DELTA-RULE MEMORIES STORE MORE THAN THEY RECALL, AND A LOAD CURRICULUM CLOSES THE GAP

Anonymous authors
Paper under double-blind review

ABSTRACT

Delta-rule layers are now common in open frontier models, where instead of a growing KV cache (as in attention), they keep a fixed $d_k \times d_k$ matrix per head and use it as an associative memory. This rank d_k matrix can return at most d_k exact associations, but a language model only needs the correct token to win the argmax, and recent theory shows that an offline memory then returns the right token for n associations whenever $d_k^2 > 2n \ln n$. We ask how many associations these models recall in practice, in two-layer, single-head DeltaNet models trained on multi-query associative recall. Using state reads and edits built from the model's own primitives, we identify a circuit in which the first layer carries each key token to its paired value and the second layer stores the associations. We then distinguish failures of writing from failures of retrieval. In width-128 models trained on 64 pairs, recall falls to 16% at a load of $4d_k$. Reading the same state with each association's write key recovers 50%, and a matrix built by ridge regression from the same keys and values raises this to 83%, so both online writing and retrieval limit recall, and many associations remain accessible after the model fails to recall them. Finally, a training curriculum that progressively increases the number of pairs (the *load*) raises the load at which the model reaches 90% recall from $1.7d_k$ to $3.6d_k$ at $d_k = 128$, and reaches it faster than training directly at the target load. Effective recall capacity therefore depends on the learned writing and retrieval procedures and can be increased through training without enlarging the memory state.

1 INTRODUCTION

Linear attention layers are increasingly adopted in open frontier models. Qwen3-Next, Qwen3.5 and Qwen3.6 (Qwen Team, 2025; 2026a;b), Kimi Linear (Kimi Team, 2025), Kimi K3 (Kimi Team, 2026) and Olmo Hybrid (Merrill et al., 2026) all interleave three gated delta-rule layers with one attention layer, because a linear attention layer keeps a fixed-size state per head in place of a KV cache that grows with context length.

Where attention keeps a complete record of past tokens, linear attention folds the whole history into one $d_k \times d_k$ matrix per head, where d_k is the key dimension. That matrix is an associative memory with interference (Section 2). Because it has rank d_k, it can hold at most d_k exact associations. The rank bound, however, only limits exact reconstruction. A language model only needs the right token to win the argmax over the vocabulary, and recent work (Barnfield et al., 2026; Giorlandino et al., 2026) shows that a $d \times d$ linear memory constructed offline with random keys returns the right target for all n pairs as long as $d^2 > 2n \ln n$, which at width 256 is close to four thousand pairs. A trained model builds its state online and will not reach this bound in practice.

Therefore, the question that we aim to answer is: how many associations can a trained delta-rule matrix hold in practice, and can that number be influenced by training?

We call the number of pairs per sequence the *load*. A model's *effective capacity* is the largest load at which its recall on fresh sequences is at least 0.9. Figure 1 shows the result. At each width, the same $d_k \times d_k$ matrix recalls 1.2 to $1.7d_k$ pairs on a naive training recipe, but improves to 2.2 to $6.2d_k$ pairs on a curriculum-aware training recipe.

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.

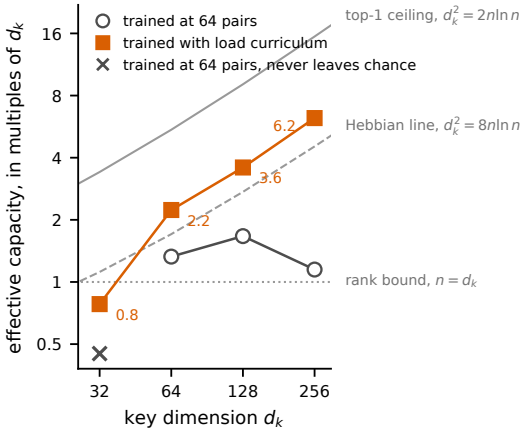


Figure 1: Effective capacity in multiples of d_k. Circles: models trained on 64 pairs, evaluated at 0.9 recall on fresh sequences. Squares: curriculum models, at the last load passed with 0.9 moving-average recall. Gray lines: theoretical references (Section 2). The cross marks chance-level recall. Seed counts and evaluation details are in Appendix A.

Recall can fail because the pair was never written, because it was written and then drowned out by later updates, or because the query cannot retrieve it. To tell these apart, we read and edit the model state.

We study the simplest setting of a two-layer, single-head DeltaNet model, trained on multi-query associative recall (MQAR, Arora et al. 2024) tasks from widths 32 to 256. In these models, the first layer state carries each key token forward to the value token that follows it, and the second layer writes out the pair into its state. This is the two-stage gather-and-aggregate circuit that Bick et al. (2025) found at head resolution in state-space models and transformers, and the induction-style solution that Arora et al. (2025) found at block resolution in attention and Based, whereas their DeltaNet retrieved in a single layer.

Our contributions are as follows.

- **Where the capacity lives** (Section 4). Associations are stored in the second layer's state matrix: zeroing it after the pairs are written removes all recall, and erasing one pair's key from it removes that pair's recall and no other. The first layer is a one-slot register that carries the key token forward for one step, and stores no association. The capacity we measure is therefore that of a single $d_k \times d_k$ matrix.
- **What recall is made of** (Section 5). Recall requires the key token to be written under a stable key (*binding*), the pair to survive in the matrix (*storage*), and the query to land on that key (*retrieval*). Past d_k both retrieval and storage fail, opening at the same load, and storage degrades last: the state keeps pairs the model's own query cannot reach, and a batch optimum on the same keys and values keeps more still.
- **Training can influence effective capacity** (Section 6). A curriculum that raises the load by ten percent whenever recall reaches 0.9 trains a width-128 model to 0.9 recall at $3.6d_k$, where training at that load from the start stays at chance for 16,000 steps and reaches 0.61 after 32,000. At width 256 the curriculum reaches $6.2d_k$, above the Hebbian line and at 0.4 of the top-1 ceiling.

We leave open the question of whether the results translate to hybrid attention architectures, where attention layers can aid in recall.

2 Background

Linear attention as an associative memory. In linear attention (Katharopoulos et al., 2020), each head keeps a matrix state $S_t \in \mathbb{R}^{d_k \times d_k}$, updated at every token from a key k_t, a value v_t and a query q_t computed from the residual stream. The simplest update adds the outer product of value and key, and the output is the state applied to the query (we drop the feature map and normalizer of the original, as the delta-rule models we study do):

$$S_t = S_{t-1} + v_t k_t^\top, \qquad o_t = S_t q_t. \tag{1}$$

We call this the *additive* rule. After n pairs with unit keys, a read with a past key k_i returns

$$Sk_i = \underbrace{v_i}_{\text{exact term}} + \underbrace{\sum_{j \neq i} (k_j^\top k_i)\, v_j}_{\text{interference}}. \tag{2}$$

The interference term vanishes only when the keys are orthogonal, and a $d_k \times d_k$ matrix has d_k orthogonal directions, so at most d_k pairs can be read back exactly. The additive rule also has no way to update a key: a new value under a key already present adds to the old value.

The delta rule. The delta rule (Schlag et al., 2021; Yang et al., 2024) removes what the matrix already returns along the new key before it writes. With a token-dependent write gate $\beta_t \in [0, 1]$,

$$S_t = S_{t-1}(I - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top, \qquad o_t = S_t q_t. \tag{3}$$

At $\beta_t = 1$ the first term removes $(S_{t-1}k_t)k_t^\top$, which is what a read with k_t would have returned, and the second term writes $v_t k_t^\top$ in its place, so a read with k_t immediately after the write returns v_t with no interference. At β_t between zero and one the update moves the stored value part of the way toward v_t. The price is that every later write with key k_t erodes the component of every earlier pair along k_t, which is how the state forgets. Gated variants (Yang et al., 2025) add a scalar decay in front of S_{t-1}, which forgets every pair at once rather than along one direction. We do not study gated variants as our task does not require forgetting.

