AVQ-ATTENTION-2: ADAPTIVE VECTOR-QUANTIZED ATTENTION WITH EXACT LOCAL BLOCKS AND LINEAR-COST TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention costs $\mathcal{O}(N^2)$ in the number of tokens. Vector-quantized attention reduces this to $\mathcal{O}(NM)$ by clustering the N keys, attending to the M codewords instead, and its adaptive form, AVQ-attention, concentrates codebook resolution where a query's attention mass actually falls. We present AVQ-Attention-2, a better design that merges it with exact local attention and attends to the mean of each cluster's keys rather than to a codeword. We show three improvements: consistently better quality at comparable cost; training that now also costs $\mathcal{O}(NM)$ rather than $\mathcal{O}(N^2)$; and an approximation accurate enough that the layer can be dropped into a pretrained transformer without fine-tuning. Across a variety of sequence lengths the result is competitive not merely with other approximate attention but with exact dense attention, running faster than FlashAttention at little to no loss in quality.

1 INTRODUCTION

Attention compares every token with every other, so its cost grows quadratically with sequence length. FlashAttention (Dao et al., 2022; Dao, 2024) removes the memory bottleneck of that computation but not its arithmetic: the quadratic term counts pairs of tokens, and an exact method has to look at all of them. Cheaper attention therefore has to decide what not to pay for, which leaves two alternatives: cut the pairs deemed less important and keep the rest exact, or keep every pair and approximate it.

In the first camp, local attention methods like windowed and neighbourhood attention (Liu et al., 2021; Hassani et al., 2023) restrict every query to a nearby subset of keys and evaluate those exactly. In doing so they lose global information, and at high resolutions the neighbourhoods have to grow in step to keep proportional coverage. Vector-quantized attention instead sorts the N keys into M clusters and attends over the clusters (Lingle, 2024), so every key still contributes, through a coarser stand-in; AVQ-attention (van den Dool et al., 2026) makes that resolution adaptive, spending a per-tile refinement budget on whichever key clusters receive the most attention mass.

However, attention mass is often concentrated near the query. Local sparse methods build that prior in directly; an adaptive method has to rediscover it from the data, and then spends part of its budget approximating keys that a local method computes exactly, and faster. Seen that way the two camps are best used as complements: compute each query's own neighbourhood exactly, and let adaptive approximation cover the rest. The benefit does not come only from the exact local attention, but also from the adaptive capacity it frees for global structure.

We propose AVQ-Attention-2, an adaptive vector-quantized attention layer that does both at once. Every query attends exactly over its own neighbourhood and through cluster representatives to everything beyond it. Critically, we develop an efficient kernel implementation that allows this to happen within a single attention pass, rather than as two mechanisms run side by side, all the while never materializing the attention matrix, as in FlashAttention. Careful bookkeeping in the kernel code allows focusing the adaptive part entirely outside that neighbourhood, effectively using the compute for better global attention at the same time.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

On top of combining AVQ-attention with exact local attention, we propose improvements to each. For the local attention, we propose a hardware-aware method that computes optimally shaped, compact neighbourhoods applied once before training. We show that such hardware-awareness and principled neighbourhood selection is Pareto-optimal compared to existing local attention frameworks (Section 5.3). For AVQ-attention the codebook vector itself is actually not required at all: only the fact that keys are clustered in the first place. We can thus use the key cluster's own mean instead of the fixed codeword to create a better approximation. As the mean of keys is a differentiable function of those keys, the layer can be trained through it directly. The codebook is kept purely as a routing mechanism, updated in the background by an exponential moving average of the clusters it produces, and never has to interact with the model's gradients. Furthermore, we improve the speed of the quantization pass by first sorting the keys by their cluster assignment, which lets each cluster's quantities be written directly rather than accumulated through contention.

As the effectiveness of linear-time attention methods depends a lot on sequence length, we evaluate our method to cover a range of different N: image classification on ImageNet-1k (Deng et al., 2009) ($N = 785$), semantic segmentation on ADE20K (Zhou et al., 2017) ($N = 901$) and a V-JEPA 2 video encoder (Assran et al., 2025) on Diving-48 (Li et al., 2018) ($N = 4608$, 9216 and 18432). While the biggest benefits come at large N, the smaller settings can be the less forgiving test, since a method cannot hide there behind better asymptotic behaviour.

Our contributions are:

- Improved design of both hardware-aware local attention and adaptive vector-quantized attention separately, as well as an implementation that combines both.
- Custom Triton (Tillet et al., 2019) kernels that fuse that combination into a single attention pass: the exact local keys and the cluster representatives share the same online softmax and normalizer, and the attention matrix is never materialized.
- We show that the result is an attention layer that runs in linear time for both training and inference, already competitive with FlashAttention at relatively small sequence lengths.

2 Related Work

Vector-quantized attention. Transformer-VQ (Lingle, 2024) quantizes each key to a learned codebook and attends over the codewords instead of the keys. AVQ-attention (van den Dool et al., 2026) adds a hierarchical codebook whose refinement is chosen per query tile from the attention mass each cluster receives. VQ-attention has also been applied to causal language models, with the codebook maintained online rather than learned (Alonso et al., 2026), and to graphs, over centroids built in input-feature space (Kong et al., 2023).

Local and sparse attention. Restricting attention to a neighbourhood around each query is among the most effective efficient-attention families in practice, and underpins the standard efficient backbones in vision as well as long-document models (Weissenborn et al., 2020; Liu et al., 2021; Hassani et al., 2023; Beltagy et al., 2020; Child et al., 2019). A second idea is to shape the sparse pattern so a fused kernel can exploit it: NSA (Yuan et al., 2025) selects whole blocks rather than individual keys, since GPUs read contiguous blocks far faster than scattered indices. We apply the same reasoning to local attention: the tile width is the granularity the attention kernel already works at, so we take it as given and spend our freedom on the shape and placement of the tiles instead. Others have varied tile shape as well, but more elaborately, adapting it to the input on every forward pass (Zhang et al., 2022; Yu et al., 2022); we optimise one arrangement against the hardware once, before training, which makes it free at run time.

Linear attention and state-space models. Kernelized and low-rank attention (Katharopoulos et al., 2020; Wang et al., 2020; Choromanski et al., 2021) replace the softmax, and state-space models (Gu et al., 2022; Gu & Dao, 2024) replace attention altogether, mixing sequences in linear time at the cost of content-based lookup. We keep standard softmax attention and reduce the number of distinct objects each query compares against.

