PARATTT: PARALLELIZING TEST-TIME TRAINING LAYERS WITH DEEP MEMORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time training (TTT) layers replace the linear fast-weight memory of modern recurrent architectures with a small neural network trained online by gradient descent. This nonlinear memory is more expressive than the linear memories of Gated DeltaNet and Mamba-2, but is difficult to parallelize: existing implementations such as TTT-MLP rely on sequential, block-wise weight updates that run several times slower than chunkwise-parallel linear recurrences. Moreover, their block-wise execution only approximates the token-by-token online recurrence, whereas linear recurrences admit an exact chunkwise-parallel evaluation. We introduce ParaTTT (PTTT), an algorithm for parallelizing arbitrarily deep nonlinear memories by iteratively refining estimates of the inner-loop gradient trajectory. Starting from a guess of the hidden-layer gradients, each refinement consists of three chunkwise-parallel passes: a forward pass that evaluates the resulting feature trajectory, an exact gated-delta-rule pass that computes the readout trajectory and its feature gradients, and a backward pass that propagates these gradients into a new estimate. The fixed point is the exact online gradient descent recurrence, while decoding requires no refinement iterations. We prove geometric convergence for a two-layer nonlinear memory under sufficient contraction conditions. In practice, two iterations suffice. With two refinement steps at 4K context, PTTT evaluates the inner-loop trajectory $32\times$ faster than TTT-MLP with the block approximation at matched error, while the full 340M model trains $4.9\times$ faster. Models trained at 340M and 1.3B parameters match or exceed Gated DeltaNet and Mamba-2 and remain competitive with same-size Transformers on language-modeling perplexity and zero-shot accuracy, with better recall than the linear recurrences.

1 INTRODUCTION

Softmax attention scales poorly to long contexts: its compute grows quadratically with sequence length, while its key–value cache grows linearly. Recurrent language models address both problems by processing the sequence in linear time and carrying a fixed-size state while decoding. Moreover, modern recurrent architectures such as gated linear attention, Gated DeltaNet, and Mamba-2 (Yang et al., 2024a; 2025; Dao & Gu, 2024) admit an efficiently parallelizable structure commonly referred to as the chunkwise-parallel form, in which tokens within each chunk are processed in parallel using dense matrix multiplications, while only the recurrent state is propagated sequentially across chunks. Given this scalable structure, the central question for this class of models is how to make the recurrent memory more expressive without sacrificing parallel evaluation.

Test-time training (TTT) addresses the expressivity side of this question by treating the recurrent state as the weights of a small model that learns from the sequence through online gradient descent (Sun et al., 2025). With a linear memory and a squared reconstruction loss, this recovers the delta rule of Widrow & Hoff (1960), the update used by DeltaNet (Schlag et al., 2021; Yang et al., 2024b), whereas a nonlinear memory can adapt its features as well as its readout. TTT-MLP, Titans, and LaCT (Sun et al., 2025; Behrouz et al., 2025b; Zhang et al., 2026) explore such neural memories, but their gradients depend nonlinearly on the current weights: each token changes the weights at which the next token's gradient should be evaluated, creating a sequential dependency. To recover parallelism, existing methods group gradient evaluations into blocks and evaluate every gradient in a block at the weights entering it. This creates a fidelity–speed tradeoff: larger blocks reduce the

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.

sequential depth but more coarsely approximate token-by-token learning, while smaller blocks are more faithful but require more sequential steps.

In this paper, we introduce ParaTTT (PTTT), an algorithm that evaluates an MLP-based TTT memory in parallel. The difficulty in parallelizing the recurrence is the inner weight trajectory: the gradient step at each token is taken at weights that depend on every earlier step, so the trajectory cannot be computed without stepping through it sequentially. PTTT instead refines an estimate of this trajectory, represented by the gradients of the inner loss with respect to the hidden-layer pre-activations. Each refinement consists of three parallel passes: a forward pass that constructs the features implied by the current gradient estimate, a readout pass that computes each token's loss gradient with respect to its features, and a backward pass that propagates these gradients through the hidden layers to form a new estimate. Crucially, each refinement step is a composition of existing parallel recurrences such as Gated DeltaNet (Yang et al., 2024b; 2025), Mamba-2 (Dao & Gu, 2024), or gated linear attention (Yang et al., 2024a), yet the resulting iteration provably converges to the exact nonlinear sequential recurrence.

Our technical contributions can be summarized as follows:

- We introduce **ParaTTT (PTTT)**, an algorithm for parallelizing MLP-based recurrent memories of arbitrary depth, and prove geometric convergence for a two-layer nonlinear memory under sufficient conditions (Section 3).
- We demonstrate PTTT's superior speed–accuracy trade-off over the block-based evaluation of TTT-MLP: at 4K context, two PTTT refinements match the output error of TTT-MLP's block approximation while running $32\times$ faster, and full 340M models train $4.9\times$ faster than TTT-MLP (Sections 4.2 and 4.3).
- At 340M and 1.3B parameters, PTTT attains lower perplexity and stronger recall on SWDE and FDA than Gated DeltaNet and Mamba-2, demonstrating the benefit of a nonlinear memory over linear recurrent baselines (Section 4.1).

2 Background

We write $\mathbf{x}_{1:T}$ for the input sequence of a layer and $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$ for the per-token query, key, and value projections; the fast-weight state is a matrix $\mathbf{S}_t$ or, for a deep memory, a set of weights $\mathbf{W}_t$; C is the chunk size of the parallel kernels. Heads are independent, and we suppress the head index. Section A extends this section.

2.1 Linear recurrences

Linear recurrences retain the associative-memory view of attention while replacing its growing key–value cache with a fixed-size state. Linear attention (Katharopoulos et al., 2020; Schlag et al., 2021) does this by dropping the softmax: it maintains a matrix state updated by an outer product, $\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t\mathbf{k}_t^\top$, and reads it with the query, $\mathbf{o}_t = \mathbf{S}_t\mathbf{q}_t$. Because this state only accumulates, practical variants introduce forgetting. Gated linear attention (Yang et al., 2024a) decays the state before the write, $\mathbf{S}_t = \alpha_t\mathbf{S}_{t-1} + \mathbf{v}_t\mathbf{k}_t^\top$, using a diagonal gate, while Mamba-2 (Dao & Gu, 2024) is the scalar-gated case. DeltaNet (Schlag et al., 2021; Yang et al., 2024b) updates the state by replacing the additive write with one step of gradient descent on $\frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$, $\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top$, which erases the value stored along $\mathbf{k}_t$ before writing the new one. Gated DeltaNet (Yang et al., 2025) combines this delta-rule update with decay. All of these updates are linear in the recurrent state. Consequently, the state at the end of a chunk of C tokens is an affine function of the state entering the chunk. This allows tokens within each chunk to be evaluated in parallel with dense matrix multiplications, while only the chunk-boundary states are propagated sequentially with a short scan. This exact evaluation is commonly referred to as the chunkwise-parallel form.

2.2 Test-time training

Test-time training (Sun et al., 2025) makes the memory more expressive than a single matrix by interpreting the recurrent state as the parameters $\mathbf{W}_t$ of a small model $f_{\mathbf{W}}$ trained online. At token t, the model takes a gradient step on the inner loss $\ell_t(\mathbf{W}) = \frac{1}{2}\|f_{\mathbf{W}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$ and produces the

layer output $f_{\mathbf{W}_t}(\mathbf{q}_t)$. The projections, gates, and learning rates that define this inner problem are trained end-to-end by backpropagating through the inner updates.

TTT-linear. The simplest choice of inner model recovers the delta-rule recurrence described above. With $f_{\mathbf{W}}(\mathbf{k}) = \mathbf{W}\mathbf{k}$, one gradient step on the squared reconstruction loss is exactly the DeltaNet update, while adding a decay term gives Gated DeltaNet. Because this update remains linear in the recurrent state, it retains the exact chunkwise-parallel form described above. We exploit this identity for the linear readout of a nonlinear memory.

TTT-nonlinear. A nonlinear inner model can adapt the features it reads and writes with, rather than only the readout; this additional expressivity motivates deep memories. TTT-MLP (Sun et al., 2025), for example, uses a two-layer MLP as $f_{\mathbf{W}}$. Exact online gradient descent evaluates each token's gradient at the weights produced by all preceding updates, which for a nonlinear memory prevents the chunkwise-parallel rearrangement of the linear case, so TTT-MLP uses a mini-batch (dual) approximation: for a block of b tokens starting at position s, every gradient is evaluated at the weights $\mathbf{W}_s$ entering the block,

$$\text{exact: } \mathbf{W}_j = \mathbf{W}_{j-1} - \eta_j \nabla \ell_j(\mathbf{W}_{j-1}), \qquad \text{block: } \mathbf{W}_t^{\text{blk}} = \mathbf{W}_s - \sum_{j=s+1}^{t} \eta_j \nabla \ell_j(\mathbf{W}_s), \quad s < t \le s + b.$$

The causal partial sum lets tokens within the block be evaluated in parallel, but the gradients are stale: updates from earlier tokens in the block do not change the weights at which later gradients are evaluated. Blocks remain sequential, requiring T/b steps, so larger blocks reduce sequential depth at the cost of a coarser approximation to token-by-token learning. Titans (Behrouz et al., 2025b) adds momentum and weight decay under the same block form, while LaCT (Zhang et al., 2026) uses blocks of thousands of tokens together with a sliding-window attention branch for local context.

2.3 Other nonlinear recurrent approaches

Sequential nonlinear RNNs. Classic recurrent networks (Elman, 1990; Hochreiter & Schmidhuber, 1997; Cho et al., 2014) carry a vector state through a nonlinear map. Their standard evaluation is sequential over tokens, limiting parallelism across the sequence. This motivates attention (Vaswani et al., 2017) and parallel linear recurrences, but the choice of transition also affects expressivity. Under their precision and architectural assumptions, Merrill & Sabharwal (2023); Merrill et al. (2024) place transformers and certain SSMs in TC^0. This rules out general permutation composition if $\mathrm{TC}^0 \neq \mathrm{NC}^1$. The xLSTM family (Beck et al., 2024) includes both a sequential sLSTM cell and a parallelizable mLSTM cell. More recently, M^{2}RNN (Mishra et al., 2026) uses a nonlinear transition with a matrix-valued state, but pays for it with sequential forward and backwards passes.

Parallelizing nonlinear recurrences. DEER (Lim et al., 2024) and ParaRNN (Danieli et al., 2026) cast a nonlinear recurrence $\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t)$ as a fixed-point problem over the state trajectory and solve it with Newton iterations, where each iteration reduces to a linear recurrence evaluated by a parallel scan; deep equilibrium models (Bai et al., 2019) share the fixed-point perspective. A generic Newton step for a state of dimension m involves an $m \times m$ transition Jacobian, and ParaRNN makes this tractable by designing recurrent cells with structured Jacobians. Instead of iterating on the fast-weight trajectory itself, ParaTTT iterates on the per-token hidden-layer pre-activation gradients, whose dimension is only the hidden width, d, compared to the $O(d^2)$ size of the full state. Conditioned on these gradients, the fast-weight trajectory becomes a linear recurrence, so each refinement can be evaluated with chunkwise linear recurrent kernels without ever forming the full weight trajectory or its transition Jacobian.

3 Method

3.1 Test-time training with a deep memory

We consider a single head and omit the head index. As in Gated DeltaNet, the input token $\mathbf{x}_t$ is mapped to a key, a query and a value, $\mathbf{k}_t, \mathbf{q}_t \in \mathbb{R}^d$ and $\mathbf{v}_t \in \mathbb{R}^{d_v}$, by linear projections followed by a short causal convolution; keys and queries are RMS-normalized, $\|\mathbf{k}_t\| = \|\mathbf{q}_t\| = \sqrt{d}$. The token also produces a decay gate $\alpha_t \in (0, 1]$ and, for each layer l of the memory defined below, a learning rate $\eta_t^{(l)} \in (0, 1)$. The gate follows Yang et al. (2025), each $\eta_t^{(l)}$ is the sigmoid of a learned linear function of $\mathbf{x}_t$, and all layers of the memory share the decay.

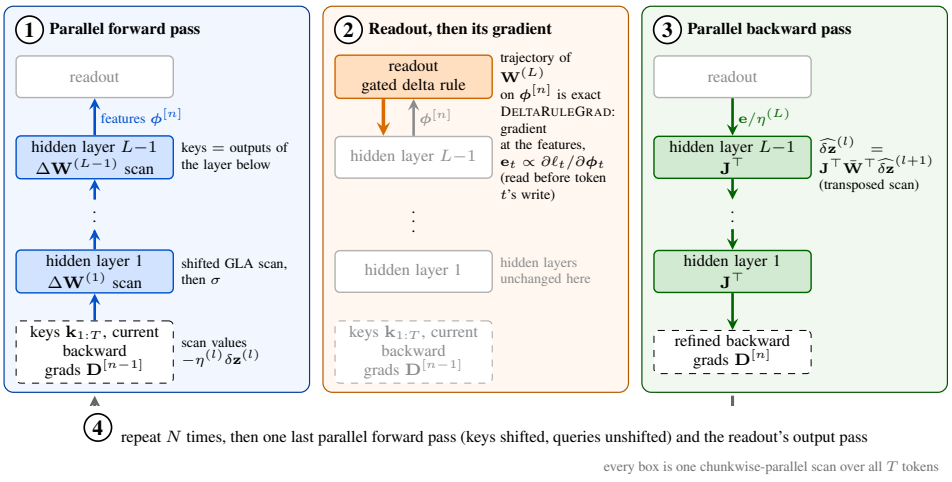


Figure 1: One iteration of ParaTTT; every box is a chunkwise-parallel scan over all T tokens. (1) A parallel forward pass evaluates the hidden layers from the current estimate of their pre-activation gradients. (2) The readout runs exactly on the resulting features and DELTARULEGRAD returns each token's feature gradient. (3) A parallel backward pass propagates it through the hidden layers with transposed scans and updates the estimate. (4) After N iterations a final forward pass and readout produce the output.

Memory. The recurrent state is the set of weights $\mathbf{W} = (\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)})$ of an L-layer MLP. For given weights $\mathbf{W}$, with pre-activations $\mathbf{z}^{(l)}$ and activations $\mathbf{h}^{(l)}$, the memory computes

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{h}^{(l)} = \sigma\big(\mathbf{z}^{(l)}\big) \quad (1 \le l < L), \quad \mathbf{h}^{(0)} = \mathbf{k}, \quad f_{\mathbf{W}}(\mathbf{k}) = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)}, \tag{1}$$

where σ is a pointwise nonlinearity and $\mathbf{b}^{(l)}$ are learned biases. The final linear layer is the *readout*, and its input $\boldsymbol{\phi}_t = \mathbf{h}_t^{(L-1)}$ defines the *features*. The features are additionally projected onto the unit ball, $n(\mathbf{x}) = \mathbf{x}/\max(1, \|\mathbf{x}\|)$, which we absorb into the last σ. This projection is used in all experiments. Our experiments use $L = 2$, as in Sun et al. (2025), but the method applies to any L. Each weight is decomposed as $\mathbf{W}_t^{(l)} = \mathbf{W}_0^{(l)} + \Delta\mathbf{W}_t^{(l)}$, where $\mathbf{W}_0^{(l)}$ is a parameter trained by backpropagation and $\Delta\mathbf{W}_t^{(l)}$ is the recurrent state, initialized to zero at the start of every sequence.

Sequential recurrence. Token t first decays the state and then takes one gradient step on the reconstruction loss $\ell_t(\mathbf{W}) = \frac{1}{2}\|f_{\mathbf{W}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$, with the layer's learning rate as the step size. For each layer l,

$$\bar{\mathbf{W}}_{t-1}^{(l)} = \mathbf{W}_0^{(l)} + \alpha_t\,\Delta\mathbf{W}_{t-1}^{(l)} \quad \text{(decay)}, \qquad \mathbf{W}_t^{(l)} = \bar{\mathbf{W}}_{t-1}^{(l)} - \eta_t^{(l)}\,\frac{\partial\ell_t}{\partial\mathbf{W}^{(l)}}\Big|_{\bar{\mathbf{W}}_{t-1}} \quad \text{(gradient step)}. \tag{2}$$

The layer output is $\mathbf{o}_t = f_{\mathbf{W}_t}(\mathbf{q}_t)$: the key is evaluated under the decayed weights, before its own gradient step, and the query after it, exactly as in Gated DeltaNet, whose memory is the linear case $L = 1$ of Eq. (1).

Equation (2) is the exact token-by-token recurrence that we seek to evaluate. Computing it directly requires T sequential gradient steps because the gradient at each token depends on all preceding updates. The block approximation used by TTT-MLP, described in Section 2.2, reduces this sequential depth by evaluating multiple gradients at shared weights, but changes the recurrence. Our goal is instead to parallelize the trajectory while retaining the exact online recurrence as the fixed point.

3.2 PARATTT: ITERATED REFINEMENT OF THE INNER-LOOP WEIGHT TRAJECTORY

We first describe the basic ParaTTT iteration, which uses a linear attention update for the hidden layers and an uncentered readout. The practical refinements used by our models are introduced in Section 3.4.

