HiCA-RNN: Multi-Scale Credit Assignment for Long-Term Dependency Learning

Anonymous authors
Paper under double-blind review

Abstract

Windowing a recurrent computation changes how credit is transported; adding intermediate targets changes where credit enters. Neither is sufficient alone: reinitializing the state at each window boundary does not shorten the backward horizon unless the boundary is also detached, and intermediate targets supply local training signals without preserving long-range credit. In this work we introduce HiCA-RNN that combines both within a single fast–slow recurrence. A shared ungated cell processes each window, a gated slow state connects the windows, and a feedback recurrence supplies detached targets to intermediate fast states. The states remain connected across the full sequence: the task loss is trained by full-graph BPTT, while the target loss introduces credit directly at selected states. We derive the resulting adjoint and identify the contribution of each target to the shared-parameter gradient. The derivation exposes a constraint on the method: local credit is useful only when its magnitude, its transport through the slow dynamics, and its direction relative to the task gradient are favorable. Neither short windows nor orthogonal feedback ensures these properties. HiCA-RNN achieves 99.0% on sequential MNIST and 97.4% on permuted MNIST, exceeding tuned LSTMs on both, obtains the lowest Copy loss on all horizons, and matches or improves on burn-in LSTMs in eight of twelve system-identification settings. On sequential CIFAR-10 HiCA-RNN uses approximately $6\times$ less peak training memory than a parameter-matched LSTM, which achieves slightly better accuracy; LSTM also remains stronger on Adding.

1 Introduction

A recurrent model can retain information about an early input and still fail to learn how to use it. The task gradient must reach the parameters that produced the relevant state, often through a long product of recurrent Jacobians. Gates and multiscale recurrences alter that product by providing routes through which information and gradients can persist (Hochreiter & Schmidhuber, 1997; Dey & Salem, 2017; Koutnik et al., 2014; Chung et al., 2015; Mujika et al., 2017). Target-based methods intervene at a different point: they place a learning signal at an intermediate representation, closer to the computation being trained (Ororbia & Mali, 2019; Ororbia et al., 2023; Manchev & Spratling, 2020). The first changes how credit travels; the second changes where it enters. Combining these interventions in a recurrent model requires care. Dividing a sequence into windows does not itself truncate the task gradient if a state connects successive windows. Conversely, a target at an intermediate state creates a local gradient but says nothing about whether that gradient supports the sequence-level objective. Likewise, controlling the norm of a feedback operator cannot establish that its targets point in a useful direction. A learning rule that uses both mechanisms must therefore specify the computational graph precisely and distinguish the signal introduced by each target from the dynamics that subsequently transport it.

We introduce HiCA-RNN to make that distinction explicit in a single learner. A shared ungated $\tanh$ cell processes each fixed-length window, and a gated slow state carries context to the next window. Within a window, a feedback recurrence constructs detached targets for the fast states. The fast and slow states remain connected across the full sequence: we differentiate the task loss through the complete recurrent graph, while a local target loss supplies additional gradients at the fast states within each window. Thus the targets change the backward signal without changing the forward

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.

state transition. Setting their loss weight λ to zero gives the identical fast–slow architecture trained only by the task loss.

Our analysis follows the learning signal from the state at which it is introduced to the shared recurrent parameters. We derive the exact adjoint equations for the implemented procedure and express each target's contribution as a source at its fast state followed by transport through the fast and slow dynamics (Theorem 1). This decomposition separates three requirements that are easy to conflate: the target must generate a suitably scaled signal; the intervening dynamics must transmit that signal; and the resulting parameter update must agree sufficiently with descent on the task loss. We characterize the within-window source magnitude (Eq. 18), give conditional bounds for slow-state transport (Proposition 2), and derive alignment and descent conditions (Propositions 3 and 4). These results are conditional statements about when target injection can help, rather than a guarantee that every target improves learning. They also show why neither short windows nor orthogonal feedback alone resolves long-range credit assignment. We study whether this combination allows a simple recurrent core to perform well on tasks that demand memory at different scales. HiCA-RNN achieves 99.0% accuracy on sequential MNIST and 97.4% on permuted MNIST, above tuned LSTM baselines. It obtains the lowest Copy loss at three of four horizons and matches or improves on burn-in LSTMs (Schiller et al., 2026) in eight of twelve system-identification settings. On sequential CIFAR-10 it uses approximately $6\times$ less peak training memory than a parameter-matched LSTM, which achieves slightly higher accuracy; LSTM also performs better on Adding. These results describe the full system.

Contributions.

- We formulate HiCA-RNN as a fast–slow recurrent model with detached intermediate targets and an untruncated task gradient (Section 3). The target mechanism can be switched off with $\lambda = 0$ without altering the forward architecture.
- We derive its exact adjoint and identify where target-generated credit enters, how it reaches shared parameters, and the conditions under which its contribution can support task descent (Section 4).
- We evaluate the full system on synthetic memory, pixel-sequence classification, and nonlinear system-identification tasks, with accuracy and training-memory comparisons against LSTM baselines, and give a per-step time and memory analysis showing where the cost difference arises (Sections 4.4 and 5).

2 Related work

Multiscale recurrence. Clockwork RNNs use modules with different update periods, hierarchical multiscale RNNs learn boundaries between time scales, and Fast–Slow RNNs couple a fast lower-level cell to a slowly updated upper-level state (Koutnik et al., 2014; Chung et al., 2015; Mujika et al., 2017). The idea that components with distinct time constants capture global structure better than a single-rate network predates these models (Mozer, 1991; Schmidhuber, 1992), and fast-weight approaches (Ba et al., 2016; Irie & Gershman, 2026) pursue a different notion of "fast" in which the weights themselves change during a sequence. All of these are trained with the task loss alone and therefore modify only how credit is transported. HiCA additionally changes where credit is introduced. An extended discussion, including gating, stabilization, and state-space approaches, appears in Appendix A. **Target-based and local learning.** LRA-E develops feedback-driven local targets, recurrent LRA extends them to temporal models, and target propagation through time studies recurrent target-based training (Ororbia & Mali, 2019; Ororbia et al., 2023; Manchev & Spratling, 2020). These methods are motivated by the locking and weight-transport constraints of BPTT (Jaderberg et al., 2017; Lillicrap et al., 2016), but recurrent LRA alone can suffer vanishing learning signals (Manchev & Garcia-Peraza-Herrera, 2025), and target propagation through time inherits the same depth problem it aims to sidestep. HiCA differs from these methods in retaining the exact task gradient through its connected fast–slow graph and using detached targets only as an augmentation. It differs from a fast–slow model trained by BPTT in having the auxiliary source terms of Theorem 1. Our analysis makes that auxiliary contribution explicit; the corresponding empirical attribution requires the matched $\lambda{=}0$ ablation discussed in Section 6.

3 HiCA-RNN

This section specifies the forward computation, the intermediate targets, and the training objective; Section 4 analyzes the gradient this produces.

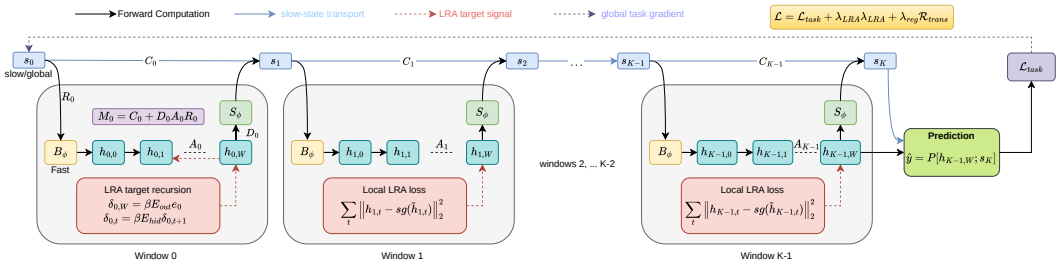


Figure 1: Fast–slow computation in `hybrid` mode. The slow state initializes each fast window, and the window endpoint enters the next slow transition; both dependencies remain differentiable within a forward call. Detached targets supply additional loss gradients at fast states without detaching the recurrent states.

3.1 Fast–slow recurrence

Divide a sequence of length T into $K = \lceil T/W \rceil$ windows, where window $k \in \{0, \dots, K-1\}$ has $W_k = \min(W, T - kW)$ steps, fast states $h_{k,t} \in \mathbb{R}^{d_h}$, and incoming slow state $s_k \in \mathbb{R}^{d_s}$. The forward recurrence is

$$h_{k,0} = B_\phi(s_k) = \tanh(W_b s_k + b_b)\,, \tag{1}$$

$$h_{k,t} = F_\theta(h_{k,t-1}, x_{k,t}) = \tanh(\mathcal{N}_f(W_x x_{k,t} + W_h h_{k,t-1} + b))\,, \qquad t = 1, \dots, W_k, \tag{2}$$

$$s_{k+1} = S_\psi(s_k,\, h_{k,W_k})\,. \tag{3}$$

with $\mathcal{N}_f$ an optional fast LayerNorm and $1 \le t \le W_k$. The slow transition used in all experiments is

$$z_k = [h_{k,W_k}; s_k], \quad g_k = \sigma(W_g z_k + b_g), \quad c_k = \tanh(W_c z_k + b_c), \tag{4}$$

$$s_{k+1} = \mathrm{LN}_s(g_k \odot s_k + (1 - g_k) \odot c_k)\,. \tag{5}$$

The fast cell has no gate; gating and LayerNorm are confined to the slow transition. The parameter groups θ, ϕ, ψ are disjoint, and "fast" and "slow" describe state-update frequency, not changes to the weights at inference time. All reported experiments use `hybrid` mode (Figure 1): neither the bridge $B_\phi(s_k)$ nor the endpoint h_{k,W_k} is detached, so the slow state carries both forward information and task gradients across window boundaries, and a boundary does not truncate BPTT. For independent examples $s_0 = 0$; when a slow state is carried between separate forward calls it is detached, $s_0 = \mathrm{sg}(s_{\text{previous}})$, where sg leaves the forward value unchanged and has zero derivative in automatic differentiation.

Task losses. We use two output heads, $O_\omega(h, s) = W_o[h; s] + b_o$ and $Q_\zeta(h) = Vh + b_q$. Sequence classification and Adding use $O_\omega(h_{K-1,W_{K-1}}, s_K)$ with cross-entropy and MSE, respectively; Copy and system identification apply $Q_\zeta(h_{k,t})$ at each step with token cross-entropy and MSE. A burn-in mask excludes specified positions from the task loss only; it does not remove their states from the recurrent graph or from the auxiliary loss. Appendix B specifies the reductions and boundary cases.

3.2 Intermediate targets and the training objective

Targets are constructed in windows with at least two steps, $\mathcal{K}_a = \{k : W_k \ge 2\}$. At the end of each eligible window, define

$$e_k = \mathrm{sg}(o_k^\star - Q_\zeta(h_{k,W_k}))\,, \qquad o_k^\star = \begin{cases} Q_\zeta(h_{k+1,1}), & k < K-1, \\ y^\star, & k = K-1. \end{cases} \tag{6}$$

An intermediate residual compares the output head at the current endpoint with its value at the first state of the next window, imposing next-window consistency; it is not recursively corrected by the final label. The final residual uses a supervised target $y^\star$: a one-hot class vector, the one-hot of the final Copy token, a regression output vector, or the scalar Adding target. A final one-step window is ineligible. Feedback matrices $E_o \in \mathbb{R}^{d_h \times q}$ and $E_h \in \mathbb{R}^{d_h \times d_h}$ define the targets

$$\delta_{k,W_k} = \beta E_o e_k, \qquad \delta_{k,t} = \beta E_h \delta_{k,t+1} \;\; (t = W_k - 1, \dots, 1), \qquad \widehat{h}_{k,t} = \mathrm{sg}(h_{k,t} + \delta_{k,t}). \tag{7}$$

E_h is orthogonally initialized and fixed; E_o is updated by a separate feedback-matching step. Each target is fixed when the forward-parameter objective is differentiated; the recurrent state $h_{k,t}$ is not.

For a batch of B equally long sequences, let $N_a = Bd_h \sum_{k\in\mathcal{K}_a} W_k$. We minimize

$$\mathcal{L}_{\text{main}} = \mathcal{L}_{\text{task}} + \lambda\mathcal{L}_a + \mu\mathcal{R}_{\text{proxy}}, \qquad \mathcal{L}_a = \frac{1}{N_a}\sum_{i=1}^{B}\sum_{k\in\mathcal{K}_a}\sum_{t=1}^{W_k}\|h_{i,k,t} - \widehat{h}_{i,k,t}\|^2, \tag{8}$$

with $\mathcal{L}_a = 0$ when $N_a = 0$. Differentiating with each target held fixed gives the direct auxiliary gradient at an eligible fast state,

$$r_{i,k,t} := \lambda\frac{\partial\mathcal{L}_a}{\partial h_{i,k,t}}\bigg|_{\widehat{h}\text{ fixed}} = -\frac{2\lambda}{N_a}\delta_{i,k,t}, \tag{9}$$

and zero at excluded states. This is the gradient introduced *at* the state; its effect on earlier states and shared parameters follows the connected fast–slow graph. Setting $\lambda = 0$ removes these sources while retaining the forward architecture, task loss, and transition regularizer.

Forward and feedback updates. After backpropagation and gradient-norm clipping, a main Adam step updates the forward parameters. A separate Adam step then updates E_o by minimizing the mean over the batch and d_h of $\|E_o\bar{e}_i - (V^+)^\top\bar{e}_i\|^2$, where $\bar{h} = \text{sg}(h_{K-1,W_{K-1}})$ is the terminal fast state retained from the forward pass, V^+ is the weight matrix of the post-update head Q_{ζ^+}, and $\bar{e} = \text{sg}(y^\star - Q_{\zeta^+}(\bar{h}))$ uses the supervised target. Q_ζ receives task gradients on dense-output tasks; on final-output tasks it has no task-gradient path and remains at initialization, so feedback matching uses its initialized projection. E_h is fixed in both steps. Algorithms 1–2 give the update order and Appendix H records the optimizer's gradient-buffer behavior.

Transition regularization. The implemented regularizer is a Frobenius-norm proxy on detached fast activations,

$$\mathcal{R}_{\text{proxy}} = \frac{\gamma}{BN_r}\sum_{i=1}^{B}\sum_{k}\sum_{t=2}^{W_k}\Big(\sqrt{\|\,\text{diag}(1-\text{sg}(h_{i,k,t})^{\odot 2})W_h\|_F^2 + 10^{-8}} - 1\Big)^2, \tag{10}$$

where $N_r = \sum_k(W_k - 1)$ and the penalty is zero when $N_r = 0$. Its direct gradient acts on W_h only, with effective coefficient $\mu\gamma$. It omits LayerNorm derivatives and does not constrain every singular value of the recurrent Jacobian, so we treat it as a heuristic rather than a gradient-preservation guarantee (Appendix D).

