DISSOCIATED RECURRENT CLUSTERING

Anonymous authors
Paper under double-blind review

ABSTRACT

The current status quo for large-scale recurrent models is to restrict the recurrence to linear operations, which are associative and can be implemented efficiently. We depart from this key constraint with the Dissociated Recurrent Clustering (DRC), whose output is a standard attention over a recurrent state composed of a set of weighted key/value centroids that are iteratively fused to keep their number fixed. To make this process parallel during training, we introduce the "dissociation" of the computation: the centroids to fuse are computed on very large batches without autograd and, once they are fixed, the forward and the backward can be formulated as parallel scans. Experiments with hybrid 1.5B models trained on sequences of 32k tokens and evaluated on standard benchmarks, without hyper-parameter tuning, show that using DRC with 1k centroids instead of Gated DeltaNet with an equivalent recurrent state size reduces the gap to using a standard full-context attention layer by a factor 2 in term of training loss and 3.5 in validation loss on source code, known to require long context operators.

1 INTRODUCTION

The standard attention operator in transformers lets a token attend to any previous token (Vaswani et al., 2017). Its state and its cost therefore grow with the context: the key/value cache holds one entry per token seen, and a step attends to all of them.

Recurrent layers instead keep a state of fixed size. Linear attention drops the softmax and accumulates a matrix (Schmidhuber, 1992; Katharopoulos et al., 2020), which can be gated (Yang et al., 2024a), decayed (Gu and Dao, 2024), or updated with a delta rule (Yang et al., 2024b; 2025). In all of them the state is a sum of outer products, so every token is superposed with all the others, and the read is linear.

Another way is to keep an explicit set of memory slots, and to bound it by iterative compression. The Compressive Transformer (Rae et al., 2020) pools old activations, the ∞-former (Martins et al., 2022) projects them onto a continuous basis, and cache eviction (Zhang et al., 2023) drops the tokens that receive little attention. They all compress at a rate fixed in advance, or only at inference.

Clustering a stream under a memory bound is an old problem. MacQueen (1967) introduced k-means with its sequential form, where every new point joins the nearest centroid and moves it, an update Bottou and Bengio (1994) read as a gradient descent. When the stream is unbounded, the standard device is to keep a bounded summary and merge points as they arrive (Guha et al., 2000; Charikar et al., 2003; Liberty et al., 2016). It has been used inside attention too: Vyas et al. (2020) group the queries to approximate the attention matrix, and Roy et al. (2021) route queries and keys to centroids maintained online. Both keep every key; here the centroids are the state.

Dissociated Recurrent Clustering (DRC) belongs to that second family. Its state is a set of R weighted key/value centroids, whose size is kept constant by fusing, at every token, the pair whose fusion increases the clustering distortion the least. Ward (1963) introduced that criterion, and Zhang et al. (1996) used it online. We apply it to the keys of a layer, and train it end to end. The read is a softmax over the centroids, so a centroid remains individually addressable.

These fusions depend non-linearly on the state, contrary to linear attention, so the recurrence is fundamentally serial and no associative scan applies to it. However, an associative scan is possible as soon as the fusions are fixed, and that is what we exploit: the "dissociation" trick computes them

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.

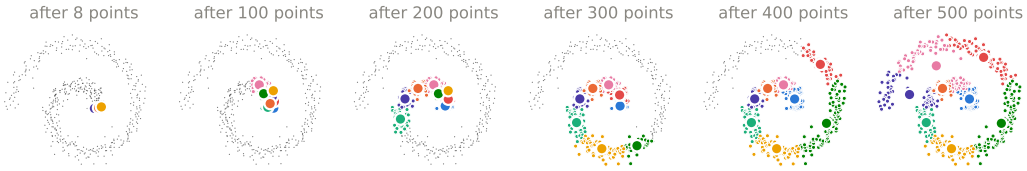


Figure 1: The update of § 2.2 on a toy example: 500 points on a spiral, processed in order, with $R = 8$ and $w_t = 1$, which makes the fusion cost (9) the Ward criterion. Each panel is the state after a given number of points. The tiny black dots are the points not seen yet, the colored dots are the points already processed, colored by cluster, and the large dots are the centroids (3). A cluster keeps its color across panels.

apart, without autograd, on batches large enough to fill the GPU. They are then refreshed far less often than the activations, which experiments show costs nothing measurable.

Our contributions are:

- a weighted online k-means as a memory mechanism (§ 2),
- a dissociation between the discrete choices of clusters to fuse and the differentiable computation that uses them, which lets us fully utilize the GPU cores (§ 3.2), and
- a key/value sequence representation that allows a fast attention implementation, even though the set of key/value pairs depends on t (§ 3.3).

Functionally, the layer is simple: the state is neither gated nor decayed, the only forgetting is the fusion itself, and the number of centroids is its one hyper-parameter. The implementation is not, and that is the price of using the GPU properly.

2 Recurrent Clustering

The DRC layer has the usual W_Q, W_K, W_V and output projection W_O, all bias-free, plus one map of its own, W_w, for a memorization weight. Everything below is per key/value head: one clustering, which every query head attached to it reads.

It takes as input a tensor $X \in \mathbb{R}^{T \times D}$, normalized as $\hat{X}_t = \mathrm{rmsnorm}(X_t)$, and computes for every t

$$Q_t = \mathrm{rmsnorm}\big(W_Q \hat{X}_t\big), \quad K_t = \mathrm{rmsnorm}\big(W_K \hat{X}_t\big), \quad V_t = \mathrm{rmsnorm}\big(W_V \hat{X}_t\big), \tag{1}$$

and a weight

$$w_t = \exp\big(W_w \hat{X}_t + b_w\big) \in \mathbb{R}_{>0} \tag{2}$$

that modulates how much the token is worth memorizing.

The layer carries no positional encoding. The clustering groups the tokens by content alone, and the only order it keeps is whatever the keys carry. We tried three ways of adding one: a sinusoidal encoding fed into the keys as is, gated by an amplitude, or gated by the cutoff of a low-pass filter. None of them improves on the plain layer everywhere, and Appendix F reports the three.

The layer maintains through time a set of R weighted centroids:

$$\big(\hat{w}_{t,r},\ \hat{K}_{t,r},\ \hat{V}_{t,r}\big), \qquad r = 1, \dots, R. \tag{3}$$

The first R tokens and their weights initialize the centroids. After that, at every t, the layer takes the new key as one more centroid, and merges the cheapest two to get back to R (§ 2.1 and § 2.2). Every key centroid $\hat{K}_{t,r}$ carries a value centroid $\hat{V}_{t,r}$, updated the same way. Figure 1 shows the process on a toy two-dimensional example.

The output Y_t is an attention of Q_t over the centroids:

$$Y_t = \sum_{r=1}^{R} \mathrm{softmax}_r\big(Q_t^\top \hat{K}_{t,r} / \sqrt{d_k} + \log \hat{w}_{t,r}\big)\, \hat{V}_{t,r}. \tag{4}$$

The $\log \hat{w}_{t,r}$ term makes the attention weight of a centroid proportional to $\hat{w}_{t,r} \exp(Q_t^\top \hat{K}_{t,r}/\sqrt{d_k})$. So a centroid receives the attention that the tokens it summarizes would have received individually.

Two guards keep control arithmetic issues related to the weight: the implementation clamps $W_w \hat{X}_t + b_w$ at 30 before the exponential in (2), and floors $\hat{w}$ in the logarithm of (4).

If the keys of a cluster are all close to their centroid, then

$$\sum_{s \in \mathcal{C}_r} w_s \exp(Q_t^\top K_s/\sqrt{d_k}) \;\simeq\; \hat{w}_{t,r} \exp(Q_t^\top \hat{K}_{t,r}/\sqrt{d_k}), \tag{5}$$

and (4) approximates the attention over all the tokens seen so far. How good that approximation is depends on the distortion, which is what the fusion keeps small.

Trained as described, the layer gives the first token an enormous weight. This is the attention sink observed in transformers (Xiao et al., 2024; Darcet et al., 2024), and it makes that token a bad candidate for any fusion. So we dedicate a cluster to it: we keep the first token as an immutable singleton, and the state is that protected centroid plus $R-1$ adaptive ones.

The formulas that follow ignore it, to stay readable. It is simply excluded from the candidates of § 2.2, both for a fusion and for a join.

