Dissecting HOPE: Do Self-Modifying Memories Actually Self-Modify?

Junghun Kim
Multi::Turn
founder@multi-turn.ai

Abstract

HOPE (Higher-Order Parametric Experience) extends the Titans memory architecture with three novel mechanisms: multi-timescale contextual memory (CMS), self-modifying memory parameters, and surprise-gated updates. While the original paper demonstrates strong aggregate performance, the individual contribution of each mechanism remains unclear. We present a systematic dissection of HOPE through component ablations, internal dynamics analysis, and scaling experiments. Our findings reveal a striking asymmetry: (1) self-modification is the sole driver of HOPE's advantage, with its importance scaling $4.5\times$ from small to larger models; (2) the multi-timescale CMS provides no benefit at either scale, with fast and slow levels learning redundant representations (cosine similarity ≈ 0.77); (3) surprise gating is effectively inoperative, as per-token losses remain far above any reasonable threshold (mean $= 66.7$, min $= 9.3$), causing the gate to fire on $>99.9\%$ of tokens. We further show that self-modification manifests as an immediate initialization shift rather than continual online adaptation. These results suggest that HOPE's complexity can be substantially reduced without sacrificing performance.

1 Introduction

Neural architectures with learnable memory systems have gained renewed interest as alternatives to the fixed key-value caches of standard Transformers. Titans (**?**) introduced a memory module that stores and retrieves associative patterns through gradient-based updates, demonstrating competitive performance with subquadratic complexity. HOPE (**?**) extends Titans with three mechanisms designed to improve memory utilization:

1. **Contextual Memory System (CMS)**: A multi-timescale hierarchy where fast, mid, and slow memory levels update at different frequencies, inspired by complementary learning systems theory (**?**).
2. **Self-modifying memory**: Meta-learned parameters that generate the memory's key, value, query, and learning rate projections at inference time, enabling the model to adapt its own update rule.
3. **Surprise gating**: A mechanism that gates memory updates based on per-token prediction loss, directing compute toward "surprising" inputs.

While HOPE reports improvements over Titans and Transformers, the original evaluation presents only aggregate metrics. This leaves open a critical question: *which mechanisms actually contribute to performance, and do they function as theoretically motivated?*

Following the spirit of mechanistic analysis works like "Titans Revisited" (**?**), we systematically ablate each component and analyze its internal dynamics. Our contributions are:

Preprint. Under review.

- A complete component ablation at two scales showing that self-modification alone accounts for HOPE's advantage (**??**).
- Evidence that CMS fast/slow levels learn redundant representations and provide no measurable benefit (**??**).
- Analysis showing surprise gating is effectively non-selective at the loss magnitudes encountered during training (**??**).
- Characterization of self-modification dynamics as an initialization shift rather than continual adaptation (**??**).
- A scaling analysis revealing that self-modification's importance grows superlinearly with model size (**??**).

2 Background

Titans. The Titans architecture (**?**) augments attention with a neural long-term memory module. Given an input sequence, the memory M is updated via gradient descent on an associative recall objective: $M \leftarrow M - \eta \nabla_M \mathcal{L}_{\text{assoc}}(M; k, v)$, where k, v are key-value pairs derived from the input. A surprise metric $S_t = \|\mathcal{L}_t\|$ determines whether each token triggers a memory update.

HOPE. HOPE extends Titans along three axes. First, it introduces a *Contextual Memory System* (CMS) that maintains memory at multiple timescales: a fast level updates every p_{fast} tokens, a slow level every p_{slow} tokens ($p_{\text{slow}} > p_{\text{fast}}$), with outputs combined via learned gating. Second, it adds *self-modification*: rather than using fixed projection matrices for keys, values, queries, and learning rates, HOPE uses meta-learned networks $m_k, m_v, m_q, m_\eta, m_\alpha, m_{\text{memory}}$ that generate these projections conditioned on a teaching signal. The generated parameters constitute a "fast state" that can diverge from the meta parameters during inference. Third, the surprise threshold is tuned as a hyperparameter controlling the selectivity of memory updates.

