AVQ-Attention: Adaptive Vector-Quantized Attention

Winfried van den Dool[1,2], Patrick Forré[3,4], Amir Habibian[5], Yuki M. Asano[6], and Max Welling[2]

[1] QUVA Lab, University of Amsterdam, The Netherlands
w.v.s.o.vandendool@uva.nl
[2] AMLab, Informatics Institute, University of Amsterdam, The Netherlands
[3] AI4Science Lab, University of Amsterdam, The Netherlands
[4] Korteweg-de Vries Institute for Mathematics, University of Amsterdam, The Netherlands
[5] Qualcomm AI Research, Amsterdam, The Netherlands
[6] FunAI Lab, University of Technology Nuremberg, Germany

Abstract. The $\mathcal{O}(N^2)$ complexity of attention over N tokens remains a computational bottleneck in transformer models. Vector-Quantized (VQ) attention reduces this to $\mathcal{O}(MN)$ by representing keys with M codewords, but applies uniform codebook capacity regardless of where attention mass concentrates: high-attention regions of key space may be coarsely approximated while low-attention regions waste representational capacity. We propose Adaptive Vector-Quantized (AVQ) Attention, which adaptively allocates codebook capacity based on attention importance. Starting from a small set of codewords, our method identifies the most important codes during the forward pass and refines them with pre-learned child codewords, achieving fine-grained quantization where it matters most while maintaining coarse quantization elsewhere. We develop an implementation using custom Triton kernels that enables the full adaptive refinement process, including importance scoring, child codeword insertion, and parent contribution replacement, to be carried out within the tiled computation paradigm of Flash Attention with minimal overhead. Our approach maintains $\mathcal{O}(MN)$ complexity while achieving improved accuracy-efficiency trade-offs compared to fixed-codebook VQ-attention.

Keywords: Efficient Attention · Vector Quantization · Adaptive Codebook

1 Introduction

Attention mechanisms have become the cornerstone of modern deep learning, enabling transformers [32] to capture rich interactions between input tokens. However, this expressiveness comes at a computational cost: computing attention between all token pairs requires $\mathcal{O}(N^2)$ operations for sequences of length N, making long-sequence processing increasingly prohibitive as models and datasets

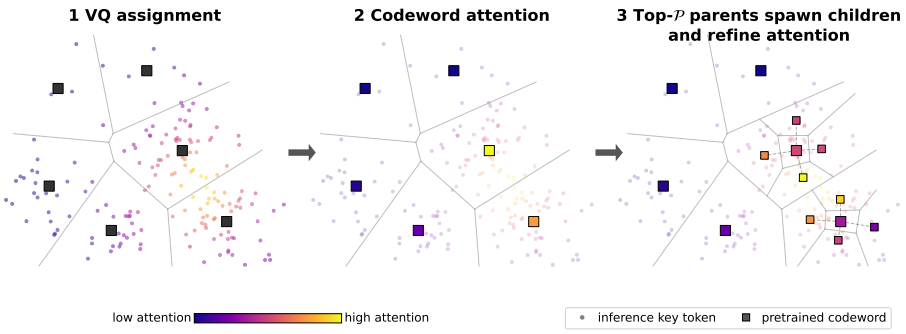

Fig. 1: key-space visualization of Adaptive VQ-Attention. Small dots represent inference key tokens, colored by how much attention they receive from a given query block; large squares represent pre-trained codewords. **(1)** Keys are assigned to their nearest codeword via vector quantization. **(2)** Per-codeword importance is computed from the attention mass each codeword receives. **(3)** The top-$\mathcal{P}$ most important parents are selected and their pre-learned children are spawned, refining codebook resolution in high-attention regions while leaving low-attention regions at the coarser parent level.

scale. Flash Attention [7] addressed the $\mathcal{O}(N^2)$ memory bandwidth bottleneck through tiling strategies that keep intermediate results in fast SRAM. While this speeds up training and inference on longer sequences, the fundamental computational complexity remains quadratic. Unlike memory traffic, reducing computational cost without approximation is fundamentally impossible: N unique tokens communicating pairwise inherently generate N^2 interactions. The challenge therefore becomes how to best trade approximation error for computational efficiency.

Various approaches have explored this trade-off, from sparse attention patterns that limit which tokens interact [1, 4, 29] to token merging methods that reduce the number of tokens processed [2]. These methods generally modify the transformer's structure by dropping or restricting token interactions. Vector-quantized attention offers an alternative by clustering keys into M representative codewords where $M < N$ implies reducing complexity to $\mathcal{O}(MN)$. This approach has shown promise [19], but introduces a new challenge: the codebook must be chosen in advance, applying uniform quantization quality across all regions of the key space. During inference, regions receiving high attention mass may suffer from coarse approximation, while low-attention regions waste representational capacity on unnecessary precision. This challenge parallels a well-studied problem in quantization research. Traditional adaptive quantization methods allocate more representational capacity, i.e. bitwidth, to important features while applying coarser quantization elsewhere. These techniques have proven effective across various domains, from image compression to neural network quantization, by concentrating limited resources where they matter most [9, 35, 36].

Inspired by adaptive quantization techniques, we propose Adaptive Vector Quantized (AVQ) Attention. Attention naturally provides an importance signal: by

measuring how much attention mass each cluster in key space receives during the forward pass, we obtain a direct importance measure—without the need to design one separately, as in standard adaptive quantization. We use this signal to dynamically allocate codebook capacity where it matters most. Concretely, we start from a set of parent codewords and compute VQ-Attention while extracting per-codeword importance scores. The top-$\mathcal{P}$ most important parents are then refined with their pre-learned child codewords, creating finer quantization in high-attention regions of key space while leaving low-attention regions at the coarser parent level. This refinement adapts dynamically to each input, concentrating codebook capacity based on where that specific input's attention mass falls (see Fig. 1).

Implementing this adaptive refinement efficiently within Flash Attention's tiled framework is non-trivial: the attention weights needed to determine per-codeword importance scores are never materialized—they only exist momentarily at the tile level and are immediately consumed. However, we show that Flash Attention's incremental computation can in fact synergize with our approach: just as Flash Attention builds up attention over blocks of keys via online softmax, AVQ-Attention first computes attention over parent codewords, then incrementally refines it with blocks of children. A geometric constraint on the codebook—each parent equals the mean of its children—allows parent contributions to be recovered directly from the child logits already being computed, enabling efficient in-register correction without revisiting parent codewords. We show that this maintains $\mathcal{O}(MN)$ complexity matching standard VQ-attention while enabling adaptive allocation.

