287	84.8
1	264	82.5
2	242	65.2
3	217	34.6
4	197	12.0

The pipeline begins by classifying inputs by complexity: simple data goes to a shallow ResNet, complex data to a novel early-exit Transformer. Our model adds a layer within each Transformer block to leverage intermediate features, enabling context-aware, efficient processing by selecting key patches and attention heads. Embeddings from each block feed into a shared classifier (Figure 2). After each block, token embeddings are classified; if a dominant class exceeds a threshold, we exit early with that prediction. We evaluate using accuracy, MACs, and parameters (Figure 1). Existing adaptive ViTs like EfficientViT (36) implements Cascaded Group Attention to perform faster inference, while methods like EfficientFormer (30) uses meta blocks for efficient token mixing and reduce computation cost. Various state-of-the-art methods inflate models with extra networks requiring separate training. UWYN minimizes overhead by selecting key features within a block and using a shared, light layer for early exits across all blocks, reducing parameters and MACs. Our gains have been demonstrated in two distinct tasks, 2D image classification and 3D shape classification. We achieve orders of magnitude gains in MACs relative to ViT (23), which is an exact, non-approximated model with the highest accuracy. We reduce MACs from 1693G to 1.2G and the number of parameters by 3.8×, from 86M to 22.86M. Our results show that UWYN achieves competitive accuracy with significantly reduced inference time, establishing it as a practical and efficient inferencing solution for real-world applications on resource-constrained devices. The main contributions of our paper are:

1. **Assessing feature relevance within blocks for adaptive early exits:** We integrate additional layers inside transformer blocks to assess feature relevance in real-time, enabling early exit *without* auxiliary networks, which simplifies training.
2. **Reusing a unified classifier with confidence-based early exits:** We use a single classifier across all transformer blocks, reducing the need for separate classifiers and simplifying the model architecture. Further, we implement a confidence score-based mechanism that continuously evaluates prediction certainty. As the model processes each block, it calculates dynamic confidence scores to determine optimal exit points.
3. **Performance gains across diverse hardware architectures:** We demonstrate inference speed up in a variety of architectures like a server-class P100 GPU and two edge devices, AGX Xavier and Jetson Orin.

2 Related Work

Transformer Efficiency Challenges and Architectural Solutions: Transformers (23; 58) have achieved strong performance in tasks like image classification but face challenges with latency and resource demands due to large model sizes and reliance on extensive datasets (47; 28; 27; 22). These limitations have driven research into adapting transformers for mobile and resource-limited environments (30; 63; 71). Several architectural modifications have been proposed to address

efficiency constraints. For instance, Efficient ViT (36) introduces cascaded group attention blocks to process features selectively, avoiding full-length processing and reducing computational load. Efficient-Former uses a dual path attention mechanism to reduce inference time (30). Similarly, MobileViT (40) separates local and global feature processing with CNNs before integration, and AdaViT (42) uses a decision network in each block to focus on essential parts of prior outputs, optimizing efficiency. This adds an overhead to each transformer block, thereby making the overall process heavyweight. In addition to architectural changes, data flow optimization has been explored to improve transformer training efficiency. DeiT (54) integrates a distillation token and pre-trained CNN teacher to guide learning. PiT (16) uses depth-wise convolution to achieve channel multiplication and spatial reduction with small parameters. In video tasks, SparseVit (9), a variant of Swin Transformer (38), prioritizes high-relevance windows based on L2 norms, improving execution efficiency in self-attention mechanisms. However, by separating local and global attention, the number of parameters is increased significantly. Other orthogonal efforts have been made to efficiently handle memory processing for hardware, such as Ring Attention (33) and Flash Attention (10). However, these need powerful tensor cores, which makes it infeasible for mobile devices like AGX-class of mobile GPUs and even earlier server-class GPUs like Tesla P100. In addition, the software dependencies for these methods have a conflict based on their current CUDA versions and JetPack packages available. In contrast to all works in this category, UWYN demonstrates reduction in inference time over a wide variety of mobile devices.

Pre-processing hinders Efficiency in 3D Shape Classification: For point cloud data, methods like PointNet (44) and PointNet++ (45) introduce symmetry and hierarchical clustering for direct, permutation-invariant processing. While transformers improve generalization and long-range dependency capture in point cloud tasks (72), these models often depend on extensive pre-processing, which can reduce inference speed and efficiency, particularly in complex or masked scenes (57). In contrast, UWYN identifies how simple and complex data instances can be processed separately, thereby making the overall pipeline more efficient. We add a pre-processing step for 3D data, which accelerates the pipeline much more compared to the existing methods. These pre-processing steps, combined with early exit, make the end-to-end inference faster.

Early Exits and Lightweight Networks in Transformers: To reduce processing time, early exit techniques in transformers and deep neural networks assess intermediate results, ending processing when confidence thresholds are met. Methods like those by Teerapittayanon *et al.* (52) and Wolczyk *et al.* (61) incorporated a small network for each network block or pathway to decide when to early-exit. LGViT (66) achieves this by combining local and global perception heads, while models like FastBERT (35) and BERxiT (65) add lightweight networks within BERT layers for early exits. However, these approaches generally increase computational complexity by requiring additional networks or classifiers. Techniques like Neural Architecture Search (NAS) (31; 74; 69) implement reinforcement learning to decide when to early exit or to choose a pathway that streamlines computations.

These methods add a significant computational overhead due to the addition of separate networks at regular intervals of the backbone network. We quantitatively compare the efficiency and the accuracy of UWYN against several of these prior works, ViT (23), EfficientViT (36), LeViT (14), EfficientFormer (30), PiT (16), MobileViT (40), AdaptFormer (8), and LGViT (66) in Table 2.

3 METHODS

As shown in Table 1, reducing the number of blocks in the ViT architecture results in substantial computational savings with only a slight decrease in accuracy. For instance, using 90% of the model maintains classification accuracy while achieving a speedup of approximately 20 seconds in inference time (10% less time). This observation provides the insight that the full model is not necessary for confidently classifying all data instances. To leverage this insight, we propose a systematic approach for accelerating inference by categorizing data based on its complexity and adapting processing accordingly.

