CONTINUAL LEARNING

Programmable Neural Memory: Recognition and Learning in a Continuously Updated Weight Array

A bounded empirical study of learned memory writers, explicit update rules, and recognition across unfamiliar object views

Revanth Matha · Matha Labs | Preprint awaiting review 1.0 | 2026-11-20

Abstract

A system encountering new information after training must decide what to retain, how to represent it, and how later queries retrieve it. We evaluate an explicit inference-weight array with separate read, write and update-policy components. The symbolic prototype combines a 17-value codec, 64 one-hot addresses and a state-conditioned writer that emits rank-one changes; residual writer adapters learn online under controlled permission and capacity rules. Across 12 synthetic trials per condition, allowing all writes yields 93.60% current-truth recall, while protecting all profile keys yields 72.25% and perfectly retains the initial anchor values. Under the same anchor policy, competence-triggered capacity reaches 87.87%, compared with 87.74% for interval unlocking and 87.07% for one online-trained adapter. These small differences are descriptive. A separate visual arm learns a 32-dimensional representation from 60 COIL-100 objects and constructs prototype classifier rows from four support views of each of 20 held-out objects. On 1,360 other-angle queries, learned features achieve 96.18%, versus 95.29% for raw pixels and 83.60% for an initial random encoder. The learned-versus-pixel difference is 12 answers net, with a wide object-cluster bootstrap interval. A simple exact-codec symbolic baseline matches the learned writer on a SET-only control. The contribution is an inspectable empirical comparison of numerical memory mechanisms, not the invention of fast weights or evidence of human-equivalent memory. Exact addressing, supervised current-event updates, one pretrained kernel, controlled image acquisition and short timing runs bound the conclusions.

Keywords: neural memory, fast weights, hypernetworks, continual learning, weight-space programming, object recognition, memory interference

Contributions

1. A complete symbolic write/read path in which learned components generate numerical changes to a 16 x 64 inference-weight matrix, with explicit state, update permissions and saved-checkpoint replay.

2. A seven-condition, 84-trial comparison of initial-fact retention, current-truth recall and adapter-capacity timing, with fixed optimization budgets, joint context controls and a separate resource-allocation experiment.

3. A held-out-object recognition experiment using real COIL-100 images, paired raw-pixel and random-encoder controls, reconstruction displays and object-level uncertainty analysis.

4. An auditable boundary between observed memory behavior, short concurrent-snapshot measurements and unsupported claims about broad generalization or biological memory.

1. The problem: remembering something that was not in training

A model can appear knowledgeable while remaining unable to retain an ordinary new event. If a user supplies a new association, a correction or an unfamiliar object, the system needs a mechanism that makes this information available beyond the immediate input. The central question is operational: what changes inside the system, what remains fixed, and what evidence shows that the change improved later answers? Calling every such mechanism memory obscures the important differences between an input sequence, a searchable record and an altered function.

This study examines memory that directly participates in numerical inference. The memory is represented by an array whose entries change as observations arrive. A query is transformed into an address, the array produces a numerical response, and a learned reader decodes that response. This is narrower than a language model reflecting on a conversation, and more specific than saving a paragraph that may later be retrieved. It also makes failures observable: the write may be inaccurate, a later write may damage an earlier association, or the reader may confuse otherwise distinct memory states.

There are two experimental questions. In the symbolic arm, does a history-conditioned writer preserve useful associations while responding to deliberate changes in what should be remembered? In the visual arm, can a representation learned from one group of objects support memory-based recognition of different objects from new viewpoints? Success in one arm does not answer the other. Symbolic addresses can make retrieval straightforward even when a visual encoder remains unreliable, while good image classification can arise without a useful temporal update policy.

The scientific value therefore depends on controlled comparisons, not on describing a weight matrix as a brain. The same observations must be available to competing methods, later test queries must not supply the correct answer to the writer, and retention must be measured after additional updates. A useful positive result is a bounded improvement with a known mechanism. A useful negative result identifies which part of the proposed interface fails.

2. An explicit separation between a reader and its changing memory

The symbolic prototype separates a value codec, a core writer, three residual writer adapters and the mutable memory state. There are 17 value categories: UNKNOWN plus 16 named values. The codec maps values into 16-dimensional vectors and reads such vectors with a 16 -> 64 -> 17 network using a tanh hidden layer. UNKNOWN is represented by the zero embedding. There are 64 explicit addresses: 32 profile columns and 32 episode columns. Supplying an address avoids entity recognition. It does not demonstrate that the system can discover where a fact belongs in unrestricted language.

The core writer is a 52 -> 96 -> 16 tanh network. Its 52 input components are the 16-dimensional current column, the 16-dimensional payload, the 16-dimensional mean of all current columns, and a four-component operation indicator. Each of three residual adapters is a 52 -> 48 -> 16 tanh network, with its last layer initialized to zero. A stage mask determines which adapter outputs contribute. This is an actual numerical weight-writing path: the generated vector is applied as a rank-one change to a 16 x 64 array, and answers are decoded from array-query multiplication.

The base value codec and core writer are frozen before stream evaluation. The residual adapters are trained online. Thus, a memory write changes W, while an optimizer step changes the writer parameters that determine later writes. These are distinct forms of adaptation. The implementation does not rewrite its own architecture or the frozen codec, and it does not use a frontier language model as the online editor.

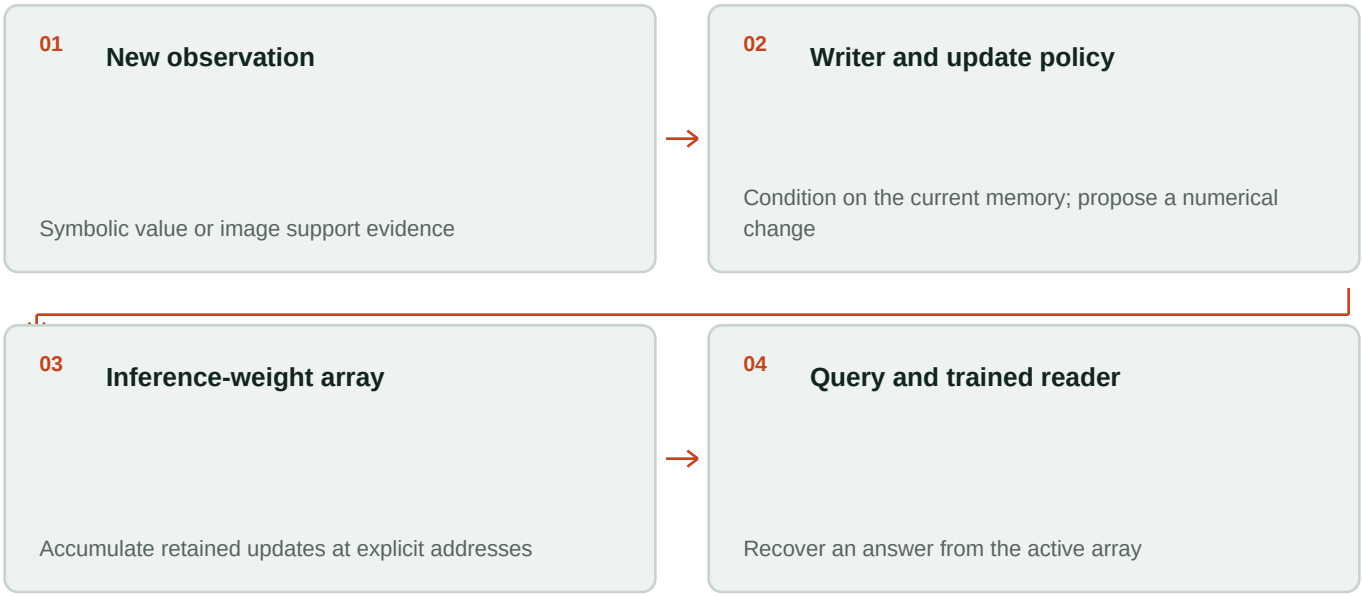
The history feature is the mean of the complete current memory matrix, together with the addressed column. It is a lossy summary of the present bank, not attention over a preserved raw history. Our control removes the old column and the aggregate together. It does not isolate the aggregate alone. Consequently, an observed difference cannot be attributed specifically to a useful global history representation. Different streams may share the same bank mean.

