Combining Recursive Weight-Sharing with Token Merging for Edge Vision Transformers

JUNSEO KIM, University of Twente, The Netherlands

Vision Transformers (ViTs) demonstrate exceptional performance in computer vision but suffer from large parameter counts and quadratic computational complexity, $O(N^2)$, severely limiting their deployment on resource-constrained edge hardware. While recursive weight-sharing reduces parameter counts and token merging mitigates computational and memory bottlenecks, integrating these two paradigms without costly retraining is non-trivial, leaving this intersection largely unexplored. We propose a post-training multi-axis compression approach that successfully combines the recursive weight-sharing of the Sliced Recursive Transformer (SReT) with the dynamic token merging algorithm of Token Merging (ToMe). By implementing an unmerge tracking stack, enforcing strict mathematical merging bounds, and applying parallel spatial tracking, our methodology resolves the spatial and merging constraints of the integration. Furthermore, we utilize an exponential token reduction schedule to stabilize the semantic densification inherent to recursive loops. Benchmarked on ImageNet-1K, our optimized configuration achieves a 27.6% increase in throughput and a 38.5% reduction in Peak Activation Memory (PAM) with a minimal 1.47% accuracy drop on a GPU at a batch size of 128. However, the algorithmic overhead negated the performance gains at a batch size of 1. Nevertheless, this approach establishes the feasibility of dynamic token reduction within recursive ViT architectures, providing a structural baseline for future edge-targeted optimizations.

Additional Key Words and Phrases: Vision Transformers, Model Compression, Recursive Weight-Sharing, Token Merging, Edge AI

1 INTRODUCTION

The self-attention mechanism of the Transformer [21] revolutionized computer vision through the introduction of the Vision Transformer (ViT) [6]. However, this architecture results in massive parameter counts and quadratic computational complexity, $O(N^2)$, with respect to sequence length, N, limiting practical deployment on resource-constrained edge hardware [12, 14]. To reduce parameter counts, recursive weight-sharing paradigms have been explored, which repeatedly route the input sequence through a single transformer block, applying identical weights across multiple passes [4, 17, 25]. However, this approach results in increased computational complexity and Peak Activation Memory (PAM).

To explicitly target these computational and memory bottlenecks, token reduction paradigms aim to reduce the token sequence during inference. Modern strategies include token pruning [10, 15], token merging [1, 19], and token fusion [8]. Among these, Token Merging (ToMe) [1] has emerged as a promising post-training framework.

Historically, recursive weight-sharing and token reduction have been treated as mutually exclusive compression paradigms. Moreover, while alternative efficient architectures exist, they rely on retraining phases, distillation, or structural redesigns [2, 12, 14, 20, 22]. This creates a literature gap for post-training compression methods. We address this gap by proposing a post-training compression approach that combines recursive weight-sharing with token merging. We validate this integration through a concrete case study that integrates the Sliced Recursive Transformer (SReT) [17] with ToMe [1].

However, a naive integration introduces severe architectural friction. First, token reduction violates the rigid 2D spatial grid required by SReT's hierarchical convolutional pooling layers. Second, unconstrained merging can violate the strict group-size boundaries inherent to the Sliced Group Self-Attention (SGA) of SReT. Third, SReT's spatial permutations break the tracking necessary for the proportional attention of ToMe. Finally, standard merging rates [1] do not account for the semantic densification generated within recursive loops. Resolving these incompatibilities is mandatory for a successful integration.

To address these problems, we formulate the following main research question:

> *How can recursive weight-sharing and token merging be combined within a ViT to simultaneously reduce parameter counts, increase throughput, and lower PAM without severe accuracy loss?*

We further subdivide the research question into three sub-questions:

- **RQ1 (Integration):** How can token merging be integrated into a hierarchical, recursive architecture while complying with various spatial and merging constraints?
- **RQ2 (Optimization):** What token reduction schedule best balances performance gains and accuracy loss?
- **RQ3 (Evaluation):** How does the proposed approach benchmark against the uncompressed recursive baseline across GPU and CPU environments?

By addressing these questions, we make the following contributions:

- The formulation of a post-training multi-axis compression approach that simultaneously reduces parameter counts, increases throughput, and lowers PAM.
- An exponential token reduction schedule compatible with recursive architectures and semantic densification.
- An evaluation of the proposed approach against an uncompressed, recursive baseline on the ImageNet-1K dataset [5].

The remainder of this paper is structured as follows: Section 2 contextualizes our work within literature, Section 3 details the SReT and ToMe integration, Section 4 details the experimental setup, Section 5 analyzes and discusses the results, Section 6 discusses the limitations of our work, and Section 7 concludes the study.

TScIT 45, July 3, 2026, Enschede, The Netherlands

2 BACKGROUND AND RELATED WORK

In this section, we review the existing literature and foundational concepts related to our proposed approach.

2.1 Vision Transformers and Spatial Pyramids

Modern ViTs stem from the Transformer [21], computing global dependencies via scaled dot-product attention over Queries (Q), Keys (K), and Values (V):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where d_k is the scaling dimension. ViT [6] adapted this to computer vision by flattening 2D images into 1D patch sequences. However, standard ViTs maintain a strictly isotropic layout where sequence length and channel capacity remain uniform, resulting in a quadratic computational complexity relative to sequence length.

To mitigate this bottleneck, hierarchical spatial pyramids introduce progressive downsampling [7, 13, 23]. While Swin [13] and PVT [23] employ localized attention or spatial reduction, PiT [7] directly integrates CNN-based convolutional pooling. This effectively models complex spatial hierarchies, establishing the exact architectural blueprint inherited by recursive variants like SReT [17].

2.2 Recursive Weight-Sharing

Weight-sharing minimizes parameters by reusing weights via recurrence. Pioneered by the Universal Transformer [4], this paradigm limits parameter counts while allowing arbitrary computational depth. Early vision adaptations like MiniViT [25] applied layer-wise weight-multiplexing to simulate depth, but lacked true parametric recursion.

Pure spatial recurrence within a hierarchical architecture was realized by SReT [17]. Built on PiT's [7] convolutional pyramid blueprint, SReT routes sequences recursively through identical blocks with frozen parameters across consecutive loops.