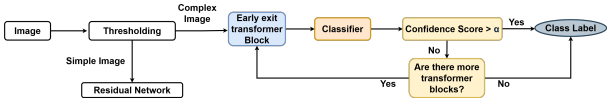


Figure 3: **Flowchart for image classification using UWYN:** A dual-path strategy for classifying images based on their complexity.

Our proposed pipeline, illustrated in Figure 3, divides input data into two

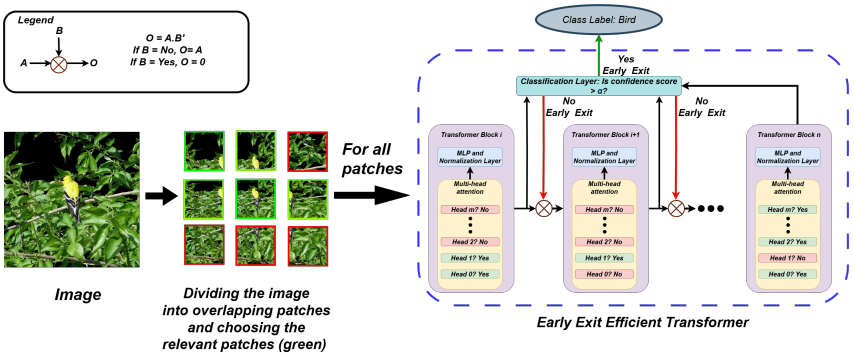


Figure 2: **Architecture of Early-Exit Efficient Vision Transformer:** The data is segmented into patches by the Soft Split Network, analyzed by the Patch and Attention Head Selector to identify key embeddings and attention heads. Each modified transformer block includes normalization, multi-head attention, MLPs, and a mechanism for selecting relevant attention heads. Embeddings are evaluated at a classifier layer; if the confidence score exceeds α or the final block is reached, classification is returned early. Otherwise, processing continues to the next transformer block, reducing redundant computations.

categories: A) simple data, which can be accurately classified with a simple network like ResNet and B) complex data, which requires heavy computations to be performed by an efficient early-exit ViT. We first demonstrate this approach on 2D image classification and then extend it to a 3D task with only minimal modifications to the pre-processing stage.

3.1 Thresholding

Data samples with a complexity value lesser than a threshold are passed through the simple network, while the rest are passed through the complex pathway.
For 2D: We classify data samples as simple or complex using the Sobel operator (detailed calculations in Appendix Section A.1), which computes the gradient magnitude by convolving the image with two 3×3 kernels for detecting horizontal (G_x) and vertical edges (G_y). The gradient magnitude is:

$$\text{Magnitude}(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2}. \tag{1}$$

Using an empirically set threshold (Section 4.1), we classify images in the top $2/3^{\text{rd}}$ of gradient magnitudes as complex, while the remainder are considered simple. Simple images have fewer edges and are easier to classify, indicating less structural detail, while complex images contain richer edge information. Figure 7 in Appendix (Section A.2) shows an example of ImageNet (12), illustrating this classification. The Sobel-based method is efficient as it leverages fast gradient calculations to categorize images by structural complexity, making it ideal for resource-constrained environments.
For 3D: The data is expressed as point clouds, consisting of multiple 3D points in (x, y, z) coordinates that define the shape under consideration. We determine complexity based on the density and spatial distribution of these point clouds. Using a fixed sample of 1,024 points, we calculate the volume they occupy. Shapes occupying a smaller volume (within the lower 70th percentile) are classified as complex, while those occupying a larger volume are considered simple. The threshold for this classification is determined empirically.

3.2 Simple Network

For 2D We use ResNet50 (15) to process the simpler data points in the dataset. The network is trained from scratch with drop path regularization (19), which enhances robustness and generalization. This approach improves pipeline efficiency by enabling faster processing of simpler data points with reduced computational resources.
For 3D The point cloud consists of 3D points. We use an architecture similar to ResNet50 (15) , where each point (x, y, z) is treated as a 3-channel input. We implement a network consisting of 3,4,4,3 residual blocks, in sequential order, to efficiently process simpler 3D data.

3.3 EARLY-EXIT EFFICIENT TRANSFORMER

For complex data instances, UWYN incorporates an early exit mechanism within the Vision Transformer, inspired by efficient attention, to speed up processing of single-object images with redundant patches (Figure 2). After the Soft Split Network creates overlapping patches, selected key patches are processed by transformer blocks in parallel. Each block's embeddings for all the selected patches are passed through a unified classifier layer, and if a confidence threshold is met or the final block is reached, the process exits with a classification result.

3.3.1 ADDITIONAL PRE-PROCESSING FOR 3D

To render the 3D points, we preprocess them by first centering and scaling the coordinates (x, y, z). A virtual camera is defined with a specific position and orientation in the 3D space, using the default perspective projection for depth. This means the depth, represented by the z-coordinate, affects the projection onto the 2D screen, with points farther from the camera appearing smaller due to this perspective effect. To calculate the 2D screen coordinates (x', y') for each point $P(x, y, z)$, we use:

$$x' = \frac{x}{z} \cdot d, \quad y' = \frac{y}{z} \cdot d \tag{2}$$

where d is the distance from the camera to the projection plane. This scaling ensures that the perspective effect is maintained, with distant points shrinking as their z-coordinate increases, effectively transforming the 3D coordinates into 2D with realistic depth perception.

3.3.2 EXTRACTION OF PATCHES

We use a Soft Split Network (SSN) (70) that segments the input $\mathbf{X} \in \mathbb{R}^{h \times w \times c}$ into *overlapping patches*, emulating a convolution operation with receptive fields, to divide the input image into overlapping patches. This segmentation captures local structural details while preserving continuity and contextual relationships between adjacent patches, unlike traditional transformers with disjoint patches. The patches, of size $t \times t$, overlap by o pixels and may include p pixels of padding, similar to a convolution with stride $t - o$. This configuration maintains token coherence while integrating pixel- and patch-level details. After flattening the patches, the data is transformed into a format compatible with subsequent transformer blocks. The number of generated tokens, N, is computed as $N = \left\lfloor \frac{h+2p-t}{t-o} + 1 \right\rfloor \times \left\lfloor \frac{w+2p-t}{t-o} + 1 \right\rfloor$, where each token is a flattened vector of size $d = ct^2$.