The numerical state is represented as a base array, a consolidated array and a list of keyed update vectors. Their sum is W. At initialization, the 32 profile columns are populated directly from learned codec embeddings, while episode columns start at zero. These initial facts therefore do not test the writer. Subsequent inference reads W, not the symbolic ground-truth dictionary used by the evaluator. That dictionary generates supervision and scores the experiment. Excluding it from prediction does not make the learning unsupervised.

```
F_t = [W_t[:,k], E(v), mean_j W_t[:,j], onehot(op)]; u_t = G(F_t) + sum_i mask_i A_i(F_t); W_(t+1) = W_t + u_t e_k^T
```

E and the reader are pretrained; G is the frozen core writer; A_i are online-trained residual adapters. e_k is the 64-component one-hot address. The write occurs only if the chosen update policy permits it. COPY uses the source memory column as the payload and RETRACT uses zero.

Figure 1. The experimental memory interface



A schematic of the experimental roles. It is not a diagram of a biological memory system. The symbolic and image arms use different inputs and are evaluated independently.

3. A memory language as a contract

The memory language in this experiment is a deliberately small interface rather than arbitrary natural language. Its four operation identifiers are SET, COPY, SHIFT and RETRACT. SET supplies a named value. COPY takes a value from another addressed column. RETRACT targets UNKNOWN. SHIFT requests a modular change among the 16 nonzero value categories. These identifiers, the 17-value vocabulary and the explicit key determine what the writer can be asked to do.

Before stream evaluation, the codec is trained for 500 Adam steps at learning rate 0.012. Its objective is cross-entropy for recovering all 17 value identities from their embeddings, plus 0.003 times mean squared embedding magnitude. The core writer is trained for 1800 batches of 256 examples at learning rate 0.003. Current-value inputs include Gaussian perturbations of scale 0.05, while the aggregate input is a mean of eight sampled value embeddings. The core receives SET, COPY and RETRACT. SHIFT is reserved for subsequent adaptation.

The core training objective combines mean squared error on the desired post-write vector with 0.05 times the decoder cross-entropy. This teaches a useful update convention rather than storing an evaluated conversation. The initial exhaustive SET probe enumerates all 289 old/new value pairs. Its 100% categorical readout and mean squared error 0.003164 are checks of this finite vocabulary interface. Since these value pairs can appear in random pretraining, that probe alone is not a held-out-combination generalization test.

A trainable code introduces a compatibility dependency. Changing the codec without rebuilding stored state can invalidate the meaning of a previously written vector. A saved bank therefore belongs with the reader and writer checkpoints, vocabulary, address convention and numerical precision used to create it. This release evaluates one such configuration and does not establish exchange of modules between independently trained models.

4. What a rank-one write does and does not imply

An outer-product update has an inspectable effect on a later query: (u e_k^T)q = u(e_k^Tq). For the one-hot keys used here, a write at k contributes only to that column. Consequently, agreement on untouched keys is partly a structural property of the addressing scheme. A score of 100% on an unaffected-key test would not by itself establish learned protection of distributed knowledge.

Interference can still arise through imperfect writes, errors inherited from a copied source, or the decoding of a changed column. Subsequent online training changes the writer used for future operations, but leaves already stored vectors and the fixed reader intact. Separating these pathways helps explain why a method can preserve unrelated columns while giving the wrong answer at the updated address.

When the retained block list reaches 32 entries, the implementation merges its oldest 16 updates into the consolidated array. This changes the representation of the sum rather than intentionally discarding information. Floating-point addition order can nevertheless change individual entries slightly. The experiment records the largest discrepancy across compactions. This is arithmetic consolidation, not a learned sleep-like process or evidence of semantic abstraction.

Every 32 stream events, episode columns 32-63 are reset to zero and their pending blocks are removed. Profile columns remain. This is an explicit scope rule supplied by the experimenter. A reset is not learned forgetting, and retention across a reset should be interpreted in terms of the scope intended to survive it.

The array has 1024 float32 entries, or 4096 bytes when materialized once. The implementation retains additional arrays and a list of update factors. A byte count for one matrix is therefore not the memory use of the complete program. The cost section distinguishes the matrix, the learned components, the replay buffer and the audit artifacts.

5. Updates require a policy, not just a writable array

Three policies determine whether a generated update enters the bank. The revisable policy permits writes to every key. The protected policy refuses writes to all 32 profile keys. The anchor policy refuses writes to the first eight profile keys and permits writes elsewhere. These are hard conditions on addresses, not learned judgments about the credibility or importance of an observation.

The evaluator's current-truth stream includes changes even when the policy refuses them. The protected and anchor conditions therefore have intentionally competing objectives: preserve selected initial answers while also tracking a changing world. Their lower current-truth scores must not be interpreted solely as failed optimization. Some disagreement is the direct consequence of refusing an update that the ground truth regards

as valid.

Initial-anchor retention is scored on keys 0-7 against their initial values after each event. Current-truth accuracy is scored against the evolving target bank. These denominators answer different questions and should appear together. The protected condition's ability to retain original anchors is a consequence of its restriction. It does not show that the system autonomously identified which facts deserved permanent protection.

The stream consists of 128 events and four 32-event phases. The first phase mainly supplies SET operations. The second introduces COPY. Later phases introduce modular SHIFT by one and then two. Every thirteenth event is a RETRACT, and periodic SET events continue in later phases. The initial 32 profile values and each event sequence are generated from the declared seed and world index. Episode state resets at phase boundaries. This controlled change in operation demand probes adaptation, not open-ended autobiographical time.

6. When should the writer itself become more adaptive?

All seven stream conditions train residual writer adapters online. The fixed condition keeps one adapter active. It does not freeze online learning. Other conditions may activate a second and third adapter. The comparison therefore tests access to additional trainable capacity and its timing, not the presence versus absence of learning.

The interval gate activates stages at events 42 and 84. The compute gate activates them after 168 and 336 previous optimizer steps. Because every event uses four optimizer steps, these two definitions are deliberately equivalent in this protocol. Identical outcomes provide a consistency check. They are not independent evidence that two different scheduling principles discovered the same solution.

The data gate uses 42 and 84 distinct (key, operation, payload, source) tuples. The competence gate checks every 16 events from event 32 onward. It evaluates the preceding 32 replay examples, explicitly excluding the current event, and activates one additional adapter if categorical accuracy reaches 95%. These are familiar supervised replay examples, not an untouched test set. The fixed gate never increases capacity beyond the first adapter.