Quantization of the attention computation. A separate line reduces the bit width of the attention computation itself: SageAttention (Zhang et al., 2025) runs the products in 8 bits, and the key–value cache can be stored at fewer bits to reduce memory traffic during decoding (Li et al., 2025; Hooper et al., 2024). That axis is orthogonal to ours, and composable with it: those methods keep N interactions and make each one cheaper, where we reduce the number of interactions, but the compute inside our layer can be quantized to lower bitwidth the same way.

3 PRELIMINARIES

3.1 VECTOR-QUANTIZED ATTENTION

Attention computes $Y_i = \sum_j \exp(Q_i K_j^\top) V_j \,/\, \sum_l \exp(Q_i K_l^\top)$ over N tokens, at $\mathcal{O}(N^2 D)$ cost. Vector-quantized attention replaces each key by a representative drawn from a codebook $\{C_a\}_{a=1}^M$ with $M \ll N$, assigning $a(K_j) = \arg\min_a \|C_a - K_j\|^2$. Keys sharing a representative then share a logit, so the sums can be regrouped per code. With

$$n_a = |\{j : a(K_j) = a\}|, \qquad \bar{V}_a = \sum_{j:\, a(K_j)=a} V_j, \tag{1}$$

attention becomes a softmax over M codes rather than N keys,

$$Y_i \;\approx\; \frac{1}{\bar{Z}_i} \sum_{a=1}^{M} \exp(Q_i C_a^\top)\, \bar{V}_a, \qquad \bar{Z}_i \;=\; \sum_{a=1}^{M} \exp(Q_i C_a^\top)\, n_a, \tag{2}$$

3.2 ADAPTIVE VQ-ATTENTION

AVQ-attention (van den Dool et al., 2026) makes the codebook hierarchical and its resolution in key space input-dependent. A codebook of M_0 parents, each with $\mathcal{C}$ children, is traversed per key: a key is assigned to its nearest parent and then, if a child is closer, to that child. Attention proceeds in two stages of one FlashAttention-style pass (Dao et al., 2022).[1] The first stage computes Equation (2) over the M_0 parents and, from the running denominator, an importance score per code and query tile I,

$$w_j(I) \;=\; \sum_{i \in I} \frac{A_{ij}\, n_j}{\bar{Z}_i}, \qquad A_{ij} = \exp(Q_i C_j^\top), \tag{3}$$

i.e. the share of tile I's attention mass that code j receives. The second stage *refines* the $\mathcal{P}$ parents with the largest $w_j(I)$: attention is also computed over their children. Writing $S_{ip} = Q_i C_p^\top$ for the parent's logit and S_{ic} for a child's, the keys that were closer to a child than to the parent have their contribution to the running numerator and denominator replaced by that child's, by accumulating the *correcting attention*

$$\Delta A_{ic} \;=\; \exp(S_{ic}) - \exp(S_{ip}) \tag{4}$$

against the child aggregates $\bar{V}_c$ and n_c: the $\exp(S_{ic})$ puts a key in at its child code, the $\exp(S_{ip})$ takes it out at its parent code, in the same dot product. To avoid having to recompute or store the parent logit, AVQ-attention constrains each parent to be the mean of its children, $C_p = \frac{1}{\mathcal{C}} \sum_c C_{p,c}$, so $S_{ip} = \frac{1}{\mathcal{C}} \sum_c S_{ic}$ is a reduction over child logits already in registers.

While inference costs $\mathcal{O}((M_0 + \mathcal{P}\mathcal{C})ND)$, training is slightly more complicated. The assignment $a(K_j)$ is a discete choice, so no gradient reaches the keys through it, and AVQ-attention trains with a straight-through estimator that attends over all N quantized keys, making the forward pass that the gradient flows through quadratic in N.

Finally, refinement is done per query tile, so different regions of the input refine different parts of the codebook. To make tiles spatially compact, tokens are reordered along a Gilbert space-filling curve (Červený, 2018).

[1] The numerator and the denominator $\bar{Z}_i$ of Equation (2) are carried as running sums per query under an online softmax; the attention matrix is never materialized.

4 METHOD

4.1 ATTENDING TO CLUSTER MEANS

Because the regrouping in Equation (2) holds for any vector shared by the keys of a cluster, we can use the mean of the keys actually assigned to it,

$$\hat{K}_j \;=\; \bar{K}_{m(j)}, \qquad \bar{K}_m \;=\; \frac{1}{n_m} \sum_{j:\, m(j)=m} K_j, \tag{5}$$

computed per input rather than stored. The codebook, trained using online k-means as in van den Dool et al. (2026), keeps its role in deciding *which* keys form a cluster, but the actual $\bar{K}_m$ is what is used in attention. Naturally, as the minimizer of $\sum_{j\in m} \|K_j - x\|^2$, this reduces quantization error. Because it is also based directly on current keys, attention over means tends to exact attention as refinement grows.

Training becomes exact and linear in N. Unlike a codeword, $\bar{K}_m$ is a differentiable function of the keys: $\partial \bar{K}_{m,d}/\partial K_{j',d} = \mathbf{1}[m(j') = m]/n_m$. Write $p_{im} = \exp(Q_i \bar{K}_m^\top)\, n_m/\bar{Z}_i$ for the weight cluster m receives from query i in Equation (2), $O_i = \sum_m p_{im} \bar{V}_m/n_m$ for the layer output and $dO_i = \partial L/\partial O_i$ for the gradient arriving at it. The chain rule collapses the per-key gradient onto its cluster,

$$\frac{\partial L}{\partial K_{j',d}} \;=\; \frac{1}{n_m} \sum_i p_{im} \left\langle dO_i,\, \bar{V}_m/n_m - O_i \right\rangle Q_{i,d}, \qquad m = m(j'), \tag{6}$$

derived in Section E. The per-key factor V_j that made the straight-through path quadratic has averaged into $\bar{V}_m$ by linearity; every quantity on the right is per cluster or per query, so the backward pass costs $\mathcal{O}(MND)$, matching the forward.