While SReT is highly parameter-efficient, repeating self-attention creates a severe computational bottleneck, and routing tokens through a shared parameter space accelerates feature densification. This severely penalizes throughput and increases PAM, necessitating sequence-level compression.

2.3 Token Reduction

To alleviate the quadratic complexity of ViTs, token reduction paradigms minimize spatial redundancy during inference via token pruning [10, 15], token merging [1, 19], or token fusion [8].

Token pruning permanently discards patches. However, methods like DynamicViT [15] require retraining, while post-training alternatives like EViT [10] rely heavily on the [CLS] token, which is incompatible with SReT's [17] global average pooling. Crucially, permanent deletion prevents reconstructing the rigid 2D grid required for inter-stage convolutional pooling.

In contrast, ToMe [1] is a post-training module that merges similar tokens via Bipartite Soft Matching (BSM). To preserve algorithmic fidelity, ToMe scales the key matrix using a token-mass vector (s):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \log(s_K)\right)V \tag{2}$$

While variants like PiToMe [19] optimize matching, ToMe's reversibility is uniquely valuable for hierarchical networks. Along with a merge operation, BSM enables an "unmerge" operation that duplicates features to restore original sequence length. This infrastructure provides the exact mechanism to satisfy the spatial layout constraints of SReT's inter-stage convolutional pooling.

Token fusion frameworks like ToFu [8] hybridize these paradigms. However, combining destructive pruning with irreversible merging destroys bi-directional tracking. Lacking a structural unmerge function, fusion remains fundamentally incompatible with SReT's hierarchical pyramid.

2.4 Alternative Efficiency Paradigms

Beyond recursive weight-sharing and token reduction, several alternative paradigms target structural efficiency. Architectural redesigns (MobileViT [14], EfficientViT [12]), reparameterizations (FastViT [20], RepViT [22]), and structured pruning (X-Pruner [24]) increase throughput by modifying layouts or cutting channels. However, their reliance on intensive retraining prevents post-hoc application to pre-trained models.

Conversely, implementation-level and precision-reduction frameworks optimize hardware execution without altering architecture. Hardware-aware kernels like FlashAttention [3] bypass memory bottlenecks via GPU SRAM tiling, but leave spatial token redundancy untouched. Similarly, post-training quantization (FQ-ViT [11]) compresses weights and activations into low-bit formats during inference. By optimizing memory access and numerical precision rather than parameter counts or sequence lengths, these approaches represent entirely complementary mechanisms that can integrate seamlessly alongside our proposed approach.

2.5 Orthogonal Compression

ViT optimization has shifted from isolated, single-axis methods toward multi-axis joint compression. For instance, MADTP++ [2] unifies structural weight pruning and dynamic token pruning but requires retraining, preventing inference-only deployment. To bypass this overhead, post-training strategies like QUOTA [9] merge low-bit quantization with deterministic sequence pruning.

While highly efficient for isotropic ViTs, these paradigms rely exclusively on destructive, non-reversible operations. Crucially, the intersection of recursive weight-sharing and dynamic token merging remains unexplored. Our approach fills this void using a post-training integration of these two axes.

3 METHODOLOGY AND IMPLEMENTATION

In this section, we detail the technical integration of SReT and ToMe. Figure 1 illustrates the complete integration of how tokens are reduced throughout the complete forward pass.

3.1 Spatial Layout Constraints

SReT [17] inherits its structure directly from PiT [7], which utilizes a spatial pyramid design. Unlike standard ViTs that split an image into non-overlapping patches using a linear projection, SReT applies overlapping convolutional embeddings to generate its initial token sequence. As depicted in Figure 2, these tokens are then processed

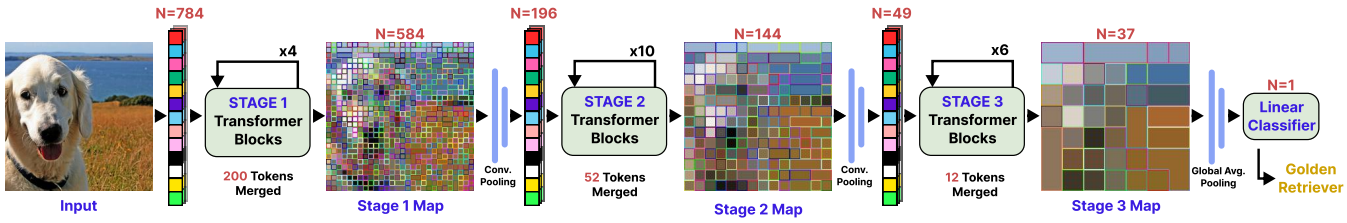

Fig. 1. **An overview of the proposed approach, integrating the recursive weight-sharing of SReT with the token merging of ToMe.** The initial token sequence $N = 784$ is reduced using an exponential schedule. Crucially, the spatial layout is reconstructed prior to each inter-stage convolutional pooling layer.

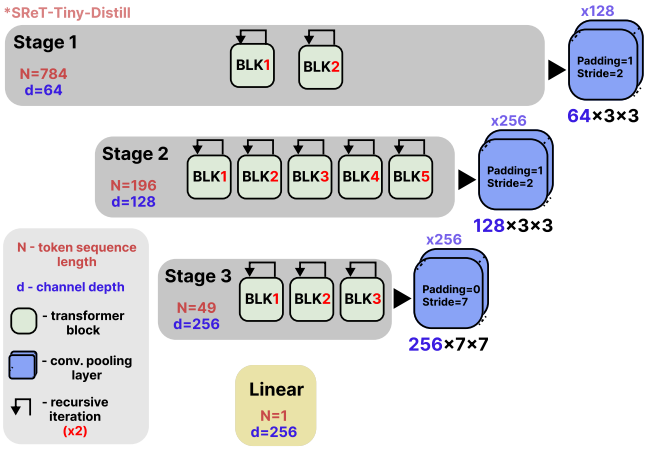

Fig. 2. **The hierarchical spatial pyramid architecture of SReT.** Dynamic token reduction must satisfy the strict spatial downsampling of convolutional pooling layers.