We experimentally validate AVQ-Attention on image classification (ImageNet-1k), semantic segmentation (ADE20K), and high-resolution image generation (Stable Diffusion). We demonstrate improved accuracy-efficiency trade-offs over fixed-codebook VQ-attention and competitiveness with a range of existing efficient-attention methods. Moreover, we show that (A)VQ-attention can be applied post-hoc to pretrained transformers and fine-tuned in a small number of epochs, analogous to quantization-aware training for model compression. In summary, our contributions include the following.

- A hierarchical VQ-codebook with a constrained parent-child structure, and a training procedure that learns the codebook end-to-end.
- A Flash Attention-compatible mechanism that efficiently increments refinements of the attention output.
- Custom Triton [31] kernels that improve VQ-attention wall-clock performance by fusing operations and minimizing memory traffic, applicable to both flat and adaptive codebooks.
- A linear complexity attention variant combining these techniques to dynamically allocate compute capacity based on attention importance, adapting to each input at inference time. We validate this experimentally, demonstrating improved accuracy-efficiency trade-offs compared to fixed-codebook VQ-attention.

2 Related Work

Adaptive quantization allocates representational capacity unevenly based on importance, applying higher precision where it matters most [35, 36]. While these methods typically require a separate mechanism to estimate importance, in the attention setting we can use attention weights directly, alleviating the need for sensitivity analysis [36], learned metrics [30], or auxiliary models [9]. This insight has also been used to guide mixed-precision KV-cache quantization, allocating higher bitwidth to tokens receiving more attention [39]. However, adaptivity in the context of VQ-attention can be significantly more effective: rather than varying the precision of N token interactions, VQ-attention reduces the number of interactions itself from N to M codewords, directly addressing the complexity bottleneck.

Vector-Quantization has been explored as a means to reduce the quadratic complexity of attention by clustering keys into a smaller set of representative codewords. By attending over codewords rather than individual keys, vector-quantized (VQ) attention reduces computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(MN)$, where $M \ll N$ is the codebook size. Prior work on VQ-attention [19] demonstrates that this approach can achieve favorable efficiency–accuracy trade-offs, but relies on a fixed codebook that applies uniform quantization quality across the key space. We build most directly on this line of work, extending VQ-attention in two directions. First, we implement VQ-attention using custom Triton kernels compatible with Flash Attention's tiled computation, minimizing memory traffic. Prior VQ-attention work targets the long-context regime where $N \gg M$ and the $\mathcal{O}(MN) \ll \mathcal{O}(N^2)$ complexity gap alone provides large speedups. We first identify opportunities to fuse the sequential steps of VQ-attention into kernels that keep memory bandwidth low, widening the gap with standard attention at large N as the codebook grows. Additionally, it makes VQ-attention competitive already at moderate sequence lengths where the complexity gap is less pronounced and memory bandwidth plays a more significant role. Second, and more centrally, we replace the fixed codebook with an adaptive one that dynamically allocates additional codewords to regions of the key space receiving high attention mass, improving approximation quality where it matters most.

Clustered attention approximates full attention by grouping queries into clusters and computing attention once per cluster centroid, achieving linear complexity [34]. This assumes queries within a cluster produce similar attention patterns. Our approach clusters keys rather than queries, so every query retains its own attention pattern over the compressed codeword set. Token merging (ToMe) merges similar tokens into aggregated representations, either permanently reducing the token count and compounding the approximation across layers [2], or requiring per-block unmerging to restore the full token set [3]. Moreover, finding merge candidates requires computing pairwise token similarities, which is itself quadratic.

Sparse attention methods reduce the quadratic cost of attention by restricting computation to a subset of token–token interactions, whether through fixed patterns [1, 4], content-based routing [29], locality-sensitive hashing [16], attention-derived selection [38], KV-cache eviction [17], or hierarchical key selection for long-context inference [14, 23]. In contrast, our method preserves dense attention semantics: all keys contribute through their codeword, effectively replacing a binary keep-or-discard decision with a graduated one where less important regions receive coarser approximation rather than being removed entirely.

Linear and low-rank attention methods replace the softmax kernel with a decomposable feature map, enabling $\mathcal{O}(N)$ complexity via associativity of matrix multiplication [15]. Low-rank methods such as Linformer [37] project keys and values to a lower-dimensional space. Both families modify the attention mechanism itself, whereas our approach preserves standard softmax attention.

3 Preliminaries

3.1 Self-Attention

Standard scaled dot-product attention [32] computes outputs as weighted averages of value vectors, where weights are determined by query-key similarities:

$$Y_i = \frac{\sum_{j=1}^{N} \exp(Q_i K_j^\top) V_j}{\sum_{l=1}^{N} \exp(Q_i K_l^\top)} \tag{1}$$

where $Q_i, K_j, V_j \in \mathbb{R}^d$ are query, key, and value vectors for tokens i and j. This formulation requires $\mathcal{O}(N^2 d)$ operations to compute all query-key dot products. It is common to scale key-query dot products by $1/\sqrt{d}$; we omit this for clarity.

3.2 Vector-Quantized Attention

VQ-attention reduces complexity by replacing keys with quantized representations from a codebook $\{C_a\}_{a=1}^{M}$ where $M \ll N$. Each key is assigned to its nearest codeword (with slight abuse of notation, we use a both as the assignment function and as a codeword index):

$$\hat{K}_j = C_{a(K_j)}, \quad \text{where} \quad a(K_j) = \arg\min_a \|C_a - K_j\|^2 \tag{2}$$

We can rewrite attention by grouping keys that map to the same codeword. Define:

$$n_a = |\{j : a(K_j) = a\}|, \quad \bar{V}_a = \sum_{j:a(K_j)=a} V_j \tag{3}$$

as the count and total aggregated value for codeword a. Then

$$Y_i \approx \frac{\sum_{a=1}^{M} \exp(Q_i C_a^\top) \bar{V}_a}{\sum_{a=1}^{M} \exp(Q_i C_a^\top) n_a}, \tag{4}$$

where the approximation error comes solely from key quantization. Indeed, the equivalence to quantized-key attention follows from regrouping the sums:

$$\begin{aligned}\sum_j \exp(Q_i K_j^\top) V_j &\approx \sum_j \exp(Q_i \hat{K}_j^\top) V_j \\ &= \sum_a \sum_{j:a(K_j)=a} \exp(Q_i C_a^\top) V_j \\ &= \sum_a \exp(Q_i C_a^\top) \bar{V}_a\end{aligned}$$

and similarly for the denominator.

