DELTANET-PC: TEST-TIME MEMORY WITHOUT BACKPROPAGATION THROUGH TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time memorization of sequential information is a foundational capability for any intelligent system that needs to rapidly store and retrieve data, as demonstrated by the attention mechanism in large language models. However, Transformers and linear attention variants are trained via backpropagation through time (BPTT), which has drawbacks such as its enormous memory requirements. Predictive coding (PC)—an algorithm that optimizes a prediction-error energy function with local updates—offers a compelling solution but still lacks test-time memory for sequential data. We develop a solution to this problem by treating the per-step memory update $\Delta \boldsymbol{S}_t$ as a latent variable, as opposed to a slow weight to be learned. By taking the log probability of a standard PC-based probabilistic model, we derive a PC-native solution for verbatim sequence memory, and relate our update rule to common choices in the linear attention literature (e.g., DeltaNet). We present an extensive set of empirical tests to probe our system's properties, including its scaling capacity, memory footprint, and ability to infer a context from evidence (e.g., bank (river) vs. bank (money)). We show how DeltaNet-PC can be used for storing single sequences of data, multiple sequences of data, and key-value associative memory pairs with perfect recall accuracy.

1 INTRODUCTION

Test-time memorization of sequential information is a foundational capability for intelligent biological and artificial systems. Much of the success of modern large language models (LLMs) can be attributed to test-time adaptivity (Von Oswald et al., 2023), in particular the self-attention mechanism found in Transformers (Vaswani et al., 2017), and more recently the linear attention mechanism found in hybrid sequence models (Katharopoulos et al., 2020; Schlag et al., 2021; Sun et al., 2023; Yang et al., 2024a; Kimi-Team et al., 2026).

However, these systems are trained using backpropagation through time (BPTT) (Werbos, 1990), which has some well-known drawbacks: memory that grows with the sequence length (Marschall et al., 2020), and the inability to learn in streaming scenarios without locking the system to wait for a backward pass to finish (Ororbia, 2023). This motivates an algorithm that relies on updates that are local in both space and time.

Predictive coding is one of the most successful backpropagation alternatives (Rao & Ballard, 1999; Whittington & Bogacz, 2017; Pinchetti et al., 2025), and its parameter updates are local in space (i.e., Hebbian, only requiring pre- and postsynaptic activity) (Bogacz, 2017). Prior PC-based works have developed methods for sequential modeling (Tang et al., 2023a) and continual updates (Yoo & Wood, 2022), though they lack what is arguably one of the most important parts of a Transformer: some notion of test-time memorization for sequential data. Self-attention, however, uses an unbounded key-value cache—extremely taxing for computers and a resource that brains seem to succeed without—which motivates our focus on a fixed-size memory.

In brief, we are storing pairs: a key $\boldsymbol{k}_t$ and a value $\boldsymbol{v}_t$, which are bound together in a memory $\boldsymbol{S}$ at time t. We connect predictive coding to the linear attention literature (e.g., DeltaNet (Schlag et al., 2021)) through a key insight: once you treat matrix-valued updates $\Delta \boldsymbol{S}_t$ to the memory as latent variables, as opposed to slowly learned parameters, predictive coding natively provides linear attention machinery.

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.

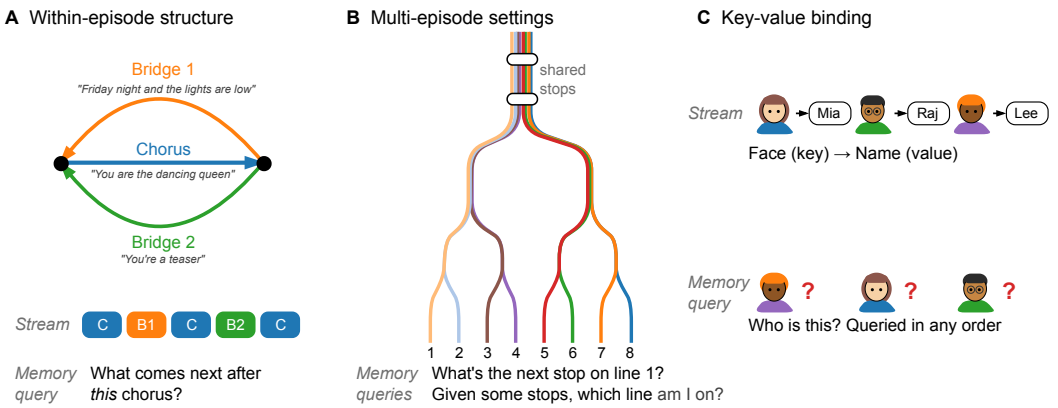


Figure 1: We evaluate our system on several sequential memory tasks. **A:** Within-episode structure tests the model's ability to handle non-Markovian dynamics in a single sequence. **B:** Multi-episode settings test the system's ability to memorize and reason about multiple sequences, particularly with highly overlapping subsequences. **C:** Key-value binding tests whether the system can retrieve a value when cued with a key.

Our tasks include three episodic memory scenarios, such as memorizing a song, where the chorus is followed by different bridges, requiring non-Markovian memory (Figure 1A), and putting names to faces (key-value binding, Figure 1C). Our system also faithfully stores memories with a high degree of overlap, like a metro map, and can infer the line given some observed stops (Figure 1B). We evaluate on community benchmarks such as MQAR (Arora et al., 2023), and image-based storage and retrieval tasks.

Mainstream BPTT architectures (e.g., Transformers, SSMs, etc.) solve verbatim recall easily and we claim no accuracy advantage over them, but they violate our locality and/or fixed-size memory constraints. We compare to fixed memory capacity baselines (i.e., not Transformers, whose key-value cache is unbounded), including a BPTT-trained DeltaNet model (Yang et al., 2024b), and several outer-product associative memory networks.

Our contributions are as follows: (i) a sequential PC model that binds new memories at test time, which uses updates that are local in both space and time and has a fixed-size memory footprint, (ii) a write rule derived from treating the memory update as an inferred latent variable, which can be related to common choices in the linear attention literature, and (iii) empirical results demonstrating our system's ability to match the perfect recall accuracy of BPTT-trained linear attention models at a fraction of the training memory cost, while overcoming the limitations of models that can't learn (e.g., static outer-product models).

2 Background & Related Work

2.1 Predictive Coding

Predictive coding (PC) is an algorithm for *inference* and *learning* that optimizes a prediction-error energy function (Rao & Ballard, 1999; Whittington & Bogacz, 2017; Pinchetti et al., 2025). In its most common form, each layer of the network has a latent variable $\boldsymbol{z}$ that is optimized to minimize a global energy function—a process called inference (also referred to algorithmically as settling). Once the latent variables have settled, the weights are updated to minimize the same energy function—learning. A standard PC assumption is that each layer's latent is a Gaussian random variable with a mean given by the output of the previous layer. This allows one to write down a probabilistic model and derive a resulting energy function from the log probability. For example, the probability distribution, and resulting energy function, for an ℓ-layer feedforward network with latents $(\boldsymbol{z}_0 = \boldsymbol{x}), \boldsymbol{z}_1, \dots, \boldsymbol{z}_\ell$,

weights $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\ell$, and activation function g are

$$p(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell) = \prod_{i=1}^{\ell} p(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-1}), \quad p(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-1}) = \mathcal{N}(\boldsymbol{z}_i; \boldsymbol{W}_i g(\boldsymbol{z}_{i-1}), \mathbf{I}),$$

$$E(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell) = -\log p(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell) = \frac{1}{2} \sum_{i=1}^{\ell} \|\boldsymbol{z}_i - \boldsymbol{W}_i g(\boldsymbol{z}_{i-1})\|^2. \tag{1}$$

Tang et al. (2023a) developed tPC for sequential modeling, which can memorize *training* data in the slow weights of their network, but they do not treat the per-step update as a latent variable so their model does not perform test-time memorization. Yoo & Wood (2022) developed BayesPCN, a PC-based continual learning system, but they do not address sequential data.

2.2 Linear attention

Linear attention is a mechanism for performing attention with a constant memory footprint (Katharopoulos et al., 2020), in contrast to the key-value cache that grows linearly in the sequence length for standard attention (Vaswani et al., 2017). It was initially conceptualized as a sum of outer product memories (Katharopoulos et al., 2020) given by $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$, where $\boldsymbol{S}$ is a linear memory, $\boldsymbol{v}_t$ is a value, and $\boldsymbol{k}_t$ is a key. The readout of the memory is given by $\boldsymbol{S}\boldsymbol{q}_t$, where $\boldsymbol{q}_t$ is a query. This style of memory has fixed capacity—proportional to the dimensionality of the key space—and is prone to interference, which motivated DeltaNet (Schlag et al., 2021), with a learnable write gate β_t and an erase term to reduce interference: $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \beta_t(\boldsymbol{v}_t - \boldsymbol{S}_{t-1}\boldsymbol{k}_t)\boldsymbol{k}_t^\top$. This is one gradient step on $\mathcal{L}_t(\boldsymbol{S}) = \frac{1}{2}\|\boldsymbol{v}_t - \boldsymbol{S}\boldsymbol{k}_t\|^2$ from $\boldsymbol{S}_{t-1}$ with step size β_t—the least mean squares (LMS) rule (Widrow & Hoff, 1960). With L2-normalized keys (Yang et al., 2024b) it is also normalized LMS (NLMS), which divides the step by $\|\boldsymbol{k}_t\|^2$.

More recent work has continued to iterate on this core idea, including the use of forgetting mechanisms (Yang et al., 2024a), more complex key-value bindings (Sun et al., 2024), and the use of recursive least squares (RLS) for key-value bindings (Von Oswald et al., 2024). Table 1 situates DeltaNet-PC in the context of prior works.

Table 1: Situating DeltaNet-PC. Memory is *one layer's* footprint; L the sequence length and D the latent space dimensionality. Local: weight updates use only pre- and postsynaptic activity (space) and only the current step (time). Sequential: stores ordered sequences. Trained: the system learns the keys and values, rather than given as raw data. Test-time: an episode is stored in one pass with no gradient steps on the weights. See Appendix A for an extended related work section and Appendix B.2, which relates our work to linear attention.

Model	Write rule	Training memory	Inference memory	Local in space	Local in time	Sequential data	Trained keys, values	Test-time write
Transformer (Vaswani et al., 2017)	append (KV cache)	LD	LD	×	×	✓	✓	✓
Linear attention (Katharopoulos et al., 2020)	outer product	LD	D^2	×	×	✓	✓	✓
DeltaNet (Schlag et al., 2021; Yang et al., 2024b)	delta rule (LMS/NLMS)	LD	D^2	×	×	✓	✓	✓
Mesa layer (Von Oswald et al., 2024; 2025)	RLS	LD	D^2	×	×	✓	✓	✓
Lin. assoc. memory (Kohonen, 1972; 1974)	outer product / RLS	–	D^2	✓	–	×	×	✓
tPC (Tang et al., 2023a)	delta rule (LMS)	D^2	D^2	✓	✓	✓	✓	×
BayesPCN (Yoo & Wood, 2022)	RLS (Bayesian)	–	D^2	✓	–	×	×	✓
DeltaNet-PC (ours)	RLS (inferred $\Delta\boldsymbol{S}_t$)	D^2	D^2	✓	✓	✓	✓	✓

3 Methods

We first define our probabilistic model and then derive its resulting energy function. We then describe the complete architecture and training procedure.

3.1 Memory module

Joint distribution. We specify a standard PC-native probabilistic graphical model, which results in the memory write—recursive least squares (Plackett, 1950) when the memory update $\Delta\boldsymbol{S}_t$ is a fully settled latent. We assume a linear memory store $\boldsymbol{S} \in \mathbb{R}^{D \times D}$ and we would like this memory to

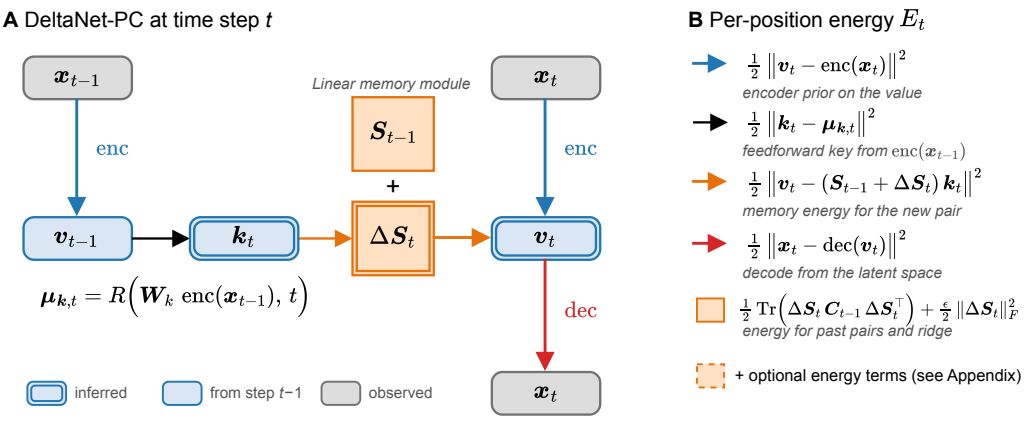


Figure 2: **A:** DeltaNet-PC architecture. **B:** The per-position energy E_t. See Appendix B.4 for the full energy function.

encode transitions at time t from keys to values such that $\boldsymbol{v}_t \approx \boldsymbol{S}_t\boldsymbol{k}_t$ for a key-value pair $(\boldsymbol{k}_t, \boldsymbol{v}_t)$. Importantly, online updates to the memory $\boldsymbol{S}$ should disturb what past keys retrieve as little as possible. Our joint distribution restricted to the memory model over a sequence of length L is given by

$$p(\boldsymbol{k}_{1:L}, \boldsymbol{v}_{1:L}, \boldsymbol{S}) = p(\boldsymbol{S}) \prod_{t=1}^{L} p(\boldsymbol{v}_t \mid \boldsymbol{k}_t, \boldsymbol{S}) p(\boldsymbol{k}_t \mid \cdot), \tag{2}$$

$$p(\boldsymbol{S}) = \mathcal{N}(\boldsymbol{S}; \boldsymbol{0}, \mathbf{I}/\epsilon), \qquad p(\boldsymbol{v}_t \mid \boldsymbol{k}_t, \boldsymbol{S}) = \mathcal{N}\big(\boldsymbol{v}_t; \boldsymbol{S}\boldsymbol{k}_t, \mathbf{I}\big), \qquad p(\boldsymbol{k}_t \mid \cdot) = \mathcal{N}\big(\boldsymbol{k}_t; \cdot, \mathbf{I}\big).$$

This allows us to make the standard predictive coding assumption that latent variables are Gaussian distributed with a mean given by the output of the previous layer.[1]

Energy function. Taking the negative log of Equation 2 and restricting our attention to terms involving the memory $\boldsymbol{S}$ gives the energy function for the memory module:

$$E(\boldsymbol{S}) = -\log p(\boldsymbol{k}_{1:L}, \boldsymbol{v}_{1:L}, \boldsymbol{S}) = \frac{\epsilon}{2}\|\boldsymbol{S}\|_F^2 + \sum_{t=1}^{L} \tfrac{1}{2}\big\|\boldsymbol{v}_t - \boldsymbol{S}\boldsymbol{k}_t\big\|^2 + \text{const.}, \tag{3}$$

which is equivalent to ridge regression on all L pairs $(\boldsymbol{k}_t, \boldsymbol{v}_t)$. However, as written, this is a batched objective that requires access to all L pairs—incompatible with our online learning scenario, where data points are observed one-at-a-time. Furthermore, our fixed-size memory footprint precludes storing the full dataset for offline learning. Past latents $(\boldsymbol{k}_i, \boldsymbol{v}_i)_{i<t}$ are committed once they have been inferred and are not revisited. We define a partial energy function that accounts for all observed data up to time t

$$E_t(\boldsymbol{S}) = E_{t-1}(\boldsymbol{S}) + \tfrac{1}{2}\|\boldsymbol{v}_t - \boldsymbol{S}\boldsymbol{k}_t\|^2, \tag{4}$$

$$E_0(\boldsymbol{S}) = \frac{\epsilon}{2}\|\boldsymbol{S}\|_F^2, \qquad E_L(\boldsymbol{S}) = E(\boldsymbol{S}),$$

$$\boldsymbol{S}_t := \arg\min_{\boldsymbol{S}} E_t(\boldsymbol{S}), \qquad \boldsymbol{S}_0 := \boldsymbol{0}, \qquad \Delta\boldsymbol{S}_t := \boldsymbol{S}_t - \boldsymbol{S}_{t-1},$$

which makes clear that at each time step, we would like to update the memory $\boldsymbol{S}_{t-1}$ by an amount $\Delta\boldsymbol{S}_t$ that minimizes the energy function $E_t(\boldsymbol{S})$.