2.1 Fusion cost

A centroid r summarizes a set $\mathcal{C}_r \subset \{1, \dots, t\}$ of tokens, and is the weighted mean of that set:

$$\hat{w}_r = \sum_{s \in \mathcal{C}_r} w_s, \qquad \hat{K}_r = \frac{1}{\hat{w}_r} \sum_{s \in \mathcal{C}_r} w_s K_s, \qquad \hat{V}_r = \frac{1}{\hat{w}_r} \sum_{s \in \mathcal{C}_r} w_s V_s. \tag{6}$$

The quantity to keep small is the weighted distortion of $\{\mathcal{C}_1, \dots, \mathcal{C}_R\}$, measured on the keys,

$$\mathcal{D} = \sum_{r=1}^{R} \sum_{s \in \mathcal{C}_r} w_s \left\| K_s - \hat{K}_r \right\|^2, \tag{7}$$

which is the weighted k-means objective. The layer never moves a token from one cluster to another. So a key can end up closer to another centroid than to its own, and $\mathcal{D}$ is then above what the same centroids would give with a nearest-centroid assignment.

Fusing two clusters a and b replaces them with the single cluster $\mathcal{C}_a \cup \mathcal{C}_b$, and from (6):

$$\hat{w}_{ab} = \hat{w}_a + \hat{w}_b, \qquad \hat{K}_{ab} = \frac{\hat{w}_a \hat{K}_a + \hat{w}_b \hat{K}_b}{\hat{w}_{ab}}, \qquad \hat{V}_{ab} = \frac{\hat{w}_a \hat{V}_a + \hat{w}_b \hat{V}_b}{\hat{w}_{ab}}. \tag{8}$$

A centroid is not renormalized after fusion. It remains the weighted mean in (7), so disagreement among its normalized keys reduces its norm.

The cost of a fusion is the increase of (7) that it induces. Write $\Delta(a, b)$ for the increase incurred by replacing the clusters a and b by their union. It has a simple closed form, and involves only the two centroids being fused,

$$\Delta(a, b) = \frac{\hat{w}_a \, \hat{w}_b}{\hat{w}_a + \hat{w}_b} \left\| \hat{K}_a - \hat{K}_b \right\|^2. \tag{9}$$

This is Proposition 1, stated and proved in Appendix B. Note that (9) involves the keys only. Equation (8) carries the value centroids along, but they play no part in the fusion decision.

2.2 Update

At step t the layer treats the new token as a singleton cluster of weight w_t, which gives the augmented set of $R+1$ candidates

$$\mathcal{S}_t = \left\{ (\hat{w}_{t-1,r}, \hat{K}_{t-1,r}, \hat{V}_{t-1,r}) \right\}_{r=1}^{R} \cup \left\{ (w_t, K_t, V_t) \right\}, \tag{10}$$

and the layer fuses the pair of minimal cost,

$$(a^\star, b^\star) = \operatorname*{arg\,min}_{\substack{a, b \in \mathcal{S}_t \\ a \neq b}} \Delta(a, b), \tag{11}$$

then replaces that pair by its weighted mean (8), which brings the state back to R centroids. There are two cases. Either the new token is in that pair, and it joins a centroid, or it is not, and the layer fuses two centroids while the token becomes a centroid of its own.

We compute the argmin (11) as two argmins, one per case. Let (i_f, j_f) be the cheapest pair of centroids to fuse, and $i_\star$ the cheapest centroid for the token to join,

$$(i_f, j_f) = \arg\min_{i<j} \frac{\hat{w}_i \hat{w}_j}{\hat{w}_i + \hat{w}_j} \left\| \hat{K}_i - \hat{K}_j \right\|^2, \qquad i_\star = \arg\min_{i} \frac{\hat{w}_i \, w_t}{\hat{w}_i + w_t} \left\| \hat{K}_i - K_t \right\|^2. \tag{12}$$

If the first cost is the smaller, the layer fuses i_f and j_f, and the token takes the slot they free. Otherwise the token joins $i_\star$. Ties are resolved in favor of joining. Initialization requires no separate rule: until R tokens have arrived, the augmented set contains at most R elements, so no fusion is performed.

Computing the fusion cost naively is $O(R^2)$ per token, which would dominate everything else. Appendix C gives an exact kernel that does it in $O(R)$ of serial work per token, and in the setting of § 3.2 the fusions cost $0.12\times$ a full attention layer.

Algorithm 1 summarizes the layer.

3 Discrete Decisions Dissociation

The key problem with the scheme of § 2 is that it is intrinsically serial and consequently cannot take advantage of the multiple cores of a GPU during training.

3.1 A scan once the decisions are fixed

The standard way to parallelize a serial recurrence is the associative, or prefix, scan (Kogge and Stone, 1973; Blelloch, 1990). It needs the maps φ_t taking the state Z from one step to the next to compose *associatively*, and a composition of any number of them to be representable in a size that does not grow with t. The prefixes $\varphi_t \circ \cdots \circ \varphi_1$ then follow from a binary tree, in $O(\log T)$ steps instead of T. In practice the φ_t are affine,

$$Z_t = A_t \, Z_{t-1} + B_t, \tag{13}$$

which meets both conditions, since affine maps compose into affine maps and a composition is one more pair (A, B). This is how what makes it possible to train at a high MFU linear attention, GLA, Mamba, Gated DeltaNet and other state space models.

The recurrence of § 2.2 does not have this property. Its argmin picks a different behavior for every candidate pair, so the update is piecewise affine, and composing such maps gives a functional with no representation of size constant in t. Associativity is not the problem, since composition is always associative: the problem is representation size. Each argmin introduces new branches, so representing a composition of t updates requires an object whose size grows with t. There is therefore no fixed-size monoid representation to scan over.

That changes as soon as the fusions are given. The trick is to store the centroids as unnormalized moments instead of means. With $d = 1 + d_k + d_v$, let

$$z_{t,r} = \left(\hat{w}_{t,r}, \; \hat{w}_{t,r} \hat{K}_{t,r}, \; \hat{w}_{t,r} \hat{V}_{t,r} \right) = \sum_{s \in \mathcal{C}_{t,r}} w_s \, (1, K_s, V_s) \; \in \; \mathbb{R}^{1 \times d}, \tag{14}$$

where the second equality comes from (6). Stack these into $Z_t \in \mathbb{R}^{R \times d}$, and let $\zeta_t = w_t \, (1, K_t, V_t)$ be the row the incoming token contributes. A moment has no denominator to normalize, so the fusion (8) is the *sum* of two rows.

A step is then one matrix product. Stack the state and that row into the augmented set (10) of $R+1$ candidates,

$$\tilde{Z}_t = \begin{bmatrix} Z_{t-1} \\ \zeta_t \end{bmatrix} \in \mathbb{R}^{(R+1) \times d}, \tag{15}$$

of which the layer merges exactly one pair: the two centroids it fuses, or the token and the centroid it joins. With $\sigma_t : \{1, \ldots, R+1\} \to \{1, \ldots, R\}$ sending each candidate to the slot it lands in,

$$Z_t = M_t \, \tilde{Z}_t, \qquad (M_t)_{r,j} = 1 \;\text{ if }\; \sigma_t(j) = r, \;\text{ and } 0 \;\text{ otherwise.} \tag{16}$$

So fusing is a left multiplication of the centroids by a $0/1$ matrix that passes $R-1$ candidates through unchanged and adds the merged pair into a single row, with one non-zero entry per column, determined by the decisions alone (Proposition 2). Splitting off its last column, $M_t = [\, A_t \quad c_t \,]$, gives (13) with $B_t = c_t\zeta_t$ of rank one, so the scan applies.

Equipping the pairs $(A, C) \in \mathbb{R}^{R\times R} \times \mathbb{R}^{R\times d}$ with

$$(A, C) \odot (A', C') = (A'A,\ A'C + C') \tag{17}$$

gives an associative product with neutral element $(I, 0)$, whose prefixes $\Phi_t = (\bar{A}_t, \bar{C}_t)$ satisfy $Z_t = \bar{A}_t Z_0 + \bar{C}_t$ (Proposition 3). Both results are standard, the first being the additivity of the clustering features of Zhang et al. (1996) and the second the usual monoid for an affine recurrence, and Appendix B proves them.