The Soft Split Network (SSN) output, $\mathbf{X}_{\text{SSN}} \in \mathbb{R}^{N \times d}$, is then prepended with a classification token [CLS], $\mathbf{X}_{\text{cls}}$, to form the final embedding input, $\mathbf{X}_{\text{emb}} = [\mathbf{X}_{\text{CLS}}; \mathbf{X}_{\text{SSN}}], \quad \mathbf{X}_{\text{emb}} \in \mathbb{R}^{(N+1) \times d}$.

3.3.3 PATCH SELECTOR AND ATTENTION HEAD SELECTOR NETWORK

To improve computation efficiency (Figure 2), we introduce a Patch Selector before the first transformer block, and an Attention Head Selector in each of the transformer blocks. These selectors filter out the most relevant patches and attention heads for further processing in the corresponding block. Computational savings occur because non-selected heads and initial patches are effectively zeroed out during calculations.

Each of the selectors has a linear layer followed by a sigmoid activation. Intuitively, a patch is selected if it is likely to have some part of the object that we are trying to classify. Mathematically, the Patch Selector has the following form:

$$\mathbf{S}_{\text{patch}} = \sigma(W_{\text{patch}} \cdot \text{vec}(\mathbf{X}_{\text{SSN}})), \quad W_{\text{patch}} \in \mathbb{R}^{N \times Nd} \tag{3}$$

where $\cdot$ is matrix multiplication, $\text{vec}(\mathbf{X}_{\text{SSN}}) \in \mathbb{R}^{(N)d}$ is the flattened version of $\mathbf{X}_{\text{SSN}}$, and $\sigma(\cdot)$ is the *sigmoid* function which is applied element-wise. All of the patches are processed concurrently. We set a threshold of 0.5 for patches to be selected. To maintain the dimension of the inputs of the various stages, we set the tokens corresponding to the non-essential patches to be 0. This masking approach skips computations, when it encounters a 0 in the selected vector.

A tensor $X_{\text{emb}} = [\mathbf{X}_{\text{CLS}}; \mathbf{X}_{\text{SSN}}] \in \mathbb{R}^{(N+1) \times d}$ is passed to the first transformer block as input. Thus, each patch generates one token plus there is the [CLS] token, which together constitute the input to

the transformer component. The Attention Head Selector has a similar architecture:

$$\mathbf{S}^{(l)}_{\text{AttHead}} = \sigma(W^{(l)}_{\text{AttHead}} \cdot \text{vec}(\mathbf{X}^{(l-1)}_{\text{emb}})), \quad W^{(l)}_{\text{patch}} \in \mathbb{R}^{H\times(N+1)d'}, \mathbf{X}^{(l-1)}_{\text{emb}} \in \mathbb{R}^{(N+1)\times d'}. \tag{4}$$

With a threshold of 0.7 (determined in Figure 6), high-impact attention heads are selected to form the new features $\mathbf{X}^{(l-1)}_{\text{emb}}$ (notation for block $l-1$) to be processed by a the next block (block l in this case). High-impact heads intuitively extract features of the image that have a bearing on the ultimate classification result.

Parallelization of Attention Calculation - Efficient calculation of Query and Keys: Standard self-attention (e.g., PyTorch (4)) computes query $\mathbf{Q}$ and key $\mathbf{K}$ separately via matrix multiplications with input $\mathbf{X}$, requiring two fetches of $\mathbf{X}$ and sequential computation, increasing memory access overhead and inference time due to GPU memory bandwidth limits and serialization. . UWYN computes $\mathbf{Q}$ and $\mathbf{K}$ simultaneously in one inference pass. We concatenate trained weight matrices $\mathbf{W_Q}$ and $\mathbf{W_K}$ into $\mathbf{W} = [\mathbf{W_Q}, \mathbf{W_K}] \in \mathbb{R}^{d\times 2d}$. Then, $\mathbf{XW} = [\mathbf{Q}, \mathbf{K}]$ is computed. This reduces inference time by fetching $\mathbf{X}$ once and parallelizing $\mathbf{Q}$ and $\mathbf{K}$ computation. This does not cause any change to the training process as the attention values are not affected by this optimization. The speedup for this parallelization has been demonstrated in Section A.3 in the Appendix.

3.3.4 Confidence Score and Early-Exit

We introduce two techniques to speed up processing within the transformer: *patch-wise prediction* and *early exit mechanism*.

Patch-wise prediction.. For image and object classification datasets, where each image prominently contains a single object (such as ImageNet (12), CIFAR-10 (25), and CIFAR-100 (24) for 2D and ModelNet40 (64) and ScanObjNN (57) for 3D), the division of images into patches generally results in most patches reflecting the same object class. To exploit this characteristic, we extend beyond the conventional reliance on the single [CLS] token for classification, and propose prediction based on *all* the patches. Specifically, with the output of a transformer block l, i.e., the embedding features $\mathbf{X}^{(l)}_{\text{emb}}$, we have a universal classifier applied onto each patch embedding:

$$p_{\text{i}} = \arg\max(\text{softmax}(W \cdot \mathbf{X}^{(l)}_{\text{emb},i})), \quad W \in \mathbb{R}^{C\times d}, \forall i = 0, 1, \cdots, N. \tag{5}$$

In this paper, we implement a one-layer classifier. Among the $N+1$ patch predictions p_{i}, the most frequently predicted class is marked as the image prediction $p = mode(p_{\text{i}})$. We compute a *confidence score* $\alpha = population(p)/(N+1)$, which is the ratio of the votes for p.