We now express E_t as a function of the latent variable $\Delta\boldsymbol{S}_t$ with the past summarized in $(\boldsymbol{S}_{t-1}, \boldsymbol{C}_{t-1})$, where $\boldsymbol{C}_{t-1} = \sum_{i<t} \boldsymbol{k}_i\boldsymbol{k}_i^\top$ is a summary of the past keys, so that we can maintain a fixed memory footprint that only involves two $D \times D$ matrices that don't grow with the sequence length.

Lemma 1 Taylor expands E_{t-1} about $\boldsymbol{S}_{t-1}$. Let $\boldsymbol{r}_t = \boldsymbol{v}_t - \boldsymbol{S}_{t-1}\boldsymbol{k}_t$.

[1] See Section 3.2 for a definition of $\boldsymbol{k}_t$ and the full joint distribution.

Lemma 1. *For any* Δ

$$E_t(\boldsymbol{S}_{t-1}+\Delta) = E_{t-1}(\boldsymbol{S}_{t-1}) + \langle \nabla E_{t-1}(\boldsymbol{S}_{t-1}), \Delta\rangle + \tfrac{1}{2}\operatorname{Tr}(\Delta(\epsilon\mathbf{I}+\boldsymbol{C}_{t-1})\Delta^\top) + \tfrac{1}{2}\|\boldsymbol{r}_t - \Delta\boldsymbol{k}_t\|^2.$$

Appendix B gives the proof and the calculations that lead to the closed form

$$E_t(\boldsymbol{S}_{t-1}+\Delta) = \underbrace{\frac{1}{2}\|\boldsymbol{r}_t - \Delta\boldsymbol{k}_t\|^2}_{\text{new pair}} + \underbrace{\frac{1}{2}\operatorname{Tr}(\Delta\boldsymbol{C}_{t-1}\Delta^\top)}_{\text{stored pairs}} + \underbrace{\frac{\epsilon}{2}\|\Delta\|_F^2}_{\text{ridge}} + \text{const.} \tag{5}$$

Recall that $\boldsymbol{S}_t$ is defined as the minimizer of $E_t(\boldsymbol{S})$, which means that $\Delta\boldsymbol{S}_t = \boldsymbol{S}_t - \boldsymbol{S}_{t-1}$ is the minimizer of $E_t(\boldsymbol{S}_{t-1}+\Delta)$. Notably, we are treating $\Delta\boldsymbol{S}_t$ as a latent variable, which means that we seek the *minimizer* of $E_t(\boldsymbol{S}_{t-1}+\Delta)$ with respect to Δ, not simply a small gradient step on the energy function, which would be the case if we were treating $\boldsymbol{S}$ as a learned weight. So we set the gradient of $E_t(\boldsymbol{S}_{t-1}+\Delta)$ with respect to Δ to zero, which gives the update

$$\Delta\boldsymbol{S}_t = \boldsymbol{r}_t\boldsymbol{k}_t^\top(\boldsymbol{k}_t\boldsymbol{k}_t^\top + \boldsymbol{A}_{t-1})^{-1} \tag{6}$$

where $\boldsymbol{A}_{t-1} = \boldsymbol{C}_{t-1} + \epsilon\mathbf{I}$. Derivations are provided in Appendix B.

Online updates. Let us denote $\boldsymbol{P} := \boldsymbol{A}^{-1}$, $s_t := \boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}\boldsymbol{k}_t$, and $g_t := \frac{1}{1+s_t}$. In Appendix B we show how at each time step we update $\boldsymbol{S}_t$ and $\boldsymbol{P}_t$ as

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + g_t\boldsymbol{r}_t\left(\boldsymbol{P}_{t-1}\boldsymbol{k}_t\right)^\top, \qquad \boldsymbol{S}_0 = \mathbf{0} \tag{7}$$

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} - g_t(\boldsymbol{P}_{t-1}\boldsymbol{k}_t)(\boldsymbol{P}_{t-1}\boldsymbol{k}_t)^\top, \qquad \boldsymbol{P}_0 = \mathbf{I}/\epsilon. \tag{8}$$

These updates are equivalent to the recursive least squares (RLS) algorithm (Plackett, 1950). Importantly, this update rule no longer requires an inverse to be computed at every time step, which makes it feasible for online learning scenarios. It only stores $\boldsymbol{S}_{t-1}$ and $\boldsymbol{P}_{t-1}$, two $D\times D$ matrices that do not grow with the sequence length L, so it complies with our fixed-size memory footprint.

Interpretation. Equations 5 and 7 can be interpreted relative to common choices that have been made in the linear attention literature, in particular (i) the outer-product form found in (Katharopoulos et al., 2020), (ii) the erase term found in (Schlag et al., 2021), (iii) the write strength (e.g., the gate β_t that Schlag et al. (2021) must learn), and (iv) the least-squares objective over all past pairs, which the Mesa layer solves with RLS (Von Oswald et al., 2024), all of which follow from our energy function. See Appendix B.2 for a complete mathematical mapping.

3.2 DeltaNet-PC architecture

The memory module is embedded in the following architecture (Figure 2):

$$p(\boldsymbol{x}_{0:L}, \boldsymbol{v}_{1:L}, \boldsymbol{k}_{1:L}, \boldsymbol{S}) \propto p(\boldsymbol{S})\prod_{t=1}^{L} p(\boldsymbol{x}_t \mid \boldsymbol{v}_t)\, p(\boldsymbol{v}_t \mid \boldsymbol{k}_t, \boldsymbol{S})\, p(\boldsymbol{v}_t \mid \boldsymbol{x}_t)\, p(\boldsymbol{k}_t \mid \boldsymbol{x}_{t-1}), \tag{9}$$

$$p(\boldsymbol{x}_t \mid \boldsymbol{v}_t) = \mathcal{N}\big(\boldsymbol{x}_t;\, \mathrm{dec}(\boldsymbol{v}_t),\, \mathbf{I}\big), \qquad p(\boldsymbol{v}_t \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{v}_t;\, \mathrm{enc}(\boldsymbol{x}_t), \mathbf{I}),$$

$$p(\boldsymbol{k}_t \mid \boldsymbol{x}_{t-1}) = \mathcal{N}\big(\boldsymbol{k}_t;\, R(\boldsymbol{W}_k\, \mathrm{enc}(\boldsymbol{x}_{t-1}), t),\, \mathbf{I}\big).$$

We use an encoder $\mathrm{enc}(\boldsymbol{x})$ to map observations $\boldsymbol{x}\in\mathbb{R}^{d_{\text{obs}}}$ to a latent space, which is parameterized by a linear layer $\boldsymbol{W}_{\text{enc}}\in\mathbb{R}^{D\times d_{\text{obs}}}$ with no bias term. The decoder $\mathrm{dec}(\boldsymbol{v})$ maps back to observations. We use LayerNorm at the input to the decoder as our activation function, writing $g(\cdot) := \mathrm{LayerNorm}(\cdot)$. $\mathrm{dec}(\boldsymbol{v}_t) = \boldsymbol{W}_{\text{dec}}g(\boldsymbol{v}_t)$ with $\boldsymbol{W}_{\text{dec}}\in\mathbb{R}^{d_{\text{obs}}\times D}$. The key map is parameterized by a linear layer $\boldsymbol{W}_k\in\mathbb{R}^{D\times D}$. A rotary position code $R(\cdot,t)$ with learned per-band frequencies (rotary positional encoding (RoPE) at initialization (Su et al., 2023)) is used to represent the position of the key in the sequence.[2] We represent context in multi-episode settings with $R(\cdot,t,\boldsymbol{c})$, where $\boldsymbol{c}$ is the context (see Appendix D.1). In our model, queries are defined as keys $\boldsymbol{q}_t = \boldsymbol{k}_t$.[3]

[2] Crucially, we do not hardcode any inductive bias about the task via RoPE frequencies—the model must *learn* whether ordering matters for the task (e.g., single episode tasks) or not (e.g., key-value binding tasks). MQAR is the one exception, where RoPE is off for a direct comparison with the DeltaNet baseline.

[3] $\boldsymbol{v}_t$ receives two Gaussian predictions so the joint is a product of experts and is defined up to a normalization constant that does not affect inference.

Note that whenever we define a Gaussian-distributed variable like the ones above, there is a corresponding energy term of the form $\frac{1}{2}\|\text{variable} - \text{mean}\|^2$ that is added to the total energy of the system. Figure 2B displays the per-position energy terms and we specify the complete energy function in Appendix B.4.

While we use a single linear layer for $\text{enc}(\boldsymbol{x})$ and $\text{dec}(\boldsymbol{v})$ in our experiments, the formulation extends straightforwardly to multi-layer encoders and decoders with interior PC latents, and other layer types (e.g., CNNs).

3.3 Training procedure

We provide a complete training procedure in Algorithm 1 and summarize the key steps here.

Initialization. We initialize $\boldsymbol{k}_t \leftarrow R(\boldsymbol{W}_k \,\text{enc}(\boldsymbol{x}_{t-1}), t)$ and $\boldsymbol{v}_t \leftarrow \text{enc}(\boldsymbol{x}_t)$ from feedforward passes.

Inference. The memory write phase uses Equation 7 to update the memory $\boldsymbol{S}_{t-1}$ to $\boldsymbol{S}_t$ given the current key-value pair $(\boldsymbol{k}_t, \boldsymbol{v}_t)$. When the task queries the memory, $\boldsymbol{k}_t$ and $\boldsymbol{v}_t$ are settled with the memory held fixed so that the rest of the system's parameters in R, $\boldsymbol{W}_k$, enc and dec can learn. Settling is local in time and depends only on information available at step t, with no unrolled computation graph across positions.

Learning. After settling at each position t, we compute the gradient of our energy function with respect to the parameters and accumulate the gradients into a running mean. After the final position L, we update the parameters once per sequence.

4 Experiments

We evaluate DeltaNet-PC on three episodic memory tasks (Figure 1 and Appendix C). Training details are in Appendix D and hyperparameters are in Appendix E.

Evaluation. Examples are drawn from a mix of token-based tasks and image-based tasks. Each task has an event stream, where events are either observations or queries. Observations fill the memory. Queries probe what the system has learned and we evaluate the system on these events. We report token-level *recall accuracy*—the fraction of correctly predicted tokens, averaged over the number of queries and test sequences. Appendices D.4 and F give evaluation details and ablations.

4.1 Single-episode sequential memory with non-Markovian dynamics

The task is to store a single sequence in memory and then recall it, even when the sequence has repeated elements, i.e., non-Markovian dynamics (see Figure 3A). Figure 1A shows an example of non-Markovian dynamics in a song, where the chorus repeats and is followed by two distinct bridges. The test set for this task is completely disjoint from the training set so we are testing the model's ability to generalize to unseen sequences at test time. Note that recall is autoregressive and *not* teacher-forced: each decoded token is re-encoded to form the next key, so the system must rely on its memory to recall the whole sequence correctly.

Figure 3 demonstrates our system's scaling properties. The dimensionality of the memory module governs the memory capacity. In panel B, we see that when the sequence length L is less than or equal to the hidden dimension D, the memory can recall sequences with perfect accuracy, consistent with the number of orthogonal directions in the latent space. However, when $L > D$ (e.g., $L = 512$ and $D = 256$), the memory is overloaded and we observe a graceful degradation in recall accuracy. Panel C shows that simply scaling the hidden dimension D solves this problem. We tested our system up to $L = D = 1{,}024$ and it maintains perfect recall accuracy as we scale.

We compare three popular write rules from the associative memory and linear attention literature in trained and untrained settings (Table 2), where untrained means all learned parameters are frozen at initialization. Untrained associative memory systems (Widrow & Hoff, 1960; Kohonen, 1972; 1974) are insufficient for performing this task. The write rule also has a large impact. In the trained setting,

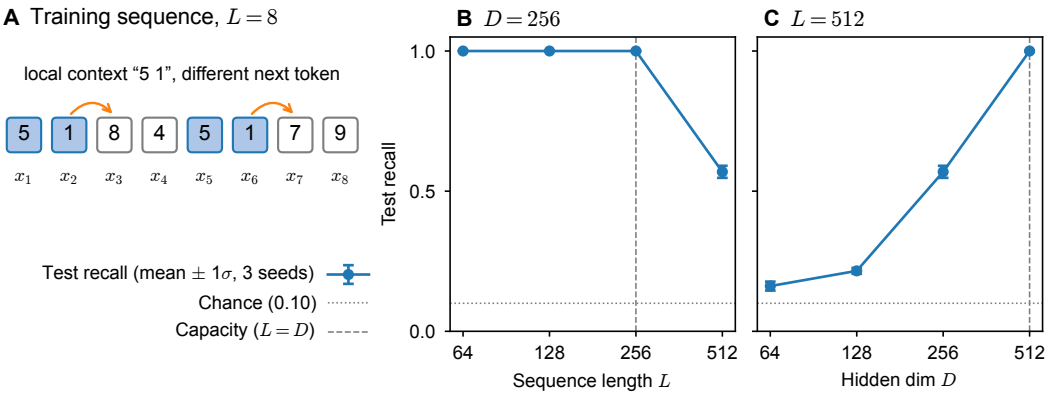


Figure 3: **A:** a sample from our single episode sequential memory task that illustrates non-Markovian dynamics. **B:** when $L \leq D$, the memory can recall sequences with perfect accuracy. **C:** scaling D increases the memory's capacity allowing it to store more memories.

Table 2: Test recall accuracy under various write rules of trained and untrained systems. Sequential recall at $L = D = 256$, chance 0.10, 3 seeds, mean $\pm 1\sigma$. Untrained systems cannot solve the task and even in trained systems, the write rule matters: the recursive least squares update of Eq. 7 is the only one that achieves perfect recall.

