# Deriving Layers from the Connectome

v3: Neuron-First Explorer and Sparse LPPs

Claude and MJC

18 February 2026

**Abstract**

In the Universal Model $u = (E, P, t, f)$, the pattern set $P$ defines a directed graph on $E$—the *connectome*—whose structure constrains but does not fully determine the forward pass $f$. When the connectome is a DAG, $f$ is determined (up to parallelism) by topological sort. When the connectome has cycles, $P$ is *ambiguous*: multiple forward passes are consistent with the same patterns. This ambiguity must be resolved by $f$ as part of the model specification; the UM runner surfaces the ambiguity rather than resolving it. We describe the RNN's input-stream synchronization as a specific choice of $f$, sketch the UM explorer as a neuron-first interactive tool, and propose sparse-matrix LPP storage as an alternative to hash tables.

## 1 The Connectome

**Definition 1** (Connectome)**.** *Given a UM instance $u = (E, P, t, f)$ with event space $E$, pattern set $P$, thought vector $t$, and forward pass $f$, the* connectome $G(P)$ *is the directed graph with:*

- *Vertices: the events $e \in E$*
- *Edges: for each atomic pattern $p = (e_i, e_j, w)$ with $w > 0$, an edge $e_i \to e_j$*

The connectome captures the *potential* information flow of the model. An edge $e_i \to e_j$ means that support at $e_i$ *could* propagate to $e_j$. Whether it does, and in what order, is the business of $f$, not $P$.

**Definition 2** (ES-level Connectome)**.** *The* coarsened connectome $\bar{G}(P)$ *has event spaces as vertices:*

- *Vertices: the event spaces $\{\mathrm{ES}_1, \ldots, \mathrm{ES}_k\}$*
- *Edges: $\mathrm{ES}_a \to \mathrm{ES}_b$ if any pattern connects an event in $\mathrm{ES}_a$ to an event in $\mathrm{ES}_b$*

For KN-6, the coarsened connectome is:

$$\mathrm{ES}_{\mathrm{input}} \to \mathrm{ES}_{\mathrm{ctx}_1} \to \mathrm{ES}_{\mathrm{ctx}_2} \to \cdots \to \mathrm{ES}_{\mathrm{ctx}_6} \to \mathrm{ES}_{\mathrm{output}}$$

plus the LPP connections from each context ES to the output ES. This is a DAG—no cycles—and the topological order uniquely determines $f$ (up to parallelism within layers).

## 2 Deriving Layers

**Proposition 1** (Layer Derivation from DAG)**.** *If $\bar{G}(P)$ is a DAG, then a topological sort gives a valid execution order for $f$. All patterns within a single layer may be applied in parallel.*

This is the acyclic case: **when $P$ is a DAG, layers are fully determined by $P$.** No additional specification is needed.

### 2.1 Feedforward Networks

A feedforward network corresponds to a DAG connectome. Each "layer" in the traditional sense is one or more ESs at the same depth in the topological sort. The KN-6 tower is an example: input $\rightarrow$ context ESs $\rightarrow$ output.

### 2.2 Cycles: Where $P$ Becomes Ambiguous

A cycle in $\bar{G}(P)$ means the connectome is not a DAG.

**Definition 3** (Recurrence)**.** *A cycle in $\bar{G}(P)$ defines a* recurrence*. The ESs participating in the cycle form a* recurrent group*.*

The sat-rnn's hidden state corresponds to an ES with a self-loop: $\text{ES}_h \rightarrow \text{ES}_h$ via $W_h$. This is the simplest possible cycle. (The sat-rnn achieves 0.079 bpc by memorizing 1024 input bytes with 128 hidden neurons; the point of this experiment was understanding RNN dynamics after convergence, not general compression.)

**The critical point:** a cyclic $P$ does not determine $f$. The same pattern set $P$ (same events, same connections, same weights) is consistent with multiple forward passes:

- Apply the cycle once, twice, or $k$ times per input.
- Iterate to convergence (if it converges).
- Apply asynchronously with different ESs updating at different rates.
- Synchronize with an external clock (input stream frequency).

Each of these is a *different model* $u$ even though $E$ and $P$ are identical. The choice is part of $f$, which is part of the model specification.

**Remark 1** (The Runner Must Not Resolve Ambiguity)**.** *The UM runner's job, when encountering a cyclic connectome, is to* surface the ambiguity*: report the cycles, identify the recurrent groups, and require $f$ to specify how they are resolved. The runner should not silently pick a resolution strategy; that would be making architectural decisions that belong to the model designer.*

### 2.3 Input Stream Synchronization

The RNN's solution to the cycle ambiguity is elegant: synchronize the recurrent group with the input stream. At each input event (each byte in our case):

1. Read the input into $\text{ES}_{\text{input}}$.
2. Apply $f$ once through the entire connectome, including the cyclic patterns.
3. The thought vector $t$ persists to the next input event.