The hierarchy, without the constraint. Refinement never recomputes a coarse representative's logit; it rebuilds it from the finer logits that replace it, which is what keeps it cheap. AVQ-attention can do so because each parent is constrained to be the mean of its children. A cluster mean satisfies no such constraint: $\bar{K}_p$ need not equal $\frac{1}{\mathcal{C}} \sum_c \bar{K}_{p,c}$. A count-weighted mean would solve this, but only if no key stays behind at the parent, so that $n_p = \sum_c n_c$. In refinement, contrary to AVQ-attention, we therefore let *every key* descend to its nearest child, and can then write

$$\bar{K}_p \;=\; \sum_{c=1}^{\mathcal{C}} \frac{n_c}{n_p}\, \bar{K}_{p,c}, \qquad \text{hence} \qquad S_{ip} \;=\; \sum_{c=1}^{\mathcal{C}} \frac{n_c}{n_p}\, S_{ic}. \tag{7}$$

The parent logit is then again a reduction over child logits already in registers. As an additional benefit the constraint has no remaining job: children are free to move to optimal positions as a plain clustering of their parent's cell.

4.2 COMBINING EXACT LOCAL AND AVQ-ATTENTION

A query's nearest keys are where most of its attention mass sits. We therefore let each query tile attend to its own Block_N keys exactly and to cluster means everywhere else. However, naively applying the two together runs into a clear issue: a key the tile attends to exactly is still a member of its cluster, so it would contribute once as itself and once through $\bar{V}_m$.

For simplicity we first consider the denominator, writing $n_m^I = |\{j \in I : m(j) = m\}|$ for the number of cluster m's keys that lie in tile I. Every cluster's count separates into the keys the tile does not hold and the keys it does,

$$\begin{aligned} \bar{Z}_I = \sum_m \exp(Q_I \bar{K}_m^\top)\, n_m &= \sum_m \exp(Q_I \bar{K}_m^\top)\big(n_m - n_m^I\big) + \sum_m \exp(Q_I \bar{K}_m^\top)\, n_m^I \\ &= \sum_m \exp(Q_I \bar{K}_m^\top)\big(n_m - n_m^I\big) + \sum_{j\in I} \exp(Q_I \bar{K}_{m(j)}^\top). \end{aligned} \tag{8}$$

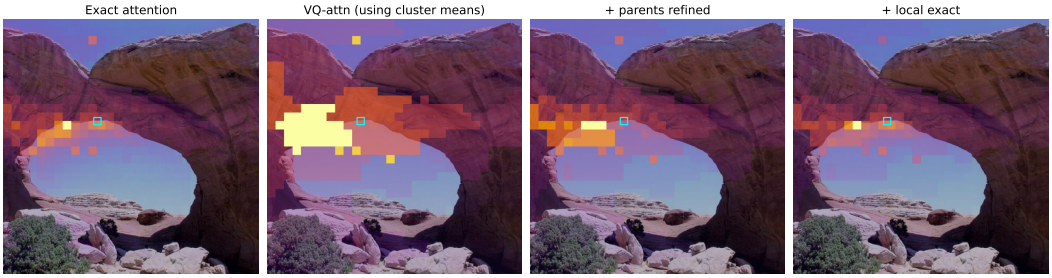

Figure 1: Attention received by one query (cyan), computed four ways. Exact local attention is meant to fix the query's own neighbourhood, but it also sharpens the rest of the map: once the tile's own keys are computed exactly they no longer attract refinement, so the budget is spent on the rest of the image. The share of the selected parents' keys lying inside the tile falls from 0.25 to 0.05.

The second sum is exactly the tile's own keys, each still counted through its cluster mean. Replacing it by the same sum over the keys themselves, every cluster is left counting only its keys outside the tile,

$$\bar{Z}_I = \sum_m \exp(Q_I \bar{K}_m^\top)\big(n_m - n_m^I\big) \;+\; \sum_{j \in I} \exp(Q_I K_j^\top). \tag{9}$$

This is what the tile computes, and it costs little: n_m^I is a histogram of M integers, subtracted before the pass begins.

The same split does not transfer to the numerator. The tempting analogue, a per-tile value sum $\bar{V}_m^I$, is a D-wide aggregate per cluster rather than a single integer, so forming it would mean a scatter inside every tile. What's more, $m(j)$ is not yet known when the cluster terms are accumulated, since which code a local key is represented by depends on whether this tile refines its parent, decided only after the parent pass. [2] The numerator instead keeps the full $\bar{V}_m$, and folds the correction directly into the epilogue that computes exact local attention, once each key has a final code:

$$\sum_m \exp(Q_I \bar{K}_m^\top)\, \bar{V}_m \;+\; \sum_{j \in I} \Big[\exp(Q_I K_j^\top) - \exp(Q_I \bar{K}_{m(j)}^\top)\Big] V_j, \tag{10}$$

which resolves back into the aggregate form the tile cannot compute,

$$\sum_m \exp(Q_I \bar{K}_m^\top)\big(\bar{V}_m - \bar{V}_m^I\big) \;+\; \sum_{j \in I} \exp(Q_I K_j^\top)\, V_j. \tag{11}$$

We thus perform a FlashAttention-style pass by first tiling over cluster means and then finishing with a key tile. The online softmax does not care that the numerator and the denominator are accumulated differently, only that the aggregate it ends with is correct. To exploit the synergy with AVQ-attention, however, it *is* important that the corrected counts $n_m - n_m^I$ are used directly: the importance score of Equation (3) is derived from them, so the parents selected for refinement are more likely to be picked from global context, rather than overlapping with the local tile that is already covered exactly (Figure 1).

A second synergy is particular to the cluster-mean formulation. Equation (6) collapses each key's gradient onto its cluster, which is exactly what makes training linear in N; the price is that a key no longer has a training signal of its own, only its share of its cluster's, and keys are updated as members of a cluster rather than as themselves. Exact local attention gives that back. A key attended to exactly by the Block_N queries of its own tile enters the loss as itself, and is moved by what it did rather than by what its cluster did. The two terms then carry complementary signal: the cluster term settles where a cluster sits, the exact term separates the keys inside it.

[2] One may end up pre-emptively removing $\bar{V}_m^I$ for a parent code m to later find that in fact parent m is replaced by its children.

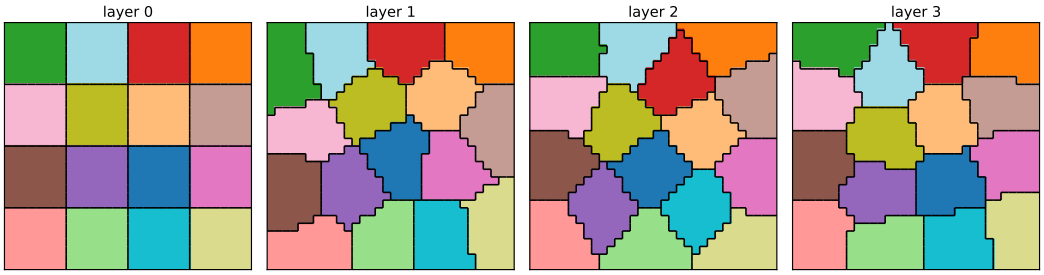

Figure 2: Four consecutive layers of a 32×32 token grid at $\text{Block}_N = 64$. Each layer places its boundaries where the previous layers did not have them.