Early exit mechanism.. After each transformer block, we process the embedding features $\mathbf{X}^{(l)}_{\text{emb}}$ using the universal classifier (Eq. 5), and the patch-wise prediction strategy for classification. When a confidence score α surpasses a predefined threshold $\alpha_{\text{threshold}}$, indicating a strong consensus among the patch predictions, an early exit of the transformer model is executed: the prediction based on the intermediate embedding features is returned as the final output, and no further computation by subsequent transformer blocks is executed, hence saving computation. Note that this classifier, including weights, is shared by all tokens and all transformer blocks. By reusing the same classifier, we reduce the training overhead; by keeping our classifier lightweight, we reduce the inference time. This training time and inference time advantage over prior works like EfficientViT (36) and EfficientFormer (30) is demonstrated in Tables 2 and 3.

4 Results

We evaluate UWYN on 2D and 3D classification tasks, comparing its performance with state-of-the-art models in terms of accuracy, inference time, and computational efficiency.

Datasets and training: We evaluate our pipeline on ImageNet (12), CIFAR-10 (25), and CIFAR-100 (24). We also show our results on two unconventional datasets, BloodMNIST (6; 67)—a low resolution medical image processing dataset, and SVHN (43)—an image dataset, which includes potential distractors. For 3D shape classification, we have used ModelNet-40 (64) and Scan Object NN (57), where the latter includes distractions beyond the intended shape. We employ the Adam

Table 2: **ImageNet classification accuracy, MACs, parameters, and inference time on Xavier, P100, and Orin:** UWYN achieves the best accuracy (84.39 %) with only 1.2 G MACs and 22.9 M parameters—over 90× fewer MACs than vanilla ViT—and runs up to 2.3× faster than EfficientFormer (30). (see Table 7 in Appendix for other model variants).

Method	MACs	Params	Accuracy	Time (Xavier)	Time (Orin)	Time (P100)
ViT-Base (23; 56) (non-approximated model)	1693 G	86 M	84.8% (49)	4960 s	4482 s	1089.52 s
EfficientViT-M5 (36)	**0.5 G**	12.57 M	77.1%	3142.23 s	3246.74 s	792.63 s
LeViT-384 (14; 59)	2.2 G	39.11 M	82.6%	1842.81 s	1934.33 s	511.8 s
EfficientFormerV2-l (29; 59)	2.6 G	26.3 M	83.6%	4162 s	4347.74 s	1091.99 s
PiT-small (16)	2.9 G	23.5 M	81.1%	1976.98 s	1919.21 s	502.6 s
MobileViT (40)	2.0 G	**5.6 M**	78.3%	1809.65 s	1747.32 s	497.74 s
MobileViT-v2 (41)	2.0 G	10.6 M	80.4%	1801.65 s	1733.49 s	492.86 s
AdaptFormer (8)	1.71 G	87.6 M	82.6%	3256.45 s	3129.56 s	719.82 s
LGViT (66)	10.65 G	101 M	80.3%	2001.9 s	1984.5 s	504.89 s
PartialFormer (**?**)	3.4 G	64 M	83.9%	1955 s	1864 s	498.42 s
UWYN	1.2 G	22.86 M	**84.39%**	**1796 s**	**1695.23 s**	**492.63 s**

optimizer (21) with an initial learning rate (with cosine annealing) of 0.1, β = (0.9, 0.99), decayed by (1e-4) for the first 100 epochs, followed by a reduced rate of (2e-5), till we reached convergence.

For our Soft Split Network (Section 3.3.2), we use 3 convolution layers with kernel sizes (7, 3, 3), stride (4, 2, 2) and padding (2, 1, 1) along each dimension, respectively. The depth (number of channels) of the output tensor depends on the number of filters (output channels) used in the convolution layer. Based on these calculations, we divide an image of size 224 × 224 into 14 × 14 patches. The Early-Exit Efficient Transformer has 12 blocks total, each block having 6 attention heads. The hidden dimension for ImageNet, ScanObjNN, and ModelNet40 is 768, while for other datasets, it is 384.

Evaluation metrics: We evaluate our pipeline by running it five times on various devices, with no other tasks running concurrently and report the average results. The metrics we use for evaluation include image or shape classification accuracy (top-1 accuracy), average MACs (Multiply-Accumulate operations), number of Parameters (Params), and the total inference time for the entire test set of the dataset. We use the python `thop` library to calculate MACs and Params. The variation in inference time across the five runs is at most 7%. We report inference time on a server-class GPU, Tesla P100 GPU, and resource-constrained edge devices - NVIDIA AGX Xavier and NVIDIA Jetson Orin.

4.1 2D RESULTS

We focus on single object classification tasks. Tables 2, 3, and 7 (Appendix) show how UWYN performs with respect to other state-of-the-art algorithms. We choose ViT as the baseline for its non-approximated structure and highest reported accuracy (23). We report the baselines' results based on their official implementations from GitHub and where available, pre-trained models from Hugging Face; we executed them ourselves on our target device types.

ImageNet Results: Table 2 demonstrates our performance on ImageNet over various state-of-the-art popular approximation methods. We compare UWYN with the highest accuracy for a given model, where it has several variants. For a comparison with other shallower versions of these models, please refer to Table 7 in Appendix.While there are models that have lower MACs (e.g., EfficientViT (36)) or fewer parameters (e.g., MobileViT (40)) individually, UWYN is consistently a better tradeoff between the MACs and number of parameters. Apart from the vanilla version of ViT (23) (which is an exact, i.e., non-approximated model), UWYN demonstrates higher accuracy than other approximation methods. We further perform inference at almost half the amount of time required by EfficientViT and EfficientFormer (30). The fact that UWYN has lower values for MACs and Params emphasizes the efficiency of our model, achieving reduced computational costs without a significant sacrifice in performance. As compared to AdaptFormer (8), UWYN uses 40% lower MACs, 25% of the number of parameters, yet outperforms it in accuracy with lesser inference time on all devices.

Unconventional 2D datasets and other popular datasets: For 2D image classification, we evaluate on unconventional datasets, such as low resolution images like BloodMNIST and noisy images like SVHN. From the results in Table 3, we see that UWYN achieves competitive accuracy, maintaining a low inference time compared to traditional models like ViT and LGViT. It is worth noting that

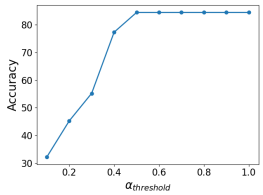


Figure 4: ImageNet accuracy vs. majority-vote threshold $\alpha_{\text{threshold}}$: from 32.5% at ($\alpha$=0.1) to 84.39% at ($\alpha$=0.5), then plateauing.