3 Experimental Setup

Model configurations. We evaluate two scales: *Small* (dim=256, 6 layers, $\sim$19M parameters) and *Scaled* (dim=512, 8 layers, $\sim$73M parameters). All models use a context length of 512 tokens and are trained for 2,000 steps with AdamW (**?**) (lr=3e-4, batch size=8). CMS uses update periods of $p_{\text{fast}} = 8$ and $p_{\text{slow}} = 32$.

Ablation variants. We train the following configurations at each scale:

- **Full HOPE**: All three mechanisms enabled (CMS + self-mod + surprise).
- **No CMS**: Self-modification and surprise without multi-timescale memory.
- **No Self-Mod**: CMS and surprise without self-modifying parameters.
- **No Surprise**: CMS and self-modification with surprise threshold=0.
- **Transformer**: Standard Transformer baseline.
- **Titans**: Original Titans architecture (CMS without self-mod or surprise).

At the scaled setting, we additionally train a *parameter-matched Transformer* (dim=768, 81.2M parameters) to control for HOPE's larger parameter count.

Data. All models are trained on the OpenWebText dataset (**?**) using a GPT-2 BPE tokenizer (vocab size 50,257).

Analysis tools. We save checkpoints every 200 steps and extract: CMS level parameter norms and inter-level similarity; self-modification parameter drift from initialization; per-token surprise value distributions; and meta-parameter vs. fast-state divergence.

4 Results

4.1 Component Ablation

?? presents the final training loss for each variant.

Table 1: Component ablation results. Δ is the loss difference from Full HOPE (positive = worse). Best result per scale in **bold**.

	Small (256d, 6L)		**Scaled (512d, 8L)**	
Variant	Loss	Δ	Loss	Δ
Full HOPE	7.216	—	7.237	—
No CMS	**7.201**	-0.015	**7.079**	-0.158
No Self-Mod	7.532	+0.316	8.649	+1.412
No Surprise	7.322	+0.106	—	—
Transformer	6.880	-0.336	7.830	+0.593
Transformer (matched)	—	—	7.936	+0.700
Titans	7.578	+0.362	—	—

Three patterns emerge. First, **removing CMS consistently improves performance**: -0.015 at small scale and -0.158 at scaled. The multi-timescale hierarchy adds parameters and computation without contributing to learning. Second, **removing self-modification severely degrades performance**: +0.316 at small scale, escalating to +1.412 at scaled—a 4.5$\times$ increase in importance. Third, at small scale the Transformer baseline outperforms all HOPE variants (-0.336), but at scaled HOPE overtakes it (+0.593), and even a parameter-matched Transformer with 81.2M parameters (vs. HOPE's 73.1M) cannot close the gap (+0.700).

4.2 CMS Multi-Timescale Dynamics

The CMS is motivated by complementary learning systems theory, which posits that fast and slow learning systems capture different types of information. We test whether this separation emerges in practice (**??**).

Parameter convergence. The L2 norms of fast and slow level parameters remain within 0.2% of each other throughout training ($\sim$26.15 for both), and their inter-checkpoint drift magnitudes are nearly identical ($\sim$29.2–29.3).

Output redundancy. The cosine similarity between fast and slow level outputs averages 0.77 across training, with no trend toward differentiation. This high similarity indicates that despite updating at different frequencies ($p_{\text{fast}} = 8$ vs. $p_{\text{slow}} = 32$), both levels converge to similar representations.

Uniform layer behavior. Per-layer analysis shows that CMS norms are identical across all 6 layers (**??**d), suggesting no layer-specific specialization of the memory hierarchy.

These findings explain why removing CMS improves performance: the two levels perform redundant computation while adding parameters that must be optimized.

4.3 Self-Modification Convergence

HOPE's self-modification mechanism generates "fast state" parameters from meta-learned networks. We analyze whether this mechanism produces meaningful adaptation (**??**).

Drift from initialization. We track 84 self-modification parameters across 6 layers and 6 memory components ($m_k, m_v, m_q, m_\eta, m_\alpha, m_{\text{memory}}$). The large weight matrices (w_1, w_2 of projections) show substantial L2 drift ($\approx$13) from their Kaiming initialization, while smaller components (skip connections in m_η, m_α) drift by $\approx$0.8. Critically, **this drift occurs within the first 200 training steps and remains essentially flat thereafter** (**??**a). This suggests that self-modification functions as a learned initialization offset rather than continual online adaptation.