Write rule	Untrained	Trained
Outer product, $\boldsymbol{S} \leftarrow \boldsymbol{S} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$ (Kohonen, 1972; Katharopoulos et al., 2020)	0.13 ± 0.00	0.42 ± 0.05
Delta rule (NLMS), $\boldsymbol{S} \leftarrow \boldsymbol{S} + (\boldsymbol{v}_t - \boldsymbol{S}\boldsymbol{k}_t)\boldsymbol{k}_t^\top / \|\boldsymbol{k}_t\|^2$ (Widrow & Hoff, 1960; Schlag et al., 2021)	0.13 ± 0.00	0.51 ± 0.02
Recursive least squares, Equation 7 (Kohonen, 1974; Von Oswald et al., 2024)	0.29 ± 0.01	$\mathbf{1.00 \pm 0.00}$

we swap the write rule at test time to evaluate its effect. The last row of Table 2 shows that RLS performs the best.[4]

4.2 Multi-episode sequential memory

Here the task is to store multiple context-dependent sequences in the memory at one time, where the episodes share overlapping segments. For example, consider two episodes of one data example

$$\text{Episode 1: } a, \boldsymbol{b}, \boldsymbol{c}, d, e$$
$$\text{Episode 2: } x, \boldsymbol{b}, \boldsymbol{c}, f, g$$

where $\boldsymbol{b}, \boldsymbol{c}$ is the overlapping segment with exactly the same contents at exactly the same positions. In the multi-episode setting, position and content are insufficient for recall because the next token depends on the episode context. Recall that in the multi-episode setting, keys are a function of both position t and context $\boldsymbol{c}$, $\boldsymbol{k}_t = R(\boldsymbol{W}_k \operatorname{enc}(\boldsymbol{x}_{t-1}), t, \boldsymbol{c})$.

Maintaining separate but overlapping memories is commonly needed. Navigation often involves routes with overlapping segments, requiring context to correctly recall the next step of the journey (see Figure 1B).

First, we evaluate our system by filling the memory with multiple episodes with overlapping segments and testing recall. Our system achieves 100% retrieval accuracy on a held-out test set—unseen sequences of multiple episodes and never-seen episode context vectors, $\boldsymbol{c}$—with $K = 8$ episodes of length $\ell = 32$ in $D = 256$ (i.e., a full memory load at $L = K \cdot \ell = 256 = D$).

Next, we probe whether the system can recover the context vector $\boldsymbol{c}$ of an episode from observed transitions (i.e., disambiguate "bank" vs. "bank" from context clues). See Appendix D.3 for complete details. We create a maximally challenging test (see Figure 5 for a data example): the $K = 8$ episodes of $\ell = 32$ tokens overlap in a tree-like fashion. All $K = 8$ episodes are *exactly* the same in the first 8

[4]For completeness, we also evaluate tPC (Tang et al., 2023a) in Appendix H, though that system is designed for a different purpose—memorizing training sequences into slow weights, not test-time memorization.

tokens of their episodes, then 4 episodes collide in the next 8 tokens, then 2 episodes collide in the next 8 tokens, and finally all 8 episodes diverge in the last 8 tokens (i.e., $8 + 8 + 8 + 8 = 32 = \ell$).

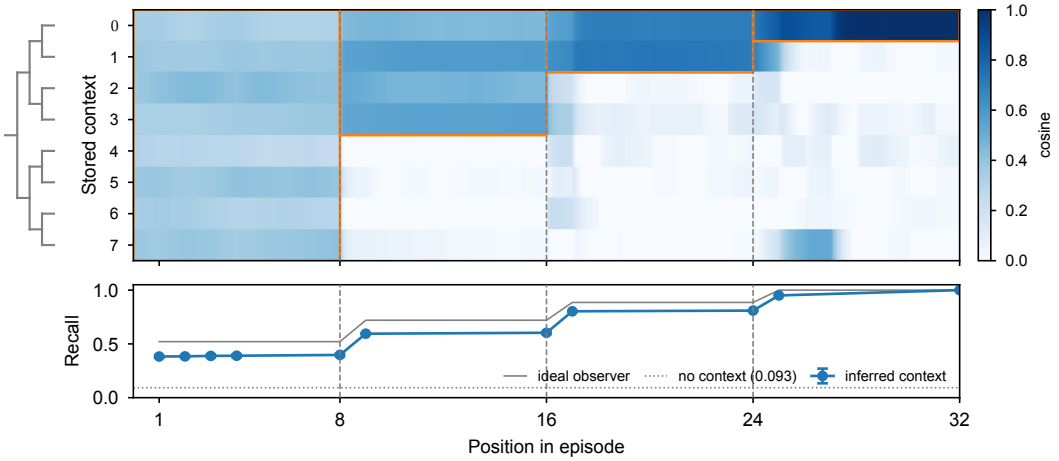


Figure 4: Contextual inference heatmap. **Top:** cosine similarity between the inferred context and the 8 possible contexts at each point in time. We observe the system resolve its uncertainty about the context as it sees more transitions. **Bottom:** given the inferred context, we measure recall accuracy for the episodes and note that it behaves nearly like the ideal observer upper bound.

We fill the system's memory with a data example ($K = 8, \ell = 32$). We then reveal observed transitions one-at-a-time and the system must infer the context $\boldsymbol{c}$ that best explains the transitions observed so far. Figure 4 shows the results of this experiment. The heat map shows the cosine similarity between the inferred context and the 8 possible contexts at each point in time. We observe the system resolve its uncertainty about the context as it sees more transitions.[5] Success of contextual inference is measured by how well the inferred context vector $\boldsymbol{c}$ can be used to recall the whole episode, shown in the bottom panel of Figure 4. We compare to an ideal observer which commits to one of the consistent context vectors until evidence refutes it (i.e., the upper bound), and find that our system tracks that closely. This experiment demonstrates the value of a system that can perform inference (i.e., "settle"): we can run the memory *backwards* to infer a hypothesis, and then run the memory forward to test that hypothesis. This is a powerful capability that requires multiple layers in standard feedforward systems (Olsson et al., 2022; Bietti et al., 2023; Sanford et al., 2024) to achieve what we do with settling "depth".

4.3 Key-value memory

The task is to store a sequence of keys paired with values, and retrieve the appropriate value when cued with a key. We compare to a modern DeltaNet implementation (Yang et al., 2024b) on the MQAR dataset (Arora et al., 2023), which tests this key-value memory capability, particularly in long-range retrieval scenarios.

Table 3 shows this comparison. Our system achieves 100% test set accuracy in as few as 32 training examples, whereas DeltaNet needs 1,024 (it is not exact with 512). We also note that by avoiding BPTT, our system's training memory does not grow with the sequence length L, whereas DeltaNet's training memory grows linearly with L. MQAR, with $V = 8{,}192$, also shows that our system handles a large vocabulary without adding parameters. See Appendix D.4 for full experimental details.

We also evaluate our system's performance on image data, where inputs can be correlated, and the set of classes is open-ended, in contrast with a fixed token vocabulary. For this we use Omniglot characters (Lake et al., 2015) as keys and MNIST digits (Lecun et al., 1998) as values (see Appendix Figure 6 for a data example). Importantly, the test set of Omniglot character keys is completely disjoint from the training set, testing generalization.

[5] Importantly, once a transition is *inconsistent* with a particular context, the cosine similarity with that context drops to near zero (see white regions in the heat map).

Table 3: Comparison of DeltaNet (Yang et al., 2024b) and DeltaNet-PC at width $D = 256$ on MQAR (512-token streams, 8,192 vocabulary). [a]Training streams needed to achieve perfect test recall. [b]Fast-weight floats: DeltaNet's per-head states, DeltaNet-PC's memory $\boldsymbol{S}$ and RLS inverse $\boldsymbol{P}$. [c]Excludes token embedding parameters. Memory measurements are based on a batch of 32 on an 80 GB GPU.

	MQAR		Local in				Peak training memory at L		
Model	Test acc.	Streams[a]	time	space	Params	State[b]	512	65k	131k
DeltaNet	1.000	1,024	×	×	0.53 M[c]	65 k	0.7 GB	55 GB	OOM
DeltaNet-PC (ours)	1.000	32	✓	✓	0.20 M	131 k	0.25 GB	0.5 GB	0.8 GB

On the Omniglot-MNIST key-value memory task with 128 key-value pairs in 256 dimensions (i.e., $L = 256$, a full memory load) our system achieves 99.99% test set accuracy showing that it supports image data and can generalize to unseen keys. We also note that our system learns to retrieve information in arbitrary order, meaning our system can learn to retrieve values by position (see single episode and multi-episode sequential memory tasks) or by content, depending on the task need.

5 Limitations

The primary limitation of any predictive coding based system is the need to settle latents, which is typically performed with iterative optimization.[6] We document our system's compute costs in Appendix G. While we have demonstrated the benefits that iterative settling can provide (Section 4.2), future work must continue to reduce the number of iterations required to perform inference, and amortized inference (Tschantz et al., 2022) provides a promising direction. We also mitigate this limitation by using the RLS update rule (Equation 7) to update the memory, which is equivalent to settling $\Delta \boldsymbol{S}_t$ under the energy function in Equation 5, but is more efficient since it doesn't require iterative optimization.

6 Discussion

In this work, we introduced DeltaNet-PC, a predictive coding system that learns to perform test-time verbatim memorization of sequential information and generalizes to content unseen in training. We demonstrated how our system tackles several common challenges for memory systems, including non-Markovian dynamics, overlapping episodes, and key-value retrieval. We also showed that our system can learn to infer context from observed transitions, which is enabled by iterative settling and compute "depth". We enforced strict requirements for local learning in space and time and bounded memory in the sequence length, which positions our system for online learning scenarios and in-memory hardware implementations.

Treating matrix-valued updates as latent variables, as opposed to slowly learned parameters, allows us to show that predictive coding natively provides linear attention machinery. We compared our work quantitatively to other common memory rules with and without learning and a modern DeltaNet implementation. We also showed mathematically and qualitatively how our work relates to other pivotal works in the linear attention and associative memory literature.

[6]On neuromorphic analog hardware, physical dynamics (i.e., Kirchhoff's and Ohm's laws) perform settling in parallel instantaneously (Kendall et al., 2020), but on digital hardware, messages must be passed between neurons.

AI USE STATEMENT

In this work, we (the authors) wrote the first version of the codebase, which provided a clear scaffold for how to run experiments, how to modify the energy function, etc. We then used AI tools to assist with writing code for additional experiments and creating figures and tables. AI was also useful for identifying related works. AI played a role in brainstorming solutions to challenges encountered throughout this project. For instance, we (the authors) intuitively expected that our energy function would require a term to account for interference with past keys. We conceptualized this term as $\sum_{i=1}^{t-1} \|\Delta \boldsymbol{S}_t \boldsymbol{k}_i\|^2$, which would naively require storing all past keys, growing our memory footprint with the sequence length L. We used AI to help us brainstorm a fixed-size memory solution, which ultimately led to the recursive least squares (RLS) update rule in Equation 7. Notably, we did not use AI tools to *write* the main text of this paper. All em dashes, typos, etc. are the work of the authors. Some sections of the appendix were initially drafted with AI assistance but all were subsequently revised by the authors.

ETHICS STATEMENT

We do not foresee any ethical concerns with our work at this stage. Our work is a proof-of-concept for a predictive coding system that learns to perform test-time verbatim memorization of sequential information.

REPRODUCIBILITY STATEMENT

The main text provides the reader with nearly enough detail to reproduce the paper. The appendices provide all remaining implementation details. Appendix B provides the complete energy function, Appendix C describes the datasets, Appendix D provides implementation details, Appendix E provides hyperparameters, Appendix F provides ablations, and Appendix G provides compute cost. We will release our codebase upon publication.

REFERENCES

Pau Vilimelis Aceituno, Sander de Haan, Reinhard Loidl, and Benjamin F. Grewe. Challenging Backpropagation: Evidence for Target Learning in the Cortex, November 2024.

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, December 2023.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using Fast Weights to Attend to the Recent Past. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a Transformer: A Memory Viewpoint, November 2023.

Rafal Bogacz. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology*, 76:198–211, February 2017. ISSN 0022-2496. doi: 10.1016/j.jmp.2015.11.003.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

Hamza Chaudhry, Jacob Zavatone-Veth, Dmitry Krotov, and Cengiz Pehlevan. Long Sequence Hopfield Memory. *Advances in Neural Information Processing Systems*, 36:54300–54340, December 2023.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking Attention with Performers. In *International Conference on Learning Representations*, October 2020.

Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):815–836, April 2005. ISSN 0962-8436. doi: 10.1098/rstb.2005.1622.

J. Hertz, A. Krogh, B. Lautrup, and T. Lehmann. Nonlinear backpropagation: Doing backpropagation without derivatives of the activation function. *IEEE Transactions on Neural Networks*, 8(6):1321–1327, November 1997. ISSN 1941-0093. doi: 10.1109/72.641455.

Geoffrey Hinton and David C. Plaut. Using Fast Weights to Deblur Old Memories. In *Cognitive Science Society*, 1987.

Georgios Iatropoulos, Johanni Brea, and Wulfram Gerstner. Kernel Memory Networks: A Unifying Framework for Memory Modeling, July 2024.

Francesco Innocenti, Paul Kinghorn, Will Yun-Farmbrough, Miguel De Llanza Varona, Ryan Singh, and Christopher L. Buckley. JPC: Flexible Inference for Predictive Coding Networks in JAX, December 2024.

Francesco Innocenti, El Mehdi Achour, and Christopher L. Buckley. μPC: Scaling Predictive Coding to 100+ Layer Networks, November 2025.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, pp. 7703–7717, Red Hook, NY, USA, December 2021. Curran Associates Inc. ISBN 978-1-7138-4539-3.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled Neural Interfaces using Synthetic Gradients, July 2017.

Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, June 2009. ISSN 1866-9964. doi: 10.1007/s12559-009-9009-8.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention, August 2020.

Jack Kendall, Ross Pantone, Kalpana Manickavasagam, Yoshua Bengio, and Benjamin Scellier. Training End-to-End Analog Neural Networks with Equilibrium Propagation, June 2020.

Patrick Kidger and Cristian Garcia. Equinox: Neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

Kimi-Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan

Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi K3: Open Frontier Intelligence, August 2026.

T. Kohonen. An Adaptive Associative Memory Principle. *IEEE Transactions on Computers*, C-23 (4):444–445, April 1974. ISSN 1557-9956. doi: 10.1109/T-C.1974.223960.

Teuvo Kohonen. Correlation Matrix Memories. *IEEE Transactions on Computers*, 21(4):353–359, April 1972. ISSN 0018-9340. doi: 10.1109/TC.1972.5008975.

Dmitry Krotov and John J. Hopfield. Dense Associative Memory for Pattern Recognition. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Dharshan Kumaran, Demis Hassabis, and James L. McClelland. What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated. *Trends in Cognitive Sciences*, 20(7):512–534, July 2016. ISSN 1364-6613. doi: 10.1016/j.tics.2016.05.004.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, December 2015. doi: 10.1126/science.aab3050.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi: 10.1109/5.726791.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference Target Propagation, November 2015.

Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, June 2020. ISSN 1471-0048. doi: 10.1038/s41583-020-0277-3.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A Unified Framework of Online Learning Algorithms for Training Recurrent Neural Networks. *Journal of Machine Learning Research*, 21 (135):1–34, 2020. ISSN 1533-7928.

James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995. ISSN 1939-1471. doi: 10.1037/0033-295X.102.3.419.