4.3 Choosing the Tiles

The kernel processes queries in tiles of Block_N, and local attention computes each tile's own keys exactly: the tile *is* the local neighbourhood. By folding a permutation of N into the kernels, however, which tokens *share* a tile can still be freely chosen. Compactness helps both the adaptive refinement — nearby queries sharing a decision which parents to refine — and the exact local attention, making it the obvious thing to optimise for. However, consecutive layers may also benefit from shifted tiles, so that a token at the boundary of its tile in one layer sits near the centre of one in the next.

Write $x_i \in \mathbb{R}^d$ for the position of token i, π for the map from tokens to tiles of exactly Block_N tokens each, and c_t for the centre of tile t. For each layer l we solve (Section C)

$$\min_{\pi,\, c} \sum_{i=1}^{N} w_i^{(l)} \|x_i - c_{\pi(i)}\|^2 \qquad \text{subject to} \qquad \left|\pi^{-1}(t)\right| = \text{Block}_N \quad \text{for every } t, \tag{12}$$

where the weight $w_i^{(l)}$ lets one token ask to sit closer to a centre than another, if earlier layers placed it badly. How badly is exactly $\|x_i - c_{\pi(i)}\|^2$, so we accumulate it over layers with an exponential moving average,

$$a_i^{(l)} \;=\; \gamma\, a_i^{(l-1)} \;+\; \left\|x_i - c_{\pi_{l-1}(i)}^{(l-1)}\right\|^2, \qquad w_i^{(l)} \;=\; 1 + \frac{a_i^{(l)}}{\bar{a}^{(l)}}, \tag{13}$$

with $a^{(0)} = 0$, $\gamma = \frac{1}{2}$ and $\bar{a}^{(l)} = \frac{1}{N}\sum_i a_i^{(l)}$ the mean over tokens. Boundary tokens then pull the next layer's centres toward them, and each layer places its cuts where the previous ones did not (Figure 2). Note that nothing refers to a particular grid: the input is the coordinate array, so the same construction can apply to 2-D and 3-D grids and point clouds alike. Section 5.3 compares multiple local attention methods and finds that local attention using our tiles with this optimised placement alone already leads to Pareto-optimal results.

4.4 Faster VQ-Precompute Kernels

Before any query tile runs, one pass over the keys assigns each key to a code and accumulates the per-code aggregates. Its cost is dominated not by the assignment but by the accumulation: every key scatter-adds its D-wide value into its code's $\bar{V}_m$, and since keys anywhere in the sequence can land in the same code, those adds have to be atomic — each one reads the accumulator, adds to it and writes it back, serialised against every other key landing in that code.

We fix this by first sorting the keys by their assignment. The sort can be an efficient $\mathcal{O}(N)$ counting sort, because we *can* already compute the count aggregates n_m cheaply: they are single integers rather than D-dimensional vectors. Ordering the buckets so that a parent's children sit next to each other then puts every code's keys consecutively in that order, and a single pass over the sorted indices can accumulate in registers, writing each output directly once.

This matters more once $\bar{K}_m$ is needed alongside $\bar{V}_m$. Each key then carries two D-wide payloads into the aggregation instead of one, so an atomic pipeline pays its dominant cost twice. The sorted

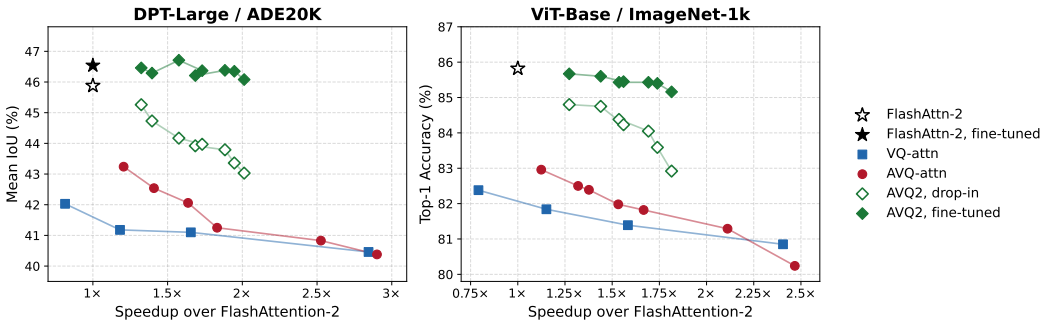

Figure 3: Quality against the attention speedup over FlashAttention-2, measured on an RTX 3090; the VQ-attention and AVQ-attention numbers are taken from van den Dool et al. (2026). Without any fine-tuning AVQ-Attention-2 already lies above both of their fine-tuned curves, indicating the approximation itself is already closer to dense attention. With fine-tuning it approaches the dense model, at a 1.3–2.0× speedup. Note that both settings are relatively small-N, where a linear-time model actually has the least room to work with: speed advantages only grow with N.

traversal pays the sorting work once and amortizes it over both $\bar{V}_m$ and $\bar{K}_m$, which the same pass accumulates side by side. Against the original atomic kernel, the sorted precompute is about $1.5\times$ faster when only $\bar{V}_m$ is accumulated and about $2.7\times$ once $\bar{K}_m$ is accumulated alongside; the precompute is about a third of the layer's time (Section B.1). Section D gives both kernels line by line.

5 Experiments

We evaluate on three datasets: semantic segmentation on ADE20K, classification on ImageNet-1k, and video classification on Diving-48. Every configuration is evaluated in two ways: as a drop-in and after fine-tuning. As a drop-in, the codebook is only fit to the keys of a few batches passed through the pretrained model and no parameter is trained. Fine-tuning uses a training setup that starts from that same codebook initialization and trains on the task loss plus a per-layer distillation term from the original attention (Section A).

5.1 ADE20K Semantic Segmentation

We replace every attention layer of DPT-Large at 480^2 ($N = 901$) and report mean IoU as a drop-in and after ten epochs of fine-tuning. The pretrained baseline scores 45.88 in our pipeline and 46.54 after the same fine-tuning; we report both.