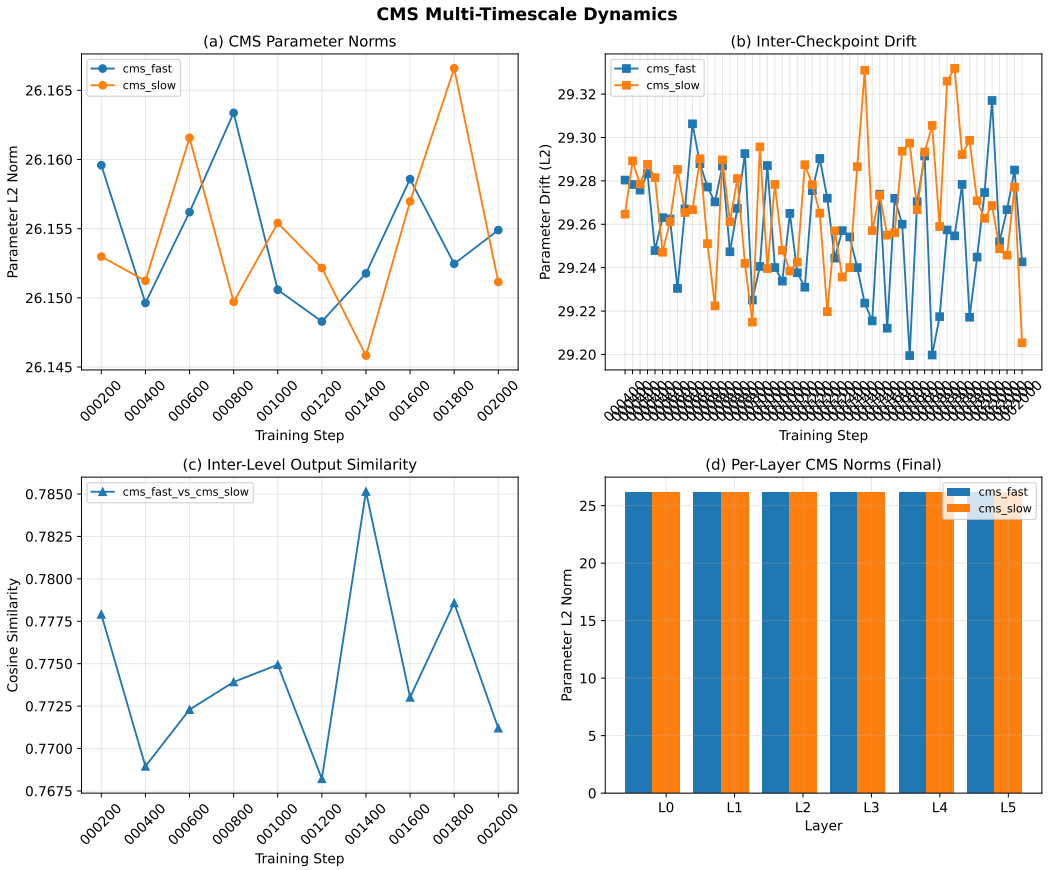

Figure 1: CMS dynamics across training. (a) Parameter norms remain nearly identical between fast and slow levels. (b) Inter-checkpoint drift is similar for both levels. (c) Cosine similarity between fast and slow level outputs averages 0.77, indicating high redundancy. (d) Per-layer norms show uniform behavior across all 6 layers.

Meta vs. fast state divergence. To verify that self-modification actually occurs at inference time, we measure $\|\theta_{\text{fast}} - \theta_{\text{meta}}\| / \|\theta_{\text{meta}}\|$. This relative drift is $\approx$1.0 for all components (**??**b), confirming that fast state parameters diverge substantially from their meta-learned generators. The consistency across components and layers indicates that the self-modification mechanism is uniformly active rather than selectively engaged.

Interpretation. Self-modification does occur—fast state parameters are meaningfully different from meta parameters. However, the temporal dynamics reveal that the meta parameters quickly find a configuration whose generated fast states are useful, and then both meta and fast states evolve together with a fixed offset. This is closer to a "learned reparameterization" than the dynamic self-modification suggested by the HOPE framework.