Theoretical results on linear memory. In a language model, we only require the interference term in Equation 2 to be small enough that the right token still wins the argmax over the vocabulary. Barnfield et al. (2026) prove, and Giorlandino et al. (2026) independently derive, that one can perform an offline construction of a $d \times d$ linear memory with random Gaussian keys that returns the right target for all n pairs as long as $d^2 > 2n \ln n$. We call this limit the *top-1 ceiling*. Barnfield et al. (2026) also analyze the plain outer-product memory of Equation 1, the sum of $kv^\top$ over all pairs. They show that perfect recall holds under the stricter condition $d^2 > 8n \ln n$, called the *Hebbian line*.

3 Experimental Setup

Models and task. The model that we study in Sections 4 to 6 is a DeltaNet (Yang et al., 2024) reduced to two pre-norm layers with one head each, tied embeddings, and no MLP, position embedding or short convolution. The width d is the head's key and value dimension, which means each layer holds one $d_k \times d_k$ state with $d_k = d$, and we use widths 32, 64, 128 and 256. Keys are unit vectors.

The task throughout is multi-query associative recall (Arora et al., 2024): the model sees n key-value token pairs and is then queried, with filler tokens in between, for the value paired with each key. *Recall* is the fraction of queries answered correctly. The vocabulary has 8192 tokens. Unless we say otherwise, a model was trained on 64 pairs per sequence with 100,000 fresh sequences per epoch for 64 epochs at batch 512.

Reading memory state. All measurements read the second layer's state after the last pair has been written. We read the state in three different ways to test for storage versus retrieval:

- *Recall* reads with the model's own query at each query position, so it is the answer that the model would have given without any intervention.
- The *own-key read* reads using the key the pair was actually written under, taken from the same forward pass at the pair's value token (Figure 2). Before the read, we project the layer's shared direction (defined below) out of the key, since every key overlaps that direction and reading with it would return every other pair's value as crosstalk. If the pair is in the state, this read usually finds it whether or not the model's query would, so it is a lower bound on what the state holds. Appendix B gives the controls and the models on which the read is not valid.
- The *ridge read* builds the least-squares memory of the keys and values for all pairs, $S^\star = VK^\top(KK^\top + \lambda I)^{-1}$, which is the best a linear memory of these keys and values could

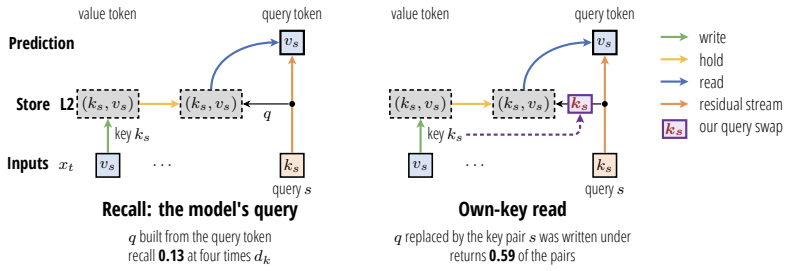


Figure 2: Two reads of the same state. **Left:** recall reads the second layer's state with the query the model builds from the query token. **Right:** the own-key read replaces that query with the key the pair was written under, taken at the pair's value token in the same forward pass. Numbers are from width-128 seed 0 at four times d_k.

do. It has the shape of the state, so we put it in the state's place after the last pair and read it exactly as the own-key read reads the real state: with each pair's own key, through the model's output path. The best λ over a small grid is reported.

These three reads define two gaps. The *retrieval gap*, own-key read minus recall, counts the pairs that are in the state but are not found by the model's query. The *storage gap*, ridge read minus own-key read, counts the pairs that the online delta-rule writes lost and a batch optimum over the same keys and values would have retained. All three reads are scored through the model's output projection and an argmax over the vocabulary, so a gap is a difference between states or keys, not between scorers. Appendix B reports the ridge read under a second scorer, the nearest stored value, which needs no trained output path.

For the load sweep in Section 5, we take each model trained on 64 pairs, draw 256 fresh sequences at each load from $d_k/4$ to $4d_k$, run the model, and read the second layer's state after the last pair. We use sixteen seeds at width 128 and four at width 64 (Figure 5, Table 1 in Appendix B). Width 256 is in Appendix B, because at that width the own-key read fails a control.

A fourth measurement is for *binding*, the map from a key token to the key it is written under. A pair is bound well if that key is a function of the key token alone, which we measure as the cosine between the write keys of the same token in different sequences, or for one pair, between its write key and the mean write key of its token in other sequences.

Edits. We modify a state with one of three edits at a chosen position, and the model then runs on without further change (Figure 8, Appendix C).

- *Erase.* For a unit direction u, set $S \leftarrow S(I - uu^\top)$. This is one delta-rule step with key u, gate one and value zero.
- *Write.* Set $S \leftarrow S + vk^\top$. This is one additive step. Erasing k and then writing (k, v) stores v under k.
- *Zero.* Set $S \leftarrow 0$.

The harness recomputes the head's projections, runs the recurrence in fp32, and applies the edit after the write and readout at the chosen position. With no edit applied, it reproduces the model's output.

Two directions recur throughout the paper, so we define them here. A layer's *shared direction* u_1 is the first principal direction of its unit keys at key-token positions. In the first layer, nearly every key token writes under nearly the same key, so this direction is the slot into which the layer writes, and we refer to it as the *register direction*.

What an erase predicts on its own. Within one head, erasing u changes the output by $\Delta o = -(Su)(u^\top q)$ for a fixed state and query. This identity predicts the head's output change to rounding error in every experiment below, but not recall, since the output then passes through normalization, the output projection and the next layer, and every later write operates on the edited state. An erase result is therefore informative only relative to controls, so each experiment also erases a random

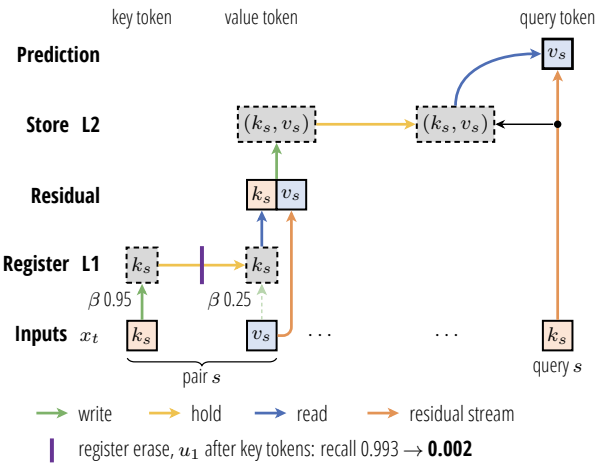


Figure 3: The two-stage circuit for one pair s. At the key token the first layer writes the token onto its register direction u_1 (write gate 0.95) and holds it through the value token (gate 0.25), where its readout passes the key token to the residual stream. The second layer writes (k_s, v_s) at the value token and holds it until query s reads it. Purple marks the register erase. Values are from width-128 seed 0.

direction of the same norm, the same direction in the other layer, and the same direction at a different position.

4 Every association lives in one $d_k \times d_k$ matrix

The two layers of the model specialize: the first acts as a previous-token register and the second as the memory (Figure 3). Figure 4 presents the evidence for each part.

The first layer holds one token and no associations. If the first layer is a previous-token register, then erasing the direction it writes to, at the step where that direction is needed, should destroy recall, whereas erasing any other direction should not. Erasing the register direction from the first layer's state after every key token, before the value token reads it, reduces recall from at least 0.91 to at most 0.016 on all 36 models that learned the task (Figure 4, left). Three controls each leave recall above 0.84: a random unit direction erased from the same head at the same positions, the register erase applied to the second layer, and the register erase applied after every value token, once the value token has read it. Recall therefore falls only when the register direction is erased, in the first layer, between a key token's write and the next token's read. One direction, held for one step, is the behavior of a previous-token register.

If the first layer only passes each key token forward, then its contents should not matter once every pair has been written. We therefore zero the first layer's state after the last pair and run the queries with an empty register. Up to d_k pairs this has no cost, on a width-128 model trained on 64 pairs and on a width-128 model that the curriculum of Section 6 trained to $3.6d_k$ (Figure 4, right). Beyond d_k the register's residual content does affect retrieval, in opposite directions for the two models, and we have not investigated why.

