Thinking Without Forgetting - Some Initial Thoughts

Varun Reddy

1 Motivation

These ideas grew out of conversations about the brain as a recurrent system. Neural populations are linked by recurrent connections within circuits and by feedback pathways between regions [1, 2]. Through these loops, earlier activity can influence subsequent processing. This raises a question for recurrent computation: how can a changing internal state support further computation while keeping task information available?

Recurrent models let us study this question with an explicit training objective. Looped transformers reuse a learned block, and additional loops can improve or degrade performance [3, 4]. Working-memory recordings show stable stimulus coding amid changing neural activity [5], while task-trained recurrent models capture context-dependent integration [6]. To be clear, recurrent connectivity alone does not establish that brains and looped transformers use the same mechanisms.

2 Task and Recurrent Computation

A task specifies input x and required output $y = f_*(x)$. We use deterministic, continuous targets and squared error. After encoding x, a network updates hidden state h_r and reads out a prediction:

$$h_0 = E_\theta(x), \qquad h_{r+1} = F_\theta(h_r),$$
$$\hat{y}_r = g_\theta(h_r), \qquad \ell_r(x) = \tfrac{1}{2}\|\hat{y}_r - y\|^2.$$

Training minimizes average loss $\mathcal{L}_R(\theta) = \mathbb{E}_x \ell_R(x)$. We hold trained θ fixed and vary loop count. Another loop helps if it reduces this task loss. Task-trained recurrent models of cognition use this input–state–readout framework [7].

For a continuous analogue of context-dependent integration [6], $x = (u, v, c)$ contains two evidence summaries and a cue $c \in \{0, 1\}$ selecting which to report: $y = cu + (1 - c)v$. Loops follow input encoding; r measures computational depth, not the arrival of new evidence.

Let's write a reduced state $h = (m, z)$, with prediction z and initial code $m_0 = s(x)$. The code is task-sufficient: equal codes imply equal required outputs. It could retain u, v, c above. We assume (m, z) determines subsequent updates; arbitrary memory probes and readouts need not form a closed state description. Consider

$$m_{r+1} = m_r, \qquad z_{r+1} = qz_r + (1 - q)\phi(m_r), \qquad 0 < q < 1. \tag{1}$$

Here $\phi(m)$ is the learned equilibrium output under a fixed code. We assume $\phi(s(x)) = f_*(x) = y$ on initial codes; after code drift it may differ from y. Following fixed-point analyses of trained RNNs [8], equation (1) models local relaxation with the Jacobian of the z-update replaced by qI. Correctness, retention, and contraction are separate assumptions about computation after learning.

For conserved codes, prediction error is $e_r = z_r - y$, and solver endpoints form $\mathcal{M}$:

$$\mathcal{M} = \{(m, \phi(m))\}, \qquad e_r = z_r - \phi(m_r), \qquad e_{r+1} = qe_r. \tag{2}$$

Prediction error shrinks by q, so task loss shrinks by q^2 per loop. Different retained codes have distinct endpoints. The Jacobian J describes small state perturbations. At a point of $\mathcal{M}$ where ϕ is differentiable, it is

$$J = \begin{pmatrix} I & 0 \\ (1-q)D\phi & qI \end{pmatrix}, \qquad J\begin{pmatrix} v \\ D\phi\, v \end{pmatrix} = \begin{pmatrix} v \\ D\phi\, v \end{pmatrix}, \qquad J\begin{pmatrix} 0 \\ w \end{pmatrix} = q\begin{pmatrix} 0 \\ w \end{pmatrix}. \tag{3}$$

$D\phi$ measures output sensitivity to the code. A perturbation $(v, D\phi\, v)$ moves along the endpoint family and is preserved; $(0, w)$ changes only the prediction and shrinks by q. These directions need not be orthogonal. We impose retention in (1). Signal-propagation theory studies Jacobians at initialization [11, 12]; this analysis concerns assumed modes after learning. Stability and input-dependent solutions are also distinguished in looped-transformer work [9].

And we don't need memory to be a fixed hidden vector [10]. An input-injected update $F_\theta(h_r, x)$ [3] can instead retain x explicitly, with $x_{r+1} = x_r$, while hidden activity contracts. Correct use of that input remains a separate question.