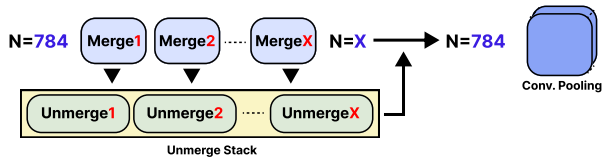

Fig. 3. **The unmerge stack mechanism for token sequence length restoration.** By applying unmerge operations, the original length ($N = 784$) is restored to satisfy the spatial grid requirements of the convolutional pooling layer.

through three distinct stages. Each stage contains a set of transformer blocks that are executed recursively. Between these stages, convolutional pooling layers decrease the spatial sequence length while increasing channel depth. After the final stage, global average pooling compresses the tokens for final classification.

This hierarchical design means that after tokens exit the transformer blocks of a given stage, they must pass through an inter-stage convolutional pooling layer. Since convolutional kernels operate on structured 2D spatial grids (e.g., 14×14), they expect a sequence length that can be perfectly reshaped into a square tensor. However, standard token reduction eliminates an arbitrary number of tokens, destroying this perfect square property. Consequently, the sequence becomes structurally invalid for the convolutional kernels, triggering a dimensional clash at the end of the stage.

The BSM algorithm of ToMe [1] provides a unique mechanism to resolve this clash. During a merge operation, ToMe returns both a merging function and a corresponding "unmerge" function. Crucially, this unmerge function does not reverse the aggregation of the features. Instead, it restores the spatial dimensions of the sequence by duplicating the merged features back into their original spatial coordinates. This implies that regardless of how aggressively a sequence is compressed, the initial sequence length can be restored, provided that the unmerge functions are stored. This property is highly beneficial for SReT, offering a direct solution to satisfy the geometric requirements of inter-stage convolutional pooling.

To utilize this capability, we implement an "unmerge stack" as shown in Figure 3. As the token reduction schedule dynamically executes merge operations within a stage, the corresponding unmerge operations are pushed onto a last-in-first-out stack. Before the compressed sequence reaches the inter-stage convolutional pooling layer, the tensor is built back up to its original dimensions by applying the stored unmerge operations in strict reverse order.

This mechanism allows the attention and Multilayer Perceptron (MLP) blocks to reap the computational benefits of processing a compressed token sequence, while ensuring that the convolutional pooling layers receive grid-aligned tokens. However, this introduces operational overhead associated with tracking the unmerge functions and restoring the token sequence at the end of each stage.

3.2 Merging Constraints

With the geometric requirements of the inter-stage convolutional pooling layers resolved by the unmerge stack, we must now address the internal constraints of the transformer blocks. Due to the SGA mechanism of SReT and the BSM algorithm of ToMe, the number of tokens merged at every iteration (r) cannot be set arbitrarily. As depicted in Figure 4, the token merging operation is inserted between the SGA and MLP. Consequently, the reduction rate r must satisfy both the group divisibility constraints of the subsequent SGA operation and the bipartite matching limits of BSM.

Within the SGA, the token sequence is divided equally into groups. As illustrated in Figure 4, attention is computed separately for each group. Thus, a strict constraint is that the sequence length entering the block must be evenly divisible by the designated "group number" (g) for that iteration, and there must be at least one token per group. For example, an incoming sequence length of $N = 784$ complies

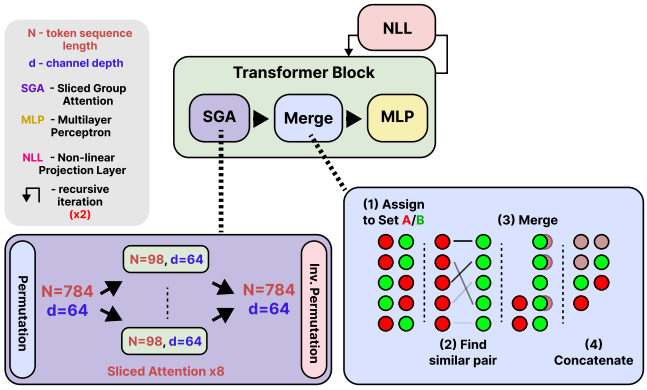

Fig. 4. **The SReT+ToMe transformer block.** The token merging module is inserted between the SGA and MLP layers. Detailed expansions illustrate the spatial permutations and grouped attention within SGA, alongside the bipartite matching steps used to merge tokens.

perfectly with a group number of $g = 8$, as each group receives exactly 98 tokens (Figure 4). Conversely, sequence lengths of $N = 15$ or $N = 7$ would trigger immediate tensor shape mismatches.

Since SReT blocks are executed recursively, the compressed sequence length exiting the current iteration ($N - r$) is the input sequence for the next iteration. Furthermore, SReT defines a different, generally lower group number (g_2) for this next iteration to allow each group to access more tokens, expanding the receptive field of the attention mechanism. To ensure the compressed sequence satisfies the requirements of SGA across all recursive passes, $N - r$ must be divisible by the Least Common Multiple (LCM) of the group numbers (g_1, g_2). Moreover, ToMe executes bipartite matching by first dividing the tokens into two sets (Figure 4). Since a token from one set is merged into a target token from the other, the algorithm cannot merge more than half of the total available tokens in a single iteration. Therefore, r must also be upper bounded by $\lfloor N/2 \rfloor$, resulting in the merging constraint:

$$(N-r) \bmod G = 0, \quad N-r \geq G, \quad G = \mathrm{LCM}(g_1, g_2), \quad r \leq \lfloor N/2 \rfloor \quad (3)$$

3.3 Spatial Tracking

As tokens are merged across the recursive stages of SReT, a single token comes to represent an increasingly large number of original patches. While standard ViT and SReT architectures treat all tokens equally, ToMe introduces a parallel token-size vector (s) to track the accumulated "mass" of every token to ensure proportional feature aggregation. Proportional attention (Equation 2) is achieved by scaling the Key matrix using s prior to the softmax operation. To integrate this into SReT, we initialize a mass tensor of ones at the beginning of each forward pass. This tensor is updated across all layers and merge operations to track the exact weight of each token.