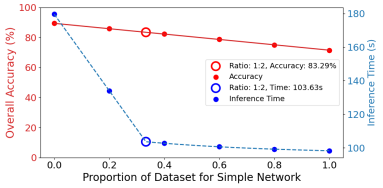


Figure 5: Accuracy (red, left) and inference time (blue, right) vs. fraction routed to the simple branch on CIFAR-100; at 33% simple, UWYN achieves 83.29% accuracy in 103.63 s, a reasonable point in the speed-accuracy trade-off.

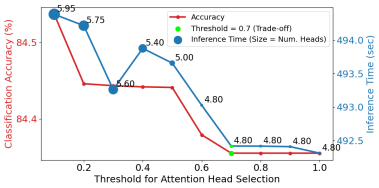


Figure 6: Varying the threshold for attention head selection for ImageNet classification task. A threshold value of 0.7 strikes a reasonable trade-off between the accuracy and the inference time.

the BloodMNIST website (68) reports achieving a 96.1% accuracy using Google AutoML Vision, with 2 hours of node time required for training. In contrast, training the ResNet model and the Early-Exit Transformer for our approach took only 20 and 100 epochs, respectively, amounting to approximately 1 hour of total training time from scratch. Our method UWYN achieved a 96.17% accuracy. However, since they have not released their code, it is difficult to directly compare the speedup. AdaptFormer achieves the highest accuracy on CIFAR100 (86.2%) with an inference time of 136.67 s. UWYN demonstrates a balance of efficiency and accuracy across CIFAR100 (83.29%, 103.63 s), CIFAR10 (94.87%, 103.4 s), BloodMNIST (96.17%, 27.88 s), and SVHN (92.16%, 225.67 s), often with fewer MACs and parameters. UWYN performs inference faster consistently over all three device classes — server class GPU as P100, and edge devices as AGX Xavier and Jetson Orin.

Table 3: **Variety of Datasets:** Performance on 2D image classification tasks across various datasets, showing accuracy and inference time on P100.

Method (2D)	CIFAR100		CIFAR 10		BloodMNIST		SVHN	
	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time
ViT (23; 56)	93.95%	219.47 s	95.29% (1)	219.52 s	89.6% (3)	58.68 s	96.40% (2)	275.8 s
LGViT (66)	82.57%	94.98 s	92.5%	95.27 s	91.5%	30.28 s	90.24%	250.66 s
AdaptFormer (8)	86.2%	136.67	93.36%	137.12	90.6%	42.5 s	93.2%	230.56 s
UWYN	83.29%	103.63 s	94.87%	103.4 s	96.17%	27.88 s	92.16%	225.67 s

Choice of Hyper-parameters: We observed that maximum accuracy was achieved at $\alpha_{\text{threshold}} = 0.5$ (equivalent to the majority case) when increasing the threshold $\alpha_{\text{threshold}}$ from 0.1 to 0.5 [Figure 4]. Beyond this point, increasing $\alpha_{\text{threshold}}$ resulted in only marginal gains in accuracy (in the second decimal place). Therefore, an early exit is triggered if the confidence score exceeds the predefined threshold of $\alpha_{\text{threshold}} = 0.5$ (or if the majority of token predictions agree on the same object), and has been maintained throughout the results for the purpose of this paper.

We also explored the impact of varying the proportions of simple and complex data instances. We notice a similar trend of all datasets, and the results for CIFAR 100 are presented in Figure 5. Processing all data through the simple network yields the lowest inference time but also the lowest accuracy. Conversely, utilizing only the complex network results in high accuracy at the cost of significantly increased inference time. However, a closer examination of the graph reveals a notable trend: If the ratio of simple data instances is less than 0.3, the accuracy is high but the inference time is unacceptably high. There is a knee in the curve at 0.3 and beyond that, the inference time drops slowly, as does the accuracy. So, we choose 33.33% of the data points for all our experiments, as going the simple (ResNet) path. This reduction can likely be attributed to the increased prevalence of early exits in the complex network beyond this ratio, leading to faster processing, especially when combined with the efficient processing of the remaining simple instances by the simple network. This observed behavior strongly motivated our decision to adopt a 1:2 ratio for simple to complex data instances throughout our evaluation. For selecting the number of attention heads, the threshold of 0.7 empirically strikes a favorable balance. It maintains a high classification accuracy while achieving a noticeable reduction in inference time compared to lower threshold values, as observed in Figure 6. The numbers overlaid on the inference time data points refer to the number of attention heads used on average at a particular threshold. There are 6 total attention heads, and we observe that after a threshold of 0.6, an average of 4.8 attention heads are used.

4.2 3D RESULTS

Table 4 presents the performance of UWYN on 3D shape classification task using the ModelNet40 (64) and ScanObjNN (57) datasets. Our method outperforms existing approaches in both accuracy and inference time, such as Point Transformer (72) and PointNet++ (45). Specifically, UWYN achieves higher accuracy on both datasets—94.03% on ModelNet40 and 92.43% on ScanObjNN—while significantly reducing inference times (78.31% reduction on ModelNet40 and 73.4% reduction on ScanObjNN inference time with respect to PointTransformer (72) on P100). These results highlight the efficiency of UWYN's approach, which reduces the dimensionality of point clouds and accelerates the processing pipeline compared to traditional methods. We repeat experiments on our edge devices and observe a speed up in inference time for all cases. As discussed earlier, in Section 2, existing pipelines are slow primarily due to their need for heavyweight pre-processing; however, UWYN overcomes this limitation through a simple conversion to 2D data format. It is worth noting that our speed up in the 3D task relative to state-of-the-art is better than in the 2D task; this is attributable to the reason just given above.

Table 4: **3D shape classification on ModelNet40 (64) and ScanObjNN (57):** UWYN achieves 94.03% and 92.43% accuracy while reducing inference time by over 75% on P100, AGX Xavier, and Orin compared to Point Transformer and PointNet++.