All the φ_t are built in parallel, so a Blelloch scan (Blelloch, 1990) gives every Z_t in $O(\log T)$ steps, and dividing each row by its first coordinate returns the centroids. A matrix with one non-zero entry per column is a map, and such matrices are closed under composition, so a partial product costs R indices instead of R^2 entries; only $\bar{C}_t$ is dense, which is why the scan runs chunkwise. Finally (16) is linear in Z_{t-1} and ζ_t, hence differentiable in w_t, K_t and V_t, although the fusions themselves are never differentiated.

3.2 Dissociation

By Proposition 3, the forward/backward can be computed by a scan as soon as the fusions are known. Computing them is the only serial part left. It cannot be parallelized over t, but it is independent from one sequence and one head to the next, and it does not require autograd. So we run it on many sequences at once: hundreds of independent serial streams keep the GPU busy, and nothing has to be kept for a backward pass. What bounds their number is the workspace of the kernel, 4.56 MiB per sequence and head, so 400 sequences on two heads take 3.6 GiB. And since it runs separately, its result can be reused for many optimizer steps.

Training is thus split into three parts.

1. **Prefix.** The layers before the DRC layer are run without autograd on 400 sequences, 100 at a time. This gives the X_t that the layer will see, hence the w_t and the K_t that the fusions need. The values play no part in the decision, and are not needed here.
2. **Fusions.** The pairs (11) of clusters to fuse are computed as in § 2.2 for those 400 sequences, also without autograd, and cached.
3. **Replay.** The ordinary optimizer step runs on 4 sequences, reads the indexes of the clusters to fuse from the cache, and implements the forward/backward with parallel scans (§ 3.3).

We call the first two parts the *collection*. A cache of 400 sequences serves the next 100 steps of 4 sequences, so each step pays a hundredth of it. The prefix is an ordinary dense forward, and runs at the MFU of the rest of the model. The fusions are serial and of low arithmetic intensity, but they are 3% of the total. That last figure is what the kernel of Appendix C buys, by going through the tokens in chunks and updating distances instead of centroids.

One prefix serves all the DRC layers together, since a forward reaches every depth. Only the fusions and the replay are per layer, and the fusions are the cheap part.

The drawback is that a step uses fusions computed up to 100 steps earlier. This is not very damaging: the clusters are slightly out of date, but the gradient still steers K, V, and w in the right direction. With a 1.5B model and a delay of 100 steps, the cross-entropy remains within 4×10^{-4} of the same model trained without dissociation (Appendix E). Appendix G runs the same trick on a small ConvNet, where the discrete decisions are the ReLU signs and the pooling argmaxes.

What the three parts cost, on one H100, for an optimizer step of 4 sequences of length 8k: the prefix 98.1 ms, the fusions 6.8 ms, and the replay 112.5 ms, so 217.4 ms for the whole DRC block, projections and feed-forward included. The first two are already divided by the 100 steps their cache serves. The prefix dominates, at 45% of the total and 93% of the collection, so that is where any further gain has to come from. For comparison, a full attention block in the same slot takes 57.0 ms, a sliding window of 64 tokens 24.6 ms, and GDN-2 43.4 ms.

On a whole 1.5B model of the kind we train, 27 windowed layers and one long-range layer, that puts a step at 882 ms against 722 ms for the full-attention hybrid and 709 ms for the GDN-2 one, so $1.22\times$ and $1.25\times$. Generation is a different matter, since there is no future batch whose decisions could be collected in advance and the layer just runs serially: 14.08 ms per token against 13.94 ms for GDN-2, a difference of 1%. What DRC pays, it pays at training time.

3.3 Replay as a single attention

The replay needs the R centroids of every step, and storing them all would take a $T \times R$ tensor, eight million centroids per stream with $T = 8192$ and $R = 1024$. The state barely moves, though: a fusion writes at most two slots, the one that receives the fused pair and, when two existing centroids are fused, the one freed for the new token.

So we store the writes instead of the states. Token t writes three *events*, each of them the new content $(\hat{w}, \hat{w}\hat{K}, \hat{w}\hat{V})$ of one slot, that is its row of (14). Two are the slots that a fusion can change. The third is a round-robin refresh, which rewrites an untouched slot with its current content. Stacking them gives a tensor $E \in \mathbb{R}^{3T\times d}$.

The refresh changes nothing, but it is what makes E useful: it cycles through the slots, so a slot the fusions leave alone is still rewritten every R tokens, and consequently

$$\left\{(\hat{w}_{t,r}, \hat{K}_{t,r}, \hat{V}_{t,r})\right\}_{r=1}^{R} \subset \left\{E_{3t-3R+1}, \ldots, E_{3t}\right\}. \tag{18}$$

The last $3R$ rows of E always contain the R centroids of step t, along with stale versions of them. Every event carries the time it is written and the time it is overwritten, so a mask drops those not alive at t.

E is therefore the key/value sequence of a single attention with a window of $3R$. For a query t the mask keeps exactly R rows, one per slot, which selects the state (3) without ever materializing it. Adding $\log \hat{w}$ to the score gives (4) for every t with one block-sparse attention, which FlexAttention (Dong et al., 2024) compiles into a fused kernel in the manner of FlashAttention (Dao et al., 2022). The backward is a reverse scan over the fixed fusion graph, so autograd never walks a token loop.

4 Experiments

Most of the experiments compare Qwen-like models of 1.5B parameters and 28 layers, trained on sequences of 8,192 tokens, that differ by a single layer. We train every model the same way: 4 nodes $\times$ 8 H100s $\times$ 4 sequences $\times$ 8,192 tokens $\times$ 45,629 steps $\simeq$ 48B tokens. The learning rate is 3×10^{-3}, with a linear warmup over 2,000 steps and a cosine annealing down to 3×10^{-5}.

§ 4.1 gives the loss, the standard evaluations and the retrieval scores on RULER. § 4.2 then changes the context length and the number of DRC layers, and § 4.3 the number of centroids. Appendix E ablates the dissociation and the token weight, and Appendix F reports three positional variants of the layer, none of which is a clear win. DRC comes out ahead of Gated DeltaNet at a comparable state size, and by a wide margin on retrieval.

In the tables, $n\times$X means n layers of type X. In every hybrid the single non-SWA layer sits at the same depth, layer 14. We combine four types of layers:

FCA is full causal attention (Vaswani et al., 2017). Its key/value cache grows with the context, so it is the upper reference rather than a competitor.

SWA is sliding-window attention, with a window of $W = 64$ tokens. It is the backbone of every hybrid, and $28\times$SWA alone is the lower reference.

GDN-2 is the second-generation Gated DeltaNet (Yang et al., 2025). It extends the channel-wise decay of Kimi Team (2025) with separate channel-wise erase and write gates, and includes width-4 causal short convolutions; collapsing its decay to one scalar per head recovers the original Gated DeltaNet. It is the recurrent baseline, and the one to compare DRC with. Its state is 4 heads $\times$ 256 $\dim_K$ $\times$ 256 $\dim_V$ = 262,144 floats.

Table 1: The 8k family. The last column is the relative variation of DRC with respect to GDN-2. It is a raw variation, so a negative entry under *Losses* means DRC is ahead. All SWA layers use $W = 64$. Evaluations at step 45,629. GDN-2 has a recurrent state of 262,144 floats and 1.513B parameters, DRC 264,192 floats and 1.505B. The ten multiple-choice tasks behind the average, and the thirteen RULER tasks behind theirs, are in Appendix D.

Metric	28×FCA	27×SWA +1×FCA	28×SWA	27×SWA +1×GDN-2	27×SWA +1×DRC	DRC vs GDN-2
			Losses, lower is better			
Final 100-step train loss	2.0936	2.1381	2.2928	2.1621	2.1540	−0.4%
			Accuracy, higher is better			
MC average	49.50	48.78	46.92	48.30	48.54	+0.5%
RULER 8k average	47.75	39.02	6.22	24.11	37.29	+54.7%
HumanEval+ pass@1	9.15	5.49	4.88	1.83	6.10	+233.3%
HumanEval+ HE-pass@1	10.37	7.32	6.71	3.05	7.32	+140.0%
MBPP pass@1	13.40	7.80	5.60	0.00	6.80	—
TriviaQA EM	20.76	20.12	15.76	16.54	19.15	+15.8%
Natural Questions EM	8.37	7.23	5.68	6.12	7.09	+15.8%

DRC is the layer of § 2. Its state is 2 key/value heads × 1,024 centroids × (64 $\dim_K$ + 64 $\dim_V$ + 1 weight) = 264,192 floats, read by 12 query heads. Everything below uses the plain layer, and Appendix F reports the three positional variants.