For each event, the labeled current example enters the replay buffer before four Adam updates. Every update samples exactly 16 entries with replacement, including during warmup, and optimizes post-write-vector MSE plus 0.08 times decoder cross-entropy at learning rate 0.0015. Each trial therefore contains 512 backward steps and 8,192 sampled example slots. The evaluator records the proposed write before training separately from the accepted bank after training and policy enforcement. The main recall score is supervised post-update performance, not prequential prediction of an unseen label.

Every arm executes all three dense adapter branches and four backward passes per event, using matched replay sample indices. Inactive branches are masked from the output. This matches the main dense optimization work but not every source of wall-clock cost: competence probes add forward passes, and accepted-write policies differ in compaction work. The paper reports those differences instead of treating similar optimization counts as identical end-to-end computation.

7. A separate test on object images

The visual arm uses the official COIL-100 archive: 7200 views of 100 objects. Each object has 72 viewpoints. Images are converted to RGB, resized bilinearly to 24 x 24 and divided by 255, giving 1728 scalar inputs. Objects 1-60 supply 4320 representation-training images. 61-80 provide validation. 81-100 are held out for the final recognition task. [16]

The image model is a multilayer perceptron with a 1728 -> 128 -> 32 encoder using ReLU, a 32 -> 128 -> 1728 decoder with ReLU and sigmoid output, and a 32 -> 60 classifier used only during representation training. It has 454556 learned parameters in total. Training uses 25 epochs, Adam at 0.002, batch size 128 and seed 1729. The loss is pixel MSE plus 0.12 times cross-entropy over the 60 training identities. The highest validation recognition score selects the checkpoint. In the observed run, this is epoch 25.

For each of the 20 held-out objects, views at 0°, 90°, 180° and 270° form the support set. The remaining 68 angles supply queries, producing 80 support images and 1360 queries. Each support feature is normalized, the four support features are averaged, and their mean is normalized into one classifier-weight row. A normalized query is assigned to the row with the greatest dot product. The 20 x 32 weight matrix is directly used for inference.

This visual writer is an analytic mean-prototype construction, not the history-conditioned hypernetwork used in the symbolic arm. It is closely related to prototype-based few-shot classification. No gradient updates occur while incorporating the held-out support objects, and their identities did not train the encoder. The link between the two arms is the explicit construction of inference weights from new observations, not a shared trained mechanism. [19]

The same prototype rule is applied to the initial random encoder and raw pixels. These controls retain the same support/query split and 20-way candidate set. They help distinguish the contribution of learned features from information already available in pixel geometry. All queries belong to one of the 20 supported classes. This is a closed-set task, with no test of rejecting an unsupported object.

Reconstruction MSE is computed over all 1440 views of the 20 held-out objects, including their 80 support views. It is not restricted to the 1360 recognition queries. The decoder reconstructs from the unnormalized learned features. Recognition and reconstruction therefore remain separate measures, and qualitative reconstructions should be displayed at their actual 24 x 24 information resolution rather than presented as recovered photographic detail.

```
f(x) = normalize(E(x)); p_c = normalize((1/4) sum_(s in support_c) f(s)); predict(x) = argmax_c f(x)^T p_c
```

There are 20 candidate held-out objects and four support views per object. The prototype p_c becomes a row in the classifier's weight matrix. E is learned only from objects 1-60; the control encoders replace E while preserving support and query membership.

8. Evaluation design and leakage controls

The symbolic protocol uses stream seeds 11, 23 and 47, four worlds per seed, 128 events per world and 64 one-hop queries after every event. This yields 12 trials and 98,304 one-hop readouts per condition. Another 98,304 readouts test two-hop composition. Seven conditions give 84 trials and 688,128 readouts per hop. All conditions share one pretrained kernel from seed 1729. These are three stream seeds, not three independent pretrained models. Repeated readouts are highly correlated and do not provide 98,304 independent observations.

One-hop ground truth is the current value at each address. In the two-hop task, a nonzero first value is interpreted as a pointer to one of the first 16 addresses, and the correct answer is the value at that address. UNKNOWN remains UNKNOWN. The system constructs the second query from its own first decoded value, so first-hop errors can propagate. This is a finite synthetic composition task, not general multistep reasoning.

Write accuracy measures the currently addressed key after the proposed update and policy decision. Unknown accuracy measures outputs only where the target value is UNKNOWN. Initial-anchor retention compares keys 0-7 with their initial values, while unaffected-key agreement compares non-addressed keys immediately before and after a write. These metrics have different reference states, and none should be substituted for another.

The image run uses a single training seed and fixed identity/angle split. Its query count is 1,360, but the 20 object identities are more appropriate units for estimating variation across objects. All three representations use the same support and query images. We report paired disagreements and a descriptive percentile bootstrap resampling whole objects 10,000 times with seed 1731. This preserves within-object angle dependence. It does not measure variation across retrained models, alternative splits or new acquisition environments.

The result records preserve seeds, dimensions, training counts and the external archive hash. Source-controlled scripts generate the tables and scientific figures from those records. Local durations measure the executed routines, and the separate concurrent benchmark records its own workload and deadlines. The final manifest binds code, checkpoints and results. A changed checkpoint requires a new record.

9. Symbolic results: retention trades against correction

The revisable interval condition achieves 93.60% one-hop current-truth accuracy and 91.90% two-hop accuracy. Protecting all profile keys lowers these scores to 72.25% and 61.52%, while preserving the initial eight anchor values at every measured step. With eight protected anchors rather than all 32 profile keys, interval unlocking reaches 87.74% and 79.23%. The apparent loss is partly the specified policy: an authorized change in the evolving oracle can conflict with a deliberately immutable initial value.