The central difficulty in parallelizing the nonlinear MLP recurrence in Eq. (2) is its inner weight trajectory: the gradient step at token t is taken at weights that depend on every earlier step, so the trajectory cannot be computed without stepping through it sequentially. ParaTTT instead iteratively

refines an estimate of this trajectory, with each refinement parallel across the sequence length. The estimate is represented by a guess of the pre-activation gradients $\delta\mathbf{z}_t^{(l)} = \partial\ell_t/\partial\mathbf{z}_t^{(l)}$ of the hidden layers. Given such a guess, a *parallel forward pass* evaluates the weight trajectory these gradients define (Step 1), the readout is evaluated exactly on the resulting features (Step 2), and a *parallel backward pass* propagates the resulting feature gradients through the trajectory to obtain a new guess (Step 3). At convergence, the gradients produced by the backward pass agree with those used in the forward pass (Fig. 1).

To see why each pass can be parallelized, write $\mathbf{h}_t^{(l)}$ for the activations of $\mathbf{k}_t$ under the decayed weights $\bar{\mathbf{W}}_{t-1}$ and obtain $\delta\mathbf{z}_t^{(l)}$ by backpropagation through Eq. (1) at $\bar{\mathbf{W}}_{t-1}$. The gradient step in Eq. (2) is the outer product of the two, so in terms of the state the recurrence reads

$$\Delta\mathbf{W}_t^{(l)} = \alpha_t\,\Delta\mathbf{W}_{t-1}^{(l)} - \eta_t^{(l)}\,\delta\mathbf{z}_t^{(l)}\,\mathbf{h}_t^{(l-1)\top} \qquad (l = 1, \dots, L). \tag{3}$$

Given a guess of the hidden-layer gradients $\mathbf{D} = (\delta\mathbf{z}_{1:T}^{(l)})_{l<L}$, one vector per token and hidden layer, Eq. (3) is linear in its state with known coefficients. Conditioned on $\mathbf{D}$, the nonlinear recurrence therefore reduces to a collection of linear recurrences. For a hidden layer, Eq. (3) is exactly gated linear attention with scalar decay, the state update of Mamba-2 (Yang et al., 2024a; Dao & Gu, 2024), with keys $\mathbf{h}_t^{(l-1)}$, values $-\eta_t^{(l)}\delta\mathbf{z}_t^{(l)}$ and gates α_t; for the readout, where $\delta\mathbf{z}_t^{(L)}$ is the readout's own error, it is a Gated DeltaNet (Yang et al., 2025). Both admit exact chunkwise-parallel kernels. The nonlinearity is thus confined to the self-consistency condition that $\mathbf{D}$ determines the state trajectory while the resulting trajectory determines $\mathbf{D}$. ParaTTT solves this fixed-point problem by iteration from $\mathbf{D} = \mathbf{0}$ (Fig. 1; pseudocode in Algorithm 1, Section C). Since the vectors of $\mathbf{D}$ enter the scans as values, we also call them *pseudo-labels*.

Step 1: parallel forward pass. Given the current estimate $\mathbf{D}$, the forward pass evaluates the memory MLP at all T tokens, each under its own decayed weights, layer by layer and without ever forming the fast weights. Unrolling Eq. (3) for a hidden layer $l < L$,

$$\alpha_t\Delta\mathbf{W}_{t-1}^{(l)}\,\mathbf{h}_t^{(l-1)} = -\sum_{j<t}\Big(\prod_{i=j+1}^{t}\alpha_i\Big)\eta_j^{(l)}\,\delta\mathbf{z}_j^{(l)}\,\big(\mathbf{h}_j^{(l-1)\top}\mathbf{h}_t^{(l-1)}\big), \tag{4}$$

which is gated linear attention (Yang et al., 2024a) with keys $\mathbf{h}_j^{(l-1)}$, values $-\eta_j^{(l)}\delta\mathbf{z}_j^{(l)}$ and gates α_j; the sum excludes $j = t$ because key t is read before its own write. We call a scan *shifted* when token t reads the state *before* its own update, $\bar{\mathbf{W}}_{t-1}$, and *unshifted*, the standard form, when it reads the state *after* the update, $\mathbf{W}_t$; keys are read shifted and queries unshifted. One chunkwise scan thus gives the fast contribution to the pre-activations of layer l at all T tokens; adding $\mathbf{W}_0^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}$ and applying σ gives the inputs of the next layer. The pass ends with the features $\boldsymbol{\phi}_{1:T}$ and stores the Jacobians $\mathbf{J}_t^{(l)}$ for Step 3. When $\mathbf{D} = \mathbf{0}$ on the first forward pass, the fast scan is identically zero and can be skipped.

Step 2: readout trajectory and its gradient. Unlike the hidden layers, the readout needs no trajectory estimate: it is a linear layer, and once its input features are fixed, its gradient-descent recurrence is exactly a Gated DeltaNet with keys $\boldsymbol{\phi}_t$, values $\mathbf{v}_t$, gates α_t and step sizes $\eta_t^{(L)}$, which the chunkwise gated-delta-rule kernel (Yang et al., 2024b; 2025) computes exactly.

The backward pass in Step 3 additionally requires each token's loss gradient with respect to its features. We obtain this during the readout pass:

$$\mathbf{e}_t = \eta_t^{(L)}\,\bar{\mathbf{W}}_{t-1}^{(L)\top}\big(\bar{\mathbf{W}}_{t-1}^{(L)}\boldsymbol{\phi}_t - \mathbf{v}_t\big) = \eta_t^{(L)}\,\partial\ell_t/\partial\boldsymbol{\phi}_t = -\,\bar{\mathbf{W}}_{t-1}^{(L)\top}\mathbf{u}_t, \tag{5}$$

where $\mathbf{u}_t = \eta_t^{(L)}(\mathbf{v}_t - \bar{\mathbf{W}}_{t-1}^{(L)}\boldsymbol{\phi}_t)$ is the update token t writes to the readout. The DELTARULEGRAD kernel computes this quantity as a small modification of the existing GDN kernel (Section C.3).

Step 3: parallel backward pass. The backward pass propagates each token's feature gradient through the hidden layers along the trajectory from Step 1, for all T tokens at once, giving a recomputed gradient trajectory $\hat{\mathbf{D}} = (\widehat{\delta\mathbf{z}}_{1:T}^{(l)})_{l<L}$. It requires the transposed Jacobian of σ and the transposed weights; the only operation involving a fast weight is the transpose of Eq. (4),

$$\big(\alpha_t\Delta\mathbf{W}_{t-1}^{(l)}\big)^\top\delta\mathbf{z}_t = -\sum_{j<t}\Big(\prod_{i=j+1}^{t}\alpha_i\Big)\eta_j^{(l)}\big(\delta\mathbf{z}_j^{(l)\top}\delta\mathbf{z}_t\big)\mathbf{h}_j^{(l-1)}, \tag{6}$$

a linear attention with the pseudo-labels as queries and keys and the layer inputs as values, computed by the same chunkwise scan. The estimate is then updated by a damped Picard step,

$$\mathbf{D}^{[n]} = \mathbf{D}^{[n-1]} + \lambda\big(\hat{\mathbf{D}} - \mathbf{D}^{[n-1]}\big), \qquad \lambda = 0.8. \tag{7}$$

Output. After N iterations, a final forward pass computes the features of the keys using shifted reads and of the queries using unshifted reads, since each query is evaluated under the updated weights. One gated-delta-rule pass with these features as keys and queries then produces the output. Section 3.3 shows that the exact sequential recurrence is a fixed point of this iteration and gives conditions under which the iterates converge to it.

Cost, training and decoding. Each iteration requires a constant number of chunkwise scans per memory layer, with sequential depth $O(T/C)$ for chunk size C (Section G). We use $N = 2$ unless stated otherwise; Section 4.3 shows empirically that two iterations already closely approximate the exact sequential recurrence. All parameters are trained by backpropagation through the N iterations. Decoding requires no iteration: with a single new token, Eq. (2) is one exact gradient step from the cached weights, so N affects prefill only.

3.3 Fixed point and convergence

Proposition 1 (Fixed point and finite-step exactness, informal)**.** *For any memory depth L, the gradient sequence $\mathbf{D}^\star$ of the sequential recurrence Eq. (2) is a fixed point of the iteration of Section 3.2, and the parallel evaluation at $\mathbf{D}^\star$ reproduces the sequential recurrence. Without damping ($\lambda = 1$), every iteration makes the estimate at one more token exact, so $N \geq T$ iterations reproduce the sequential recurrence exactly.*

Proposition 1, proved in Section B, needs no contraction assumption. For a geometric rate we analyze the basic iteration with $L = 2$, the linear-attention hidden update Eq. (4) and the uncentered readout, with the unit-ball projection replaced, for the analysis only, by the soft clip $n_{\mathrm{soft}}(\mathbf{x}) = \mathbf{x}/\sqrt{1 + \|\mathbf{x}\|^2}$, so that the features $\boldsymbol{\phi}_t = \psi(\mathbf{z}_t)$, $\psi = n_{\mathrm{soft}} \circ \sigma$, have a globally Lipschitz Jacobian (Section B restates the setting). Write $\|\mathbf{D}\|_\infty = \max_t \|\delta\mathbf{z}_t^{(1)}\|$ and F for one iteration's map $\mathbf{D} \mapsto \hat{\mathbf{D}}$.

Theorem 1 (Convergence of ParaTTT with soft clipping, $L = 2$)**.** *Assume $\|\mathbf{k}_t\| \leq 1$, $\|\mathbf{v}_t\| \leq V$, $\|\mathbf{W}_0^{(2)}\|_{\mathrm{op}} \leq w_0$, a shared decay $\alpha_t \leq \bar{\alpha} < 1$, a shared learning rate $\eta_t^{(1)} = \eta_t^{(2)} \leq \bar{\eta} \leq 1$, and $|\sigma'| \leq c_\sigma$, $|\sigma''| \leq h_\sigma$. With $\epsilon = \bar{\eta}/(1-\bar{\alpha})$ and $Q = V + w_0$, F is globally Lipschitz in $\|\cdot\|_\infty$ with constant*

$$\rho = \epsilon(1+\epsilon)Q^2\big[c_\sigma^2(1+\epsilon)(5+4\epsilon) + h_\sigma(2+\epsilon)\big] \;\leq\; (10c_\sigma^2 + 3h_\sigma)\,\epsilon Q^2 \quad \text{for } \epsilon \leq \tfrac{1}{4}. \tag{8}$$

If $\rho < 1$, F has a unique fixed point $\mathbf{D}^\star$, the gradient sequence of the sequential recurrence Eq. (2) with n_{soft} in place of the projection, and for any initialization and damping $\lambda \in (0, 1]$ the iteration Eq. (7) satisfies $\|\mathbf{D}^{[n]} - \mathbf{D}^\star\|_\infty \leq (1 - \lambda(1-\rho))^n \|\mathbf{D}^{[0]} - \mathbf{D}^\star\|_\infty$; the features, readout states and outputs converge at the same rate, up to constants.

The proof (Section B) controls the features and their changing Jacobians; SiLU satisfies the derivative assumptions, and $\rho < 1$ is independent of the sequence length. The bound does not cover the hard projection, the hidden delta rule or the centering of Section 3.4, so Section 4.3 measures the trained models' convergence directly.

3.4 Practical refinements

The analysis above uses the simplest form of the iteration. Our reported models use two refinements that substantially improve practical performance: a hidden-layer delta rule that preserves the target sequential recurrence, and centering, which changes the target to the sequential recurrence with the centered readout.

Delta rule for the hidden layers. Step 1 of the basic iteration evolves each hidden layer's fast weights by gated linear attention Eq. (4), which sums the writes $-\eta_j^{(l)}\delta\mathbf{z}_j^{(l)}\mathbf{h}_j^{(l-1)\top}$ with a scalar decay. The same weights can instead be evolved by a gated delta rule whose target is built from the

Table 1: Main results at 340M and 1.3B. Recall follows the generation protocol of Arora et al. (2024a); NIAH is the RULER needle average at the given length. Best per column and size in bold, second best underlined.

		Perplexity ↓		Zero-shot acc. ↑							Recall ↑				NIAH ↑			
Model	N	Wiki	Lamb	PIQA	Hella	ARC-e	Wino	ARC-c	BoolQ	SciQ	SWDE	FDA	TQA	DROP	4k	8k	16k	32k
340M, 24 layers, 15.7B tokens																		
GDN	–	26.57	29.66	65.2	36.5	47.1	51.5	24.1	56.6	78.4	32.5	17.8	35.7	15.3	25.9	17.0	13.7	9.3
Mamba2	–	26.99	45.87	65.7	35.1	45.5	51.9	23.4	57.8	75.2	25.8	5.7	30.6	12.6	8.1	3.0	1.1	1.2
Transformer	–	**25.34**	39.25	64.3	35.8	46.3	52.8	22.4	60.8	76.7	**54.6**	**28.4**	40.5	14.5	38.6	23.2	**16.3**	10.5
TTT-MLP	–	26.12	40.35	65.2	35.1	46.3	50.8	23.3	58.5	77.2	33.1	16.2	35.2	13.9	31.5	20.4	13.0	12.4
LaCT (no SWA, conv)	–	463.2	9.4×10^4	53.7	25.0	28.8	50.1	24.1	60.1	21.2	0.1	0.0	0.1	7.6	0.0	0.0	0.0	0.0
PTTT (ours)	2	25.46	26.93	65.3	**37.7**	48.6	51.2	24.1	59.8	79.0	41.9	20.3	39.6	14.4	32.6	20.7	15.6	**16.0**
no centering	2	26.06	32.62	64.5	36.8	48.4	50.6	**25.3**	58.2	78.9	36.6	19.6	37.0	14.3	29.0	21.1	15.0	12.2
linear-attention body	2	25.62	**25.88**	**65.8**	37.2	**48.7**	**53.2**	23.5	**61.0**	78.9	28.4	12.9	33.2	11.2	7.2	5.5	4.0	2.8
trained at N=5	5	25.47	28.44	64.9	37.6	46.2	51.4	24.5	58.2	**79.3**	45.0	22.0	**41.9**	**15.6**	**40.1**	**25.9**	15.9	5.8
1.3B, 24 layers, 100B tokens																		
GDN	–	15.78	10.56	72.2	54.9	60.0	**58.9**	28.1	53.9	86.7	59.0	27.0	52.9	**15.2**	50.8	34.1	18.8	**17.5**
Mamba2	–	15.42	11.62	**73.6**	53.5	58.0	54.1	28.2	53.9	87.7	56.7	21.5	**54.5**	14.8	33.4	19.6	11.3	4.9
Transformer	–	**15.04**	10.91	71.5	53.5	58.0	56.4	**29.9**	57.7	88.3	**75.3**	**31.9**	53.7	**15.2**	**78.1**	**56.7**	**43.4**	17.4
PTTT (ours)	2	15.21	**9.00**	71.9	**56.0**	**61.0**	58.6	**29.9**	**61.3**	**89.2**	62.8	28.1	53.7	15.0	53.8	37.1	17.5	14.7

previous iteration's read. The update is Eq. (3) plus a correction that erases stale writes along the iterates (Section C.2). This correction vanishes at the fixed point, so the target sequential recurrence is unchanged. The delta rule costs slightly more but gives substantially better recall (Table 1), and we therefore use it throughout.

Centering. The hidden layers receive gradient only through the readout. By Eq. (5), $\mathbf{e}_t$ is proportional to $\bar{\mathbf{W}}_{t-1}^{(L)\top}$, so a zero-initialized readout gives the hidden layers no gradient at the start of a sequence. Conversely, a nonzero $\mathbf{W}_0^{(L)}$ provides a nonzero gradient immediately, but also adds a persistent bias $\mathbf{W}_0^{(L)}\boldsymbol{\phi}_t$ to the output. We use *centering* to obtain a nonzero readout Jacobian at initialization while keeping the initial output zero. Centering trains $f_\theta - f_{\theta_0}$ in place of f_θ (Chizat et al., 2019); related zero-output initializations appear in Woodworth et al. (2020); Hu et al. (2020). We subtract from the readout its value on the features $\boldsymbol{\phi}_{0,t}$ of the slow MLP, that is, $f_{\mathbf{W}}(\mathbf{k}_t) = \mathbf{W}^{(L)}\boldsymbol{\phi}_t - \mathbf{W}_0^{(L)}\boldsymbol{\phi}_{0,t}$. At the start of a sequence, $\boldsymbol{\phi}_t = \boldsymbol{\phi}_{0,t}$ so the readout is zero, while its Jacobian $\mathbf{W}_0^{(L)} + \Delta\mathbf{W}^{(L)}$ is nonzero from the first token. Centering changes only the values passed to Step 2 and adds one term to the output (Section C.5).

Other choices. We use $L = 2$ with a hidden width of d, SiLU as σ, and a learned per-head gain on the fast part of the hidden pre-activation. Setting the hidden learning rates to zero yields a Gated DeltaNet on the features of the frozen MLP, the $N = 0$ control.

4 Empirical Study

Setup. All models share a backbone of 24 pre-norm residual blocks, each consisting of a token-mixing layer followed by a SwiGLU MLP, with $d = 1024$ at 340M parameters and $d = 2048$ at 1.3B, and a head dimension of 128. The token-mixing layer is ParaTTT, Gated DeltaNet, Mamba-2, attention, TTT-MLP or LaCT. We use short convolutions on the QKV projections, as in Gated DeltaNet (Yang et al., 2025). ParaTTT uses one hidden layer with width equal to the head dimension, together with the hidden-layer delta rule and centered readout of Section 3.4, and is trained with $N = 2$ iterations unless stated otherwise. For the main model-quality comparison, we remove LaCT's sliding-window attention branch to isolate the fast-weight memory; Section F.2 separately evaluates the published hybrid with sliding-window attention and discusses the role of this branch in prior TTT architectures. Section D gives every configuration.