4.1 Loss, evaluations and retrieval

Table 1 gives the final training loss, the standard evaluations and the retrieval scores on RULER at the training length of 8k. It says three things.

The loss orders the five models by how much attention they keep. Full attention everywhere is best at 2.0936, one full layer among 27 windowed ones costs 0.0445, and dropping that layer costs 0.1547 more. A recurrent layer in the same slot recovers most of what the window loses, GDN-2 to within 0.0240 of the hybrid and DRC to within 0.0159.

The multiple-choice benchmarks do not separate the two. DRC is at 48.54 against GDN-2's 48.30, half a percent, and both sit between the sliding window and the hybrid. These tasks fit in a 64-token window, so they measure the backbone, not the long-range layer. The generative ones behave the same way, except where the answer has to be retrieved: TriviaQA and Natural Questions give DRC 16% over GDN-2.

Retrieval is where the state design shows. On RULER at 8k, the thirteen tasks of Table D.2, DRC averages 37.29 against 24.11 for GDN-2, a state of the same size, and 39.02 for a full-attention layer whose KV cache grows with the context. So DRC keeps 96% of what a growing cache retrieves while GDN-2 keeps 62%. The gain is on the needles, where a centroid stays addressable: 100.00 against 27.00 on NIAH single 2, 83.60 against 16.40 on single 3. The two losses are VT and FWE, the tasks that need the order of the tokens, which the layer does not represent. Appendix F gives three positional variants; adding the encoding to the keys takes VT from 0.56 to 32.36 and FWE from 21.65 to 35.80, and the average from 37.29 to 39.12, at the cost of the needles and of MMLU.

At 4k, half the training length, the same 13 tasks give 42.45 for DRC, 31.47 for GDN-2, and 42.52 for the full-attention layer, which DRC matches.

4.2 Longer context, several layers

Everything above has one long-range layer and sequences of 8k. A second family keeps the 1.5B architecture and trains on sequences of 32,768, with seven long-range layers: seven repetitions of three SWA-128 layers and one layer under test. The budget is the one of § 4, 47.8B tokens in 45,629 steps. The states are matched, 524,288 floats per layer for a window of 1,024 tokens, 540,672 for GDN-2, and 526,336 for DRC with 1,024 centroids of dimension 128. GDN-2 also carries 4.8% more parameters, since it buys its state with value heads.

Table 2: The 32k family, all four models being $7\times(3\times\text{SWA-128}+1\times\text{X})$. GitHub Neo is a held-out set of source code. The last column is the relative variation of DRC with respect to GDN-2, raw, so a negative entry under *Losses* means DRC is ahead. Evaluations at step 45,629. GDN-2 has a recurrent state of 540,672 floats per layer, DRC 526,336 with 1,024 centroids of dimension 128, and GDN-2 carries 4.8% more parameters. The full table is in Appendix D.

Metric	21×SWA-128 + 7×FCA	21×SWA-128 + 7×SWA-1K	21×SWA-128 + 7×GDN-2	21×SWA-128 + 7×DRC	DRC vs GDN-2
			Losses, lower is better		
Final 100-step train loss	2.0725	2.1500	2.0894	2.0807	−0.4%
GitHub Neo loss	0.7356	0.9055	0.7833	0.7489	−4.4%
GitHub Neo perplexity	2.0867	2.4731	2.1887	2.1147	−3.4%
			Accuracy, higher is better		
MC average	48.80	48.39	48.53	48.90	+0.8%
HumanEval+ pass@1	7.32	6.10	6.71	9.15	+36.4%
HumanEval+ HE-pass@1	9.15	7.32	9.76	10.98	+12.5%
MBPP pass@1	10.20	9.80	10.80	11.40	+5.6%
TriviaQA EM	19.20	18.89	20.44	19.29	−5.6%
Natural Questions EM	7.26	6.95	7.73	7.26	−6.1%

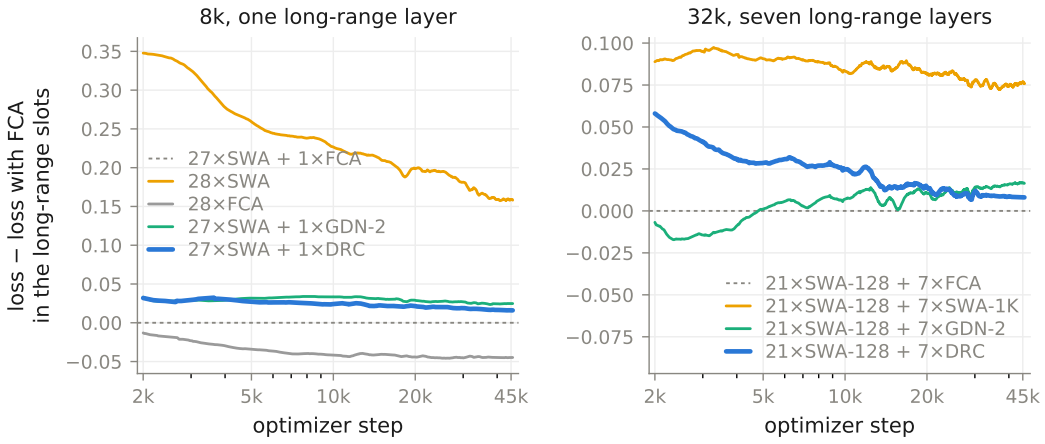


Figure 2: Training loss minus the loss of the model that has full attention in the same long-range slots, so $27\times\text{SWA}+1\times\text{FCA}$ on the left and $7\times(3\times\text{SWA-128}+1\times\text{FCA})$ on the right. The zero line is that model. $28\times$FCA on the left is full attention in every layer, given for reference. The curves are averaged over a trailing 1,000 steps, as the tables are. The scales differ between the two plots.

Figure 2 gives both families, each against the model that has full attention in the same slots. Removing the long-range layer costs 0.155 at 8k, and replacing it with a 1,024-token window costs 0.077 at 32k. DRC recovers 90% of the first and 89% of the second, GDN-2 85% and 78%, so one DRC layer among 28 and seven among 28 end within 0.016 and 0.008 of full attention. They do not get there the same way, and Figure 2 shows the two trajectories.

The training loss averages over everything the model reads. A held-out set of source code separates the four 32k models better, since code is where a long context pays. On GitHub Neo (Table 2) the perplexities are 2.087 with full attention in the long-range slots, 2.473 with a 1,024-token window, 2.189 for GDN-2 and 2.115 for DRC. In loss, DRC ends 0.013 above full attention and GDN-2 0.048, so DRC closes 92% of what the window leaves on the table against GDN-2's 72%, and its residual gap is 3.6 times smaller. The separation is twice what the training loss shows.

4.3 State size

The 1.5B runs all use 1,024 centroids, so they say nothing about how many are needed. A 470M sweep does. It has sixteen layers, four of them long-range, and repeats the same run at six state sizes, 128 to 4,096, for a sliding window and for DRC. Neither pays parameters for its state: a window has none, and the DRC projections do not depend on R. We leave GDN-2 out of this comparison, since it buys its state with value heads, which costs it up to 24% more parameters at 4,096.

Figure 3 inverts the two loss curves. At the end of training 128 centroids match a window of 1,134 tokens, and 1,017 match a window of 4,096. A centroid is thus worth between four and nine tokens of window. The rate falls as the state grows, because DRC saturates: it gains 0.006 of loss from 1,024 centroids to 4,096, where the window gains 0.046.

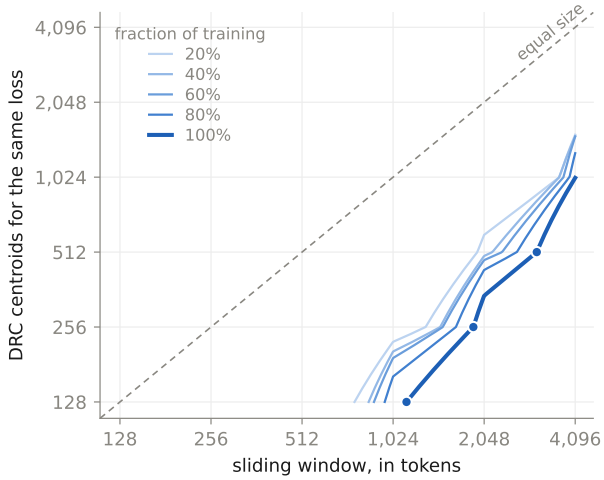


Figure 3: How much sliding window one DRC centroid replaces, on a 470M model with four long-range layers among sixteen. For every loss that both reach, the curve gives the window and the number of centroids that reach it, interpolating each loss piecewise-linearly in the state size. The dots are the measured DRC runs. The curve covers the losses both reach, which is why it starts at a window of 1,134.