4 Analysis of the Learning Signal

We separate three questions: where credit enters the state graph, how it is transported, and when its direction helps the task. The analysis is for a fixed-length batch (allowing a common final partial window), fixed parameters during a trajectory, and detached targets. We suppress the batch index and omit the regularizer, whose direct derivative is given in Appendix D.

Define $J_{k,t} = \partial h_{k,t}/\partial h_{k,t-1}$, $R_k = DB_\phi(s_k)$, $C_k = \partial_s S_\psi(s_k, h_{k,W_k})$, and $D_k = \partial_h S_\psi(s_k, h_{k,W_k})$.
Set

$$A_k = J_{k,W_k}\cdots J_{k,1}, \quad M_k = C_k + D_k A_k R_k, \quad P_{k,j:t} = J_{k,j}\cdots J_{k,t+1}, \tag{11}$$

with $P_{k,t:t} = I$. Derivatives include gates and normalization. Let $u_{k,t}$ and v_k be direct partial derivatives of the task loss at the fast and slow nodes, with other state nodes held fixed; let $r_{k,t}$ be the direct auxiliary derivative of Eq. equation 9, including its factor λ; and put $f_{k,t} = u_{k,t} + r_{k,t}$.

4.1 WHERE CREDIT ENTERS

Theorem 1 (Hierarchical credit-source decomposition). *Let a_k and $b_{k,t}$ be the total slow and fast adjoints. Then $a_K = v_K$ and*

$$b_{k,t} = P_{k,W_k:t}^\top D_k^\top a_{k+1} + \sum_{j=t}^{W_k} P_{k,j:t}^\top f_{k,j}, \tag{12}$$

$$a_k = M_k^\top a_{k+1} + v_k + R_k^\top \sum_{j=1}^{W_k} P_{k,j:0}^\top f_{k,j}. \tag{13}$$

For $G_{k,t} = \partial_\theta F_\theta(h_{k,t-1}, x_{k,t})$ with inputs held fixed, $\nabla_\theta(\mathcal{L}_{\text{task}} + \lambda\mathcal{L}_a) = \sum_{k,t} G_{k,t}^\top b_{k,t}$, provided θ occurs only in the fast cell and s_0 is independent of θ within the forward call.

The result follows by unrolling $b_{k,t} = J_{k,t+1}^\top b_{k,t+1} + f_{k,t}$ and differentiating both outgoing edges of s_k; the proof is in Appendix C. The term $G_{k,t}^\top r_{k,t}$ enters the shared-parameter gradient without traversing any future fast Jacobian; the remaining terms transport the same source to earlier states and windows. Target sources add terms to the adjoint without removing task-gradient paths, but they can oppose the task gradient, and the theorem does not assert that a source is nonzero or helpful: $E_o e_k$ can vanish, and $G_{k,t}^\top$ can annihilate its direction. Sections 4.2 and 4.3 address these two gaps.

4.2 HOW CREDIT IS TRANSPORTED, AND WHAT WINDOWING DOES NOT BUY

The slow-state derivative factors as

$$\frac{\partial s_K}{\partial s_0} = M_{K-1} \cdots M_0. \tag{14}$$

There are K macro factors, but each contains W_k fast transitions through A_k. Expanding the product gives a sum of paths, including the direct slow path $C_{K-1} \cdots C_0$ and paths traversing every fast block, so the number of macro factors does not establish a K-step backward horizon. Contributions can also cancel: $C_k = I$ and $D_k A_k R_k = -I$ yield $M_k = 0$.

Proposition 2 (Conditional transport). *Let $\mathcal{V}_0 \neq \{0\}$ and $\mathcal{V}_{k+1} = M_k \mathcal{V}_k$.*
If $m_k\|v\| \le \|M_k v\| \le L_k\|v\|$ on $\mathcal{V}_k$, with $0 < m_k \le L_k$, then

$$\left(\prod_k m_k\right)\|v\| \le \|M_{K-1} \cdots M_0 v\| \le \left(\prod_k L_k\right)\|v\|, \qquad v \in \mathcal{V}_0. \tag{15}$$

If $e^{-\epsilon_k} \le m_k \le L_k \le e^{\epsilon_k}$ and $\sum_k \epsilon_k \le C$, gains lie in $[e^{-C}, e^{C}]$.

The proof multiplies the one-step inequalities along transported vectors. The bound is on forward sensitivities restricted to $\mathcal{V}_0$; the adjoint uses the transposed factors $M_k^\top$ and needs a corresponding bound on its own directions. Constant bounds can still produce exponential decay or growth: in a scalar recurrence with $C_k = 0$, $D_k = R_k = 1$, and $J_{k,t} = a$, the total gain is still a^T. Restricted subspaces are necessary because LayerNorm has null directions. Neither the penalty nor the architecture guarantees the premises; they must be measured.

The same factors govern how a source at one window reaches parameters used in earlier windows. Writing $U_k = ds_k/d\theta$ and $H_{k,t} = \sum_{j=1}^{t} P_{k,t:j} G_{k,j}$ for the fast-state parameter sensitivity with s_k held fixed,

$$U_{k+1} = M_k U_k + D_k H_{k,W_k}, \qquad \frac{dh_{k,t}}{d\theta} = H_{k,t} + P_{k,t:0} R_k U_k, \tag{16}$$

with $U_0 = 0$. Even a single auxiliary loss at (k,t) therefore has a within-window derivative and an inherited cross-window derivative transported by products of the M_k. This is why bounded target magnitude (next subsection) and well-conditioned transport are separate requirements: the first controls the source, the second controls what survives the journey.

Credit from inputs to the actual prediction. Let $\ell = K - 1$ and let $Q_r^x = \partial h_{r,W_r}/\partial \xi$ describe an input perturbation ξ in window r with s_r fixed. With $O_h = \partial_h O_\omega$ and $O_s = \partial_s O_\omega$, define

$Z_\ell = O_h A_\ell R_\ell + O_s M_\ell$. The final-output sensitivity is

$$\frac{\partial \widehat{y}}{\partial \xi} = \begin{cases} Z_\ell M_{\ell-1} \cdots M_{r+1} D_r Q_r^x, & r < \ell, \\ (O_h + O_s D_\ell) Q_\ell^x, & r = \ell, \end{cases} \tag{17}$$

where an empty product is the identity. Input injection and readout can annihilate a direction even when intermediate transitions preserve it, so a state-conditioning bound alone cannot prove useful task credit or predict an accuracy crossover. Both readout branches are necessary for the implemented final-output model.

4.3 Magnitude and direction of target-based credit

The feedback recursion gives, in exact arithmetic,

$$\delta_{k,t} = \beta^{W_k - t + 1} E_h^{W_k - t} E_o e_k, \qquad \|\delta_{k,t}\| = |\beta|^{W_k - t + 1} \|E_o e_k\| \tag{18}$$

when E_h is orthogonal, so by Eq. equation 9 $\|r_{k,t}\| = (2\lambda/N_a)|\beta|^{W_k - t + 1}\|E_o e_k\|$. Within one window, $|\beta| < 1$ attenuates direct sources geometrically toward the window's beginning; across windows and examples the profile is scaled by the varying e_k. Orthogonality controls a magnitude factor, not alignment with the true recurrent Jacobian transpose, and in finite precision small displacements can round away entirely, so practical attenuation can exceed the formula.

Proposition 3 (Endpoint alignment)**.** *Fix $Q(h) = Vh + b_q$, $o^\star$, and E_o. Let $e = o^\star - Q(h)$, $p = V^\top e \neq 0$, and $\Phi(h) = \frac{1}{2}\|Q(h) - o^\star\|^2$. If $\|(E_o - V^\top)e\| \leq \epsilon\|p\|$ for $0 \leq \epsilon < 1$, then $\langle \nabla_h \Phi, E_o e\rangle \leq -(1-\epsilon)\|p\|^2$. Moreover, $\Phi(h + \beta E_o e) < \Phi(h)$ whenever,*
$0 < \beta < 2(1-\epsilon)/(\|V\|_2^2(1+\epsilon)^2)$.

This follows by writing $E_o e = p + z$ and expanding the quadratic surrogate (Appendix C). The condition concerns a frozen projected squared-error objective at the window endpoint. Feedback matching does not establish it for every residual, and it does not imply descent of cross-entropy or of the shared-parameter task loss. The step from endpoint alignment to task descent requires one more condition.

Proposition 4 (Conditional task descent)**.** *Consider an unclipped gradient step without transition regularization or Adam preconditioning. Let $g = \nabla \mathcal{L}_{\text{task}}(\vartheta)$ and let $a = \nabla \mathcal{L}_a(\vartheta)$ with current targets fixed. If $\mathcal{L}_{\text{task}}$ has L-Lipschitz gradient along the update, then for $d = g + \lambda a$ and $\vartheta^+ = \vartheta - \eta d$,*

$$\mathcal{L}_{\text{task}}(\vartheta^+) \leq \mathcal{L}_{\text{task}}(\vartheta) - \eta\big(\|g\|^2 + \lambda\langle g, a\rangle\big) + \tfrac{L\eta^2}{2}\|d\|^2, \tag{19}$$

so strict descent holds for $0 < \eta < 2A/(L\|d\|^2)$ whenever $A = \|g\|^2 + \lambda\langle g, a\rangle > 0$.

The proof is the standard smoothness inequality (Appendix C). The content is the sign of $\langle g, a\rangle$: with $a = -2g/\lambda$ for $\lambda > 0$, the combined direction is $-g$ and the step ascends. Nothing in Eq. equation 18 or Proposition 3 fixes this sign, because the earlier-state surrogate derivative contains $P_{k,W_k:t}^\top V^\top e_k$ while the injected displacement contains $E_h^{W_k - t} E_o e_k$; orthogonality of E_h preserves norms, not directions. For the realized update direction d_{Adam}, the same bound applies with d_{Adam} in place of d, and momentum, clipping, and the proxy gradient can change $\langle g, d_{\text{Adam}}\rangle$. Taken together, Theorem 1, Proposition 2, Eq. equation 18, and Proposition 4 identify three separately measurable quantities—source magnitude, restricted transport gain, and $\langle g, a\rangle$—that we return to in Section 6.

4.4 Time and space complexity

The mechanism above changes the backward signal, not the forward cost; we now quantify the latter. Table 1 gives leading operation counts for the displayed architecture and a single-layer LSTM at equal fast width (derivation in Appendix D.3); C_{out} is the cost of the task-specific readout schedule, identical for both models. For fixed widths and window size, both have $O(T)$ training time and activation storage under full BPTT. The ungated fast recurrence uses one $d_h \times d_h$ product per token where an LSTM uses four; HiCA adds one dense E_h product per target step and bridge and gated-summary products once per window. In the regime $d_s = d_h = d$, $d_x, q \ll d$, and $W \geq 10$, the leading terms give $4d^2 + 15d^2/W$ multiply–adds per token for forward, backward, and target

Table 1: Leading per-sequence costs (batch factor omitted; $d_x = d_{\text{in}}$, $K = \lceil T/W \rceil$; l.o. collects biases and normalization gains).

Quantity	HiCA-RNN (hybrid)	LSTM
Stored parameters	$2d_h^2 + d_h(d_x{+}d_s{+}3q) + 2d_s(d_h{+}d_s) + d_s q$ + l.o.	$4d_h(d_x{+}d_h) + d_h q$ + l.o.
Forward time	$Td_h(d_x{+}d_h) + K[d_h d_s + 2d_s(d_h{+}d_s)] + C_{\text{out}}$	$4Td_h(d_x{+}d_h) + C_{\text{out}}$
Target construction (no grad)	$Td_h^2 + Kd_h q$	—
Backward time	$\approx 2\times$ forward	$\approx 2\times$ forward
Training activations	$O(Td_h + Kd_s)$	$O(Td_h)$
Inference state	$d_h + d_s$	$2d_h$
Sequential graph depth	$O(T + K)$ (targets: $O(W)$, parallel over windows)	$O(T)$

construction against about $12d^2$ for the LSTM: a factor of 2.2 at $W = 10$, approaching 3 as W grows or as d_s/d_h shrinks; outside that regime the ratio depends on d_x, d_s, W, and the readout schedule and must be read from the table. The stored-parameter count includes both HiCA heads and the feedback matrices (E_h is fixed but stored); the LSTM count assumes one head. Elementwise operations, normalization, the transition proxy, and feedback matching are lower order and omitted. The measured peak-memory gap in Section 5 is an implementation-level result, not a consequence of these counts.

5 Experiments

Setup We evaluate three task families: synthetic long-range dependency tasks (Temporal Order, Adding, Copy), pixel-sequence classification (sequential and permuted MNIST, sequential CIFAR-10), and nonlinear system identification (Silverbox, Wiener–Hammerstein, RLC). Baselines are a vanilla RNN and an LSTM. All models are tuned per task and, for the synthetic tasks, per sequence length, with Optuna under the same trial budget (Appendix G); selected configurations and parameter counts are listed in Table 7. All HiCA runs use the connected `hybrid` mode of Section 3. Synthetic and MNIST results are means and standard deviations over three seeds. System-identification results are reported over the top-3 runs of the tuning study and should be read as descriptive. Dataset details are in Appendix E.

Synthetic dependency tasks Temporal Order is four-way classification of the order of two marked symbols among distractors. Adding regresses the sum of two marked values. Copy reproduces a ten-symbol pattern after a delay and is scored by token cross-entropy over the full sequence. We evaluate $T \in \{50, 100, 150, 200\}$. Table 2 shows three patterns. On Temporal Order, HiCA and LSTM both reach 100% at every length, whereas the vanilla RNN degrades to chance at $T = 200$. On Copy, HiCA attains the lowest mean cross-entropy at $T = 50, 150, 200$, by one to two orders of magnitude over LSTM at $T = 50$ and 150; at $T = 100$ its mean is above LSTM with a large seed variance. On Adding, LSTM has the lowest MSE at every length; HiCA is within a factor of 1.4–3.6 of LSTM at $T = 50, 100, 200$ and a factor of 25 at $T = 150$, while the vanilla RNN fails for $T \geq 100$. Because configurations and capacities are selected independently per length, these curves do not trace a single model across horizons and should not be read as a monotonic length effect; Copy cross-entropy also includes blank positions, so recall-segment accuracy would be a sharper memory metric. We show learning trajectory of all models for copy task over all T's in Figure 6.

Sequential and permuted MNIST Each image is a sequence of 784 scalar pixels, classified after the final step; permuted MNIST applies one fixed permutation to every image. HiCA outperforms both baselines on both variants (Table 3), with the largest margin on permuted MNIST ($+5.2$ points over LSTM, $+2.7$ over RNN), the variant that removes local spatial structure and depends most on long-range memory. These are comparisons between tuned architectures and do not separate the auxiliary targets from the fast–slow architecture, normalization, or regularization (Section 6).