3.3 Flash Attention

Flash Attention [6, 7] computes attention without materializing the full $N \times N$ matrix $\exp(QK^\top)$, instead processing tiles that fit in on-chip SRAM. The algorithm maintains running numerators and denominators, scaled by a running maximum for numerical stability [24]. As we will see, this incremental structure synergizes naturally with our adaptive refinement. We adopt tile-level notation for the following sections: I and J denote contiguous index sets for query and key (or codeword) tiles respectively, with each tile assumed to fit in SRAM. Rewriting Eq. (1), the actual computation on hardware more closely resembles:

$$\begin{aligned}A_{IJ} &= \exp(Q_I K_J^\top) \\ X_I(J) &= A_{IJ} V_J \\ Z_I(J) &= \sum_{j \in J} A_{Ij} \\ Y_I &= \frac{\sum_J X_I(J)}{\sum_J Z_I(J)}.\end{aligned}$$

4 Method

4.1 Hierarchical Codebook

We describe the single-head, single-layer case; multi-head attention maintains one codebook per head. The codebook consists of M parent codewords $\{C_j\}_{j=1}^M$, each with $\mathcal{C}$ children $\{C_{j,c}\}_{c=1}^{\mathcal{C}}$, giving $M(1+\mathcal{C})$ codewords in total, as children supplement rather than replace their parents.

Training. All codewords are learned via online k-means [22] with exponential moving averages (EMA) [25]. At each training step, keys are first quantized to their nearest parent, and parent centroids are updated via EMA. Each parent's assigned keys are then further quantized among its $\mathcal{C}$ children, with the parent itself remaining as an option—keys that are already well-represented by the parent need not move to a child. Child centroids are updated via EMA on their respectively assigned keys (see Fig. 2).

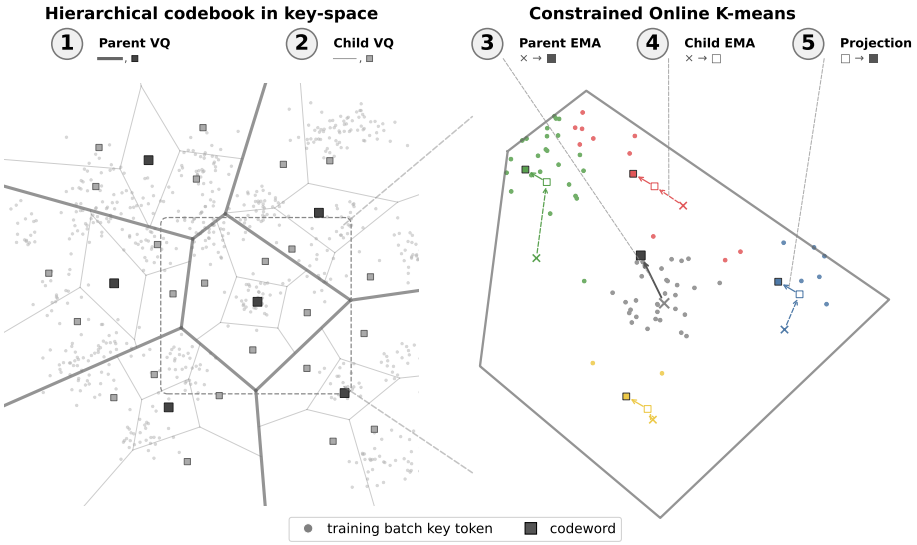

Fig. 2: Hierarchical codebook in key space, illustrating the five online learning steps. **Left:** An incoming training batch (gray dots) arrives at the existing codebook. ① Keys are assigned to the nearest parent codeword (thick Voronoi boundaries), then ② further quantized to child codewords within each parent's cell (thin boundaries). **Right:** Zoom into one parent's cell showing the codebook update. ③ The parent updates its position via EMA over all keys in its cell. ④ Each child updates independently via count-weighted EMA. ⑤ Children are projected to restore the constraint $C_p = \frac{1}{\mathcal{C}} \sum_c C_{p,c}$; heavier children (larger n_c) resist displacement more (Supp. A).

Parent-child constraint. We impose the constraint that each parent equals the mean of its children:

$$C_p = \frac{1}{\mathcal{C}} \sum_{c=1}^{\mathcal{C}} C_{p,c} \tag{5}$$

Since the unconstrained EMA updates on children will generally violate this, we project child positions back onto the constraint surface after each update via a closed-form mass-weighted projection (see Supp. A). This constraint is central to our method: it enables efficient removal of parent attention contributions at inference time, as we show in Sec. 4.3.
At inference, all codeword positions are fixed.

4.2 Vector Quantization for (A)VQ-Attention

Before computing attention, each key must be assigned to a codeword. Unlike standard vector quantization, which only requires assignments, the vector quantization step in VQ-attention must additionally aggregate the values and counts per codeword ($\bar{V}_a$ and n_a in Eq. (3)), since these serve as the inputs to the subsequent attention computation. We compute both assignments and aggregates

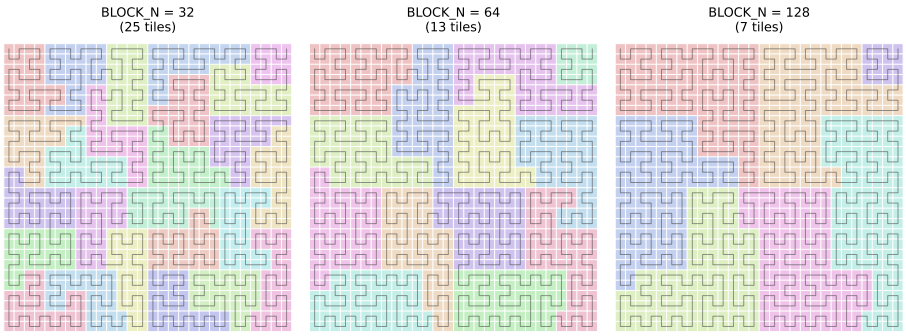

Fig. 3: Spatial locality of query tiles under Gilbert curve reordering [33] on a 28×28 patch grid. Tokens are reordered along a Gilbert space-filling curve (gray line) so that contiguous tiles (colored regions) form spatially compact 2D regions, regardless of tile size. Since each tile independently selects which parents to refine, spatial compactness ensures that queries sharing a refinement decision attend to similar regions.