The second layer holds every association, one pair under one key. Zeroing the second layer's state after the last pair gives recall 0.000 at every load, on the model trained on 64 pairs and on every curriculum model. No other part of the model can therefore answer a query.

Within that state, each pair is stored under its own key, which we verify by erasing one key at a time (Figure 9, left, Appendix C). After all 64 pairs are written, we erase the key of pair s, namely the key the layer computed at that pair's value token, and record the recall of every pair t (Figure 4, middle). On a width-128 model this costs pair s between 0.52 and 1.00 of its recall, 0.70 on average, whereas every other pair loses at most 0.02. We hypothesize that the surviving part of pair s is due to the query landing partly on directions other than the write key, but we have not tested this.

The same key also allows us to replace the pair's value (Figure 9, right). Erasing the key of pair s and writing a donor value from another pair under it causes the model to answer query s with the donor on 0.99 to 1.00 of queries across 35 models, while the other pairs change by 0.016 on average. Writing under a random key instead changes the answer on at most 0.16 of queries (Appendix C).

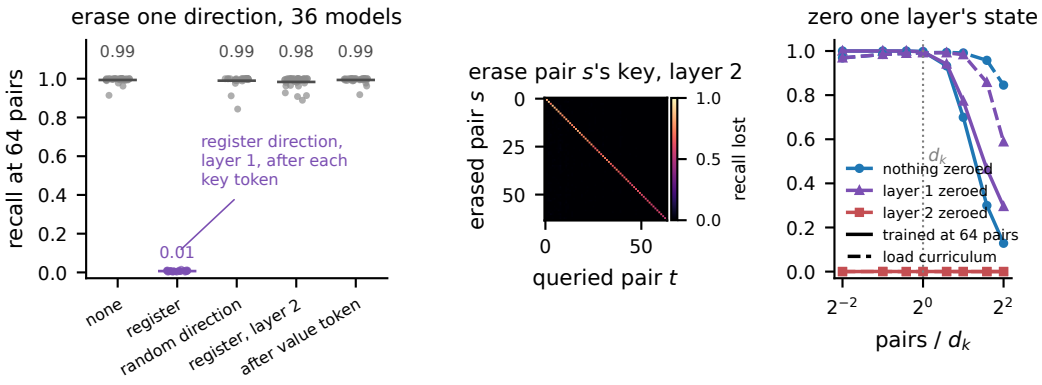


Figure 4: Where the associations are. **Left:** recall at 64 pairs after erasing one direction, one point per model, with the mean as a bar. The register erase is applied after every key token, the random direction and the layer-2 erase at the same positions, and the last column erases the register after every value token. **Middle:** on a width-128 model, the recall lost by pair t after erasing pair s's key from layer 2 once all pairs are written. **Right:** recall against load after zeroing one layer's state after the last pair, for a model trained on 64 pairs (solid) and a curriculum model trained to $3.6d_k$ (dashed), both width 128.

The memory is therefore the single $d_k \times d_k$ matrix that Barnfield et al. (2026) analyze. Past d_k, the failure must lie either in the key under which a pair is written (the register and the layer-2 key projection) or in the query with which it is read back. Section 5 measures both. The same erase and write also localize a word recalled from ordinary text to one head of a public 1.3B Gated DeltaNet model (Appendix E).

5 Writing and retrieval both limit recall

Models trained on 64 pairs lose recall as the number of pairs increases beyond d_k (Figure 1). We ask whether this decline reflects information lost from the state or a failure to retrieve information that remains stored. To distinguish these possibilities, we first change how the state is read, then how it is constructed, using the reads defined in Section 3.

The state retains associations that the model fails to retrieve. We read each association using the key under which it was written, replacing the model's query while keeping its state. If this own-key read recovers more answers, then failed recall cannot be explained entirely by information lost during writing. At width 128 and a load of $4d_k$, the own-key read returns 0.50 of the pairs, whereas the model recalls only 0.16 (Figure 5). Thus, many associations remain accessible in the state even when the model fails to retrieve them.

The writing procedure also limits recall. The own-key read still fails on half the pairs. We therefore ask whether a different matrix of the same size could return more of them. We construct a state by ridge regression using the same keys and values, then read it with the same own-key readout. Accuracy rises from 0.50 to 0.83 (Table 1, Appendix B). The matrix's size and the extracted representations therefore permit higher recall than the model's online writing procedure achieves under this readout.

However, replacing the state alone does little for the model's own query: recall rises only from 0.16 to 0.17. The learned query fails to access much of the information available in either state. These results identify limitations in both writing and retrieval, and show why improving the state need not improve recall unless the reader can use it.

Both limitations emerge as the load increases. At $n = d_k$, all three reads are nearly perfect. Beyond this load, recall and the own-key read decline, while the ridge read remains higher (Table 1). At $4d_k$, changing the read gives a gain of 0.34, and additionally replacing the state gives a further

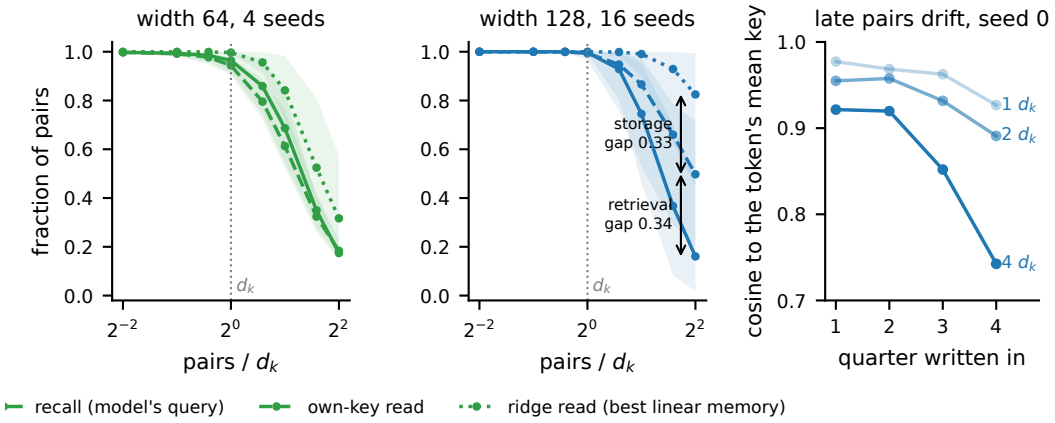


Figure 5: Recall, own-key read and ridge read against load. **Left and middle:** models trained on 64 pairs. Recall with the model's query (solid), the own-key read of the same state (dashed) and the ridge read of the same keys and values (dotted), mean over seeds with the range shaded. The two gaps marked at four times d_k are the retrieval gap and the storage gap. **Right:** binding on width-128 seed 0, the cosine between a pair's write key and the mean write key of its token in other sequences, by the quarter of the sequence in which the pair was written.

gain of 0.33. Width 64 shows the same ordering with smaller gaps. The results therefore do not identify a load at which retrieval alone fails while storage remains intact.

To investigate the writing limitation, we ask whether the learned keys interfere with one another less than random keys would. Under the delta rule, each later write with key k_t removes the component along k_t from every stored key, so the fraction of a pair's key direction that remains after all later writes, which we call *key survival*, measures how much later writes have eroded it. We compute it with the model's learned keys and, with the same write gates, with random unit keys. Averaged over the sixteen width-128 seeds, the two differ by at most 0.01 at the loads in Table 1. By this measure, the learned keys provide no greater protection against subsequent writes than random keys.

We next investigate the retrieval limitation: why does the model's query fail to recover associations that remain accessible through their write keys?

Binding deteriorates with load. The own-key read uses the key under which an association was actually written. The model's query, by contrast, is computed from the key token alone, so it can reach the association only if the write key was itself a function of the key token. This is the property defined as binding in Section 3. We measure it as the cosine between the write keys of the same token in different sequences. On a width-128 model this cosine is 0.98 at a quarter of d_k and 0.74 at four times d_k. The decline is concentrated on pairs written late in the sequence (Figure 5, right). We interpret this as later pairs being written while the register still carries traces of earlier tokens, so that those traces enter the write key.