System identification We follow the burn-in protocol of Schiller et al. (2026) on Silverbox, Wiener–Hammerstein, and RLC, reporting scaled MSE ($\times 10^2$). RLC predicts capacitor voltage and inductor current; the other two datasets have scalar outputs. HiCA has the lowest mean error in eight of twelve settings: all four Wiener–Hammerstein lengths, Silverbox at $N \geq 200$, and RLC at $N = 100$. The tuned burn-in LSTM is better on Silverbox at $N = 100$ and on RLC for $N \geq 200$, where HiCA also shows high variance across runs. HiCA uses 554–10,160 parameters against a 361-parameter

Table 2: Test results on synthetic sequence tasks (mean ± std over 3 seeds; best in bold).

Task	T	**RNN**	**LSTM**	**HiCA-RNN**
Adding (MSE ↓)	50	$(1.22 \pm 0.25) \times 10^{-4}$	$\mathbf{(2.79 \pm 1.22) \times 10^{-5}}$	$(3.91 \pm 1.70) \times 10^{-5}$
	100	$(1.12 \pm 0.85) \times 10^{-1}$	$\mathbf{(1.47 \pm 0.30) \times 10^{-5}}$	$(5.29 \pm 1.59) \times 10^{-5}$
	150	$(1.16 \pm 0.92) \times 10^{-1}$	$\mathbf{(1.15 \pm 0.71) \times 10^{-5}}$	$(2.86 \pm 0.91) \times 10^{-4}$
	200	$(1.73 \pm 0.01) \times 10^{-1}$	$\mathbf{(1.52 \pm 0.56) \times 10^{-5}}$	$(3.99 \pm 0.33) \times 10^{-5}$
Copy (CE loss ↓)	50	1.6987 ± 0.0583	$(8.84 \pm 4.91) \times 10^{-4}$	$\mathbf{(3.85 \pm 0.70) \times 10^{-4}}$
	100	1.7244 ± 0.5493	0.4950 ± 0.5433	$\mathbf{(4.33 \pm 2.64) \times 10^{-4}}$
	150	1.9225 ± 0.1834	0.4055 ± 0.3008	$\mathbf{(4.64 \pm 1.12) \times 10^{-3}}$
	200	1.6298 ± 0.1706	0.3105 ± 0.0820	$\mathbf{0.0151 \pm 0.0070}$
Temporal Order (2-bit) (Acc. % ↑)	50	74.9 ± 41.9	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
	100	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
	150	62.1 ± 37.3	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
	200	27.0 ± 0.8	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$

Table 3: Sequential and permuted MNIST test accuracy (%, mean ± std over 3 seeds).

Task (Acc. % ↑)	RNN	LSTM	HiCA-RNN (ours)
Sequential MNIST	88.36 ± 3.51	98.64 ± 0.10	$\mathbf{99.0 \pm 0.1}$
Permuted MNIST	94.7 ± 0.2	92.2 ± 0.8	$\mathbf{97.4 \pm 0.22}$

LSTM, and stateful execution and burn-in settings differ across rows (Table 12), so the comparison characterizes tuned systems rather than isolating the learning rule.

Sequential CIFAR-10 and training memory Color sequential CIFAR-10 ($T = 1024$, one RGB pixel per step) is the longest horizon we test and the only one where we hold the parameter budget fixed across architectures. Table 5 compares HiCA and LSTM at ∼50K and ∼135K parameters under one protocol, alongside a vanilla RNN and a truncated-BPTT RNN at 50K; published results are listed in Appendix G.4. HiCA is within one point of LSTM at both budgets (-0.9 and -1.1), with lower seed variance, and is more than 30 points above both ungated RNN baselines, which fail at this horizon. It also uses $6.4\times$ and $6.1\times$ less peak training memory than the matched LSTM. These are implementation-level measurements: both models store $O(T)$ activations, so the gap reflects the ungated fast cell and small slow state rather than a better asymptotic bound (Section 4.4).

6 Limitations and Diagnostic Tests

Attribution. The experiments compare HiCA-RNN with tuned RNN and LSTM baselines but do not isolate target injection from the fast–slow architecture. Setting $\lambda = 0$ removes the auxiliary sources in Theorem 1 without changing the forward computation, so a comparison at fixed architecture and training budget, with λ and the proxy coefficient μ varied separately, is needed before a gain can be attributed to either component; recurrent target-based methods and established multiscale models are the natural further baselines. The reported gains on MNIST, Copy, and system identification are results for the full system.

Diagnostic tests. The analysis identifies conditions under which target injection helps; it does not predict that they always hold, and each is measurable during training. Eq. equation 18 fixes the direct auxiliary-source norm at each position within a window, conditional on its residual e_k, orthogonal E_h, and exact arithmetic; recording these norms tests the within-window scaling, whereas comparing norms across windows does not, because their residuals differ. Proposition 3 gives an endpoint condition, not a guarantee from feedback matching: measure $\|(E_o - V^\top)e_k\|/\|V^\top e_k\|$ where the denominator is nonzero and the angle between $E_o e_k$ and $V^\top e_k$. For task-level effects, record the task and auxiliary parameter gradients, their inner product $\langle g, a \rangle$ (Proposition 4), and the norm and direction of the realized optimizer update. For transport, measure the action of the full

Table 4: System identification, scaled MSE ($\times 10^2$, lower is better); HiCA over top-3 tuning runs. TBP $\bar{m}$ / m^* / sf: stateless LSTM with no burn-in / tuned burn-in (Schiller et al., 2026) / stateful LSTM.

Dataset	N	HiCA-RNN (top-3)	TBP $\bar{m}$	TBP m^* (burn-in)	TBP sf
Silverbox	100	0.1862 ± 0.0355	0.291	**0.142**	0.222
	200	$\mathbf{0.1249 \pm 0.0364}$	0.140	0.135	0.221
	500	$\mathbf{0.1093 \pm 0.0258}$	0.121	0.122	0.185
	1000	$\mathbf{0.0874 \pm 0.0494}$	0.117	0.112	0.146
Wiener–Hammerstein	100	$\mathbf{0.4836 \pm 0.0319}$	0.511	0.488	1.014
	200	$\mathbf{0.4264 \pm 0.0187}$	0.454	0.476	0.961
	500	$\mathbf{0.3534 \pm 0.0265}$	0.397	0.397	0.875
	1000	$\mathbf{0.3056 \pm 0.0174}$	0.372	0.372	0.737
RLC	100	$\mathbf{1.0793 \pm 0.1258}$	3.527	1.093	1.183
	200	1.4095 ± 1.0326	1.462	**0.307**	1.186
	500	1.1076 ± 0.8069	0.538	**0.228**	0.932
	1000	0.9940 ± 0.3263	0.402	**0.362**	0.871

Table 5: Sequential CIFAR-10, T=1024, identical protocol (45k/5k/10k split, Adam, lr 10^{-3}, batch 128, $\leq$100 epochs, early stopping), mean $\pm$ std over 3 seeds; HiCA uses W=16, λ=0.05. Peak memory: PyTorch peak allocated GPU memory during training.

Model	Params	Accuracy (%)	Peak mem. (MB)
This work, ∼50K parameters			
HiCA-RNN	49.6K	61.20 ± 0.20	**252**
LSTM	49.9K	$\mathbf{62.12 \pm 1.04}$	1603
Truncated BPTT RNN	49.9K	30.52 ± 0.19	23
RNN	49.9K	28.89 ± 0.52	1875
This work, ∼135K parameters			
HiCA-RNN	134.6K	61.64 ± 0.16	**406**
LSTM	135.0K	$\mathbf{62.74 \pm 0.40}$	2467

$M_k = C_k + D_k A_k R_k$ on relevant forward and adjoint directions; C_k alone omits the fast path and cannot test cross-window transport.

Scope. The synthetic experiments cover the reported horizons only, and where model size or tuning changes with sequence length they cannot isolate length dependence. HiCA-RNN retains full BPTT within each forward call and therefore retains its sequential backward dependencies. The next-window consistency targets are a modeling choice requiring ablation. On final-output tasks Q_ζ receives no task gradient, so feedback matching uses a projection that remains at initialization; a trained projection or an alternative target rule should be compared. The derivation assumes the stated fixed-length batching and detachment rules and does not establish behavior for other padding or state-carrying procedures.

7 Conclusion

Long-range credit assignment in RNNs has two separable parts: the paths along which error travels, set by the architecture, and the points at which error enters, set by the learning rule. HiCA-RNN changes the second while keeping the first exact, adding detached window-wise targets to full-graph BPTT on a fast–slow recurrence, and its adjoint equations make the two parts explicit. Each target is an additive source at its fast state, transported by the same Jacobian products that carry the task gradient. The analysis identifies when such sources help and shows that windowing and orthogonal feedback do not by themselves guarantee it. An ungated recurrent core trained this way competes with tuned LSTMs on pixel-sequence classification, Copy, and most system-identification settings at 2–3$\times$ fewer operations per step, while LSTM remains ahead on Adding and sequential CIFAR-10.

AI assistance. Generative AI assisted with code generation and debugging, literature discovery, language revision, analysis, and sentence restructuring. The authors are responsible for all statements, proofs, citations, implementation details, and experimental results.

Reproducibility. Section 3 and Appendix B specify the task-specific target rules, differentiation boundaries, feedback updates, normalization, loss reductions, and optimizer ordering. Appendix G lists every selected configuration, parameter count, and tuning protocol. Experiment scripts, tuning spaces, seed records, and checkpoint provenance will be released with the code.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pp. 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31, 2018.

Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications*, 11(1):3625, 2020. doi: 10.1038/s41467-020-17236-y.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Frederik Benzing, Marcelo Matheus Gauy, Asier Mujika, Anders Martinsson, and Angelika Steger. Optimal Kronecker-sum approximation of real time recurrent learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 604–613. PMLR, 2019.

Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3280–3287, Jul. 2019. doi: 10.1609/aaai.v33i01.33013280. URL https://ojs.aaai.org/index.php/AAAI/article/view/4200.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks. In *International Conference on Learning Representations*, 2019. URL https://mlanthology.org/iclr/2019/chang2019iclr-antisymmetricrnn/.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A. Hasegawa-Johnson, and Thomas S. Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017.

Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pp. 2067–2075, 2015.

Federico Danieli, Pau Rodriguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. Pararnn: Unlocking parallel training of nonlinear rnns for large language models. In *ICLR*, 2026. URL https://arxiv.org/abs/2510.21450.

Susmita Das, Amara Tariq, Thiago Santos, Sai Sandeep Kantareddy, and Imon Banerjee. Recurrent neural networks (rnns): architectures, training tricks, and introduction to influential research. *Machine learning for Brain disorders*, pp. 117–138, 2023.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, oct 2012. doi: 10.1109/msp.2012.2211477.

Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600. IEEE, 2017.

Shiyu Duan and José C. Príncipe. Training deep architectures without end-to-end backpropagation: A survey on the provably optimal methods. *IEEE Computational Intelligence Magazine*, 17(4): 39–51, 2022. doi: 10.1109/MCI.2022.3199624.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems 8 (NIPS)*, pp. 493–499, 1995.

N. Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W. Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-N7PBXqOUJZ.

Marco Forgione and Dario Piga. Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control*, 59:69–81, 2021. ISSN 0947-3580. doi: https://doi.org/10.1016/j.ejcon.2021.01.008. URL https://www.sciencedirect.com/science/article/pii/S0947358021000169.

Benyamin Ghojogh and Ali Ghodsi. *Recurrent Neural Networks and Long Short-Term Memory Networks*, pp. 205–229. Springer Nature Switzerland, Cham, 2026. ISBN 978-3-032-10738-1. doi: 10.1007/978-3-032-10738-1_8. URL https://doi.org/10.1007/978-3-032-10738-1_8.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

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

Kazuki Irie and Samuel J. Gershman. Fast weight programming and linear transformers: from machine learning to neurobiology. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=TDG8EkNmQR. Survey Certification.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pp. 1627–1635. PMLR, 2017.

Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks – with an erratum note. GMD Report 148, German National Research Center for Information Technology (GMD), 2001.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. doi: 10.1126/science.1091277.

Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk, Jonathan Binas, Michael C. Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 7640–7651, 2018.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1863–1871, Beijing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/koutnik14.html.

Federico Landi, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. Working memory connections for lstm. *Neural Networks*, 144:334–341, sep 2021. doi: 10.1016/j.neunet.2021.08.030. URL https://arxiv.org/pdf/2109.00020.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2015)*, volume 9284 of *Lecture Notes in Computer Science*, pp. 498–515. Springer, 2015. doi: 10.1007/978-3-319-23528-8_31.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018a.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466. IEEE, 2018b.

Zhong Li, Jiequn Han, Weinan E, and Qianxiao Li. On the curse of memory in recurrent neural networks: Approximation and optimization analysis. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=8Sqhl-nF50.

Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1): 13276, 2016.

Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

Soroor Malekmohamadi Faradonbe, Faramarz Safi-Esfahani, and Morteza Karimian-Kelishadrokhi. A review on neural turing machine (ntm). *SN Computer Science*, 1(6):333, 2020.

Ankur Mali, Alexander G Ororbia, Daniel Kifer, and C Lee Giles. Recognizing and verifying mathematical equations using multiplicative differential neural units. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 5006–5015, 2021.

Ankur Mali, Alexander Ororbia, Daniel Kifer, and Lee Giles. On the computational complexity and formal hierarchy of second order recurrent neural networks. *arXiv preprint arXiv:2309.14691*, 2023.

Nikolay Manchev and Luis C. Garcia-Peraza-Herrera. Can local representation alignment rnns solve temporal tasks? In *2025 9th International Conference on Computer, Software and Modeling (ICCSM)*, pp. 63–72, 2025. doi: 10.1109/ICCSM66818.2025.00019.

Nikolay Manchev and Michael Spratling. Target propagation in recurrent neural networks. *Journal of Machine Learning Research*, 21(7):1–33, 2020. URL http://jmlr.org/papers/v21/18-141.html.

Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations (ICLR)*, 2021.

Michael C. Mozer. Induction of multiscale temporal structure. In *Advances in Neural Information Processing Systems 4 (NIPS)*, pp. 275–282, 1991.

Asier Mujika, Florian Meier, and Angelika Steger. Fast-slow recurrent neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random Kronecker factors. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2584–2593. JMLR.org, 2017.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Alexander Ororbia, Ankur Mali, C. Lee Giles, and Daniel Kifer. Continual learning of recurrent neural networks by locally aligning distributed representations. *IEEE Transactions on Neural Networks and Learning Systems*, 31(10):4267–4278, 2020. doi: 10.1109/TNNLS.2019.2953622.

Alexander G. Ororbia. Brain-inspired machine intelligence: A survey of neurobiologically-plausible credit assignment, 2023. URL https://arxiv.org/abs/2312.09257.