During the forward pass, redundant tokens are identified via BSM. Crucially, we evaluate token similarity using the unweighted Key matrix (K) extracted from the attention layer. Isolating the similarity metric from the token mass ensures that merges are determined purely by semantic feature similarity, preventing "heavy" tokens with high accumulated mass from dominating the matching process.

Once BSM determines the optimal pairs, we merge the tokens using a weighted average. Since the tokens accumulate varying "masses" (the number of original spatial patches they represent), applying a naive, unweighted average would create a representational imbalance. Specifically, it would allow single-patch tokens equal influence over the merged result as multi-patch tokens. To prevent this, we pre-weight the input feature tensor by its corresponding mass via element-wise multiplication. The merging function then applies a sum reduction to independently aggregate both the weighted features and the parallel mass metrics. Finally, dividing the aggregated feature tensor by the updated mass tensor yields the correct weighted average representation.

A final structural challenge arises from SReT's use of token permutations. As depicted in Figure 4, SGA applies permutations and inverse permutations to the input sequence to shift the attention window across recursive passes, ensuring global token interaction. If the mass tensor is not permuted identically, the weights will misalign with their corresponding tokens, corrupting the proportional attention. By decoupling the token-mass tensor (s) and routing it in parallel through the exact same permutation and inverse permutation functions as the feature sequence, we guarantee that the mass metrics remain synchronized with their corresponding tokens.

3.4 Token Reduction Scheduling

The original ToMe framework utilizes two token reduction schedules: constant and linearly decreasing [1]. A constant schedule merges a fixed number of tokens (r) at every layer. Conversely, a linearly decreasing schedule starts by merging $2 \times r$ tokens in the first layer and reduces this amount to 0 by the final layer. While both schedules remove nearly the same total number of tokens for a given value of r, their temporal distributions differ significantly.

However, in recursive architectures like SReT where weights are reused across multiple iterations, removing tokens at every layer prevents the sequence representations from stabilizing. To allow a stabilized, near-fixed sequence length to propagate through the recursive loops, we propose a per-stage exponential reduction schedule:

$$r_d = \left\lfloor r_{\text{init}} \cdot \alpha^{(d-1)} \right\rfloor, \quad \alpha \in [0, 1] \quad (4)$$

where r_d is the reduction rate at block depth d within the current stage, r_{init} is the initial reduction rate, and α is the decay factor.

Unlike the baseline schedules that apply reductions globally across the entire network, our exponential schedule is applied per-stage. By manipulating the α parameter, we can range from a one-shot bulk removal entirely in the first layer ($\alpha = 0$) to a constant reduction across all layers in the stage ($\alpha = 1$). Crucially, regardless of the schedule's output (r_d), the actual value of r at any given layer remains strictly bounded by the merging constraints of Equation 3. If r_d violates group divisibility or bipartite matching limits, it is dynamically altered to the nearest number that ensures compliance.

4 EXPERIMENTAL SETUP

In this section, we detail the experimental setup used to evaluate our proposed approach, ensuring full reproducibility[1].

[1]Source code available at: https://github.com/junseokm/sret-tome

4.1 Dataset

All experiments are evaluated on the standard ImageNet-1K (ILSVRC 2012) validation dataset [5], which contains 50,000 images distributed evenly across 1,000 object classes. To ensure fair comparative benchmarking, we preprocess all images using the standard ViT pipeline: resizing images to a 256-pixel shorter edge, extracting a 224×224 center crop, and normalizing the pixel values using the standard channel-wise ImageNet mean and standard deviation.

4.2 Hardware and Software Environment

To ensure consistency, all evaluations are conducted on a single workstation equipped with an Intel Core Ultra 9 285K CPU and an NVIDIA RTX 4060 Ti GPU. While not edge hardware, we aim to provide a hardware-agnostic proxy for edge viability by tracking theoretical efficiency. The software stack utilizes Python 3.10, CUDA 13.0, PyTorch 2.11.0, Torchvision 0.26.0, and `timm` 0.4.12. Theoretical complexity, measured in Floating-Point Operations (FLOPs), is calculated using the `thop` 0.1.1 library. Hardware throughput is measured utilizing native CUDA timing events for GPU evaluations and custom performance snapshot utilities for CPU evaluations.

To guarantee fair comparative benchmarking, specific execution parameters are enforced. For GPU evaluations, cuDNN auto-tuning is enabled to optimize backend execution kernels. For CPU evaluations, strict GPU isolation is enforced, and computation is explicitly constrained to four hardware threads. Furthermore, the MKLDNN backend is intentionally disabled during CPU inference to prevent known C++ buffer overruns triggered by ToMe's underlying sequence reduction mechanisms.

4.3 Metrics

To evaluate the trade-offs introduced by the proposed integration, we measure hardware efficiency using throughput (img/s) and PAM (MB), theoretical complexity using FLOPS (G) and parameter counts (M), and task performance using Top-1 accuracy (%).

We evaluate GPU throughput and PAM under two conditions: a saturated batch size ($BS = 128$) to assess peak parallelization, and a minimal batch size ($BS = 1$) to simulate single-stream edge inference. To isolate the true mathematical memory footprint, PAM is tracked via PyTorch's native CUDA allocator. Conversely, CPU throughput is evaluated strictly at $BS = 1$.

To guarantee temporal accuracy, throughput metrics are recorded only after a warm-up phase (20 iterations for GPU, 50 iterations for CPU). This phase ensures that CUDA contexts on the GPU are fully initialized and CPU caches are stabilized. Following the warm-up, 100 iterations are executed. To mitigate the impact of random system spikes, we extract the median execution time across these 100 runs to calculate the final reported results.

4.4 Baselines

We compare our results against two primary baselines: DeiT-Tiny-Distill [18], which serves as a standard isotropic ViT, and the uncompressed SReT-Tiny-Distill [17], which serves as a recursive hierarchical ViT. Pre-trained weights for DeiT are sourced via the `timm` 0.4.12 library, and pre-trained weights for SReT are acquired directly from the authors' public repository [16].