4.4 Surprise Gating Effectiveness

Surprise gating is designed to focus memory updates on tokens that the model finds difficult to predict. We evaluate whether this selective mechanism actually discriminates between tokens (**??**).

Loss distribution. At the model's training scale, per-token cross-entropy losses are concentrated in a high range: mean $= 66.7$, std $= 16.5$, with even the minimum loss at 9.3 (**??**a). The 10th percentile is 45.0, meaning that even the "easiest" tokens have losses far exceeding any reasonable surprise threshold.

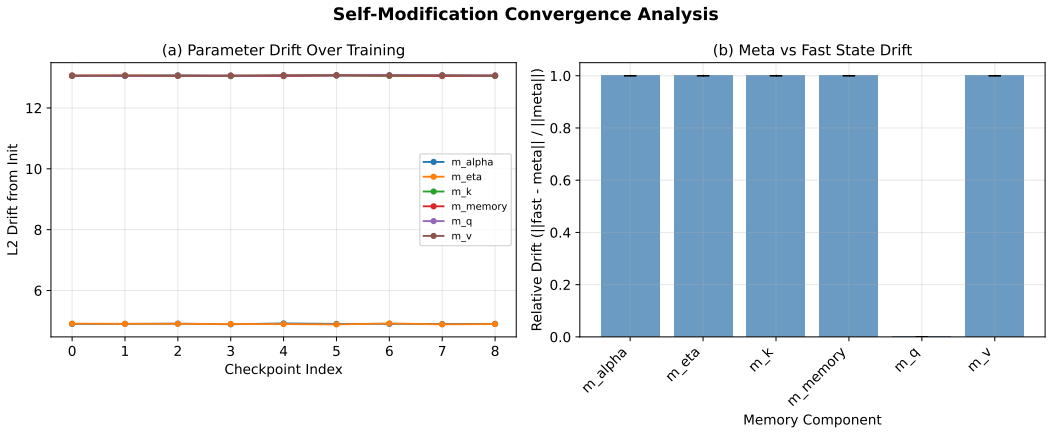

Figure 2: Self-modification dynamics. (a) L2 drift of self-mod parameters from initialization over training. Large projection matrices (m_k, m_v, m_{memory}) show drift ≈13 that plateaus immediately; small components (m_η, m_α) drift ≈0.8–5. (b) Relative drift between meta parameters and fast state is ≈1.0 for all components, confirming self-modification occurs.

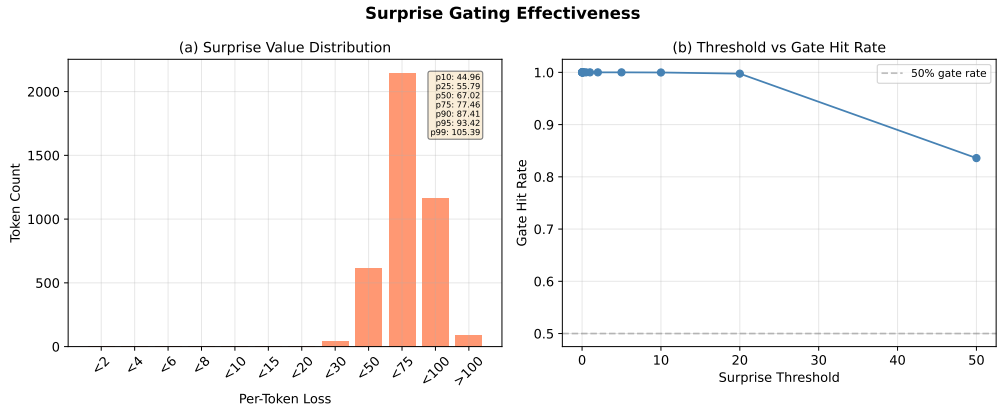

Figure 3: Surprise gating analysis. (a) Distribution of per-token loss values, with the bulk concentrated between 50 and 100 (median = 67.0). (b) Gate hit rate vs. surprise threshold: even at threshold=20, 99.8% of tokens pass the gate.