Beren Millidge, Tommaso Salvatori, Yuhang Song, Thomas Lukasiewicz, and Rafal Bogacz. Universal Hopfield Networks: A General Framework for Single-Shot Associative Memory Models, June 2022.

Beren Millidge, Mufeng Tang, Mahyar Osanlouy, Nicol S. Harper, and Rafal Bogacz. Predictive coding networks for temporal prediction. *PLOS Computational Biology*, 20(4):e1011183, April 2024. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1011183.

Davide Noè, Hideaki Yamamoto, Yuichi Katori, and Shigeo Sato. Predictive E-prop: A biologically inspired approach to train predictive coding-based recurrent spiking neural networks, February 2026. ISSN 2692-8205.

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, September 2022.

Randall C. O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Comput.*, 8(5):895–938, July 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.5.895.

Alexander G. Ororbia. Brain-Inspired Machine Intelligence: A Survey of Neurobiologically-Plausible Credit Assignment, December 2023.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random Feature Attention. In *International Conference on Learning Representations*, October 2020.

Luca Pinchetti, Chang Qi, Oleh Lokshyn, Gaspard Olivers, Cornelius Emde, Mufeng Tang, Amine M'Charrak, Simon Frieder, Bayar Menzat, Rafal Bogacz, Thomas Lukasiewicz, and Tommaso Salvatori. Benchmarking Predictive Coding Networks – Made Simple, February 2025.

R. L. Plackett. Some Theorems in Least Squares. *Biometrika*, 37(1/2):149–157, 1950. ISSN 0006-3444. doi: 10.2307/2332158.

T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3): 623–641, May 1995. ISSN 1941-0093. doi: 10.1109/72.377968.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield Networks is All You Need, April 2021.

Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, January 1999. ISSN 1546-1726. doi: 10.1038/4580.

Tommaso Salvatori, Yuhang Song, Yujian Hong, Simon Frieder, Lei Sha, Zhenghua Xu, Rafal Bogacz, and Thomas Lukasiewicz. Associative Memories via Predictive Coding, September 2021.

Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Zheng R. Xu, Lei Sha, Cornelius Emde, R. Bogacz, and Thomas Lukasiewicz. A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks. In *International Conference on Learning Representations*, 2024.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding Transformer Reasoning Capabilities via Graph Algorithms. In *Advances in Neural Information Processing Systems*, volume 37, pp. 78320–78370. Curran Associates, Inc., 2024. doi: 10.52202/079017-2490.

Benjamin Scellier and Yoshua Bengio. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*, 11, May 2017. ISSN 1662-5188. doi: 10.3389/fncom.2017.00024.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers, June 2021.

Antonino Emanuele Scurria, Dimitri Vanden Abeele, Bortolo Matteo Mognetti, and Serge Massar. Equilibrium Propagation for Non-Conservative Systems, February 2026.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding, November 2023.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at Test Time): RNNs with Expressive Hidden States, August 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive Network: A Successor to Transformer for Large Language Models, August 2023.

Mufeng Tang, Helen Barron, and Rafal Bogacz. Sequential Memory with Temporal Predictive Coding. *Advances in Neural Information Processing Systems*, 36:44341–44355, December 2023a.

Mufeng Tang, Tommaso Salvatori, Beren Millidge, Yuhang Song, Thomas Lukasiewicz, and Rafal Bogacz. Recurrent predictive coding models for associative memory employing covariance learning. *PLOS Computational Biology*, 19(4):e1010719, April 2023b. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1010719.

Alexander Tschantz, Beren Millidge, Anil K. Seth, and Christopher L. Buckley. Hybrid Predictive Coding: Inferring, Fast and Slow, April 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, pp. 35151–35174, Honolulu, Hawaii, USA, July 2023. JMLR.org.

Johannes Von Oswald, Maximilian Schlegel, Alexander Meulemans, Seijin Kobayashi, Eyvind Niklasson, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Blaise Agüera y Arcas, Max Vladymyrov, Razvan Pascanu, and João Sacramento. Uncovering mesa-optimization algorithms in Transformers, October 2024.

Johannes Von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Guillaume Lajoie, Rif A. Saurous, Charlotte Frenkel, Razvan Pascanu, Blaise Aguera y Arcas, and Joao Sacramento. MesaNet: Sequence Modeling by Locally Optimal Test-Time Training. In *The Fourteenth International Conference on Learning Representations*, October 2025.

P.J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990. ISSN 1558-2256. doi: 10.1109/5.58337.

James C. R. Whittington and Rafal Bogacz. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation*, 29(5):1229–1262, May 2017. ISSN 0899-7667. doi: 10.1162/NECO_a_00949.

Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*. IRE, 1960.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pp. 56501–56523, Vienna, Austria, July 2024a. JMLR.org.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 37 of *NIPS '24*, pp. 115491–115522, Red Hook, NY, USA, December 2024b. Curran Associates Inc. ISBN 979-8-3313-1438-5.

Jason Yoo and Frank Wood. BayesPCN: A continually learnable predictive coding associative memory. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, pp. 29903–29914, Red Hook, NY, USA, November 2022. Curran Associates Inc. ISBN 978-1-7138-7108-8.

A Extended Related Work

Predictive coding. Predictive coding (PC) (Rao & Ballard, 1999; Friston, 2005) casts cortical learning as energy descent on layerwise prediction errors, resulting in a local Hebbian-style learning rule (Bogacz, 2017). Recent work has improved PC at scale via scaling-aware weight parameterizations (Innocenti et al., 2025), JAX-based inference (Innocenti et al., 2024), and benchmarking (Pinchetti et al., 2025; Salvatori et al., 2024). We build on this core machinery and add a memory module, where matrix-valued updates are themselves PC latents, as opposed to slowly learned weights.

Sequential predictive coding. Several models extend PC to sequences. Tang et al. (2023a) introduce *temporal predictive coding* (tPC), a recurrent PC model in which a hidden state predicts the next observation. Millidge et al. (2024) link PC to Kalman filtering for temporal prediction. These systems treat the recurrent weight as a slowly learned parameter, so they cannot learn new key-value information at test time. Once training ends the model can recall only what it was trained on. Our model adds a fast-weight memory whose update emerges from the energy function used for inference.

Predictive coding for associative memory. A complementary line of work connects PC to associative memory in the static setting: covariance-based hetero-associative memory (Tang et al., 2023b), auto-associative variants (Salvatori et al., 2021), the continually-learnable BayesPCN (Yoo & Wood, 2022), and a unification of PC with modern Hopfield networks (Millidge et al., 2022; Ramsauer et al., 2021; Krotov & Hopfield, 2016; Chaudhry et al., 2023) formalized as kernel memory networks (Iatropoulos et al., 2024). These systems operate on static patterns. We instead tie a hetero-associative memory to a sequential model, with each step binding a key (the previous latent) to a value (the current latent) inside a fixed-size matrix.

Linear attention and test-time learning. Linear-attention transformers re-express attention as a recurrent state with a fixed-size matrix-valued memory (Katharopoulos et al., 2020; Choromanski et al., 2020; Peng et al., 2020; Sun et al., 2023; Yang et al., 2024a). Schlag et al. (2021) establish that linear transformers are equivalent to fast-weight programmers, with the canonical update being the DeltaNet / Widrow-Hoff rule (Widrow & Hoff, 1960). Earlier fast-weight programmer work includes Hinton & Plaut (1987); Ba et al. (2016); Irie et al. (2021). In these works the update rule is a design decision: Schlag et al. (2021) weigh the delta rule against purely additive and gated alternatives, select it for how well it manages memory, and learn a per-step gate β_t to set the write strength. The surrounding architecture is then trained by backpropagation through time, as is the case with more modern DeltaNet variants (Yang et al., 2024b). More recent work solves the regularized least-squares problem exactly at every step, with recursive least squares (RLS) (Von Oswald et al., 2024) or with conjugate gradients (Von Oswald et al., 2025). In our system many of the design decisions of prior works emerge naturally from our energy function: the outer-product form, the erase term, the memory write strength, and the use of RLS. Importantly, our memory model is embedded within a system that is trained without backpropagation through time (BPTT).

Biologically plausible learning beyond BPTT. Our work draws inspiration from the enviable properties of neuroscience and biology, which learn without backpropagation through time (Lillicrap et al., 2020), support online updates, operate within a fixed memory footprint, and have remarkable energy efficiency. Our goal is to build a system with these properties, not necessarily pursue bio-realism for its own sake. Prior works that have explored biologically plausible learning include target propagation (Lee et al., 2015; Aceituno et al., 2024), equilibrium propagation (Scellier & Bengio, 2017; Scurria et al., 2026), generalized recirculation (O'Reilly, 1996), nonlinear backpropagation (Hertz et al., 1997), synthetic gradients (Jaderberg et al., 2017), and predictive E-prop (Noè et al., 2026).

Complementary learning systems. Our split between fast online memory ($\boldsymbol{S}$) and slow synaptic weights mirrors the hippocampus/neocortex split of McClelland et al. (1995); Kumaran et al. (2016).

B Energy Function Derivations

B.1 Partial energy function

We now prove Lemma 1 stated in the main text.

Lemma 2. *For any* Δ

$$E_t(\boldsymbol{S}_{t-1}+\Delta) = E_{t-1}(\boldsymbol{S}_{t-1}) + \langle \nabla E_{t-1}(\boldsymbol{S}_{t-1}), \Delta\rangle + \tfrac{1}{2}\operatorname{Tr}(\Delta(\epsilon\mathbf{I}+\boldsymbol{C}_{t-1})\Delta^\top) + \tfrac{1}{2}\|\boldsymbol{r}_t - \Delta\boldsymbol{k}_t\|^2.$$

Proof. Recall that $E_t(\boldsymbol{S}_{t-1}+\Delta) = E_{t-1}(\boldsymbol{S}_{t-1}+\Delta) + \frac{1}{2}\|\boldsymbol{v}_t - (\boldsymbol{S}_{t-1}+\Delta)\boldsymbol{k}_t\|^2$.

A second-order Taylor expansion of $E_{t-1}(\boldsymbol{S}_{t-1}+\Delta)$ about $\boldsymbol{S}_{t-1}$, which is exact because E_{t-1} is quadratic, gives

$$E_{t-1}(\boldsymbol{S}_{t-1}+\Delta) = E_{t-1}(\boldsymbol{S}_{t-1}) + \langle \nabla E_{t-1}(\boldsymbol{S}_{t-1}), \Delta\rangle + \frac{1}{2}\operatorname{Tr}(\Delta\boldsymbol{H}_{t-1}\Delta^\top),$$

where $\boldsymbol{H}_{t-1} = \nabla^2 E_{t-1}$ is the Hessian of E_{t-1}, which does not depend on the expansion point.

Differentiating $E_{t-1}(\boldsymbol{S})$ once gives

$$\begin{aligned}\nabla E_{t-1}(\boldsymbol{S}) &= \nabla\left(\frac{\epsilon}{2}\|\boldsymbol{S}\|_F^2 + \sum_{i<t}\tfrac{1}{2}\|\boldsymbol{v}_i - \boldsymbol{S}\boldsymbol{k}_i\|^2\right)\\ &= \epsilon\boldsymbol{S} - \sum_{i<t}(\boldsymbol{v}_i - \boldsymbol{S}\boldsymbol{k}_i)\boldsymbol{k}_i^\top.\end{aligned}$$

Differentiating $E_{t-1}(\boldsymbol{S})$ a second time gives

$$\begin{aligned}\nabla^2 E_{t-1}(\boldsymbol{S}) &= \nabla\left(\epsilon\boldsymbol{S} - \sum_{i<t}(\boldsymbol{v}_i - \boldsymbol{S}\boldsymbol{k}_i)\boldsymbol{k}_i^\top\right)\\ &= \epsilon\mathbf{I} + \sum_{i<t}\boldsymbol{k}_i\boldsymbol{k}_i^\top\\ &= \epsilon\mathbf{I} + \boldsymbol{C}_{t-1}.\end{aligned}$$

This completes the proof for the first three terms of the lemma.

The last term follows from our definition of $\boldsymbol{r}_t = \boldsymbol{v}_t - \boldsymbol{S}_{t-1}\boldsymbol{k}_t$, which means that $\boldsymbol{v}_t - (\boldsymbol{S}_{t-1}+\Delta)\boldsymbol{k}_t = \boldsymbol{r}_t - \Delta\boldsymbol{k}_t$. □

Since $\boldsymbol{S}_{t-1}$ is a "fixed" quantity with respect to E_t, the first term in Lemma 1 will be a constant when optimizing E_t. Furthermore, $\boldsymbol{S}_{t-1}$ minimized $E_{t-1}(\boldsymbol{S})$ (and this holds inductively down to the base case of $\boldsymbol{S}_0 = 0 = \arg\min_{\boldsymbol{S}} E_0(\boldsymbol{S})$), which means that $\nabla E_{t-1}(\boldsymbol{S}_{t-1}) = 0$. This allows us to drop the second term in Lemma 1. We now have a closed-form energy function for E_t:

$$E_t(\boldsymbol{S}_{t-1}+\Delta) = \underbrace{\frac{1}{2}\|\boldsymbol{r}_t - \Delta\boldsymbol{k}_t\|^2}_{\text{new pair}} + \underbrace{\frac{1}{2}\operatorname{Tr}(\Delta\boldsymbol{C}_{t-1}\Delta^\top)}_{\text{stored pairs}} + \underbrace{\frac{\epsilon}{2}\|\Delta\|_F^2}_{\text{ridge}} + \text{const.} \tag{10}$$

Algebra for $\Delta\boldsymbol{S}_t$ update. Below we show algebra for Equation 6. We take the gradient of $E_t(\boldsymbol{S}_{t-1}+\Delta)$ with respect to Δ and set it to zero to find the stationary point:

$$\begin{aligned}\nabla_\Delta E_t(\boldsymbol{S}_{t-1}+\Delta) &= -(\boldsymbol{r}_t - \Delta\boldsymbol{k}_t)\boldsymbol{k}_t^\top + \Delta\boldsymbol{C}_{t-1} + \epsilon\Delta = 0,\\ \Delta(\boldsymbol{k}_t\boldsymbol{k}_t^\top + \boldsymbol{C}_{t-1} + \epsilon\mathbf{I}) &= \boldsymbol{r}_t\boldsymbol{k}_t^\top,\\ \Delta &= \boldsymbol{r}_t\boldsymbol{k}_t^\top(\boldsymbol{k}_t\boldsymbol{k}_t^\top + \boldsymbol{C}_{t-1} + \epsilon\mathbf{I})^{-1},\end{aligned} \tag{11}$$

where we have used the fact that $\boldsymbol{k}_t\boldsymbol{k}_t^\top + \boldsymbol{C}_{t-1} + \epsilon\mathbf{I}$ is a positive definite matrix and therefore invertible, which also means that due to E_t's convexity in Δ, the stationary point is the global minimum. Let us define $\boldsymbol{A}_{t-1} = \boldsymbol{C}_{t-1} + \epsilon\mathbf{I}$ and we now have the following update rule for the latent variable $\Delta\boldsymbol{S}_t$:

$$\Delta\boldsymbol{S}_t = \boldsymbol{r}_t\boldsymbol{k}_t^\top(\boldsymbol{k}_t\boldsymbol{k}_t^\top + \boldsymbol{A}_{t-1})^{-1} \tag{12}$$

Online updates. Recall that $\boldsymbol{P}_{t-1} = \boldsymbol{A}_{t-1}^{-1}$. The Sherman-Morrison formula states that for an invertible matrix $\boldsymbol{A}$ with inverse $\boldsymbol{P}$ we have

$$(\boldsymbol{A} + \boldsymbol{k}\boldsymbol{k}^\top)^{-1} = \boldsymbol{P} - \frac{\boldsymbol{P}\boldsymbol{k}\boldsymbol{k}^\top\boldsymbol{P}}{1 + \boldsymbol{k}^\top\boldsymbol{P}\boldsymbol{k}}. \tag{13}$$

Let $s_t := \boldsymbol{k}_t^\top \boldsymbol{P}_{t-1}\boldsymbol{k}_t$ and $g_t := \frac{1}{1+s_t}$. Plugging the Sherman-Morrison formula into Equation 6 gives the following update rule for $\Delta\boldsymbol{S}_t$:

$$\begin{aligned}
\Delta\boldsymbol{S}_t &= \boldsymbol{r}_t\boldsymbol{k}_t^\top \left(\boldsymbol{P}_{t-1} - \frac{\boldsymbol{P}_{t-1}\boldsymbol{k}_t\boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}}{1 + \boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}\boldsymbol{k}_t}\right) \\
&= \boldsymbol{r}_t \left(\boldsymbol{k}_t^\top \boldsymbol{P}_{t-1} - \frac{\boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}\boldsymbol{k}_t\boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}}{1 + \boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}\boldsymbol{k}_t}\right) \\
&= \boldsymbol{r}_t \left(\boldsymbol{k}_t^\top \boldsymbol{P}_{t-1} - \frac{s_t\boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}}{1 + s_t}\right) \\
&= \boldsymbol{r}_t \left(\boldsymbol{k}_t^\top \boldsymbol{P}_{t-1}(1 - \frac{s_t}{1 + s_t})\right) \\
&= \boldsymbol{r}_t \left(\frac{\boldsymbol{k}_t^\top\boldsymbol{P}_{t-1}}{1 + s_t}\right) \\
&= g_t\boldsymbol{r}_t \left(\boldsymbol{P}_{t-1}\boldsymbol{k}_t\right)^\top,
\end{aligned}$$

where the last line relies on the symmetry of $\boldsymbol{P}_{t-1}$. This *is* the recursive least squares (RLS) update rule (Plackett, 1950).