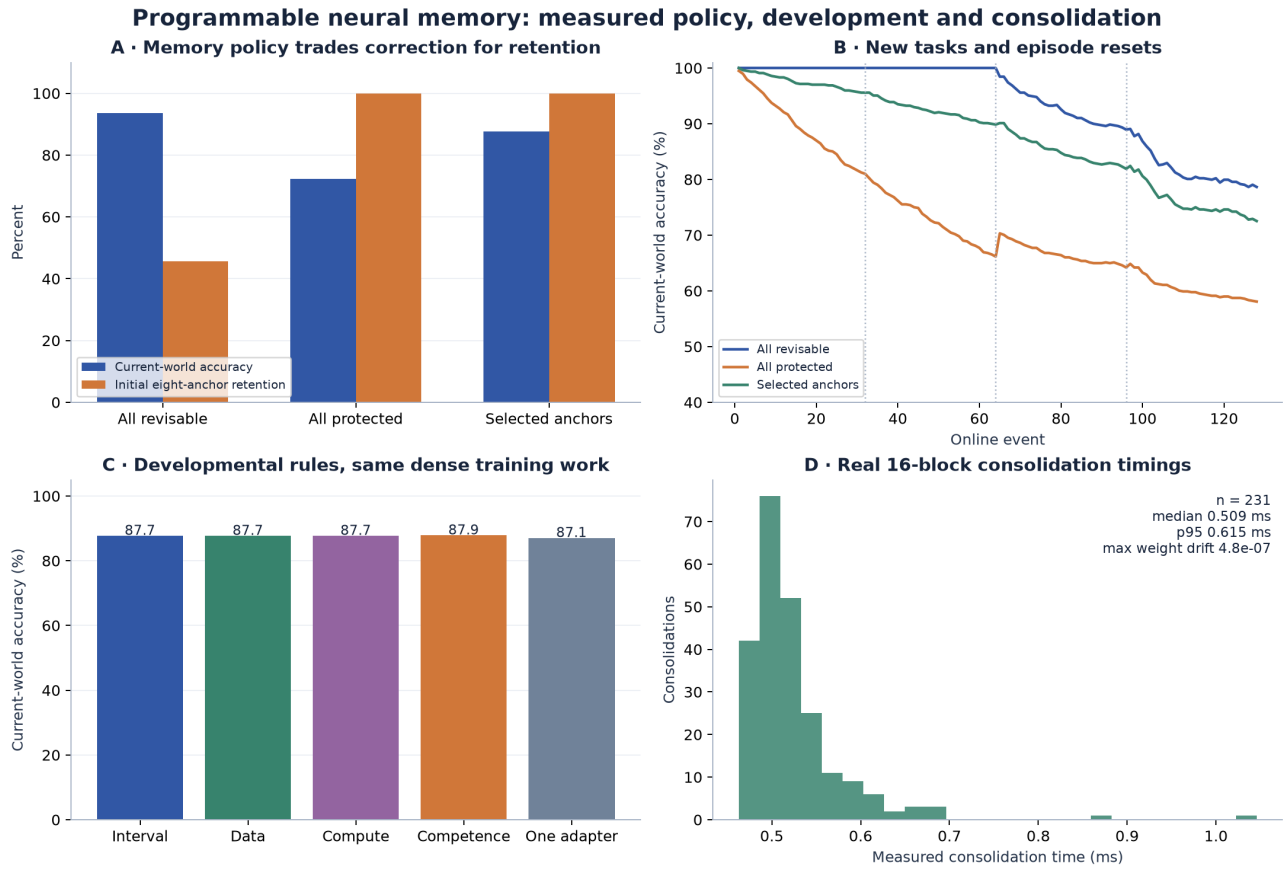
Initial-anchor retention is 100% under every protected or anchor condition, but 45.66% under the revisable condition. This is a tradeoff between two reference states, not evidence that one policy is universally better. The experiment does not infer which facts deserve protection. Their addresses are supplied. The strongest defensible result is that the chosen policy visibly controls this tradeoff.

Within the anchor policy, competence-triggered unlocking reaches 87.87% one-hop recall, compared with 87.74% for interval unlocking and 87.07% for one online-trained adapter. The competence-minus-interval difference is 0.13 percentage points: five of the 12 paired trials improve, one declines and six tie, with trial differences ranging from -0.35 to +0.74 points. Against one adapter, all 12 pairs improve, averaging 0.81 points. These are small conditional differences for one pretrained kernel and fixed streams, not proof of a generally superior capacity-selection rule.

Data-triggered unlocking yields 87.73%. Compute-triggered and interval-triggered unlocking produce exactly the same scores, as expected from their equivalent thresholds at four steps per event. Reporting that equality matters: treating it as corroboration from independent mechanisms would overstate the evidence. Immediate post-policy write accuracy is lower than overall recall because the bank also contains many untouched or easy retained values.

All 84 trials preserve the pretrained codec and core-writer hashes and show 100% agreement on non-addressed keys immediately around accepted writes. Exact one-hot addressing explains the latter property. Arithmetic compaction changes materialized entries by at most 4.77 x 10^-7 in the recorded trials. These checks establish properties of the executed configuration. They do not constitute a formal proof about arbitrary floating-point states.

Figure 2. Actual memory-stream measurements



The plotted data come from all recorded main trials. Policy comparisons use the same generated worlds; trajectories show the programmed operation phases. Adapter schedules are rules chosen for this experiment. Compaction discrepancies reflect float32 arithmetic, not semantic forgetting. Source: results/stream-results.json.

Table 1. Main symbolic conditions

Policy / capacity gate	One-hop recall	Two-hop recall	Post-policy write	Initial anchors
Revisable / interval	93.60%	91.90%	82.42%	45.66%
Protected / interval	72.25%	61.52%	46.81%	100.00%
Anchors / interval	87.74%	79.23%	72.85%	100.00%
Anchors / data	87.73%	79.30%	72.85%	100.00%
Anchors / compute	87.74%	79.23%	72.85%	100.00%
Anchors / competence	87.87%	79.37%	73.11%	100.00%
Anchors / one adapter	87.07%	78.64%	70.83%	100.00%

Each row aggregates 12 trials x 128 events. Recall denominators are 98,304 per hop; write accuracy has 1,536 addressed events. Anchor retention repeatedly measures eight original values. Values are percentages; the oracle changes even when a policy refuses a write. Source: results/stream-results.json.

10. What the controls establish

The core codec and writer pass an exhaustive 289-pair SET probe with 100% categorical accuracy and post-write-vector MSE 0.003164. The finite value pairs may occur during randomized pretraining, so this is an interface sanity check rather than held-out combination generalization. The new stream seeds change event order and state, not the symbolic vocabulary. SHIFT introduces an operation absent from core pretraining, but its online labels are supplied during evaluation.

A separate SET-only sequence uses seed 1931, 128 updates and no online gradient steps. Starting from an empty bank, the full writer correctly answers all 8,192 accumulated key readouts and all 128 immediate addressed writes. Zeroing both the old addressed vector and the bank aggregate lowers recall to 85.39% and immediate writes to 72.66%. This shows that jointly removing those state inputs harms this overwrite task. It cannot isolate the global aggregate: subtracting the old value is already necessary to replace an occupied additive slot.

The exact-codec baseline sets the update to the payload embedding minus the old vector. It also reaches 100% on both measures. Consequently, the learned writer has no demonstrated accuracy advantage over this simple arithmetic construction on the SET-only task. An untrained random reader yields 4.33% recall and 6.25% immediate-write accuracy, illustrating dependence on the learned interface. It is an interface disruption control, not a competitive algorithm.

This distinction makes the negative result useful. The prototype establishes that a trained writer can implement the finite convention, but neither its learned status nor its bank-wide input establishes necessity. A stronger follow-up would compare old-only and old-plus-aggregate writers, hold out operation compositions or vocabulary groups, and evaluate their errors before current-event supervision. Those experiments are proposed, not included in the reported measurements.

Table 2. SET-only state and reader controls

Condition	Correct recall / 8,192	Recall	Correct writes / 128
Full state, learned reader	8,192	100.00%	128
Old vector and aggregate zeroed	6,995	85.39%	93
Random reader	355	4.33%	8
Exact-codec arithmetic	8,192	100.00%	128

One 128-event SET stream, seed 1931, with no online adaptation. The joint ablation does not separately identify the contribution of the aggregate. The exact baseline uses the same learned code but computes replacement arithmetically. Source: results/context-ablation.json.