Gate selectivity. **??**b shows that the gate fires on 100% of tokens for thresholds up to 5.0, 99.97% at threshold=10, and still 99.8% at threshold=20. Only at threshold=50 does the gate begin to filter tokens meaningfully (83.6% pass rate), but at this level the gate is rejecting the easiest tokens that the model has already partially learned.

Threshold sweep. **??** shows that surprise thresholds in the range 0.02–0.05 yield marginal improvement ($\Delta = +0.006$), while extreme values hurt. The optimal range corresponds to thresholds so low relative to actual losses that they are functionally equivalent to no gating.

Table 2: Surprise threshold sensitivity (small scale). All thresholds pass >99.9% of tokens, yielding negligible performance differences.

Threshold	0.005	0.01	0.02	0.05	0.1
Final Loss	7.289	7.243	7.222	7.222	7.304
Δ vs. Full HOPE	+0.073	+0.026	+0.006	+0.006	+0.088
Gate pass rate	100%	100%	100%	100%	100%

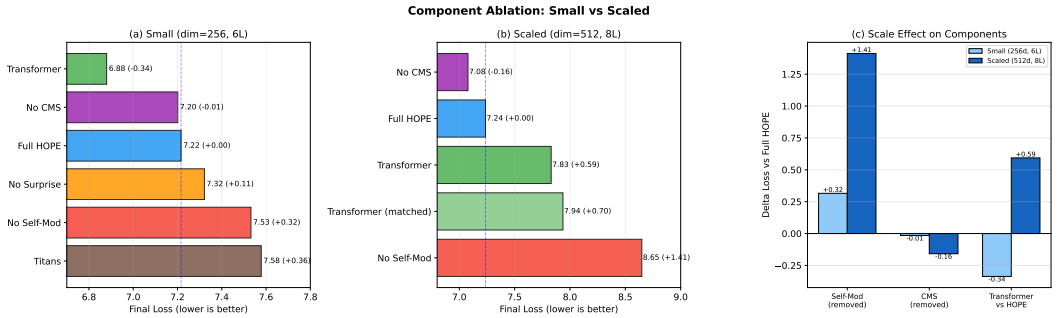

Figure 4: Component importance across scales. (a) Small-scale ablation (dim=256, 6L). (b) Scaled ablation (dim=512, 8L). (c) Scale effect comparison showing that self-modification importance grows 4.5×, while the scale reversal between Transformer and HOPE shifts from −0.34 to +0.59.

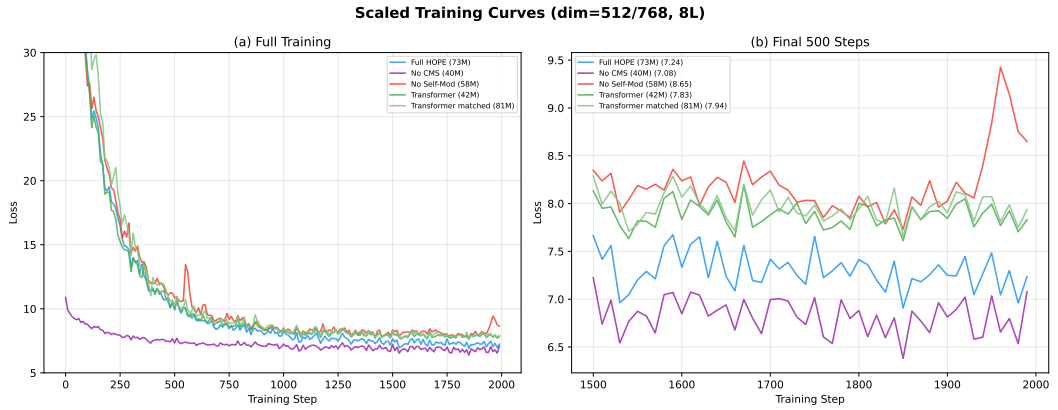

Figure 5: Scaled training curves (dim=512, 8L). (a) Full training showing convergence patterns. The No Self-Mod variant converges significantly slower and to a higher loss. (b) Final 500 steps showing clear separation between variants.

Interpretation. The surprise gate is effectively a no-op in practice. The per-token losses at this model scale are orders of magnitude above the threshold values that yield the best performance. This mechanism would require either (a) substantially larger, better-trained models where some tokens genuinely have near-zero loss, or (b) a relative rather than absolute surprise metric. In its current form, every token is "surprising."

