Q1 Implementation Notes

Claude and MJC

11 February 2026

1 What We Compute

For each of the 1024 positions, we want to know which UM patterns are "active"—i.e. carrying signal in the backward attribution chain. This requires three passes.

1.1 Pass 1: Forward

Run the sat-rnn forward on all 1024 bytes. Store:

- $h_t \in \mathbb{R}^{128}$ for $t = 0, \ldots, 1023$ (needed for saturation gates and sign decisions)
- $P_t \in \mathbb{R}^{256}$ for each t (needed for the output gradient)

This is straightforward and already implemented in every tool we have. Memory: 1024×128 floats for h, 1024×256 for P. About 1.5 MB total.

1.2 Pass 2: Backward gradient

For each position t, compute:

1. The output gradient: $[g_t]_j = W_y[y, j] - \sum_o W_y[o, j]\, P_t(o)$ where $y = x_{t+1}$.
2. The backward gradient at each offset $d = 1, \ldots, D_{\max}$:

$$g_{t,d} = J_{t-d+1}^{\top} \cdot g_{t,d-1}$$

   where $J_s = \mathrm{diag}(1 - h_s^2) \cdot W_h$ and $g_{t,0} = g_t$.

   Expanding one step: $[g_{t,d}]_j = (1 - h_j(t{-}d{+}1)^2) \sum_k W_h[k, j]\, [g_{t,d-1}]_k$.

$D_{\max} = 50$ (the BPTT horizon). In practice we can stop early if $\|g_{t,d}\| < \epsilon_{\text{stop}}$, since the gradient is dying.

1.3 Pass 3: Per-pattern attribution

Given g_t and all $g_{t,d}$, we score each pattern.

W_y patterns. A W_y pattern is the pair (j, b): neuron j predicting byte b. Its attribution for position t (predicting $y = x_{t+1}$) is:

$$a_{W_y}(j) = |[g_t]_j|$$

This is the absolute gradient at the output layer for neuron j. Note: $[g_t]_j$ already encodes how much neuron j matters for predicting y specifically (it is the $W_y[y, j]$ column minus the P-weighted average). We do not need to enumerate over all output bytes b—the gradient has already done the marginalization. So there are 128 W_y attributions, not 128×256.

W_x patterns. A W_x pattern is the pair (b, j): input byte b activating neuron j. For position t and offset d, the input at position $t - d$ is the byte x_{t-d}. The attribution is:

$$a_{W_x}(j, d) = |W_x[j, x_{t-d}] \cdot [g_{t,d}]_j|$$

This is nonzero only for the specific byte x_{t-d} that actually occurred. Over all offsets $d = 1, \ldots, D_{\max}$, each W_x pattern (x_{t-d}, j) may appear at most once (at the unique d where that byte is the input). But different offsets may have the same byte, so the same pattern (b, j) could appear at multiple offsets. We take the maximum.

W_h patterns. A W_h pattern is the pair (j, k): neuron j at time s influencing neuron k at time $s + 1$. The attribution at offset d (where the W_h pattern connects step $t - d$ to $t - d + 1$) is:

$$a_{W_h}(j, k, d) = |(1 - h_j(t-d)^2) \cdot W_h[k, j] \cdot [g_{t,d}]_j|$$

Wait—this is not quite right. The backward gradient $g_{t,d}$ at offset d has already been propagated through all the Jacobians from t back to $t-d+1$. The W_h pattern (j, k) at step $s = t-d$ contributes to the propagation from step $t - d$ to $t - d + 1$. Its contribution is absorbed into $g_{t,d}$ via the matrix multiply $J_{t-d+1}^\top g_{t,d-1}$.

More precisely: the W_h pattern (j, k) at step s appears in $J_s = \text{diag}(1 - h_s^2) \cdot W_h$, specifically as the entry $J_s[k, j] = (1 - h_j(s)^2) \cdot W_h[k, j]$.

The attribution of this specific pattern instance is:

$$a_{W_h}(j, k, s) = |J_s[k, j] \cdot [g_{t,t-s}]_k|$$

That is: the Jacobian entry (pattern strength $\times$ gate) times the backward gradient that has arrived at neuron k at step $s + 1$ (which is offset $t - s - 1$ from the output, so $d' = t - s - 1$, meaning $g_{t,d'} = g_{t,t-s-1}$). Correction: $g_{t,d}$ is at offset d from position t, meaning it refers to position $t - d$. So at step s, the backward gradient is $g_{t,t-s}$.

Let us be more careful. Define $d = t - s$ as the offset of step s from position t. Then:

- $g_{t,d-1}$ is the backward gradient at step $s + 1 = t - d + 1$.
- $J_{s+1} = J_{t-d+1}$ propagates from step $s + 1$ to step s.
- The W_h pattern (j, k) at step $s + 1$ contributes $J_{t-d+1}[k, j] = (1 - h_j(t-d+1)^2) \cdot W_h[k, j]$ to the propagation.
- Its attribution is: $|(1 - h_j(t-d+1)^2) \cdot W_h[k, j] \cdot [g_{t,d-1}]_k|$

A W_h pattern (j, k) may be active at multiple offsets. We take the maximum over all offsets.

2 Counting Active Patterns

The UM has $\sim$3048 significant patterns (with weight above ϵ). For each position t, we compute attributions as above, then count how many patterns exceed threshold τ.

But note: the 3048 figure counts *time-independent* patterns. A W_h pattern (j, k) is one pattern regardless of how many timesteps it is active at. So the maximum possible count per position is ≤ 3048, even though a single W_h pattern could be active at 50 different offsets.