Importantly, this update rule no longer requires an inverse to be computed at every time step, which makes it feasible for online learning scenarios. It only requires the storage of $\boldsymbol{P}_{t-1}$, which is a $D \times D$ matrix, and complies with our fixed-size memory footprint constraint. Furthermore, recall that $\boldsymbol{S}_t$ is also a $D \times D$ matrix, so the total memory footprint of our memory module is $2D^2$, which is fixed and does not grow with the sequence length L. At each timestep, we update $\boldsymbol{S}_t$ and $\boldsymbol{P}_t$ as

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + g_t\boldsymbol{r}_t \left(\boldsymbol{P}_{t-1}\boldsymbol{k}_t\right)^\top, \qquad \boldsymbol{S}_0 = \boldsymbol{0} \tag{14}$$

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} - g_t(\boldsymbol{P}_{t-1}\boldsymbol{k}_t)(\boldsymbol{P}_{t-1}\boldsymbol{k}_t)^\top, \qquad \boldsymbol{P}_0 = \mathbf{I}/\epsilon. \tag{15}$$

B.2 Relating the energy function to linear attention

Equations 5 and 7 can be interpreted relative to common choices that have been made in the linear attention literature, in particular the (i) outer-product form found in (Katharopoulos et al., 2020), (ii) the erase term found in (Schlag et al., 2021), (iii) the write strength (e.g., the gate β_t that Schlag et al. (2021) must learn), and (iv) the least-squares objective over all past pairs, which the Mesa layer solves with RLS (Von Oswald et al., 2024), all of which follow from our energy function.

(i) Outer-product form. If we write $\boldsymbol{w}_t := g_t\,\boldsymbol{P}_{t-1}\boldsymbol{k}_t$, then Equation 7 becomes

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{r}_t\boldsymbol{w}_t^\top, \qquad \boldsymbol{S}_0 = \boldsymbol{0} \tag{16}$$

consistent with the outer-product form chosen for linear attention (Katharopoulos et al., 2020).

(ii) Erase term. Moving from vanilla linear attention $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$ to DeltaNet, Schlag et al. (2021) introduced an erase term to reduce interference, given by $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \beta_t\,\boldsymbol{v}_t\boldsymbol{k}_t^\top - \beta_t\,\boldsymbol{S}_{t-1}\boldsymbol{k}_t\boldsymbol{k}_t^\top$. Expanding $\boldsymbol{r}_t$ in Equation 7 and writing $\boldsymbol{w}_t := \boldsymbol{P}_t\boldsymbol{k}_t$ gives

$$\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \underbrace{\boldsymbol{v}_t\boldsymbol{w}_t^\top}_{\text{write}} - \underbrace{\boldsymbol{S}_{t-1}\boldsymbol{k}_t\boldsymbol{w}_t^\top}_{\text{erase}} \tag{17}$$

where our system has the same write and same erase, with $\boldsymbol{w}_t$ in place of $\beta_t\boldsymbol{k}_t$, where $\boldsymbol{w}_t$ is set by the energy.

(iii) Write strength. DeltaNet's write strength gate β_t is a learned scalar applied to every direction. Ours is a matrix $\boldsymbol{P}_t = (\epsilon \mathbf{I} + \boldsymbol{C}_t)^{-1}$. It results from an interaction with past keys (recall $\boldsymbol{C}_t = \sum_{i=1}^{t} \boldsymbol{k}_i \boldsymbol{k}_i^\top$) and the current key $\boldsymbol{k}_t$. Where DeltaNet has $\beta_t \boldsymbol{k}_t$, we have $\boldsymbol{P}_t \boldsymbol{k}_t$. The Mesa layer of Von Oswald et al. (2024) uses the same matrix, with a learned ridge in place of our fixed ϵ.

(iv) Memory interference term. The term $\frac{1}{2}\operatorname{Tr}(\Delta \boldsymbol{C}_{t-1} \Delta^\top)$ from our energy Equation 5 can be rewritten as: $\frac{1}{2}\sum_{i<t} \|\Delta \boldsymbol{k}_i\|^2$. It has an elegant interpretation: penalize any write Δ in proportion to how much it changes the retrieval on any past key $\boldsymbol{k}_i$. Any nonzero $\Delta \boldsymbol{k}_i$ contributes to interference so the energy function is actively trying to minimize it. Von Oswald et al. (2024) define the Mesa layer as the exact minimizer of the same ridge-regression objective over all past pairs (Equation 3) and compute it with RLS. Their motivation is in-context learning: the layer computes the least-squares solution in one step that trained Transformers approximate across many layers. The interference term is implicit in their objective. Our derivation makes it explicit and arrives at it from the PC energy.

B.3 Value-orthogonality regularizer

In order to further improve key geometry, we can also engage an optional regularizer that encourages values $\boldsymbol{v}_t$ to be orthogonal to prior values $\boldsymbol{v}_i$ for $i < t$. Recall that keys are readouts of values—$\boldsymbol{v}_t \approx \mathrm{enc}(\boldsymbol{x}_t)$ and $\boldsymbol{k}_{t+1} \approx R(\boldsymbol{W}_k \boldsymbol{v}_t, t+1)$—so keys inherit any improved geometry from the values. We remove magnitude information so that we only encourage angular separation of values. Let $\hat{\boldsymbol{v}} = \boldsymbol{v}/\|\boldsymbol{v}\|$ be a normalized value vector, $\boldsymbol{C}^v_{t-1} = \sum_{i<t} \hat{\boldsymbol{v}}_i \hat{\boldsymbol{v}}_i^\top$ be the accumulated outer-product of prior normalized values, and $\tilde{\boldsymbol{C}}^v_{t-1} = \boldsymbol{C}^v_{t-1} / \max(\operatorname{Tr}(\boldsymbol{C}^v_{t-1}), 1)$ be the normalized version of $\boldsymbol{C}^v_{t-1}$ that is invariant to the number of prior values. The vanilla form of the regularizer is

$$\mathcal{R}^{\boldsymbol{v}}_t \;=\; \tfrac{1}{2}\hat{\boldsymbol{v}}_t^\top \tilde{\boldsymbol{C}}^v_{t-1} \hat{\boldsymbol{v}}_t \;=\; \frac{1}{2\max(t-1,\,1)} \sum_{i<t} (\hat{\boldsymbol{v}}_t^\top \hat{\boldsymbol{v}}_i)^2. \tag{18}$$

Note that in some cases, one may observe the same value multiple times in a sequence, and in that case, we may not want to try to orthogonalize the value with respect to itself. In that case, we can use a modified version of the regularizer that ignores the current value $\boldsymbol{v}_t$ in the accumulated outer-product. Let $\hat{\boldsymbol{e}}_t = \mathrm{enc}(\boldsymbol{x}_t)/\|\mathrm{enc}(\boldsymbol{x}_t)\|$ (stop-gradiented) and $m = \hat{\boldsymbol{e}}_t^\top \tilde{\boldsymbol{C}}^v_{t-1} \hat{\boldsymbol{e}}_t$ be the magnitude of the current value in the accumulated outer-product. The modified regularizer is

$$\mathcal{R}^{\boldsymbol{v}}_t \;=\; \tfrac{1}{2}\hat{\boldsymbol{v}}_t^\top \tilde{\boldsymbol{C}}^v_{t-1} \hat{\boldsymbol{v}}_t - \tfrac{1}{2} m (\hat{\boldsymbol{e}}_t^\top \hat{\boldsymbol{v}}_t)^2 \;=\; \tfrac{1}{2}\hat{\boldsymbol{v}}_t^\top \boldsymbol{M}_t \hat{\boldsymbol{v}}_t, \tag{19}$$

$$\boldsymbol{M}_t \;=\; \tilde{\boldsymbol{C}}^v_{t-1} - m \hat{\boldsymbol{e}}_t \hat{\boldsymbol{e}}_t^\top. \tag{20}$$

Notably, the memory footprint of this regularizer is $O(D^2)$, which is independent of the sequence length L.

B.4 Full energy function

Every factor of Eq. 9 contributes one quadratic term of the form $\frac{1}{2}\|\text{variable} - \text{mean}\|^2$, the stored pairs term comes from Lemma 1, and the value-orthogonality regularizer is detailed in §B.3. The energy at position t is a function of three latent variables $(\boldsymbol{v}_t, \boldsymbol{k}_t, \Delta \boldsymbol{S}_t)$ and is given by

$$\begin{aligned} E_t(\boldsymbol{v}_t, \boldsymbol{k}_t, \Delta \boldsymbol{S}_t) \;=\; & \underbrace{\tfrac{1}{2}\|\boldsymbol{x}_t - \mathrm{dec}(\boldsymbol{v}_t)\|^2}_{\text{decoder}} + \underbrace{\tfrac{\lambda_{\text{ev}}}{2}\|\boldsymbol{v}_t - \mathrm{enc}(\boldsymbol{x}_t)\|^2}_{\text{encoder}} + \underbrace{\tfrac{1}{2}\|\boldsymbol{v}_t - (\boldsymbol{S}_{t-1} + \Delta \boldsymbol{S}_t)\,\boldsymbol{k}_t\|^2}_{\text{memory energy (new pair)}} \\ & + \underbrace{\tfrac{\lambda_k}{2}\|\boldsymbol{k}_t - R(\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}_{t-1}), t)\|^2}_{\text{address prior}} + \underbrace{\tfrac{1}{2}\operatorname{Tr}(\Delta \boldsymbol{S}_t\, \boldsymbol{C}_{t-1}\, \Delta \boldsymbol{S}_t^\top)}_{\text{stored pairs}} \\ & + \underbrace{\tfrac{\epsilon}{2}\|\Delta \boldsymbol{S}_t\|_F^2}_{\text{ridge}} + \lambda_v\, \mathcal{R}^{\boldsymbol{v}}_t(\boldsymbol{v}_t) \;+\; \text{const}, \end{aligned} \tag{21}$$

with $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$ observed, and $\boldsymbol{S}_{t-1}$, $\boldsymbol{C}_{t-1}$ and $\boldsymbol{C}^v_{t-1}$ the memory as of the previous step, $t-1$. Figure 2 visualizes Eq. 21: one Gaussian term per arrow, plus the memory terms. A deeper encoder or decoder would add one term of the form $\frac{1}{2}\|\boldsymbol{z}_\ell - \boldsymbol{W}_\ell\, g(\boldsymbol{z}_{\ell-1})\|^2$ per layer (Eq. 1).

Write and learn passes. The training procedure (Algorithm 1) minimizes Eq. 21 over different variables depending on the type of event observed in the stream: an observation $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$ (the write) or a query $(\boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$ (the learn pass).

The write (an observation $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$) minimizes the energy function over $\Delta \boldsymbol{S}_t$ alone, with the value and the address at their prior means, $\boldsymbol{v}_t = \text{enc}(\boldsymbol{x}_t)$ and $\boldsymbol{k}_t = R(\boldsymbol{W}_k\ \text{enc}(\boldsymbol{x}_{t-1}), t)$. Only the three $\Delta \boldsymbol{S}_t$ terms depend on $\Delta \boldsymbol{S}_t$, so the minimizer is the closed form of Eq. 6, computed recursively as Eq. 7. It commits updates to $\boldsymbol{S}_t$, $\boldsymbol{P}_t$, and $\tilde{\boldsymbol{C}}_t^v$.

The learn pass (a query $(\boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$, Table 4) minimizes the same energy over the value and the address key, $(\boldsymbol{v}, \boldsymbol{k})$, with the memory held fixed and the observation $\boldsymbol{y}$ clamped at the output. In particular, that is the energy function restricted to the following:

$$\begin{aligned} E_t^{\text{learn}}(\boldsymbol{v}, \boldsymbol{k};\, \boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p) \;&=\; \tfrac{1}{2}\|\boldsymbol{y} - \text{dec}(\boldsymbol{v})\|^2 \;+\; \tfrac{\lambda_{\text{ev}}}{2}\|\boldsymbol{v} - \text{enc}(\boldsymbol{y})\|^2 \;+\; \tfrac{1}{2}\|\boldsymbol{v} - \boldsymbol{S}_t\boldsymbol{k}\|^2 \\ &\quad +\; \tfrac{\lambda_k}{2}\|\boldsymbol{k} - R(\boldsymbol{W}_k\ \text{enc}(\boldsymbol{x}^{\text{in}}), p)\|^2 \;+\; \lambda_v\, \mathcal{R}_t^{\boldsymbol{v}}(\boldsymbol{v}). \end{aligned} \tag{22}$$

The pair is optimized for T gradient steps, initialized from the same prior means the write used, and the parameter step is the gradient of Eq. 22 at the settled point with respect to the parameters: dec, enc, $\boldsymbol{W}_k$ and the rotary frequencies from the address prior.

Recall. At test time, the default is to read out $\boldsymbol{S}_t\boldsymbol{k}_t$ at the feedforward address and decode to the observation space. The decoded token is re-encoded to form the next address. Nothing is written (Algorithm 1). Optionally, a write-time settle can be run to refine the value and memory update (with the key held fixed at the feedforward address).

B.5 Algorithmic description of DeltaNet-PC

Algorithm 1 gives the full training step described in §3.3.