Alexander G. Ororbia and Ankur Mali. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014651. URL https://doi.org/10.1609/aaai.v33i01.33014651.

Alexander G Ororbia, Ankur Mali, Daniel Kifer, and C Lee Giles. Backpropagation-free deep learning with recursive local representation alignment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 9327–9335, 2023.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26670–26698. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/orvieto23a.html.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. Pmlr, 2013.

T Konstantin Rusch and Siddhartha Mishra. Unicornn: A recurrent model for learning very long time dependencies. In *International Conference on Machine Learning*, pp. 9168–9178. PMLR, 2021.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Julian D Schiller, Malte Heinrich, Victor Lopez, and Matthias Müller. Tuning the burn-in phase in training recurrent neural networks improves their performance. In *International Conference on Learning Representations*, volume 2026, pp. 105660–105684, 2026.

Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992. doi: 10.1162/neco.1992.4.2.234.

Joannes Schoukens, Johan Suykens, and Lennart Ljung. Wiener-hammerstein benchmark. In *15th IFAC Symposium on System Identification (SYSID 2009), July 6-8, 2009, St. Malo, France*, July 2009. ISBN 978-3-902661-47-0. Finds and Results from the Swedish Cyprus Expedition: A Gender Perspective at the Medelhavsmuseet ; Conference date: 21-09-2009 Through 25-09-2009.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJQDjk-0b.

Matteo Tiezzi, Michele Casoni, Alessandro Betti, Marco Gori, and Stefano Melacci. State-space modeling in long sequence processing: a survey on recurrence in the transformer era. *Neural Networks*, pp. 108039, 2025a.

Matteo Tiezzi, Michele Casoni, Alessandro Betti, Tommaso Guidi, Marco Gori, and Stefano Melacci. Back to recurrent processing at the crossroad of transformers and state-space models. *Nature Machine Intelligence*, 7(5):678–688, 2025b.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. Rnns are not transformers (yet): The key bottleneck on in-context retrieval, December 2024.

Torbjörn Wigren and Johan Schoukens. Three free data sets for development and benchmarking in nonlinear system identification. In *2013 European Control Conference (ECC)*, pp. 2933–2938, 2013. doi: 10.23919/ECC.2013.6669201.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Russ R Salakhutdinov. On multiplicative integration with recurrent neural networks. *Advances in neural information processing systems*, 29, 2016.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkFqf0lAZ.

A Extended Related Work

Sophisticated recurrent networks like LSTMs and GRUs rely on gating mechanisms for stability and long-range dependency. Gating can improve memory retention, although saturated gates and deep recurrent compositions can still impede optimization (Li et al., 2018a). Vanilla RNNs carry information in their hidden states, but standard evaluation is sequential and learning long-lived memories can be difficult. Li et al. (2021) provide a theoretical study of instability in RNNs and the curse of memory. Many attempts have been made to enhance RNNs, as surveyed by Tiezzi et al. (2025b); Das et al. (2023), including tensor Mali et al. (2023) or multiplicative methods Mali et al. (2021); Wu et al. (2016) , parallelization (Danieli et al., 2026), memory augmentation (Yogatama et al., 2018; Malekmohamadi Faradonbe et al., 2020), and state-space models (Gu et al., 2022; Smith et al., 2023). Efforts to stabilize RNNs over long ranges include Neil et al. (2017); Chang et al. (2019); Rusch & Mishra (2021); Orvieto et al. (2023).

BPTT is the primary method for training recurrent networks, but it has several well-documented limitations Ororbia et al. (2020). The most established are vanishing and exploding gradients: as the temporal span of a dependency grows, repeated multiplication of Jacobians along the unrolled computational graph causes gradient magnitudes to decay or blow up exponentially, creating a fundamental trade-off between efficient gradient-based learning and retaining information over long horizons (Bengio et al., 1994; Pascanu et al., 2013). A second, related limitation is memory cost: exact BPTT stores activations from the unrolled sequence; checkpointing and recomputation can trade computation for storage, so long sequences force a choice between storing all past activations at high memory cost or truncating the backward pass into shorter windows, which discards long-term credit assignment and can bias the resulting gradient estimate (Tallec & Ollivier, 2018). Third, BPTT imposes forward, update, and backward locking: no module or timestep can be updated until the complete forward pass has executed and the corresponding error has been backpropagated

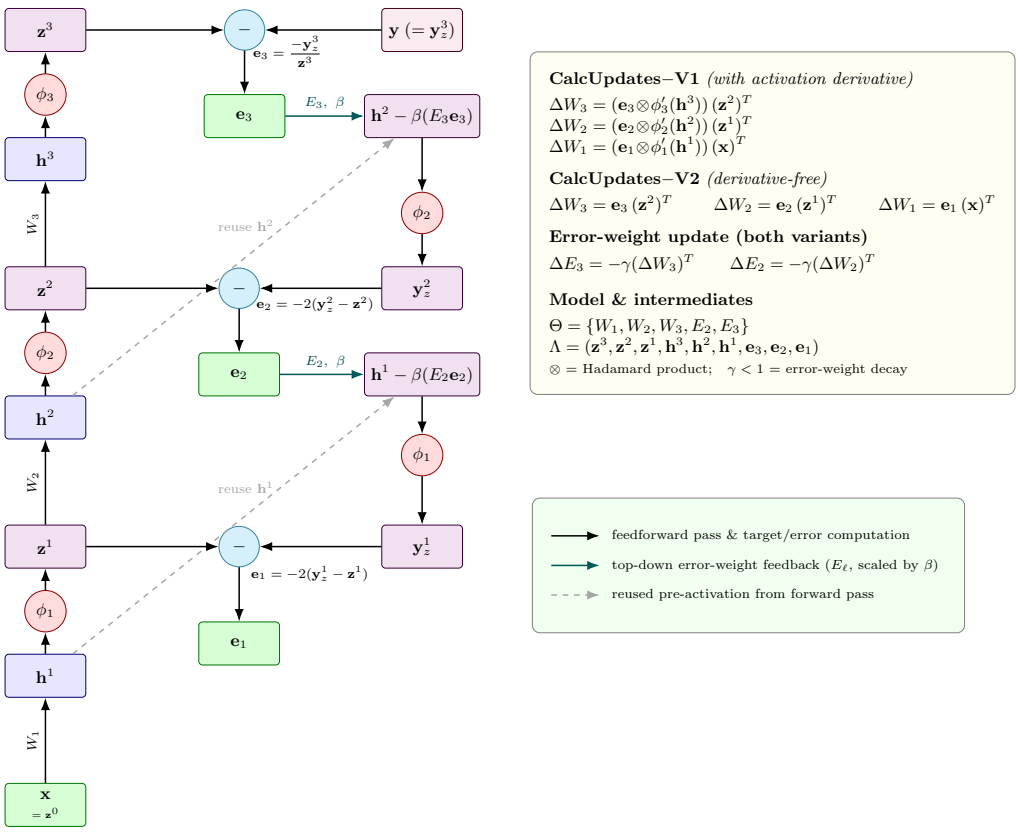


Figure 2: A computational graph representing the LRA-E algorithm of Ororbia & Mali (2019).

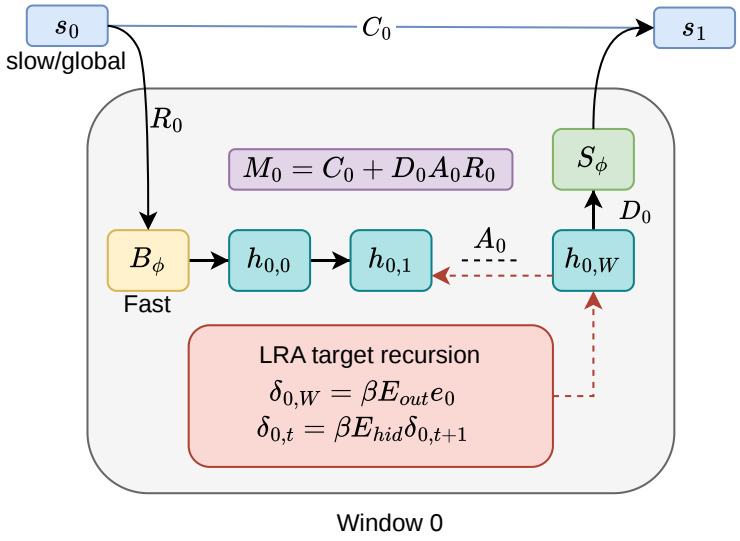


Figure 3: One individual block of HiCA-RNN from Figure 1.

through every subsequent step, which motivates research into decoupled updates (Jaderberg et al., 2017). Finally, BPTT is widely regarded as biologically implausible on at least three grounds: it requires symmetric forward and backward weights, the weight-transport problem (Lillicrap et al., 2016); it requires error signals to propagate backward through the network without interfering with forward neural activity (Bartunov et al., 2018); and it requires exact recall of past activations across the full sequence for temporal credit assignment, a capacity with no known biological analogue (Lillicrap & Santoro, 2019). Local training algorithms—including LRA (Ororbia & Mali, 2019) (Figure 2), target propagation (Manchev & Spratling, 2020), feedback alignment (Nøkland, 2016), and equilibrium propagation (Scellier & Bengio, 2017)—respond to these limitations by replacing the single global loss and its long gradient chain with local, layer- or unit-specific targets or losses. These methods differ in their feedback, locality, and scheduling assumptions. Such properties must be established for each algorithm; adding detached targets to full-graph BPTT does not itself remove locking or exact task-gradient computation.

One way to capture long-term dependencies in RNNs is to process information at multiple temporal scales. Mozer (1991); El Hihi & Bengio (1995); Schmidhuber (1992) were among the earliest to show that networks with components operating at distinct time constants can capture global sequence structure better than a single-rate network. Koutnik et al. (2014); Chung et al. (2015) extended this line with fixed or learned update rates for sub-modules. Ba et al. (2016) introduced fast weights that iteratively update through associative memory before the standard slow weights update by SGD. Fast–Slow RNNs place fast-operating RNN cells in a lower hierarchical layer that interacts with one slow recurrent state in a higher layer (Mujika et al., 2017). The multiscale update mechanism of HiCA-RNN, which combines fast recurrent updates with propagation of information over long interval, is similar to that of Clockwork RNN (Koutnik et al., 2014). This history motivates separating architectural time scales from learning-signal construction. HiCA retains full-graph BPTT and adds window-wise representation targets inspired by recurrent LRA (Ororbia et al., 2023). It does not establish that all multiscale models are unstable, or that this combination resolves every limitation of BPTT.

Broader recurrent modeling context. Recent recurrent and state-space modeling studies motivate efficient sequence processing and stable long-range learning (Wen et al., 2024; Ghojogh & Ghodsi, 2026; Tiezzi et al., 2025a). Non-saturating recurrent designs (Chandar et al., 2019), independently recurrent architectures (Li et al., 2018b), and surveys of brain-inspired learning and target propagation (Ororbia, 2023; Duan & Príncipe, 2022) provide complementary perspectives. Fast-weight programming (Irie & Gershman, 2026) changes weights during sequence processing;

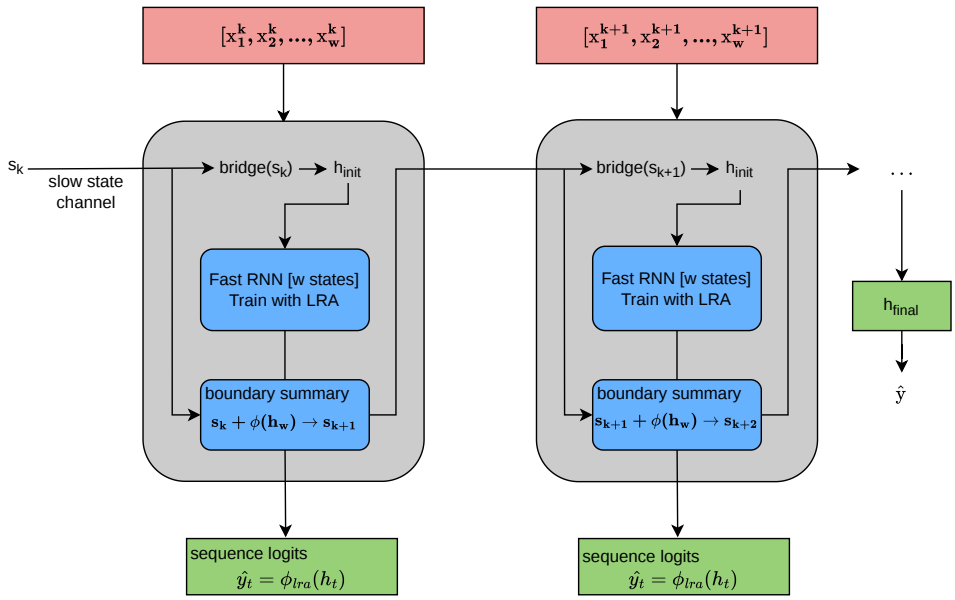


Figure 4: The architecture of the proposed fast and slow RNN model.

the fast state in HiCA instead uses shared recurrent parameters held fixed during each forward pass. These are distinct meanings of "fast."

Online and forward-mode credit assignment. A separate line of work avoids storing the trajectory at all by propagating sensitivities forward in time. Real-time recurrent learning (RTRL) computes the exact gradient online but maintains a $d_h \times |\theta|$ influence matrix, giving $O(d_h^3)$ memory and $O(d_h^4)$ time per step for a dense cell (Williams & Zipser, 1989). Approximations trade bias or variance for tractability: UORO keeps an unbiased rank-one stochastic estimate (Tallec & Ollivier, 2018), KF-RTRL and its optimal Kronecker-sum refinement use Kronecker-factored approximations (Mujika et al., 2018; Benzing et al., 2019), SnAp sparsifies the influence matrix (Menick et al., 2021), and e-prop reduces it to per-synapse eligibility traces combined with a broadcast learning signal (Bellec et al., 2020). HiCA is a reverse-mode method: it stores the fast trajectory, its cost is $O(T)$ in memory (Section 4.4), and its targets are constructed backward within a window. It is therefore not an online learner, and the forward-mode family is the natural comparison for any future claim about removing backward locking. Sparse attentive backtracking (Ke et al., 2018) takes a third route, learning a sparse attention mechanism that routes the task gradient directly to a few past hidden states; in the language of Section 4, it selects transport paths, whereas HiCA adds credit sources. Dilated and skip-connected recurrences (Chang et al., 2017) shorten paths architecturally without changing where credit enters.