5 Conclusion

DRC keeps a bounded set of weighted key/value centroids, and fuses at every token the pair that costs the least distortion. Its state is not a sum of outer products but a set of items, each of them still addressable on its own. The content of the sequence decides what it forgets, and not a gate.

That decision is discrete and depends on the state, so the layer is not an associative scan. It becomes one as soon as the fusions are given, which is what the dissociation exploits: compute them apart, without autograd, on hundreds of sequences at once, and reuse them for a hundred optimizer steps. What remains is a linear recurrence, and the event representation of § 3.3 makes it a single block-sparse attention.

At 1.5B parameters, one DRC layer among 27 sliding-window ones is ahead of a Gated DeltaNet of the same state size, on the loss and on the standard evaluations. On RULER at 8k it is thirteen points ahead, and within two points of a full-attention layer whose cache grows with the context. Training the model costs $1.22\times$ a full-attention one, and generating with it costs what a Gated DeltaNet costs.

What is left open is the scaling. We trained the layer at 1.5B only. § 4.3 varies the number of centroids and § 4.2 the number of layers and the context, but at a fixed model size, and we have not tested how any of them behaves past its training length. Giving it a sense of position only trades tasks against each other so far (Appendix F). And the prefix is 45% of what the layer costs at training time, spent on activations that the hundred steps it serves will compute again. That part is duplicated work, not a price of the method.

Reproducibility Statement

The method we propose is simple and can be integrated in a standard decoder transformer. It does not require substantial changes of learning procedure or data fetching.

AI Use Statement

We used generative AI tools to produce initial drafts of substantial portions of the manuscript and to revise them through multiple iterations under detailed author direction, primarily to improve clarity, structure, and concision. We used generative AI tools to implement the experiments, and create plots and tables. We did not use generative AI tools to draw scientific conclusions; other required disclosure categories are not applicable. Additionally, we used generative AI tools for proofreading and language editing. The authors reviewed and edited all AI-assisted text and verified the equations, technical claims, numerical results, and citations against the implementation, experimental records, and cited sources. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, Carnegie Mellon University, 1990.

L. Bottou and Y. Bengio. Convergence properties of the k-means algorithms. In NIPS, 1994.

M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In STOC, 2003.

T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022.

T. Darcet, M. Oquab, J. Mairal, and P. Bojanowski. Vision transformers need registers. In *ICLR*, 2024.

J. Dong, B. Feng, D. Guessous, Y. Liang, and H. He. Flex attention: A programming model for generating optimized attention kernels. *arXiv:2412.05496*, 2024.

A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *COLM*, 2024.

S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In FOCS, 2000.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

Kimi Team. Kimi Linear: An expressive, efficient attention architecture. arXiv:2510.26692, 2025.

P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, C-22(8):786–793, 1973.

G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies. I. Hierarchical systems. *The Computer Journal*, 9(4):373–380, 1967.

E. Liberty, R. Sriharsha, and M. Sviridenko. An algorithm for online k-means clustering. In ALENEX, 2016.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

P. H. Martins, Z. Marinho, and A. F. T. Martins. ∞-former: Infinite memory transformer. In *ACL*, 2022.

Algorithm 1 The DRC layer, in serial form. The dissociation of § 3.2 splits line 7, run without autograd, from the rest, which an associative scan then computes.

Require: $X \in \mathbb{R}^{T \times D}$, number of centroids R

1: $\mathcal{S} \leftarrow \emptyset$ ▷ the state, a set of at most R weighted centroids
2: **for** $t = 1, \dots, T$ **do**
3: $\quad Q_t, K_t, V_t \leftarrow$ the projections of $\hat{X}_t$ ▷ (1)
4: $\quad w_t \leftarrow \exp(W_w \hat{X}_t + b_w)$ ▷ (2)
5: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{(w_t, K_t, V_t)\}$ ▷ (10)
6: $\quad$ **if** $|\mathcal{S}| > R$ **then**
7: $\quad\quad (a^\star, b^\star) \leftarrow \arg\min_{a \neq b \in \mathcal{S}} \frac{\hat{w}_a \hat{w}_b}{\hat{w}_a + \hat{w}_b} \left\| \hat{K}_a - \hat{K}_b \right\|^2$ ▷ (9), (11)
8: $\quad\quad$ replace $a^\star$ and $b^\star$ in $\mathcal{S}$ by their weighted means ▷ (8)
9: $\quad$ **end if**
10: $\quad Y_t \leftarrow \sum_r \mathrm{softmax}_r\left(Q_t^\top \hat{K}_r / \sqrt{d_k} + \log \hat{w}_r\right) \hat{V}_r$ ▷ (4)
11: **end for**
12: **return** Y

J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. In *ICLR*, 2020.

A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention. In NeurIPS, 2020.

J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.

S. Yang, B. Wang, Y. Shen, R. Panda, and Y. Kim. Gated linear attention transformers with hardware-efficient training. In *ICML*, 2024.

S. Yang, B. Wang, Y. Zhang, Y. Shen, and Y. Kim. Parallelizing linear transformers with the delta rule over sequence length. In *NeurIPS*, 2024.

S. Yang, J. Kautz, and A. Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *ICLR*, 2025.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD*, 1996.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, 2023.

A Algorithm

B Proofs

Proposition 1 (Fusion cost)**.** *Let $\Delta(a, b)$ denote the increase of the distortion (7) incurred by replacing the clusters a and b by their union. Then $\Delta(a, b)$ is given by (9).*

Proof. For any cluster $\mathcal{C}$ of weight $\hat{w}$ and centroid $\hat{K}$, and any $z \in \mathbb{R}^{d_k}$, the weighted bias–variance decomposition gives

$$\sum_{s\in\mathcal{C}} w_s \|K_s - z\|^2 = \sum_{s\in\mathcal{C}} w_s \|K_s - \hat{K}\|^2 + \hat{w}\,\|\hat{K} - z\|^2, \tag{19}$$

since the cross term $-2\big(\sum_s w_s(K_s - \hat{K})\big)^\top(\hat{K} - z)$ vanishes by definition of $\hat{K}$. Applying (19) to $\mathcal{C}_a$ and to $\mathcal{C}_b$ with $z = \hat{K}_{ab}$, and subtracting the distortions of a and b taken separately,

$$\Delta(a,b) = \hat{w}_a\,\|\hat{K}_a - \hat{K}_{ab}\|^2 + \hat{w}_b\,\|\hat{K}_b - \hat{K}_{ab}\|^2. \tag{20}$$

From (8),

$$\hat{K}_{ab} - \hat{K}_a = \frac{\hat{w}_b}{\hat{w}_a + \hat{w}_b}\big(\hat{K}_b - \hat{K}_a\big), \qquad \hat{K}_{ab} - \hat{K}_b = \frac{\hat{w}_a}{\hat{w}_a + \hat{w}_b}\big(\hat{K}_a - \hat{K}_b\big), \tag{21}$$

so that

$$\Delta(a,b) = \frac{\hat{w}_a\hat{w}_b^2 + \hat{w}_b\hat{w}_a^2}{(\hat{w}_a + \hat{w}_b)^2}\,\big\|\hat{K}_a - \hat{K}_b\big\|^2 = \frac{\hat{w}_a\hat{w}_b}{\hat{w}_a + \hat{w}_b}\,\big\|\hat{K}_a - \hat{K}_b\big\|^2. \tag{22}$$

□

Proposition 2 (Linearity at fixed decisions)**.** *Let the decisions $(a_t^\star, b_t^\star)$ be fixed. Then the state obeys (16), with $M_t \in \{0,1\}^{R\times(R+1)}$ determined by those decisions alone and carrying exactly one non-zero entry per column.*

Proof. The selected pair determines σ_t, since σ_t identifies it and is injective elsewhere, and M_t is read off σ_t. By (14) the moment of a cluster is the sum of the moments of its members, so row r of Z_t is the sum of the rows of $\tilde{Z}_t$ whose image under σ_t is r, which is (16). Each column j carries the single non-zero entry $(M_t)_{\sigma_t(j),j}$, because σ_t is a map. □