Method (3D)	ModelNet40				ScanObjNN			
	Accuracy	Inference Time			Accuracy	Inference Time		
		Xavier	Orin	P100		Xavier	Orin	P100
Point Transformer (72)	92.4%	3892.23 s	3695.62 s	842.41 s	90.5%	900.25 s	821.45 s	171.2
PointNet++ (45)	91.8%	2896 s	2667.34 s	705.04 s	84.2%	825.23 s	798.66 s	142.5
UWYN	**94.03%**	**503.5 s**	**440.12 s**	**182.7 s**	**92.43%**	**150.27 s**	**139.55 s**	**45.45 s**

4.3 OBJECT DETECTION RESULTS

In this section, we evaluate UWYN on object detection using an SSD (34) backbone and report results on the COCO dataset (32). We measure performance in terms of mean average precision (mAP) across all classes and compare against models of comparable scale as well as the strongest variants within each family. As shown in Table 5, UWYN delivers the highest accuracy while requiring less than half the number of parameters. Notably, UWYN achieves a 20% improvement in mAP over RT-DETRv2 (39) while using under 30% of its trainable parameters, demonstrating both the efficiency and extensibility of our approach across diverse problem settings.

Table 5: **Object Detection Performance:** UWYN performs with a higher mAP using less than half of the number of parameters.

Method	MACs (G)	Params (M)	mAP (%)
RT-DETRv2S (39)	30	20	48.1
RT-DETR-R18 (73)	30	20	46.5
EfficientDet (51)	99.6	64	59.9
YOLO-12m (53)	33.75	20.2	52.5
YOLO-v8s (48; 20)	14.4	11.16	43.2
SSD (34)+UWYN	**10.84**	**4.31**	**64.22**

5 CONCLUSION AND BROADER IMPACT

In conclusion, this work introduces UWYN, a novel approach to enhance the efficiency of classification tasks through the strategic reduction of redundant computations. Quantitatively, UWYN achieves a 25% reduction in MACs relative to comparable methodologies while maintaining competitive performance levels. These findings underscore the potential of UWYN as a viable and efficient solution for vision model deployment in resource-constrained real-world applications, including edge devices and unmanned aerial vehicles. Future research directions may explore the generalization of UWYN to other data modalities and its adaptability to evolving hardware and devices.

Reproducibility statement: We present training protocols, architectural specifications, and hyperparameter choices in the Results section (Section 4). The underlying mathematical formulations are described in the Methods section (Section 3), while implementation details related to Sobel operators are provided in the Appendix (Section A.1). The source code will be released publicly upon acceptance of this paper.

REFERENCES

[1] Cifar 10 weights. https://huggingface.co/nateraw/vit-base-patch16-224-cifar10.

[2] Svhn weights. https://huggingface.co/edadaltocg/vit_base_patch16_224_in21k_ft_svhn.

[3] Vit - bloodmnist. https://huggingface.co/TaLong/ViT_bloodmnist.

[4] Attention - pytorch. https://pytorch.org/docs/stable/_modules/torch/nn/modules/activation.html#MultiheadAttention.forward, 2022.

[5] thop — pypi.org. https://pypi.org/project/thop/, 2022.

[6] Andrea Acevedo, Anna Merino, Santiago Alférez, Ángel Molina, Laura Boldú, and José Rodellar. A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. *Data in Brief*, 30:105474, 2020.

[7] Mengzhao Chen, Mingbao Lin, Ke Li, Yunhang Shen, Yongjian Wu, Fei Chao, and Rongrong Ji. Cf-vit: A general coarse-to-fine method for vision transformer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 7042–7052, 2023.

[8] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition. *Advances in Neural Information Processing Systems*, 35:16664–16678, 2022.

[9] Xuanyao Chen, Zhijian Liu, Haotian Tang, Li Yi, Hang Zhao, and Song Han. Sparsevit: Revisiting activation sparsity for efficient high-resolution vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2061–2070, 2023.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[11] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] Jianwei Feng, Hengliang Tan, Wangwang Li, and Ming Xie. Conv2next: Reconsidering conv next network design for image recognition. In *2022 International Conference on Computers and Artificial Intelligence Technologies (CAIT)*, pages 53–60. IEEE, 2022.

[14] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Herve Jegou, and Matthijs Douze. Levit: A vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12259–12269, October 2021.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. In *International Conference on Computer Vision (ICCV)*, 2021.

[17] Youbing Hu, Yun Cheng, Anqi Lu, Zhiqiang Cao, Dawei Wei, Jie Liu, and Zhijun Li. Lf-vit: Reducing spatial redundancy in vision transformer for efficient image recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 2274–2284, 2024.

[18] Youbing Hu, Yun Cheng, Anqi Lu, Dawei Wei, and Zhijun Li. Sac-vit: Semantic-aware clustering vision transformer with early exit. *arXiv preprint arXiv:2503.00060*, 2025.

[19] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer, 2016.

[20] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8. `https://github.com/ultralytics/ultralytics`, 2023.

[21] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[22] Jing Yu Koh, Daniel Fried, and Russ R Salakhutdinov. Generating images with multimodal language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[24] Alex Krizhevsky and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). Technical report, University of Toronto, 2009.

[25] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[26] Mike Lewis, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[27] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19730–19742. PMLR, 23–29 Jul 2023.

[28] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR, 2022.

[29] Yanyu Li, Ju Hu, Yang Wen, Georgios Evangelidis, Kamyar Salahi, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Rethinking vision transformers for mobilenet size and speed. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16889–16900, 2023.

[30] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *Advances in Neural Information Processing Systems*, 35:12934–12949, 2022.

[31] Yuhong Li, Jiajie Li, Cong Hao, Pan Li, Jinjun Xiong, and Deming Chen. Extensible and efficient proxy for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6199–6210, 2023.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[33] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ringattention with blockwise transformers for near-infinite context. In *The Twelfth International Conference on Learning Representations*, 2024.

[34] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[35] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. FastBERT: a self-distilling BERT with adaptive inference time. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, Online, July 2020. Association for Computational Linguistics.