Target propagation and reservoir computing. Difference target propagation (Lee et al., 2015) constructs layer-wise targets with a learned approximate inverse and corrects each target by the inverse's reconstruction error; its recurrent extension is target propagation through time (Manchev & Spratling, 2020). HiCA's targets use fixed and learned feedback matrices rather than learned inverses, and its intermediate residual compares a window's endpoint with the uncorrected first state of the next window, which is a consistency constraint rather than a difference-corrected label target. Reservoir computing, including echo state networks and liquid state machines, fixes a random recurrent map and trains only a readout (Jaeger, 2001; Jaeger & Haas, 2004; Maass et al., 2002; Lukoševičius & Jaeger, 2009). The fixed orthogonal E_h in HiCA plays a reservoir-like role on the feedback path, in the spirit of feedback alignment (Lillicrap et al., 2016; Nøkland, 2016), but the forward recurrence itself is trained; the echo-state property that reservoirs rely on for stability is not assumed here and would have to be verified through the transport bounds of Proposition 2.

B MODEL AND TRAINING PROCEDURE

B.1 CONNECTED RECURRENCE AT TWO TEMPORAL SCALES

Partition an input sequence of length T into $K = \lceil T/W \rceil$ windows, indexed by $k = 0, \ldots, K-1$. Set $W_k = \min\{W, T - kW\}$ and $x_{k,t} = x_{kW+t}$ for $1 \le t \le W_k$. The shared fast state $h_{k,t} \in \mathbb{R}^{d_h}$ and slow state $s_k \in \mathbb{R}^{d_s}$ evolve as

$$h_{k,0} = B_\phi(s_k) = \tanh(W_b s_k + b_b), \tag{20}$$

$$h_{k,t} = F_\theta(h_{k,t-1}, x_{k,t}) = \tanh(\mathcal{N}_f(W_x x_{k,t} + W_h h_{k,t-1} + b)), \tag{21}$$

$$s_{k+1} = S_\psi(s_k, h_{k,W_k}), \tag{22}$$

where $\mathcal{N}_f$ is LayerNorm when enabled and the identity otherwise. The parameter groups θ, ϕ, ψ are disjoint; θ includes fast LayerNorm parameters when present. The default slow transition uses

$$z_k = [h_{k,W_k}; s_k], \qquad g_k = \sigma(W_g z_k + b_g), \tag{23}$$

$$c_k = \tanh(W_c z_k + b_c), \qquad s_{k+1} = \mathrm{LN}_s(g_k \odot s_k + (1 - g_k) \odot c_k). \tag{24}$$

Slow-state LayerNorm is always applied, including when fast normalization is disabled. The fast cell is ungated; gating occurs at window boundaries. Other implemented boundary maps are specified in Appendix H.

All reported runs use `hybrid` mode. Within a forward call, the path $h_{k,W_k} \mapsto s_{k+1} \mapsto h_{k+1,0}$ remains differentiable: initializing a new fast window does not detach the preceding one. Independent examples use $s_0 = 0$. When a state is carried from a previous call, we set $s_0 = \mathrm{sg}(s_{\text{previous}})$, so differentiation stops at the call boundary. The operator sg preserves the forward value and has zero automatic-differentiation derivative.

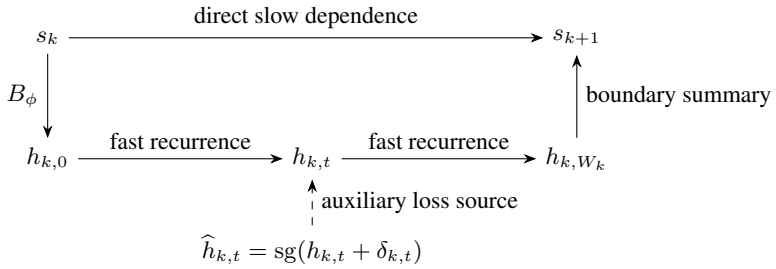


Figure 5: The connected fast–slow computation for one window. Solid arrows denote forward dependencies through which reverse-mode differentiation propagates. The dashed arrow denotes the direct auxiliary-loss derivative at a fast state; the target itself is detached. The next window starts from $B_\phi(s_{k+1})$, retaining cross-window gradient paths.

Algorithm 1: Forward pass in hybrid mode

1. Input sequence $x_{1:T}$ and window size W; initialize $s_0 = 0$ or detach a carried state from the preceding forward call.
2. For $k = 0, \ldots, K-1$, set $W_k = \min(W, T - kW)$ and $h_{k,0} = B_\phi(s_k)$.
3. Compute $h_{k,t} = F_\theta(h_{k,t-1}, x_{kW+t})$ for $t = 1, \ldots, W_k$ and dense readouts $Q_\zeta(h_{k,t})$ when required.
4. Compute $s_{k+1} = S_\psi(s_k, h_{k,W_k})$, including slow LayerNorm. Retain both bridge and boundary dependencies for differentiation.
5. Return $O_\omega(h_{K-1,W_{K-1}}, s_K)$ for final-output tasks, dense outputs for sequence tasks, and the state trajectories required for auxiliary targets.

B.2 Task readouts and losses

Let q be the output dimension and define two separate affine heads,

$$O_\omega(h, s) = W_o[h; s] + b_o, \qquad Q_\zeta(h) = Vh + b_q. \tag{25}$$

Final-output tasks use $\widehat{y} = O_\omega(h_{K-1,W_{K-1}}, s_K)$. Their task loss is cross-entropy for sequence-level classification and mean squared error (MSE) for Adding or vector forecasting. Dense-output tasks use $\widehat{y}_{k,t} = Q_\zeta(h_{k,t})$. Copy uses token-wise cross-entropy averaged over the full sequence. System identification uses MSE averaged over output channels and the time positions retained by the burn-in mask. Thus a dense task's loss enters at fast states directly and can propagate to earlier slow states through the bridge.

For a batch of B equally long examples, let $\mathcal{I}$ be the retained time indices. Dense regression uses

$$\mathcal{L}_{\text{task}} = \frac{1}{B|\mathcal{I}|q} \sum_{i=1}^{B} \sum_{\tau \in \mathcal{I}} \|\widehat{y}_{i,\tau} - y_{i,\tau}\|_2^2. \tag{26}$$

Chunk burn-in removes the first m positions of the entire call. Window burn-in removes the first m positions of each window; these are distinct protocols. Burn-in masks the task loss only and does not detach states or mask the auxiliary loss. A mask that removes every position is rejected.

B.3 Detached representation targets

Let $\mathcal{K}_a = \{k : W_k \geq 2\}$; the implementation skips auxiliary targets in one-token windows. For each eligible window, define a terminal target in output space,

$$o_k^\star = \begin{cases} Q_\zeta(h_{k+1,1}), & k < K-1, \\ y^\star, & k = K-1. \end{cases} \qquad e_k = \operatorname{sg}(o_k^\star - Q_\zeta(h_{k,W_k}))\,. \tag{27}$$

For final-output classification, $y^\star = \text{onehot}(y)$. For Copy, $y^\star = \text{onehot}(y_T)$. For dense regression, $y^\star = y_T$; for Adding or forecasting, $y^\star = y$. The output dimension of each head matches its task. Only window $K-1$ can receive the terminal supervised target through this rule, and only if it is eligible. If the last window has one token, it is skipped and no window receives that supervised LRA target. Dense task supervision still acts at all retained positions through equation 26 or token cross-entropy.

An intermediate residual compares output-head values at the current endpoint and the next window's first fast state. It uses the latter's uncorrected state, so the target rule does not carry a label-corrected target backward across windows. For affine Q_ζ, the bias cancels from these intermediate residuals.

With feedback matrices $E_o \in \mathbb{R}^{d_h \times q}$ and $E_h \in \mathbb{R}^{d_h \times d_h}$, construct

$$\delta_{k,W_k} = \beta E_o e_k, \tag{28}$$

$$\delta_{k,t} = \beta E_h \delta_{k,t+1}, \quad t = W_k - 1, \ldots, 1, \tag{29}$$

$$\widehat{h}_{k,t} = \operatorname{sg}(h_{k,t} + \delta_{k,t}). \tag{30}$$

All target construction runs without automatic differentiation. E_h is initialized orthogonally and receives no gradient in either optimization phase, so it remains fixed. E_o is adapted separately as described below. When feedback matrices are disabled, the target routine uses $V^\top$ in place of E_o and I in place of E_h. This is a distinct configuration; it has no trainable E_o and omits the feedback update below.

For equal-length batches, let $N_a = Bd_h \sum_{k \in \mathcal{K}_a} W_k$. The alignment loss is

$$\mathcal{L}_a = \frac{1}{N_a} \sum_{i=1}^{B} \sum_{k \in \mathcal{K}_a} \sum_{t=1}^{W_k} \|h_{i,k,t} - \widehat{h}_{i,k,t}\|_2^2, \qquad N_a > 0. \tag{31}$$

Set $\mathcal{L}_a = 0$ if $N_a = 0$. For variable lengths, the implementation counts only positions passing its auxiliary validity mask; its additional padding behavior is discussed in Appendix H. The analysis in Section 4 and Appendix C concerns fixed-length sequences, including a final partial window shared by the batch.

Targets are held fixed during differentiation. Hence the direct auxiliary derivative at an eligible fast-state node is

$$r_{i,k,t} = -\frac{2\lambda}{N_a}\delta_{i,k,t}; \qquad r_{i,k,t} = 0 \text{ at excluded nodes.} \tag{32}$$

The derivative enters at the fast state. Its effect on earlier states and shared parameters still follows the connected recurrent graph. Here locality describes where the target is introduced, not the extent of the parameter gradient.

B.4 Two optimization phases

The forward parameters are updated using

$$\mathcal{L}_{\text{main}} = \mathcal{L}_{\text{task}} + \lambda\mathcal{L}_a + \mu\mathcal{R}_{\text{proxy}}. \tag{33}$$

The proxy regularizer, including its detached activation factor, is given in Section D.2. A single Adam optimizer has separate learning rates for the fast cell and for the bridge, slow channel, and readouts. The implementation applies gradient clipping before this update and uses Adam weight decay; these operations are additional to the loss derivative analyzed below.

After the forward-parameter update, let $\bar{h}_i = \text{sg}(h_{i,K-1,W_{K-1}})$ be each example's terminal fast state retained from the preceding forward pass, and let $Q_{\zeta^+}(h) = V^+h + b_q^+$ be the post-update auxiliary projection. The feedback phase uses

$$\bar{e}_i = \text{sg}\big(y_i^\star - Q_{\zeta^+}(\bar{h}_i)\big)\,, \tag{34}$$

$$\mathcal{L}_{\text{fb}} = \frac{1}{Bd_h}\sum_i \big\|E_o\bar{e}_i - \text{sg}\big((V^+)^\top\bar{e}_i\big)\big\|_2^2\,. \tag{35}$$

A separate Adam optimizer updates E_o. The retained state, terminal residual, and post-update projection are constants in this phase. This squared-error objective matches both the direction and magnitude of the feedback response on the observed residuals; it does not update E_h.

Q_ζ learns through the dense task loss on Copy and system identification. On final-output tasks, the task trains O_ω, while Q_ζ receives no task gradient and remains at initialization under the specified trainer. Feedback matching on those tasks is therefore defined relative to an untrained auxiliary projection, not the derivative of the trained final-output head. The endpoint-alignment result in Section 4 concerns a surrogate formed with Q_ζ and does not imply alignment with the final-output task gradient. Conversely, O_ω is unused by dense task losses. Exact optimizer ordering and gradient-buffer behavior are recorded in Appendix H.

C Complete theoretical analysis and proofs

For readability, state derivatives below are written for one example; batch gradients sum these contributions with the loss normalizations above. All Jacobians are evaluated on the current forward trajectory. Parameters remain fixed during that trajectory. For clarity, first omit the regularizer; its derivative can be added separately. Define

$$J_{k,t} = \frac{\partial h_{k,t}}{\partial h_{k,t-1}}, \qquad R_k = \frac{\partial B_\phi(s_k)}{\partial s_k}, \tag{36}$$

$$C_k = \partial_s S_\psi(s_k, h_{k,W_k}), \qquad D_k = \partial_h S_\psi(s_k, h_{k,W_k}), \tag{37}$$

$$A_k = J_{k,W_k}\cdots J_{k,1}, \qquad M_k = C_k + D_k A_k R_k. \tag{38}$$

Let $P_{k,j:t} = J_{k,j}\cdots J_{k,t+1}$ for $j > t$ and $P_{k,t:t} = I$. The derivatives C_k, D_k include every gate, candidate-state, and normalization derivative.

C.1 Credit sources and backward transport

Let $u_{k,t}$ and v_k denote the direct partial derivatives of $\mathcal{L}_{\text{task}}$ with respect to fast and slow state nodes, treating the nodes as independent arguments of the task-loss expression. Define $f_{k,t} = u_{k,t} + r_{k,t}$. Let $b_{k,t}$ and a_k be the total adjoints at fast and slow nodes, respectively.

Algorithm 2: HiCA training in connected hybrid mode

1. Set $s_0 = 0$, or detach a state carried from the previous call.
2. For each window k, compute $h_{k,0} = B_\phi(s_k)$, the fast states $h_{k,1:W_k}$, and $s_{k+1} = S_\psi(s_k, h_{k,W_k})$. Retain every within-call state dependency.
3. Compute the final or dense task readout and its loss, applying task burn-in only where specified.
4. For each $k \in \mathcal{K}_a$, construct $e_k, \delta_{k,t}, \widehat{h}_{k,t}$ without differentiation using equation 27–equation 30.
5. Form $\mathcal{L}_a$, the proxy regularizer, and $\mathcal{L}_{\text{main}}$.
6. Clear the main optimizer's gradient buffers and backpropagate $\mathcal{L}_{\text{main}}$ through the connected state graph.
7. Clip all model gradient buffers to norm 1 and take the main Adam step. As implemented, this clipping call covers every model parameter, so the E_o buffer left by the preceding iteration's feedback step is included in the norm even though the main optimizer does not update E_o; Appendix H quantifies the effect.
8. In the learned-feedback configuration, clear feedback gradients. Form equation 35 using retained terminal fast states and the current auxiliary projection; backpropagate and take the feedback Adam step. Omit this step when feedback matrices are disabled.

Theorem 1 (restated). *For the full-graph rule with detached targets,*

$$a_K = v_K, \tag{39}$$

$$b_{k,t} = P_{k,W_k:t}^\top D_k^\top a_{k+1} + \sum_{j=t}^{W_k} P_{k,j:t}^\top f_{k,j}, \tag{40}$$

$$a_k = M_k^\top a_{k+1} + v_k + R_k^\top \sum_{j=1}^{W_k} P_{k,j:0}^\top f_{k,j}. \tag{41}$$

If $G_{k,t} = \partial_\theta F_\theta(h_{k,t-1}, x_{k,t})$ denotes the direct fast-parameter derivative with its inputs held fixed, then

$$\nabla_\theta(\mathcal{L}_{\text{task}} + \lambda \mathcal{L}_a) = \sum_{k,t} G_{k,t}^\top b_{k,t}. \tag{42}$$

Here θ occurs only in the fast cell, the task heads have no additional direct dependence on θ, and s_0 is independent of θ within the forward call. The derivative follows the stated stop-gradient convention: targets are held fixed during backpropagation.