A better reader alone does not close the retrieval gap. Reading with the best key the model could build from the key token, the mean write key of that token across sequences, returns 0.26 of the pairs at four times d_k on width-128 seed 0, against 0.59 for the write key itself. Retraining the query projection with all other parameters frozen recovers 0.04 of a 0.19 gap at twice d_k (Appendix B).

How much a matrix could hold. Solving the two thresholds of Section 2 for n at width 128 gives a Hebbian line of 350 pairs and a top-1 ceiling of 1161, against a rank bound of 128 (all widths in Table 3). The lines are reference points rather than predictions, since the theory assumes Gaussian keys, an all-pairs criterion and an offline optimum (Section 8). A model trained on 64 pairs reaches recall 0.9 at 1.2 to $1.7d_k$, below the Hebbian line at every width, whereas the width-256 model trained on 1024 pairs reaches 0.986 at $4d_k$, just below its line. The matrix can therefore hold pairs past d_k, and what the model trained on 64 pairs lacks is binding and retrieval trained past that load.

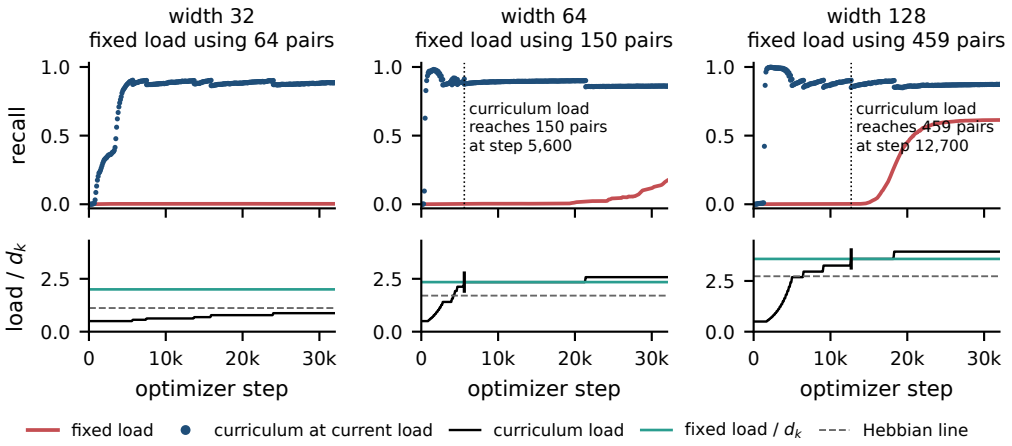


Figure 6: Curriculum and fixed-load training by width, through 32,000 steps. **Top:** recall at the current training load for the curriculum (points) and for the fixed-load control (red). **Bottom:** curriculum load in units of d_k, the control's load, and the Hebbian line. Ticks mark the step at which the curriculum's load first reaches the control's load.

6 A LOAD CURRICULUM INCREASES EFFECTIVE CAPACITY

Higher training loads increase recall capacity, but training from scratch at high loads can remain at chance for thousands of steps. We therefore test whether gradually increasing the load allows the same architecture to learn higher-capacity recall.

Curriculum. We start at $d_k/2$ pairs and increase the load by ten percent whenever moving-average recall exceeds 0.9, with three quarters of the training sequences at the current load and the remainder at lower loads as replay. We test widths 32, 64, 128 and 256, with two seeds at widths 64 and 128 and one at the other widths. Appendix A gives the training procedure.

Achieved capacity. The curriculum reaches 2.1–$2.3d_k$ pairs at width 64, $3.6d_k$ at width 128 and $6.2d_k$ at width 256. These loads exceed the Hebbian reference line and reach approximately 0.4 of the top-1 ceiling, subject to the comparison limits in Section 5. Width 32 reaches only 25 pairs, below d_k. The final curriculum checkpoints at widths 64 to 256 recall 0.85 to 0.99 of the pairs at four times d_k, where the models trained on 64 pairs recall 0.17 or less (Table 3, Appendix D). These are achieved loads within the training budgets, not established capacity limits: recall was still rising slowly when runs stopped below the promotion threshold.

Recall at the lowest replayed load remains at 0.98–1.00 across the curriculum runs, whereas the width-256 model trained only on 1024 pairs recalls 0.75 at 64 pairs.

Fixed-load comparison. For seed 0, we train controls with the same architecture, optimizer and batch size for 32,000 steps, holding the load fixed at 150 pairs for width 64 and 459 for width 128. At width 64, the curriculum reaches recall 0.90 at 150 pairs at step 21,400, whereas the control remains at chance until 20,000 steps and reaches 0.17 at 32,000. At width 128, the curriculum reaches 0.90 at 459 pairs at step 18,300, whereas the control leaves chance at 16,000 steps and reaches 0.61 at 32,000 (Figure 6). Fixed-load training can therefore learn given enough steps, and the step at which it leaves the plateau varies widely between runs that differ only in hardware (Appendix A). The curriculum improves training speed and achieved recall in these comparisons, without establishing that it is necessary for learning. There is no fixed-load control at width 256.

Different internal solutions. The two width-128 seeds reach the same load through different mechanisms. Seed 1 and both width-64 seeds use soft write gates and keys with little energy in a shared direction, whereas width-128 seed 0 keeps a gate of 1 and a shared key direction while recall improves (Appendix D). Its reader uses the component the own-key read removes, so that diagnostic is invalid for this model (Appendix B). Why the curriculum helps, and where it stalls, remain open questions.

7 Related work

Recall in linear attention and the delta rule. The recurrent matrix has been read as an associative memory since linear attention and the fast-weight programmers, where the delta rule was introduced so that a write overwrites rather than accumulates (Katharopoulos et al., 2020; Schlag et al., 2021), and DeltaNet made it practical at scale (Yang et al., 2024; 2025). Test-time regression interprets the rule as an online approximation to the least-squares memory that our ridge read computes exactly (Wang et al., 2025b). MQAR measures these models' recall against their state size (Arora et al., 2024), with the rank of the state as the capacity measure. We measure how far a trained model departs from that rank, and in which direction.

Capacity of linear associative memories. Barnfield et al. (2026) prove that a $d \times d$ linear memory with Gaussian keys returns the correct target for all n pairs if and only if $d^2/(n \ln n) > 2$ asymptotically, and that the plain outer-product memory succeeds once the same ratio exceeds 8, a sufficient condition only. Giorlandino et al. (2026) reach the same threshold by a statistical-physics route. Both describe an offline optimum, whereas we measure what a model writing online reaches after training.

Mechanisms of recall in recurrent models. Arora et al. (2025) apply interchange interventions at block resolution across seven architectures and find that attention and Based solve associative recall by a two-stage induction circuit, whereas Mamba and DeltaNet retrieve in a single layer, with Mamba forming the association in its short convolution. Bick et al. (2025) find two-stage gather-and-aggregate heads by zeroing heads. Our models have no short convolution and learn the two-stage circuit, which we localize to single directions of a head's state. Previous-token information is carried by induction circuits in transformers (Olsson et al., 2022) and by the shift or convolution in H3 and Mamba (Fu et al., 2023; Wang et al., 2025a). Our models have neither, so the first layer's register carries it in the recurrent state.

Editing a model's memory. Activation patching and its relatives test which internal variables affect the output (Vig et al., 2020; Geiger et al., 2021; Zhang & Nanda, 2024; Heimersheim & Nanda, 2024), rank-one editing modifies weights (Meng et al., 2022; 2023; Sen Sharma et al., 2024), and directional erasure removes a direction from a representation (Ravfogel et al., 2020; Belrose et al., 2023; Arditi et al., 2024). Our unit of intervention is a key direction in the running matrix state, which has a closed-form local effect because the update is linear. The nearest operation is the whole-state swap in hybrid models of Afendulev et al. (2026). Hase et al. (2023) found that where a fact appears to be stored in a transformer does not predict where editing it succeeds, whereas in the delta-rule state the direction where a pair is stored is also where erasing and writing it succeed. Hatamizadeh et al. (2026) use the terms erase and write for components of a learned update rather than for inference-time edits.