for the full codebook tree in a single fused kernel pass over the keys. Each key is first assigned to its nearest parent among M_0 root codewords. Then, it is compared against only the $\mathcal{C}$ children of its assigned parent, and reassigned if a child is closer than the (cached) parent distance. At each level, the key's value is scattered to the assigned codeword, accumulating $\bar{V}_a$ and n_a.
The tree structure makes this efficient: the cost per key is $\mathcal{O}(M_0 + \mathcal{C})$ distance computations, yielding a codebook with $M_{\text{total}} = M_0(1+\mathcal{C})$ codewords. One may be tempted to defer the computation of child codeword aggregates until after the most important parents have been identified, computing only for children that will actually be used. However, precomputing the full tree brings two key advantages:

- **Reduced HBM traffic.** The subsequent attention kernel can remain fully fused: after computing attention over parent codewords and determining importance, child aggregates are immediately available, and the kernel can proceed to refine without interrupting to perform additional vector quantization. This avoids writing and re-reading intermediate accumulators to global memory.
- **Per-query-tile amortization and adaptivity**. The precomputed aggregates are shared across all query tiles, amortizing the VQ cost. Each tile may then independently select different parents for refinement, making the method adaptive not only per input, but per query tile. To ensure that queries within a tile attend to similar spatial regions, we reorder tokens along a Gilbert space-filling curve [33], a generalization of the Hilbert curve to non-power-of-two grids. This produces spatially compact tiles regardless of tile size, allowing different regions of the input to refine different parts of the codebook (see Fig. 3, Supp. H).

Algorithm 1 VQ Precompute kernel.

Input: $K, V \in \mathbb{R}^{BH \times N \times D}$, $C \in \mathbb{R}^{H \times M_{\text{total}} \times D}$ with $M_{\text{total}} = M_0(1 + \mathcal{C})$
Output: $\bar{V} \in \mathbb{R}^{BH \times M_{\text{total}} \times D}$, $n \in \mathbb{R}^{BH \times M_{\text{total}}}$

Initialize $\bar{V} = 0$, $n = 0$
for each (bh, Block_N) **in parallel do** ▷ Separate GPU programs
  $\mathbf{k} = K[bh, \text{Block}_N]$ ▷ $[\text{Block}_N, D]$, stays in registers
  $\mathbf{v} = V[bh, \text{Block}_N]$ ▷ $[\text{Block}_N, D]$, stays in registers
  // Parent assignment (tiled over M_0 if codebook exceeds SRAM)
  $\mathbf{c}_p = C[h,\ 0\!:\!M_0]$ ▷ $[M_0, D]$, loaded into SRAM
  $\mathbf{d} = \|\mathbf{k} - \mathbf{c}_p\|^2$ ▷ $[\text{Block}_N, M_0]$ pairwise
  $\mathbf{a} = \arg\min(\mathbf{d}, \text{axis=1})$; $\mathbf{d}_{\text{best}} = \min(\mathbf{d}, \text{axis=1})$ ▷ $[\text{Block}_N]$ each
  $\bar{V}[bh, \mathbf{a}]$ += $\mathbf{v}$; $n[bh, \mathbf{a}]$ += 1 ▷ Atomic add
  // Child assignment — children of parent m at $C[h,\ M_0 + m\mathcal{C}\!:\!M_0 + (m{+}1)\mathcal{C}]$
  $\mathbf{c}_0 = M_0 + \mathbf{a} \cdot \mathcal{C}$ ▷ $[\text{Block}_N]$ first-child index per key
  $\mathbf{d}_{\text{child}} = \text{full}(\text{Block}_N,\ \infty)$; $\mathbf{c}^* = \mathbf{c}_0$
  for $c = 0, \ldots, \mathcal{C} - 1$ **do**
    $\mathbf{c}_{\text{code}} = C[h,\ \mathbf{c}_0 + c]$ ▷ $[\text{Block}_N, D]$
    $\mathbf{d}_c = \|\mathbf{k} - \mathbf{c}_{\text{code}}\|^2$ ▷ $[\text{Block}_N]$
    where $\mathbf{d}_c < \mathbf{d}_{\text{child}}$: $\mathbf{d}_{\text{child}} = \mathbf{d}_c$, $\mathbf{c}^* = \mathbf{c}_0 + c$
  end for
  where $\mathbf{d}_{\text{child}} < \mathbf{d}_{\text{best}}$: $\bar{V}[bh, \mathbf{c}^*]$ += $\mathbf{v}$; $n[bh, \mathbf{c}^*]$ += 1 ▷ Masked atomic add
end for

VQ-attention introduces several sequential operations — quantization, value aggregation, and attention — with intermediate results passing through global memory. While $\mathcal{O}(MN)$ complexity guarantees asymptotic efficiency, we improve wall-clock performance by fusing quantization and value aggregation into a single kernel that keeps each key in registers across both parent and child assignment. Each key's value is then scattered directly to its assigned codeword's accumulator via atomic operations at the tile level. The reduced memory traffic becomes increasingly important as N and M grow, and additionally allows VQ-attention to compete with Flash Attention already at moderate sequence lengths where the complexity gap alone is insufficient. While we present this for AVQ-attention, the fused VQ kernel is equally an improvement for standard VQ-attention. We provide pseudocode in Algorithm 1.

4.3 Computing (A)VQ-Attention with Flash Attention

Tiled VQ-attention. The VQ-attention formulation maps directly to Flash Attention's tiling strategy. Continuing with the notation from Sec. 3.3, we replace K_J with C_J, so that indices J and j now refer to codewords rather than keys. We write $\bar{V}_j = \sum_{k:a(K_k)=j} V_k$ for the aggregated values per codeword. The key difference from standard attention is that each codeword represents n_j keys, so the denominator accumulates counts rather than row sums:

$$\begin{aligned} A_{IJ} &= \exp(Q_I C_J^\top) \\ X_I(J) &= A_{IJ} \bar{V}_J \\ Z_I(J) &= A_{IJ} n_J \\ \bar{X}_I &= \sum_J X_I(J), \quad \bar{Z}_I = \sum_J Z_I(J) \\ Y_I &= \bar{X}_I \,/\, \bar{Z}_I \end{aligned} \tag{6}$$

where n_j denotes the count of keys quantized to codeword C_j. The accumulators $\bar{X}_I$ and $\bar{Z}_I$ are built up incrementally across tiles using the online softmax trick for numerical stability, as in Flash Attention. We leave the subtraction of a stable maximum in the exponent implicit throughout the text; the pseudocode in Algorithm 2 shows the full computation and Supp. D discusses the choice of running maximum in the presence of codeword counts.