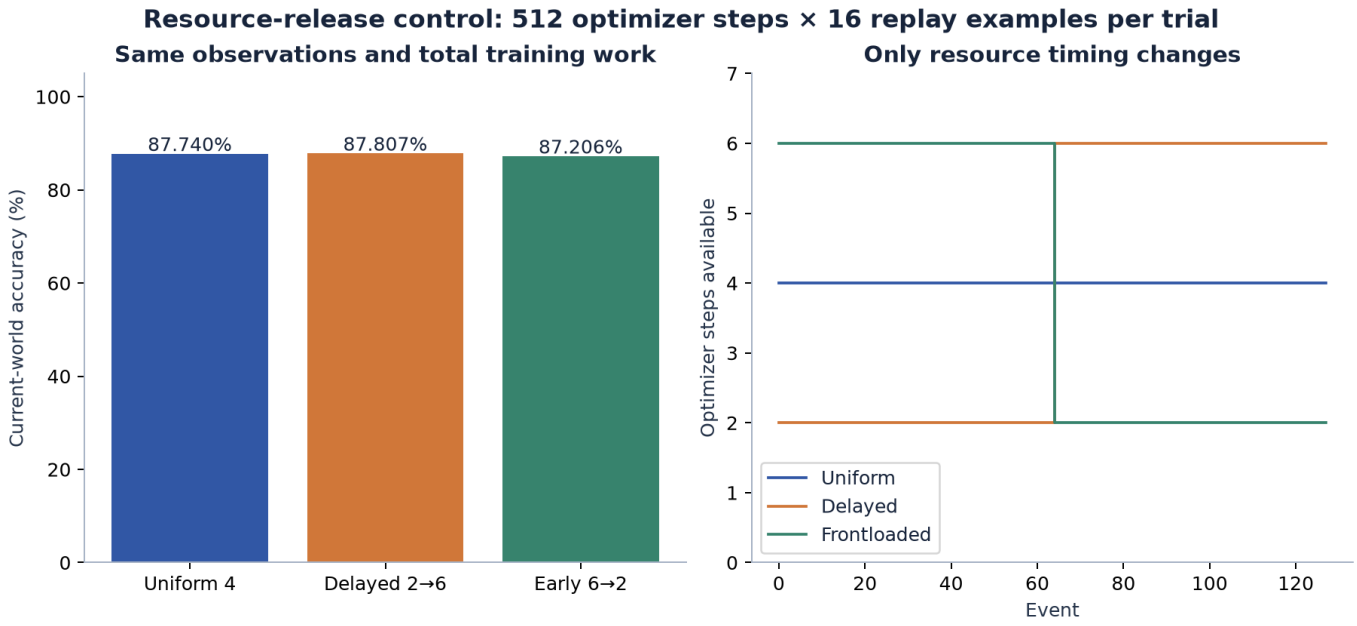
11. Releasing the same optimization budget at different times

To distinguish total optimization from its temporal allocation, a separate experiment retains the anchor policy and compute-based capacity gate while changing when updates are spent. Uniform allocation performs four backward steps per event. The delayed schedule performs two steps for the first 64 events and six thereafter. The front-loaded schedule reverses this order. All three spend 512 steps and 8,192 sampled example slots per trial. The two additional schedules use the same 12 worlds as the main experiment.

The delayed schedule achieves 87.81% one-hop recall, uniform 87.74%, and front-loaded 87.21%. Their two-hop scores are 79.28%, 79.23% and 78.78%. These changes are small and conditional on this sequence of task phases. They do not establish that delayed computation is generally beneficial. Because capacity depends on previous optimization steps, changing the budget schedule also changes when additional adapters become active.

The delayed schedule activates stages at events 71 and 99, compared with 42 and 84 under uniform allocation, and 28 and 56 under front-loading. Thus, the intervention combines the timing of parameter updates and the timing of access to trainable capacity. A further factorial experiment would hold those two factors apart. Our present comparison tests the combined resource policy and identifies this confound explicitly.

Figure 3. Resource timing is a separate experimental intervention



Actual uniform, delayed and front-loaded conditions. Equal total optimizer work does not imply identical adaptation trajectories. The capacity gate responds to cumulative work, so earlier training also unlocks capacity earlier. Sources: results/resource-results.json and results/stream-results.json.

Table 3. Same total budget, different allocation

Allocation	Steps, first / last 64	One-hop	Two-hop	Stage activations
Uniform	4 / 4	87.74%	79.23%	42, 84
Delayed	2 / 6	87.81%	79.28%	71, 99
Front-loaded	6 / 2	87.21%	78.78%	28, 56

Each schedule uses 512 backward steps and 8,192 replay-example slots per trial, with a batch of 16 sampled with replacement from the first event. Uniform is the main anchor/compute condition. Source: results/resource-results.json and results/stream-results.json.

12. Image results: strong controls limit the interpretation

On the held-out-object query set, learned features recognize 1,308 of 1,360 images (96.18%). The same prototype rule with raw pixels recognizes 1,296 (95.29%). An untrained random encoder recognizes 1,137 (83.60%). The learned representation exceeds raw pixels by 12 answers net, or 0.88 percentage points. The strong raw-pixel score is central: much of this controlled-view task is already tractable from low-level image similarity.

The paired outcomes reveal 1,253 images that both learned features and pixels classify correctly, 55 correct only with learned features, 43 correct only with pixels, and nine wrong under both. Four object identities improve, three decline and 13 tie. Object 84 improves by 33 of its 68 queries, while object 85 declines by 18 and object 90 by 11. Aggregate accuracy therefore hides substantial object-specific behavior.

Resampling the 20 whole objects yields a descriptive 95% percentile interval of -4.41 to +7.21 percentage points for the learned-minus-pixel difference. This interval includes zero and is wide relative to the observed 0.88-point gain. It is conditional on the trained checkpoint and fixed object set. Treating individual angles as independent trials would give a misleading impression of certainty. No broad superiority claim follows.

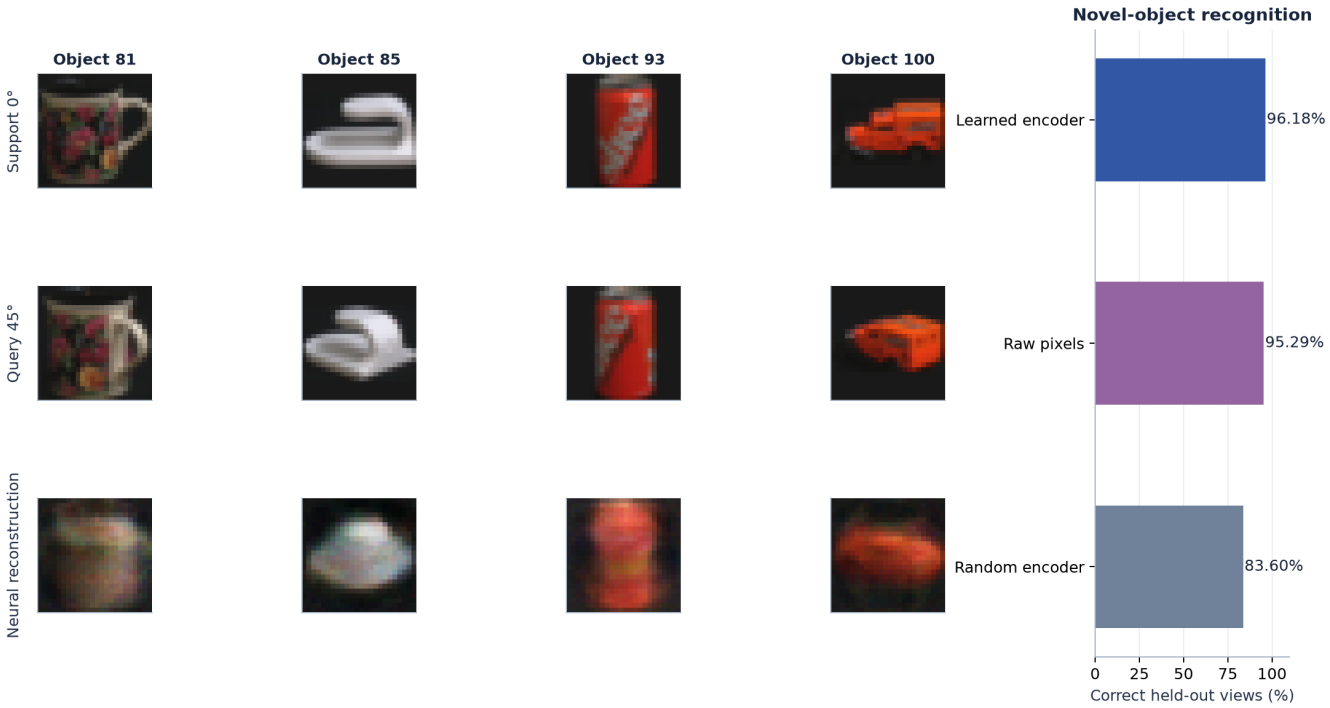
The learned model's reconstruction MSE is 0.010547 on RGB intensities scaled between 0 and 1, computed across all 1,440 held-out-object images, including support views. Actual reconstructions in the figure show the information retained at 24 x 24 resolution. Recognition accuracy does not establish photographic recall, and reconstruction MSE does not establish preservation of every task-relevant detail.