8 Limitations and conclusion

The results are on two-layer models with one head per layer, on one synthetic task, and Appendix E only begins to ask how much carries to a multi-head layer inside a language model. The curriculum has at most two seeds per width, and the widths differ in learning rate, so the comparison across widths carries that confound. Two seeds establish that there is more than one internal solution, not how often each occurs, and we have no account of where the curriculum stalls. The 1024-pair width-256 run had four times the step budget of the 384-pair and 512-pair runs, so that plateau comparison is confounded by batch size. The theory lines assume Gaussian keys, an all-pairs criterion and an offline optimum, none of which a trained model writing online satisfies. The own-key read is not valid on models whose reader uses the shared key direction (Section 6). Within these limits the picture is consistent: recall past d_k tracks how far binding and retrieval were trained rather than how much the matrix can hold, and the curriculum trains them further than a fixed load does.

AI use statement

AI assistants, large language models run as coding agents, were used throughout this project, and the author is responsible for every claim in the paper. Their use covered the following items of the ICLR policy.

Research content. The assistants helped develop the conceptual framework, including the separation of storage from retrieval and the three reads that measure it. They proposed and refined hypotheses, wrote predictions before confirmatory runs, proposed experiments and controls including the own-key read and the ridge read, suggested some experimental parameters, and drafted interpretations of results. They first stated some of the mathematical identities used, namely the closed form for an erase, the key-survival recurrence, and the threshold values solved from the bounds of Barnfield et al. (2026). They wrote the MQAR generator and the curriculum's data schedule, so the synthetic training and test data were produced by AI-written code.

Implementation. The assistants wrote the training scripts, the intervention harness, the sweep and analysis scripts, the figures and the schematics. The author wrote the edit operations, the accuracy metric and the reference recurrences for the update rules, and checked the AI-written code against them.

Writing. The assistants drafted full and partial drafts of this paper, suggested its structure and title, edited for readability, identified and summarized related work, and formatted the references. The author edited the drafts, verified every citation against the cited paper, and verified every number against the result file behind it.

The research question was the author's. The assistants did not write proofs, translate, clean datasets, or perform qualitative analysis.

REFERENCES

Kirill Afendulev, Alexey Dontsov, Elena Tutubalina, and Anton Korznikov. What attention recalls and recurrence controls in hybrid language models. *arXiv preprint arXiv:2609.04434*, 2026. To appear in Findings of EMNLP 2026.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

Aryaman Arora, Neil Rathi, Nikil Roashan Selvam, Róbert Csordás, Dan Jurafsky, and Christopher Potts. Mechanistic evaluation of Transformers and state space models. *arXiv preprint arXiv:2505.15105*, 2025.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Nicholas Barnfield, Juno Kim, Eshaan Nichani, Jason D. Lee, and Yue M. Lu. Sharp capacity thresholds in linear associative memory: From top-1 retrieval to tail-average learning. *arXiv preprint arXiv:2605.05189*, 2026.

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. LEACE: Perfect linear concept erasure in closed form. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Aviv Bick, Eric Xing, and Albert Gu. Understanding the skill gap in recurrent language models: The role of the gather-and-aggregate mechanism. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4324–4344, 2025.

Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. MoM: Linear sequence modeling with mixture-of-memories. *arXiv preprint arXiv:2502.13685*, 2025.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards language modeling with state space models. In *International Conference on Learning Representations (ICLR)*, 2023.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021.

Alessio Giorlandino, Sebastian Goldt, and Antoine Maillard. Factual recall in linear associative memories: sharp asymptotics and mechanistic insights. *arXiv preprint arXiv:2605.10795*, 2026.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. Does localization inform editing? Surprising differences in causality-based localization vs. knowledge editing in language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

Stefan Heimersheim and Neel Nanda. How to use and interpret activation patching. *arXiv preprint arXiv:2404.15255*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

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

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations (ICLR)*, 2023.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Chuan Li, Kyle Lo, Saumya Malik, DJ Matusz, Benjamin Minixhofer, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A. Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. Olmo Hybrid: From theory to practice and back. *arXiv preprint arXiv:2604.03444*, 2026.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. Blog post and model card, `https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct`, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. Blog post, `https://qwen.ai/blog?id=qwen3.5`, 2026a.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all. Blog post, `https://qwen.ai/blog?id=qwen3.6-35b-a3b`, 2026b.

Shauli Ravfogel, Yanai Elazar, Hila Gonen, Michael Twiton, and Yoav Goldberg. Null it out: Guarding protected attributes by iterative nullspace projection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Arnab Sen Sharma, David Atkinson, and David Bau. Locating and editing factual associations in Mamba. In *First Conference on Language Modeling (COLM)*, 2024.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Junxuan Wang, Xuyang Ge, Wentao Shu, Qiong Tang, Yunhua Zhou, Zhengfu He, and Xipeng Qiu. Towards universality: Studying mechanistic similarity across language model architectures. In *International Conference on Learning Representations (ICLR)*, 2025a.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025b.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations (ICLR)*, 2025.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations (ICLR)*, 2024.

A TRAINING DETAILS

All models are the two-layer, one-head DeltaNet of Section 3. The details not given there are as follows. The residual blocks are pre-norm, with embedding and attention dropout 0.1. The head's key, value and query projections are linear in its input, keys are ℓ_2-normalized, and the write gate is a sigmoid of a linear projection. Sequences use a constant filler token outside pair and query positions. Query positions are drawn with the power-law exponent 0.01 of the MQAR generator of Arora et al. (2024), so most queries fall near the end of the sequence.

Models trained on 64 pairs. These runs see 64 pairs in sequences of length 512, with 100,000 fresh sequences per epoch for up to 64 epochs. The optimizer is AdamW with batch 512, weight decay 0.1 and learning rate 10^{-4} (two of the four width-64 seeds use 4.6×10^{-4} instead), with no warmup or schedule. Under this recipe widths 16 and 32 never leave chance, whereas width 64 reaches 0.91 to 1.00 and widths 128 and 256 reach 1.00.

Fixed-load runs at width 256. These use the same recipe with the load raised to 128, 256, 384 or 512 pairs (32 epochs, batch 512) or to 1024 pairs (batch 128), and the sequence length is always four times the load.

Curriculum and its controls. The curriculum runs use batch 128, AdamW with weight decay 0.01 and betas $(0.9, 0.95)$, gradient clipping at 1, a 200-step linear warmup followed by a constant learning rate, and bf16. The learning rate is 4.6×10^{-4} at widths 32 and 64 and 10^{-4} at widths 128 and 256, selected by a width sweep at 64 pairs.

The curriculum proceeds as follows. Every 100 steps we redraw the training set. Three quarters of its sequences are at the current load, and the rest are spread over seven loads log-spaced between $d_k/2$ and the current load, all at sequence length four times the current load. After each 100 steps we test the model at the current load and update an exponential moving average of that recall with weight 0.5. The load rises by ten percent once the average passes 0.9 and the model has spent at least 200 steps at the current load. The average is restarted at every raise.

The fixed-load controls use the same optimizer settings with the load held at 64, 150 or 459 pairs for widths 32, 64 and 128, trained for 32,000 steps. These are the runs reported in Section 6. An earlier set of controls with the same seeds, run on a different GPU and stopped at 20,000, 22,000 and 19,000 steps (the step counts at which the curriculum passed each load), left the plateau at different points. In that set the width-64 run left chance at about 7,000 steps and reached 0.73 by 22,000, and the width-128 run was at 0.045 after 19,000 steps. The 32,000-step reruns left chance at 20,000 and 16,000 steps. The two sets differ only in hardware, so the length of the plateau is not stable across runs.

pairs / d_k	recall	own-key read	ridge read	key survival, learned vs random
1	0.997 ± 0.003	0.994 ± 0.009	1.000 ± 0.000	0.71 vs 0.72
2	0.75 ± 0.18	0.87 ± 0.12	0.99 ± 0.03	0.59 vs 0.58
3	0.37 ± 0.21	0.66 ± 0.13	0.93 ± 0.09	0.50 vs 0.49
4	0.16 ± 0.14	0.50 ± 0.11	0.83 ± 0.15	0.43 vs 0.43

Table 1: Recall, own-key read, ridge read and key survival against load, for the sixteen width-128 models trained on 64 pairs. Mean and standard deviation over seeds on 256 fresh sequences per load. The three reads are defined in Section 3. Key survival is the fraction of a pair's key direction left after all later writes, computed with the model's learned keys and, for comparison, with random unit keys.