Codeword importance. We can use A_{IJ} and the accumulated denominator $\bar{Z}_I$ to extract per-codeword importance scores. We define importance for each query tile I as:

$$w_j(I) = \sum_{i \in I} \frac{A_{ij} \cdot n_j}{\bar{Z}_i} \tag{7}$$

where $\bar{Z}_i$ is the i-th element of $\bar{Z}_I$. Importance is thus computed from the denominator that the online softmax accumulation already maintains, at minimal extra cost. When M_0 is small enough that all codewords fit in SRAM, $\bar{Z}_i$ is exact after a single pass; for larger M_0 requiring tiling over codewords, an approximate denominator can be used (see Supp. B).

Child spawning. For each query tile I, the top-$\mathcal{P}$ parents by importance are selected for refinement. For each selected parent, its $\mathcal{C}$ (contiguous) children are looked up in the codebook tree. Child aggregated values $\bar{V}_c$ and counts n_c are immediately available, having been precomputed by the VQ kernel (Algorithm 1). Together with the child codewords $C_{p,c}$ loaded from the codebook, this provides everything needed for child attention.

Child attention. When children are added, some keys shift from their originally assigned parent to a closer child codeword. The parent's aggregated values $\bar{V}_p$ and counts n_p no longer fully represent these keys, so its contribution in the accumulators $\bar{X}_I$ and $\bar{Z}_I$ must be corrected. Crucially, we want to avoid recomputing the parent logits $S_{ip} := Q_i C_p^\top$ for two reasons: it would cost extra FLOPs, and it would require keeping parent codewords in SRAM while processing children. We structure the computation as a tiled Flash Attention pass where the first tile(s) consist of parents and subsequent tiles consist of children. To correct the parent contribution without revisiting it, we exploit the parent-child constraint (Eq. (5)): since $C_p = \frac{1}{\mathcal{C}} \sum_c C_{p,c}$, we have

$$S_{ip} = Q_i C_p^\top = \frac{1}{\mathcal{C}} \sum_{c=1}^{\mathcal{C}} Q_i C_{p,c}^\top = \frac{1}{\mathcal{C}} \sum_{c=1}^{\mathcal{C}} S_{ic} \tag{8}$$

so parent logits are recovered directly from the child logits already being computed. Since children are stored contiguously in the codebook, the sum in Eq. (8) reduces over adjacent entries in a register tile and adds negligible cost. We then define the *correcting attention*:

$$\Delta A_{ic} = \exp(S_{ic}) - \exp(S_{ip}) \tag{9}$$

Updating the accumulators with ΔA in place of A implicitly removes the parent's contribution for keys that moved to children and replaces it with their respective child's attention weight (derivation in Supp. C). This requires only a single dot product per parent tile. Pseudocode for the full fused attention kernel is given in Algorithm 2.

Algorithm 2 Flash AVQ-Attention kernel.

Input: $Q \in \mathbb{R}^{BH\times N\times D}$, $C \in \mathbb{R}^{H\times M_{\text{total}}\times D}$, $\bar{V} \in \mathbb{R}^{BH\times M_{\text{total}}\times D}$, $n \in \mathbb{R}^{BH\times M_{\text{total}}}$
Output: $Y \in \mathbb{R}^{BH\times N\times D}$

for each (bh, Block_N) **in parallel do** ▷ Separate GPU programs
  $\mathbf{q} = Q[bh, \text{Block}_N]$ ▷ $[\text{Block}_N, D]$, stays in registers
  $\mathbf{c} = C[h,\ 0{:}M_0]$; $\bar{\mathbf{v}} = \bar{V}[bh,\ 0{:}M_0]$; $\mathbf{n} = n[bh,\ 0{:}M_0]$ ▷ Into SRAM
  $\mathbf{s} = \mathbf{q}\,\mathbf{c}^\top$ ▷ $[\text{Block}_N, M_0]$ logits
  $\mathbf{m} = \max_{j:\,\mathbf{n}_j>0} \mathbf{s}_{:,j}$ ▷ Stable max (empty codes excluded)
  $\mathbf{A} = \exp(\mathbf{s} - \mathbf{m})$; $\mathbf{A}_{:,j} = 0\ \ \forall\, j{:}\,\mathbf{n}_j{=}0$
  $\bar{\mathbf{x}} = \mathbf{A}\,\bar{\mathbf{v}}$; $\bar{\mathbf{z}} = \sum_j \mathbf{A}_{:,j}\cdot\mathbf{n}_j$
  $\mathbf{w}_j = \sum_i \mathbf{A}_{ij}\,\mathbf{n}_j \,/\, \bar{\mathbf{z}}_i$ ▷ $[M_0]$ importance (Eq. (7))
  // Top-$\mathcal{P}$ selection and child refinement
  $\mathcal{S} = \text{top-}\mathcal{P}(\mathbf{w})$ ▷ Selected parent indices
  for each selected parent $p \in \mathcal{S}$ **do** ▷ Optionally in $\lceil \mathcal{P}/\text{Block}_{\mathcal{P}} \rceil$ tiles
    Load $\mathbf{c}_c$, $\bar{\mathbf{v}}_c$, $\mathbf{n}_c$ for children of p
    $\mathbf{s}_c = \mathbf{q}\,\mathbf{c}_c^\top$ ▷ $[\text{Block}_N, \mathcal{C}]$ child logits
    // Online softmax merge
    $\mathbf{m}' = \max\big(\mathbf{m},\ \max_{c:\,\mathbf{n}_c>0} \mathbf{s}_{:,c}\big)$
    $\bar{\mathbf{x}} = \bar{\mathbf{x}}\cdot\exp(\mathbf{m}-\mathbf{m}')$; $\bar{\mathbf{z}} = \bar{\mathbf{z}}\cdot\exp(\mathbf{m}-\mathbf{m}')$; $\mathbf{m} = \mathbf{m}'$
    // Parent correction via Eq. (8)
    $\mathbf{s}_p = \frac{1}{\mathcal{C}}\sum_c \mathbf{s}_c$ ▷ Recover parent logits from children
    $\mathbf{A}_c = \exp(\mathbf{s}_c - \mathbf{m})$; $\mathbf{A}_p = \exp(\mathbf{s}_p - \mathbf{m})$
    $\Delta\mathbf{A} = \mathbf{A}_c - \mathbf{A}_p$; $\Delta\mathbf{A}_{:,c} = 0\ \ \forall\, c{:}\,\mathbf{n}_c{=}0$
    $\bar{\mathbf{x}}$ += $\Delta\mathbf{A}\ \bar{\mathbf{v}}_c$; $\bar{\mathbf{z}}$ += $\sum_c \Delta\mathbf{A}_{:,c}\cdot\mathbf{n}_c$
  end for
  $Y[bh, \text{Block}_N] = \bar{\mathbf{x}} \,/\, \bar{\mathbf{z}}$
end for

Summary. The full AVQ-attention forward pass consists of two fused kernels (Fig. 4): VQ Precompute (Algorithm 1) and Flash Attention (Algorithm 2). Table 1 compares per-step costs with flat VQ-attention.