4.1 Language Modeling Performance

Model set and evaluation. At 340M parameters, we compare Gated DeltaNet, Mamba-2, attention, TTT-MLP, LaCT and ParaTTT, together with two ParaTTT ablations (without centering and with linear attention in place of the hidden-layer delta rule) and a ParaTTT model trained with $N = 5$. At 1.3B parameters, we compare Gated DeltaNet, Mamba-2, attention, and ParaTTT. All models are trained on SlimPajama with the same training recipe, data order, and seed, for 15.7B tokens at 340M and 100B tokens at 1.3B (Section D). We evaluate perplexity on WikiText-103 and LAMBADA,

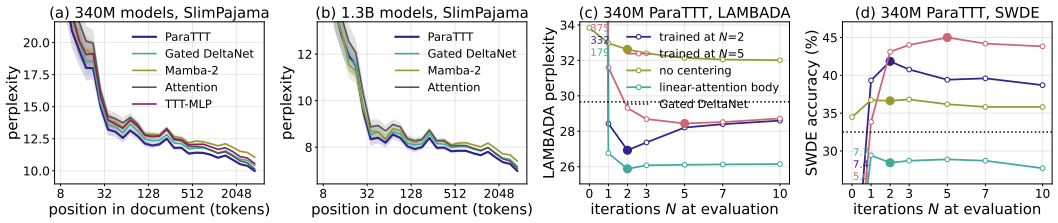


Figure 2: (a,b) Perplexity along 698 held-out SlimPajama documents at 340M and 1.3B. (c,d) LAMBADA perplexity and SWDE accuracy of the 340M ParaTTT models against the number of iterations at evaluation; filled marker: training N; dotted line: Gated DeltaNet; off-chart loop-off values are printed at the edge.

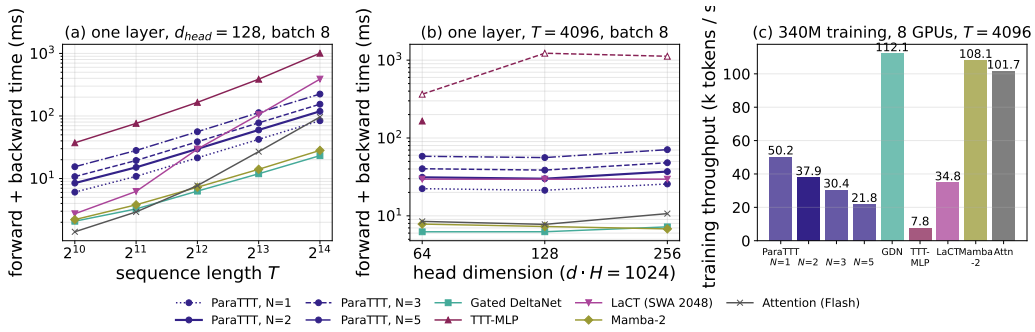


Figure 3: Efficiency. Forward+backward time of one token-mixing layer against (a) sequence length and (b) head dimension (hollow: TTT-MLP's compiled scan, as its fused kernel supports head dimension 64 only); (c) training throughput of the full 340M models on eight MI325X GPUs.

zero-shot accuracy with lm-evaluation-harness (Gao et al., 2023), recall on SWDE, FDA, TriviaQA, and DROP under the generation protocol of Arora et al. (2024a;b), and RULER needle-in-a-haystack (Hsieh et al., 2024) from 4K to 32K tokens, using YaRN (Peng et al., 2024b) for attention. Section E gives the full evaluation protocol.

Main results. Table 1 gives the language-modeling results. ParaTTT has the best perplexity of the recurrent models at both scales, the best LAMBADA perplexity overall, and outperforms each of Gated DeltaNet and Mamba-2 on three of the four recall tasks at both scales. We note that ParaTTT performs well on recall tasks using exact recurrent gradient-descent decoding despite being trained with the iterated approximation, suggesting that **the prefill–decode discrepancy is benign**. Attention, unsurprisingly, performs best on many of the recall and long-context retrieval metrics. TTT-MLP, which evaluates its nonlinear memory in blocks at a much higher training cost, matches Gated DeltaNet on WikiText but is far behind on LAMBADA and provides little improvement on recall. We ablate our two refinements from Section 3.4: removing centering degrades perplexity and recall and greatly reduces the gains from iteration (Fig. 2(c,d)), while replacing the hidden-layer delta rule with linear attention leaves perplexity similar but substantially degrades recall. Training with more iterations improves recall further at a proportionally higher cost per step. Removing the sliding-window attention branch from LaCT severely degrades its performance; Section F.2 discusses the reliance of prior TTT layers on this branch.

Perplexity as a function of token position. Figure 2(a,b) follows perplexity along held-out SlimPajama documents. ParaTTT has the lowest perplexity at every position at both scales, and its lead over Gated DeltaNet persists to the end of the 4096-token window, indicating that the gain is not limited to the beginning of the sequence. Attention has the worst cold start and only catches up with the recurrent models after a few hundred tokens.

Performance as a function of the evaluation-time iteration count. Figure 2(c,d) evaluates the 340M ParaTTT models at iteration counts other than their training N. Without iteration, the centered models perform poorly (LAMBADA perplexity above 300), whereas the non-centered model performs well but gains little from additional iterations, suggesting that **centering is necessary to unlock ParaTTT gains**. Models perform best near their training N: over-iterating the model trained at $N = 2$ degrades performance slowly, while the model trained at $N = 5$ exhibits little degradation. This suggests that the model adapts to the finite-N prefill used during training, even though exact recurrent decoding remains strong on the recall tasks.

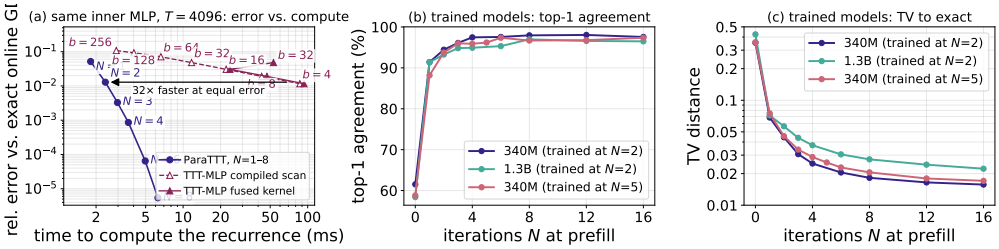


Figure 4: Convergence to the sequential recurrence. (a) One random inner MLP at $T = 4096$: relative output error against exact online GD versus the time to compute one 340M layer's recurrence, for ParaTTT at $N = 1$–8 and TTT-MLP's block form at $b = 4$–256 (filled: fused kernel; hollow: compiled scan); the arrow marks equal error to $N = 2$. (b,c) Trained models on real text: top-1 agreement and total-variation distance between the prefill at N iterations and the exact recurrence.

4.2 Efficiency

On MI325X GPUs in bf16, we time one token-mixing layer (forward and backward, $d = 1024$, 8 heads of dimension 128, batch 8) as a function of sequence length and head dimension, and measure the training throughput of the full 340M models on one node of eight GPUs (Section G). We compare ParaTTT at $N \in \{1, 2, 3, 5\}$ with Gated DeltaNet (Yang et al., 2025), Mamba-2 (Dao & Gu, 2024), softmax attention (FlashAttention), TTT-MLP with its default block size of 16 (Sun et al., 2025), and the published LaCT configuration with a 2048-token sliding-window attention branch (Zhang et al., 2026). Figure 3 shows the results.

Results. ParaTTT scales linearly in T like the linear recurrences it is built from: one layer at $N = 2$ costs $4.7\times$ a GDN layer, and each additional iteration adds a fixed number of chunkwise scans. TTT-MLP's block-sequential form is $5.5\times$ slower than ParaTTT at $T = 4096$ and $8.5\times$ at $T = 16{,}384$. At the model level, ParaTTT at $N = 2$ trains at 37.9k tokens/s per node against 7.8k for TTT-MLP, a $4.9\times$ speedup, while retaining roughly one-third the throughput of the linear baselines.

4.3 Convergence analysis

ParaTTT better approximates the true gradient trajectory than TTT-MLP. We compare ParaTTT's N-iteration approximation with TTT-MLP's sequential block approximation at block size b on one fixed random inner MLP ($d = 64$, eight heads, constant step size 0.2 and decay 0.98). Figure 4(a) plots each configuration's output error relative exact gradient descent against the time required to compute the recurrence, with TTT-MLP timed using either its fused kernel or a compiled version of its scan, whichever is faster. TTT-MLP's error is determined by the block size and can only be reduced by using smaller blocks and a longer sequential loop, whereas ParaTTT's error decreases with N. At $T = 4096$, $N = 2$ matches the error of TTT-MLP at $b \approx 5$ in $32\times$ less time, and $N = 8$ approaches the bf16 noise floor. Section I repeats the comparison at other sequence lengths.

Trained models converge quickly on real text. For the trained models, we use the exact token-by-token recurrence as the reference. Figure 4(b,c) compares the chunkwise prefill at N iterations against this reference using top-1 agreement and total-variation distance between the next-token distributions. No looping gets about 60% agreement, one iteration raises this above 90%, two to about 94%, and further iterations saturate near 98% at the bf16 noise floor. Thus, even for the practical models outside the assumptions of Theorem 1, only a few iterations are needed to closely approximate the exact recurrence.

5 Conclusion

In this work, we introduce ParaTTT (PTTT), an algorithm for parallelizing test-time training layers with arbitrarily deep memories through iterative refinement of the inner-loop gradient trajectory. We show that the exact online gradient descent recurrence is a fixed point of the iteration and prove geometric convergence for a two-layer nonlinear memory under sufficient contraction conditions. In practice, only a few refinement steps are needed: at 4K context, full 340M models train $4.9\times$ faster than TTT-MLP, while models at 340M and 1.3B achieve strong language-modeling performance and improved recall over linear recurrent baselines. These results narrow the long-standing tradeoff between the expressivity of nonlinear recurrence and the efficiency of parallel sequence processing.

AI USE STATEMENT

In this work we used generative AI tools for implementing methods, including the Triton kernels and the training and evaluation code; for designing and giving feedback on experiments; for assisting with the formulation and proofs of the theoretical results; and for interpreting results. We did not use generative AI tools to generate synthetic datasets, to clean or reformat datasets, or for translation; qualitative data analysis is not applicable to this work. Among tasks with recommended disclosure, we used these tools to draft and edit parts of the paper, to create figures, to search and summarize related literature, and to format references. We have reviewed all AI-assisted work: AI-generated code was tested against reference implementations (gradient checks and parity tests against eager PyTorch), all proofs were checked line by line by the authors, and all reported numbers come from experiments the authors ran and verified. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section D gives every model configuration and the training recipe, Section E the evaluation protocol for every number in the paper, Section G the kernel structure and the measurement setup of the efficiency figure, and Section B the proofs. The training and evaluation code, the kernels and the model configurations will be released.

ETHICS STATEMENT

This work studies language-model architectures trained on public pretraining data and evaluated on public benchmarks; it does not involve human subjects or personal data, and we are not aware of ethical concerns specific to it beyond those common to language-model pretraining.

REFERENCES

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. In *International Conference on Machine Learning*, 2024a. URL https://proceedings.mlr.press/v235/arora24a.html.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: Closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*, 2024b. URL https://arxiv.org/abs/2407.05483.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c2ce2f2701c10a2b2f2ea0bfa43cfaa3-Abstract-Conference.html.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. ATLAS: Learning to optimally memorize the context at test time. *arXiv preprint arXiv:2505.23735*, 2025a. URL https://arxiv.org/abs/2505.23735.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4ca07aa108036f80cbb5b82285fd4b1-Abstract-Conference.html.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*,

2019. URL https://proceedings.neurips.cc/paper/2019/hash/ae614c557843b1df326cb29c57225459-Abstract.html.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

Federico Danieli, Pau Rodríguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. ParaRNN: Unlocking parallel training of nonlinear RNNs for large language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mX8b64iUaa.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. URL https://doi.org/10.1207/s15516709cog1402_1.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Wenhao Huang, Di He, and Tianle Cai. In-place test-time training. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ba6f5879d7f382241316f7dda7e6b8ac-Abstract-Conference.html.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Xavier Gonzalez, Andrew Warrington, Jimmy T. H. Smith, and Scott W. Linderman. Towards scalable and stable parallelization of nonlinear RNNs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2407.19115.

Xavier Gonzalez, Leo Kozachkov, David M. Zoltowski, Kenneth L. Clarkson, and Scott W. Linderman. Predictability enables parallelization of nonlinear state space models. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1b7a239766f041abcd4d286ec5548037-Abstract-Conference.html.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear RNNs through negative eigenvalues. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5a0ce3abb720b740419e193c87afd080-Abstract-Conference.html.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Improving bilinear RNNs with closed-loop control. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9a439efaa34fe37177eba00737624824-Abstract-Conference.html.

Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hke3gyHYwH.

Yulong Huang, Xiang Liu, Hongxiang Huang, Xiaopeng Lin, Zunchang Liu, Xiaowen Chu, Zeke Xie, and Bojun Cheng. MDN: Parallelizing stepwise momentum for delta linear attention. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/63901.

Mahdi Karami, Razvan Pascanu, and Vahab Mirrokni. Lattice: Learning to efficiently compress the memory. *arXiv preprint arXiv:2504.05646*, 2025. URL https://arxiv.org/abs/2504.05646.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Kimi Team. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025. URL https://arxiv.org/abs/2510.26692.

Asher Labovich. Stability and generalization in looped transformers. *arXiv preprint arXiv:2604.15259*, 2026. URL https://arxiv.org/abs/2604.15259.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8abd2043b71a074278d5f687947bff9c-Abstract-Conference.html.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing nonlinear sequential models over the sequence length. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=E34AlVLN0v.

Junchen Liu, Sven Elflein, Or Litany, Zan Gojcic, and Ruilong Li. Test-time training with KV binding is secretly linear attention. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64516.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. URL https://aclanthology.org/2023.tacl-1.31/.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/88dd7aa6979e352fda7c4952ca8eac59-Abstract-Conference.html.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/merrill24a.html.

Mayank Mishra, Shawn Tan, Ion Stoica, Joseph E. Gonzalez, and Tri Dao. M^{2}RNN: Non-linear RNNs with matrix-valued states for scalable language modeling. *arXiv preprint arXiv:2603.14360*, 2026. URL https://arxiv.org/abs/2603.14360.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026. URL https://arxiv.org/abs/2606.18206.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Haowen Hou, Kranthi Kiran GV, Satyapriya Krishna, Ronald McClelland, Jr., Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Jian Zhu, and Rui-Jie Zhu. Eagle and Finch: RWKV with matrix-valued states and dynamic recurrence. In *First Conference on Language Modeling (COLM)*, 2024a. URL https://openreview.net/pdf/9f3d829735b231725665ff226a4f732293ad4aa8.pdf.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with expressive dynamic state evolution. In *Second Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=ayB1PACN5j.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=wHBfxhZu1u.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. URL https://doi.org/10.1038/323533a0.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2676109d49d1eb26d6bc584a8f556305-Abstract-Conference.html.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Mark Schöne, Babak Rahmani, Heiner Kremer, Fabian Falck, Hitesh Ballani, and Jannes Gladrow. Implicit language models are RNNs: Balancing parallelization and expressivity. In *International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/schone25a.html.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. DeltaProduct: Improving state-tracking in linear RNNs via Householder products. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/e1ea2520fbf9cd1600b287dde67e0a3c-Abstract-Conference.html.

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. In *International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025. URL https://arxiv.org/abs/2512.23675.

Neehal Tumma, Noel Loo, and Daniela Rus. Preconditioned DeltaNet: Curvature-aware sequence modeling for linear recurrences. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/63733.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Rif A. Saurous, Guillaume Lajoie, Charlotte Frenkel, Razvan Pascanu, Blaise Agüera y Arcas, and João Sacramento. MesaNet: Sequence modeling by locally optimal test-time training. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/2c3d5a4eff20d24390a445762d61bfbb-Abstract-Conference.html.

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

Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. URL https://doi.org/10.1109/5.58337.

Bernard Widrow and Marcian E. Hoff, Jr. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pp. 96–104, 1960. URL https://isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf.

Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory*, 2020. URL https://proceedings.mlr.press/v125/woodworth20a.html.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, 2024a. URL https://proceedings.mlr.press/v235/yang24ab.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2406.06484.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ffbfff81fb78e4bb558273b91e12e318-Abstract-Conference.html.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.

A Extended related work

Linear recurrences. The lineage in Section 2.1 runs from linear attention (Katharopoulos et al., 2020) through gated linear attention (Yang et al., 2024a), Mamba-2 (Dao & Gu, 2024), DeltaNet (Schlag et al., 2021; Yang et al., 2024b) and Gated DeltaNet (Yang et al., 2025); Wang et al. (2025) give the unifying view of all of these as test-time regression. The RWKV family follows the same path from a vector-valued to a matrix-valued state (Peng et al., 2024a) and, in RWKV-7, to a generalized delta rule with vector-valued gates and in-context learning rates (Peng et al., 2025). The most recent linear recurrences refine the delta rule's write and forget. KDA (Kimi Team, 2025) replaces Gated DeltaNet's scalar decay by a channel-wise one; Gated DeltaNet-2 (Hatamizadeh et al., 2026) further decouples the erase gate from the write gate, recovering both KDA and Gated DeltaNet as special cases; Mamba-3 (Lahoti et al., 2026) revisits the discretization of the state-space recurrence and adds complex-valued and multi-input multi-output updates; DeltaProduct (Siems et al., 2025) takes several delta steps per token so that the transition is a product of generalized Householder matrices; Comba (Hu et al., 2025) adds output feedback to the bilinear state transition; Lattice (Karami et al., 2025) writes only the component of a token orthogonal to the stored memory slot, with an approximate chunkwise evaluation; Momentum DeltaNet (Huang et al., 2026) gives the online gradient step a momentum term; Preconditioned DeltaNet (Tumma et al., 2026) preconditions it with an online estimate of the curvature; and MesaNet (von Oswald et al., 2026) solves the in-context regression to optimality with a conjugate-gradient solver. These methods retain matrix-valued memories but use different update and evaluation schemes: momentum introduces an auxiliary state, MesaNet adds a per-query solve, and Lattice approximates a state-dependent update. ParaTTT uses chunkwise linear recurrent kernels as building blocks. Extending its readout to these other updates is a possible direction, but would require checking the corresponding solver and gradient computation.