Capacity estimates in Figure 1. For models trained on 64 pairs, each circle is the geometric mean over seeds of the load at which recall on fresh sequences crosses 0.9, interpolated in log load, over 4, 16 and 2 seeds at widths 64, 128 and 256. For curriculum models, each square is the geometric mean over seeds of the last load passed, over 1, 2, 2 and 1 seeds at widths 32, 64, 128 and 256. The curriculum's moving-average recall criterion agrees with the fresh-sequence crossing to within one pair. No model trained on 64 pairs at width 32 leaves chance. From width 64 upward, curriculum capacity is 0.40 to 0.43 of the top-1 ceiling.

B THE OWN-KEY READ, AND WHERE IT IS NOT VALID

The own-key read replaces layer 2's query at each query position with the key the pair was written under, taken from the same forward pass at the pair's value token. Two details of the implementation matter. First, the layer's shared direction is projected out of the key before the read, for the reason given in Section 3: every key has a large component on that direction, so a read with the raw key collects every other pair's value as crosstalk. Second, the sign of the key is matched to the sign of the query, since the output projection can absorb a sign. Three controls check that the read finds pairs in the state and nowhere else: reading with a random unit key, reading another sequence's state, and reading after every pair's key has been erased. All three return chance at widths 64 and 128.

At width 256 the second control fails. Another sequence's state returns 0.97 of the pairs at four times d_k on both seeds. The reason is that at this width the write key encodes the value token as well as the key token. A state built from a thousand such writes is then a generic map from key space to value space, and it returns the value of a key it never saw. A read with a key that carries only the key-token part, the mean write key of that token across sequences, returns 0.30 at twice d_k and 0.01 at four times, within 0.02 of the model's recall (Figure 7, left). We therefore leave the width-256 own-key read out of Section 5.

A separate control checks that the loss of recall past d_k is caused by the number of pairs and not by the length of the sequence. With the pair count fixed and the sequence stretched to 16 times the pair count, recall moves by under 0.01 at widths 64 and 128.

The read is also not valid on the width-128 curriculum model of seed 0, whose reader uses the very shared component that the projection removes. There the read falls to 0.36 while recall is 0.99 (Section 6).

The ridge read under a second scorer. The ridge read of Section 5 passes the least-squares state through the model's output path, so that all three reads share one scorer. A scorer that needs no trained output path reads the least-squares state with each pair's own key and counts the pair as returned when the readout is closer, by cosine, to its own value than to any other value in the sequence. Table 2 gives the ridge read and the own-key read under both scorers, averaged over the sixteen width-128 models. The storage gap at four times d_k is 0.33 through the model's path and 0.32 under the nearest-value scorer. The two scorers do disagree on single models. On one seed the model's path returns 0.45 of the least-squares state where the nearest value returns 1.00, so on that model the output path, and not the state, limits the own-key read.

pairs / d_k	through the model's output path own-key read	through the model's output path ridge read	nearest stored value own-key read	nearest stored value ridge read
1	0.994	1.000	0.96	0.999
2	0.87	0.99	0.85	0.97
3	0.66	0.93	0.70	0.93
4	0.50	0.83	0.57	0.89

Table 2: The own-key read and the ridge read under the two scorers, mean over the sixteen width-128 models trained on 64 pairs. The first pair of columns repeats Table 1. The second scores a read as correct when its readout is closer by cosine to the pair's own value than to any other value in the sequence. The storage gap, ridge read minus own-key read at four times d_k, is 0.33 under the first scorer and 0.32 under the second.

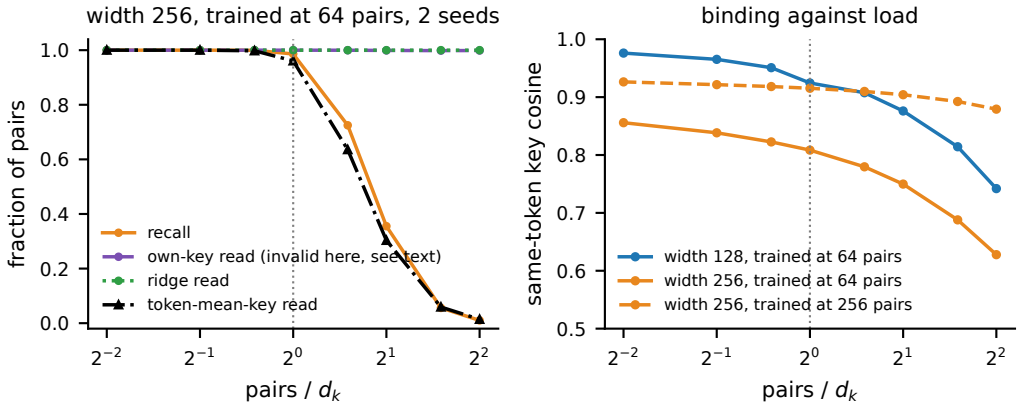


Figure 7: **Left:** width 256 trained on 64 pairs. The own-key read stays at 1.0 because the write key carries the value token, whereas a read with the token's mean key tracks recall. **Right:** binding against load, measured as the cosine between the write keys of the same token in different sequences, for width 128 and width 256 trained on 64 pairs and width 256 trained on 256 pairs. The model trained on 256 pairs keeps its keys consistent where the models trained on 64 pairs drift.

C THE EDITS, AND THE WRITE TEST

After all pairs are written, for each pair s we apply $\mathrm{write}_{k_s, v_{\mathrm{donor}}}(\mathrm{erase}_{k_s}(S))$ to the second layer's state, where k_s is the layer's key at pair s's value token and the donor is a real value from another pair of the same sequence, and then score whether query s answers with the donor. On the 35 models tested, the donor is returned on 0.993 to 1.000 of queries, and the other pairs' recall changes by 0.016 on average and at most 0.06 on one model. A random key in place of k_s makes the model answer with the donor on at most 0.16 of queries. Erase and write therefore agree on the location of a pair, the key it was written under. They do not agree on the size of the effect: erase damage varies across pairs (Figure 4, middle), whereas write success does not.

D THE CURRICULUM CHECKPOINT BY CHECKPOINT

Every curriculum run saved its weights at every raise. Figure 10 shows the final checkpoints of the four runs at widths 64 and 128 against load, and the same measurements on every earlier checkpoint give the trajectories described in Section 6. At width 64 the write gate, the fraction of key variance on the shared direction, and key survival all change over the curriculum, on both seeds. At width 128 seed 0 they remain constant while recall rises to meet an own-key read that was already high at the first checkpoint.

The top-1 ceiling of Section 2 is stated for every pair in a sequence being right, so we also score the final checkpoints by that stricter criterion. The final width-64 checkpoint answers every query

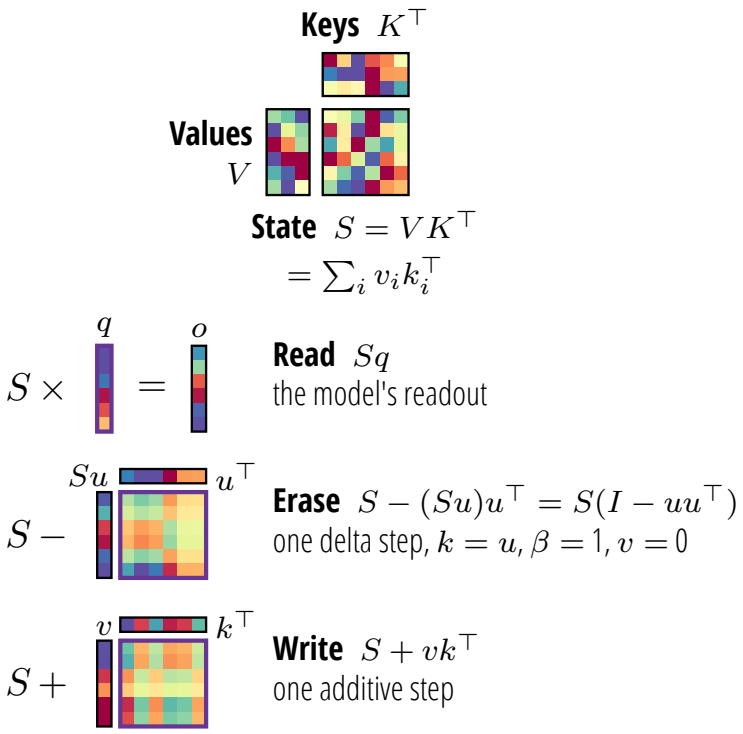


Figure 8: The read and the two rank-one edits of Section 3, on a synthetic 6×6 state that holds three pairs. Purple marks what we supply. Zeroing the state is not drawn.

