Working Memory in the Universal Model

From External Scaffolding to UM-Native Temporal State

Claude and MJC — February 22, 2026

Abstract

The UM's pattern set P defines static connections between events—the connectome. But predicting the next byte requires knowing previous bytes: temporal state that persists across time steps. Current implementations fake this with an external byte buffer and hash-based context lookup. We replace this scaffolding with UM-native working memory: a chain of *memory event spaces* connected by absolute shift patterns, with the output ES feeding back as input. The model is an agent whose "action" is a byte prediction; training is the special case where the environment provides the correct action. We develop the bigram case and show that the remaining temporal ambiguity—what the connectome's topological sort leaves undetermined—is localized to the organism's boundaries: when output is sampled, when observation arrives, and when learning occurs.

1. The Problem

Consider the simplest prediction task: given input byte a at time t, predict byte b at time $t+1$. The bigram distribution $P(b \mid a)$ requires observing the joint event (a, b).

In the UM, a joint event between ES_I (input) and ES_O (output) would create an LPP connecting them: $I[a] \to O[b]$ with learned support. But when we observe b at time $t+1$, a is no longer in ES_I—it was replaced by b. The input event a is gone.

This is the fundamental temporal problem: the UM's forward pass f operates on a single thought vector T, but prediction requires information from *previous* thought vectors.

1.1. The Current Hack

In `#umr_settle` and `#umr_kn6`, the solution is external scaffolding:

- A C array `prev[]` stores the last k input bytes.
- A hash function `mkh()` maps these bytes to a context key.
- The context key indexes into a hash table that stores counts.

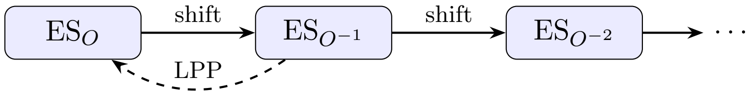
This works for scoring but is not UM-native: the byte buffer exists outside (E, P, T), the hash function is arbitrary, and nothing is representable in SN format. The model's temporal structure is invisible to the UM formalism.

2. The Agent Model

The solution begins with reframing: text prediction is a special case of an agent interacting with an environment. The agent has an ability to "speak"—to produce an output byte. In the Hutter Prize setting, we judge the agent on prediction (because we have the full text), but the agent's architecture is that of an organism acting in the world.

2.1. Output as Input

The key insight: O is both a prediction and an input. After the agent produces (or observes) a byte, that byte becomes part of the context for the next prediction. Rather than maintaining a separate input ES that duplicates the output, we let O feed directly into the memory chain:



Input happens "at the output layer." The environment (or, during training, the dataset) sets O to the observed byte. This then shifts back through the memory chain: $O \to O^{-1} \to O^{-2} \to \cdots$. The separate ES_I that appeared in our earlier formulation is unnecessary.

2.2. Training vs. Inference

The only difference between training and inference is what happens at O:

- **Training (teacher forcing)**: the environment provides the correct byte. Set O to the observed value (support 255). Learn. Shift.
- **Inference (sampling)**: sample from the output distribution at O. The sampled byte collapses the distribution to a single choice (support 255 for the sampled event, 0 for others). Shift.

In both cases, after O is determined, the same shift-and-learn sequence executes. The architecture is identical; only the source of the observation differs. This is biologically plausible: an organism acts, observes the consequence, and learns—the same circuit handles both prediction and reaction.

For the Hutter Prize, we do teacher-forced training on the full dataset: at each position, we predict the next byte (scoring the prediction), then observe the actual byte (setting it in O), learn from the observation, and shift.

2.3. Training Completion

The agent model gives an elegant criterion for when training is done. An LPP that has been trained on the entire dataset has already seen the full distribution of joint events it can observe. Further passes over the same data cannot improve its counts—it has learned everything the data contains.

This extends naturally to late-born neurons. When threshold creation produces a new conjunction neuron (say the "th" neuron, born at position t_0), the neuron and its downstream LPPs have missed all data before t_0. But nothing prevents a second pass: retrain from the beginning up to t_0 (since the neuron now exists and can accumulate support), then continue forward through the rest of the dataset. After one complete pass from t_0 forward plus one retroactive pass from the start to t_0, the neuron's LPPs have seen the full dataset and can be frozen.

This is a natural training schedule: online learning creates new structure, retroactive passes fill in what was missed, and each LPP freezes after seeing the complete data exactly once from the perspective of its events.