Test-time training. TTT-Linear and TTT-MLP (Sun et al., 2025) introduced the layer whose state is the weights of a small model trained online, evaluated in the mini-batch form of Section 2.2. It is worth writing this form out, since it is the approximation that ParaTTT removes. Let a block consist of tokens $s+1, \dots, s+b$ and let $\mathbf{W}_s$ be the weights entering it (TTT-MLP uses no decay, so we drop α). Exact online gradient descent evaluates the gradient of token j at the weights left by token $j-1$, $\mathbf{W}_j = \mathbf{W}_{j-1} - \eta_j \nabla \ell_j(\mathbf{W}_{j-1})$; the block form evaluates every gradient of the block at the weights that entered it and scores token t with the causal partial sum of these *stale* gradients,

$$\mathbf{W}_t^{\text{blk}} \;=\; \mathbf{W}_s - \sum_{j=s+1}^{t} \eta_j \, \nabla \ell_j(\mathbf{W}_s), \qquad s < t \le s+b. \tag{9}$$

For a squared-loss MLP with a linear readout, the readout update has an attention-style form inside the block. In the next display, $\mathbf{W}$ denotes the readout weights only. With $\boldsymbol{\phi}_j$ the features of token j, computed with the block-start hidden weights, the gradient at $\mathbf{W}_s$ is the outer product $(\mathbf{W}_s\boldsymbol{\phi}_j - \mathbf{v}_j)\boldsymbol{\phi}_j^\top$ of a residual with a feature. Let $\boldsymbol{\phi}_{\mathbf{q},t}$ denote the query features under the causally updated hidden weights at token t; then the output is

$$\mathbf{o}_t \;=\; \mathbf{W}_t^{\text{blk}}\boldsymbol{\phi}_{\mathbf{q},t} \;=\; \mathbf{W}_s\boldsymbol{\phi}_{\mathbf{q},t} - \sum_{j=s+1}^{t} \eta_j \left(\mathbf{W}_s\boldsymbol{\phi}_j - \mathbf{v}_j\right)\left(\boldsymbol{\phi}_j^\top\boldsymbol{\phi}_{\mathbf{q},t}\right), \tag{10}$$

an attention-style readout with keys $\boldsymbol{\phi}_j$, values the residuals $\mathbf{W}_s\boldsymbol{\phi}_j - \mathbf{v}_j$ and queries $\boldsymbol{\phi}_{\mathbf{q},t}$, so the readout sum can be evaluated with matrix multiplications once the features are given. The price is in the residuals: exact online descent would use the preceding readout weights and the current online features, so that the write of token j erases what earlier tokens of the block stored under similar keys (the delta rule), whereas the block form's residuals see none of the block's earlier writes and the writes simply add up, as in linear attention without a delta rule. The hidden-layer gradients are also evaluated at block-start weights. This freezes the features used to compute those gradients, but does not freeze the hidden weights or query features used for each token's output. At $b = 1$ the block form recovers online descent; at $b = T$ all gradient evaluations share the initial weights. TTT-MLP's default $b = 16$ and LaCT's larger blocks trade sequential depth against update frequency, and Fig. 4(a) measures the resulting error against the exact trajectory. ParaTTT instead targets the $b = 1$ recurrence by iterating on the gradients.

Liu et al. (2026) study LaCT-style and vision TTT layers. They report that adding inner steps only at inference can hurt downstream performance, that retraining with gradient ascent can preserve

performance, and that replacing queries with keys can have little effect in their experiments. Their attention-style representation allows updated hidden features; it is not limited to a frozen-feature model. They also obtain parallel variants through further simplifications, including restricting adaptation to the final layer. These findings motivate testing which parts of nonlinear adaptation matter. Our experiments address this question for the online recurrence, but do not establish that the block approximation explains their observations.

Titans (Behrouz et al., 2025b) adds momentum and weight decay to the inner optimizer and ATLAS (Behrouz et al., 2025a) optimizes the memory over a window of past tokens rather than the current one; LaCT (Zhang et al., 2026) makes the block very large so that the inner update is a dense matrix multiplication, and pairs it with a sliding-window attention branch (Section F.2). Two recent lines move the fast weights into the backbone. In-place TTT (Feng et al., 2026) treats the down-projection of every MLP block as the fast weights, trains it with an objective aligned with next-token prediction, and updates it chunk-wise; end-to-end TTT (Tandon et al., 2025) keeps a standard transformer with sliding-window attention and adapts selected MLP weights on the context by next-token prediction, meta-learned so that the test-time updates help. In its reported configuration, adaptation is restricted to the last quarter of the blocks. Both inherit the block structure: the weights are frozen inside a chunk or a window and updated between them. ParaTTT is orthogonal to these choices of what to adapt and with which loss; it changes how the inner trajectory of a nonlinear memory is computed, so that every token's update is taken at the weights left by the previous token, and it does so with the parallel kernels of the linear recurrences above.

Sequential nonlinear RNNs. Nonlinear recurrent networks (Elman, 1990; Hochreiter & Schmidhuber, 1997; Cho et al., 2014) are trained by backpropagation through time (BPTT) (Rumelhart et al., 1986; Werbos, 1990), which unrolls the recurrence over the sequence and runs the backward pass through every step in reverse. Their straightforward forward evaluation and BPTT are sequential over tokens, limiting sequence parallelism compared with attention (Vaswani et al., 2017). The sLSTM cell of xLSTM (Beck et al., 2024) keeps a sequential nonlinear transition next to a parallel linear one. M^{2}RNN (Mishra et al., 2026) is the most direct recent attempt to make the sequential path competitive: it keeps a matrix-valued state, updated by a $\tanh$ of a linear map of the previous state plus an outer-product write with a learned forget gate, so that the state size scales as it does in linear attention, and it evaluates the recurrence sequentially in a Triton kernel that parallelizes over batch and heads while looping over time; the backward pass is full BPTT over the 4096-token training sequence, recomputing the forward states and running the reverse loop over every step. The nonlinear transition prevents direct evaluation by the linear scans used by SSMs. The paper reports perfect length generalization on its evaluated state-tracking tasks, with training throughput below its linear recurrent baselines. ParaTTT targets the same expressivity, a nonlinear transition on a large state, but it obtains a parallel evaluation of it, and its nonlinearity acts on the features that the memory reads and writes with rather than on the state transition directly.

Parallelizing nonlinear recurrences and its limits. DEER (Lim et al., 2024) cast the evaluation of a nonlinear recurrence as a fixed-point problem over the whole trajectory and solved it with Newton's method, each step a linear recurrence evaluated by a parallel scan; ELK and quasi-DEER (Gonzalez et al., 2024) stabilize the Newton iteration through its connection to Kalman smoothing and make it scale by using diagonal Jacobians; ParaRNN (Danieli et al., 2026) designs LSTM and GRU cells with structured Jacobians so that the Newton step is cheap and trains 7B-parameter nonlinear RNNs this way; Gonzalez et al. (2025) show that the number of iterations these methods need is governed by the predictability of the dynamics, measured by the largest Lyapunov exponent, and is polylogarithmic in the sequence length for contractive systems. Our Theorem 1 is a statement of the same kind for our iteration: the contraction condition bounds the learning rate times the memory horizon, which is the quantity that controls how far a token's update can propagate. A small iteration count improves parallelism, but its expressivity requires a separate analysis. An associative scan generally has logarithmic circuit depth; a fixed number of scans alone does not establish a TC^0 upper bound. The bounds of Merrill & Sabharwal (2023); Merrill et al. (2024) apply to the architectures and precision assumptions they analyze. For transformers, Merrill & Sabharwal (2025) show that logarithmic depth suffices to recognize regular languages. We do not claim a circuit-complexity classification for ParaTTT. Our finite-step result instead shows that $N \geq T$ recovers its sequential recurrence in the undamped case (Proposition 1). The practical question, which Section 4.3 addresses, is how many iterations are needed for the tasks that matter.

Looped and implicit models. A second family iterates a block of layers in depth rather than a recurrence in time. Universal transformers (Dehghani et al., 2019) apply the same layer repeatedly with adaptive halting; deep equilibrium models (Bai et al., 2019) take the limit and define the output as the fixed point of a layer, found by a root solver and differentiated implicitly; looped transformers were shown to be programmable computers (Giannou et al., 2023) and to match the reasoning ability of much deeper unlooped models (Saunshi et al., 2025), and recent language models scale test-time compute by unrolling a recurrent block to a chosen depth (Geiping et al., 2025; Zhu et al., 2025). Fixed-point reasoners (Movahedi et al., 2026) use convergence to a fixed point as the halting criterion of a looped transformer and add damping to remove oscillations of the iterates, and Labovich (2026) analyze when the fixed points of a looped transformer are reachable, input-dependent and stable under backpropagation, and find that recall connections with outer normalization improve stability and generalization in their studied architectures. Closest to our work, Schöne et al. (2025) iterate a state-space model over the whole sequence until convergence to a fixed point and show that the implicit model implements the nonlinear state transition of an RNN; like ParaTTT, they observe that approximate convergence suffices during training, and they scale the approach to 1.3B parameters. The essential difference is at decoding time. In recurrent-depth models, the chosen number of loops is part of the token computation; in fixed-point models, the output is defined by an equilibrium. Both require repeated computation per generated token, including in the sequential mode of Schöne et al. (2025). For ParaTTT the loop is only a method for computing a prefill; the object being computed is the exact sequential recurrence of Eq. (2), which for one new token is a single gradient step from the cached weights. Decoding therefore requires no iteration at all (Section 3.2), and N affects the cost and the accuracy of the prefill only.

B PROOFS

Setting, restated. We analyze the basic iteration with $L = 2$, the linear-attention hidden update Eq. (4), and the uncentered readout. For this analysis only, replace the unit-ball projection used in the experiments with the soft clip $n_{\mathrm{soft}}(\mathbf{x}) = \mathbf{x}/\sqrt{1+\|\mathbf{x}\|^2}$. This keeps the features bounded and, under the activation assumptions below, makes their Jacobian globally Lipschitz. Write $\psi = n_{\mathrm{soft}} \circ \sigma$, where σ acts coordinatewise. The features of this smooth variant are

$$\boldsymbol{\phi}_t = \psi(\mathbf{z}_t), \qquad \mathbf{z}_t = \mathbf{W}_0^{(1)}\mathbf{k}_t + \mathbf{b}^{(1)} + \alpha_t \Delta\mathbf{W}_{t-1}^{(1)}\mathbf{k}_t.$$

Step 3 uses the exact Jacobian $\mathbf{H}_t = D\psi(\mathbf{z}_t)$ at the current pre-activation. Write $\|\mathbf{D}\|_\infty = \max_t \|\delta\mathbf{z}_t^{(1)}\|$ and F for one iteration's map $\mathbf{D} \mapsto \hat{\mathbf{D}}$.

Theorem (Theorem 1, restated)**.** *For the soft-clipped variant defined above, assume* $\|\mathbf{k}_t\| \le 1$, $\|\mathbf{v}_t\| \le V$, *and* $\|\mathbf{W}_0^{(2)}\|_{\mathrm{op}} \le w_0$*; shared decay* $0 \le \alpha_t \le \bar{\alpha} < 1$ *and shared learning rates* $0 < \eta_t^{(1)} = \eta_t^{(2)} = \eta_t \le \bar{\eta} \le 1$*; and* $\sigma \in C^2(\mathbb{R})$ *with* $|\sigma'(x)| \le c_\sigma$ *and* $|\sigma''(x)| \le h_\sigma$ *for every* $x \in \mathbb{R}$. *Set* $\epsilon = \bar{\eta}/(1-\bar{\alpha})$ *and* $Q = V + w_0$. *Then* F *is globally Lipschitz in* $\|\cdot\|_\infty$ *with constant*

$$\rho = \epsilon(1+\epsilon)Q^2\big[c_\sigma^2(1+\epsilon)(5+4\epsilon) + h_\sigma(2+\epsilon)\big].$$

For $\epsilon \le \frac{1}{4}$, $\rho \le (10c_\sigma^2 + 3h_\sigma)\epsilon Q^2$. *If* $\rho < 1$, F *has a unique fixed point* $\mathbf{D}^\star$*, the gradient sequence of the sequential recurrence Eq. (2) with* n_{soft} *in place of the projection. For any initialization and damping* $\lambda \in (0, 1]$, *Eq. (7) converges geometrically:*

$$\|\mathbf{D}^{[n]} - \mathbf{D}^\star\|_\infty \le \big(1 - \lambda(1-\rho)\big)^n \|\mathbf{D}^{[0]} - \mathbf{D}^\star\|_\infty.$$

The features, readout states and outputs converge to those of the sequential recurrence at the same rate, up to constant factors.

Proof strategy. We compare two guesses of the hidden-gradient sequence and follow their differences through one iteration. First, the hidden scan turns the guesses into pre-activations and features. Next, these features determine the readout trajectory and its feature gradients. Finally, the backward pass multiplies each feature gradient by the activation Jacobian. Both factors in this last product change between the two guesses, so we must control both changes. The lemmas below give these bounds in the order they are needed.

Notation. We use the Euclidean norm for vectors and the operator norm for matrices. For a sequence, $\|\cdot\|_\infty$ is the maximum of the vector norms over tokens. There is one hidden layer, so we omit its layer index and write $\mathbf{D} = (\delta \mathbf{z}_t)_{t \le T}$. A prime denotes the quantity obtained from a second guess $\mathbf{D}'$, with the keys, values and model parameters held fixed.

Past writes are multiplied by the decay factors

$$\gamma_{j,t} = \prod_{i=j+1}^{t-1} \alpha_i.$$

The assumption $\alpha_i \le \bar{\alpha} < 1$ bounds their total contribution, independently of sequence length:

$$\bar{\eta} \sum_{j<t} \gamma_{j,t} \le \bar{\eta} \sum_{r \ge 0} \bar{\alpha}^r = \frac{\bar{\eta}}{1-\bar{\alpha}} = \epsilon.$$

We will use this geometric-series bound for both the hidden and readout states. For brevity, write $n = n_{\text{soft}}$ throughout the proof.

Lemma 1 (Soft clip)**.** *For $n(\mathbf{x}) = \mathbf{x}/\sqrt{1+\|\mathbf{x}\|^2}$, the output and Jacobian satisfy*

$$\|n(\mathbf{x})\| \le 1, \qquad \|Dn(\mathbf{x})\| \le 1, \qquad \|Dn(\mathbf{x}) - Dn(\mathbf{y})\| \le 2\|\mathbf{x}-\mathbf{y}\|.$$

Proof. The output bound follows directly from the definition. To bound the derivatives, set $s = \sqrt{1+\|\mathbf{x}\|^2}$ and $\mathbf{a} = n(\mathbf{x})$. Differentiation gives

$$Dn(\mathbf{x}) = s^{-1}\mathbf{I} - s^{-3}\mathbf{x}\mathbf{x}^\top = s^{-1}(\mathbf{I} - \mathbf{a}\mathbf{a}^\top).$$

Along $\mathbf{x}$, this matrix has eigenvalue s^{-3}; in directions perpendicular to $\mathbf{x}$, it has eigenvalue s^{-1}. At $\mathbf{x} = 0$ it is the identity. Thus $\|Dn(\mathbf{x})\| \le s^{-1} \le 1$, which also shows that n is nonexpansive.