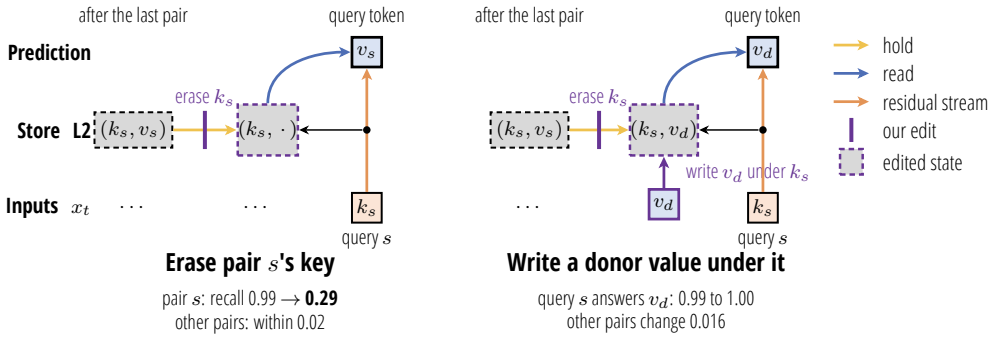


Figure 9: The two edits on the store, each a separate run on a clean model. After all pairs are written, the second layer's state is edited once and the model then runs on. **Left:** erasing the key of pair s, the key the layer computed at that pair's value token, removes pair s and leaves the others. **Right:** erasing that key and writing a donor value v_d under it makes query s answer with the donor. Numbers are from width-128 seed 0 (left) and the 35 solved models of the write test (right).

in a sequence correctly on 0.80 of sequences at 64 pairs and on none at 128, where its mean recall is 0.95. The final width-128 seed-1 checkpoint does so on 0.77 of sequences at 128 pairs and on 0.04 at 256, where its mean recall is 0.99. Past d_k the errors are therefore spread as a few wrong queries per sequence rather than concentrated in a few sequences.

E THE READS AND EDITS ON A PRETRAINED 1.3B MODEL

The main text studies two-layer models constructed to make the mechanism visible. This appendix applies the same reads and edits, and the same separation of storage from retrieval, to a model trained

width	training	last load passed	load / d_k	Hebbian line	top-1 ceiling	recall at $1, 2, 3, 4 \times d_k$
32	curriculum, seed 0	25	0.8	36	109	0.85, 0.33, 0.09, 0.04
32	64 pairs, 2 seeds			36	109	0.00, 0.00, 0.00, 0.00
64	curriculum, seed 0	150	2.3	109	350	1.00, 0.95, 0.76, 0.52
64	curriculum, seed 1	136	2.1	109	350	1.00, 0.94, 0.74, 0.50
64	64 pairs, 4 seeds			109	350	0.96, 0.69, 0.35, 0.17
128	curriculum, seed 0	459	3.6	350	1161	0.99, 0.99, 0.96, 0.85
128	curriculum, seed 1	459	3.6	350	1161	1.00, 0.99, 0.96, 0.87
128	64 pairs, 16 seeds			350	1161	1.00, 0.75, 0.37, 0.16
256	curriculum, seed 0	1593	6.2	1161	3956	1.00, 1.00, 1.00, 0.99
256	64 pairs, 2 seeds			1161	3956	0.99, 0.36, 0.06, 0.01

Table 3: The load curriculum against the models trained on 64 pairs, per width. Last load passed uses a moving-average recall threshold of 0.9. Final-checkpoint recall is evaluated on 256 fresh sequences per load. Comparison rows average models trained on 64 pairs at the same width.

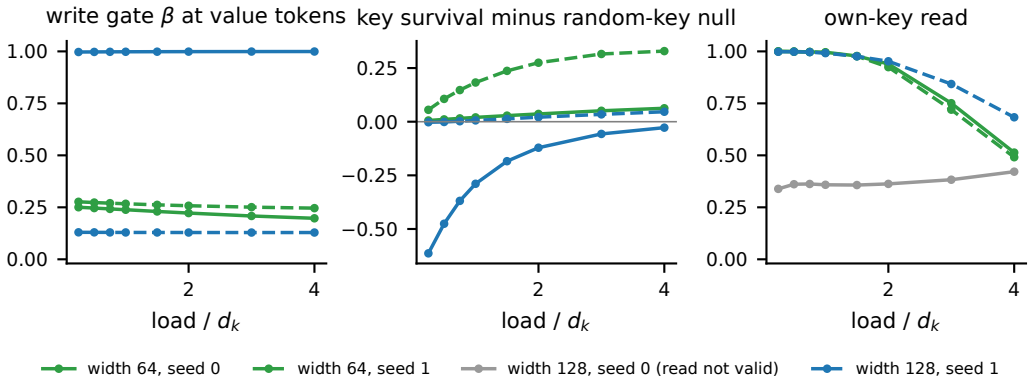


Figure 10: Final checkpoints of the four curriculum runs, read against load. Three runs write with a gate below one, onto keys that survive later writes better than random keys, and their own-key read matches their recall. Width-128 seed 0 writes with gate 1 onto keys that share one direction and survive worse than random keys, and its own-key read (gray) is not a valid measurement.

on language, as a check that the instrument is not specific to the two-layer setting. We use a public 1.3B Gated DeltaNet (Yang et al., 2025) checkpoint released by Du et al. (2025) and trained on 100B tokens of SlimPajama, which has 24 layers of 4 heads with $d_k = 256$ and no short convolution. We edit it at inference time only.

The edits transfer to ordinary text. We first ask whether an association that the model acquires from prose resides in one key direction of one head, as in the two-layer models. We take 200 documents of 512 tokens from the model's pretraining distribution. We define a *recall event* as a word that occurs twice with the same word following it both times, where the model predicts that next word at the second occurrence and finds it at least one nat easier there than at the first. The association was written when the token after the first occurrence was processed, under the model's own unit key at that position. We therefore erase that key from one layer's state immediately after the write and score the drop in log probability of the recalled word. Over 161 events, layer 16 costs 0.50 nats and flips the prediction on 0.21 of events, layer 11 costs 0.28 and flips 0.13, and every other layer costs under 0.10 (Figure 11). Within those two layers the effect is concentrated in one head each: L16.H2 costs 0.29 nats and L11.H0 costs 0.28, whereas L16.H3 costs 0.10 and every other head in those layers costs under 0.02. Three controls are at zero in every layer: the same key erased just before the write, a random direction, and the key of a token five positions later. Collateral damage on the other tokens of the document is under 0.002 nats. The write test provides the sufficiency half. For each event we take a donor event from another document, read what the donor's state holds under the donor's key, and write that value under this document's key at the same cut. Over all 96 heads, the gain from writing and the loss from erasing have a Pearson correlation of 0.99. The two heads where erasing hurts most, L16.H2 and L11.H0, are the two where writing