Proof. At the final fast state, $b_{k,W_k} = D_k^\top a_{k+1} + f_{k,W_k}$. At earlier states the chain rule gives $b_{k,t} = J_{k,t+1}^\top b_{k,t+1} + f_{k,t}$. Unrolling yields equation 40. The slow state has a direct edge into S_ψ and another into the bridge, and $h_{k,0}$ carries no loss term, so

$$a_k = v_k + C_k^\top a_{k+1} + R_k^\top J_{k,1}^\top b_{k,1}.$$

Substitution gives equation 41, using $J_{k,1}^\top P_{k,W_k:1}^\top = A_k^\top$ and $J_{k,1}^\top P_{k,j:1}^\top = P_{k,j:0}^\top$. Summing the contribution of every occurrence of the shared fast parameters gives equation 42. □

The theorem makes the proposed mechanism explicit. A target at state (k,t) contributes $G_{k,t}^\top r_{k,t}$ to the shared fast-parameter gradient without first traversing the future fast Jacobians. The same target also influences earlier occurrences of the shared parameters through the remaining terms. Thus targets add intermediate sources of credit; they do not remove the original task-gradient paths. Neither a nonzero source nor its immediate parameter contribution is guaranteed: $E_o e_k$ can vanish, and $G_{k,t}^\top$ can annihilate the injected direction.

The boundary equation shows that injected credit can propagate between windows through $M_k^\top$. Architectural transport and target-based injection are distinct: setting $\lambda = 0$ removes the latter while leaving all forward transitions unchanged. This gives the central experimental control.

C.2 What the macro-Jacobian does and does not establish

The chain rule gives $\partial s_{k+1}/\partial s_k = M_k$ and

$$\frac{\partial s_K}{\partial s_0} = M_{K-1} \cdots M_0. \tag{43}$$

This identity is exact but does not show that BPTT has only K elementary steps. Each M_k includes $D_k A_k R_k$, containing W_k fast transitions. Expanding equation 43 yields an ordered sum of paths, including the direct slow path $C_{K-1} \cdots C_0$ and a path traversing every fast block. Path contributions can cancel: even $C_k = I$ gives $M_k = 0$ if $D_k A_k R_k = -I$.

For shared fast parameters, define $U_k = ds_k/d\theta$ and $H_{k,t} = \sum_{j=1}^{t} P_{k,t:j} G_{k,j}$. Then $U_0 = 0$ and

$$U_{k+1} = M_k U_k + D_k H_{k,W_k}, \qquad \frac{dh_{k,t}}{d\theta} = H_{k,t} + P_{k,t:0} R_k U_k. \tag{44}$$

For a source at (k, t), Eq. equation 44 separates its within-window parameter sensitivity $H_{k,t}$ from the sensitivity U_k inherited from earlier windows. Either contribution may vanish. These expressions also assume disjoint parameter groups and a parameter-independent s_0.

Proposition 2 (restated). *Let $\mathcal{V}_0$ be a nonzero subspace and $\mathcal{V}_{k+1} = M_k \mathcal{V}_k$. Suppose, for all $v \in \mathcal{V}_k$,*

$$m_k \|v\| \le \|M_k v\| \le L_k \|v\|, \qquad 0 < m_k \le L_k < \infty.$$

Then

$$\left(\prod_{k=0}^{K-1} m_k \right) \|v\| \le \|M_{K-1} \cdots M_0 v\| \le \left(\prod_{k=0}^{K-1} L_k \right) \|v\|, \quad v \in \mathcal{V}_0. \tag{45}$$

If $m_k \ge e^{-\varepsilon_k}$, $L_k \le e^{\varepsilon_k}$, and $\sum_k \varepsilon_k \le C$, the product gains lie between e^{-C} and e^{C}.

Proof. Apply each assumed bound to $v_{k+1} = M_k v_k$ and multiply the inequalities. For the last statement, exponentiate the sum of the log-gain bounds. □

Uniform bounds m, L yield m^K, L^K, which can still decay or grow exponentially. Their values depend on W, the learned parameters, normalization, and trajectory. Merely rewriting a product at window resolution supplies no conditioning advantage: in a scalar fast–slow recurrence with $C_k = 0$, $D_k = R_k = 1$, and $J_{k,t} = a$, the macro gain is a^{W_k} and the total gain remains a^T. The example is illustrative; the LayerNorm slow cell need not realize $C_k = 0$. A useful sufficient condition for preventing cancellation on $\mathcal{V}_k$ is

$$\inf_{\substack{v \in \mathcal{V}_k \\ \|v\|=1}} \|C_k v\| \ge c_k, \qquad \|(D_k A_k R_k)|_{\mathcal{V}_k}\| \le \epsilon_k < c_k,$$

which gives an admissible lower bound $m_k = c_k - \epsilon_k$ by the reverse triangle inequality. These conditions require measurement or separate analysis; the architecture alone does not establish them. The proposition bounds forward sensitivities on $\mathcal{V}_0$; it does not supply a lower bound for arbitrary backward adjoint directions, which involve the transposed macro-Jacobians $M_k^\top$ acting on subspaces that are not, in general, the forward-propagated $\mathcal{V}_k$.

Input sensitivity and the actual readout. State transport alone does not characterize input credit. Let ξ perturb the inputs of window r with s_r held fixed, and write $Q_r^x = \partial h_{r,W_r}/\partial \xi$. For the actual final head in equation 25, let O_h and O_s be its partial derivatives at the terminal fast and slow states. Set $\ell = K - 1$ and

$$Z_\ell = O_h A_\ell R_\ell + O_s M_\ell.$$

For $r < \ell$, the input-to-output derivative is

$$\frac{\partial \widehat{y}}{\partial \xi} = Z_\ell M_{\ell-1} \cdots M_{r+1} D_r Q_r^x, \tag{46}$$

with an identity product for $r = \ell - 1$. For $r = \ell$, it is $(O_h + O_s D_\ell) Q_\ell^x$. These identities follow by differentiating the injection into s_{r+1}, transport to s_ℓ, and both terminal readout branches. A loss gradient additionally multiplies by the output-loss derivative. Input injection or readout can annihilate a direction even if every intermediate state transition preserves it. Accordingly, the restricted-gain proposition is a conditional statement about state transport, not a lower bound on arbitrary task gradients or a prediction of an accuracy crossover.

C.3 Feedback magnitude and useful credit

Proposition 5 (Target propagation profile). *For $k \in \mathcal{K}_a$, in exact arithmetic and with fixed feedback matrices within a target-construction pass,*

$$\delta_{k,t} = \beta^{W_k-t+1} E_h^{W_k-t} E_o e_k. \tag{47}$$

Consequently,

$$\|\delta_{k,t}\| \le |\beta| \big(|\beta| \|E_h\|_2\big)^{W_k-t} \|E_o e_k\|.$$

If E_h is orthogonal, this simplifies exactly to

$$\|\delta_{k,t}\| = |\beta|^{W_k-t+1} \|E_o e_k\|. \tag{48}$$

Proof. Repeated substitution in equation 30 gives equation 47. Use submultiplicativity for the upper bound and norm preservation of each orthogonal factor for equation 48. □

Orthogonal feedback removes amplification due to powers of E_h, but it does not remove geometric attenuation from β. When $0 < |\beta| < 1$, targets decay toward the beginning of each window; when $|\beta| > 1$, they grow. Their maximum propagation distance is bounded by $W - 1$, rather than the full sequence horizon. Furthermore, the direct source in equation 32 includes $1/N_a$: a bounded target magnitude does not imply a sequence-length-independent source under the reported normalization. Comparisons across lengths must specify loss reduction and coefficient scaling. The number of sources also changes with T, so this observation alone does not prove decay of the summed parameter gradient.

For example, with $\beta = 0.1$ and $W_k = 10$, the endpoint multiplier is 10^{-1} whereas the earliest target multiplier is 10^{-10}. These values illustrate the recursion; they are not a statement of the tuned settings used in every experiment.

C.4 What feedback matching guarantees locally

The feedback phase in equation 35 matches $E_o e$ to $V^\top e$ on the terminal residuals it observes. The following statement identifies the significance of that match without equating it with descent of the full task objective.

Proposition 3 (restated). *Hold a target $o^\star$, a projection $Q(h) = Vh + b_q$, and feedback matrix E_o fixed. Let $e = o^\star - Q(h)$ and $\Phi(h) = \frac{1}{2}\|Q(h) - o^\star\|^2$. Suppose*

$$p = V^\top e \neq 0, \qquad \|(E_o - V^\top)e\| \le \epsilon \|p\|, \qquad 0 \le \epsilon < 1.$$

Then $d = E_o e$ satisfies

$$\langle \nabla_h \Phi(h), d \rangle \le -(1-\epsilon)\|p\|^2 < 0. \tag{49}$$

For $\beta > 0$, the endpoint target $h + \beta d$ strictly decreases Φ whenever

$$0 < \beta < \frac{2(1-\epsilon)}{\|V\|_2^2 (1+\epsilon)^2}. \tag{50}$$

Proof. Write $d = p + z$ with $\|z\| \le \epsilon\|p\|$. Since $\nabla_h \Phi = -p$, Cauchy–Schwarz gives equation 49. The affine head yields the exact expansion

$$\Phi(h + \beta d) - \Phi(h) = -\beta \langle p, d \rangle + \frac{\beta^2}{2} \|Vd\|^2.$$

Using $\langle p, d \rangle \ge (1-\epsilon)\|p\|^2$ and $\|d\| \le (1+\epsilon)\|p\|$ gives the stated sufficient step-size bound. □

The proposition concerns a hidden-space endpoint displacement for a frozen projected squared-error surrogate. It does not establish that updating shared parameters realizes this displacement, nor does it establish descent of cross-entropy or of the separate final readout. The trainer minimizes an average feedback error over terminal residuals, which does not enforce the stated relative-error condition for every residual, particularly the intermediate consistency residuals. At an earlier state (k, t), the negative endpoint-surrogate gradient is $P_{k,W_k:t}^\top V^\top e_k$, while the proposed displacement is $\beta^{W_k-t+1} E_h^{W_k-t} E_o e_k$. Orthogonality of E_h controls magnitudes but does not align these two directions. The distinction motivates measuring feedback error and task/auxiliary gradient alignment separately.

Proposition 4 (restated). *Consider an unclipped gradient step without the transition regularizer or Adam preconditioning. At the current iterate ϑ, let $g = \nabla\mathcal{L}_{\text{task}}(\vartheta)$ and let $a = \nabla_\vartheta\mathcal{L}_a(\vartheta)$ be the auxiliary gradient with current targets held fixed. Assume $\mathcal{L}_{\text{task}}$ has L-Lipschitz gradient along the update segment. For $d = g + \lambda a$ and $\vartheta^+ = \vartheta - \eta d$,*

$$\mathcal{L}_{\text{task}}(\vartheta^+) \le \mathcal{L}_{\text{task}}(\vartheta) - \eta\big(\|g\|^2 + \lambda\langle g, a\rangle\big) + \frac{L\eta^2}{2}\|d\|^2. \tag{51}$$

If $A = \|g\|^2 + \lambda\langle g, a\rangle > 0$ and $L > 0$, then $0 < \eta < 2A/(L\|d\|^2)$ ensures strict task descent.

Proof. Apply the smoothness inequality to $\vartheta^+ - \vartheta = -\eta d$ and expand $\langle g, d\rangle$. The step-size condition makes the upper-bound change strictly negative. □

This result is conditional, not a claim that LRA targets align automatically. For $\lambda > 0$, the choice $a = -2g/\lambda$ makes the combined direction $d = -g$, giving task ascent to first order when $g \neq 0$. This demonstrates why an auxiliary target does not guarantee task descent merely by adding a nonzero signal. A regularizer adds its gradient to d and needs its own alignment analysis. Adam with momentum does not in general step along $g+\lambda a$; the generic smoothness bound instead uses its actual direction d_{Adam} and inner product $\langle g, d_{\text{Adam}}\rangle$. Thus equation 51 is a mechanism diagnostic, not an Adam convergence theorem.

D Normalization, regularization, and computational cost

D.1 Jacobians with LayerNorm

For $v \in \mathbb{R}^d$, let $P_0 = I - \mathbf{1}\mathbf{1}^\top/d$, $c = P_0 v$, and $\sigma = (\|c\|^2/d + \epsilon_{\text{LN}})^{1/2}$, where $\epsilon_{\text{LN}} > 0$. For affine LayerNorm with scale γ_{LN} and bias b_{LN},

$$\text{LN}(v) = \gamma_{\text{LN}} \odot c/\sigma + b_{\text{LN}}, \qquad D\,\text{LN}(v) = \text{diag}(\gamma_{\text{LN}})\left(\frac{P_0}{\sigma} - \frac{cc^\top}{d\sigma^3}\right). \tag{52}$$

In particular, $D\,\text{LN}(v)\mathbf{1} = 0$. When all entries of γ_{LN} are nonzero, the rank is $d-1$ for $\epsilon_{\text{LN}} > 0$: the centered radial direction is not an additional null direction. With $\epsilon_{\text{LN}} = 0$ and a nonzero centered input, it is also a null direction.

For the fast cell, the temporal Jacobian is

$$J_{k,t} = \text{diag}(1 - h_{k,t}^{\odot 2})D\mathcal{N}_f(v_{k,t})W_h, \qquad v_{k,t} = W_x x_{k,t} + W_h h_{k,t-1} + b. \tag{53}$$

Here $D\mathcal{N}_f = I$ when fast normalization is disabled. The regularizer below uses an unnormalized proxy; the Jacobians in the credit analysis include the normalization derivative.

For completeness, let $u = g \odot s + (1 - g) \odot c$ be the pre-normalization slow update. Its partial derivatives are

$$\begin{aligned}\partial_s u &= \text{diag}(g) + \text{diag}(s - c)\partial_s g + \text{diag}(1 - g)\partial_s c,\\ \partial_h u &= \text{diag}(s - c)\partial_h g + \text{diag}(1 - g)\partial_h c.\end{aligned}$$

For $a \in \{s, h\}$, $\partial_a g = \text{diag}(g\odot(1-g))W_{g,a}$ and $\partial_a c = \text{diag}(1-c^{\odot 2})W_{c,a}$, using the corresponding input block of each weight matrix. Thus $C_k = D\,\text{LN}_s(u)\partial_s u \in \mathbb{R}^{d_s\times d_s}$ and $D_k = D\,\text{LN}_s(u)\partial_h u \in \mathbb{R}^{d_s\times d_h}$. An open gate alone does not imply $C_k \approx I$, since the gate, candidate, and LayerNorm derivatives all enter C_k.