The visual arm therefore supports a bounded conclusion: a learned encoder trained on other objects can support new classifier weights derived from only four views, but a simple pixel baseline nearly matches it here. This is a transfer-of-representation result under controlled acquisition. It does not validate the symbolic writer, prove temporal memory consolidation, or demonstrate recognition of arbitrary real-world scenes.

An additional identity decision probe gives every query a separate match decision against each of the 20 possible object prototypes. We select the cosine threshold separately for learned features and raw pixels by minimizing balanced error on objects 61-80, using a 0.001 grid. On objects 81-100, learned features miss 0 of 1,360 true matches and accept 1,989 of 25,840 nonmatches (7.70%). Raw pixels miss 122 true matches (8.97%) and accept 2,260 nonmatches (8.75%). A missed match can split repeated observations of one object.

A false match can merge observations of distinct objects. These rates concern individual comparisons among correlated views. They do not measure the lifetime frequency of cluster splits or merges in a deployed stream. The false matches also illustrate why a threshold alone can be insufficient when several objects look similar. [16]

Figure 4. Real support views, query views and reconstructed images



Actual COIL-100 photographs • learned encoder, persisted classifier-weight memory

Train: objects 1–60. Validation: 61–80. Test: entirely unseen objects 81–100.
Four support views per test object become normalized classifier weight rows; 68 other angles per object are queries.

1,308 / 1,360 correct with learned features; raw pixels: 1,296 / 1,360. Difference: 12 queries.
Single training seed. Reconstructions are 24×24 model outputs; no claim of general visual understanding.

COIL-100 images are resized to 24 x 24 RGB for this experiment. Support and query examples and model reconstructions are actual artifacts, not generated illustrative objects. The comparison chart uses the fixed 1,360-query test set. Source: official COIL-100 archive and results/image-results.json. [16]

Table 4. Closed-set object recognition

Representation	Correct / queries	Accuracy	Training and inference boundary
Learned encoder	1,308 / 1,360	96.18%	Encoder trained on objects 1-60; held-out prototypes from support views.
Raw pixels	1,296 / 1,360	95.29%	Same support/query images and prototype decision rule.
Random encoder	1,137 / 1,360	83.60%	Initial untrained encoder; same architecture and prototype rule.

One training seed, 20 held-out identities, four support angles and 68 query angles per identity. The object-cluster bootstrap interval for learned minus pixels is -4.41 to +7.21 percentage points. Source: results/image-results.json.

13. Concurrent reads, updates and consolidation

The statistical experiments above concern accuracy. A separate timing check asks whether updates and saved snapshots can coexist in this small implementation. Eight short conditions combine retained-block capacities of 32 or 128 with snapshot requests every 0.5, 1 or 2 seconds, or every five committed updates. Each condition schedules 50 synthetic SET updates at 10 Hz and performs four online optimizer steps per update. There remain 64 logical keys in every condition. Block capacity is not a count of remembered entities.

Across 400 scheduled updates, all complete and none exceeds the recorded 100 ms update deadline. The largest condition-specific 99th-percentile update time is 33.43 ms. The workers write 50 snapshots in total, with a largest snapshot file of 48,026 bytes, no dropped requests, maximum observed queue length one and empty queues after the workers finish. These are observed outcomes, not deterministic upper bounds.

The snapshot worker writes a consistent copy associated with a committed memory revision. The every-five-commits trigger is a count rule. Its internal label evidence5 does not represent learned confidence or a judgment that evidence is sufficient. Snapshotting serializes state for later use. It does not independently improve recall or perform semantic consolidation.

Each condition lasts approximately five seconds and runs once on an Apple M1 Max CPU with two PyTorch threads. Startup state, operating-system scheduling and shared hardware load can affect the measurements. Tail percentiles from 50 updates are especially coarse. These runs establish that the code path executes under a modest concurrent workload, but they cannot establish a production service-level objective, sustained capacity, crash consistency under faults or superiority of one snapshot interval.

A second run uses the same eight conditions and lets the background worker combine all retained blocks in its copy of each committed revision before saving it. The foreground makes one address read per update. Across 400 reads and 50 compacted copies, every read met the 100 ms deadline. The highest condition-level reader p99 was 2.72 ms. Queue depth peaked at 1 with zero drops. Both the effective weight matrix and all 64 decoded addresses matched the original revision in every saved copy.

The foreground run uses the committed live array while a worker prepares a compacted revision for later loading. The measured consolidation adds contributions at the same weight addresses. A live system that also decides whether similar observations refer to one real object would require its own evaluation of the resulting associations. The supplementary report is results/concurrent-consolidation.json.

Table 5. Short concurrent-update and snapshot runs

Block capacity	Snapshot trigger	Updates	Update p99	Snapshots	Max snapshot bytes
32	Every 0.5 s	50	17.19 ms	9	19,790
32	Every 1.0 s	50	13.09 ms	4	19,407
32	Every 2.0 s	50	8.37 ms	2	17,557
32	Every 5 commits	50	13.43 ms	10	19,699
128	Every 0.5 s	50	22.75 ms	9	45,952
128	Every 1.0 s	50	19.18 ms	4	44,283
128	Every 2.0 s	50	33.43 ms	2	44,283
128	Every 5 commits	50	19.91 ms	10	48,026

All rows schedule 50 updates at 10 Hz, have zero observed 100 ms deadline misses, and finish with no queued or dropped snapshot requests. Conditional percentiles describe one short run each; they are not latency guarantees. Source: results/concurrent-benchmark.json.