3. The Shift Chain

Each memory ES O^{-k} holds the output from k steps ago. The *shift patterns* copy events down the chain:

$$O[e] \to O^{-1}[e], \quad O^{-1}[e] \to O^{-2}[e], \quad \ldots$$

Each shift pattern has weight $w = 255$ (absolute/definitional). The direction of the shift is determined by P: atomic patterns $e_i \to e_j$ have a direction, and these patterns point from O toward O^{-k}, not the reverse.

3.1. The Bigram Case

For bigram prediction, we need one memory ES: O^{-1}.

Event spaces:

- ES_O: 256 events (output/prediction byte).
- $\text{ES}_{O^{-1}}$: 256 events (previous byte).

Patterns (P):

- 256 shift patterns: $O[b] \to O^{-1}[b]$ for each byte b, weight $w = 255$.
- 1 LPP: connecting O^{-1} to O, recording the bigram distribution. Entries learned via ω_0 (log-stochastic counting).

Time step at position t:

1. **Forward pass** (f): apply patterns via topological sort. The LPP from O^{-1} to O propagates support, producing the predicted distribution at O.
2. **Score**: measure $\log_2 P(x_t)$ from the distribution at O.
3. **Observe**: set O to the actual byte x_t (support 255).
4. **Learn** (ω): observe the joint event $(O^{-1}[x_{t-1}], O[x_t])$ and update the LPP.
5. **Clear + Shift**: clear memory ESs, then apply shift patterns. O's current contents (x_t) are copied to O^{-1}.

After this sequence, O^{-1} holds x_t, ready for the next step's bigram prediction.

4. Clearing: The Transience of Support

The shift chain requires that memory ESs be cleared before new values are shifted in. Patterns in the UM are positive only: $O[a] \to O^{-1}[a]$ at $w = 255$ will set $O^{-1}[a]$ to $\min(255, 255) = 255$. But it does nothing to the *previous* contents of O^{-1}. If $O^{-1}[b]$ was at 255 from the last step, it stays at 255. After the shift, two events in O^{-1} would have full support—not what we want.

The target ES must be *cleared* before the shift: all events set to support 0.

This is biologically natural. Neurons fire transiently; it is the *persistence* of support across time steps that requires explanation, not its decay. An ES that clears at each time step is the default; an ES that persists is the special case.

5. Dynamics in SN

The current SN format represents (E, P, T): event spaces, patterns, and the thought vector. It does not represent the dynamics f. For purely acyclic connectomes this is fine—f is determined by the topological sort of P.

Working memory introduces temporal structure that P's topology alone does not determine. What remains undetermined is localized to the organism's boundaries: when observation arrives, when sampling occurs, when ω runs, and which ESs are cleared. The interior of the organism—the patterns and their execution order—is handled by topological sort.

5.1. Boundary Pragmas

We propose extending the SN format with a set of *pragmas*: instructions that specify aspects of f at the organism's boundaries. The boundaries are unavoidably "magical"—they are where the model meets the world—so privileging them in the format is appropriate.

```
ES "O" OUTPUT
ES "O(-1)" CLEAR
ES "O(-2)" CLEAR

DYNAMICS {
  f           -- forward pass (topo sort of P)
  score       -- read output distribution at O
  observe O   -- set O from environment / sample
  omega       -- learning pass on all LPPs
  clear       -- zero all CLEAR-annotated ESs
  f           -- forward pass again (propagates shift)
}
```

The `OUTPUT` tag marks the organism's boundary. The `CLEAR` tag marks ESs with transient support (zeroed at the clearing step). The `DYNAMICS` block is a short imperative program specifying the order of standard operations within each time step.

5.2. Design Decision

There are two clean approaches to representing dynamics:

1. **Imperative**: a `DYNAMICS` block names the phases and their order explicitly, as above. Clear, direct, easy to implement.

2. **Annotative**: annotations on patterns and LPPs (e.g., "this LPP participates in ω after observation") that the runner fits into an execution plan by topological sort plus boundary constraints.

These are different perspectives on the same structure. The imperative approach makes the execution order explicit; the annotative approach derives it. We choose the imperative approach for now: it is simpler to implement, easier to debug, and makes the dynamics visible in the SN file. This decision can be revisited when models grow more complex and the execution order becomes harder to write by hand.

5.3. What P Still Determines

The shift direction is fully determined by P. Atomic patterns $O[b] \to O^{-1}[b]$ have an explicit direction; the topological sort of the coarsened connectome $\bar{G}(P)$ gives the execution order for the forward pass. Shift patterns are not privileged with a special name—they are ordinary absolute patterns that happen to copy between memory ESs.