This pattern is observed biologically: sensory processing synchronizes neural oscillations with the input stream (auditory cortex locks to speech rhythm, visual cortex locks to saccade frequency). Cyclic neural circuits *need* an external clock to be well-defined.

### 2.4 Multi-Frequency $f$

Input stream synchronization is one clock. But data has multiple temporal scales: bytes, words, sentences, documents. Different recurrent groups synchronized to different clocks—a *multi-frequency $f$*—would let the model maintain state at multiple temporal scales simultaneously.

This connects to the brief FFT treatment in the um-arithmetic papers (20260216 archive), where the frequency decomposition of the input stream corresponds to different temporal resolutions of the forward pass. Multi-frequency $f$ is coming as a research direction; the connectome framework provides the structural foundation for specifying which groups run at which frequency.

### 2.5 Other Resolutions

Beyond single- and multi-frequency synchronization:

- **Fixed-depth unrolling**: apply cyclic patterns $k$ times per input (BPTT-$k$). A family of models parameterized by $k$.
- **Convergence**: iterate until $\|t^{(k+1)} - t^{(k)}\| < \epsilon$. Well-defined only if the dynamics are contractive. Our chaos results (20260207) show the sat-rnn does *not* converge.
- **Asynchronous**: events fire when their inputs change, with no global clock. Biologically plausible (spike timing) but computationally complex.

## 3 $f$-Patterns and $\omega$-Patterns

The distinction between structural patterns (weight $> 0$) and recording patterns (weight $= 0$) corresponds to the two components of the UM update:

- **$f$-patterns** (weight $> 0$): executed during the forward pass. Propagate support through the connectome.
- **$\omega$-patterns** (weight $= 0$, i.e. LPPs): observed during learning. After $f$ completes, $\omega$ examines the thought vector and updates recorded counts.

Two-phase structure per time step:

1. **Execute** ($f$): apply $f$-patterns to propagate support. The connectome's causal structure (with cycle resolution from $f$) determines order.
2. **Record** ($\omega$): examine the thought vector. For each $\omega$-pattern (LPP) between $\text{ES}_a$ and $\text{ES}_b$, record the joint event and update counts.

### 3.1 Sparse-Matrix LPP Storage

The UM runner currently stores LPPs as hash tables (FNV-1a hashed context keys with KN count slots). An alternative: sparse matrices storing log counts via log-stochastic increment, with each entry a triple $(e_a, e_b, \text{log-supp})$—two event indices and a log-support value.

Advantages:

- Explicit event pairs rather than hashed keys: no collisions, interpretable.
- Natural format for serialization and inspection: one triple per line.
- Sparse matrices can be memory-mapped, sorted, and merged.

- Compatible with the UM explorer's need to visualize individual LPP entries.

Disadvantages: potentially slower lookup than hash tables for large LPPs. But for the UM explorer (which needs random access to individual entries for visualization), the tradeoff may favor sparse matrices. Both implementations can coexist, with hash tables for batch scoring and sparse matrices for interactive exploration.

This remains to be worked out empirically once we have the UM runner viz in the browser.

# 4 How Context Events Change the Connectome

Adding a context event $e_c \in \mathrm{ES}_c$ adds new edges to the connectome. Adding a context ES to a DAG keeps it a DAG (assuming causal priority), so no new ambiguity is introduced and $f$ is still determined by topological sort.

**Proposition 2** (Context Events Add Depth)**.** *Adding a context ES to a DAG connectome increases the graph's depth by at most 1.*

If a context event *does* introduce a cycle (e.g., a context ES that depends on the output), then $f$ must be extended to resolve the new ambiguity—a form of self-attention or deliberation.

# 5 Composability of P-Programs

If P-program $A$ has connectome $G_A$ and P-program $B$ has connectome $G_B$, their composition has connectome $G_A \cup G_B$ plus interface edges. If composition introduces cycles, the runner surfaces the ambiguity.

# 6 The UM Runner's Execution Model

Given the connectome:

1. **Build** $\bar{G}(P)$ from all registered patterns and LPPs.
2. **Detect cycles**. If DAG, topological sort determines $f$. If cyclic, surface the ambiguity.
3. **Execute** $f$: apply $f$-patterns layer by layer.
4. **Record** $\omega$: update all $\omega$-patterns (LPPs) based on the final thought vector.

# 7 The UM Explorer

The connectome perspective enables a powerful interactive tool: the *UM explorer*, a web-based viewer for stepping through a model's execution on data.

## 7.1 Neuron-First Design

The explorer's primary view is **neuron-first**: each ES is visualized as a cluster of events (neurons), with support values shown as activation levels. This is the same visualization as the LPP viewer from the 20260216 archive, but applied to *every ES in the model*.