14. Human memory motivates the questions, not the numerical claims

Human development suggests that useful behavior emerges from interactions among internal state, the body and the environment. In Thelen and colleagues' stepping experiments, adding a small load to infants' legs reduced stepping, while buoyancy increased it. Such results warn against explaining every behavioral change as a change in an internal program: the same organism can perform differently under different physical constraints. Our use of these findings is motivational, not a claim that a matrix update reproduces infant motor development. [14]

Social feedback also provides structured information. Goldstein and Schwade compared contingent and noncontingent responses to infants' babbling. The timing of caregivers' responses affected short-session changes in vocal form. This was not simply a claim that exposing infants to more sounds improves every aspect of language. The relevant design question for an adaptive system is whether feedback is attached to the operation whose quality is being assessed. The experiment here does not model speech acquisition. [15]

Human memory tasks also differ sharply in what they measure. Recognizing a studied image when shown two alternatives is different from freely reconstructing an experience. Brady and colleagues demonstrated detailed recognition after studying thousands of object images, while naturalistic camera-assisted research distinguishes the accuracy of recalled details from the completeness of experience recalled. Neither supplies a conversion between minutes of life and bytes of durable memory. [12], [13]

For this reason, all storage sizes in the present study refer to files, tensors or logged records in the implemented system. All recognition scores refer to the declared test examples. No human-equivalent capacity is inferred from the number of stored associations, and no failure of the prototype is evidence against the flexibility of human learning.

15. Relationship to prior neural-memory methods

Fast-changing weights and generated weights have long antecedents. Fast-weight models distinguish rapidly changing associations from slower learned parameters, while hypernetworks use one network to generate parameters of another. These ideas establish that a numerical inference function can be modified by a separate learned mechanism. The present system uses that general design space. It does not introduce the idea of storing experience in weights or of generating one network's weights with another. [2], [3]

Recent methods develop different versions of test-time memory. Titans adds a neural memory trained to retain context during processing. MIRAS organizes designs around memory architecture, update objective, retention and optimization rule. Nested Learning treats update processes at different frequencies as part of the system's structure. FwPKM uses sparse, locally updated memory parameters. These approaches motivate reporting what is updated and on what timescale, rather than treating every test-time update as the same algorithm. [4], [5], [6], [17]

Adapter generation is another nearby route. Text-to-LoRA generates task adapters from descriptions, and Doc-to-LoRA turns a document into an adapter for later queries. LoRA itself provides the familiar low-rank parameterization for adapting a frozen model. A rank-one memory write in this prototype is related algebraically to that family, but is not automatically equivalent in training, parameter placement or capability. Our bounded association task is not a reproduction of a document-to-language-model result. [8], [9], [11]

Direct weight programming is a particularly close recent motivation. The MIT CSAIL authors' November 2026 report describes general-purpose language models inspecting, constructing and editing weights. A subsequent preprint shows that models allowed to choose their structure often write sparse fact tables with fixed readers. These demonstrations make it necessary to state what is learned in the present system and what is supplied by an explicit interface. A readable table and a numerical array are not sufficient evidence of a new theory of memory. [1], [7]

Model editing provides a complementary warning about local success. ROME studies targeted changes to factual associations, while EngramEdit investigates updates through a separate conditional-memory mechanism. A successful individual write does not establish stable cumulative behavior. The present study consequently treats retention and update interactions as primary outcomes and uses the earlier literature to frame its contribution as a bounded empirical test. [10], [18]

The image arm is more precisely situated alongside prototype-based few-shot methods than alongside unrestricted language-model weight editing. Prototypical Networks learns a space in which class prototypes support classification of new classes. Here, the encoder objective and fixed COIL split differ from that paper's episodic method, so the implementation is a related mean-prototype baseline rather than a reproduction. [19]

Table 6. Nearby approaches and the distinction tested here

Approach	Relevant mechanism	Boundary of the present claim
Fast weights / Titans / FwPKM	Change a memory state during processing	This study evaluates a small explicit write/read interface.
Hypernetworks / generated LoRA	A learned network emits another component's parameters	We do not claim to invent parameter generation.
Direct weight programming	A general-purpose model writes or edits numerical parameters	The prototype uses trained bounded components, not unrestricted LLM reasoning.
Object representation learning	Learn a representation that transfers across images	Held-out object recognition is evaluated separately from symbolic retention.

These are functional distinctions, not rankings. Training data, scale, update rules and evaluation tasks differ. [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [17], [18], [19]

16. Storage, computation and the meaning of compression

A compact memory matrix is not the complete model. The symbolic inference array has 16 x 64=1024 float32 values, occupying 4096 bytes when materialized once. The program additionally retains its base and consolidated arrays, update factors, codec, writer, online optimizer state and replay examples. Its source and evidence logs have separate storage costs. We do not label 4096 bytes as the total cost of the learner.

In the image arm, 20 prototype rows of 32 float32 values occupy 2560 bytes. The learned encoder has 225440 parameters (901760 bytes at float32), and the decoder has 227136 parameters (908544 bytes). The 1980-parameter training classifier is included in the 454556 training-parameter count but is not required for held-out prototype recognition. Thus, small per-object weights rely on a substantially larger learned representation.

The raw-pixel prototype matrix has 20 x 1728 entries (138240 bytes at float32). Comparing it with 2560 bytes of learned prototypes illustrates a representation tradeoff, not an end-to-end 54-fold compression claim: the learned encoder must also be stored and executed. Whether that tradeoff becomes useful depends on the number of objects, reuse of the encoder and the desired accuracy.

Runtime figures describe one CPU environment with two threads. The initial codec/core training routine takes 1.546 seconds, while image training and evaluation take 7.468 seconds in the recorded runs. The image duration includes local loading of the preprocessed data but excludes initial archive download. The separate concurrent measurements include their actual online update and snapshot work. No hardware-independent

speedup or deployment latency guarantee follows.

17. Threats to validity and alternative explanations

An association task may be easier because the address is provided explicitly. That is appropriate for isolating a memory operation, but it removes entity resolution, relevance selection and natural-language parsing. Strong symbolic recall can coexist with failure on those upstream problems. The interpretation of a positive result must preserve this decomposition.

The writer can receive information about the current state that a comparator does not receive. Such an asymmetry can explain an apparent advantage. Likewise, an aggregate over the memory bank may encode useful global information even when a baseline is restricted to one slot. Every comparator should therefore be described by its available inputs as well as its parameter count. A difference in information access is not evidence of a superior optimization rule.

The competence gate excludes the current event from its preceding-32-example probe, avoiding direct use of that new label for the gate decision. Those preceding examples remain supervised replay data. Passing the probe therefore measures familiarity with recent training material, not out-of-distribution competence. Moreover, the eventual bank update learns from the current label before the main readout. Both boundaries must remain visible when interpreting the small observed gate differences.

The controlled image collection has additional shortcuts. Background, lighting, pose regularity and repeated acquisition conditions can support recognition in ways that do not transfer to natural scenes. Withholding object identities is important, but independent environments and capture devices would be required for broader claims. A decoder may also favor common visual structure rather than preserve the exact evidence needed for a decision.

Only one symbolic kernel and one image encoder are trained. The 12 stream trials per condition vary event generation and replay randomness, not the pretrained parameters. The image bootstrap varies object membership with the fitted model held fixed. Replication across training seeds and independently selected splits is required before making a stable generalization claim. Retaining every condition and paired failure case is more informative than selecting the cleanest learning curve.

18. Reproduction and provenance

The reproducibility unit comprises source code, configuration, ordered streams, the external image archive, learned checkpoints and machine-readable results. A hash identifies bytes. It does not alone establish how those bytes were produced. The supplied training, stream, control, image and audit scripts specify that procedure, and the release manifest binds the resulting artifacts. The fixed source reference accompanies this paper separately from any later manuscript-only revision.