D.2 Implemented transition penalty

Let $N_r = \sum_k (W_k - 1)$ and $\epsilon_r = 10^{-8}$. For $N_r > 0$, the trainer computes

$$\bar{d}_{i,k,t} = 1 - \text{sg}(h_{i,k,t})^{\odot 2}, \tag{54}$$

$$n_{i,k,t} = \left(\|\,\text{diag}(\bar{d}_{i,k,t})W_h\|_F^2 + \epsilon_r\right)^{1/2}, \tag{55}$$

$$\mathcal{R}_{\text{proxy}} = \frac{\gamma_r}{BN_r}\sum_{i=1}^{B}\sum_{k}\sum_{t=2}^{W_k}(n_{i,k,t} - 1)^2. \tag{56}$$

Set the penalty to zero when $N_r = 0$. Its coefficient in the main objective is $\mu\gamma_r$. The activation factors are detached, so its direct gradient acts only on W_h. The first recurrent step of each window is excluded. With row index a and column index b, the direct derivative is

$$\frac{\partial \mathcal{R}_{\text{proxy}}}{\partial (W_h)_{ab}} = \frac{2\gamma_r}{BN_r} \sum_{i=1}^{B} \sum_{k} \sum_{t=2}^{W_k} \frac{n_{i,k,t} - 1}{n_{i,k,t}} \bar{d}^2_{i,k,t,a} (W_h)_{ab}. \tag{57}$$

This is the derivative of the implemented proxy with its activation factors fixed, not the derivative of the norm of the full temporal Jacobian when fast LayerNorm is active. Frobenius norm does not control every singular value: a rank-one matrix can have unit Frobenius norm, while a $d \times d$ orthogonal matrix has Frobenius norm $\sqrt{d}$. We use the penalty as a regularizer, without assuming that it preserves temporal gradients.

D.3 Time and space complexity

We count leading multiply–adds per sequence, suppressing the batch factor B. Let d_{in}, d_h, d_s, and q denote the input, fast, slow, and output widths, with $K = \lceil T/W \rceil$. These counts refer to the modules specified in Appendix B; actual time and peak memory also depend on the execution path and backend.

Parameters. The fast cell contains $W_x \in \mathbb{R}^{d_h \times d_{\text{in}}}$, $W_h \in \mathbb{R}^{d_h \times d_h}$, and one fast bias. The bridge contains $W_b \in \mathbb{R}^{d_h \times d_s}$ and one bias. The slow channel contains $W_g, W_c \in \mathbb{R}^{d_s \times (d_h + d_s)}$, their two biases, and slow LayerNorm scale and bias. The heads contain $W_o \in \mathbb{R}^{q \times (d_h + d_s)}$ and $V \in \mathbb{R}^{q \times d_h}$, each with a bias. The feedback matrices are $E_o \in \mathbb{R}^{d_h \times q}$ and $E_h \in \mathbb{R}^{d_h \times d_h}$, without biases. Counting both heads and both feedback matrices, with fast LayerNorm enabled, gives

$$\begin{aligned} \#P_{\text{HiCA}} = {} & 2d_h^2 + d_h(d_{\text{in}} + d_s + 3q) + 2d_s(d_h + d_s) + d_s q \\ & + 4d_h + 4d_s + 2q. \end{aligned} \tag{58}$$

The last line counts the fast and bridge biases ($2d_h$), fast LayerNorm scale and bias ($2d_h$), slow biases and LayerNorm parameters ($4d_s$), and head biases ($2q$). Without fast LayerNorm, subtract $2d_h$. For $d_{\text{in}} = q = 1$ and $d_h = d_s = 8$, the displayed configuration has $128 + 96 + 256 + 8 + 66 = 554$ stored parameters. The d_h^2 entries of E_h are fixed and are excluded from the trainable count. Neither feedback matrix is needed for inference, so removing them reduces stored inference parameters by $d_h^2 + d_h q$; a task-specific inference implementation may also omit its unused head.

For comparison, a conventional single-layer LSTM with one q-dimensional readout has $4d_h(d_{\text{in}} + d_h) + 8d_h + d_h q + q$ parameters, or 361 when $d_{\text{in}} = q = 1$ and $d_h = 8$. This comparison counts one task head for the LSTM but both task heads and feedback matrices for the stated HiCA training module.

Forward computation. Each fast step costs $d_h(d_{\text{in}} + d_h)$ for its affine maps, plus normalization and elementwise work. Each window costs

$$d_h d_s + 2d_s(d_h + d_s) = 3d_h d_s + 2d_s^2 \tag{59}$$

for the bridge and two slow affine maps. The leading recurrent forward cost is therefore $Td_h(d_{\text{in}} + d_h) + K(3d_h d_s + 2d_s^2)$. Readout cost depends on what the routine evaluates. A final O_ω evaluation costs $(d_h + d_s)q$; evaluating Q_ζ at all fast steps adds $Td_h q$. If Q_ζ is computed only where the target rule requires it, that latter cost is reduced accordingly. The corresponding LSTM recurrent forward cost is $4Td_h(d_{\text{in}} + d_h)$, with the same task-specific prediction schedule used for comparison.

Target construction and backward computation. Each eligible window computes one $E_o e_k$ product at cost $d_h q$ and $W_k - 1$ dense E_h products at cost d_h^2 each. The leading target cost is $O(Td_h^2 + Kd_h q)$, with no differentiation through this construction. The feedback recursion has sequential depth $O(W)$ per window; it can run independently across windows once the forward states are available. Reverse-mode differentiation of an affine map requires products for both its input adjoint and weight gradient. Thus a useful leading-order estimate is about twice the differentiable forward affine cost for either architecture, with separate costs for normalization, the proxy, feedback matching, and optimizer operations.

The proxy has an avoidable implementation cost. The row norms of W_h do not depend on the example, yet the implementation recomputes them at every penalized step, costing $O(N_r d_h^2)$ per training step in the forward pass and again in the backward pass, in addition to the $O(BN_r d_h)$ elementwise work. Since

$$\| \operatorname{diag}(\bar{d}_{i,k,t}) W_h \|_F^2 = \sum_{a=1}^{d_h} \bar{d}_{i,k,t,a}^2 \sum_{b=1}^{d_h} (W_h)_{ab}^2,$$

the row norms can instead be computed once per training step and reused. This reduces the arithmetic for the proxy to $O(d_h^2 + BN_r d_h)$ without changing its value or direct derivative.

Leading per-token comparison. Suppose $d_h = d_s = d$, $d_{\text{in}}, q \ll d$, and ignore normalization, readouts, the proxy, feedback matching, and optimizer overhead. Counting backward affine work as twice forward affine work gives approximately $3d^2$ for the fast cell's forward and backward passes, d^2 for target construction, and $15d^2/W$ for the bridge and slow channel per token. The resulting $4d^2 + 15d^2/W$ should be compared with approximately $12d^2$ for an equal-width LSTM under the same simplified accounting. At $W = 10$ these are $5.5d^2$ and $12d^2$, a factor of about 2.2; the factor approaches 3 as W grows or as d_s/d_h shrinks, and the reported configurations typically use d_s between one quarter and one half of d_h. They are operation estimates, not measured training-time ratios; proxy cost and backend behavior can materially alter the comparison. At an equal parameter budget, the chosen HiCA fast width may also differ from the LSTM width.

Training memory. Hybrid BPTT retains fast-state dependencies over all T steps. Its recurrent activation storage is $O(B(Td_h + Kd_s))$; storing dense q-dimensional readouts adds $O(BTq)$ when they are materialized at every step. A single-layer LSTM uses $O(BTd_h)$ recurrent activation storage, with the same possible dense-readout term under an equal output schedule. The constants depend on saved normalization intermediates, auxiliary targets, readouts, the proxy, and backend workspaces; a count of a few state vectors per step does not determine peak allocated memory. The measured sequential CIFAR-10 peak-memory result in Table 5 is therefore an empirical comparison, and its precise cause cannot be assigned to a particular buffer or backend without memory profiling. Windowing does not reduce hybrid BPTT storage to $O(W + K)$.

Inference. A streaming implementation needs $d_h + d_s$ state values for HiCA and $2d_h$ for an LSTM, independent of T. The training-oriented forward routine may retain the trajectory even under no-grad; streaming inference must discard intermediate fast states when they are no longer needed.

E Datasets

We evaluate synthetic dependency tasks, pixel-sequence classification, and nonlinear system identification. Table 6 summarizes the core benchmarks; sequential CIFAR-10 is described in Section G.4.

E.1 Synthetic Long-Range Dependency Tasks

Three tasks from the classical RNN long-range dependency literature test a model's ability to propagate and selectively retrieve information over long sequences (Hochreiter & Schmidhuber, 1997). All three are synthetically generated with a fixed NumPy random seed per split (train: seed 42, test: seed 123), making generation reproducible; distinct seeds alone do not guarantee the absence of duplicate examples across finite splits.

Temporal Order: A length-T sequence of i.i.d. noise symbols drawn from a 4-symbol vocabulary contains two additional marker symbols (`A` or `B`), placed at random positions in the first and second quarter of the sequence respectively. The model must classify the *order* in which the two markers appear (`AA`, `AB`, `BA`, `BB`; 4-way classification). Inputs are one-hot over a 6-symbol vocabulary (4 noise symbols + 2 markers).

Adding Problem: A length-T sequence has two input channels: (i) i.i.d. values drawn uniformly from $[0, 1]$, and (ii) a binary indicator channel that is 1 at exactly two positions (one drawn from the first half of the sequence, one from the second half) and 0 elsewhere. The target is the sum of the two marked values (scalar regression, MSE loss).

Table 6: Summary of tasks and datasets. Synthetic tasks are author generated and remaining tasks use real data. T is sequence length; $^{\dagger}\, T \in \{50, 100, 150, 200\}$; $^{\ddagger}\, T \in \{100, 200, 500, 1000\}$.

Task	Data source	Type	T	Output	Metric
Temporal Order	synthetic	classification	†	4-way	accuracy
Adding Problem	synthetic	regression	†	scalar	MSE
Copy	synthetic	seq. classification	†	9-way/step	CE loss
Sequential MNIST	MNIST	classification	784	10-way	accuracy
Permuted MNIST	MNIST	classification	784	10-way	accuracy
	Silver-Box	regression	‡	scalar	MSE
Dynamical Systems	RLC Circuit	regression	‡	2-dim	MSE
	Wiener-Hammerstein	regression	‡	scalar	MSE

Copy Task: The input presents a length-K pattern drawn from an n-symbol vocabulary, followed by a blank run, a delimiter symbol, and another blank run, for a total length T; the target is the blank class everywhere except the final K steps, where the original pattern must be reproduced in order. This is a per-timestep classification task (cross-entropy at every step) that directly probes memory capacity across the blank gap, independent of any selection or computation. We use $K = 10$ pattern length and $n = 8$ symbols in the vocabulary.

All three tasks were evaluated at sequence lengths $T \in \{50, 100, 150, 200\}$ to trace performance as a function of the memory horizon.

E.2 Sequential and Permuted MNIST

MNIST digits are flattened into a length-784 sequence of single-pixel (scalar) inputs, and the model must classify the digit (10-way) only after observing all 784 steps (Deng, 2012; Landi et al., 2021). In the *permuted* variant, a single fixed random pixel permutation (seeded, generated once) is applied identically to every image in both splits, which removes the local 2D spatial structure a model might otherwise exploit and forces reliance on long-range sequential memory instead. Standard MNIST train/test splits are used ($60,000/10,000$ images); unlike the synthetic tasks above, sequence length is fixed at 784 by the nature of the task.

E.3 Dynamical Systems - System Identification

Following Schiller et al. (2026), we evaluate on three public nonlinear system identification (sysid) benchmarks: Silver-Box (Wigren & Schoukens, 2013), an electronic implementation of a forced Duffing oscillator; Wiener-Hammerstein (Schoukens et al., 2009), a static diode-resistor nonlinearity sandwiched between two linear dynamical blocks; and an RLC circuit dataset (Forgione & Piga, 2021) yielding two output channels (capacitor voltage and inductor current). The model receives an input excitation signal and must predict the corresponding output trajectory at every timestep—a sequence-to-sequence regression task evaluated with mean squared error (MSE). We use the same train/validation/test splits as Schiller et al. (2026): $20{,}000/10{,}000/10{,}000$ samples for Silver-Box and W-H, and $6{,}667/3{,}333/3{,}333$ for RLC.

F Training Protocol

All models are trained with Adam and gradient-norm clipping at 1.0. All HiCA runs use hybrid differentiation. Stateful/stateless in the system-identification table describes state carry across calls, not the choice of hybrid versus detached differentiation. Hyperparameters (hidden size, learning rate, weight decay, and any architecture-specific parameters) are tuned independently per task and sequence length using Optuna (Akiba et al., 2019) with a TPE sampler and median-stopping pruning, optimizing held-out validation loss. The selected configuration is then retrained and evaluated on the held-out test split. System-identification results (Table 4) are instead reported over the top-3 runs of the tuning study and should be read as descriptive. In our setup, each trial trains one model to a validation-loss checkpoint under one sampled hyperparameter configuration, reports that loss back to the Optuna study, and TPE uses the full history of prior trials to choose the next configuration.

Table 7: Hyperparameter configurations selected by Optuna HPO (30 trials, 50-epoch proxy) and used for 500-epoch final evaluation. H = fast hidden size, S = slow state size (HiCA-RNN only), W = window size (HiCA-RNN only), #P = total trainable parameters.