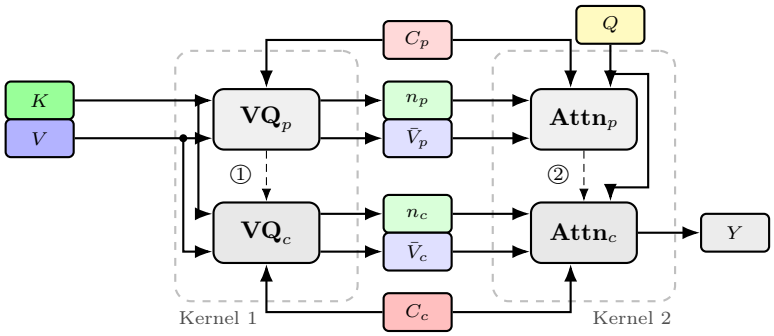

Fig. 4: AVQ-Attention inference pipeline. **Kernel 1** (VQ Precompute): keys and values are quantized against the parent codebook C_p, producing aggregated values $\bar{V}_p$ and counts n_p. Child quantization reuses parent assignments ①, so each key is compared only against the $\mathcal{C}$ children of its assigned parent. **Kernel 2** (Flash Attention): Attn_p computes attention over parent codewords and extracts importance. The online softmax accumulators and per-tile importance scores are carried forward ② to Attn_c, which refines the top-$\mathcal{P}$ most important parents with child attention using correcting attention weights.

Table 1: Complexity comparison: flat VQ-attention (M codewords) vs. AVQ-attention (M_0 parents, $\mathcal{C}$ children per parent, $\mathcal{P}$ parents refined).

Step	**VQ-Attention**	**AVQ-Attention**
VQ assignment	$\mathcal{O}(NMD)$	$\mathcal{O}(N(M_0+\mathcal{C})D)$
Value aggregation	$\mathcal{O}(ND)$	$\mathcal{O}(ND)$
(Parent) Attention	$\mathcal{O}(NMD)$	$\mathcal{O}(NM_0D)$
Child attention	—	$\mathcal{O}(N\mathcal{P}\mathcal{C}D)$
Total FLOPs	$\mathcal{O}(NMD)$	$\mathcal{O}(N(M_0+\mathcal{P}\mathcal{C})D)$
Codebook resolution	M	$M_0(1+\mathcal{C})$

5 Experiments and Results

Starting from pretrained transformers, we replace attention layers with (A)VQ-attention and fine-tune for a small number of epochs. We evaluate on image classification (ImageNet-1k [8]) using a ViT-Base [10] (N=785 tokens, 85.8% top-1) and semantic segmentation (ADE20K [40]) using DPT-Large [27] (N=901 tokens, 49.0% mIoU). Full training details are in Supp. F. Figure 5 reports task performance vs. attention kernel time under identical training conditions. On both tasks, AVQ-attention consistently outperforms flat VQ-attention at comparable cost, confirming that adaptive codebook allocation improves the accuracy–efficiency trade-off. We further analyze codebook properties and attention-mass concentration in Supp. H.

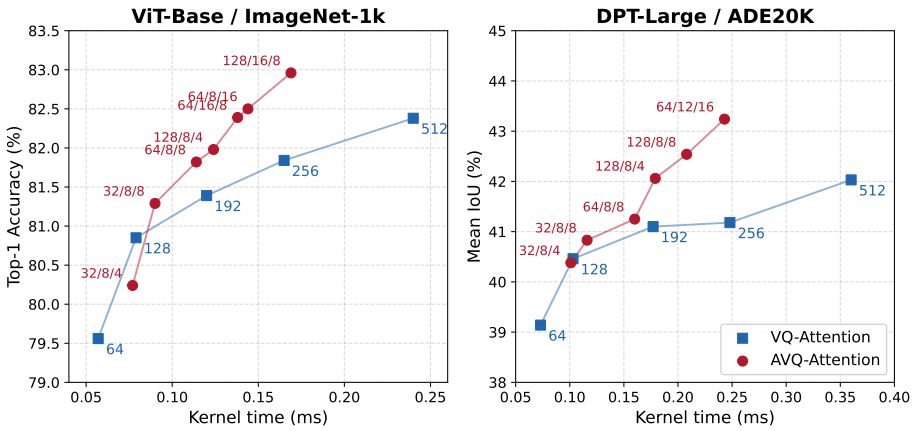

Fig. 5: Task performance vs. attention kernel time (ms) for VQ-attention (blue, labeled by M) and AVQ-attention (red, labeled by $M_0/\mathcal{P}/\mathcal{C}$). **Left:** Top-1 accuracy on ImageNet-1k (N=785). **Right:** Mean IoU on ADE20K (N=901). AVQ achieves higher quality than VQ at comparable cost on both tasks.

5.1 Scaling Analysis

The efficiency advantages of (A)VQ-attention grow with sequence length. To validate the complexity analysis from Tab. 1, we benchmark wall-clock kernel time across sequence lengths; Supp. E confirms the predicted linear scaling with both N and the number of codewords (setup details there). Table 2 reports absolute wall-clock times for the larger configurations most relevant to long-sequence deployment, including an unfused VQ baseline to isolate the speedup from fusing the VQ precompute step. At N=65,536, our fused AVQ-attention configurations are 81–127× faster than Flash Attention. For flat VQ-attention, comparing against the unfused baseline shows that fusing the VQ precompute step alone provides a ∼2× speedup.