Figure 3 (left) reports every configuration we train against its attention speedup over FlashAttention-2. As a drop-in the layer is within 3 mIoU of the baseline, and the gap closes with codebook size; after fine-tuning every configuration loses little to no quality against the baseline, at a 1.3–2.0× speedup.

5.2 ImageNet-1k Classification

We replace the self-attention of a ViT-B/8 at 224^2 ($N = 785$) and report top-1 as a drop-in and after one epoch of fine-tuning — a quarter of the budget of van den Dool et al. (2026). The pretrained baseline scores its published 85.82 in our pipeline and does not benefit from fine-tuning.

Figure 3 (right) reports quality against speedup on ImageNet. As a drop-in the layer is within 3 points of the baseline's top-1, and within 0.7 points after one epoch of fine-tuning, at a 1.3–1.8× speedup.

Table 1: Building the layer up from AVQ-attention. ADE20K mean IoU (DPT-Large, 10 epochs) and ImageNet-1k top-1 (ViT-B/8, 1 epoch) after fine-tuning, at two codebook configurations $M_0/\mathcal{P}/\mathcal{C}$; the published row is at van den Dool et al.'s budget (4 instead of our 1 epoch on ImageNet); every other row is ours, cumulative from that row down. Without any fine-tuning AVQ2 already reaches most of its accuracy, where AVQ collapses entirely.

	ADE20K (mIoU)		ImageNet-1k (top-1)	
	$32/8/4$	$32/8/8$	$32/8/4$	$32/8/8$
FlashAttention-2 baseline	45.88	45.88	85.82	85.82
FlashAttention-2, fine-tuned	46.54	46.54	—	—
AVQ-attention, as published	40.38	40.83	80.24	81.29
AVQ, our training setup	41.74	43.09	73.35	76.58
+ cluster means	43.36	44.47	80.74	82.43
+ exact local attention	45.49	45.35	84.56	84.77
+ both = AVQ2	**46.38**	**46.71**	**85.43**	**85.60**
Drop-in, no training				
AVQ	13.98	18.22	5.59	12.55
AVQ2	43.79	44.17	84.05	84.75

5.3 Ablations

Building up from AVQ-attention better shows the distinct changes we make with respect to previous work: Table 1 adds one component at a time. Cluster means and exact local attention each help on their own and are best together. The difference is sharpest without any fine-tuning at all: with the codebook fitted and no gradient step taken, AVQ-attention collapses to 13.98 mIoU and 5.59 top-1, where AVQ2 already reaches 43.79 and 84.05.

Exact local attention, with no AVQ-attention involved, is compared against other local attention methods in Table 2: our tiles, in the Gilbert tiling of van den Dool et al. (2026) and the optimised tiling of Section 4, against Swin (Liu et al., 2021) and NATTEN (Hassani et al., 2023) under the same training setup. Our exact local attention based on optimised tiles is Pareto-optimal against the other baselines on both tasks.

5.4 Video at Larger Sequence Lengths

To verify that AVQ2's performance holds up at larger sequence lengths, we turn to video. V-JEPA 2 (Assran et al., 2025) tokenizes video into tubelets spanning two frames, giving $(F/2) \times (c/16)^2$ tokens for F frames at crop c. We replace self-attention in every block of the ViT-B/16 encoder and follow V-JEPA 2's protocol: the encoder is frozen and an *attentive probe* — a small attention-pooling classifier over its output tokens — is trained on top to determine the encoder's quality. The replaced layers are fine-tuned by distillation on unlabelled clips. Different frames per clip give $N = 4608$, 9216 and 18432 from one set of weights.

Table 3 reports classification accuracy and the cost per attention layer. After distillation our layer is within a point of dense attention at the longest setting, and above it at the shortest. Again, direct drop-in is shown to work here too, losing only a few points in accuracy. Furthermore we use this experiment setting to verify that AVQ-Attention-2 can train from scratch: replacing the attention of the probe instead of that of the encoder, and training it from initialization, lands in the same range.

6 Discussion and Conclusion

We have presented AVQ-Attention-2, improving on AVQ-attention in three ways: faster training, the possibility to drop it into existing networks with little to no fine-tuning, and Pareto-optimal results, competitive with FlashAttention already at relatively small sequence lengths.

Table 2: Local attention alone. Quality after fine-tuning under the setup of Section A, and speedup over FlashAttention-2. Note how optimised tiles beat Gilbert tiles at every setting, showing how the seemingly simple decision of how to divide tokens over tiles already gives a free win. Swin's speed is not monotone in window size, a kernel artifact at these window sizes.

		ADE20K		ImageNet-1k	
Method	Tile	mIoU	Speedup	top-1	Speedup
Swin	window 5	37.40	2.0×	77.50	2.4×
	window 7	40.76	2.9×	80.98	2.3×
	window 9	42.23	2.1×	82.15	1.7×
	window 11	43.69	2.5×	83.12	2.1×
	window 13	43.86	1.6×	83.81	1.4×
NATTEN	kernel 5	39.31	2.2×	79.71	1.7×
	kernel 7	41.94	2.0×	82.46	1.7×
	kernel 9	43.49	1.9×	83.82	1.5×
	kernel 11	44.60	1.4×	84.45	1.1×
Exact local only, Gilbert tiles	32	38.50	3.5×	77.96	2.9×
	64	40.35	3.4×	81.41	2.8×
	128	43.64	2.9×	83.34	2.3×
Exact local only, optimised tiles	32	38.74	3.5×	78.67	2.9×
	64	41.97	3.4×	81.89	2.8×
	128	44.16	2.9×	83.62	2.3×

Table 3: Diving-48, frozen V-JEPA 2 ViT-B/16 encoder with an attentive probe. Top-1 above, and below it the cost of one attention layer (Section B): inference in an encoder layer ($3.9/8.1/16.9\times$), training in a probe layer ($1.2/3.1/6.7\times$).

	Sequence length N		
	4608	9216	18432
Top-1			
Dense	77.72	85.99	**90.20**
AVQ2, drop-in	76.09	82.59	86.90
AVQ2 + distillation	79.70	83.20	**89.29**
Dense encoder, AVQ2 probe	81.17	85.43	88.32
Cost per attention layer (ms)			
Encoder layer, FlashAttention	1.068	3.967	15.895
Encoder layer, AVQ2	0.271	0.489	**0.943**
Probe layer fwd + bwd, FlashAttention	5.13	19.23	74.33
Probe layer fwd + bwd, AVQ2	4.26	6.19	**11.05**

The experiments test the layer as a general-purpose replacement rather than a benchmark-specific one: we take pretrained transformers, replace every attention layer, and fine-tune briefly or not at all. The margin grows with sequence length. In a video encoder at 18432 tokens both a forward layer and a training step of it cost a fraction of what FlashAttention costs, at an accuracy within a point of dense attention. Exact attention grows quadratically in N where ours grows linearly, so that gap widens for the larger N settings.