[36] Xinyu Liu, Houwen Peng, Ningxin Zheng, Yuqing Yang, Han Hu, and Yixuan Yuan. Efficientvit: Memory efficient vision transformer with cascaded group attention. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14420–14430, 2023.

[37] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer V2: scaling up capacity and resolution. *CoRR*, abs/2111.09883, 2021.

[38] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[39] Wenyu Lv, Yian Zhao, Qinyao Chang, Kui Huang, Guanzhong Wang, and Yi Liu. Rt-detrv2: Improved baseline with bag-of-freebies for real-time detection transformer. *CoRR*, 2024.

[40] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022.

[41] Sachin Mehta and Mohammad Rastegari. Separable self-attention for mobile vision transformers. *Transactions on Machine Learning Research*, 2023.

[42] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022.

[43] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 7. Granada, Spain, 2011.

[44] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[45] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[46] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, et al. Mobilenetv4: universal models for the mobile ecosystem. In *European Conference on Computer Vision*, pages 78–96. Springer, 2024.

[47] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[48] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[49] Andreas Steiner, Alexander Kolesnikov, , Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[50] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[51] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[52] Surat Teerapittayanon and Bradley McDanel. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[53] Yunjie Tian, Qixiang Ye, and David Doermann. Yolov12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524*, 2025.

[54] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[55] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers distillation through attention, 2021.

[56] Ching-Hsun Tseng, Hsueh-Cheng Liu, Shin-Jye Lee, and Xiaojun Zeng. Perturbed gradients updating within unit space for deep learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 01–08. IEEE, 2022.

[57] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019.

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[59] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[60] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[61] Maciej Wołczyk, Bartosz Wójcik, Klaudia Bałazy, Igor T Podolak, Jacek Tabor, Marek Śmieja, and Tomasz Trzcinski. Zero time waste: Recycling predictions in early exit neural networks. *Advances in Neural Information Processing Systems*, 34:2516–2528, 2021.

[62] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision, 2020.

[63] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. In *European Conference on Computer Vision*, pages 68–85. Springer, 2022.

[64] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[65] Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. BERxiT: Early exiting for BERT with better fine-tuning and extension to regression. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 91–104, Online, April 2021. Association for Computational Linguistics.

[66] Guanyu Xu, Jiawei Hao, Li Shen, Han Hu, Yong Luo, Hui Lin, and Jialie Shen. Lgvit: Dynamic early exiting for accelerating vision transformer. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9103–9114, 2023.

[67] Jiancheng Yang, Rui Shi, and Bingbing Ni. Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis. In *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 191–195, 2021.

[68] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.

[69] Shangshang Yang, Xiaoshan Yu, Ye Tian, Xueming Yan, Haiping Ma, and Xingyi Zhang. Evolutionary neural architecture search for transformer in knowledge tracing. *Advances in Neural Information Processing Systems*, 36:19520–19539, 2023.

[70] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 558–567, October 2021.

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

[72] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[73] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16965–16974, 2024.

[74] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A APPENDIX

Here, we will demonstrate the additional results from our work.

A.1 SOBEL OPERATOR

The Sobel operator calculates the gradient of an image by convolving the image with two 3x3 kernels: one for detecting horizontal edges (G_x) and one for detecting vertical edges (G_y). The convolution of the image I at each pixel (i, j) with these kernels is expressed as:

$$G_x(i,j) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} G_x(m,n) \cdot I(i+m, j+n) \tag{6}$$

$$G_y(i,j) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} G_y(m,n) \cdot I(i+m, j+n) \tag{7}$$

where the Sobel kernels are defined as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{8}$$

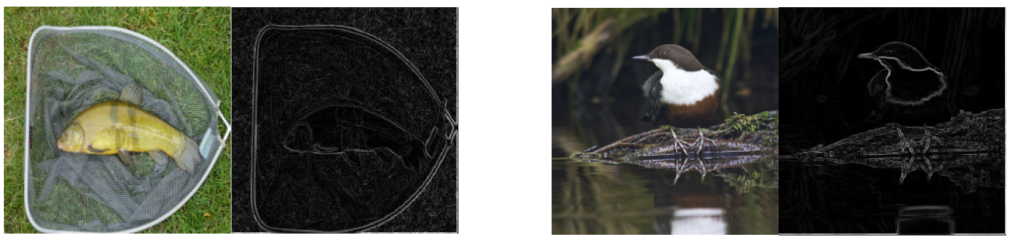
The gradient magnitude is then calculated as:

$$\text{Magnitude}(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2} \tag{9}$$

The gradient magnitude is a measure of the sharpness of the image - a measure of how much curvature or information is there in the image.

A.2 VISUAL REPRESENTATION OF SIMPLE AND COMPLEX IMAGES

Figure 7 is a visual representation of a simple and a complex image from the ImageNet dataset. We identify few curves on the simple image and more detailed silhouettes on complex images, based on our thesholding mechanism in Section 3.1.



(a) Simple Image Thresholding **(b) Complex Image Thresholding**

Figure 7: Examples from the ImageNet dataset (12) illustrating the application of a Sobel operator [A.1] for edge detection. (a) A "simple" image (left) with minimal details, processed with the Sobel operator (right), showing fewer prominent edges. (b) A "complex" image (right) with intricate details, processed with a Sobel operator (right), revealing a dense edge pattern.

A.3 SPEED-UP DUE TO ATTENTION PARALLELIZATION

As discussed in Section 3.3.3, parallelizing attention computations significantly accelerates inference in the Efficient Early-Exit Transformer, with a more pronounced effect on 2D tasks than 3D. In 3D tasks, the extra pre-processing for point cloud data and 2D projection introduces additional computational overhead, limiting the speedup.

Table 6: **Parallel attention computation speed-up:** Comparison of data instances processed per second for Early-Exit Efficient Transformer, evaluated under both parallel and sequential attention schemes. The speedup in 2D images is more than 3D shapes.