This gives a "brain-like" visualization: as models grow, you see more neurons, more connections, more activation patterns. Stepping forward through the data, you watch the whole model activate and respond—an intuition pump for understanding what the model is doing and why.

Design principles:

- **Per-ES LPP visualization**: each ES shows its events as circles or nodes, sized/colored by support level. LPP connections shown as weighted edges between ES clusters.
- **Oversupport and undersupport both red**: surprise of either kind is highlighted the same way. We don't prejudge what surprise "means"—only that the ES is in a problematic state.
- **Query by Boolean expression**: click on events to build Boolean queries. A popup affordance lets you compose AND/OR/NOT over events, and the explorer highlights all data positions matching the query.
- **Hypothetical testing**: manually set support values for specific events, re-run $f$, see how the prediction changes. Test context events before implementing them.

### 7.2 Scalability

The explorer will eventually handle large models with many ESs and events:

- Zoom and collapse: 3D visualization with ability to zoom into individual ESs or collapse distant ones.
- Low data volume: load perhaps 1024 input events at a time for performance. Larger queries hit the server.
- Multiple alternative approaches: start simple and iterate, keeping parallel design tracks for rapid discovery.

### 7.3 Acceleration of P-Programming

The explorer transforms P-programming from a write-run-analyze cycle into an interactive conversation with the model:

1. Score the data with the current model (runner mode).
2. Identify high-oversupport positions (surprise analysis).
3. Open the explorer at those positions. Step through byte by byte.
4. Query: "what context events are active here? What is the model predicting?"
5. Formulate a hypothesis: "the model needs a context event for [feature X]."
6. Test: set the hypothetical context event's support manually, re-run $f$, see if oversupport decreases.
7. Implement as a P-program.

## 8 The Connectome as a Theory of the Data

A model's connectome is its implicit theory of the data's causal structure. Each edge in $G(P)$ asserts a causal relationship. Each ES asserts that its events form a meaningful mutually-exclusive group. The connectome's topology—depth, cycles, connected components—reflects the model's understanding of how the data is generated.

Improving the model means improving this theory. Adding a context event means asserting a new causal relationship. Removing a dead pattern means retracting a false claim. The connectome is not just a computational graph; it is a *scientific theory* about the data, expressed in the language of events and patterns.

## 9 $P$ Is Knowledge; $f$ Is Reasoning

The connectome $G(P)$ encodes everything the model *knows*: which events exist, how they relate, what has been learned from data. The forward pass $f$ encodes how the model *reasons*: in what order it considers its knowledge, how it resolves ambiguity, how it synchronizes with the world.

Two models with the same $P$ but different $f$ have the same knowledge but different reasoning strategies. Much of neural architecture search is really a search over $f$ (layer ordering, skip connections, attention patterns) holding $P$'s capacity roughly constant.

## 10 $f$ and the Arrow of Time

In a DAG connectome, $f$ is determined: information flows in one direction. Time is implicit in the causal ordering. When cycles are present, $f$ must choose how time works: instantaneous convergence, per-input synchronization, or multi-frequency.

The multi-frequency direction (different recurrent groups at different rates) connects to the carrier signal concept: a persistent event spanning multiple input steps provides temporal context that a single-step pass cannot. The FFT decomposition of the input stream (um-arithmetic papers, 20260216) formalizes this: each frequency component corresponds to a temporal scale, and a multi-frequency $f$ processes them in parallel.

## 11 The Limit: When $P$ Fully Determines $f$

In the limit of a fully acyclic model, $P$ fully determines $f$. No ambiguity, no cycles, no architectural choices. Whether this limit is desirable is open. Biological neural networks are deeply cyclic; perhaps cycles are essential for temporal modeling. Or perhaps sufficiently rich acyclic connectomes (with enough context events) can capture any temporal structure without recurrence.

For enwik9, the acyclic KN-6 tower achieves 1.784 bpc. The cyclic sat-rnn achieves 0.079 bpc on 1024 input bytes via temporal recurrence (memorizing the training data with 128 hidden neurons). The gap between them reflects the value of cycles—or equivalently, the cost of the context events needed to replace them.

## 12 Conclusion

The relationship between $P$, $f$, and $\omega$ in the UM is:

- $P$ defines the connectome: a directed graph on $E$, encoding the model's knowledge.
- When $P$ is acyclic, $f$ is determined: topological sort gives layers. When cyclic, $f$ resolves ambiguity; the runner surfaces cycles.
- $f$-patterns propagate support; $\omega$-patterns record joint events. LPP storage may use hash tables (batch scoring) or sparse matrices (interactive exploration).
- Multi-frequency $f$ (different groups at different clocks) extends input-stream synchronization to multiple temporal scales.
- The UM explorer: neuron-first, per-ES LPP visualization, Boolean query by clicking, both surprise types red, hypothetical testing. Start simple, iterate.
- The connectome is a scientific theory of the data. Improving the model means improving the theory, one context event at a time.