4.5 Scaling Effects

The most striking finding is how component importance changes with scale (**??**c):

- **Self-modification scales superlinearly**: Removing self-mod costs +0.32 at small scale but +1.41 at scaled—a 4.5× increase. This suggests that the self-modification mechanism becomes increasingly important as the model's capacity grows, possibly because larger models have more parameters that benefit from adaptive reparameterization.
- **CMS remains harmful at both scales**: The penalty for including CMS grows from +0.015 to +0.158, suggesting that the redundancy problem worsens with scale.
- **Transformer-HOPE reversal**: At small scale, the Transformer wins by 0.34; at scaled, HOPE wins by 0.59. This reversal is entirely attributable to self-modification, since the "No CMS" variant (self-mod without CMS) achieves the best performance at both scales.

The parameter-matched comparison is particularly informative. A Transformer with 81.2M parameters (exceeding HOPE's 73.1M) achieves loss 7.936—worse than HOPE's 7.237. Increasing Transformer capacity does not replicate the benefit of self-modification, confirming that HOPE's advantage stems from the *mechanism* rather than *parameter count*.

5 Discussion

The essential HOPE. Our results suggest that HOPE's effective architecture is considerably simpler than presented: a self-modifying memory module without CMS hierarchy or surprise gating. The "No CMS" variant—which retains only self-modification and surprise (with surprise being a no-op)—achieves the best performance at both scales tested. This implies that future work should focus on understanding and improving the self-modification mechanism rather than the surrounding infrastructure.

Why does CMS fail? The multi-timescale hierarchy assumes that different update frequencies will naturally induce specialization. Our analysis shows this does not occur: both levels converge to similar representations despite $4\times$ different update periods. We hypothesize that the gradient signal through both levels is too similar (both receive the same loss), and without an explicit diversity-promoting objective, the system finds it easier to learn a single good representation twice.

Nature of self-modification. The self-modification mechanism is better characterized as a "learned reparameterization" than dynamic online adaptation. Meta parameters converge to generators that produce useful fast states, but this mapping stabilizes early in training. This is reminiscent of hypernetwork (**?**) behavior, where the generating network learns a compact representation of the target network's weights. The performance benefit may stem from the implicit regularization and parameter sharing this introduces, rather than from dynamic adaptation during inference.

Limitations. Our experiments use a single GPU with models up to 73M parameters and 2,000 training steps. The conclusions about CMS and surprise gating may not hold at larger scales where (a) per-token losses could be low enough for surprise gating to activate, and (b) longer training might allow CMS levels to differentiate. The self-modification scaling trend, however, suggests its importance only increases with scale.

6 Related Work

Memory-augmented architectures. Neural Turing Machines (**?**) and Memory Networks (**?**) introduced external memory for neural networks. More recently, Titans (**?**) proposed gradient-based memory updates, and "Titans Revisited" (**?**) provided independent analysis of that architecture. Our work extends this analytical tradition to HOPE.

Self-modifying networks. The idea of networks that modify their own weights dates to Schmidhuber's self-referential weight matrices (**?**). Modern instantiations include hypernetworks (**?**) and fast weight programmers (**?**). HOPE's self-modification mechanism is closest to hypernetworks, as our analysis suggests.

Surprise and gating. Surprise-based learning has roots in predictive coding (**?**) and has been applied to attention mechanisms (**?**). Our finding that surprise gating is inoperative at practical scales highlights a gap between theoretical motivation and implementation.

7 Conclusion

We presented a systematic dissection of HOPE's three novel mechanisms. Our analysis reveals that self-modification is the sole effective contributor to HOPE's performance advantage, with its importance scaling superlinearly with model size ($4.5\times$ from small to scaled). The multi-timescale CMS learns redundant representations and hurts performance at both scales tested, while surprise gating is rendered inoperative by the high per-token losses encountered during training. We further characterize self-modification as a learned initialization shift rather than continual online adaptation.

These findings have practical implications: HOPE's architecture can be simplified by removing CMS and surprise gating without loss of performance. More broadly, our work highlights the importance of component-level analysis for complex neural architectures, where the interaction between mechanisms may differ substantially from theoretical expectations.