Proposition 3 (Associativity)**.** *Equip the pairs $(A, C) \in \mathbb{R}^{R\times R} \times \mathbb{R}^{R\times d}$ with the product (17). Then $\odot$ is associative, with neutral element $(I, 0)$, and for $\varphi_t = (A_t,\, c_t\zeta_t)$ the prefix products $\Phi_t = \varphi_1 \odot \cdots \odot \varphi_t = (\bar{A}_t, \bar{C}_t)$ satisfy $Z_t = \bar{A}_t Z_0 + \bar{C}_t$.*

Proof. (A, C) represents the affine map $\psi : Z \mapsto AZ + C$, and $\odot$ is the composition of those maps in order: $\psi'(\psi(Z)) = A'(AZ + C) + C' = (A'A)Z + (A'C + C')$. Associativity of $\odot$ is then associativity of function composition, and $(I, 0)$ represents the identity. The last claim follows by induction, Z_t being the image of Z_0 under $\varphi_t \circ \cdots \circ \varphi_1$. □

C THE DECISION KERNEL

The fusions are the serial part, and the point of the dissociation is that they still have to keep the GPU busy. Two things make that possible: many independent streams, and a chunking that leaves only the decisions themselves serial.

A stream is one sequence and one key/value head. We sort the sequences by decreasing length and group similar ones, so that padding stays small, then flatten sequence and head into a single dimension. With two key/value heads, a launch of 100 streams carries 50 sequences.

Inside a stream, the tokens go by chunks of S. Entering a chunk, two parallel kernels compute everything it will need: the $S \times R$ distances between the new keys and the current centroids, and the $S(S-1)/2$ distances between the new keys. That is $O(SRd+S^2d)$ of dense work, at full occupancy. A single persistent program then takes the S decisions in sequence, with no kernel launch and no host round trip between two tokens.

A stream keeps one matrix of $(1 + R + S) \times R$ floats, allocated once and reused across chunks. Its regions are deliberately overloaded. Row 0 holds, for every column j, the smallest fusion cost against an $i < j$, so the best fusion is the smallest entry of that row. The upper triangle holds the centroid–centroid squared distances. The lower triangle, which $S \le R$ leaves unused, holds the token–token distances of the chunk. The last S rows hold the token–centroid distances. Three vectors complete it: the row achieving each column minimum, the second smallest cost of that column, and its row. The weighted key sums and the masses are kept apart.

Table D.1: The 8k family, every metric. The last column is the relative variation of DRC with respect to GDN-2. It is a raw variation, so it is negative for the loss, where lower is better. All SWA layers use $W = 64$. Evaluations at step 45,629. GDN-2 has a recurrent state of 262,144 floats and 1.513B parameters, DRC 264,192 floats and 1.505B.

Metric	28×FCA	27×SWA +1×FCA	28×SWA	27×SWA +1×GDN-2	27×SWA +1×DRC	DRC vs GDN-2
Final 100-step train loss ↓	2.0936	2.1381	2.2928	2.1621	2.1540	−0.4%
MC average ↑	49.50	48.78	46.92	48.30	48.54	+0.5%
HellaSwag ↑	59.39	58.39	54.17	58.24	58.63	+0.7%
WinoGrande ↑	60.22	60.06	56.51	58.96	59.19	+0.4%
ARC Easy ↑	58.90	61.18	58.31	60.63	59.49	−1.9%
ARC Challenge ↑	33.05	32.53	30.64	31.93	31.85	−0.3%
PIQA ↑	73.83	73.18	72.80	72.96	73.78	+1.1%
OpenBookQA ↑	36.80	34.80	36.40	37.20	35.80	−3.8%
RACE Middle ↑	52.23	48.40	44.71	46.10	48.05	+4.2%
RACE High ↑	38.79	37.02	34.39	35.96	36.05	+0.2%
CommonsenseQA ↑	18.59	19.74	19.33	18.26	20.39	+11.7%
BoolQ ↑	63.21	62.54	61.93	62.72	62.20	−0.8%
MMLU macro ↑	26.58	24.51	25.77	25.51	26.82	+5.1%
HumanEval+ pass@1 ↑	9.15	5.49	4.88	1.83	6.10	+233.3%
HumanEval+ HE-pass@1 ↑	10.37	7.32	6.71	3.05	7.32	+140.0%
MBPP pass@1 ↑	13.40	7.80	5.60	0.00	6.80	—
TriviaQA EM ↑	20.76	20.12	15.76	16.54	19.15	+15.8%
Natural Questions EM ↑	8.37	7.23	5.68	6.12	7.09	+15.8%

One decision is then two reductions. The kernel reduces over the R slots for the best join, reads the best fusion from row 0, compares the two, and records the pair.

The update is where the layout pays off. Merging two clusters a and b of masses $\hat{w}_a$ and $\hat{w}_b$ leaves, for any third point x,

$$d(a \cup b, x) = \alpha\, d(a, x) + \beta\, d(b, x) - \alpha\beta\, d(a, b), \qquad \alpha = \frac{\hat{w}_a}{\hat{w}_a + \hat{w}_b}, \quad \beta = 1 - \alpha, \tag{23}$$

with d the squared distance, which is the Lance–Williams update (Lance and Williams, 1967) for this criterion. So the kernel rewrites the row and the column of the changed slot, a second row and column when a fusion frees one, and the matching entries of every token row not yet processed, without a single new d-dimensional dot product. The distances computed at chunk entry are corrected as the centroids move, and stay exact.

The minima are then repaired. Only a cost involving a changed slot can differ, so for every column the kernel tries the cached minimum if its row did not change, the cached runner-up under the same condition, and the new costs against the one or two changed slots. It rescans a column in full only when those candidates cannot establish the best two with certainty. The result is the exact argmin, and not an approximate cache.

What is left per token is $O(R)$ of serial work, with no d factor, and the number of kernel launches is $O(T/S)$ instead of $O(T)$. With $R = 1{,}024$ centroids of dimension 64, one of them the anchor of § 2, and $S = 144$, the matrix is $1{,}168 \times 1{,}023$ floats, or 4.56 MiB per stream. Everything here runs in FP32: nothing downstream depends on the precision of these numbers, but the argmin does.

One thing is left on the table. An exact restriction of the join candidates is implemented, but not enabled: keeping, at chunk entry, the best $\min(R, 2S + 1)$ slots of every token is enough, since S decisions change at most $2S$ slots, so the true minimum is either one of those or one of the changed ones. The runs reported here reduce over all R.

D Full evaluations

The tables of § 4.1 and § 4.2 give the averages; these give every task behind them.

Table D.2: RULER at a context length of 8k. The last column is the relative variation of DRC with respect to GDN-2. All SWA layers use $W = 64$. GDN-2 has a recurrent state of 262,144 floats and 1.513B parameters, DRC 264,192 floats and 1.505B.

RULER 8k task	28×FCA	27×SWA +1×FCA	28×SWA	27×SWA +1×GDN-2	27×SWA +1×DRC	DRC vs GDN-2
NIAH single 1	100.00	100.00	4.40	100.00	100.00	0.0%
NIAH single 2	86.20	100.00	6.00	27.00	100.00	+270.4%
NIAH single 3	97.40	68.60	3.40	16.40	83.60	+409.8%
NIAH multikey 1	89.00	52.40	7.00	24.40	42.60	+74.6%
NIAH multikey 2	9.60	13.40	0.40	0.40	0.40	0.0%
NIAH multikey 3	0.60	5.80	0.00	0.00	0.00	—
NIAH multivalue	54.55	35.10	5.85	20.70	44.95	+117.1%
NIAH multiquery	69.95	34.80	5.55	17.05	47.20	+176.8%
VT	15.88	42.08	0.08	37.64	0.56	−98.5%
CWE	3.58	7.24	6.90	9.86	10.06	+2.0%
FWE	42.69	9.52	22.17	31.20	21.65	−30.6%
QA 1	23.94	16.87	7.93	10.60	15.73	+48.4%
QA 2	27.40	21.40	11.20	18.20	18.00	−1.1%
Average	47.75	39.02	6.22	24.11	37.29	+54.7%

Table D.3: The 32k family, every metric, all four models being $7\times(3\times$SWA-128 $+ 1\times$X$)$. GitHub Neo is a held-out set of source code. The last column is the relative variation of DRC with respect to GDN-2, raw, so it is negative for the two losses, where lower is better. Evaluations at step 45,629. GDN-2 has a recurrent state of 540,672 floats per layer, DRC 526,336 with 1,024 centroids of dimension 128, and GDN-2 carries 4.8% more parameters.