It remains to bound the change in the Jacobian. In a direction $\mathbf{h}$, the derivative of the scalar factor s^{-1} has magnitude at most $\|\mathbf{x}\|\|\mathbf{h}\|/s^3$. The derivative of $\mathbf{a}\mathbf{a}^\top$ has norm at most $2\|\mathbf{a}\|\|Dn(\mathbf{x})\mathbf{h}\|$. Applying the product rule therefore gives

$$\|D(Dn)(\mathbf{x})[\mathbf{h}]\| \le \frac{3\|\mathbf{x}\|}{(1+\|\mathbf{x}\|^2)^{3/2}}\|\mathbf{h}\| \le \frac{2}{\sqrt{3}}\|\mathbf{h}\| \le 2\|\mathbf{h}\|.$$

Here we used $\max_{r \ge 0} r/(1+r^2)^{3/2} = 2/(3\sqrt{3})$. Integrating this bound along the segment from $\mathbf{x}$ to $\mathbf{y}$ proves the final inequality. □

Lemma 2 (Composite activation)**.** *The full activation $\psi = n \circ \sigma$ satisfies*

$$\|D\psi(\mathbf{z})\| \le c_\sigma, \qquad \|D\psi(\mathbf{z}) - D\psi(\mathbf{z}')\| \le (2c_\sigma^2 + h_\sigma)\|\mathbf{z}-\mathbf{z}'\|.$$

Proof. The chain rule factors the Jacobian into the derivatives of the soft clip and the coordinatewise activation:

$$D\psi(\mathbf{z}) = \mathbf{N}(\mathbf{z})\mathbf{J}(\mathbf{z}), \qquad \mathbf{N}(\mathbf{z}) = Dn(\sigma(\mathbf{z})), \qquad \mathbf{J}(\mathbf{z}) = \operatorname{diag}\sigma'(\mathbf{z}).$$

Since $\|\mathbf{N}\| \le 1$ and $\|\mathbf{J}\| \le c_\sigma$, their product has norm at most c_σ. For its variation, the activation assumptions and Lemma 1 give

$$\begin{aligned}\|\mathbf{N}(\mathbf{z}) - \mathbf{N}(\mathbf{z}')\| &\le 2\|\sigma(\mathbf{z}) - \sigma(\mathbf{z}')\| \le 2c_\sigma\|\mathbf{z}-\mathbf{z}'\|,\\ \|\mathbf{J}(\mathbf{z}) - \mathbf{J}(\mathbf{z}')\| &\le h_\sigma\|\mathbf{z}-\mathbf{z}'\|.\end{aligned}$$

Now split the difference of the products as $\mathbf{N}\mathbf{J} - \mathbf{N}'\mathbf{J}' = (\mathbf{N}-\mathbf{N}')\mathbf{J} + \mathbf{N}'(\mathbf{J}-\mathbf{J}')$. The first term contributes at most $2c_\sigma^2\|\mathbf{z}-\mathbf{z}'\|$, and the second contributes at most $h_\sigma\|\mathbf{z}-\mathbf{z}'\|$. Adding them proves the claim. □

Lemma 3 (Features)**.** *For any two gradient guesses, let $\delta = \|\mathbf{D} - \mathbf{D}'\|_\infty$. Then*

$$\|\boldsymbol{\phi}_t\| \le 1, \qquad \|\mathbf{z}_t - \mathbf{z}'_t\| \le \epsilon\delta, \qquad \|\boldsymbol{\phi}_t - \boldsymbol{\phi}'_t\| \le c_\sigma\epsilon\delta.$$

Proof. Only the fast weights depend on the gradient guess. Their contribution to the pre-activation is, by Eq. (4),

$$\mathbf{u}_t = \alpha_t \Delta \mathbf{W}_{t-1}^{(1)} \mathbf{k}_t = -\alpha_t \sum_{j<t} \gamma_{j,t} \eta_j (\mathbf{k}_j^\top \mathbf{k}_t) \delta \mathbf{z}_j.$$

The normalized keys satisfy $|\mathbf{k}_j^\top \mathbf{k}_t| \le 1$. Subtracting the two sums and using $\alpha_t \le 1$ gives

$$\|\mathbf{z}_t - \mathbf{z}_t'\| = \|\mathbf{u}_t - \mathbf{u}_t'\| \le \bar{\eta} \sum_{j<t} \gamma_{j,t} \|\delta \mathbf{z}_j - \delta \mathbf{z}_j'\| \le \epsilon \delta.$$

Applying the c_σ-Lipschitz map ψ gives the feature-difference bound. The bound $\|\boldsymbol{\phi}_t\| \le 1$ follows from the soft clip. □

Lemma 4 (Readout trajectory)**.** *Let $\mathbf{S}_t = \Delta \mathbf{W}_t^{(2)}$ be the readout's fast weights for features satisfying $\|\boldsymbol{\phi}_t\| \le 1$. Then*

$$\|\mathbf{S}_t\| \le \epsilon Q, \qquad \|\bar{\mathbf{W}}_{t-1}^{(2)}\| \le (1+\epsilon) Q.$$

For two such feature sequences, writing $\delta_\phi = \|\boldsymbol{\phi} - \boldsymbol{\phi}'\|_\infty$, we also have

$$\|\mathbf{S}_t - \mathbf{S}_t'\| \le 2\epsilon(1+\epsilon) Q \, \delta_\phi.$$

Proof. Separate each readout update into the part carried from the previous state and the new write:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} (\mathbf{I} - \eta_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top) + \eta_t (\mathbf{v}_t - \mathbf{W}_0^{(2)} \boldsymbol{\phi}_t) \boldsymbol{\phi}_t^\top, \qquad \mathbf{S}_0 = \mathbf{0}. \tag{11}$$

We first bound the state itself, then compare the states generated by different features.

State size. Because $\eta_t \|\boldsymbol{\phi}_t\|^2 \le 1$, the matrix $\mathbf{I} - \eta_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top$ has operator norm at most one. The new write has norm at most $\bar{\eta}(V + w_0) = \bar{\eta} Q$. Hence

$$\|\mathbf{S}_t\| \le \bar{\alpha} \|\mathbf{S}_{t-1}\| + \bar{\eta} Q.$$

Starting from zero and summing the geometric series gives $\|\mathbf{S}_t\| \le \epsilon Q$. Adding the slow weights then gives

$$\|\bar{\mathbf{W}}_{t-1}^{(2)}\| = \|\mathbf{W}_0^{(2)} + \alpha_t \mathbf{S}_{t-1}\| \le w_0 + \epsilon Q \le (1+\epsilon) Q.$$

Sensitivity to the features. Write $\delta_t = \|\mathbf{S}_t - \mathbf{S}_t'\|$. Two elementary bounds control the feature-dependent terms in Eq. (11):

$$\|\boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top - \boldsymbol{\phi}_t' \boldsymbol{\phi}_t'^\top\| \le 2\delta_\phi,$$
$$\|(\mathbf{v}_t - \mathbf{W}_0^{(2)} \boldsymbol{\phi}_t) \boldsymbol{\phi}_t^\top - (\mathbf{v}_t - \mathbf{W}_0^{(2)} \boldsymbol{\phi}_t') \boldsymbol{\phi}_t'^\top\| \le (V + 2w_0) \delta_\phi.$$

The first follows by changing one factor of the outer product at a time. For the second, the value term contributes $V\delta_\phi$ and the quadratic feature term contributes $2w_0\delta_\phi$. Subtracting the two recurrences now gives

$$\begin{aligned} \delta_t &\le \bar{\alpha} \delta_{t-1} + 2\bar{\eta}\bar{\alpha} \|\mathbf{S}_{t-1}'\| \delta_\phi + \bar{\eta}(V + 2w_0) \delta_\phi \\ &\le \bar{\alpha} \delta_{t-1} + 2\bar{\eta}(1+\epsilon) Q \, \delta_\phi. \end{aligned}$$

In the last line we used the state-size bound and $V + 2w_0 \le 2Q$. Since $\delta_0 = 0$, another geometric sum yields $\delta_t \le 2\epsilon(1+\epsilon) Q \, \delta_\phi$. □

Lemma 5 (Feature gradient)**.** *For features in the unit ball, let*

$$\mathbf{g}_t = \bar{\mathbf{W}}_{t-1}^{(2)\top} (\bar{\mathbf{W}}_{t-1}^{(2)} \boldsymbol{\phi}_t - \mathbf{v}_t).$$

Its size and sensitivity to the feature sequence satisfy

$$\|\mathbf{g}_t\| \le B, \qquad \|\mathbf{g}_t - \mathbf{g}_t'\| \le A \|\boldsymbol{\phi} - \boldsymbol{\phi}'\|_\infty,$$

where

$$B = (1+\epsilon)(2+\epsilon) Q^2, \qquad A = (1+\epsilon)(1 + 7\epsilon + 4\epsilon^2) Q^2.$$

Proof. Fix a token and abbreviate its decayed readout by $\bar{\mathbf{W}}$ and its residual by $\mathbf{r} = \bar{\mathbf{W}}\boldsymbol{\phi}_t - \mathbf{v}_t$. Set $M = (1+\epsilon)Q$. By Lemma 4,

$$\|\bar{\mathbf{W}}\| \le M, \qquad \|\mathbf{r}\| \le M + V.$$

Therefore $\|\mathbf{g}_t\| \le M(M+V) \le B$, using $V \le Q$.

For the difference bound, there are two sources of change: the readout weights and the features at this token. Writing $\Delta_W = \|\bar{\mathbf{W}} - \bar{\mathbf{W}}'\|$ and $\delta_\phi = \|\boldsymbol{\phi} - \boldsymbol{\phi}'\|_\infty$, we have

$$\mathbf{r} - \mathbf{r}' = (\bar{\mathbf{W}} - \bar{\mathbf{W}}')\boldsymbol{\phi}_t + \bar{\mathbf{W}}'(\boldsymbol{\phi}_t - \boldsymbol{\phi}'_t), \qquad \|\mathbf{r} - \mathbf{r}'\| \le \Delta_W + M\delta_\phi.$$

Splitting the gradient difference in the same way gives

$$\begin{aligned}\|\mathbf{g}_t - \mathbf{g}'_t\| &\le \Delta_W\|\mathbf{r}\| + \|\bar{\mathbf{W}}'\|\,\|\mathbf{r} - \mathbf{r}'\| \\ &\le (2M+V)\Delta_W + M^2\delta_\phi.\end{aligned}$$

The readout lemma supplies $\Delta_W \le 2\epsilon(1+\epsilon)Q\delta_\phi$. Substituting this and $2M + V \le (3+2\epsilon)Q$ yields

$$\|\mathbf{g}_t - \mathbf{g}'_t\| \le \big[2\epsilon(1+\epsilon)(3+2\epsilon) + (1+\epsilon)^2\big]Q^2\delta_\phi = A\delta_\phi.$$

□

Proof of Theorem 1. Bounding one iteration. Let $\delta = \|\mathbf{D} - \mathbf{D}'\|_\infty$. The backward pass returns $F_t(\mathbf{D}) = \mathbf{H}_t^\top \mathbf{g}_t$, where $\mathbf{H}_t = D\psi(\mathbf{z}_t)$. The preceding lemmas give the four bounds needed to compare these products:

$$\begin{aligned}\|\mathbf{H}_t\| &\le c_\sigma, & \|\mathbf{H}_t - \mathbf{H}'_t\| &\le (2c_\sigma^2 + h_\sigma)\epsilon\delta, \\ \|\mathbf{g}'_t\| &\le B, & \|\mathbf{g}_t - \mathbf{g}'_t\| &\le Ac_\sigma\epsilon\delta.\end{aligned}$$

Here A bounds the sensitivity of the feature gradient and B bounds its size, as defined in Lemma 5. The change in the gradient contributes the first term below; the change in the Jacobian contributes the second:

$$\begin{aligned}\|F_t(\mathbf{D}) - F_t(\mathbf{D}')\| &\le \|\mathbf{H}_t\|\,\|\mathbf{g}_t - \mathbf{g}'_t\| + \|\mathbf{H}_t - \mathbf{H}'_t\|\,\|\mathbf{g}'_t\| \\ &\le \epsilon\big[c_\sigma^2 A + (2c_\sigma^2 + h_\sigma)B\big]\delta.\end{aligned}$$

Substituting A and B and collecting terms gives exactly the constant ρ in Eq. (8). Taking the maximum over tokens proves $\|F(\mathbf{D}) - F(\mathbf{D}')\|_\infty \le \rho\delta$. The simpler bound for $\epsilon \le 1/4$ follows from

$$(1+\epsilon)^2(5+4\epsilon) \le \frac{75}{8} < 10, \qquad (1+\epsilon)(2+\epsilon) \le \frac{45}{16} < 3.$$

Identifying the limit. If $\rho < 1$, the map is a contraction on the complete space $(\mathbb{R}^{T\times m}, \|\cdot\|_\infty)$. Banach's theorem therefore gives a unique fixed point $\mathbf{D}^\star$. At that point, every guessed gradient equals the actual gradient returned by the backward pass. This identifies the resulting trajectory with the sequential recurrence using the soft clip: both start from zero fast weights; if their states agree before a token, their features and gradients agree at that token, so their updates give the same next states. Induction over tokens proves agreement of the whole trajectory.

Damping and the convergence rate. The damped map is $G(\mathbf{D}) = (1-\lambda)\mathbf{D} + \lambda F(\mathbf{D})$. It has the same fixed point and satisfies

$$\|G(\mathbf{D}) - G(\mathbf{D}')\|_\infty \le (1 - \lambda + \lambda\rho)\|\mathbf{D} - \mathbf{D}'\|_\infty.$$

Applying this inequality repeatedly with $\mathbf{D}' = \mathbf{D}^\star$ gives the stated geometric rate.

Features, states and outputs. The feature and readout lemmas show that these quantities change by at most a constant times the error in $\mathbf{D}$. The same is true for query-side features: their unshifted hidden scan has gain at most $\epsilon \max_t \|\mathbf{q}_t\|$, followed by the c_σ-Lipschitz map ψ. Finally, the output is $\mathbf{o}_t = \mathbf{W}_t^{(2)}\boldsymbol{\phi}_t^{\mathbf{q}}$, a product of bounded quantities that are Lipschitz in $\mathbf{D}$. Thus their errors all inherit the same geometric rate, up to constant factors. □

SiLU satisfies the assumptions. For $\sigma(x) = xs(x)$ with $s(x) = (1+e^{-x})^{-1}$, $\sigma'(x) = s(x) + xs'(x)$ and $\sigma''(x) = 2s'(x) + xs'(x)(1-2s(x))$. Since $s'(x) \le \min(1/4, e^{-|x|})$ and $|x|e^{-|x|} \le 1/e$, valid global bounds are $c_\sigma = 1 + 1/e$ and $h_\sigma = 1/2 + 1/e$.

Fixed point and finite-step exactness. This is the formal version of Proposition 1.

Proposition 2 (Finite-step exactness of the undamped basic iteration)**.** *For the basic iteration of Section 3.2 with $\lambda = 1$ and any initialization, the estimated hidden gradients at tokens $1, \dots, \min(n, T)$ equal their sequential values after n iterations. Consequently, for $N \geq T$, the final evaluation reproduces the sequential recurrence exactly in exact arithmetic. The argument applies at any memory depth, with either projection or soft clipping, using the same derivative convention in the parallel and sequential evaluations, and requires no contraction condition. For $0 < \lambda < 1$, finite-step exactness fails in general: already at token 1, $\mathbf{D}_1^{[n]} - \mathbf{D}_1^\star = (1-\lambda)^n(\mathbf{D}_1^{[0]} - \mathbf{D}_1^\star)$.*

Proof. Group all hidden-layer gradient estimates at token t into $\mathbf{D}_t$. The basic iteration is strictly causal: $F_t(\mathbf{D})$ depends only on $\mathbf{D}_1, \dots, \mathbf{D}_{t-1}$. To see this at arbitrary depth, proceed layer by layer in the forward pass. At token $s \leq t$, each hidden scan uses only writes $j < s$, and its layer inputs depend only on estimates at tokens preceding their own positions. Thus all key-side features through token t depend only on $\mathbf{D}_{<t}$. The readout before token t is determined by earlier features and updates. The backward pass uses these same hidden and readout states, so it introduces no dependence on $\mathbf{D}_t$ or later estimates.

The sequential gradient sequence $\mathbf{D}^\star$ satisfies $F(\mathbf{D}^\star) = \mathbf{D}^\star$: at $\mathbf{D}^\star$ the shifted scans of Step 1 sum the writes of tokens $j < t$ and reproduce the decayed weights $\bar{\mathbf{W}}_{t-1}^{(l)}$ of Eq. (2) at every token, Step 2 reproduces $\bar{\mathbf{W}}_{t-1}^{(L)}$, and Step 3 is the backpropagation of the sequential model, so $\widehat{\delta\mathbf{z}} = \delta\mathbf{z}$; the parallel evaluation at $\mathbf{D}^\star$ therefore coincides with the sequential recurrence, and damping changes the iterates but not this fixed point. Since F_1 has no dependence on $\mathbf{D}$, the first undamped iteration gives $\mathbf{D}_1^{[1]} = \mathbf{D}_1^\star$. Suppose the first n tokens are exact after n iterations. For every $t \leq \min(n+1, T)$, all arguments of F_t then agree with those of $F_t(\mathbf{D}^\star)$, hence $\mathbf{D}_t^{[n+1]} = \mathbf{D}_t^\star$. Induction proves the prefix claim and gives $\mathbf{D}^{[N]} = \mathbf{D}^\star$ whenever $N \geq T$. The final forward and readout passes therefore reproduce the sequential states and outputs. The argument uses only causality and identical local derivative evaluations, so it applies to either clipping choice; at a nondifferentiable point of the hard projection, both evaluations must use the same fixed derivative convention. □

C Algorithm Details

C.1 Shifted and Unshifted Scans

Every scan in Algorithm 1 is one of two forms of the same recurrence, which differ only in whether token t reads the state before or after its own write. For a gated linear attention with keys $\mathbf{k}_j$, values $\mathbf{v}_j$ and gates α_j, the state is $\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$, and the two reads of a query $\mathbf{q}_t$ are

$$\begin{aligned} \text{unshifted:} \quad \mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t &= \sum_{j \leq t} \Big(\prod_{i=j+1}^{t} \alpha_i \Big) \mathbf{v}_j \, (\mathbf{k}_j^\top \mathbf{q}_t), \\ \text{shifted:} \quad \mathbf{o}_t = \alpha_t \mathbf{S}_{t-1} \mathbf{q}_t &= \sum_{j < t} \Big(\prod_{i=j+1}^{t} \alpha_i \Big) \mathbf{v}_j \, (\mathbf{k}_j^\top \mathbf{q}_t). \end{aligned} \tag{12}$$

The unshifted read is the standard one, in which the sum includes the current token $j = t$; the shifted read excludes it. We need both because the sequential recurrence Eq. (2) evaluates everything about the key of token t at the decayed weights $\bar{\mathbf{W}}_{t-1}$, before its own gradient step, and the query at $\mathbf{W}_t$, after it. The hidden-layer pre-activation of key t, Eq. (4), is therefore a shifted read with $\mathbf{q}_t = \mathbf{h}_t^{(l-1)}$ and $\mathbf{v}_j = -\eta_j^{(l)} \delta\mathbf{z}_j^{(l)}$; the transposed read of Step 3, Eq. (6), is a shifted read with the roles of keys and values exchanged; DeltaRuleGrad reads the readout state before token t's write, Eq. (21), which is again shifted; and the final output pass reads the query-side features under the updated weights, which is unshifted. **The shift is also what makes the fixed-point argument of Section 3.3 work**: with a shifted read no token's update depends on its own write, so all T updates can be evaluated simultaneously from an estimate of $\mathbf{D}$; with an unshifted read the update of token t would depend on $\delta\mathbf{z}_t$ itself and the parallel evaluation would not reproduce the sequential recurrence. The use of shifted kernels for keys is consistent with standard formulation of online gradient descent used in GDN and TTT-MLP, where the gradient (on the keys) is evaluated after applying the decay (shifted), the the readout from the query is applied using the update state (unshifted).