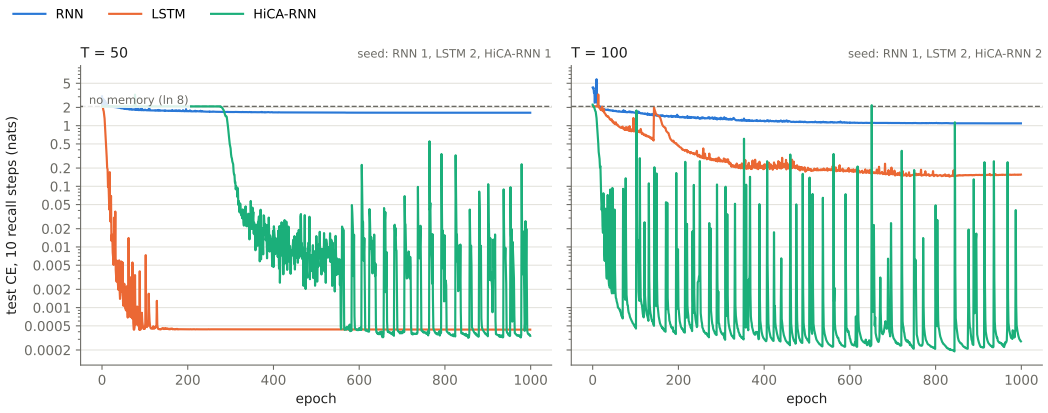
Task	Arch	T	H	S	W	#P
Adding	RNN	50	128	—	—	17,025
	RNN	100	64	—	—	4,417
	RNN	150	128	—	—	17,025
	RNN	200	64	—	—	4,417
	LSTM	50	512	—	—	1,057,281
	LSTM	100	64	—	—	17,473
	LSTM	150	128	—	—	67,713
	LSTM	200	128	—	—	67,713
	HiCA-RNN	50	64	32	5	17,122
	HiCA-RNN	100	64	64	10	29,570
	HiCA-RNN	150	128	32	10	48,418
	HiCA-RNN	200	128	32	10	48,418
Copy	RNN	50	512	—	—	272,905
	RNN	100	256	—	—	70,921
	RNN	150	256	—	—	70,921
	RNN	200	256	—	—	70,921
	LSTM	50	512	—	—	1,077,769
	LSTM	100	256	—	—	276,745
	LSTM	150	512	—	—	1,077,769
	LSTM	200	512	—	—	1,077,769
	HiCA-RNN	50	256	128	10	274,322
	HiCA-RNN	100	256	32	10	168,626
	HiCA-RNN	150	128	32	10	52,786
	HiCA-RNN	200	256	64	10	199,762
Temporal Order	RNN	50	128	—	—	17,924
	RNN	100	256	—	—	68,612
	RNN	150	128	—	—	17,924
	RNN	200	64	—	—	4,868
	LSTM	50	512	—	—	1,067,012
	LSTM	100	512	—	—	1,067,012
	LSTM	150	256	—	—	271,364
	LSTM	200	128	—	—	70,148
	HiCA-RNN	50	256	128	10	268,808
	HiCA-RNN	100	64	128	10	67,976
	HiCA-RNN	150	64	64	10	30,600
	HiCA-RNN	200	64	32	10	18,056

Median-stopping pruning terminates a trial early if its intermediate validation loss is worse than the median of other trials at the same point in training, so compute is not wasted completing runs that are unlikely to improve on what has already been found.

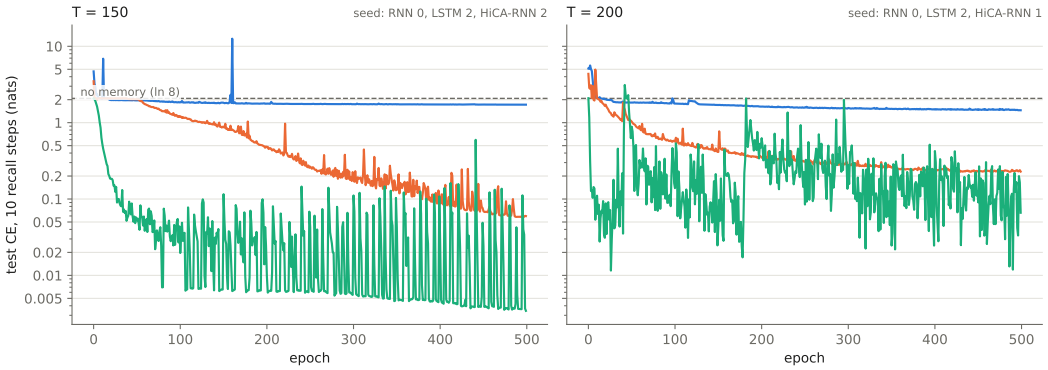
G Experimental details

G.1 Synthetic Tasks for long range dependency

Table 7 provides the configuration of every network and its number of parameters. Figure 6 shows Cross Entropy loss trajectory of RNN, LSTM, HiCA-RNN while training over 500 epochs for all sequence lengths. To ensure finite gradient optimization in lstm we set its $forget\ gate\ bias = 2$. This improves its performance over standard lstm version. We used pattern matching CE loss for these experiments meaning, all the blank symbols were ignored and only the symbols following



(a) $T \in \{50, 100\}$ (1000 epochs)



(b) $T \in \{150, 200\}$ (500 epochs)

Figure 6: Copy task: test cross-entropy on the 10 recall steps per epoch (log scale; dashed line: memoryless floor $\ln 8$). Each curve is that model's best of 3 seeds, ranked by final-epoch loss.

Table 8: HiCA-RNN tuned hyperparameters per T-slot, Sequential MNIST.

Param	T50	T100	T150	T200
hidden_size	128	64	256	64
slow_size	128	64	128	128
window_size	10	20	20	10
lra_beta	0.0784	0.5339	0.0393	0.0317
lambda_lra	0.2537	0.0836	0.4475	0.0657
slow_lr	3.84×10^{-4}	2.11×10^{-3}	3.91×10^{-4}	2.42×10^{-3}
fast_lr	2.51×10^{-4}	5.30×10^{-4}	3.44×10^{-4}	8.47×10^{-4}
batch_size	64	128	128	64
weight_decay	4.42×10^{-5}	2.06×10^{-4}	6.00×10^{-6}	2.10×10^{-5}

delimiter symbol used for prediction were used for loss calculation. This ensures that any model correctly prediting blank symbols does not get higher performance score.

G.2 Sequential and Permuted MNIST

The T-slot labels below are configuration identifiers from the tuning sweep; the MNIST sequence length is always 784. Tables 8 and 9 show HiCA-RNN settings for sequential and permuted MNIST. Tables 10 and 11 show RNN and LSTM configurations.

Table 9: HiCA-RNN tuned hyperparameters per T-slot, Permuted MNIST.

Param	T50	T100	T150	T200
hidden_size	256	256	256	256
slow_size	128	128	128	128
window_size	50	10	50	50
lra_beta	0.1929	0.0929	0.0377	0.0370
lambda_lra	1.2673	1.0879	0.7531	0.6207
slow_lr	2.27×10^{-4}	2.29×10^{-4}	3.27×10^{-4}	3.20×10^{-4}
fast_lr	2.40×10^{-4}	1.47×10^{-3}	2.04×10^{-4}	2.70×10^{-4}
batch_size	128	128	128	128
weight_decay	5.15×10^{-5}	2.11×10^{-4}	3.00×10^{-6}	4.00×10^{-6}

Table 10: RNN and LSTM tuned hyperparameters per T-slot, Sequential MNIST.

	RNN				LSTM			
T	hidden	lr	batch	weight Decay	hidden	lr	batch	weight Decay
50	128	1.50×10^{-4}	128	1.93×10^{-4}	64	3.14×10^{-4}	64	5.00×10^{-6}
100	128	1.36×10^{-4}	64	8.09×10^{-4}	256	1.48×10^{-4}	128	7.00×10^{-6}
150	128	1.03×10^{-4}	32	1.00×10^{-5}	64	5.14×10^{-4}	64	2.00×10^{-6}
200	128	1.23×10^{-4}	64	3.06×10^{-4}	64	4.23×10^{-4}	128	4.00×10^{-6}

G.3 Dynamical Systems Experimentation

Table 12 shows the HiCA-RNN configuration for every system-identification row.

G.4 Pixel-by-Pixel CIFAR-10 classification task

We conducted a parameter-budget comparison on color sequential CIFAR-10 (sCIFAR; T=1024 with one RGB pixel per step). Table 5 in the main text reports models trained under the same protocol at a ∼50K parameter budget. In line with the published settings of Lipschitz RNN (Erichson et al., 2021), we also tested HiCA-RNN and LSTM at an extended budget of 135K parameters. Peak memory was measured with `torch.cuda.max_memory_allocated` over the full training run. Table 13 lists published results on the same task under their own protocols; HiCA at 50K exceeds the Antisymmetric RNN and LSTM rows but not Lipschitz RNN at 134K.

H Implementation details and gradient routing

H.1 Gradient coverage in the experimental mode

The table records objective-dependent gradient paths for `hybrid`. A connected path permits a gradient but does not imply a nonzero derivative on every batch. Weight decay and optimizer state are separate from these loss paths. In dense prediction, the final slow-state update is downstream of all dense readouts and receives no direct task signal through that final occurrence. Shared slow parameters can still learn through earlier transitions feeding later windows. With only one window, this particular task-loss route to the slow summary is absent. Bridge parameters can receive gradients even when the initial state is fixed; individual parameter derivatives can nevertheless vanish at special inputs such as $s_0 = 0$.

H.2 Connected and detached modes are different algorithms

The optional `detached` mode sets $h_{k,0} = \mathrm{sg}(B_\phi(s_k))$, uses $S_\psi(s_k, \mathrm{sg}(h_{k,W_k}))$, and detaches the fast readouts. It is not the mode used for the reported experiments. In that mode the backward macro operator is C_k, although recomputing the numerical forward recurrence after perturbing s_k

Table 11: RNN and LSTM tuned hyperparameters per T-slot, Permuted MNIST.

	RNN				LSTM			
T	hidden	lr	batch	weight Decay	hidden	lr	batch	weight Decay
50	256	1.75×10^{-4}	128	6.00×10^{-6}	512	2.12×10^{-3}	32	8.40×10^{-5}
100	256	1.57×10^{-4}	128	1.54×10^{-4}	256	4.02×10^{-3}	64	4.00×10^{-6}
150	256	1.75×10^{-4}	128	6.00×10^{-6}	512	1.16×10^{-3}	32	2.00×10^{-6}
200	256	1.48×10^{-4}	128	1.20×10^{-5}	512	2.08×10^{-3}	32	2.00×10^{-6}

Table 12: HiCA-RNN configuration for each row of Table 4 (the LSTM baseline is 361 parameters total, H=8, at every N). Burn-in m: number of initial positions omitted from the task loss, applied only in stateless mode; it is not gradient detachment. These values exceed W and are interpreted at chunk level. Silverbox uses $m = 120$ at $N = 200$ and $m = 94$ otherwise.

Dataset	N	Mode	H	S	W	Learning rate	Burn-in m	Params
Silverbox	100	stateless	8	8	10	1×10^{-4}	94	554
	200	stateless	8	8	20	1×10^{-4}	120	554
	500	stateless	8	8	20	1×10^{-4}	94	554
	1000	stateless	8	8	20	1×10^{-4}	94	554
Wiener–Hammerstein	100	stateless	32	16	10	3×10^{-4}	–	4,434
	200	stateless	32	16	10	3×10^{-4}	–	4,434
	500	stateless	32	16	20	3×10^{-4}	–	4,434
	1000	stateless	32	16	10	3×10^{-4}	–	4,434
RLC	100	stateful	32	6	5	1×10^{-3}	–	3,088
	200	stateless	32	6	5	1×10^{-4}	110	3,088
	500	stateless	64	6	5	1×10^{-4}	130	10,160
	1000	stateless	64	6	5	1×10^{-4}	110	10,160

still produces sensitivity $C_k + D_k A_k R_k$. This distinction is necessary because stop-gradient changes the differentiation rule, not the forward values.

H.3 Alternative boundary summaries

The default summary is the gated map in equation 24. The implemented linear option uses $s_{k+1} = \mathrm{LN}_s(s_k + Uh_{k,W_k} + b_u)$. The option named `attention` uses affine query, key, and value maps

Table 13: Published sequential CIFAR-10 results, cited and not reproduced.

Model	Params	Accuracy (%)
Antisymmetric RNN (Chang et al., 2019)	36K	58.7
LSTM (Chang et al., 2019)	69K	59.7
Lipschitz RNN (Erichson et al., 2021)	34K	60.5
Lipschitz RNN (Erichson et al., 2021)	134K	64.2

Table 14: Loss paths in the hybrid trainer. Final-output tasks use O_ω; dense tasks use Q_ζ.

Parameters	Final-output tasks	Dense-output tasks
Fast cell θ	task, LRA; proxy on W_h	task, LRA; proxy on W_h
Bridge ϕ	task, LRA	task, LRA
Slow summary/norm ψ	task, LRA	task, LRA via later windows
Final head ω	task	no loss path
Auxiliary/dense head ζ	no loss path	task
Feedback E_o	feedback loss	feedback loss
Recurrent feedback E_h	no loss path	no loss path

$q(s), k(h), v(h)$ and a scalar sigmoid gate

$$\alpha = \sigma\Big(q(s_k)^\top k(h_{k,W_k})/\sqrt{d_s}\Big), \qquad s_{k+1} = \mathrm{LN}_s(s_k + \alpha v(h_{k,W_k})).$$

It acts on the terminal fast state, not a softmax over all window states. The generic Jacobian and adjoint identities apply to each differentiable summary map. The summary choice must be included in a complete experimental configuration.

H.4 Exact update order

The main Adam optimizer includes fast, slow, bridge, and both readout parameter groups. The separate feedback optimizer includes E_o and E_h, but only E_o receives a gradient. The main backward pass does not differentiate through targets or feedback construction. The feedback-free target option leaves an empty feedback parameter list; this trainer requires a guarded optimizer construction to execute that option. The feedback phase uses the retained pre-update fast state with the post-update auxiliary projection, rather than recomputing the entire forward pass. Feedback optimization is not clipped by a separate clipping call.

In the implementation, the main optimizer clears only its own parameter gradients before backpropagation, whereas the norm-clipping call includes all model parameters. After a feedback step, E_o's gradient buffer remains populated until the next feedback phase. Thus a subsequent main clipping operation includes that previous feedback gradient in its norm, even though the main optimizer does not update E_o. If the current main gradient has norm g_m and the retained feedback gradient has norm g_f, the clipping norm is $\sqrt{g_m^2 + g_f^2}$, rather than g_m. This affects the main update whenever clipping is active. The loss-gradient identities in the main text describe pre-clipping derivatives and remain valid; they are not an equivalence claim for the complete Adam update.

H.5 Partial windows and finite precision

A one-token final window contributes to the forward graph and task loss but not to $\mathcal{L}_a$ or the transition penalty. The feedback phase still uses the last state and supervised terminal target. If $W = 1$, the auxiliary loss and transition penalty are both zero throughout the call. The feedback norm identities hold in exact arithmetic. In floating-point target construction, the subtraction $\widehat{h}_{k,t+1} - h_{k,t+1}$ can become exactly zero when a small target displacement rounds away. Consequently, practical attenuation can be stronger than the real-arithmetic formula predicts.

H.6 Variable-length batches

The fixed-length analysis does not assume correctness of every padded-batch path. The implementation chooses a last valid within-window state for active examples and holds the slow state unchanged after an example ends. However, a final fast readout in a fully inactive last batch window can still use that window's clamped padded state instead of a retained last valid state. Targets are generated according to batch windows: an example ending earlier may receive a consistency target based on padding, while the terminal supervised LRA target is assigned at the final batch window. The auxiliary loss masks invalid positions but does not repair those target choices. Dense task losses apply no length mask in this trainer; the transition penalty is also unmasked. Variable-length correctness therefore requires a separate implementation revision and validation before those paths support experimental claims.