Metric	21×SWA-128 +7×FCA	21×SWA-128 +7×SWA-1K	21×SWA-128 +7×GDN-2	21×SWA-128 +7×DRC	DRC vs GDN-2
Final 100-step train loss ↓	2.0725	2.1500	2.0894	2.0807	−0.4%
MC average ↑	48.80	48.39	48.53	48.90	+0.8%
HellaSwag ↑	58.89	57.87	59.18	58.46	−1.2%
WinoGrande ↑	58.48	59.04	58.17	58.48	+0.5%
ARC Easy ↑	59.62	55.86	60.38	58.99	−2.3%
ARC Challenge ↑	31.85	29.96	32.53	31.67	−2.6%
PIQA ↑	73.23	73.07	73.18	73.72	+0.7%
OpenBookQA ↑	34.40	37.00	35.40	37.20	+5.1%
RACE Middle ↑	51.67	51.32	46.80	50.91	+8.8%
RACE High ↑	37.79	37.68	36.16	37.65	+4.1%
CommonsenseQA ↑	19.33	19.74	20.39	19.57	−4.0%
BoolQ ↑	62.78	62.39	63.06	62.39	−1.1%
MMLU macro ↑	25.81	25.66	27.14	24.93	−8.1%
HumanEval+ pass@1 ↑	7.32	6.10	6.71	9.15	+36.4%
HumanEval+ HE-pass@1 ↑	9.15	7.32	9.76	10.98	+12.5%
MBPP pass@1 ↑	10.20	9.80	10.80	11.40	+5.6%
TriviaQA EM ↑	19.20	18.89	20.44	19.29	−5.6%
Natural Questions EM ↑	7.26	6.95	7.73	7.26	−6.1%
GitHub Neo loss ↓	0.7356	0.9055	0.7833	0.7489	−4.4%
GitHub Neo perplexity ↓	2.087	2.473	2.189	2.115	−3.4%

E Ablations

Three controls, each differing from DRC in one respect only. All run in the 1.5B setting of § 4, on the same data and for the same 45,629 steps, and all are evaluated at that last step. Table E.1 gathers them.

Table E.1: Loss and evaluations of the three controls, next to DRC. *No dissociation* recomputes the fusions at every optimizer step; *no weight* fixes $w_t = 1$; *keys and values* takes the fusion cost in the concatenated key/value space, (24). Same 1.5B setting as Table 1, at step 45,629. The MC average is over the ten tasks listed under it, MMLU excluded.

Metric	DRC	No dissociation	No weight	Keys and values
Final 100-step train loss ↓	2.1540	2.1537	2.1634	2.1549
MC average ↑	48.54	48.57	48.63	48.71
HellaSwag ↑	58.63	58.22	58.26	58.40
WinoGrande ↑	59.19	59.98	61.01	60.85
ARC Easy ↑	59.49	60.17	58.73	60.76
ARC Challenge ↑	31.85	32.45	32.88	31.07
PIQA ↑	73.78	73.12	73.78	73.67
OpenBookQA ↑	35.80	35.80	36.20	35.20
RACE Middle ↑	48.05	48.33	47.84	48.61
RACE High ↑	36.05	35.85	36.76	36.34
CommonsenseQA ↑	20.39	19.66	18.26	19.74
BoolQ ↑	62.20	62.17	62.57	62.48
MMLU macro ↑	26.82	25.00	25.55	23.98
HumanEval+ pass@1 ↑	6.10	6.10	6.71	6.71
HumanEval+ HE-pass@1 ↑	7.32	7.93	7.93	7.93
MBPP pass@1 ↑	6.80	6.80	7.60	7.60
TriviaQA EM ↑	19.15	19.44	19.56	19.38
Natural Questions EM ↑	7.09	7.81	6.79	6.87

Table E.2: RULER at a context of 8k for the same three controls, exact match. The average is over the thirteen tasks.

RULER 8k task	DRC	No dissociation	No weight	Keys and values
NIAH single 1 ↑	100.00	100.00	100.00	100.00
NIAH single 2 ↑	100.00	96.80	98.40	94.00
NIAH single 3 ↑	83.60	56.60	46.00	42.20
NIAH multikey 1 ↑	42.60	33.00	26.20	43.40
NIAH multikey 2 ↑	0.40	2.40	0.80	1.20
NIAH multikey 3 ↑	0.00	0.00	0.00	0.00
NIAH multivalue ↑	44.95	26.30	26.20	30.65
NIAH multiquery ↑	47.20	28.75	42.60	27.35
VT ↑	0.56	0.12	7.04	3.56
CWE ↑	10.06	6.38	11.50	7.42
FWE ↑	21.65	37.83	37.32	29.97
QA 1 ↑	15.73	14.00	13.70	14.74
QA 2 ↑	18.00	18.60	18.00	21.40
Average ↑	37.29	32.37	32.90	31.99

E.1 No dissociation

The dissociation of § 3.2 is what lets the layer train at scale, and it is also the one approximation it makes: an optimizer step fuses according to a model up to 100 steps old. This control removes the approximation. It recomputes the fusions at every step, from the current weights, and pays the full serial cost for it.

The loss does not move. It is 2.1537 against 2.1540, three ten-thousandths in favor of the control, which is the scale at which two runs of the same model differ. The evaluations move as much as two runs of equal loss would, and not in one direction: 48.57 against 48.54 on the multiple-choice average, 25.00 against 26.82 on MMLU, and five points of RULER at 8k, these last two favoring the dissociated run. Nothing in the method predicts a gain from stale decisions, so we read them as noise rather than as evidence that the delay helps.

What this buys is the whole point of § 3.2: the fusions become a separate computation, run without autograd on hundreds of sequences at once, and amortized over a hundred optimizer steps.

E.2 No token weight

Fixing $w_t = 1$ removes W_w and (2). The weight of a cluster becomes the number of tokens it holds, rather than what those tokens are worth, and the $\log \hat{w}$ bias of (4) turns into a count. Everything else is unchanged: the layer still clusters, still fuses the pair of least distortion, and still reads out by attention over the centroids.

It costs 0.009 of loss, 2.1634 against 2.1540. That is more than the whole gap between DRC and GDN-2 at 32k, 0.008: without the weight, the layer falls behind the recurrent baseline it otherwise beats. Retrieval pays too, 32.90 against 37.29 on RULER at 8k. The multiple-choice average does not move, 48.63 against 48.54.

So the weight earns its parameters on the two things the state is for, the loss and retrieval, and is invisible to the multiple-choice benchmarks. It is one linear map per key/value head, and the cheapest part of the layer.

E.3 Keys and values in the fusion cost

The cost (9) involves the keys alone. The value centroids are carried along by (8) but have no say in which pair is fused. The opposite choice is defensible: the layer returns values, and two clusters whose keys agree while their values disagree are exactly the ones a fusion damages. This control makes that choice. It measures the distortion in the concatenated key/value space, which gives the fusion cost

$$\Delta_{KV}(a,b) = \frac{\hat{w}_a\,\hat{w}_b}{\hat{w}_a + \hat{w}_b}\Big(\big\|\hat{K}_a - \hat{K}_b\big\|^2 + \big\|\hat{V}_a - \hat{V}_b\big\|^2\Big), \tag{24}$$

the same Ward criterion, read on (K, V) instead of K.

It does not pay. The loss is 2.1549 against 2.1540, and retrieval drops: 31.99 against 37.29 on RULER at 8k, and 23.98 against 26.82 on MMLU. Only the multiple-choice average moves the other way, 48.71 against 48.54. Of the three controls, this is the one that costs the most retrieval.

The reason is presumably that the readout (4) selects on keys. A fusion is harmless when no future query can tell the two clusters apart, and that is a statement about keys alone, which is also how the approximation (5) is written. Spending half of the criterion on values makes the layer merge pairs that a query can still separate, and no amount of agreement between values repairs that. The extra dimensions also cost a little at generation, 1.640 ms per token for the layer against 1.545, since every distance now runs over $d_k + d_v$ coordinates.