Algorithm 1: DeltaNet-PC on one training example / event stream containing observations and queries (Table 4).

Input: Event stream $e_1, \ldots, e_N$, each event an observation $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$ or a query $(\boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$: a cue, a target, and the position; parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}, \boldsymbol{W}_k, \boldsymbol{\theta}_R)$.

$\boldsymbol{S} \leftarrow \boldsymbol{0}$; $\boldsymbol{P} \leftarrow \boldsymbol{I}/\epsilon$; $\boldsymbol{C}^v \leftarrow \boldsymbol{0}$; $\bar{\boldsymbol{g}} \leftarrow \boldsymbol{0}$; $n \leftarrow 0$

foreach *event* e *in the stream* **do**

  if e *is an observation* $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$ **then**

    ▷ memory update: minimizer of E_t (Eq. 5)

    $\boldsymbol{k} \leftarrow R(\boldsymbol{W}_k\ \text{enc}(\boldsymbol{x}_{t-1}), t)$; $\boldsymbol{v} \leftarrow \text{enc}(\boldsymbol{x}_t)$

    $\boldsymbol{r} \leftarrow \boldsymbol{v} - \boldsymbol{S}\boldsymbol{k}$; $\boldsymbol{w} \leftarrow \boldsymbol{P}\boldsymbol{k}$; $g \leftarrow 1/(1 + \boldsymbol{k}^\top \boldsymbol{w})$

    $\boldsymbol{S} \leftarrow \boldsymbol{S} + g\, \boldsymbol{r}\, \boldsymbol{w}^\top$; $\boldsymbol{P} \leftarrow \boldsymbol{P} - g\, \boldsymbol{w}\boldsymbol{w}^\top$; $\boldsymbol{C}^v \leftarrow \boldsymbol{C}^v + \hat{\boldsymbol{v}}\hat{\boldsymbol{v}}^\top$

  else query $(\boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$

    ▷ settle key and value with $\boldsymbol{\theta}$ and $\boldsymbol{S}$ frozen

    $\tilde{\boldsymbol{k}} \leftarrow R(\boldsymbol{W}_k\ \text{enc}(\boldsymbol{x}^{\text{in}}), p)$; $\tilde{\boldsymbol{v}} \leftarrow \text{enc}(\boldsymbol{y})$

    for $\tau = 1, \ldots, T$ **do**

      $(\tilde{\boldsymbol{k}}, \tilde{\boldsymbol{v}}) \leftarrow (\tilde{\boldsymbol{k}}, \tilde{\boldsymbol{v}}) - \eta_x\, \nabla_{(\tilde{\boldsymbol{k}}, \tilde{\boldsymbol{v}})}\, E^{\text{learn}}(\tilde{\boldsymbol{k}}, \tilde{\boldsymbol{v}};\, \boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$

    $\bar{\boldsymbol{g}} \leftarrow \bar{\boldsymbol{g}} + \nabla_{\boldsymbol{\theta}}\, E^{\text{learn}}(\tilde{\boldsymbol{k}}, \tilde{\boldsymbol{v}};\, \boldsymbol{x}^{\text{in}}, \boldsymbol{y}, p)$; $n \leftarrow n + 1$ ▷ Parameter update

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_\theta\, \frac{256}{n}\, \bar{\boldsymbol{g}}$ ▷ one step per episode: scaled by $\times 256$

return $\boldsymbol{\theta}$

C Datasets

Here we describe each of our datasets in more detail. Table 5 summarizes our four datasets. Each dataset provides a stream of observations $\boldsymbol{x}_t \in \mathbb{R}^{d_{\text{obs}}}$, where tokens are represented as $d_{\text{obs}} = 256$-dimensional vectors drawn from a Gaussian codebook that is unit-normalized, and images are represented as 28×28 pixel arrays flattened to $d_{\text{obs}} = 784$. All tasks use disjoint training, validation, and test splits.

Table 4: The event streams for the three tasks.

Task	Event stream	Test scoring
Sequential recall	Observations and queries alternate: after each observation $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$ the learner poses its own query $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$, to retrieve what was just written at its own position, against the memory as it stands. L observations and L queries per stream; online learning.	Ground truth $\boldsymbol{x}_t$ compared to prediction $\hat{\boldsymbol{x}}_t$ decoded from $\boldsymbol{S}_L R(\boldsymbol{W}_k \operatorname{enc}(\hat{\boldsymbol{x}}_{t-1}), t)$ for $t = 1, \ldots, L$, the final memory frozen.
Official MQAR	The 128 context observations (64 pairs), then the query section: at each labelled position t the task's query $(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t, t)$, the query key and its value, precedes the observation that writes the filler $\boldsymbol{x}_t$. Every position is written.	Ground truth $\boldsymbol{y}_t$ compared to prediction $\hat{\boldsymbol{y}}_t$ decoded from $\boldsymbol{S}_{t-1} R(\boldsymbol{W}_k \operatorname{enc}(\boldsymbol{x}_{t-1}), t)$ for each t containing a query.
Omniglot binding	Stream of L observations, then the task's queries: all queries have the form $(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, L)$ meaning they are queries at the final position L.	Ground truth $\boldsymbol{x}_t$ compared to prediction $\hat{\boldsymbol{x}}_t$ decoded from $\boldsymbol{S}_L\, R(\boldsymbol{W}_k \operatorname{enc}(\boldsymbol{x}_{t-1}), L)$.

Table 5: Summary of datasets.

Task	Observation	d_{obs}	L, D	V	Train / Val. / Test	Scored units per test set
Single-episode sequences	token	256	$L = 64$ to 512 at $D = 256$, $L = 512$ at $D = 64$ to 512, $L = D = 1{,}024$	10 tokens	1,024 / 128 / 128	$128L$ relay tokens
Multi-episode sequences	token	256	256 ($K = 8 \times \ell = 32$), $D = 256$	10 tokens	1,024 / 128 / 128	32,768 relay tokens
MQAR (Arora et al., 2023)	token	256	512, $D = 256$	8,192 tokens (4,095 keys, 4,096 values)	100,000 (or 32) / 128 / 3,000	192,000 labelled positions
Omniglot-MNIST binding	image 28×28	784	256 (128 pairs), $D = 256$	10 digits	1,024 / 128 / 128	16,384 probes

C.1 Single-episode sequential memory

Generator. A sequence of L tokens is drawn uniformly from a vocabulary of size V. One non-Markovian transition is inserted into each sequence, so that a given token is followed by two different successors—the chorus with two bridges of the song analogy in the main text.

Observations. Each token is represented as its codebook row in $\mathbb{R}^{256}$, shared by all splits and all seeds. Decoding is performed by finding the minimum distance codebook entry by Euclidean distance.

Sizes. Each split is composed of independently generated sequences: 1,024 for training and 128 each for validation and test. The lengths used are $L \in \{64, 128, 256, 512\}$ at $D = 256$, $L = 512$ at $D \in \{64, 128, 256, 512\}$, and $L = D = 1{,}024$, three seeds each. $n = 0$ is the untrained control (every learning rate zero), three seeds each.

Metric. Recall accuracy is the fraction of the L ground truth tokens that are recalled after the memory has been filled. The model starts decoding from a start observation $\boldsymbol{x}_0$, queries the memory with the resulting key $\boldsymbol{k}_1 = R(\boldsymbol{W}_k \operatorname{enc}(\boldsymbol{x}_0), 1)$, and decodes the result to form $\hat{\boldsymbol{x}}_1$ from $\min_{\boldsymbol{x} \in \text{codebook}} \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2$ where $\hat{\boldsymbol{x}}$ is the model's prediction. The model then forms the next key, reads the memory, and decodes to form $\hat{\boldsymbol{x}}_2$, and so on until $\hat{\boldsymbol{x}}_L$. Notably, we use no teacher forcing—

the model must stay on track through the entire L-length sequence to succeed. The test set scores $128 \times L$ tokens, 32,768 at $L = 256$.

C.2 MULTI-EPISODE SEQUENTIAL MEMORY

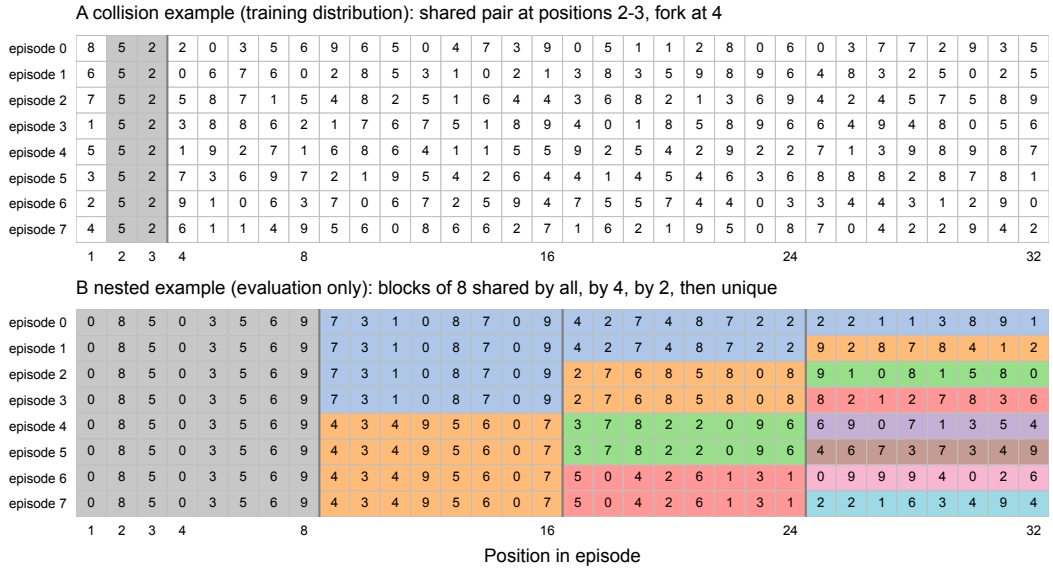


Figure 5: Multi-episode examples. **A**: A single training example. **B**: An evaluation example for the contextual inference task from Section 4.2.

Generator, collision examples (training and evaluation). An example is $K = 8$ episodes of $\ell = 32$ tokens, written into a single stream of length $L = 256$. Within-episode tokens are drawn uniformly from a vocabulary of size V. All K episodes of an example have the same two tokens at within-episode positions 2 and 3. The predecessors at position 1 are pairwise distinct across the episodes, as are the successors at position 4. At the end of the shared block the pair (position, token) is identical in every episode, so the successor is ambiguous without context, and a memory keyed on content and position alone reads out the superposition of the K successors. Figure 5A shows an example.

Generator, nested examples (evaluation only). The contextual inference experiment evaluates DeltaNet-PC on a harder task structure. See Figure 5B for an example. The episode is four blocks of 8 tokens ($\ell = 32$). Block 1 is shared by all 8 episodes, block 2 by 4, block 3 by 2, and block 4 is unique in each episode. Sibling groups differ at the first token of their block. These examples are generated at evaluation time only.

Episode context vectors. The dataset provides each episode with a context vector $\boldsymbol{c} \in \mathbb{R}^D$, which the model consumes as the context tag on the key (Appendix D.1). Context vectors are unit-modulus phasors—one uniformly random angle per rotation plane. During training the K context vectors are drawn once per run from the run seed and shared by every training example. At validation and test, *every example* draws fresh context vectors, so all reported multi-episode numbers are for never-seen context vectors. Rotary positions for the learned RoPE frequencies restart at every episode start. Recall that in the multi-episode setting, keys are defined as $\boldsymbol{k}_t = R(\boldsymbol{W}_k \ \mathrm{enc}(\boldsymbol{x}_{t-1}), t, \boldsymbol{c})$, where R applies a rotation in each plane of the key vector by the learned RoPE frequency for that plane, scaled by the position t and the context vector $\boldsymbol{c}$.

Sizes and metric. Splits are 1,024 / 128 / 128 examples. Recall begins at the start of an episode and proceeds through the episode, as in the single-episode setting. Every position is scored: $128 \times 256 = 32{,}768$ tokens on the test set.

C.3 MQAR

Generator. We use the community-standard multi-query associative recall stream of Arora et al. (2023) in its hardest published setting. A stream has 512 tokens and a vocabulary of 8,192. The first 128 tokens are 64 key-value pairs, and the remaining 384 tokens are the query section: each of the 64 keys appears exactly once more and every other position holds the filler token 0. The label at a query position is that key's value, which is the target of the prediction made after reading the query key, while the next input token is the filler. Every other position is unlabelled. The model writes every position of the stream, filler and queries included, and is supervised and scored only at the 64 labelled positions.

Observations. Tokens are drawn from rows of an $8{,}192 \times 256$ Gaussian codebook for DeltaNet-PC. The DeltaNet baseline receives its embedding table input. Decoding is performed by finding the nearest of all 8,192 rows by Euclidean distance for DeltaNet-PC and argmax of the logits for the DeltaNet baseline.

Sizes. We train on Zoology's full 100,000-stream training set, early-stopped on validation set recall accuracy. The test set is Zoology's 3,000-stream test segment. Early stopping uses our own 128-stream validation split.

Metric. Accuracy at the labelled positions of the test set, $3{,}000 \times 64 = 192{,}000$ scored positions, at $D = 256$. The answer at a query is read from the memory before that position is written.

C.4 Omniglot-MNIST key-value binding

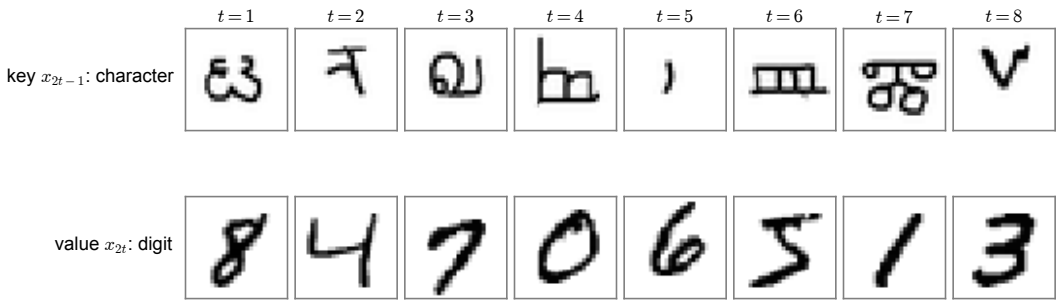


Figure 6: Omniglot-MNIST key-value memory example.

Images. Keys are Omniglot characters (Lake et al., 2015) and values are MNIST digits (Lecun et al., 1998), both at 28×28 pixels and flattened to $\mathbb{R}^{784}$. Omniglot images are inverted to bright strokes on a dark background, matching MNIST. Every image is centered by subtracting its own pixel mean, then the mean image of the training pool of its type is subtracted (mean for characters, separate from the mean for digits), and the result is scaled to unit norm. The value vocabulary is one fixed MNIST exemplar per digit—the first occurrence of each digit in the MNIST training set.

Character pools. Character classes are disjoint by split. Training draws from 964 background archive classes, and validation and test draw from disjoint halves of the 659-class evaluation archive (329 and 330 classes), so every character at test time belongs to a class the model never saw in training. Each class has 20 exemplars, and each character in an example is drawn at random.

Stream layout. An example consists of $P = 128$ characters from 128 distinct classes, each once, as interleaved (character, digit) image pairs in a shuffled order, so the stream has $L = 2P = 256$ observations and fills a memory of width $D = 256$. Each character is bound to a digit, where digits are drawn uniformly with replacement, so many characters (keys) share a value. Figure 6 shows an example.