Algorithm 1 ParaTTT layer (one head; all operations act on the full sequence of T tokens)

Require: keys $\mathbf{k}_{1:T}$, queries $\mathbf{q}_{1:T}$, values $\mathbf{v}_{1:T}$; gates $\alpha_{1:T}$, learning rates $\eta_{1:T}^{(1..L)}$; weights $\mathbf{W}_0^{(l)}, \mathbf{b}^{(l)}$; iterations N; damping λ

$\delta\mathbf{z}_{1:T}^{(l)} \leftarrow \mathbf{0}$ for every hidden layer $l < L$ ▷ initial estimate of the backward gradients
for $n = 1, \dots, N$ **do**
  Step 1: parallel forward pass
  $\mathbf{h}^{(0)} \leftarrow \mathbf{k}$
  for $l = 1, \dots, L-1$ **do**
    $\mathbf{a}^{(l)} \leftarrow \mathbf{W}_0^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} + \textsc{ShiftedScan}\big(\text{keys } \mathbf{h}^{(l-1)},\ \text{values } -\eta^{(l)}\delta\mathbf{z}^{(l)},\ \text{gates } \alpha\big)$
    $\mathbf{h}^{(l)} \leftarrow \sigma(\mathbf{a}^{(l)})$; store the Jacobian $\mathbf{J}^{(l)}$
  end for
  Step 2: readout trajectory and its gradient
  $\mathbf{e} \leftarrow \textsc{DeltaRuleGrad}\big(\text{keys } \mathbf{h}^{(L-1)},\ \text{values } \mathbf{v},\ \text{gates } \alpha,\ \text{step sizes } \eta^{(L)}\big)$ ▷ Eq. (5)
  Step 3: parallel backward pass
  $\mathbf{g} \leftarrow \mathbf{e}/\eta^{(L)}$
  for $l = L-1, \dots, 1$ **do**
    $\widehat{\delta\mathbf{z}}^{(l)} \leftarrow \mathbf{J}^{(l)\top}\mathbf{g}$
    if $l > 1$ **then**
      $\mathbf{g} \leftarrow \mathbf{W}_0^{(l)\top}\widehat{\delta\mathbf{z}}^{(l)}$
      $\mathbf{g} \leftarrow \mathbf{g} + \textsc{TransposedScan}\big(\text{keys } \delta\mathbf{z}^{(l)},\ \text{values } -\eta^{(l)}\mathbf{h}^{(l-1)},\ \text{gates } \alpha\big)\,\widehat{\delta\mathbf{z}}^{(l)}$
    end if
  end for
  $\delta\mathbf{z}^{(l)} \leftarrow \delta\mathbf{z}^{(l)} + \lambda\,(\widehat{\delta\mathbf{z}}^{(l)} - \delta\mathbf{z}^{(l)})$ for every hidden layer l
end for
Output
$\boldsymbol{\phi}_{\mathbf{k}} \leftarrow$ features of $\mathbf{k}$ from a final Step 1 (shifted); $\boldsymbol{\phi}_{\mathbf{q}} \leftarrow$ features of $\mathbf{q}$ from the same scans, unshifted
return $\mathbf{o} \leftarrow \textsc{DeltaRule}\big(\text{queries } \boldsymbol{\phi}_{\mathbf{q}},\ \text{keys } \boldsymbol{\phi}_{\mathbf{k}},\ \text{values } \mathbf{v},\ \text{gates } \alpha,\ \text{step sizes } \eta^{(L)}\big)$

In the chunkwise form the difference is confined to the intra-chunk term. Writing the chunk's queries, keys and values as matrices and $\boldsymbol{\Gamma}_{/}$ for the matrix of decay ratios, the two reads are

$$\text{unshifted:}\quad \mathbf{O} = \boldsymbol{\Gamma}\mathbf{Q}\mathbf{S}_c + \text{tril}\left(\mathbf{Q}\mathbf{K}^\top \odot \boldsymbol{\Gamma}_{/}\right)\mathbf{V}, \qquad \text{shifted:}\quad \mathbf{O} = \boldsymbol{\Gamma}\mathbf{Q}\mathbf{S}_c + \text{tril}\left(\mathbf{Q}\mathbf{K}^\top \odot \boldsymbol{\Gamma}_{/}, -1\right)\mathbf{V}, \tag{13}$$

where $\text{tril}(\cdot, -1)$ is the strictly lower-triangular part. The boundary state $\mathbf{S}_c$ contains only tokens of earlier chunks, so the inter-chunk term is identical in both forms; only the causal mask of the intra-chunk attention changes, from $j \le t$ to $j < t$. This is how the shifted kernels are implemented: we copied FLA's chunkwise gated-linear-attention and gated-delta-rule kernels and changed the intra-chunk mask in the forward output kernel and its two backward kernels (for the query/key gradients and for the value gradient) from a causal to a strictly causal comparison of the token indices, leaving the inter-chunk state recurrence, the WY solve and the chunk-boundary bookkeeping untouched. The same one-line change applied to the delta-rule kernel's final read gives the shifted delta-rule scan used for the hidden layer in Section C.2 and, with the query and value operands exchanged, DeltaRuleGrad. At decode time ($T = 1$) the shifted read is the read of the cached state before the token's update, so no special case is needed.

Algorithm 1 is implemented with three chunkwise kernels of the FLA library and their transposes: the gated-linear-attention scan for the hidden layers (Eq. (4) and its transpose Eq. (6)), the gated-delta-rule scan for the hidden layer in the delta-rule form of Eq. (14), and the gated-delta-rule kernel with the query and value operands of its final read exchanged for DeltaRuleGrad (Section C.3). The shift is implemented inside the chunk: the intra-chunk attention is strictly lower-triangular for the key-side read and lower-triangular including the diagonal for the query-side read, and the inter-chunk state is the same for both. Every iteration recomputes the hidden-layer scans from the current pseudo-labels; the slow-weight terms $\mathbf{W}_0^{(l)}\mathbf{h}_t^{(l-1)}$ and the iteration-0 features are computed once. Training backpropagates through all N iterations with the kernels' chunkwise backward passes;

iteration n stores the pseudo-labels, the features and the Jacobians of σ, and the readout states at chunk boundaries are recomputed in the backward pass rather than stored.

C.2 Delta rule for the hidden layers

Step 1 evolves each hidden layer's fast weights by the gated linear attention Eq. (4), which sums the writes $-\eta_j^{(l)}\delta\mathbf{z}_j^{(l)}\mathbf{h}_j^{(l-1)\top}$ without ever erasing. The same weights can be evolved by a gated delta rule. With the unit key $\hat{\mathbf{k}}_t = \mathbf{h}_t^{(l-1)}/\|\mathbf{h}_t^{(l-1)}\|$ and write rate $\beta_t = \eta_t^{(l)}$, the delta rule is

$$\begin{aligned}\Delta\mathbf{W}_t^{(l)} &= \alpha_t\Delta\mathbf{W}_{t-1}^{(l)} + \beta_t\big(\mathbf{y}_t - \mathbf{r}_t\big)\hat{\mathbf{k}}_t^\top, \\ \mathbf{r}_t &= \alpha_t\Delta\mathbf{W}_{t-1}^{(l)}\hat{\mathbf{k}}_t, \qquad \mathbf{y}_t = \tilde{\mathbf{r}}_t - \frac{\eta_t^{(l)}\|\mathbf{h}_t^{(l-1)}\|}{\beta_t}\,\delta\mathbf{z}_t^{(l)},\end{aligned} \tag{14}$$

where $\mathbf{r}_t$ is the read of the key from the decayed weights and the target $\mathbf{y}_t$ is built from $\tilde{\mathbf{r}}_t$, the same read taken on the trajectory of the previous iteration. Substituting $\mathbf{y}_t$,

$$\Delta\mathbf{W}_t^{(l)} = \alpha_t\Delta\mathbf{W}_{t-1}^{(l)} - \eta_t^{(l)}\,\delta\mathbf{z}_t^{(l)}\,\mathbf{h}_t^{(l-1)\top} + \beta_t\big(\tilde{\mathbf{r}}_t - \mathbf{r}_t\big)\hat{\mathbf{k}}_t^\top, \tag{15}$$

which is Eq. (3) plus a correction that erases the part of the previous iteration's read that the current trajectory does not reproduce. At the fixed point $\tilde{\mathbf{r}}_t = \mathbf{r}_t$ and the correction vanishes, so the two forms agree there; along the iterates the delta-rule form erases stale writes rather than accumulating them. It is computed by the same chunkwise gated-delta-rule kernel as the readout, at the same cost as the linear-attention scan; Table 1 compares the two.

C.3 The readout gradient

Following the notation of Yang et al. (2024b); Tumma et al. (2026), write $\mathbf{S}_t = \Delta\mathbf{W}_t^{(L)\top}$ for the *fast* part of the readout, which decays as $\bar{\mathbf{S}}_{t-1} = \alpha_t\mathbf{S}_{t-1}$, and keep the slow readout $\mathbf{W}_0^{(L)}$ as a fixed offset, so that the weight token t reads is $\bar{\mathbf{S}}_{t-1} + \mathbf{W}_0^{(L)\top}$. (Equivalently, the full state $\mathbf{S}_t + \mathbf{W}_0^{(L)\top}$ decays toward $\mathbf{W}_0^{(L)\top}$: $\alpha_t(\mathbf{S}_{t-1} + \mathbf{W}_0^{(L)\top}) + (1-\alpha_t)\mathbf{W}_0^{(L)\top}$.) With the corrected values $\tilde{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{W}_0^{(L)}\boldsymbol{\phi}_t$ (or their centered form, Section C.5) and $\mathbf{u}_t = \eta_t^{(L)}(\tilde{\mathbf{v}}_t - \bar{\mathbf{S}}_{t-1}^\top\boldsymbol{\phi}_t)$ the value that token t writes, the fast readout's recurrence is $\mathbf{S}_t = \bar{\mathbf{S}}_{t-1} + \boldsymbol{\phi}_t\mathbf{u}_t^\top$, and token t's inner loss and the gradient that Step 3 needs are

$$\ell_t = \tfrac{1}{2}\big\|\bar{\mathbf{S}}_{t-1}^\top\boldsymbol{\phi}_t - \tilde{\mathbf{v}}_t\big\|^2, \qquad \mathbf{e}_t = \eta_t^{(L)}\,\frac{\partial\ell_t}{\partial\boldsymbol{\phi}_t} = -\big(\bar{\mathbf{S}}_{t-1} + \mathbf{W}_0^{(L)\top}\big)\,\mathbf{u}_t. \tag{16}$$

The chunkwise gated-delta-rule kernel of Yang et al. (2024b; 2025), as implemented in FLA, already forms every quantity on the right: the chunk-boundary states, the intra-chunk decays and the corrected values $\mathbf{u}_j$ of a chunk, which it obtains jointly from the WY representation. DeltaRuleGrad therefore reuses that kernel and changes only its final read, exchanging the roles of the queries and the values; its cost equals one delta-rule pass. The chunkwise form follows.

C.4 Chunkwise computation of the readout gradient

We use the chunk notation of Yang et al. (2024b; 2025); Tumma et al. (2026). Partition the sequence into chunks of C tokens and, for a chunk, stack its features, values, step sizes and outputs as $\boldsymbol{\Phi}, \mathbf{V} \in \mathbb{R}^{C\times d}$, $\boldsymbol{\beta} = (\eta_t^{(L)})_{t\in c}$ and $\mathbf{O}$, with $\mathbf{S}_c$ the readout state at the chunk's start and $\gamma_t = \prod_{i\le t}\alpha_i$ the cumulative decay within the chunk (so that $\gamma_{t,j} = \gamma_t/\gamma_j$); $\boldsymbol{\Gamma} = \mathrm{diag}(\gamma_t)$ and $\mathbf{L} = \mathrm{diag}(\boldsymbol{\beta})$. In this transposed convention the gated delta rule advances the state by $\mathbf{S}_t = (\mathbf{I} - \eta_t\boldsymbol{\phi}_t\boldsymbol{\phi}_t^\top)\,\bar{\mathbf{S}}_{t-1} + \eta_t\boldsymbol{\phi}_t\tilde{\mathbf{v}}_t^\top$, and the transitions of a chunk, applied in order from the left, have the WY form $(\mathbf{I} - \eta_C\boldsymbol{\phi}_C\boldsymbol{\phi}_C^\top)\cdots(\mathbf{I} - \eta_1\boldsymbol{\phi}_1\boldsymbol{\phi}_1^\top) = \mathbf{I} - \boldsymbol{\Phi}^\top\mathbf{U}$ with

$$\mathbf{U} = \mathbf{T}\,\mathbf{L}\,\boldsymbol{\Phi}, \qquad \mathbf{T} = \big(\mathbf{I} + \mathrm{tril}(\mathbf{L}\,\boldsymbol{\Phi}\boldsymbol{\Phi}^\top, -1)\big)^{-1}, \tag{17}$$

a $C \times C$ triangular solve against the Gram matrix of the chunk's features (with the decays folded into $\boldsymbol{\Phi}$ as $\boldsymbol{\Gamma}^{-1}\boldsymbol{\Phi}$ and $\boldsymbol{\Gamma}\boldsymbol{\Phi}$ on the two sides, which we suppress). The rows $\mathbf{u}_j$ of the corrected values

$$\mathbf{U}_\mathbf{v} = \mathbf{T}\,\mathbf{L}\,\big(\tilde{\mathbf{V}} - \boldsymbol{\Phi}\,\mathbf{S}_c\big) \tag{18}$$

are exactly the writes $\mathbf{u}_j = \eta_j(\tilde{\mathbf{v}}_j - \bar{\mathbf{S}}_{j-1}^\top \boldsymbol{\phi}_j)$ of Section C.3, expressed without the token-by-token state, and the state entering token t of the chunk is

$$\bar{\mathbf{S}}_{t-1} \;=\; \gamma_t\, \mathbf{S}_c \;+\; \sum_{\substack{j\in c\\ j<t}} \gamma_{t,j}\, \boldsymbol{\phi}_j\, \mathbf{u}_j^\top. \tag{19}$$

The kernel computes, per chunk, $\mathbf{T}$, $\mathbf{U_v}$, the next boundary state $\mathbf{S}_{c+1} = \gamma_C \mathbf{S}_c + \boldsymbol{\Phi}^\top \boldsymbol{\Gamma}_C \mathbf{U_v}$ (the inter-chunk recurrence, a short scan over chunks), and the output

$$\mathbf{O} \;=\; \boldsymbol{\Gamma}\, \boldsymbol{\Phi_q}\, \mathbf{S}_c \;+\; \mathrm{tril}\left(\boldsymbol{\Phi_q} \boldsymbol{\Phi}^\top \odot \boldsymbol{\Gamma}_{/}\right) \mathbf{U_v}, \tag{20}$$

where $\boldsymbol{\Gamma}_{/}$ holds the ratios $\gamma_{t,j}$ and the tril mask is causal; this is Eq. (19) substituted into $\mathbf{o}_t = \mathbf{S}_t^\top \boldsymbol{\phi}_{\mathbf{q},t}$ (the slow term $\mathbf{W}_0^{(L)} \boldsymbol{\phi}_{\mathbf{q},t}$ is added outside the kernel), a product with the boundary state plus a causal intra-chunk attention with scores $\boldsymbol{\phi}_{\mathbf{q},t}^\top \boldsymbol{\phi}_j$ and values $\mathbf{u}_j$.