5.2 Elastic Inference via Top-$\mathcal{P}$ Adjustment

Since $\mathcal{P}$ can be changed at inference time without retraining, a single model can trade speed for accuracy. Supp. H reports this for M_0=64, $\mathcal{C}$=8, trained with full spawning ($\mathcal{P}$=M_0). We also report capture: the fraction of true attention weight falling on keys assigned to the selected $\mathcal{P}$ parents. At $\mathcal{P}$=16, capture exceeds 82% on both tasks and performance is within 0.3% of the maximum, explaining the diminishing returns at higher $\mathcal{P}$.

5.3 Comparison Across Efficient-Attention Methods

We compare against several representative efficient-attention methods (Tab. 3). AVQ reaches the highest mIoU at comparable kernel cost, confirming it is competitive as a drop-in replacement for the attention layer.

Table 2: Wall-clock kernel time (ms) vs. sequence length N (B=4, H=12, D=64). For fair comparison, all methods use flash-attention-style kernels for the attention step; "unfused VQ" uses `torch.compile`'d PyTorch only for VQ precompute, so the speedup to our fused rows isolates the contribution of fusing the VQ precompute step. AVQ configs denoted $M_0/\mathcal{P}/\mathcal{C}$. Setup details in Supp. E.

Configuration	N=1k	N=4k	N=16k	N=64k
Baselines				
Flash-Attn ($\mathcal{O}(N^2)$)	0.21	3.11	49.6	847
VQ-attn (unfused VQ) M=256	0.35	1.29	5.16	20.7
VQ-attn (unfused VQ) M=512	0.61	2.31	9.28	37.4
Ours (fused kernels)				
VQ-attn M=256	0.18	0.66	2.62	10.6
VQ-attn M=512	0.30	1.10	4.40	17.9
AVQ-attn 64/8/8	0.13	0.43	1.65	6.67
AVQ-attn 64/16/8	0.16	0.51	1.99	7.95
AVQ-attn 128/8/8	0.17	0.60	2.37	9.46
AVQ-attn 128/16/8	0.21	0.67	2.59	10.4

Table 3: ADE20K (DPT-L); mIoU ± std from 3 seeds. All methods use an identical 30-epoch recipe to give each the chance to saturate (Supp. F). †not FA-compatible.

Method	**mIoU (%)**	**Kernel (ms)**
Flash Attention-v2 (baseline)	49.0	0.313
AVQ (M_0=32, $\mathcal{P}$=8, $\mathcal{C}$=8)	**43.33** ± 0.12	0.116
Flat VQ (M=128)	42.70 ± 0.08	0.103
Flat VQ (M=192)	43.04 ± 0.05	0.164
Swin (window 7) [21]	42.90 ± 0.09	0.108
NATTEN (window 5) [11]	40.99 ± 0.21	0.126
Linformer (k=64) [37]	38.80	0.139
Performer (r=256) [5]	25.29	0.54†

5.4 High-Resolution Diffusion

We evaluate AVQ-attention in the Stable Diffusion 1.5 (SD1.5) UNet [28] following the distillation setup of LinFusion [20]. We adopt the same distillation objective, data, and hyperparameters, training for 50k steps (half of their 100k). As the UNet's inner-most self-attention blocks operate on only $N \leq 256$ tokens, exact attention is already trivially cheap and a marginal part of the cost, leaving little to gain from replacing it. We therefore apply AVQ only to the five outer (level-0) blocks (N=4096) and for one experiment also to the five level-1 blocks (N=1024). Using ScaleCrafter [12] we further test the models on unseen 1024^2 inference resolutions, making AVQ-attention handle token numbers up to $N \approx$16k. We report FID [13] and CLIP score [26] on COCO [18] under LinFusion's evaluation protocol in Table 4.

Table 4: SD1.5 on COCO under the LinFusion protocol; FID, CLIP, and UNet forward time per denoising step. †additionally replaces the five level-1 (N=1k) blocks.

	$\mathbf{512}^2$ (train)			$\mathbf{1024}^2$ (w/ ScaleCrafter)		
Method	FID↓	CLIP↑	ms	FID↓	CLIP↑	ms
SD1.5 (FA-v2)	12.75	0.318	44.37	41.03	0.292	213.43
LinFusion (Mamba)	12.52	0.318	43.70	36.37	0.295	141.57
ToMe-SD (r=0.5)	**12.40**	0.317	41.99	43.00	0.290	163.30
AVQ (ours)						
M32P8C8	12.50	0.319	**39.09**	**35.39**	**0.297**	133.48
M64P16C8	12.51	**0.320**	39.50	35.65	0.296	135.06
M128P32C8	12.55	0.319	40.23	35.51	**0.297**	138.16
M64P16C8 (+L1)†	12.48	**0.320**	39.41	36.29	**0.297**	**128.59**

6 Discussion and Conclusion

We have presented AVQ-attention, an adaptive extension of vector-quantized attention that dynamically allocates codebook capacity to regions of key space receiving high attention mass. By combining a hierarchical codebook with importance-driven refinement, AVQ-attention achieves better task performance than flat VQ-attention at comparable cost. The experiments validate AVQ-attention as a general-purpose attention mechanism rather than targeting benchmark-specific performance: we take pretrained transformers, replace their attention layers, and fine-tune for only a few epochs. Our focus was on the controlled comparison between AVQ, VQ and other efficient attention types, and given the promise of (A)VQ-attention as a competitive layer type, we leave the design of efficient end-to-end transformer architectures around it as future research. Some training choices remain lightly explored. For instance, the optimal M_0, $\mathcal{P}$, and $\mathcal{C}$ likely vary across layers — some layers may need far fewer codewords (Supp. H) — making per-layer configuration an attractive direction. AVQ-attention facilitates this by exposing informative measures such as captured attention mass and quantization error. The latter could additionally serve as a refinement criterion alongside importance, prioritizing parents whose children differ most from the parent representation. Together, these signals could enable efficient architecture search without full retraining. Beyond per-layer tuning, deeper hierarchies are a natural next step: each additional depth adds only $\mathcal{PC}$ codes to the attention cost while multiplying codebook resolution by $\mathcal{C}$, potentially yielding exponential resolution growth for linear cost.

Acknowledgements

This work is financially supported by Qualcomm Technologies Inc., the University of Amsterdam, and the allowance Top consortia for Knowledge and Innovation from the Netherlands Ministry of Economic Affairs and Climate Policy.