As future work we plan to move to higher resolutions, longer video, and the decoder setting, where a causal variant of the layer would meet much longer sequences. We showcase AVQ-Attention-2 as an efficient attention layer only, but at larger sequence lengths attention makes up a larger share of the model, and the claim can move from the layer to the throughput of the whole model, aiming for results on specific tasks rather than for a general-purpose replacement.

AI USE STATEMENT

In this work, we used generative AI tools to design methodology, implement methods, analyse data and interpret results. We have not used generative AI tools to generate synthetic data sets or to clean datasets, and the remaining tasks requiring disclosure are not applicable to this work. Additionally, we used generative AI tools to write code, to draft and edit sections of this manuscript, to summarise related literature and to produce figures. We have reviewed all AI-assisted work: AI-assisted code was tested against reference implementations, and every number, claim and citation reported here was checked by the authors against its source. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The training setup and evaluation protocol are given in Section A, timing methodology in Section B, and the derivations behind Sections 4.1 and 4.2 in Section E. Code, kernels and the configuration files for every reported run will be released.

REFERENCES

Nick Alonso, Tomas Figliolia, and Beren Millidge. Online vector quantized attention. *arXiv preprint*, 2602.03922, 2026. URL https://arxiv.org/abs/2602.03922.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint*, 2506.09985, 2025. URL https://arxiv.org/abs/2506.09985.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint*, 2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Paul S. Bradley, Kristin P. Bennett, and Ayhan Demiriz. Constrained k-means clustering. Technical Report MSR-TR-2000-65, Microsoft Research, 2000.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint*, 1904.10509, 2019. URL https://arxiv.org/abs/1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with Performers. In *ICLR*, 2021. URL https://arxiv.org/abs/2009.14794.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2205.14135.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2111.00396.

Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood attention transformer. In *CVPR*, 2023. URL https://arxiv.org/abs/2204.07143.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2401.18079.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML*, 2020. URL https://arxiv.org/abs/2006.16236.

Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C. Bayan Bruss, and Tom Goldstein. GOAT: A global transformer on large-scale graphs. In *Proc. International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17375–17390. PMLR, 2023. URL https://proceedings.mlr.press/v202/kong23a.html.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. CommVQ: Commutative vector quantization for KV cache compression. In *Proc. International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2506.18879.

Yingwei Li, Yi Li, and Nuno Vasconcelos. RESOUND: Towards action recognition without representation bias. In *Proc. European Conference on Computer Vision (ECCV)*, 2018. URL https://openaccess.thecvf.com/content_ECCV_2018/html/Yingwei_Li_RESOUND_Towards_Action_ECCV_2018_paper.html.

Lucas D Lingle. Transformer-VQ: Linear-time transformers via vector quantization. In *International Conference on Learning Representations*, 2024.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. URL https://arxiv.org/abs/2103.14030.

Philippe Tillet, H. T. Kung, and David Cox. 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.

Winfried van den Dool, Patrick Forré, Amir Habibian, Yuki M. Asano, and Max Welling. AVQ-Attention: Adaptive Vector-Quantized Attention. In *European Conference on Computer Vision (ECCV)*, 2026. URL https://arxiv.org/abs/2607.12789.

Jakub Červený. 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.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. URL https://arxiv.org/abs/2006.04768.

Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1906.02634.

Tan Yu, Gangming Zhao, Ping Li, and Yizhou Yu. BOAT: Bilateral Local Attention Vision Transformer. In *British Machine Vision Conference (BMVC)*, 2022. URL https://arxiv.org/abs/2201.13027.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, 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.

Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. SageAttention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.02367.

Qiming Zhang, Yufei Xu, Jing Zhang, and Dacheng Tao. VSA: Learning Varied-Size Window Attention in Vision Transformers. In *European Conference on Computer Vision (ECCV)*, 2022. URL https://arxiv.org/abs/2204.08446.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *CVPR*, pp. 633–641, 2017.

A Training Details

Segmentation (ADE20K). DPT-Large (`Intel/dpt-large-ade`) on `scene_parse_150`, every attention layer replaced. Images are 480^2, giving $N = 901$ tokens. We train 10 epochs with an effective batch size of 16, AdamW at learning rate 10^{-5}, linear warmup over the first two epochs and a constant rate thereafter, and per-layer distillation. Our evaluation pipeline is not identical to the one the released checkpoint was produced under, so we also fine-tune the unmodified model with this same setup and report both references. Training augmentation resizes to $480s$ for $s \in \{1, 1.25, 1.5, 1.75, 2\}$, then random-crops to 480 and flips horizontally; validation resizes to 480^2. Every fine-tuned entry of Figure 3 and Table 1 uses this setup.

Classification (ImageNet-1k). ViT-B/8 (`vit_base_patch8_224.augreg_in21k_ft_in1k`) at 224^2 ($N = 785$), one epoch, effective batch size 256, learning rate warmed up linearly to 5×10^{-6} over the epoch, and the same per-layer distillation. The final block computes only the class-token query, exactly.

Video (Diving-48). V-JEPA 2.1 ViT-B/16 at a 384 crop, every attention layer in the 12-block encoder replaced, with $M_0 = 64$, $\mathcal{P} = 16$ and $\mathcal{C} = 8$ and exact local attention over 128-token tiles. Diving-48 (Li et al., 2018) separates its 48 classes by temporal structure rather than appearance — every clip shows the same pool and board — so a dive is recognised only by relating tokens across the whole clip. We use its train and test split as distributed with the dataset. The attentive probe is trained for 20 epochs at an effective batch size of 16, AdamW at learning rate 10^{-3} with a cosine schedule and no warmup, weight decay 0.8, and fp16. Adaptation runs for 5 epochs at learning rate 10^{-4} on unlabelled training clips. We report the peak top-1 over the 20 probe epochs.

Codebook initialization. The codebook is initialized from real keys collected by passing a few batches through the pretrained model, at a fixed budget of keys per codeword, so that the amount of initialization data scales with codebook size. A drop-in evaluation runs this step and nothing else: no weight of the model is changed. During fine-tuning the codebook receives no gradient and is maintained by EMA alone, and a cluster that stops receiving keys is revived by splitting the highest-distortion live cluster.