Sizes and metric. Splits are 1,024 / 128 / 128 examples, and three seeds. After the stream is written, every character (key) of the example is presented again as a query at the final position and the digit read from the finished memory is decoded by the nearest of the ten digit prototypes. The test set scores $128 \times 128 = 16{,}384$ probes.

D Architecture, Training, and Experimental Details

In this section, we provide additional details on the architecture, training, and experimental setup used in our work.

D.1 Rotary position code

Definition. Write the D coordinates of a key as $D/2$ planes $(k_i, k_{i+D/2})$. The rotary code $R(\boldsymbol{k}, t)$ rotates plane i by the angle $t\,\theta_i$,

$$R(\boldsymbol{k}, t)_i \;=\; \begin{pmatrix} \cos t\theta_i & -\sin t\theta_i \\ \sin t\theta_i & \cos t\theta_i \end{pmatrix} \begin{pmatrix} k_i \\ k_{i+D/2} \end{pmatrix}, \qquad \boldsymbol{\theta}_R = (\theta_1, \ldots, \theta_{D/2}) \in [0, \pi]^{D/2}. \tag{23}$$

This is rotary positional encoding (Su et al., 2023) in its non-interleaved form. $R(\cdot, t)$ is orthogonal, so it preserves the key norm, and rotations in the same planes compose by adding angles. Queries are keys, $\boldsymbol{q}_t = \boldsymbol{k}_t = R(\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}_{t-1}), t)$, and the write, the learn-pass read and the recall relay all call the same map, so write and read addresses agree by construction. The official MQAR row uses no rotary code to be consistent with DeltaNet (Yang et al., 2024b), so our key is defined as $\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}_{t-1})$.

Learned frequencies. The frequencies are parameters of the model, one scalar per plane in radians per position. They are initialized at the standard table $\theta_i = b^{-2(i-1)/D}$ with $b = 10^4$, so a fresh model addresses exactly like standard RoPE, and they are updated by SGD, after which each θ_i is projected onto $[0, \pi]$. A band at 0 ignores position and a band at π alternates every position. Nothing in the recipe depends on the sequence length.

How the frequencies learn. The frequencies appear in exactly one energy term, the address prior of the learn pass, $\frac{\lambda_k}{2}\|\boldsymbol{k} - R(\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}^{\mathrm{in}}), p)\|^2$ at the query position p. They are held constant on every other path, so they only learn from retrieval error. In plane i write the content component as the complex number a_i and the settled address as k_i. When the settled address points at the address the same content was written under at position t_w, $k_i = e^{\mathrm{i}t_w\theta_i} a_i$, the gradient is

$$\frac{\partial E^{\mathrm{learn}}}{\partial \theta_i} \;=\; -\,\lambda_k\, p\, |a_i|^2 \sin\big(\theta_i\,(t_w - p)\big). \tag{24}$$

Each band therefore behaves as a phase-locked loop between the query position and the write position, with a force proportional to p. Where the task places its queries decides what the frequencies learn (Figure 7):

- **Sequential recall** queries each transition at its own position, so the binding just written exerts no force. What moves the frequencies is interference with other occurrences of the same token stored at other positions. The 76 bands that start slower than one cycle over the sequence are lifted by a median factor of 170 to 230 across seeds, to a spectrum near one radian per position, and 4 to 6 bands end static. In brief, the code speeds up to avoid collisions of the same token at other positions, thus improving address orthogonality.
- **Multi-episode recall** does the same over the episode: the 105 bands slower than one cycle per episode are lifted by a median factor of 23 to 29, and 13 to 15 end static.
- **Omniglot binding** reads every pair at the final position, after all writes, so $t_w < p$ for every stored pair and the force lowers the slow bands. Of the 128 bands, 88 to 94 end below 10^{-3} radians per position, while the fastest 30 or so keep values near their initialization. The address stops depending on when a pair was written, which is what a content-addressed probe needs.

We emphasize that we were very intentional about designing a system that could *learn* to use positional information when needed and ignore it when not needed. The alternative is to hard-code the RoPE frequencies based on task need, which would make our system less general. In general, we made efforts to remove inductive biases, wherever possible.

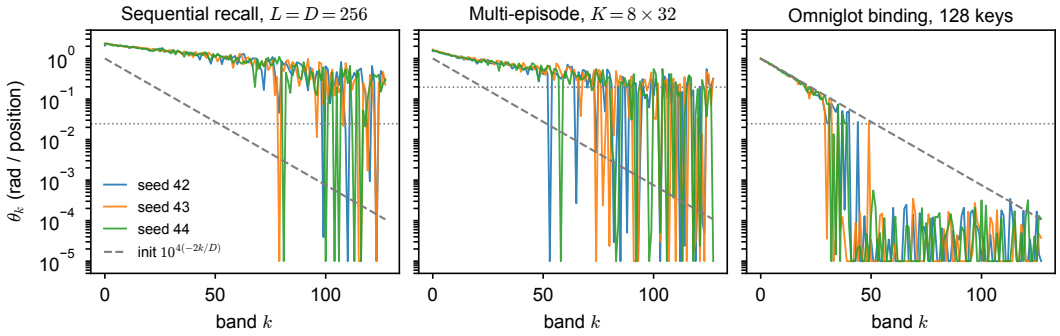


Figure 7: Learned rotary frequencies of the three rows that use them, three seeds each, against the base-10^4 initialization (dashed). The dotted line marks one cycle per horizon: $2\pi/256$ on the sequential and Omniglot rows and $2\pi/32$, one episode, on the multi-episode row. Sequential and multi-episode recall raise the slow bands, and Omniglot binding, which reads after every write, sends most of them to zero.

D.2 Context vectors and the multi-episode protocol

Context vectors. Every episode e of a multi-episode example carries a context vector $\boldsymbol{c}_e \in \mathbb{R}^D$ made of $D/2$ unit phasors: draw $\boldsymbol{\psi}_e \sim \mathcal{U}(-\pi, \pi)^{D/2}$ and set $\boldsymbol{c}_e = (\cos \boldsymbol{\psi}_e, \sin \boldsymbol{\psi}_e)$. During training the K context vectors are drawn once per run from the training seed and shared by every training example. At validation and test every example draws fresh context vectors, so every reported multi-episode number is recall under contexts never seen in training.

How a context vector enters the key. The context vector is present at every position of its episode. It does not pass through the encoder or the key map. It multiplies each plane of the key as a complex number, $(k_i + \mathrm{i}\, k_{i+D/2})(\cos \psi_{e,i} + \mathrm{i} \sin \psi_{e,i})$, which is a rotation of plane i by $\psi_{e,i}$, the same planes RoPE rotates. With positions counted within the episode, the address of the transition at within-episode position p of episode e is

$$\boldsymbol{k} \;=\; R\big(\boldsymbol{W}_k \,\mathrm{enc}(\boldsymbol{x}_{t-1}),\; p\,\boldsymbol{\theta}_R + \boldsymbol{\psi}_e\big), \tag{25}$$

so position and context compose by adding phases. We wrote this as $R(\cdot, t, \boldsymbol{c})$ in §3.2. Both factors are orthogonal and commute, so the tag changes neither the key norm nor anything else in the recipe, and a single-episode run is the case $\boldsymbol{\psi} = \mathbf{0}$. A read under the same context vector as the write cancels the tag exactly in the readout, $R(\boldsymbol{k}, \boldsymbol{\psi}_e)^\top R(\boldsymbol{k}', \boldsymbol{\psi}_e) = \boldsymbol{k}^\top \boldsymbol{k}'$. A read under a different context vector sees a random rotation in every plane, $\sum_i |k_i|^2 \cos(\psi_{e,i} - \psi_{e',i})$, which averages to zero, so keys from different contexts are nearly orthogonal. Multiplying a vector componentwise by random unit phasors is the binding operation of Fourier holographic reduced representations (Plate, 1995; Kanerva, 2009). What is new here is its use as the context channel of a least-squares fast-weight memory with contexts never seen in training, and the inference of a withheld context by settling (§D.3).

Protocol. The K episodes of an example are written as one stream of length $K\ell$ into one memory, which is not reset inside the example. Rotary positions restart at every episode start, and the first key of every episode is formed from one fixed start observation, the same for all episodes, examples and splits, so the first keys of different episodes differ only through their context vectors. Recall provides each episode with its context vector and runs the decode relay against the finished memory, exactly as in the single-episode protocol.

D.3 CONTEXTUAL INFERENCE BY SETTLING

Examples. An example is $K = 8$ episodes of $\ell = 32$ tokens written into one memory under 8 context vectors. In the *collision examples*, the training distribution, every episode has its own first token, all eight share the tokens at positions 2 and 3, and each has its own token at position 4, so the shared block is a collision only the context vector resolves. In the *nested examples* each episode is four blocks of eight tokens arranged as a binary tree: all eight episodes agree on block one, groups of four on block two, pairs on block three, and every episode has its own block four (Figure 5). Nested examples are only used at evaluation, on the models trained on collision examples.

Settling (i.e., inference). The memory is filled under the true context vectors and then held fixed. For each query episode, the context vector is withheld, and a context *latent* $\boldsymbol{c} \in \mathbb{R}^D$ is inferred from the episode's first n transitions, revealed one at a time. For each revealed transition $(\boldsymbol{x}_{p-1}, \boldsymbol{x}_p)$ the value is pinned at the encoder's output, $\boldsymbol{v}_p = \mathrm{enc}(\boldsymbol{x}_p)$, which is what the write stored, and the address is taken to equal its prediction under the current context latent, the limit of infinite address-prior precision. The context latent then settles on the read error of the value,

$$E_p(\boldsymbol{c}) \;=\; \tfrac{1}{2}\big\|\mathrm{enc}(\boldsymbol{x}_p) - \boldsymbol{S}_L\, R\big(\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}_{p-1}),\, p\,\boldsymbol{\theta}_R + \boldsymbol{\psi}(\boldsymbol{c})\big)\big\|^2, \tag{26}$$

where $\boldsymbol{\psi}(\boldsymbol{c})$ reads $\boldsymbol{c}$ as $D/2$ phasors.

We maintain the outer-product matrix $\boldsymbol{G} = \sum_e \hat{\boldsymbol{c}}_e \hat{\boldsymbol{c}}_e^\top$ of the stored context vectors, where $\hat{\boldsymbol{c}}_e = \boldsymbol{c}_e/\|\boldsymbol{c}_e\|$, a $D \times D$ matrix, and their mean $\bar{\boldsymbol{c}}$. The context latent starts at $\bar{\boldsymbol{c}}$, is carried from transition to transition, and is settled by Eq. 26 with updates given by $\boldsymbol{c} \leftarrow \boldsymbol{c} - 2\,\boldsymbol{G}\,\nabla_{\boldsymbol{c}} E_p(\boldsymbol{c})$, which keeps the inferred context in the span of the stored context vectors. A new transition gets 64 settling steps, followed by one sweep over the previously observed transitions with 4 settling steps per transition, so that the context latent integrates the evidence of the whole prefix instead of updating solely based on whichever transition it saw last. The episode is then recalled by the decode relay under the settled context latent and scored token by token. The state never grows with K or L: the context latent, the outer-product matrix, and the mean are $D + D^2 + D$ numbers.

The ideal observer. The reference knows which stored episodes are consistent with the revealed prefix and commits to one of them uniformly at random. Its expected recall is the mean token overlap between the consistent episodes and the target. On nested examples it is an upper bound.

Results. The top panel of Figure 4 shows a single example, and the bottom panel shows the mean recall over 96 evaluation examples. In particular, we use 32 8-episode examples from models trained under 3 seeds (i.e., $32 \cdot 3 = 96$ evaluation examples). Under the true context vector every episode is recalled exactly. Under a zero context vector recall is at the floor, at 0.093 for nested examples.

D.4 EVALUATION DETAILS

Reported scores are computed on the test split for each task. Models are trained on a training split and early stopping is applied based on a validation set.

Write-rule comparison (Table 2). The task is single-episode sequential recall at $L = D = 256$ (Appendix C.1). We evaluate under three random seeds. Each write rule starts from an empty memory and at each position, we evaluate the feedforward key $\boldsymbol{k}_t = R(\boldsymbol{W}_k\ \mathrm{enc}(\boldsymbol{x}_{t-1}), t)$ and value $\boldsymbol{v}_t = \mathrm{enc}(\boldsymbol{x}_t)$. The three rules are the outer product $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$, the delta rule $\boldsymbol{S}_t = \boldsymbol{S}_{t-1} + (\boldsymbol{v}_t - \boldsymbol{S}_{t-1}\boldsymbol{k}_t)\boldsymbol{k}_t^\top / \|\boldsymbol{k}_t\|^2$ with a unit step, and RLS (Equation 7). In the *untrained* systems every learning rate is zero, so the encoder, decoder, key map and RoPE frequencies keep their initial values. The *trained* systems are three seeds at $L = D = 256$, and they were trained with the RLS write. For these, we swap the write rule at test time and leave the learned maps unchanged.

Recall uses the same free-running relay of Appendix C.1, with one change: it does not decode through dec. An untrained decoder is a random map, and it will perform at chance accuracy (0.10) even when handed the correct stored value. So we match each memory read by cosine similarity to the nearest of the $V = 10$ encoded tokens $\mathrm{enc}(\boldsymbol{x}_j)$ and feed that token's observation forward to form the next key. This clean-up step is the classical readout of a linear associative memory (Kohonen, 1972). We use it in both columns so that the comparison isolates the write rule and the learned address code. We report the mean and sample standard deviation over the three seeds.

Comparison with DeltaNet (Table 3). Both models see the same MQAR streams (Appendix C.3) at width 256 and are scored on the 192,000 test queries. Model predictions are compared to the vocabulary entries. For DeltaNet-PC this means finding the nearest codebook row to the decoded output. For DeltaNet it means the argmax of the logits.

DeltaNet. We train the DeltaNet of Yang et al. (2024b) as implemented in Zoology's code (Arora et al., 2023) with the `fla` kernels.[7] The model has two layers, two heads of width 128, short convolutions, no MLP, and $d_{\text{model}} = 256$. Its learned token embedding is tied to the output layer. Training follows Zoology's code: AdamW with weight decay 0.1, a cosine schedule over 64 epochs, batch 128, and learning rate 4.6×10^{-4}.

DeltaNet-PC. We use our default hyperparameters (Table 6) with any MQAR deviations listed in Appendix E. The answer at a labelled position t is decoded from $\boldsymbol{S}_{t-1}\boldsymbol{k}_t$, which is before position t is written. During testing, each write first runs a 16-step settle of the value latent on Equation 21, with the stream token clamped at the output, the key at its feedforward value, and the memory update re-solved in closed form at every step.

Training streams to exact recall. Recall that we evaluated how few training streams are needed for exact recall. We train each model on the first n of Zoology's training streams and score it on the full test set. DeltaNet still trains for 64 epochs of 782 batches of 128, with batches drawn from the n streams. Its test score is taken at the epoch with the best accuracy on our 128-stream validation split.