3 Useful Loop Depth in a Simple Model

Now specialize to scalar regression: $x = m_0 > 0$, the required output is $y = x$, and the initial prediction is $z_0 = 0$. Here $\phi(m) = m$, and the solver receives the input through its memory code without refreshing the original x. Let that code fade:

$$m_{r+1} = a m_r, \qquad z_{r+1} = q z_r + (1-q) m_r, \qquad 0 < q < a < 1. \tag{4}$$

Since $q < a < 1$, memory attenuation is slower than prediction refinement. The prediction follows the current code while the required output stays $y = m_0$. Both eigenvalues lie inside the unit circle: the state converges to $(0, 0)$, a stable but wrong endpoint. This is an exact toy model; applying a local approximation at arbitrary depth would need validation.

After r loops, $m_r = a^r m_0$. This noiseless code remains invertible at finite depth R, since $m_0 = a^{-R} m_R$. The solver misreads information that is still present: it follows the fading value without compensation. Substituting this trajectory into the answer update gives

$$\frac{z_R}{m_0} = (1-q) \sum_{t=0}^{R-1} q^{R-1-t} a^t = (1-q) \frac{a^R - q^R}{a - q}. \tag{5}$$

The answer combines earlier memory values, giving less weight to older ones. The nonnegative weights sum to $1 - q^R$. Since $0 < a^t \le 1$, we have $0 < z_R < m_0$ for $R \ge 1$: maximizing z_R minimizes both absolute and squared error. Compare successive answers:

$$\frac{z_{R+1} - z_R}{m_0} = \frac{1-q}{a-q} [(1-q)q^R - (1-a)a^R]. \tag{6}$$

Initially this difference is positive, meaning that another loop helps. Later it becomes negative: another loop hurts. Solving for the single sign change gives the best integer loop count,

$$\boxed{R_* = \left\lceil \frac{\log[(1-q)/(1-a)]}{\log(a/q)} \right\rceil}. \tag{7}$$

The ceiling rounds up; an integer argument gives a tie with the next depth. Beyond the peak, task loss increases despite stability. With $a = 1$, equation (5) becomes $z_R/m_0 = 1 - q^R$.

3.1 Memory Retention and Refinement Rates

There are two rates to keep track of. Let $\delta = 1 - a$ measure memory leakage per loop, and let $\gamma = -\log q$ measure how quickly refinement removes error. For our model, the inequality $1 - a^t \le \delta t$ says that memory drift is bounded by the leakage per loop times the number of loops. Using it in (5) gives

$$q^R \le 1 - \frac{z_R}{m_0} \le q^R + \delta \left[R - \frac{1 - q^R}{1 - q} \right] \le e^{-\gamma R} + \delta R. \tag{8}$$

The last expression makes the tradeoff easier to read. The term $e^{-\gamma R}$ gets smaller as we compute longer; δR bounds the cost of following a drifting problem. Suppose we want relative error at most $\epsilon \in (0, 1)$. One sufficient approach is to make each term at most $\epsilon/2$. That gives a window of loop counts:

$$\left\lceil \frac{\log(2/\epsilon)}{\gamma} \right\rceil \le R \le \left\lfloor \frac{\epsilon}{2\delta} \right\rfloor. \tag{9}$$

The lower limit allows refinement; the upper limit controls accumulated memory drift. Any integer in this window has the stated error guarantee. An empty window does not imply failure: the bound is sufficient. For fixed q and $\delta \to 0$, optimal depth grows only logarithmically: $R_* \sim \gamma^{-1} \log[(1-q)/\delta]$.

4 Memory and Its Use

The possibility I find interesting is this concept of "target motion": the solver's equilibrium output changes with its code, while the required output y stays fixed. In (4), these are m_r and m_0. If this turns out to be true in trained networks, useful depth could depend on continued interpretation of evolving representations.

With the same fading code, let the solver read $a^{-r} m_r$. Then $z_{r+1} = q z_r + (1-q) a^{-r} m_r$ gives $z_R = (1 - q^R) m_0$: every loop helps. Compensation requires known a and loop index and amplifies noise. Matching memory codes to decoding rules is established in recurrent memory models [13].