References

1. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. arXiv preprint **2004.05150** (2020), https://arxiv.org/abs/2004.05150
2. Bolya, D., Fu, C.Y., Dai, X., Zhang, P., Feichtenhofer, C., Hoffman, J.: Token merging: Your vit but faster. In: International Conference on Learning Representations (ICLR) 2023 (2022), https://arxiv.org/abs/2210.09461, oral presentation
3. Bolya, D., Hoffman, J.: Token merging for fast stable diffusion. arXiv preprint **2303.17604** (2023), https://arxiv.org/abs/2303.17604
4. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. arXiv preprint **1904.10509** (2019), https://arxiv.org/abs/1904.10509
5. Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, Ł., Belanger, D., Colwell, L., Weller, A.: Rethinking attention with Performers. In: ICLR (2021), https://arxiv.org/abs/2009.14794
6. Dao, T.: FlashAttention-2: Faster attention with better parallelism and work partitioning. In: ICLR (2024), https://arxiv.org/abs/2307.08691
7. Dao, T., Fu, D.Y., Ermon, S., Rudra, A., Ré, C.: Flashattention: Fast and memory-efficient exact attention with io-awareness. In: Advances in Neural Information Processing Systems. vol. 35 (2022), https://arxiv.org/abs/2205.14135
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: CVPR. pp. 248–255 (2009)
9. van den Dool, W., Zhdanov, M., Asano, Y.M., Welling, M.: Adaptive mesh-quantization for neural PDE solvers. arXiv preprint **2511.18474** (2025), https://arxiv.org/abs/2511.18474
10. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2021), https://arxiv.org/abs/2010.11929
11. Hassani, A., Walton, S., Li, J., Li, S., Shi, H.: Neighborhood attention transformer. In: CVPR (2023), https://arxiv.org/abs/2204.07143
12. He, Y., Yang, S., Chen, H., Cun, X., Xia, M., Zhang, Y., Wang, X., He, R., Chen, Q., Shan, Y.: ScaleCrafter: Tuning-free higher-resolution visual generation with diffusion models. In: ICLR (2024), https://arxiv.org/abs/2310.07702
13. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs trained by a two time-scale update rule converge to a local nash equilibrium. In: NeurIPS (2017), https://arxiv.org/abs/1706.08500
14. Hooper, C., Kim, S., Mohammadzadeh, H., Maheswaran, M., Zhao, S., Paik, J., Mahoney, M.W., Keutzer, K., Gholami, A.: Squeezed attention: Accelerating long context length LLM inference. In: Proc. Annual Meeting of the Association for Computational Linguistics (ACL) (2025), https://arxiv.org/abs/2411.09688
15. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are RNNs: Fast autoregressive transformers with linear attention. In: ICML (2020), https://arxiv.org/abs/2006.16236
16. Kitaev, N., Kaiser, Ł., Levskaya, A.: Reformer: The efficient transformer. In: ICLR (2020), https://arxiv.org/abs/2001.04451
17. Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., Chen, D.: Snapkv: Llm knows what you are looking for before generation. Advances in Neural Information Processing Systems **37**, 22947–22970 (2024)

18. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: ECCV (2014), https://arxiv.org/abs/1405.0312
19. Lingle, L.D.: Transformer-vq: Linear-time transformers via vector quantization. In: International Conference on Learning Representations. vol. 2024, pp. 6121–6150 (2024)
20. Liu, S., Yu, W., Tan, Z., Wang, X.: LinFusion: 1 GPU, 1 minute, 16K image. arXiv preprint arXiv:2409.02097 (2024), https://arxiv.org/abs/2409.02097
21. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin Transformer: Hierarchical vision transformer using shifted windows. In: ICCV (2021), https://arxiv.org/abs/2103.14030
22. Lloyd, S.P.: Least squares quantization in PCM. IEEE Transactions on Information Theory **28**(2), 129–137 (1982)
23. Mao, Y., Wang, Q., Ester, M., Li, K.: IceCache: Memory-efficient KV-cache management for long-sequence LLMs. In: ICLR (2026), https://arxiv.org/abs/2604.10539
24. Milakov, M., Gimelshein, N.: Online normalizer calculation for softmax. arXiv preprint arXiv:1805.02867 (2018), https://arxiv.org/abs/1805.02867
25. van den Oord, A., Vinyals, O., Kavukcuoglu, K.: Neural discrete representation learning. In: NeurIPS (2017), https://arxiv.org/abs/1711.00937
26. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: ICML (2021), https://arxiv.org/abs/2103.00020
27. Ranftl, R., Bochkovskiy, A., Koltun, V.: Vision transformers for dense prediction. In: ICCV. pp. 12179–12188 (2021), https://arxiv.org/abs/2103.13413
28. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: CVPR (2022), https://arxiv.org/abs/2112.10752
29. Roy, A., Saffar, M., Vaswani, A., Grangier, D.: Efficient content-based sparse attention with routing transformers. Transactions of the Association for Computational Linguistics **9**, 53–68 (2021)
30. Tang, C., Ouyang, K., Wang, Z., Zhu, Y., Ji, W., Wang, Y., Zhu, W.: Mixed-precision neural network quantization via learned layer-wise importance. In: European Conference on Computer Vision (ECCV) (2022)
31. Tillet, P., Kung, H.T., Cox, D.: Triton: An intermediate language and compiler for tiled neural network computations. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL). pp. 10–19 (2019)
32. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: NeurIPS. pp. 5998–6008 (2017), https://arxiv.org/abs/1706.03762
33. Červený, J.: Gilbert: Generalized Hilbert ("Gilbert") space-filling curve for rectangular domains of arbitrary (non-power of two) sizes. https://github.com/jakubcerveny/gilbert (2018), gitHub repository
34. Vyas, A., Katharopoulos, A., Fleuret, F.: Fast transformers with clustered attention. In: Advances in Neural Information Processing Systems (NeurIPS) (2020), https://arxiv.org/abs/2007.04825, arXiv:2007.04825
35. Wallace, G.K.: The JPEG still picture compression standard. Communications of the ACM **34**(4), 30–44 (1991)

36. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: HAQ: Hardware-aware automated quantization with mixed precision. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8612–8620 (2019)
37. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020), https://arxiv.org/abs/2006.04768
38. Yuan, J., Gao, H., Dai, D., Luo, J., Zhao, L., Zhang, Z., Xie, Z., Wei, Y., Wang, L., Xiao, Z., et al.: Native sparse attention: Hardware-aligned and natively trainable sparse attention. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 23078–23097 (2025)
39. Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., Chen, B.: H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
40. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ADE20K dataset. In: CVPR. pp. 633–641 (2017)