Class token under local attention. A class token has no position on the grid, so it belongs to no tile, window or neighbourhood. In every local-attention layer — our exact-local-only layer, Swin and NATTEN — it is therefore handled by one extra exact pass per layer: the class-token query attends to all N keys, and the patch queries attend within their tile with the class token's key excluded. The pass is shared by the three implementations and is included in their reported attention time (Section B). AVQ-attention needs no such pass, since every query, the class token included, reaches the whole sequence through the codebook.

Per-layer distillation. Every replaced layer is regressed onto the output of the layer it replaced in the frozen pre-replacement model. The loss is scale-free — each layer's error is divided by the teacher's own mean square — and weighted by the teacher's measured task loss so that one weight means the same thing across tasks (1.9411 on ADE20K, 0.5432 on ImageNet). Every fine-tuned result uses it except the AVQ-attention reference row, which is trained as in the original paper; Table 1 separate the two.

B Benchmark Details

All attention times are measured on a single NVIDIA RTX 3090 in fp16 and cover both kernels of the layer. The two are captured in one CUDA graph: three eager calls populate the allocator and twenty replays settle the caches, after which two hundred timed replays are recorded, with the events taken outside the graph. Each reported number is the median of five such measurements, and the exact-attention reference is timed the same way in the same process. Query tiles are 64 tokens on ADE20K and ImageNet, and 128 on Diving-48.

B.1 SORTED VERSUS ATOMIC AGGREGATION

Section 4.4 replaces the per-key atomic scatter of the original AVQ precompute with a counting sort followed by one sorted traversal. Table 4 times the two pipelines on the same clustered keys and the same codebook, at one geometry across sequence lengths, accumulating value sums alone and value and key sums together. The atomic kernel is the original one, run at its fastest documented setting (fp16 accumulation). The sorted pipeline is about $1.5\times$ faster on value sums alone and about $2.7\times$ once key sums are accumulated alongside, and the ratio does not depend on the sequence length: adding the second aggregate costs the sorted pass 30% and the atomic one 150%, since each atomic scatter is paid once per key per aggregate while the sorted traversal reads each key once and writes each cluster once. At this geometry the precompute is 41% of the full layer at 1k tokens and 32–35% from 4k up.

Table 4: Quantization precompute, sorted pipeline against the original atomic scatter, accumulating value sums alone and value and key sums together. Milliseconds on one RTX 3090 in fp16; B=4, H=16, D=64, M_0=64, $\mathcal{C}$=8, depth 1.

	$\bar{V}_m$ only			$\bar{V}_m$ and $\bar{K}_m$		
N	sorted	atomic	ratio	sorted	atomic	ratio
1,024	0.079	0.112	1.4×	0.106	0.276	2.6×
4,096	0.240	0.400	1.7×	0.304	0.827	2.7×
16,384	0.936	1.543	1.6×	1.113	3.043	2.7×
65,536	3.697	6.172	1.7×	4.326	11.985	2.8×

C SOLVING FOR THE TILES

Equation (12) is k-means with equal-sized clusters, and we solve it the same way. With π fixed the centres are weighted means; with c fixed the assignment under the capacity constraint is solved exactly as an assignment problem (Bradley et al., 2000), so every tile is full at every iteration rather than only in the limit. We seed the iteration with a recursive bisection of the token set.

Dividing by $\bar{a}^{(l)}$ in Equation (13) is what keeps γ interpretable. It holds the weights at mean 2 in every layer however large the accumulated displacements have grown, so γ sets only how far back the memory reaches and not how strongly the weighting acts. Without it the two would be confounded, and a long memory would also mean a heavier one.

The tiling is solved once, when the model is built, and applied as a permutation folded into the kernels, so it costs nothing per forward pass. One tiling per layer is stored, of one integer per token.

D KERNEL PSEUDOCODE

The two kernels of Section 4.4, line by line. Algorithm 1 runs once over the keys; Algorithm 2 runs once per query tile and consumes its output.

E BACKWARD PASS FOR CLUSTER MEANS

Write $\ell_{im} = Q_i \bar{K}_m^\top$ for the logit of cluster m at query i, $p_{im} = \exp(\ell_{im})\, n_m / \sum_{m'} \exp(\ell_{im'})\, n_{m'}$ for its weight, and

$$O_i \;=\; \sum_m p_{im}\, \bar{V}_m / n_m \tag{14}$$

for the layer output, which is Equation (2) with $\bar{K}_m$ in place of the codeword. Let $dO_i = \partial L / \partial O_i$. Three steps give Equation (6).

Algorithm 1 Sorted VQ precompute: one pass over the keys, feeding every query tile of Algorithm 2.

Input: $K, V \in \mathbb{R}^{BH\times N\times D}$, codebook $C \in \mathbb{R}^{H\times M_0(1+\mathcal{C})\times D}$
Output: per code: count n, value sum $\bar{V}$, key mean $\bar{K}$

$n = 0$; $\bar{V} = 0$; $\bar{K} = 0$
// Assign
for each (bh, Block_N) **in parallel do** ▷ $\mathbf{k} = K[bh, \text{Block}_N]$ in registers
  $\mathbf{p} = \arg\min_{m<M_0} \|\mathbf{k} - C[h,m]\|^2$; $\mathbf{c} = \arg\min_{c\,\in\,\text{children}(\mathbf{p})} \|\mathbf{k} - C[h,c]\|^2$ ▷ [Block_N] each
  $n[bh,\mathbf{p}]$ += 1; $n[bh,\mathbf{c}]$ += 1; store $(\mathbf{p},\mathbf{c})$ ▷ Integer atomics
end for
// Counting sort, children of one parent adjacent: every code's keys form one run
start = exclusive cumsum of n over the children, parent by parent
for each (bh, Block_N) **in parallel do**
  sorted$[bh,\ \text{atomicAdd}(\text{start}[bh,\mathbf{c}], 1)]$ = key index ▷ One integer atomic per key
end for
// Reduce, parents and children in one pass
for each $(bh, \text{Block}_N$ of sorted) **in parallel do**
  $\mathbf{x}_p = \mathbf{y}_p = \mathbf{x}_c = \mathbf{y}_c = 0$ ▷ $[D]$ each: open parent and child run
  for each key j of the block, in sorted order **do**
    $\mathbf{x}_p, \mathbf{x}_c$ += $V[bh,j]$; $\mathbf{y}_p, \mathbf{y}_c$ += $K[bh,j]$
    where the child run ends at j: $\bar{V}[bh,c]$ += $\mathbf{x}_c$; $\bar{K}[bh,c]$ += $\mathbf{y}_c \,/\, n[bh,c]$; $\mathbf{x}_c = \mathbf{y}_c = 0$
    where the parent run ends at j: the same for p
  end for
end for
// Only a run cut by a block boundary is written by two programs; only there is += atomic.