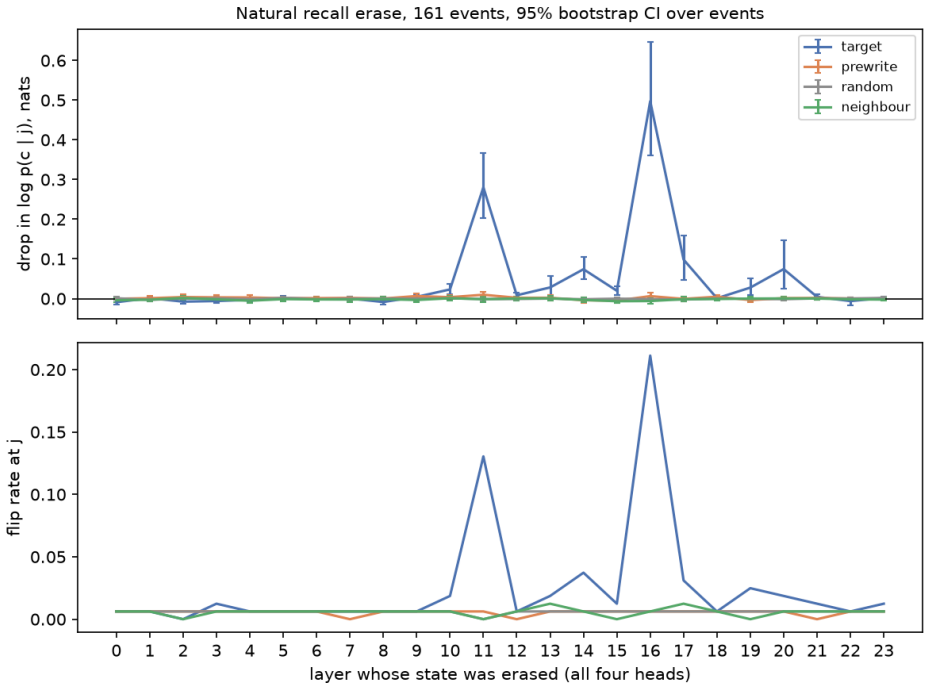
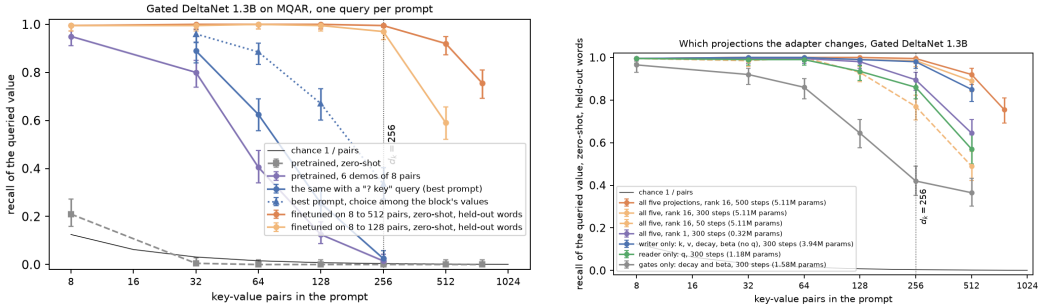


Figure 11: Erasing one word's key from one layer of the 1.3B model, on 161 recall events from ordinary text. Blue: the drop in log probability of the recalled word when the key is erased right after its write, one layer at a time. Orange, gray and green: the same key erased just before the write, a random direction, and the key of a token five positions later. Bootstrap intervals over events.

works best. Writing in both raises the log probability of the donor's word by 2.26 nats, whereas the same value under a neighbor's key or a random key moves it by at most 0.02. In this model, as in the two-layer models, a pair therefore resides in one key direction of one head, and the location where erasing removes it is the location where writing installs it. The model also has a previous-token register, found by the same search as in Section 4. Erasing one direction of head L10.H2 after every token of a repeated span reduces copy accuracy from 0.866 to 0.491, whereas a random direction leaves it at 0.866. Writing a stale value into that direction causes the model to copy the token two positions ahead on 0.539 of positions, against 0.002 before. That register is in a different layer from the two store heads, and erasing it costs 0.016 nats on the recall events, so register and store are separate in this model as well.

How many pairs the pretrained model recalls. The question of how many associations the matrix holds can be asked of this model directly with MQAR in a prompt. A bare prompt does not pose the question fairly. Given "key value, key value, . . . , key" the model recalls 0.32 of 8 pairs and 0.11 of 32, and on most of its misses past 32 pairs its top prediction is a comma, which indicates that it has lost track of whether the last token was a key or a value. Six short demonstration blocks of 8 pairs on fresh keys, together with a "? key" marker at each query, resolve most of this, and every other template we tried is worse. Under that prompt, recall by argmax over the whole vocabulary is 0.89 at 32 pairs, 0.63 at 64, 0.26 at 128 and 0.025 at 256, which is d_k. Restricted to a choice among the values in the prompt, the same predictions are correct 0.96, 0.89, 0.67 and 0.335 of the time (Figure 12a, blue). The two curves differ only in how the same prediction is scored, so the difference between them is not the storage gap of Section 5, which compares two states under one scorer. What the difference shows is that the model's prediction separates the correct value from the other values in the prompt well past the point where that value is the top of the vocabulary.

How much room the state has. To measure the capacity of the state rather than the quality of the reading, we train a small adapter and leave the checkpoint frozen. The adapter has rank 16 on the



(a) MQAR recall against pairs on the 1.3B model.

(b) Adapters on subsets of the projections.

Figure 12: The pretrained model on MQAR, 200 prompts per point with 95% intervals. **(a)** Gray: the bare prompt. Purple: six 8-pair demonstrations on fresh keys. Blue: the same with a "? key" query marker, the best prompt found, solid for the argmax over the vocabulary and dotted for the choice among the prompt's values. Orange: the 5.1M-parameter adapter, zero-shot on held-out words. Chance is one over the number of pairs. **(b)** Zero-shot recall of adapters trained on subsets of the projections, 300 steps each unless marked.

query, key, value, decay and β projections of every layer, 5.1M parameters in total, and is trained for 500 steps on MQAR sequences of 8 to 512 pairs of random single-token words, which takes about four minutes. It is evaluated zero-shot on words it never saw. Recall is 1.00 at 32 to 128 pairs, 0.995 at 256, 0.92 at 512 and 0.755 at 768, which is three times d_k. The next-token loss on a fixed English paragraph moves from 1.87 to 1.91 (Figure 12a, orange). Trained only up to 128 pairs, the adapter still reaches 0.97 at 256 and 0.59 at 512. Which projections carry this improvement indicates where the pretrained model falls short. Adapting the query projection alone, which changes what is read out of the state and nothing about what is written into it, raises recall at 256 pairs from 0.00 to 0.86. The writer side alone (key, value, decay and β, without the query) gives 0.98 at 256 and 0.85 at 512, whereas the query alone gives 0.57 at 512. The two gates alone, two scalars per head per token, give 0.645 at 128 pairs (Figure 12b). The pretrained writer therefore already stores the bindings, and the pretrained reader does not return them. This is the same separation as in the two-layer models, obtained here by training instead of by reading the state. Past $2d_k$, how pairs are written must change as well. Scale alone does not produce this. Under the same best prompt, six public checkpoints from 340M to 2.7B in three families all lie between 0.12 and 0.44 at 128 pairs and at or below 0.14 at 256. The 340M and 1.3B gated models with the same head shape are within 0.07 of each other at every load, and the step from 1.3B to 2.7B adds at most 0.05. Head shape and update rule move the curve more than parameter count does, with the DeltaNet family of many narrow heads reading best and the additive GLA models reading worst.

What these results suggest. We interpret the two-layer and pretrained results together as follows, and we state these as directions rather than as results. First, a recall failure on a delta-rule model should be diagnosed before more state is added. In every model we measured, the state holds more pairs than the model returns, and the fix lies in how the state is read and written rather than in its size. Second, the training load determines the capacity. A curriculum over association density, with replay of lower loads so that earlier levels are not forgotten, is therefore the natural way to train the state to hold more, and the adapter result, on one model, indicates that a pretrained model needs little such training. Third, the choice among the prompt's values, or an adapter on the query projection alone, provides a way to measure the gap between storage and retrieval of any linear-attention model from its outputs, without reading its state. Whether any of this changes the behavior of a hybrid model, where the attention layers can carry what the state does not, is the open question left at the end of the introduction.