COIL-100 is obtained from the official Columbia distribution. The recorded ZIP is 130,688,843 bytes with SHA-256 fcb4b8b01c0778f416d8d7a09f45082c4f7075cc74743dcddd5c44307bd4afe7. Preprocessing, object IDs and support/query angles are retained in the result record. External data remain distinct from authored

synthetic streams. Providing a manifest does not transfer ownership of the images. [16]

The main trial audit finds unchanged frozen-component hashes and matching predictions after in-memory state restoration in all 84 trials. A stronger, separate audit starts new Python processes, loads the saved symbolic or image checkpoint and recomputes outputs. Symbolic logits and predictions match across all 64 addresses, while image logits and predictions match for 20 unlabeled probe images. The workers do not load the source event list, oracle labels or result metrics. This demonstrates that the saved numerical state supports those outputs without a fact dictionary in the read path.

Fresh-process output replay is narrower than resuming an identical training trajectory. The inference checkpoint need not include the optimizer and replay-buffer state required to reproduce future online updates exactly. Nor does replay prove that remembered values are true. A small command-line example separately exercises setting a value to 12, updating it to 4, retracting it to UNKNOWN and restoring an earlier saved state to recover 12. Its role is an inspectable behavior check, not another accuracy benchmark.

The paper tables and three empirical graphics derive from the locked result files. They include failed writes, raw-pixel strengths and small or absent improvements. A later change to a checkpoint, a stream, a threshold or the preprocessing warrants a separately identified result record rather than silent replacement.

19. What this experiment leaves unresolved

The prototype does not determine which events from an unconstrained day are important. It does not discover entities across arbitrary text and video, judge source credibility, or implement a human autobiographical narrative. Those tasks would require additional perception, selection and temporal-reasoning mechanisms. The reported array is a memory component, not a complete memory-using agent.

Higher-order adaptation also remains bounded. A residual writer that learns during a stream is not a general system that rewrites its own architecture or invents a new learning algorithm. The gate controls a permitted update process specified by the experimenter. Open-ended self-modification would require different evaluation and control mechanisms.

These limits are useful design boundaries. A subsequent study could add noisy addressing, contradictory sources, explicit historical queries, out-of-distribution visual settings or resource-constrained module selection. Each addition should create a measurable question and a corresponding control. Scaling the same array or increasing the number of examples would not, by itself, answer those different questions.

20. A falsifiable route to programmable memory

A numerical weight array can provide an inspectable, saved and reloadable memory state, and learned writers can operate through that interface. In this prototype, hard address policies determine a clear retention-versus-correction tradeoff. Extra adapter capacity makes only small differences under the tested schedules. The SET-only exact-codec baseline matches the learned writer, so a learned update generator is not necessary for that task.

The independent image experiment demonstrates transfer from a learned representation to classifier weights constructed for new objects. Its 96.18% accuracy is encouraging within the controlled task, but the 95.29% pixel result and wide object-level interval bound the interpretation. These observations justify more demanding tests, not a claim of a new general memory architecture.

The next decisive experiments would remove supplied addresses, separate old-column and global-state contributions, score before current-label updates, repeat across independently trained kernels, and test shifted acquisition conditions. Each should retain an exact arithmetic or retrieval baseline where possible. Programmable neural memory becomes scientifically useful when its numerical operations, information access and failures remain as inspectable as its successes.

References

[1] Sparks of Weight-Space Programming: LLMs can natively read & write neural network weights. Takehi, Teoh, Zhang and Khattab. Authors' research blog, November 2, 2026; early direct-weight programming experiments, not a general memory benchmark. Accessed 2026-11-20.

[2] HyperNetworks. Ha, Dai and Le. First submitted September 27, 2016; generating network weights with another network. Accessed 2026-11-20.

[3] Using Fast Weights to Attend to the Recent Past. Ba, Hinton, Mnih, Leibo and Ionescu. 2016; temporary associations in rapidly changing weights. Accessed 2026-11-20.

[4] Titans: Learning to Memorize at Test Time. Behrouz, Zhong and Mirrokni. Submitted December 31, 2024; neural long-term memory updated during processing. Accessed 2026-11-20.

[5] It's All Connected: A Journey Through Test-Time Memorization, Attentional Bias, Retention, and Online Optimization. Behrouz, Razaviyayn, Zhong and Mirrokni. April 17, 2025; MIRAS design framework. Accessed 2026-11-20.

[6] Introducing Nested Learning: A new ML paradigm for continual learning. Google Research, November 7, 2025. Author summary of Nested Learning and HOPE; multiple update frequencies. Accessed 2026-11-20.

[7] Weight-Space Programming Doesn't Need Transformers. Ok and Lee. November 17, 2026, work-in-progress preprint; sparse authored memories, fixed readers and scope limits. Accessed 2026-11-20.

[8] Doc-to-LoRA: Learning to Instantly Internalize Contexts. Charakorn, Cetin, Uesaka and Lange. February 13, 2026; learned document-to-adapter generation. Accessed 2026-11-20.

[9] Text-to-LoRA: Instant Transformer Adaption. Charakorn, Cetin, Tang and Lange. ICML 2025; task descriptions used to generate adapters. Accessed 2026-11-20.

[10] EngramEdit: Decoupled Knowledge Updates in LLMs through Conditional Memory. Cai and colleagues. October 7, 2026 preprint; updating a separate conditional-memory mechanism. Accessed 2026-11-20.

[11] LoRA: Low-Rank Adaptation of Large Language Models. Hu and colleagues. 2021; trainable low-rank parameterizations around a frozen pretrained model. Accessed 2026-11-20.

[12] Visual long-term memory has a massive storage capacity for object details. Brady, Konkle, Alvarez and Oliva. PNAS 105, 14325-14329, 2008; detailed two-alternative object recognition, not free recall of daily life. Accessed 2026-11-20.

[13] Accuracy and completeness of autobiographical memory: evidence from a wearable camera study. Finley and Brewer. Memory 32(8), 1012-1042, 2024; separates recalled-detail accuracy from an upper-bound completeness estimate. Accessed 2026-11-20.

[14] The Relationship between Physical Growth and a Newborn Reflex. Thelen, Fisher and Ridley-Johnson. Infant Behavior and Development 7, 479-493, 1984; body-load and buoyancy manipulations of infant stepping. Accessed 2026-11-20.

[15] Social Feedback to Infants' Babbling Facilitates Rapid Phonological Learning. Goldstein and Schwade. Psychological Science 19(5), 515-523, 2008; contingent and yoked caregiver feedback. Accessed 2026-11-20.

[16] Columbia Object Image Library (COIL-100). Nene, Nayar and Murase. Columbia Technical Report CUCS-006-96, February 1996; official processed object-image archive and technical documentation. Accessed 2026-11-20.

[17] Fast-weight Product Key Memory. Zhao and Jones. Submitted January 2, revised February 22, 2026; sparse inference-time gradient updates to memory parameters. Accessed 2026-11-20.

[18] Locating and Editing Factual Associations in GPT. Meng, Bau, Andonian and Belinkov. NeurIPS 2022; targeted rank-one factual model editing and specificity evaluation. Accessed 2026-11-20.

[19] Prototypical Networks for Few-shot Learning. Snell, Swersky and Zemel. 2017; learned representations and class prototypes for new-class recognition. Related context, not a claim to reproduce its training protocol. Accessed 2026-11-20.

Reproducibility record

Private review materials: https://github.com/RevanthM/programmable-neural-memory | Research snapshot: 05ce418e645d0096d0f74d13edec135b03ad0ba5