F Positional encoding

The layer of § 2 has no notion of position. We tried three ways of giving it one, all feeding the sinusoidal encoding of Vaswani et al. (2017)

$$P_t[2i] = \sin(t/\lambda_i), \qquad P_t[2i+1] = \cos(t/\lambda_i), \qquad \lambda_i = 10000^{2i/D}, \tag{25}$$

into the projections, and differing only in how much of it reaches the key,

$$\begin{aligned} Q_t &= \mathrm{rmsnorm}\big(W_Q(\hat{X}_t + P_t)\big), \qquad V_t = \mathrm{rmsnorm}\big(W_V \hat{X}_t\big), \\ K_t &= \mathrm{rmsnorm}\big(W_K(\hat{X}_t + G_t \odot P_t)\big). \end{aligned} \tag{26}$$

The query always gets the whole encoding. The value gets none, and neither does the weight w_t. Since the keys are what the fusion sees, G_t decides how much of the clustering is spent on position rather than on content. The three variants differ by G_t:

DRC-SPE, for static positional encoding, takes $G_t = 1$. The key is encoded like the query, and there is nothing to learn.

Table F.1: Loss and evaluations of the three positional variants, next to the plain layer, on the same 1.5B setting as Table 1, at step 45,629.

Metric	DRC	DRC-SPE	DRC-GPE	DRC-MPE
Final 100-step train loss ↓	2.1540	2.1532	2.1515	2.1522
MC average ↑	48.54	49.08	48.64	48.46
HellaSwag ↑	58.63	58.60	58.21	58.19
WinoGrande ↑	59.19	60.54	59.51	60.06
ARC Easy ↑	59.49	61.56	58.35	59.70
ARC Challenge ↑	31.85	33.30	32.53	32.96
PIQA ↑	73.78	73.34	73.50	73.78
OpenBookQA ↑	35.80	36.60	38.00	33.60
RACE Middle ↑	48.05	48.05	47.35	48.19
RACE High ↑	36.05	35.93	36.13	35.59
CommonsenseQA ↑	20.39	20.56	20.07	20.15
BoolQ ↑	62.20	62.35	62.72	62.42
MMLU macro ↑	26.82	25.45	25.65	26.23
HumanEval+ pass@1 ↑	6.10	6.10	6.10	5.49
HumanEval+ HE-pass@1 ↑	7.32	9.15	8.54	6.10
MBPP pass@1 ↑	6.80	8.00	10.00	8.60
TriviaQA EM ↑	19.15	19.14	19.20	19.08
Natural Questions EM ↑	7.09	7.40	7.87	7.04

The other two make G_t depend on the token, through one scalar per token and per key/value head

$$a_t = \sigma\big(W_a X_t + b_a\big) \in (0, 1), \tag{27}$$

which adds a fifth linear map, W_a. It starts nearly closed, at $W_a = 0$ and $b_a = \mathrm{logit}(10^{-3})$, so the keys carry almost no position at initialization, and training has to find where it pays.

DRC-GPE, for gated positional encoding, scales the encoding as a whole, $G_t = a_t$. The gate is an amplitude: the key gets all of the frequencies or none of them.

DRC-MPE, for modulated positional encoding, makes the gate the cutoff of a *low-pass filter*. It keeps the coarse frequencies and suppresses the fine ones, so that $a_t = 0$ retains nothing, $a_t = 1$ retains everything, and an intermediate a_t retains the scales down to $\lambda_{\min} = C^{1-a_t}$, with $C = 8192$ the maximum context. A hard cutoff would keep the pair i when $a_t \geq \tau_i$, with

$$\tau_i = \mathrm{clip}\Big(1 - \frac{\log_2 \lambda_i}{\log_2 C},\ 0,\ 1\Big), \tag{28}$$

and G_t replaces it by a cubic smoothstep over half an octave,

$$G_t[2i] = G_t[2i+1] = u_{t,i}^2(3 - 2u_{t,i}), \quad u_{t,i} = \mathrm{clip}\Big(\frac{a_t - s_i}{\delta},\ 0,\ 1\Big), \quad \delta = \frac{0.5}{\log_2 C}, \tag{29}$$

where s_i centers the transition on τ_i, and the sine and the cosine of a pair get the same value. The implementation evaluates K_t as $W_K X_t + W_K(G_t \odot P_t)$, which avoids materializing one positional tensor per key/value head.

Nothing else changes. The fusions of § 2.2 and the replay both use the position-aware keys. The gradient reaches W_a through the keys, the centroids and the attention, and none passes through the discrete choices.

What the encoding does is move tasks around. The static one takes VT from 0.56 to 32.36 and FWE from 21.65 to 35.80, the two tasks that need the order of the tokens, and pays for it on NIAH single 3, on NIAH multivalue and multiquery, and on MMLU. On the evaluations the three sit within a point of the plain layer on the MC average, and below it on MMLU.

The gates do not earn their keep. On RULER at 8k the static encoding is the one that comes out ahead, 39.12 against 37.29 for the plain layer. Learning how much position to keep takes part of that back, 37.20 for the amplitude, and DRC-MPE ends four points below the plain layer. At 4k, where the plain layer is already at 42.45, none of the three gains.

Table F.2: RULER at a context of 8k for the three positional variants, next to the plain layer. At 4k the averages are 42.45, 41.13, 39.29 and 37.77, against 31.47 for GDN-2.

RULER 8k task	**DRC**	**DRC-SPE**	**DRC-GPE**	**DRC-MPE**
NIAH single 1	100.00	100.00	100.00	100.00
NIAH single 2	100.00	99.80	100.00	99.80
NIAH single 3	83.60	72.40	68.40	56.80
NIAH multikey 1	42.60	48.40	42.60	37.60
NIAH multikey 2	0.40	5.00	3.60	1.20
NIAH multikey 3	0.00	0.00	0.00	0.00
NIAH multivalue	44.95	31.35	40.25	41.55
NIAH multiquery	47.20	39.45	49.00	44.20
VT	0.56	32.36	12.52	7.40
CWE	10.06	8.74	10.68	6.88
FWE	21.65	35.80	26.07	10.89
QA 1	15.73	13.87	12.86	9.13
QA 2	18.00	21.40	17.60	14.80
Average	37.29	39.12	37.20	33.10

Table G.1: MNIST, one seed, on the full training and test sets. The 0.11 point of test accuracy is within the run-to-run spread of such a model, and we do not read it as a gain.

After 10 epochs	**Vanilla**	**Dissociated**
Train loss ↓	0.0097	0.0087
Train accuracy ↑	99.71	99.70
Test loss ↓	0.0434	0.0442
Test accuracy ↑	99.02	99.13

G Dissociation on a ConvNet

Nothing in § 3.2 is specific to clustering. A ReLU chooses which coordinates pass, a max-pooling chooses which input of each cell passes, and once those choices are made both are linear: $\mathrm{ReLU}(z) = m \odot z$ and $\mathrm{maxpool}(z) = \mathrm{gather}(z, i)$. So the same trick applies, and we can check it where every decision is visible.

To check whether the lag between the decisions and the forward/backward that uses them may hurt, we applied that recipe to a small ConvNet on MNIST, composed of two convolutions and two linear layers, 105,866 parameters, with no normalization and no dropout.

It has five families of decisions: three ReLU masks and two pooling argmaxes. Training runs for 10 epochs of Adam at 10^{-3}, on minibatches of 10. The dissociated model collects the decisions of 1,000 examples in one forward without autograd, then does the 100 optimizer steps of that block with them held fixed. A decision is thus between 0 and 99 steps old, 49.5 on average. The two models start from the same weights and see the same examples in the same order, on the same minibatch boundaries.

The replay is exact. With decisions collected from the very parameters that replay them, the logits match the ordinary network to 0, and the parameter gradients to 3.3×10^{-7}. So what the experiment measures is staleness, not an implementation gap.

Table G.1 gives the result: the two models are on top of each other. This is despite a real drift. Recollecting the decisions of the same 1,000 examples after the 100 steps, and comparing them with those used, 2.0% of the first ReLU mask has changed, and 5.7% of the last one, averaged over the 600 blocks of the run. The deeper the decision, the more it moves, and the drift decreases as training advances. Stale choices remain usable well past the point where they are all correct.

Two limits are worth stating. This is one seed, so the accuracies compare but do not separate. And we tried only one block length, 100 steps; nothing here says where staleness starts to hurt.