4.5 Token Reduction Schedule Configurations

To evaluate the impact of different token reduction schedules, we benchmark all three schedule types: constant, linear, and exponential. We intentionally select the following configurations to expose the performance boundaries of recursive architectures:

- **Constant Schedule (r_{const}):** As the default strategy in ToMe [1], this schedule merges a fixed number of tokens at every layer. We test both a light reduction ($r_{\text{const}} = 10$) and a heavy reduction ($r_{\text{const}} = 20$) to establish a baseline.
- **Linear Schedule (r_{lin}):** Also introduced by ToMe [1], this schedule linearly decreases the number of merged tokens deeper into the network, scaling from $2 \times r_{\text{lin}}$ down to 0. While prior work [1] indicates this maximizes throughput in standard ViTs by heavily frontloading the token reductions, we evaluate matched configurations ($r_{\text{lin}} = 10$ and $r_{\text{lin}} = 20$) to test whether aggressively removing tokens early benefits hierarchical, recursive architectures.
- **Exponential Schedule (r_{init}, α):** For our proposed approach, we evaluate three distinct configurations of the exponential schedule to map its performance across varying reduction trajectories. Specifically, we test a *gradual decay* ($r_{\text{init}} = 0.10, \alpha = 0.6$) representing a light initial reduction followed by a smooth drop, a *single-shot reduction* ($r_{\text{init}} = 0.25, \alpha = 0.0$) representing a large initial reduction followed by a flat trajectory, and an *aggressive decay* ($r_{\text{init}} = 0.40, \alpha = 0.2$) representing a heavy initial reduction followed by a sharp drop.

5 RESULTS AND DISCUSSION

In this section, we evaluate the proposed approach on the ImageNet-1K dataset and discuss the results.

5.1 Baseline Architecture Analysis

Table 1 details the evaluation results. While not a strictly one-to-one comparison, the baseline difference between DeiT and SReT clearly depicts the implications of a recursive architecture. SReT requires fewer parameters while achieving better accuracy and lower FLOPs than DeiT. However, SReT shows significantly lower throughput and higher PAM across all fronts. This illustrates the intrinsic trade-off of recursive architectures: higher accuracy and theoretical efficiency come at the cost of throughput and memory due to the repeated execution of attention mechanisms.

5.2 Evaluation of Token Reduction Schedules

We first apply the constant reduction schedule as an initial baseline. A reduction rate of $r_{\text{const}} = 10$ results in a 9.6% increase in throughput and a 3.3% decrease in PAM on the GPU at batch size 128. However, we also see a 6.41% decrease in accuracy, a loss that outweighs the gains. A more aggressive rate of $r_{\text{const}} = 20$ shows massive accuracy drops, invalidating any efficiency benefits. This initial result validates our hypothesis: a constant reduction schedule is intrinsically unsuitable for recursive architectures, as continuously removing tokens prevents the sequence representation from stabilizing.

As an improvement, we explore the linear reduction schedule. A reduction rate of $r_{\text{lin}} = 10$, which removes a nearly identical

Table 1. **Performance results of the proposed approach evaluated on ImageNet-1K.** GPU metrics are measured on an NVIDIA RTX 4060 Ti, while CPU metrics are measured on an Intel Core Ultra 9 285K constrained to four threads.

Configuration	Properties			GPU Metrics				CPU Metrics
	Top-1 Acc. (%)	Params. (M)	FLOPs (G)	Throughput (img/s)		Peak Act. Mem. (MB)		Throughput (img/s)
				BS = 128	BS = 1	BS = 128	BS = 1	BS = 1
Standard ViT								
DeiT-Tiny-Distill [18]	74.40	5.91	2.17	1825.88	730.20	227.20	1.76	141.79
Recursive ViT								
SReT-Tiny-Distill [17]	77.42	4.76	1.91	1072.86	223.58	795.76	6.22	83.32
ToMe: Constant Reduction								
$r_{\text{const}} = 10$	71.01 -6.41	–	1.32 (-30.9%)	1176.27 (+9.6%)	110.24 (-50.7%)	769.79 (-3.3%)	6.01 (-3.4%)	75.97 (-8.8%)
$r_{\text{const}} = 20$	41.06 -36.36	–	1.06 (-44.5%)	1331.23 (+24.1%)	112.56 (-49.7%)	756.22 (-5.0%)	5.91 (-5.0%)	83.16 (-0.2%)
ToMe: Linear Reduction								
$r_{\text{lin}} = 10$	74.64 -2.78	–	1.46 (-23.6%)	1177.32 (+9.7%)	116.12 (-48.1%)	756.22 (-5.0%)	5.91 (-5.0%)	73.06 (-12.3%)
$r_{\text{lin}} = 20$	14.87 -62.55	–	1.07 (-44.0%)	1427.14 (+33.0%)	117.21 (-47.6%)	729.46 (-8.3%)	5.70 (-8.4%)	85.30 (+2.4%)
ToMe: Exponential Reduction ***(Ours)***								
$r_{\text{init}} = 0.10, \alpha = 0.6$	76.35 -1.07	–	1.61 (-15.7%)	1186.62 (+10.6%)	128.30 (-42.6%)	664.75 (-16.5%)	5.19 (-16.6%)	76.44 (-8.3%)
$r_{\text{init}} = 0.25, \alpha = 0.0$	75.95 -1.47	–	1.49 (-22.0%)	1368.67 (+27.6%)	174.37 (-22.0%)	489.38 (-38.5%)	3.90 (-37.3%)	84.15 (+1.0%)
$r_{\text{init}} = 0.40, \alpha = 0.2$	69.71 -7.71	–	1.13 (-40.8%)	1886.90 (+75.9%)	147.91 (-33.8%)	391.51 (-50.8%)	3.90 (-37.3%)	88.12 (+5.8%)

total number of tokens as its constant counterpart, achieves a 9.7% increase in throughput and a 5.0% decrease in PAM on the GPU at batch size 128, while only suffering a 2.78% accuracy drop. This confirms that shifting the bulk of the reduction to the earlier layers improves recursive performance. However, for larger values of r_{lin}, the accuracy drops much faster than the constant schedule. This is likely caused by aggressive, continuous token removal in the critical early feature-extraction stages.