Locality. A rule is temporally local when no gradient crosses positions. In DeltaNet-PC, the write depends only on $\boldsymbol{k}_t$, $\boldsymbol{v}_t$, $\boldsymbol{S}_{t-1}$ and $\boldsymbol{P}_{t-1}$, and each parameter gradient depends only on the settled prediction errors at the current position. A rule is spatially local when each weight's update is the product of the activity and the prediction error at the two ends of that weight. The one exception is LayerNorm's mean and variance, which pool over the units of a layer. DeltaNet is trained by backpropagation through time and through its layers, so its rule is neither local in time nor in space.

Parameters and state. Parameters count every trained weight. DeltaNet's 0.53 M excludes its $8{,}192 \times 256$ token embedding (2.10 M, with 2.63 M in total). DeltaNet learns its embedding and DeltaNet-PC uses a fixed codebook. DeltaNet-PC's 197,376 parameters are $\boldsymbol{W}_{\text{enc}}$, $\boldsymbol{W}_k$ and the decoder weight (256×256 each), plus the decoder bias and LayerNorm. There are no RoPE frequencies because RoPE is off on MQAR. State counts fast-weight floats. For DeltaNet this is two layers times two heads of 128×128, or 65,536. For DeltaNet-PC it is $\boldsymbol{S}$ plus the RLS inverse $\boldsymbol{P}$, or $2 \times 256^2 = 131{,}072$. The value accumulator $\boldsymbol{C}^v$ of Equation 18 is an optional $D \times D$ matrix that can be used by the learn pass and the test-time settle, but is not needed (see Appendix F).

Peak training memory. This is the peak GPU memory of one training step at batch 32, on random streams in the MQAR layout with the stream length L varied. For DeltaNet we record `torch.cuda.max_memory_allocated` over the forward pass, the backward pass and the AdamW step. We measure a leaner variant that reads our fixed codebook instead of the embedding, computes logits only at the 64 labelled positions, and uses no activation checkpointing. For DeltaNet-PC we record JAX's peak bytes in use during training, learn pass included. The trainer processes the stream in chunks of 512 positions and carries the $D \times D$ state across chunks.

D.5 Compute resources

Experiments ran on a combination of a single Nvidia RTX 4090 (24 GB) for development and primarily Nvidia H100 (80GB and 94GB variants) GPUs for multi-run experiments involving a range of configurations and 3 random seeds, though some V100s were also used. Individual runtimes range from a few minutes to about 5.5 hours depending on the size of the model and difficulty of the task, but median runtimes are about 10 minutes.

Our work is built on JAX (Bradbury et al., 2018) and Equinox (Kidger & Garcia, 2021), and DeltaNet-PC runs in single precision.

[7] https://github.com/HazyResearch/zoology

E HYPERPARAMETERS

Table 6 lists the default hyperparameter values used throughout the paper. We made an effort to use a single set of hyperparameters across all experiments, not only to demonstrate generality of the recipe, but also to ease future development efforts for any researchers who wish to build on this work.

Here we list any deviations from the default hyperparameters.

MQAR. MQAR uses 64 settling steps on the learn pass during training. At test-time, it uses 16 settling steps on the value latent and memory writes when updating the memory, in contrast to the other experiments that perform no settling when writing to memory at test time. We disable RoPE for MQAR, consistent with the Zoology DeltaNet implementation.

Omniglot. Omniglot uses 64 settling steps on the learn pass during training.

F ABLATIONS

We remove one component of the full recipe at a time and retrain from scratch on each task, holding everything else fixed. Each ablation is a single seed, and the full recipe row is the task's reference. Table 7 reports test accuracy.

Table 7: Ablations of several system components—each row removes one component from the full recipe. Test accuracy on sequential recall ($L = D = 256$), multi-episode recall ($K = 8 \times 32$), MQAR ($D = 256$) and Omniglot-MNIST binding with 128 keys. On Omniglot the full recipe makes 2 errors in 16,384 probes and the variant without the clip makes 1. The official MQAR row runs with RoPE off, so freezing it does not apply there.

	Sequential	Multi-episode	MQAR	Omniglot
Full recipe	1	1	1	0.9999
No activity clip	1	1	1	0.9999
No value orthogonality ($\lambda_v = 0$)	1	1	1	0.926
NLMS write (no interference term)	0.282	0.178	0.978	0.807
Frozen RoPE	0.359	0.689	–	0.940

No activity clip. The settling gradient is not clipped. The recipe clips each of its components to magnitude 2. Removing it has no effect on task performance. We nevertheless keep the clip to safeguard against loss spikes.

No value orthogonality. $\lambda_v = 0$, which removes $\mathcal{R}_t^{\boldsymbol{v}}$ and its leave-same-item-out correction from the energy (§B.3). This regularizer has no impact on the token-based tasks but results in an approximately 7-point reduction in accuracy on Omniglot, consistent with the idea that image data is more correlated than the token codebook rows. It is therefore an optional regularizer that may or may not be needed but we keep it active in all experiments for consistency.

NLMS write. The stored-pairs term $\frac{1}{2}\operatorname{Tr}(\Delta \boldsymbol{S}_t \boldsymbol{C}_{t-1} \Delta \boldsymbol{S}_t^\top)$ is removed from the energy. The minimizer of what remains is $\Delta \boldsymbol{S}_t = \boldsymbol{r}_t \boldsymbol{k}_t^\top / (\epsilon + \|\boldsymbol{k}_t\|^2)$, the normalized delta rule of Table 2 with unit step and regularization constant ϵ. The write keeps its ridge but no longer protects past keys. We observe a drop in accuracy on all tasks, which provides further evidence for the value of the RLS objective.

Frozen RoPE. The rotary frequencies stay at their initialization, the standard RoPE schedule with base 10^4, instead of being learned. MQAR does not use RoPE, consistent with Yang et al. (2024b)'s implementation. For the remaining tasks, we observe an intuitive drop in accuracy. The single episode sequential task is heavily reliant on position because sequences are of length 256 and therefore, keys must learn to orthogonalize to avoid collisions and rotating key planes is a mechanism for achieving this. The multi-episode sequential task relies on episodes of length $\ell = 32$ and is slightly less reliant

on learned rotation frequencies. Finally, Omniglot is a position-invariant task, and therefore even less reliant on learning fast rotation frequencies and the standard RoPE initialization is already largely sufficient.

G Compute Cost

We compare compute on MQAR, the one task on which both DeltaNet-PC and DeltaNet are implemented. Each stream has $L = 512$ tokens and $P = 64$ queries. The width is $D = 256$, the vocabulary has $V = 8{,}192$ tokens, and each token is D-dimensional. We count the floating-point operations for one stream, counting a multiply-add as two. For DeltaNet-PC we sum the operations of Algorithm 1 over the positions of the stream. For DeltaNet we count the recurrent form of its two layers and take the backward pass to cost twice the forward pass. Table 8 summarizes the comparison.

Table 8: Compute per MQAR stream ($L = 512$, $P = 64$ queries, $D = 256$, $V = 8{,}192$, $T = 64$) and measured time on one H100. DeltaNet-PC's test figures in parentheses include the 16-step test-time settle, e.g., 0.75 for no test-time settle, and 4.0 with a 16-step test-time settle. Training times are per stream at batch 32 for DeltaNet-PC and 128 for DeltaNet.

	DeltaNet-PC	DeltaNet
Training, operations	$O(LD^2 + PTD^2)$, 3.2 GFLOP	$O(L(D^2 + DV))$, 8.7 GFLOP
Test, operations	$O(LD^2 + PDV)$, 0.75 (4.0) GFLOP	$O(L(D^2 + DV))$, 2.9 GFLOP
Training, time	18 ms	0.42 ms
Test, time	3 (12–20) ms	0.8 ms
To exact recall	1,024 streams, 3.3 TFLOP, 18 s	200,192 streams, 1.7 PFLOP, 88 s

Operations. In training, DeltaNet-PC writes every position with the RLS update, at $O(D^2)$ per position. At each of the P queries it also runs a T-step learn pass, in which each step is a memory read, a decoder pass and the gradient of that pass, for $O(TD^2)$ per query. A training stream therefore costs $O(LD^2 + PTD^2)$, and with $T = 64$ the learn pass dominates. DeltaNet costs $O(D^2)$ per token in each layer, plus $O(DV)$ per token for its output logits, and backpropagation through time multiplies this by a constant, for $O(L(D^2 + DV))$ per stream. At test time, DeltaNet-PC writes every position at $O(D^2)$ and decodes each query against the V codebook rows at $O(DV)$, for $O(LD^2 + PDV)$ per stream. The 16-step test-time settle adds $O(16\,LD^2)$. DeltaNet runs its forward pass, at $O(L(D^2 + DV))$. Per stream, the two models are within a small factor of each other in both training and test. The large difference is in how many training streams each model needs. DeltaNet-PC is exact after 32 updates of 32 streams, about 3 TFLOP in total. DeltaNet's fastest route to exact recall is two epochs of the 100,000 training streams, which requires about 1.7 PFLOP.

Wall-clock. The operation counts do not carry over to wall-clock time on a GPU. On one H100, DeltaNet-PC trains at 18 ms per stream and DeltaNet at 0.42 ms per stream, even though a DeltaNet-PC training stream takes fewer operations. Our implementation scans the stream one position at a time, and at each query the learn pass is a chain of T dependent matrix-vector products on a batch of 32. The time to exact recall is nonetheless shorter for DeltaNet-PC: 18 s of training for DeltaNet-PC and 88 s for DeltaNet. These times describe our current implementation and future work could improve our DeltaNet-PC code to be more parallel (e.g., chunked updates).

H tPC reference

The tPC capacity curve was produced by training tPC at $L = 8$, $D = 256$, with a unique BOS token per training sequence. This is tPC's most favorable condition for memorization: each sequence has its own indexing token, so the slow weights can in principle dedicate distinct capacity to each. We were unable to reliably train tPC at longer L. The plot shows recall degrading as N grows, exposing the slow-weight capacity ceiling. We do not present this as a direct comparison against DeltaNet-PC: the two systems target different problems—tPC memorizes a fixed set of training sequences in the slow weights and DeltaNet-PC binds arbitrary unseen sequences into $\boldsymbol{S}$ at test time.

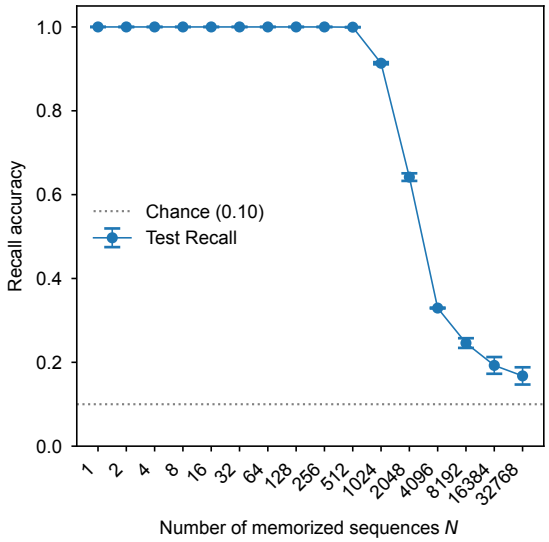


Figure 8: tPC's intrinsic memorization ceiling. Test recall accuracy on the N training sequences as a function of N, with a unique BOS token per training sequence. $L = 8$, $D = 256$.

Symbol	Name	Value	Description
Task			
L	Sequence length	256	Positions per sequence (swept in Figure 3).
V	Vocabulary size	10	Distinct tokens, each a fixed Gaussian code of dimension d_{obs}. 8,192 for MQAR.
d_{obs}	Observation dimension	256	Codebook row dimension (images: pixel count).
n_{train}	Training sequences	1024	Unique training sequences.
n_{test}	Test sequences	128	Held-out sequences. The validation split is the same size.
B	Batch size	32	Sequences per parameter update.
Architecture			
D	Latent and memory width	256	Dimension of $\boldsymbol{k}_t$ and $\boldsymbol{v}_t$, and the side of $\boldsymbol{S}_t, \boldsymbol{P}_t \in \mathbb{R}^{D\times D}$.
$\boldsymbol{W}_{\text{enc}}$	Encoder	1	Linear layers of enc (bias-free, $\mathbb{R}^{d_{\text{obs}}} \to \mathbb{R}^{D}$).
$\boldsymbol{W}_{\text{dec}}$	Decoder layers	1	LayerNorm then one linear layer $\mathbb{R}^{D} \to \mathbb{R}^{d_{\text{obs}}}$.
$\boldsymbol{W}_k$	Key map init	identity	Initialization of the square key map $\boldsymbol{W}_k$ (the address starts at enc's geometry).
b	RoPE base (init)	10000	Initialization of the $D/2$ learnable frequencies, $\theta_i = b^{-2(i-1)/D}$.
Memory write			
ϵ	Ridge	0.25	Prior precision on $\boldsymbol{S}$ (Eq. 2), the RLS ridge.
T_{write}	Settling steps at the write	0	The write uses the feedforward $\boldsymbol{k}_t$, $\boldsymbol{v}_t$ (exact minimizer of Eq. 5).
$T^{\text{test}}_{\text{write}}$	Settling steps at the write (test)	0	Same at evaluation (a 16-step test settle only where noted).
Learn pass			
T	Settling steps	16	Gradient steps per position on the learn-pass energy, over $(\tilde{\boldsymbol{k}}_t, \tilde{\boldsymbol{v}}_t)$.
η_x	Settling step size	0.02	Gradient-descent step on the latents during settling.
–	Activity gradient clip	2	Bound on each component of the latent gradient (elementwise, so each unit clips its own error).
λ_{ev}	Evidence precision	1	Weight of $\frac{1}{2}\|\boldsymbol{v}_t - \text{enc}(\boldsymbol{x}_t)\|^2$, which trains enc.
λ_k	Address-prior precision	1	Weight of $\frac{1}{2}\|\tilde{\boldsymbol{k}}_t - R(\boldsymbol{W}_k\ \text{enc}(\boldsymbol{x}_{t-1}), t)\|^2$, which trains $\boldsymbol{W}_k$ and R.
Regularizers			
λ_v	Value orthogonality	10	On $\mathcal{R}^{\boldsymbol{v}}_t$ (Eq. 18).
Parameter learning			
η_θ	Learning rate (enc, dec)	0.00025	Gradient descent on $\boldsymbol{W}_{\text{enc}}$, $\boldsymbol{W}_{\text{dec}}$ and LayerNorm, one step per batch of B.
η_k	Learning rate ($\boldsymbol{W}_k$)	0.05	SGD on the key map.
η_R	Learning rate (RoPE)	0.001	SGD on the frequencies, projected onto $[0, \pi]$ after each step.
L_{ref}	Position-mean reference	256	Every parameter gradient ($\boldsymbol{W}_{\text{enc}}$, $\boldsymbol{W}_{\text{dec}}$, LayerNorm, $\boldsymbol{W}_k$, R) is the mean over a sequence's queries times L_{ref}, then its group's rate applies.
D_{ref}	Width-scale reference	256	The gradient of $\boldsymbol{W}_{\text{dec}}$ alone is further scaled by $\min(1, (D_{\text{ref}}/D)^p)$.
p	Width-scale power	2	Exponent of the width scale ($\boldsymbol{W}_{\text{dec}}$ only).
Early stopping			
–	Metric	recall_accuracy	Validation recall (binding accuracy on the key-value rows).
–	Interval	1	Every N epochs (every 32 batches on the official MQAR).
–	Patience	25	Evaluations without improvement (cap 400 epochs). The best checkpoint is tested.

Table 6: DeltaNet-PC default hyperparameters.