4.1 Constraining the Recurrent Update

It has been shown that recall combined with progressive training can prevent overthinking on algorithmic tasks [14]. Yet input injection does not resolve it in the controlled setting of [4]. Recent mechanistic work also finds that an intermediate entity can remain decodable while becoming ineffective for the next operation, with interventions restoring that use [15]. These results motivate the distinction; they do not establish that our scalar mechanism explains those networks.

One possible constraint uses a projection P, where $P^2 = P$. It selects task information defined independently of final accuracy, such as a context cue and presented evidence. Let h_0 be the initialized state and F_θ the learned update. We could try

$$h_{r+1} = Ph_0 + (I - P)F_\theta(h_r, x). \tag{10}$$

Since $P(I-P) = 0$, we have $Ph_{r+1} = Ph_0$: the selected component is preserved. Choosing P is unresolved; freezing the wrong component can obstruct computation. For a linear task $y = Am_0$, a changing code $m_{r+1} = Tm_r$ retains the required quantity whenever $AT = A$. Task-irrelevant directions can change or disappear.

With a fixed interpretation ϕ, a small code change dm shifts the solver's target by $D\phi(m)\,dm$. For $e = z - \phi(m)$, the corresponding first-order change is

$$de = dz - D\phi(m)\,dm. \tag{11}$$

This tracks error against the current equilibrium, whereas task error remains $z-y = z-\phi(m_0)$. Approaching that equilibrium can increase task loss. Compensation can keep the solver's effective target fixed despite code changes. Otherwise, $D\phi$ determines how strongly code drift shifts it.

This connects familiar ingredients; the proposed transformer mechanism remains unproven. Restoring access to known task facts could extend useful depth while their code keeps changing. The test is improved subsequent computation, beyond external decoding. Overthinking can also involve lost computational progress [15] or overly large updates [16]. Whether trained networks organize their computation this way is a question worth pursuing!

References

[1] S. Song et al. Highly Nonrandom Features of Synaptic Connectivity in Local Cortical Circuits. PLOS Biology 3:e68, 2005.

[2] N. T. Markov et al. Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. Journal of Comparative Neurology 522:225–259, 2014.

[3] J. Geiping et al. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171, 2025.

[4] H. Kohli, S. Parthasarathy, H. Sun, and Y. Yao. Loop, Think, & Generalize: Implicit Reasoning in Recurrent-Depth Transformers. COLM, 2026. arXiv:2604.07822.

[5] J. D. Murray et al. Stable population coding for working memory coexists with heterogeneous neural dynamics in prefrontal cortex. PNAS 114:394–399, 2017.

[6] V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. Nature 503:78–84, 2013.

[7] G. R. Yang, M. R. Joglekar, H. F. Song, W. T. Newsome, and X.-J. Wang. Task representations in neural networks trained to perform many cognitive tasks. Nature Neuroscience 22:297–306, 2019.

[8] D. Sussillo and O. Barak. Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. Neural Computation 25:626–649, 2013.

[9] A. Labovich. Stability and Generalization in Looped Transformers. arXiv:2604.15259, 2026.

[10] A. Parthasarathy et al. Mixed selectivity morphs population codes in prefrontal cortex. Nature Neuroscience 20:1770–1779, 2017.

[11] M. Chen, J. Pennington, and S. S. Schoenholz. Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal Propagation in Recurrent Neural Networks. ICML, PMLR 80:873–882, 2018.

[12] D. Gilboa et al. Dynamical Isometry and a Mean Field Theory of LSTMs and GRUs. arXiv:1901.08987, 2019.

[13] E. P. Frady, D. Kleyko, and F. T. Sommer. A theory of sequence indexing and working memory in recurrent neural networks. arXiv:1803.00412, 2018.

[14] A. Bansal et al. End-to-end Algorithm Synthesis with Recurrent Networks: Extrapolation without Overthinking. NeurIPS 35:20232–20242, 2022.

[15] J. Liang, X. Jin, and L. Pan. Beyond the Training Horizon: Mechanisms and Limits of Length Generalization in Looped Transformers. arXiv:2609.33144, 2026.

[16] Z. Guo et al. Right Direction, Wrong Step: Geometric Analysis of Finite-Step Failure in Looped Transformers. arXiv:2609.16665, 2026.

Thank you to Sab and Gyu for the discussions and ideas that motivated this note, and to the Kempner Institute for facilitating these conversations.