The results of the proposed exponential reduction schedule confirm that it delivers the optimal balance. It suffers the least accuracy drops while achieving considerable boosts in both throughput and PAM. Most notably, the configuration $r_{\text{init}} = 0.25, \alpha = 0.0$ results in a 27.6% increase in throughput and a 38.5% decrease in PAM on the GPU at batch size 128, with an accuracy drop of only 1.47%.

Although these results, detailed in Table 1, establish the dominance of the exponential schedule, verifying the fairness of this comparison requires analyzing the actual reduction trajectories. Figure 5 isolates three specific schedule configurations that remove a nearly identical total number of tokens.

In Figure 5(a), the constant schedule ($r_{\text{const}} = 10$) removes a total of 230 tokens and achieves 71.01% accuracy. In Figure 5(b), the linear schedule ($r_{\text{lin}} = 10$) removes 223 tokens and achieves 74.64% accuracy. Finally, in Figure 5(c), the exponential schedule ($r_{\text{init}} = 0.10, \alpha = 0.6$) removes 235 tokens and achieves 76.35% accuracy. The exponential schedule significantly outperforms the others. This explicitly proves our hypothesis: recursive models require heavy initial reduction followed by a stabilized, near-fixed sequence length to successfully reuse weights across deeper iterations.

Furthermore, Figure 5 highlights the discrepancy between the requested reduction (r_{target}) and the actual reduction (r_{safe}). This difference is caused by the merging constraints of SGA and BSM (Equation 3). Notably, under these rigid boundaries, the exponential schedule shows the least amount of mathematical friction, with the least amount of constraint adjustments.

5.3 Parameter Trade-offs and Pareto Analysis

Having established the exponential schedule as the most effective and structurally compliant method, we now evaluate the effects of its parameters. As shown in Table 1, varying the parameters indicates an inverse relationship between accuracy and throughput. To explore this behavior, we plot the effect of the decay parameter (α) against fixed values of r_{init} on the GPU at batch size 128 in Figure 6.

The data indicates that as α increases, accuracy decreases while throughput increases. In contrast, PAM is not affected by α and is strictly dependent on r_{init}: larger initial token reductions directly lead to larger memory reductions. Thus, r_{init} must be large enough for memory reduction but small enough to avoid high initial accuracy drops, while α must be tuned to retain both accuracy and throughput. To identify the optimal balance across all combinations, we visualize the Pareto frontier in Figure 7.

The Pareto analysis evaluates r_{init} values ranging from 0.05 to 0.5 (in steps of 0.05) against α values ranging from 0.0 to 1.0 (in steps of 0.1). The frontier confirms that, out of the tested configurations, the single-shot reduction ($r_{\text{init}} = 0.25, \alpha = 0.0$) provides the optimal balance between accuracy retention, throughput acceleration, and PAM reduction. While derived from the exponential schedule, this

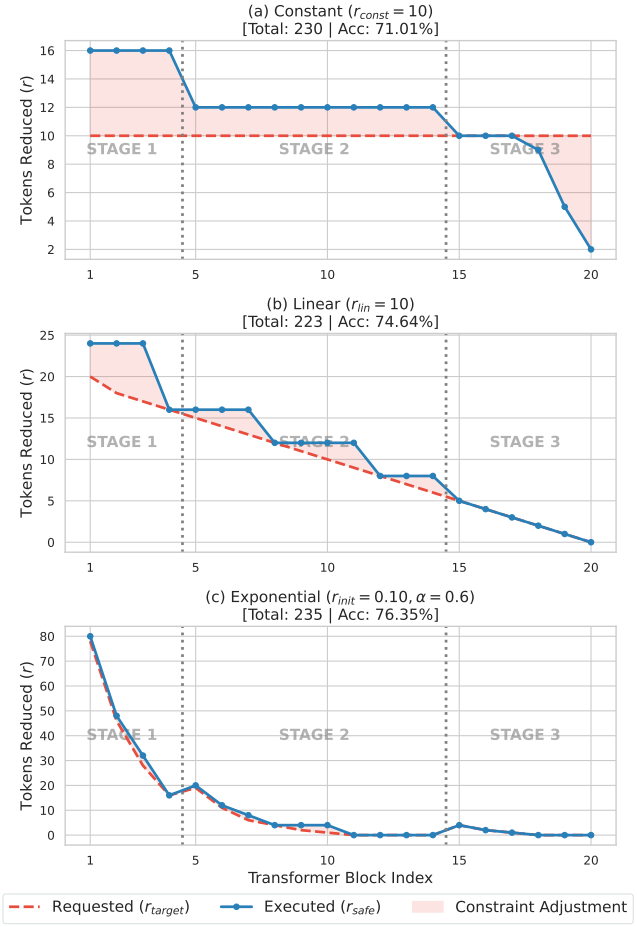

Fig. 5. **Token reduction trajectories per schedule.** The total token removal is comparable, but the exponential schedule performs best. The constraint adjustments illustrate the merging constraints of Equation 3.

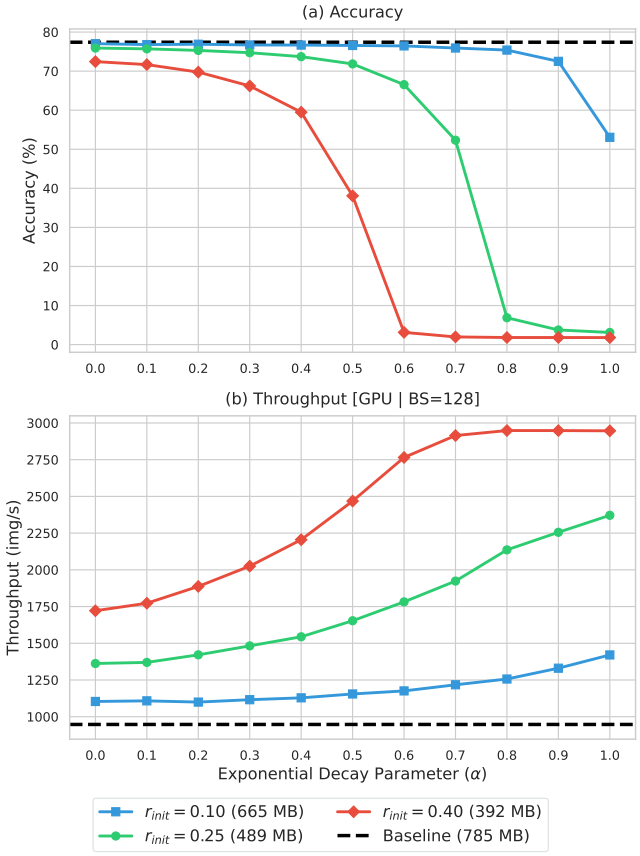

Fig. 6. **Impact of the exponential decay parameter (α) on accuracy and GPU throughput at $BS = 128$.** The trajectories illustrate the accuracy-throughput trade-off as α increases.

indicates that single-shot token reduction yields better results for recursive architectures than gradual decay.