Dataset	Data instances processed per second	
	Attention parallel	Attention sequential
CIFAR 100 (24)	60.15	58.37
ImageNet (12)	28.23	26.40
ModelNet40 (64)	13.74	13.57

A.4 Comparison with extremely small models

Despite the existence of models with considerably lower MACs in Table 7, our evaluation reveals a clear advantage for our approach. Our model exhibits faster inference times, notably on edge devices, and maintains a higher accuracy than any of the identified low-MAC models. These experiments have been performed on ImageNet. We also add entries from recent models whose implementation is not publicly available at the time of submission. We use the symbol: − to indicate observations that are not accessible due to the aforementioned reason.

Table 7: **Comparison with other models:** Comparison of small MAC Methods with UWYN on ImageNet. We demonstrate higher accuracy than all and faster inference for some.

Method	MACs	Params	Accuracy	Time (Xavier)	Time (P100)	Time (Orin)
Efficient ViT-M0 (36)	0.1 G	5.4 M	71.9%	1415 s	346.87 s	1279.38 s
LeViT-128S (14; 59)	0.2 G	7.77 M	76.5%	1582.14 s	401.16 s	1632.71 s
EfficientFormerV2-s0 (29; 59)	0.3 G	3.5 M	76.1%	1994.42 s	494.13 s	2068.05 s
LF-ViT (17)	1.85 G	-	82.2%	-	-	-
CF-ViT (7)	2.4 G	-	81.9%	-	-	-
SAC-ViT (18)	1.6 G	-	82.3%	-	-	-
UWYN	1.2 G	22.86 M	84.39%	1796 s	492.63 s	1695.23 s

A.5 Using a larger batch size to compare results

Here we compare the inference time when we use a batch size of 32 during inference time. There is a larger speedup in the 3D pre-processing due to efficient handling of data as compared to the other methods. We identify the simple and complex data instances from beforehand, batch them together and then perform inference.

Figure 8: **Batch inference:** This table shows the time required for inference using a batch size of 32 on the ModelNet40 dataset.

Method	Inference Time
PointNet++ (45)	695.45 sec
Point Transformer (72)	701.88 sec
UWYN	166.39 sec

Figure 9: **Batch inference:** This table shows the time required for inference using a batch size of 32 on the CIFAR 10 dataset.

Method	Inference Time
LGViT (66)	59.83 sec
AdaptFormer (8)	62.18 sec
UWYN	40.45 sec

A.6 MACs for other networks

In this section we will compare the MACs of our networks with other popular transformer architectures. We are using the maximum MAC for our methods for comparison. From the table below, we demonstrate that our MACs are minimal with respect to other architectures as well.

Table 8: **MACs of other methods:** From the table, it is evident that our method is much more computationally efficient than the existing popular architecture choices. We use the thop (5) python library to calculate the MACs. Our MACs are very low compared to the other state of the art.

Architecture	MACs
DeIT Small (55; 62; 12)	4249 M
Swin V2 Tiny (37)	5760 M
Mobile ViT small (40)	347.5 M
EfficientNet-B0 (50)	380.55 M
ConvNeXt-T (13)	4.5 G
MNv4-Hybrid-L (46)	7.2 G
Our Complex Net (CIFAR 100)	337 M
UWYN (CIFAR 100)	1186 M

A.7 Feasibility of our early exit

To explore if our concept of early exit is feasible or not, we have implemented the early exit from on a ViT (23). This indicates that after execution of each Transformer block, the output was sent to the classifier, and based on the classifier features and labels of the images, a cross-entropy loss was implemented. Table 9 demonstrates the results when we train these pipelines over a limited number of epochs, thereby testifying the feasibility of our work. We start the early exit after and confidence score calculation after at least 4 transformer blocks during inference.

Table 9: **Possibility of Early Exit:** Performance comparison of full capacity vs. early exit ViTs across various datasets, trained from scratch for 100 epochs. This table illustrates the generalizability of our method, showing that performance acceleration is consistent across different datasets and not solely reliant on the patch and attention head selector networks. From the full capacity accuracy (Acc.), there is a limited dip by 1%, while the other metrics, such as MACs, Parameters (Params), and Inference time (Time), have reduced significantly.

Metric	CIFAR-10 (Full Capacity)	BloodMNIST (Full Capacity)	CIFAR-10 (Early Capacity)	BloodMNIST (Early Exit)
Accuracy (%)	80.21	97.02	80.08	96.89
MACs (M)	1384	1401.8	836.6	596.4
Params (M)	21.31	21.31	12.34	12.34
Time (sec)	273.07	12.52	247.09	5.59

A.8 Other motivational examples

As mentioned in the Introduction, Section 1, the redundancy in transformers is also apparent in Natural Language Processing tasks as well. We examine BART (26), a widely used transformer model comprising 12 encoder and 12 decoder blocks, in the context of text classification task on the MNLI dataset (60). In this task, Lewis *et al*. (26) categorizes if a pair of sentences as contradictory or not. We systematically drop later blocks in both the encoder and decoder to evaluate the impact on performance and computational efficiency. As Table 10 illustrates, reducing the number of blocks yields significant computational savings with only minor accuracy decreases. For instance, using 10 encoder blocks and 8 decoder blocks results in a mere 0.07% accuracy reduction while saving approximately 30 ms per CPU and GPU computation time, reducing FLOPs by almost 25%. These findings suggest a trade-off between accuracy and efficiency, which could be leveraged for applications where rapid inference is critical or resources are constrained.

Table 10: **Motivation from NLP:** Experimental results of BART (26) on the MNLI dataset, demonstrating the impact of reducing model components (encoder/decoder blocks) on computational demand and accuracy.

Encoders	Decoders	Params	CPU (ms)	GPU (ms)	KFLOPs	Accuracy (%)
12	12	4.07×10^8	81.51	87.73	12.03×10^3	83
10	10	3.49×10^8	51.77	61.08	10.02×10^3	82.35
8	8	2.90×10^8	46.52	50.19	8.02×10^3	52.66
10	8	3.15×10^8	53.86	57.76	8.88×10^3	82.93
10	6	2.81×10^8	37.58	34.55	7.73×10^3	70.08

Table 10 highlights trade-offs between efficiency (CPU/GPU time for inference on the entire test set, parameters, KFLOPs) and classification performance, supporting the hypothesis that not all model components are essential for model efficiency. This indicates that intermediate features are also well learnt in most cases.