Through the softmax. p_{im} is a softmax over ℓ_{im} with fixed positive weights n_m, so $\partial p_{im'}/\partial \ell_{im} = p_{im'}(\mathbf{1}[m'{=}m] - p_{im})$ as for a plain softmax. Differentiating Equation (14) and using $\sum_{m'} p_{im'}\bar{V}_{m'}/n_{m'} = O_i$,

$$\frac{\partial L}{\partial \ell_{im}} \;=\; p_{im}\Big(\big\langle dO_i, \bar{V}_m/n_m\big\rangle - \sum_{m'} p_{im'}\big\langle dO_i, \bar{V}_{m'}/n_{m'}\big\rangle\Big) \;=\; p_{im}\,\big\langle dO_i,\ \bar{V}_m/n_m - O_i\big\rangle. \quad (15)$$

The bracket is where the per-key value V_j disappears: it enters only through $\bar{V}_m = \sum_{j\in m} V_j$, so a sum over N keys has already become one over M clusters.

Through the logit. $\partial \ell_{im}/\partial \bar{K}_{m,d} = Q_{i,d}$, and n_m does not depend on $\bar{K}_m$. Summing Equation (15) over queries,

$$\frac{\partial L}{\partial \bar{K}_{m,d}} \;=\; \sum_i p_{im}\,\big\langle dO_i,\ \bar{V}_m/n_m - O_i\big\rangle\, Q_{i,d}. \quad (16)$$

Through the mean. The assignment $m(\cdot)$ is piecewise constant, so neither it nor n_m carries a gradient, and the only path from $\bar{K}_m$ back to a key is the mean itself: $\partial \bar{K}_{m,d}/\partial K_{j',d} = \mathbf{1}[m(j') = m]/n_m$. Composing with Equation (16) gives Equation (6).

Check. Summing Equation (6) over the n_m keys of a cluster returns Equation (16) exactly: the cluster's gradient is shared equally among its keys, and none of it is created or lost. Every factor on the right of Equation (6) is either per cluster ($\bar{V}_m$, n_m) or per query (p_{im}, O_i, dO_i, Q_i), and p_{im} is already formed in the forward pass, so the backward costs $\mathcal{O}(MND)$ like the forward. No $N \times N$ matrix is built at any point.

Algorithm 2 AVQ attention over cluster means with exact local keys, for one query tile I. Refinement runs in rounds of $\text{Block}_{\mathcal{P}}$ parents, $\mathbf{w}$ updated between rounds; the running maximum of the online softmax is left implicit. With codewords instead of means, $\bar{K}$ is C and the parent logit is rebuilt as $\frac{1}{\mathcal{C}} \sum_c \mathbf{s}_c$.

Input: Q, K, V; $\bar{K}, \bar{V}, n$ and each key's (p, c) from Algorithm 1
Output: $Y \in \mathbb{R}^{BH \times N \times D}$

for each (bh, I) **in parallel do** ▷ $\mathbf{q} = Q[bh, I]$ in registers
  $\mathbf{n}^{\text{all}} = n[bh]$; $\mathbf{n} = \mathbf{n}^{\text{all}} - \mathbf{n}^{I}$ ▷ $\mathbf{n}^{I}$: histogram of the tile's own (p, c)
  // Parents
  $\mathbf{s} = \mathbf{q}\,\bar{K}[bh, 0\!:\!M_0]^\top$; $\mathbf{A} = \exp(\mathbf{s})$; $\mathbf{A}_{:,m} = 0 \;\; \forall m\colon \mathbf{n}^{\text{all}}_m = 0$ ▷ $[\text{Block}_N, M_0]$
  $\bar{\mathbf{x}} = \mathbf{A}\,\bar{V}[bh, 0\!:\!M_0]$; $\bar{\mathbf{z}} = \sum_m \mathbf{A}_{:,m}\,\mathbf{n}_m$; $\mathbf{w}_m = \sum_i \mathbf{A}_{im}\,\mathbf{n}_m / \bar{\mathbf{z}}_i$ ▷ Equations (3) and (9)
  // Refine
  for each $p \in$ top-$\mathcal{P}(\mathbf{w})$ **do**
    $\mathbf{s}_c = \mathbf{q}\,\bar{K}[bh, \text{children}(p)]^\top$ ▷ $[\text{Block}_N, \mathcal{C}]$
    $\mathbf{s}_p = \sum_c (\mathbf{n}^{\text{all}}_c / \mathbf{n}^{\text{all}}_p)\,\mathbf{s}_c$ ▷ Equation (7)
    $\Delta\mathbf{A} = \exp(\mathbf{s}_c) - \exp(\mathbf{s}_p)$; $\Delta\mathbf{A}_{:,c} = 0 \;\; \forall c\colon \mathbf{n}^{\text{all}}_c = 0$ ▷ Equation (4)
    $\bar{\mathbf{x}} \mathrel{+}= \Delta\mathbf{A}\,\bar{V}[bh, \text{children}(p)]$; $\bar{\mathbf{z}} \mathrel{+}= \sum_c \Delta\mathbf{A}_{:,c}\,\mathbf{n}_c$
  end for
  // Exact local keys
  $\mathbf{k}, \mathbf{v} = K[bh, I],\, V[bh, I]$; $m(j) =$ key j's c if its p was refined, else its p
  $\mathbf{A}^{\text{loc}} = \exp(\mathbf{q}\,\mathbf{k}^\top)$; $\bar{\mathbf{x}} \mathrel{+}= \left(\mathbf{A}^{\text{loc}} - \exp(\mathbf{q}\,\bar{K}[bh, m(\cdot)]^\top)\right)\mathbf{v}$; $\bar{\mathbf{z}} \mathrel{+}= \sum_j \mathbf{A}^{\text{loc}}_{:,j}$ ▷ Equation (10)
  $Y[bh, I] = \bar{\mathbf{x}} \,/\, \bar{\mathbf{z}}$
end for