Formally, the inclusion of the boundary conditions $\alpha = 0.0$ and $\alpha = 1.0$ extends the exponential reduction schedule into a unified reduction schedule. While the open interval $0 < \alpha < 1$ would be the true exponential reduction, the empirical dominance of the $\alpha = 0.0$ boundary exposes a vital structural characteristic of recursive weight-sharing models: they are highly sensitive to variance in token sequence lengths in the deeper recursive loops. Frontloading the reduction using a single-shot reduction schedule eliminates this variance, ensuring maximal hardware throughput while preserving the semantic integrity of the tokens.

5.4 Performance Bottlenecks at Batch Size 1

Finally, the performance analysis shifts significantly when operating at a batch size of 1, which is the most realistic sequential processing scenario for edge devices. As shown in Table 1 and Figure 8, throughput scaling on both the GPU and CPU for batch size 1 diverges from the GPU results for batch size 128. On the GPU, increasing α strictly decreases throughput. On the CPU, increasing α initially dips performance before eventually recovering.

We attribute this deviation to the algorithmic overhead associated with BSM of ToMe and the restoration of token sequence lengths at the end of each stage. At large batch sizes ($BS = 128$), the parallel compute capabilities of the GPU easily mask this overhead. However, in single-stream inference ($BS = 1$), sequential execution costs dominate execution time, reducing the benefits of having fewer tokens. Exploring and optimizing this low-level hardware execution overhead remains a vital area for future work to fully realize these theoretical edge gains at a batch size of 1.

6 LIMITATIONS AND FUTURE WORK

While we successfully demonstrate the viability of post-training dynamic token reduction in recursive architectures, several limitations present clear directions for future work.

First, our evaluations are constrained to SReT-Tiny-Distill. Having resolved the geometric and algorithmic clashes inherent to this specific architecture, future work could evaluate larger model variants and other recursive ViTs. Benchmarking against a wider array

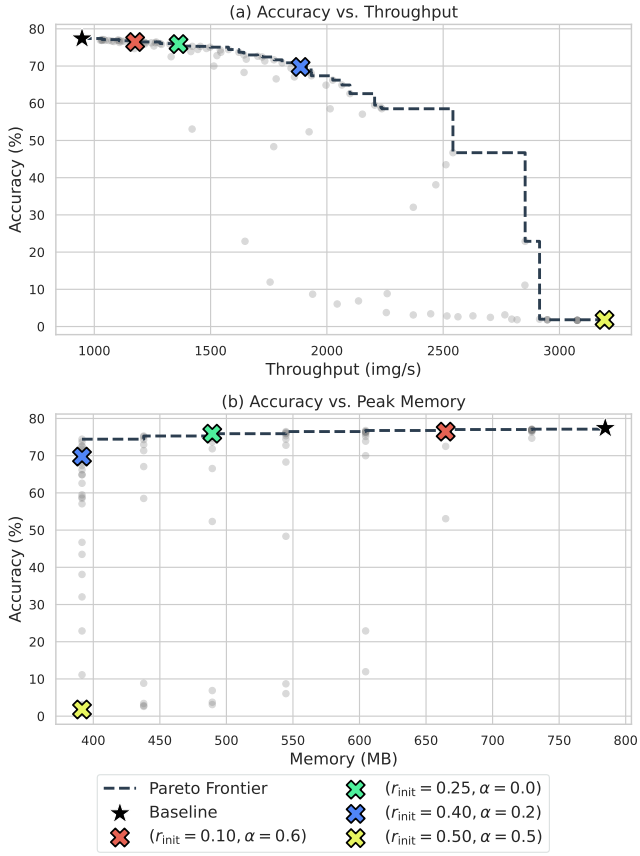

Fig. 7. **Pareto analysis of the exponential reduction schedule parameters (r_{init}, α) evaluated on GPU at $BS = 128$.** The frontier illustrates the trade-offs between accuracy, throughput, and PAM, identifying the ($r_{\text{init}} = 0.25, \alpha = 0.0$) configuration as most optimal.

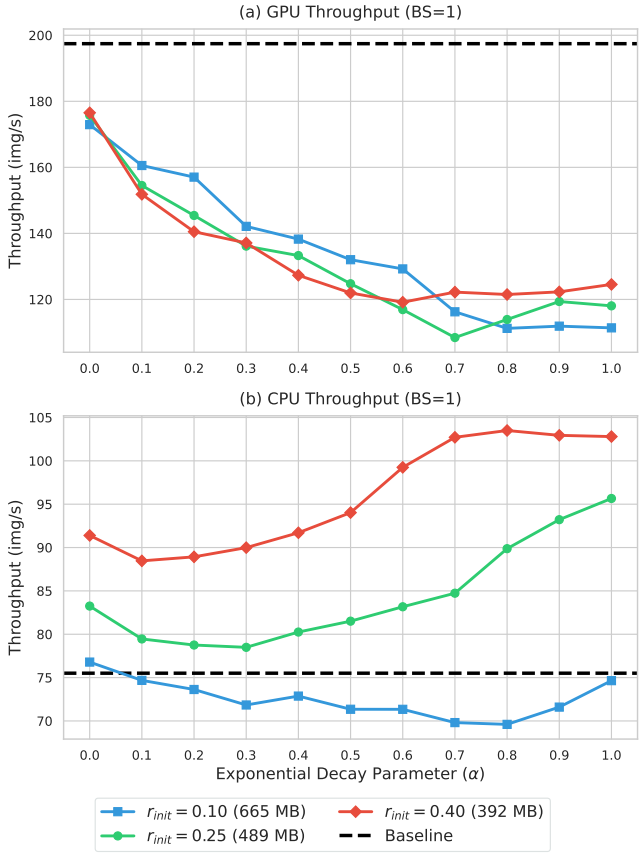

Fig. 8. **Impact of the exponential decay parameter (α) on GPU and CPU throughput at $BS = 1$.** The trajectories illustrate how single-stream inference exposes the throughput overhead of bipartite matching and sequence restoration.

of diverse compression methodologies is also necessary to fully validate generalizability.