The feature gradient $\mathbf{e}_t = -(\bar{\mathbf{S}}_{t-1} + \mathbf{W}_0^{(L)\top})\mathbf{u}_t$ of Eq. (16) substitutes the same Eq. (19) and gives, per token and per chunk,

$$\begin{aligned} \mathbf{e}_t \;&=\; -\left(\gamma_t\, \mathbf{S}_c + \mathbf{W}_0^{(L)\top}\right) \mathbf{u}_t \;-\; \sum_{\substack{j\in c\\ j<t}} \gamma_{t,j}\, \left(\mathbf{u}_j^\top \mathbf{u}_t\right) \boldsymbol{\phi}_j, \\ \mathbf{E} \;&=\; -\boldsymbol{\Gamma}\, \mathbf{U_v}\, \mathbf{S}_c^\top - \mathbf{U_v} \mathbf{W}_0^{(L)} \;-\; \mathrm{tril}\left(\mathbf{U_v} \mathbf{U_v}^\top \odot \boldsymbol{\Gamma}_{/}, -1\right) \boldsymbol{\Phi}, \end{aligned} \tag{21}$$

a product with the boundary state plus a causal intra-chunk attention in which the corrected values act as both queries and keys and the features as values. Comparing Eqs. (20) and (21): the two expressions share $\mathbf{S}_c$, $\boldsymbol{\Gamma}$, $\mathbf{T}$ and $\mathbf{U_v}$, and differ only in which operands enter the final read, $(\boldsymbol{\Phi_q}, \boldsymbol{\Phi}, \mathbf{U_v})$ for the output against $(\mathbf{U_v}, \mathbf{U_v}, \boldsymbol{\Phi})$ for the gradient, and in the strict lower-triangular mask, which excludes $j = t$ and implements the shifted arrangement. DELTARULEGRAD is therefore the gated-delta-rule kernel with the query and value operands of its final read exchanged, with the offset $\mathbf{W}_0^{(L)\top}$ entering its final read only; its cost equals that of one delta-rule pass. Because $\mathbf{T}$ depends on the features, which change with every iteration, it is recomputed in each of the N iterations and in the final scans; what can be computed once per layer is the corresponding solve of the hidden layer's delta rule (Section C.2), whose keys are the fixed inputs $\mathbf{k}_t$. The transposed reads of Step 3, Eq. (6), have the same structure with plain linear attention: values $-\eta_j^{(l)} \mathbf{h}_j^{(l-1)}$ in place of $\mathbf{U_v}$, and no triangular solve.

C.5 CENTERING

The centered readout of Section 3.4 is

$$f_{\mathbf{W}}(\mathbf{k}_t) = \mathbf{W}^{(L)} \boldsymbol{\phi}_t - \mathbf{W}_0^{(L)} \boldsymbol{\phi}_{0,t} = \Delta\mathbf{W}^{(L)} \boldsymbol{\phi}_t + \mathbf{W}_0^{(L)} \left(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{0,t}\right), \tag{22}$$

the fast readout applied to the features plus the slow readout applied to their drift from the slow features. With it, the inner loss of token t is $\ell_t = \frac{1}{2}\|\Delta\bar{\mathbf{W}}_{t-1}^{(L)} \boldsymbol{\phi}_t + \mathbf{W}_0^{(L)}(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{0,t}) - \mathbf{v}_t\|^2$. Writing $\tilde{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{W}_0^{(L)}(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{0,t})$ for the corrected values, this is the uncentered loss with $\mathbf{v}_t$ replaced by $\tilde{\mathbf{v}}_t$, so the fast readout $\Delta\mathbf{W}^{(L)}$ follows the same gated delta rule as before on the corrected values, and its trajectory is computed by the same kernel. The feature gradient of Eq. (5) becomes

$$\mathbf{e}_t \;=\; \eta_t^{(L)} \left(\Delta\bar{\mathbf{W}}_{t-1}^{(L)} + \mathbf{W}_0^{(L)}\right)^\top \left(\Delta\bar{\mathbf{W}}_{t-1}^{(L)} \boldsymbol{\phi}_t - \tilde{\mathbf{v}}_t\right), \tag{23}$$

which DELTARULEGRAD returns when its readout weight is offset by $\mathbf{W}_0^{(L)}$; the offset enters the kernel's final read only. The output is $\mathbf{o}_t = \Delta\mathbf{W}_t^{(L)} \boldsymbol{\phi}_{\mathbf{q},t} + \mathbf{W}_0^{(L)}(\boldsymbol{\phi}_{\mathbf{q},t} - \boldsymbol{\phi}_{0,\mathbf{q},t})$, one gated-delta-rule pass on the corrected values plus a matrix product. The slow features $\boldsymbol{\phi}_0$ of the keys are exactly those produced by iteration 0, where $\mathbf{D} = \mathbf{0}$, so they cost nothing extra; the slow features of the queries need one additional pass through the slow MLP. The fixed point of Section 3.3 is unchanged in form: it is the gradient sequence of the sequential recurrence with the centered readout. $\mathbf{W}_0^{(L)}$ is trained by backpropagation like every other slow parameter; because the forward pass depends on it through $\boldsymbol{\phi}_t - \boldsymbol{\phi}_{0,t}$, its gradient is a true gradient and no stop-gradient is involved.

D MODEL CONFIGURATIONS AND TRAINING RECIPE

The reported experiments use the hard unit-ball projection $\mathbf{x}/\max(1, \|\mathbf{x}\|)$ specified below. The soft clip is used only for the theoretical analysis in Sections B and 3.3.

All models share one backbone: a stack of pre-norm residual blocks, each a token-mixing layer followed by a SwiGLU MLP, with RMSNorm and tied embeddings, 24 layers at both scales, $d = 1024$ with 8 heads of dimension 128 at 340M and $d = 2048$ with 16 heads of dimension 128 at 1.3B. The token-mixing layer is ParaTTT, Gated DeltaNet, Mamba-2, softmax attention with RoPE, or TTT-MLP; the recurrent layers apply a short causal convolution of width 4 to the query, key and value projections and an output gate. Mamba-2 uses expansion 1 and 4 groups, TTT-MLP its published layout of 16 heads of dimension 64 with a block of 16 tokens, and LaCT its published recipe with a chunk of 2048 tokens, without the sliding-window branch, so that it is a pure fast-weight model. ParaTTT uses the memory of Section 3.1 with one hidden layer of width d and SiLU, the delta rule for the hidden layer and centering (Section 3.4), a per-token learning rate that is the sigmoid of a linear function of the input, the decay gate of Gated DeltaNet shared by both layers, damping $\lambda = 0.8$, and $N = 2$ iterations during training unless stated. Every ParaTTT model starts from the initialization of the Gated DeltaNet model of the same size, so that the two differ only in the memory. Table 2 lists the models of Table 1. All models are trained with FLAME on SlimPajama (parquet shards of the 100B-token subset) with the Llama tokenizer, sequence length 4096, 8 sequences per GPU with two accumulation steps on 8 GPUs (0.5M tokens per step), AdamW ($\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.01), peak learning rate 4×10^{-4} with 1024 warmup steps and cosine decay to 4×10^{-5}, gradient clipping at norm 1, bf16, seed 42 and the same data order. The 340M models train for 30k steps and the 1.3B models for 190k steps; the 1.3B ParaTTT model trains on two nodes with a sharding that reproduces the one-node data order. Every ParaTTT model is initialized from the Gated DeltaNet initialization of the same size (the slow weights of the memory are new parameters, initialized with the standard scheme and zero fast weights), so that at initialization the two models differ only in the memory.

Model	Params	Mixing layer	Memory / state	Notes
340M, 24 layers, $d = 1024$				
Gated DeltaNet	342M	8 heads $\times$ 128	128×128 per head	conv 4, output gate
Mamba-2	342M	8 heads $\times$ 128, expand 1, 4 groups	128×128 per head	scalar decay
Transformer	341M	8 heads $\times$ 128, RoPE	KV cache	YaRN for RULER
TTT-MLP	349M	16 heads $\times$ 64, block 16	MLP $64 \to 256 \to 64$ per head	inner LR base 0.1
LaCT (no SWA, conv)	340M	4 heads, chunk 2048	SwiGLU fast weights	Muon updates, conv 4
ParaTTT	348M	8 heads $\times$ 128	MLP $128 \to 128 \to 128$ per head	$N = 2$, $\lambda = 0.8$, delta-rule body, centered
1.3B, 24 layers, $d = 2048$				
Gated DeltaNet	1.30B	16 heads $\times$ 128		
Mamba-2	1.30B	16 heads $\times$ 128, expand 1, 4 groups		
Transformer	1.30B	16 heads $\times$ 128, RoPE		
ParaTTT	1.31B	16 heads $\times$ 128	MLP $128 \to 128 \to 128$ per head	$N = 2$

Table 2: Models of Table 1. Parameter counts include embeddings. The ParaTTT memory has a hidden layer of width d_{head} with SiLU, a bias, the unit-ball projection on the features, and a learned per-head gain on the fast pre-activation; its learning rates are $\eta_t = \text{sigmoid}(\mathbf{w}_\eta^\top \mathbf{x}_t + b_\eta)$ per head and layer, its decay is the Gated DeltaNet gate shared by both layers.

E EVALUATION PROTOCOL

Perplexity and zero-shot. wikitext-103 word perplexity and LAMBADA (OpenAI) perplexity, and accuracy on PIQA, HellaSwag (normalized), ARC-easy, ARC-challenge (normalized), WinoGrande, BoolQ and SciQ, all zero-shot with lm-evaluation-harness (Gao et al., 2023). ParaTTT models are evaluated at their training N with the chunkwise prefill.

Recall. SWDE, FDA, TriviaQA (the cloze version of Arora et al. (2024b)) and DROP are generation tasks: the model greedily decodes an answer after the prompt and the score is whether the gold answer appears in the generation, case-insensitively, on the full evaluation sets. ParaTTT prefills the prompt with the chunkwise iteration at its training N and decodes token by token with the exact recurrence; TTT-MLP decodes with a block-faithful cache at batch 1; the other models use their standard cached decoding.

RULER. The eight needle-in-a-haystack tasks of RULER (Hsieh et al., 2024) (single 1–3, multi-key 1–3, multiquery, multivalue) at 4k, 8k, 16k and 32k tokens with 100 samples per task; the table reports the average. Attention models use YaRN (Peng et al., 2024b) with a scale of the evaluation length over 4096.

Synthetic tasks. Two-layer models of width 256 (head dimension 128) with the same backbone as the language models, trained from scratch for 4k steps with AdamW, batch 128, over learning rates $\{3\times10^{-4}, 10^{-3}, 3\times10^{-3}\}$ and three seeds; the long-horizon settings (parity at 512 and 1024, MQAR at 512 with 32 pairs) are also trained for 8k steps at $\{10^{-4}, 3\times10^{-4}\}$, and MQAR with 64 pairs at width 512. Figure 5 reports, per model and setting, the learning rate (and budget) with the best mean final accuracy over the three seeds. ParaTTT models train and evaluate at $N = 2$ (or $N = 5$).

Per-position perplexity. 698 held-out SlimPajama documents (validation and test splits) of at least 4097 tokens, each contributing its first 4096 tokens, so that every position is the same number of tokens into a document; the mean NLL per position is binned in 27 log-spaced bins and reported as perplexity with one standard error.

F ADDITIONAL RESULTS

F.1 SYNTHETIC NONLINEAR-STATE TASKS

Three synthetic tasks isolate what a nonlinear memory adds, each learned from scratch by two-layer models of width 256 with the backbone of Section D (protocol in Section E; Fig. 5). *Cumulative parity* asks, at every position, for the parity of the number of ones seen so far; *k-hop* asks for the token reached by following k pointer links back through the sequence; *multi-query associative recall* (MQAR) asks for the values of previously stored keys.

Parity probes whether a memory can maintain a state that flips with selected inputs. It is not a universal separation between nonlinear and linear memories: negative transition eigenvalues allow linear RNNs to solve it. The relevant limitation concerns finite-precision linear RNNs restricted to nonnegative real eigenvalues at arbitrary sequence lengths (Grazzi et al., 2025). Our finite-length experiments test the architectures and training settings used here. In Fig. 5(a), Gated DeltaNet and Mamba-2 fall from 74–77% at length 128 to 63% at 256 and 52–55% at 1024, approaching chance, and the two-layer attention model stays below 63% at every length, whereas ParaTTT reaches 96–100% at lengths up to 256, 79–90% at 512 and 78–80% at 1024, with the model trained at $N = 5$ best at every length. Turning the loop off ($N = 0$) drops ParaTTT to chance, indicating that the iteration is important in this setup. TTT-MLP sits between the two groups (87% at length 128, 58% at 1024). Its block form evaluates gradients at the weights entering each 16-token block, although each token's readout incorporates preceding within-block updates (Sun et al., 2025). The frozen gradient evaluations differ from the online recurrence that ParaTTT targets; these results alone do not establish that this difference causes the parity gap.

The other two tasks are solvable by a linear memory and serve as controls. On k-hop (Fig. 5b) the recurrent models with an erasing update, ParaTTT and Gated DeltaNet, solve every k, Mamba-2

breaks at $k = 4$, and the two-layer attention model cannot compose hops at this depth. MQAR (Fig. 5c) is a capacity test: Gated DeltaNet and ParaTTT solve it up to 32 pairs, with ParaTTT more seed-sensitive at length 1024 and the $N = 5$ model the more reliable of the two, and only attention solves 64 pairs, which exceeds the capacity of every recurrent state at this width. ParaTTT thus keeps the linear recurrences' recall while achieving stronger state-tracking performance in these experiments.

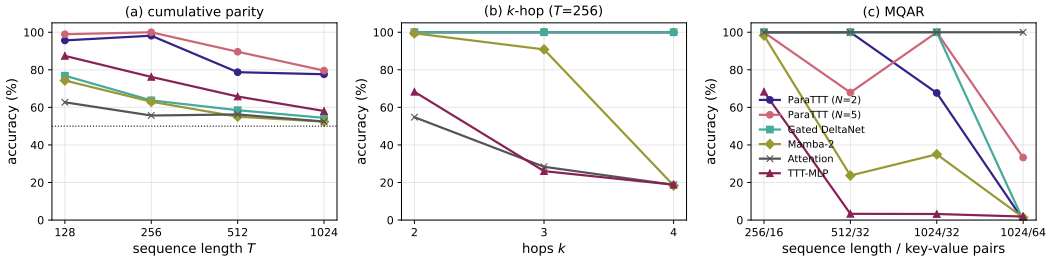


Figure 5: Synthetic nonlinear-state tasks (two layers, width 256; mean over three seeds at the best learning rate). (a) Cumulative parity against sequence length; the dotted line is chance. (b) k-hop pointer following at $T = 256$. (c) Multi-query associative recall for increasing sequence length and number of key-value pairs.

Evaluation at other iteration counts. Table 3 gives the full sweeps behind Fig. 2(c,d), including the 1.3B model, on wikitext, LAMBADA and SWDE.

Model	Metric	$N = 0$	1	2	3	5	7	10
340M, trained at $N = 2$	wikitext	61.3	25.76	**25.46**	25.54	25.67	25.75	25.87
	LAMBADA	332	28.43	**26.93**	27.37	28.21	28.40	28.60
	SWDE	7.4	39.3	**41.9**	40.8	39.4	39.6	38.7
340M, trained at $N = 5$	wikitext	61.7	26.29	25.63	25.51	**25.47**	25.49	25.52
	LAMBADA	375	31.60	29.31	28.69	**28.44**	28.52	28.72
	SWDE	5.0	33.8	43.1	44.0	**45.0**	44.2	43.8
340M, no centering	wikitext	26.28	26.09	26.06	26.06	26.08	26.09	26.12
	LAMBADA	33.84	32.97	32.62	32.40	32.14	32.05	32.01
	SWDE	34.5	36.7	36.6	36.8	36.2	35.8	35.8
340M, linear-attention body	wikitext	55.4	26.43	**25.62**	25.75	26.00	25.74	25.64
	LAMBADA	179	26.76	**25.88**	26.08	26.11	26.13	26.15
	SWDE	7.7	29.4	28.4	28.7	28.9	28.7	27.7
1.3B, trained at $N = 2$	wikitext	73.1	15.36	**15.21**	15.32	15.49	15.58	15.70
	LAMBADA	348	**8.94**	9.00	9.40	9.91	10.16	10.38
	SWDE	6.8	62.4	**62.8**	58.2	53.6	48.9	44.4

Table 3: Perplexity (wikitext, LAMBADA) and SWDE accuracy of the ParaTTT models as a function of the number of iterations at evaluation. Bold: best N for the centered models.

LongBench. Table 4 reports LongBench at four context lengths for every model in Table 1.

F.2 Prior TTT layers rely on a sliding-window attention branch

The published TTT architectures that scale beyond the original TTT-Linear and TTT-MLP do not evaluate their memory alone. LaCT (Zhang et al., 2026) pairs its large-chunk fast-weight update with a sliding-window attention branch in every layer and reports only the combination; Titans (Behrouz et al., 2025b) reports its long-context results for the hybrids that add attention to the memory (MAC, MAG and MAL). A window of 2048 tokens covers half of a 4096-token training context, so the attention branch alone already accounts for local modeling, and the fast-weight memory only has to contribute beyond the window.

Table 5 measures how much of LaCT's performance comes from this branch, at 340M with the training recipe of Section D. With its sliding window LaCT is a competitive language model. Re-

Model	4k	8k	16k	32k
340M, 24 layers, 15.7B tokens				
GDN	11.9	12.1	12.3	12.2
Mamba2	11.8	11.9	11.9	11.9
Transformer	10.9	11.5	10.9	10.2
TTT-MLP	9.8	9.0	8.1	8.1
LaCT (no SWA, conv)	4.9	4.5	4.2	4.2
PTTT (ours)	12.4	12.5	<u>12.4</u>	12.1
no centering	<u>12.6</u>	<u>12.6</u>	**12.5**	**12.5**
linear-attention body	9.5	9.6	9.3	9.7
trained at $N{=}5$	**12.7**	**12.7**	**12.5**	<u>12.4</u>
1.3B, 24 layers, 100B tokens				
GDN	<u>17.4</u>	<u>17.7</u>	<u>17.8</u>	<u>17.7</u>
Mamba2	15.1	15.1	15.1	15.2
Transformer	**19.3**	**20.8**	**21.0**	**19.2**
PTTT (ours)	16.4	16.9	17.2	17.3

Table 4: LongBench: average score over the 16 English tasks (100 samples per task) at context lengths 4k–32k, for the models of Table 1. Best per column and size in bold, second best underlined. All models trained at 4k; ParaTTT is within a point of Gated DeltaNet at every length and scale, and attention leads at 1.3B.