3 Implementation

The tool (`q1_sparsity.c`) does:

1. Load model weights from `sat_model.bin`.
2. Load data (first 1024 bytes of enwik9).
3. Forward pass: compute and store all h_t and P_t.
4. For each position $t = 0, \ldots, 1022$:
   (a) Compute g_t (128 mults + 256×128 for the P-weighted average).
   (b) Score the 128 W_y attributions.
   (c) For $d = 1, \ldots, D_{\max}$:
      - Propagate: $g_{t,d} = J_{t-d+1}^{\top} \cdot g_{t,d-1}$ (one mat-vec, 128^2 mults).
      - Score W_x attributions for byte x_{t-d} (128 mults).
      - Score W_h attributions (128×128 entries, but we only need those above ϵ).
   (d) Sweep thresholds, count active patterns.
5. Output: table of $(t, \tau, n_x, n_h, n_y, n)$.

3.1 Complexity

Per position: $D_{\max}$ Jacobian mat-vecs at 128^2 each, plus $D_{\max} \times 128^2$ W_h attribution scores. Total per position: $O(D_{\max} \cdot 128^2) \approx 800$K multiplies.

Over 1023 positions: $\approx$ 820M multiplies. At $\sim$1 GHz effective throughput, this is under 1 second. Memory is negligible.

4 Protocol B: Exact Computation via $E \to \mathbb{N}$

The f32 backward trace (Protocol A above) analyzes the RNN as it actually runs. But we can also compute pattern attributions exactly in the UM, using the prime power encoding from the event arithmetic paper.

4.1 Event assignment

Map each event $e \in E_{\text{sup}}$ to a distinct prime p_e. A pattern over events $(e_1, \ldots, e_k)$ maps to the product $\prod_i p_{e_i} \in \mathbb{N}$. Pattern composition is multiplication in $\mathbb{N}$. GMP (GNU Multiple Precision) gives exact arithmetic with no rounding.

4.2 The pattern space of u_{iso}

We can now precisely define the pattern space of the isomorphic UM. The elementary patterns are the nonzero weight entries:

- $P_x = \{(b, j, \pm) : W_x[j, b] \neq 0\}$ — input byte b excites/inhibits neuron j
- $P_h = \{(j, k, \pm) : W_h[k, j] \neq 0\}$ — neuron j excites/inhibits neuron k (one step later)

- $P_y = \{(j, b, \pm) : W_y[b, j] \neq 0\}$ — neuron j excites/inhibits output byte b

Each elementary pattern has strength $2|w|$ (the doubled-E factor). The full pattern space is the closure under composition:

$$P_{\text{iso}} = P_x \cup P_h \cup P_y \cup (P_x \cdot P_h^*) \cup (P_h^* \cdot P_y) \cup (P_x \cdot P_h^* \cdot P_y)$$

where $P_h^* = \bigcup_{n=0}^{\infty} P_h^n$ is the Kleene closure of recurrent patterns (chains of arbitrary length through hidden neurons), and $\cdot$ denotes pattern composition. In u_{sup} (with time), P_h^* is bounded by the BPTT horizon: $P_h^{\leq 49}$.

$P_{\text{iso}} \subset P_{\text{sup}}$: every pattern representable by the RNN is a pattern in the superset UM, but not vice versa. The architectural constraint is that every pattern in P_{iso} must factor through the 128 binary hidden ESs. A pattern in P_{sup} that correlates input at offset d_1 with input at offset d_2 exists as a data-countable pattern regardless of any model; it appears in P_{iso} only if the RNN can represent it as a chain through H. The 128-neuron bottleneck limits how many such correlations can be represented simultaneously—this is the capacity constraint that Q1 measures from the output side.

4.3 From exact to f32

Protocol A works in f32 because that is what the RNN computes in. Protocol B works in exact arithmetic because that is what the UM computes in (log-stochastic counting over $\mathbb{N}$). Comparing the two reveals where f32 precision destroys or creates patterns—i.e. where the RNN's representation diverges from the UM it is isomorphic to in theory.

4.4 The entropy coding identity

The UM's pattern strengths are log-frequencies: for a pattern p observed n_p times in N positions, the strength is $\omega(p) = \log(n_p/N)$. This is exactly the log-probability assigned by an entropy coder using the empirical distribution. This is not an analogy—the UM *is* the codebook of an arithmetic coder, and Hebbian learning (log-stochastic counting) *is* the update rule that builds it.

Slogan: Hebbian learning is all you need—given the right event spaces and a log representation of sensory inputs.

4.5 Prior over ω_{RNN}

Given a dataset D (the memory trace), the UM's pattern inventory in u_{sup} defines a prior over which patterns should appear in any model trained on D. Specifically, for each pattern p in u_{sup}, the data determines $\omega(p) = \log(n_p/N)$. Patterns with large $|\omega(p)|$ are the ones the model should learn; patterns with $\omega(p) \approx 0$ are noise.

This gives a prior over ω_{RNN}: the prediction that the RNN's active patterns (as measured by Protocol A) will be precisely those with large $|\omega(p)|$ in the data (as measured by Protocol B). We can then predict the RNN's learning behavior: which patterns it learns first (largest $|\omega|$), which it never learns (below the f32 noise floor or the 128-neuron capacity), and the order in which patterns appear during training.

4.6 Experimental comparison

For Q1, this means two parallel measurements:

1. **Protocol A** (f32): How many patterns are active in the backward trace of the trained RNN?
2. **Protocol B** (exact): How many patterns have $|\omega(p)| > \tau$ in u_{sup} computed from data?

If the counts match and the *same* patterns are active, the RNN has learned exactly what the data prescribes. If Protocol A finds fewer, the RNN's capacity (128 neurons, f32, BPTT-50) is the bottleneck. If Protocol A finds patterns absent from Protocol B, the RNN has hallucinated structure.