Second, single-stream inference ($BS = 1$) reveals throughput bottlenecks on both GPU and CPU. Since edge applications strictly rely on real-time, single-stream processing, full edge-viability remains theoretical until low-level execution overheads are optimized. We attribute these bottlenecks to the algorithmic overhead of BSM and sequence length restoration, and future low-level profiling is required to mitigate this.

Finally, while our approach improves efficiency without retraining, edge deployment of recursive ViTs remains challenging. Future iterations could explore integrating other compression techniques, such as post-training quantization, to achieve more definitive edge viability.

7 CONCLUSION

In this work, we demonstrated that the recursive weight-sharing of SReT and the dynamic token merging of ToMe can be successfully combined for multi-axis compression. By resolving the structural incompatibilities between hierarchical, recursive architectures and dynamic sequence reduction, our approach enables efficient post-training deployment. Benchmarked on ImageNet-1K, our evaluations establish that frontloading an exponential token reduction schedule accelerates GPU throughput by 27.6% and cuts PAM by 38.5%, with a negligible 1.47% accuracy drop at a batch size of 128. While low-level execution bottlenecks currently limit strict single-stream edge deployment, this study successfully provides the foundational baseline required for future optimizations in resource-constrained environments.

ACKNOWLEDGMENTS

The author acknowledges and sincerely thanks dr. ir. Uraz Odyurt and dr. Amirreza Yousefzadeh for their excellent supervision. Their guidance and feedback have been instrumental throughout the research process, and this project would not have been possible without their continued support.

REFERENCES

[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2023. Token Merging: Your ViT But Faster. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2210.09461

[2] Jianjian Cao, Chong Yu, Peng Ye, and Tao Chen. 2026. MADTP++: Bridge the Gap Between Token and Weight Pruning for Accelerating VLTs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 48 (2026), 5180–5194. https://doi.org/10.1109/TPAMI.2025.3650545

[3] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Vol. 35. 16344–16359. https://dl.acm.org/doi/10.5555/3600270.3601459

[4] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal Transformers. *arXiv preprint arXiv:1807.03819* (2018).

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. OpenReview.net. https://openreview.net/forum?id=YicbFdNTTy

[7] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. 2021. Rethinking Spatial Dimensions of Vision Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 11916–11925. https://doi.org/10.1109/iccv48922.2021.01172

[8] Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. 2024. Token Fusion: Bridging the Gap between Token Pruning and Token Merging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 4418–4427. https://doi.org/10.1109/WACV57729.2024.00435

[9] Xinqing Li, Xin He, Xindong Zhang, Ming-Ming Cheng, Lei Zhang, and Yun Liu. 2026. Towards Joint Quantization and Token Pruning of Vision-Language Models. *arXiv preprint arXiv:2604.17320* (2026). https://doi.org/10.48550/arXiv.2604.17320

[10] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. 2022. Not All Patches are What You Need: Expediting Vision Transformers via Token Reorganizations. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2202.07800

[11] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. 2022. FQ-ViT: Post-Training Quantization for Fully Quantized Vision Transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 1173–1179. https://doi.org/10.24963/ijcai.2022/164

[12] Xinyu Liu, Houwen Peng, Ningxin Zheng, Yuqing Yang, Han Hu, and Yixuan Yuan. 2023. EfficientViT: Memory Efficient Vision Transformer with Cascaded Group Attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 636–646. https://doi.org/10.1109/CVPR52729.2023.00069

[13] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 10012–10022. https://doi.org/10.1109/ICCV48922.2021.00986

[14] Sachin Mehta and Mohammad Rastegari. 2022. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2110.02178

[15] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. 2021. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 1393–1405. https://arxiv.org/abs/2106.02034

[16] Zhiqiang Shen. 2022. Sliced Recursive Transformer (SReT). https://github.com/szq0214/SReT. Accessed: 2026-04-28.

[17] Zhiqiang Shen, Zechun Liu, and Eric Xing. 2022. Sliced Recursive Transformer. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 727–744. https://doi.org/10.1007/978-3-031-20053-3_42

[18] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*. PMLR, 10347–10357. https://proceedings.mlr.press/v139/touvron21a.html

[19] Hoai-Chau Tran, Duy M. H. Nguyen, Duy M. Nguyen, Trung-Tin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Y. Zou, Binh T. Nguyen, and Mathias Niepert. 2024. Accelerating Transformers with Spectrum-Preserving Token Merging. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/2405.16148

[20] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. 2023. FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5762–5772. https://doi.org/10.1109/ICCV51070.2023.00532

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 6000–6010. https://dl.acm.org/doi/10.5555/3295222.3295349

[22] Ao Wang, Hui Chen, Zijia Lin, Jungong Han, and Guiguang Ding. 2024. RepViT: Revisiting Mobile CNN from ViT Perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 15909–15920. https://ieeexplore.ieee.org/document/10657330

[23] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. 2021. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 568–578. https://doi.org/10.1109/iccv48922.2021.00061

[24] Lu Yu and Wei Xiang. 2023. X-Pruner: eXplainable Pruning for Vision Transformers. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 24355–24363. https://doi.org/10.1109/cvpr52729.2023.02333

[25] Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. 2022. MiniViT: Compressing Vision Transformers with Weight Multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11415–11424. https://doi.org/10.1109/CVPR52688.2022.01114

A AI STATEMENT

During the preparation of this work, the author used Gemini 3.5 to understand fundamental concepts and grammatically polish the writing. After using this tool, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.