	Perplexity ↓		Zero-shot acc. ↑						
Model	Wiki	Lamb	PIQA	Hella	ARC-e	Wino	ARC-c	BoolQ	SciQ
LaCT with SWA-2048 (as published)	25.51	33.3	66.2	35.8	45.1	51.7	23.6	61.1	79.3
LaCT without SWA (Table 1)	463.2	$9.4{\times}10^4$	53.7	25.0	28.8	50.1	24.1	60.1	21.2
PTTT (ours), no attention	25.46	26.9	65.3	37.7	48.6	51.2	24.1	59.8	79.0

Table 5: LaCT at 340M with and without its sliding-window attention branch, same recipe and data as Table 1; recall for the no-SWA model is in Table 1.

moving the window and training the identical fast-weight memory alone, with the same chunk size of 2048, yields a model that is essentially context-free: wikitext perplexity in the hundreds, LAMBADA at chance, and zero recall (Table 1). The large-chunk update itself is the cause, since every token of a 2048-token chunk is scored with the weights from before the chunk, so within a chunk the memory sees nothing of the current text and the model has to learn the local statistics from attention. The sliding window is therefore not an add-on but a crutch that the large-chunk approximation needs. ParaTTT uses no attention branch: every model in Table 1 is a pure recurrence, its memory is updated at every token, and it is trained and evaluated without a local branch of any kind.

F.3 Centering ablation

Table 6 isolates the readout parameterization. All three models share the recipe, initialization and data order of Section D and differ only in how the readout layer $\mathbf{W}^{(L)}$ is parameterized:

- **Zero-initialized readout.** $\mathbf{W}_0^{(L)} = \mathbf{0}$ and $f_{\mathbf{W}}(\mathbf{k}_t) = \Delta\mathbf{W}^{(L)}\boldsymbol{\phi}_t$, the standard delta-rule state that starts at zero and decays toward zero (the "no centering" row of Table 1). By Eq. (5) the hidden layers receive no gradient until the fast readout has grown.
- **Learned initial readout, uncentered.** A slow $\mathbf{W}_0^{(L)}$ is learned and $f_{\mathbf{W}}(\mathbf{k}_t) = (\Delta\mathbf{W}^{(L)} + \mathbf{W}_0^{(L)})\boldsymbol{\phi}_t$: the fast weights decay toward $\mathbf{W}_0^{(L)}$ instead of zero, so the hidden layers receive gradient from the first token, but the output carries the bias $\mathbf{W}_0^{(L)}\boldsymbol{\phi}_t$ before any test-time learning has taken place. This model has exactly the parameters of the centered one.
- **Centered readout.** Eq. (22), the ParaTTT default.

Read against the Gated DeltaNet reference, the three rows form a ladder. The zero-initialized loop already improves recall over the linear recurrence (SWDE +4, NIAH at 4k +3) but costs 3 points of LAMBADA perplexity. The learned initial readout repairs LAMBADA to the Gated DeltaNet

	Perplexity ↓		Zero-shot acc. ↑							Recall ↑				NIAH ↑				LongBench ↑			
Readout	Wiki	Lamb	PIQA	Hella	ARC-e	Wino	ARC-c	BoolQ	SciQ	SWDE	FDA	TQA	DROP	4k	8k	16k	32k	4k	8k	16k	32k
Gated DeltaNet	26.57	29.66	65.2	36.5	47.1	51.5	24.1	56.6	78.4	32.5	17.8	35.7	15.3	25.9	17.0	13.7	9.3	11.9	12.1	12.3	12.2
Zero-init	26.06	32.62	64.5	36.8	48.4	50.6	**25.3**	58.2	78.9	36.6	19.6	37.0	14.3	29.0	**21.1**	15.0	12.2	**12.6**	**12.6**	**12.5**	**12.5**
Learned init, uncentered	26.13	29.06	**65.8**	36.7	46.1	**52.6**	24.4	59.7	**80.4**	37.7	19.2	37.1	13.5	32.5	19.2	14.4	14.4	11.6	11.2	11.2	11.4
Centered (PTTT)	**25.46**	**26.93**	65.3	**37.7**	**48.6**	51.2	24.1	**59.8**	79.0	**41.9**	**20.3**	**39.6**	**14.4**	**32.6**	20.7	**15.6**	**16.0**	12.4	12.5	12.4	12.1

Table 6: Centering ablation at 340M. Three parameterizations of the readout layer, trained with the same recipe, initialization and data order; the gated DeltaNet baseline is shown for reference. Columns as in Table 1; best of the three PTTT variants per column in bold.

level and leaves recall essentially unchanged, showing that giving the hidden layers a gradient at the start of a sequence matters. Centering buys the rest with the same parameters: 0.7 wikitext and 2.1 LAMBADA perplexity, 4 points of SWDE and 2.5 of TriviaQA, and the largest NIAH gain at 32k. The commonsense tasks move within noise across the ladder, and each row is a single seed. The gain of centering over the uncentered learned readout is therefore not explained by the extra slow readout: it comes from removing the bias $\mathbf{W}_0^{(L)}\boldsymbol{\phi}_t$ so that the readout at the start of a sequence is zero while its Jacobian is not.

Centering for TTT-MLP. We also trained TTT-MLP with the centered readout of Eq. (22). It does not require centering: the results moved by about two points in either direction (SWDE $33.1 \to 35.7$, FDA $16.2 \to 18.2$, perplexity unchanged within noise), so we report the original recipe.

G Kernel implementation and efficiency details

The ParaTTT layer is built from the chunkwise Triton kernels of FLA with a chunk size of 64: the gated-linear-attention and gated-delta-rule forward and backward kernels, the transposed reads of Eq. (6) (the same kernel with the roles of keys and values exchanged) and DeltaRuleGrad, which is the gated-delta-rule kernel with the final read transposed; the pointwise epilogues (nonlinearity, Jacobian, projection, damping) are fused into two Triton kernels described in Section G.1. Per iteration and hidden layer the layer stores the pseudo-labels, the features and the Jacobian, each $T \times d_{\text{head}}$ per head, and the readout states at chunk boundaries are recomputed in the backward pass; peak memory at $N = 2$ is $2.5\times$ that of Gated DeltaNet, both for one layer at batch 8 (6.5 against 2.6 GiB at $T = 4096$) and for the full 340M model in training (Table 7). For Fig. 3(a,b) we time one token-mixing layer in isolation (forward and backward, bf16, batch 8, 20 iterations after 3 warm-up iterations) on one MI325X; TTT-MLP uses its fused Triton kernel where it exists (head dimension 64, block size 16) and a `torch.compile`d version of its PyTorch scan otherwise, which is $2.2\times$ slower than the fused kernel where both exist. For (c) we measure the training throughput of the full 340M models on one node of eight MI325X GPUs with the training recipe of Section D, averaged over 50 steps after warm-up.

G.1 The fused epilogue kernels

With one hidden layer ($L = 2$) the only operations of an iteration that are not scans are pointwise in the token: the nonlinearity and projection that turn the hidden pre-activation into the features (Step 1), and their transposed Jacobians, which are all that Step 3 needs, because the transposed fast weight of Eq. (6) is only required to reach a layer below the first. Both are implemented as one Triton kernel each, with one program per (sequence, token, head) that holds the d_{head}-vector of its row in fp32 registers; every reduction is over this vector, so the kernels are memory-bound and read and write each activation once. We write $s(x) = 1/(1 + e^{-x})$ for the sigmoid, $\sigma(x) = x\, s(x)$ for SiLU and $n(\mathbf{x}) = \mathbf{x} \min(1, 1/\|\mathbf{x}\|)$ for the unit-ball projection.

Forward epilogue. The hidden pre-activation of token t is the sum of a slow part and a fast part,

$$\mathbf{z}_t \;=\; \underbrace{\mathbf{W}_0^{(1)}\mathbf{k}_t + \mathbf{b}^{(1)}}_{\text{one matmul per forward}} \;+\; \kappa\, \underbrace{\alpha_t \Delta\mathbf{W}_{t-1}^{(1)}\mathbf{k}_t}_{\text{shifted scan, Eq. (4)}}, \qquad \mathbf{h}_t = \sigma(\mathbf{z}_t), \qquad \boldsymbol{\phi}_t = n(\mathbf{h}_t), \tag{24}$$

where κ is the learned per-head gain of Section 3.4. The slow part does not depend on the pseudo-labels, so it is computed once per layer forward and shared by all iterations; the scan output is read

in bf16 straight from the layout the scan writes, without the copy that separating the two arms of a SwiGLU would need (the value arm is the constant 1 in our models). The kernel forms $\mathbf{z}_t$, $\mathbf{h}_t$ and $\boldsymbol{\phi}_t$ in fp32, stores $\boldsymbol{\phi}_t$ in bf16 for the readout kernels, and saves $\mathbf{z}_t$ and $\mathbf{h}_t$ for the backward epilogue; $\|\mathbf{h}_t\|$ is recomputed there from $\mathbf{h}_t$.

Backward epilogue. DELTARULEGRAD returns $\mathbf{e}_t = \eta_t^{(L)}\, \partial \ell_t / \partial \boldsymbol{\phi}_t$ Eq. (5). The backward epilogue turns it into the value the hidden scan consumes in the next iteration, in five register-resident steps:

$$\mathbf{g}_t = \mathbf{e}_t / \eta_t^{(L)}, \qquad \mathbf{g}'_t = \begin{cases} \big(\mathbf{g}_t - \hat{\mathbf{h}}_t\, \hat{\mathbf{h}}_t^\top \mathbf{g}_t\big)/\|\mathbf{h}_t\| & \|\mathbf{h}_t\| \geq 1 \\ \mathbf{g}_t & \|\mathbf{h}_t\| < 1 \end{cases}, \qquad \widehat{\delta \mathbf{z}}_t = \sigma'(\mathbf{z}_t) \odot \mathbf{g}'_t, \tag{25}$$

with $\hat{\mathbf{h}}_t = \mathbf{h}_t/\|\mathbf{h}_t\|$ and $\sigma'(x) = s(x)\big(1 + x(1 - s(x))\big)$. The second step is the transposed Jacobian of the projection, $(\mathbf{I} - \hat{\mathbf{h}}\hat{\mathbf{h}}^\top)/\|\mathbf{h}\|$ outside the unit ball and the identity inside it, and the third that of SiLU; together they are the transposed Jacobian $\mathbf{H}_t^\top$ of Section 3.3, with the hard projection in place of the soft clip. The value the scan receives, $\hat{\mathbf{v}}_t = -\eta_t^{(1)} \widehat{\delta \mathbf{z}}_t$, attaches the hidden learning rate and the sign with which the pseudo-label enters as a value. The damped Picard step Eq. (7) is fused into the same store, $\mathbf{v}_t^{[n]} = \mathbf{v}_t^{[n-1]} + \lambda(\hat{\mathbf{v}}_t - \mathbf{v}_t^{[n-1]})$, with $\mathbf{v}^{[0]} = \mathbf{0}$ handled without a read, and the result is written directly in the $(B, T, H, d_{\text{head}})$ layout the scan reads, so an iteration is four kernel launches per layer: the hidden scan, the forward epilogue, DELTARULEGRAD and the backward epilogue. The kernels also implement a pre-activation RMSNorm, a per-token norm clip of $\widehat{\delta \mathbf{z}}_t$ (hard or soft), a skip connection and a second SwiGLU arm, none of which the reported models use.

Training through the iteration. End-to-end training backpropagates through all N iterations, so each epilogue is an autograd function whose backward is itself a fused kernel: the vector-Jacobian product of Eq. (24) (through σ' and the projection Jacobian) and that of Eq. (25), which differentiates the SiLU Jacobian a second time (σ'') and the learning rates. For decoding, the hidden scan, the forward epilogue and the DELTARULEGRAD step of one token are fused into a single kernel per (sequence, head) with the states resident in registers, since at that length the launch latency of the chunkwise kernels dominates.

H COST OF THE HIDDEN-LAYER UPDATE AND OF CENTERING

Table 7 measures what the two refinements of Section 3.4 cost, in the two settings of Fig. 3: one token-mixing layer in isolation and the full 340M model in training.

Readout parameterization. The three readouts of Table 6 run on the same kernels and differ only in the readout offset and in the extra terms of Eq. (22). Relative to the zero-initialized readout, the uncentered learned readout adds the offset $\mathbf{W}_0^{(L)}$ to the final read of DELTARULEGRAD and costs 12% of the layer time at 4k and 9% of training throughput. Centering adds to this one more pass of the slow MLP over the queries (the features $\boldsymbol{\phi}_{0,\mathbf{q},t}$ of Section C.5) and the corrected values, which is a further 1–2% of layer time and throughput and 9 GiB of activation memory. Centering therefore costs about 11% of training throughput in total, against the gains of Table 6.

Hidden-layer update. The delta-rule body replaces the gated-linear-attention scan of the hidden layer with a gated-delta-rule scan of the same shape, plus the transposed reads of Eq. (5) that the delta rule's gradient needs (Section C.2); the readout side of the layer is unchanged. In the reference implementation, which supports both updates, the delta-rule body costs 17% of the layer time at every length (35.6 against 30.4 ms at 4k) and 1.7 GiB more activation memory per layer for the extra chunk states; in training this is 13% of throughput (32.5k against 37.4k tokens/s) and 30 GiB of peak memory per GPU. Together with Table 1 (SWDE 28.4 → 41.9, NIAH at 4k 7.2 → 32.6), this is the better-spent of the two refinements.

Variant	Layer forward+backward (ms) at T 1k	2k	4k	8k	16k	Training, 8 GPUs tokens/s ($\times 10^3$)	GiB/GPU
Readout parameterization (delta-rule body, optimized kernels)							
Zero-init readout	7.3	13.1	26.2	52.2	104.1	42.7	131
Learned-init readout, uncentered	8.3	14.8	29.4	58.8	116.7	38.7	136
Centered readout (PTTT)	8.5	15.1	29.8	59.8	118.8	38.0	145
Hidden-layer update (centered readout, reference implementation)							
Linear-attention body	8.4	15.3	30.4	60.7	120.3	37.4	107
Delta-rule body (PTTT)	10.0	18.0	35.6	70.9	140.3	32.5	138
Reference							
Gated DeltaNet	2.1	3.3	6.3	11.9	23.2	112.1	58

Table 7: Cost of the hidden-layer update and of the readout parameterization at 340M, $N = 2$. Left: forward+backward time of one token-mixing layer (bf16, batch 8, $d = 1024$, 8 heads of dimension 128, one MI325X), the setting of Fig. 3(a). Right: training throughput and peak memory of the full model on one node of eight MI325X GPUs with the recipe of Section D. The readout rows use the optimized kernels of Section G; the two bodies are timed in the reference implementation, which supports both updates, so the body rows are comparable with each other but slower than the readout rows.

I Convergence versus compute at other sequence lengths

Figure 6 shows the comparison of Fig. 4(a) at every sequence length from 256 to 4096 with the same inner MLP, step size and decay. The picture is the same at every length: TTT-MLP's error is set by its block size and its time by the number of sequential blocks T/b, whereas ParaTTT's error is set by the iteration count and its time, being a fixed number of chunkwise-parallel scans, is nearly independent of T at batch 1, so the gap widens with the sequence length.

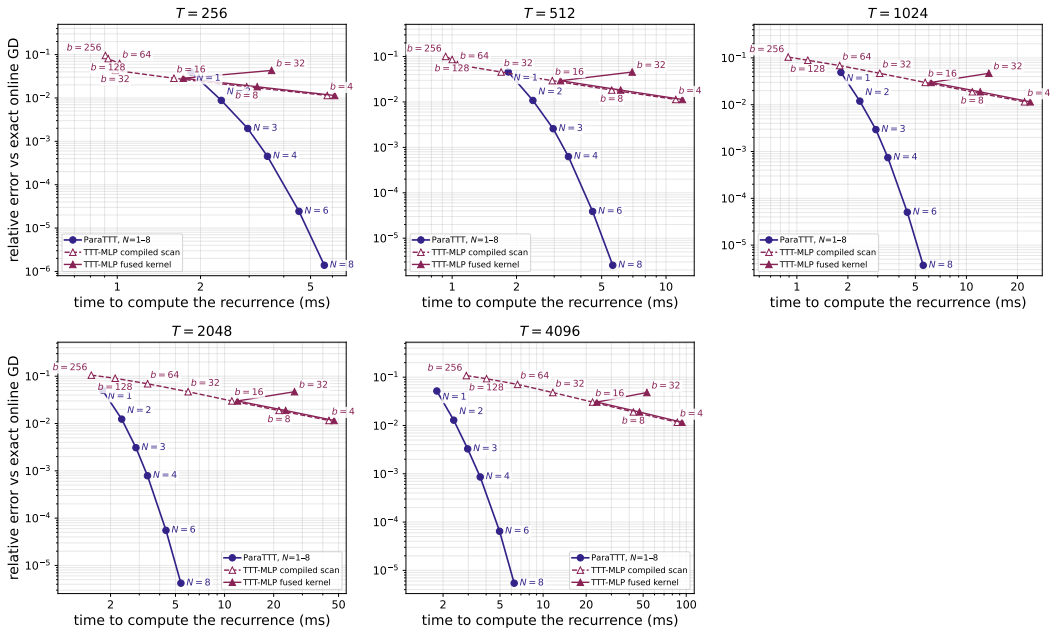


Figure 6: Relative error against exact online GD versus the time to compute the forward recurrence of one 340M layer (batch 1), at five sequence lengths; same setting and legend as Fig. 4(a).