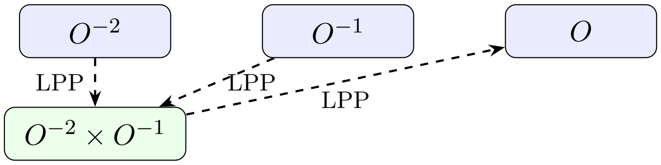
The `DYNAMICS` block does not name specific patterns. It names *operations* (f, ω, clear, observe) that act on the entire model or on boundary ESs. The internal ordering of patterns within f is always derived from P.

6. Joint Events and Conjunction Patterns

Higher-order patterns require conjunction: a trigram "the" needs both $O^{-2}[\mathtt{t}]$ and $O^{-1}[\mathtt{h}]$ to be active before predicting $O[\mathtt{e}]$. With only positive patterns, how do we accomplish this?

The answer is already present in the UM's learning machinery. Joint events are what LPPs record. Consider the LPP between O^{-2} and O^{-1}:

1. The LPP observes joint events: $(O^{-2}[\mathtt{t}], O^{-1}[\mathtt{h}])$.
2. When this joint event reaches the creation threshold τ, a new neuron is born: the event "$\mathtt{t} \wedge \mathtt{h}$" in the conjunction space of O^{-2} and O^{-1}.
3. This neuron receives support from the same LPP mechanism that created it: whenever $O^{-2}[\mathtt{t}]$ and $O^{-1}[\mathtt{h}]$ are both active, the conjunction neuron gets support.
4. A second LPP connects the conjunction space to O, recording the trigram distribution.



6.1. Conjunction as Product

The conjunction space of two ESs is their product: $O^{-2} \times O^{-1}$. This product "already exists" in the sense that the LPP between O^{-2} and O^{-1} records exactly its joint events. The conjunction neurons are not in a separate ES so much as they are the materialized elements of this product—the joint events that have reached sufficient support to become first-class citizens in E.

This is naturally hierarchical. Bigram conjunctions ($O^{-1} \times O$) feed into trigram conjunctions ($(O^{-2} \times O^{-1}) \times O$), and so on. Each level of the hierarchy adds one layer to the DAG, and the topological sort handles execution order automatically.

The SN representation is efficient: conjunction events are indexed by the LPP's observed joint events. Since the LPP already records these (that is its function), the conjunction events need not duplicate the information—they reference it. A conjunction event's identity *is* its position in the LPP's joint event table.

7. Remaining Uncertainty

7.1. Clearing Semantics

Clearing an ES (setting all supports to 0) is not a pattern operation—it is a dynamics operation, part of f. The UM's max-min forward pass is purely positive: $f_P(T)_j = \max_i \min(T_i, P_{ij})$. Clearing introduces a reset that operates outside this framework.

For now, we represent clearing as a pragma in SN and implement it as an explicit zeroing step in the runner. This is pragmatic: it works, and the biological analogy (transient firing) is natural. Whether clearing can be derived from a more fundamental principle—perhaps complementary events, or a decay rate in support—is an open question.

7.2. Settling and Multi-Scale Memory

With pattern chains operating at different time scales, different patterns may have different support gaps at different times. A bigram pattern may be sharp (high $s_1 - s_2$) while a trigram pattern is still accumulating evidence. Some mechanism in P is needed to detect and gate on these multi-scale states—connecting to the ring pattern construction and sharpness preference work. We defer this to future work.

7.3. SN Round-Trip

A complete working-memory model in SN must include:

- Event spaces with `OUTPUT` and `CLEAR` annotations.
- Shift patterns (absolute, $w = 255$).
- LPPs with their learned entries.
- Conjunction events indexed by LPP joint events.
- A `DYNAMICS` block specifying boundary timing.

The round-trip invariant must hold: save the model after training, load it, run inference on the same data, get the same predictions. This requires that clearing semantics and boundary timing are preserved across save/load.

8. Summary

1. The model is an agent. O is both prediction and input. Training = teacher forcing (environment sets O). Inference = sampling (agent sets O). Same architecture.

2. Training completion is natural: each LPP freezes after seeing the complete dataset once from its events' perspective. Late-born neurons get a retroactive pass.

3. The shift chain $O \to O^{-1} \to O^{-2} \to \cdots$ is UM-native: absolute patterns ($w = 255$) in P. Direction determined by P. No separate ES_I needed.

4. Memory ESs have transient support (cleared each step). This is biological default. Annotated as `CLEAR` in SN.

5. What P's topology leaves undetermined is localized to the organism's boundaries. SN `DYNAMICS` block specifies the boundary timing as a short imperative program.

6. Conjunction patterns are product events materialized from LPP joint observations. Naturally hierarchical, efficiently indexed.

7. The bigram case is the minimal exercise: one memory ES, one LPP, clear + shift. If this matches bigram counting, we have a working P-programming pattern for temporal state that extends to higher orders.