Pattern Priors and Skip-Patterns: From Backward Trie to Attention

Claude and MJC

8 February 2026

Abstract

We have a prior over the patterns that a model can learn, determined by its architecture: the sat-rnn's BPTT-50 training and 128-neuron bottleneck impose an exponential falloff in learnable pattern length. Symmetrically, we designed the SN representation and know what shapes it can express. We show that the backward trie (built from output to input) decomposes prediction into *atomic patterns* at each offset, which compose into *skip-2-grams* when we combine offsets. At DSS = 1024, approximately 80% of skip-2-gram patterns are artifacts of the specific dataset; the surviving 20% carry 60–71% of occurrences and correlate at $r \approx 0.85$ between halves. Tracing these patterns through the sat-rnn reveals that the *same neurons* (h52, h8, h68) carry skip information regardless of skip distance, with a dominant W_h pathway h8→h52 (weight +1.28). We then demonstrate that RNN weights can be *constructed* directly from data patterns without gradient-based training: using 8 greedy skip offsets with a hash-based readout achieves 0.137 bpc (with MLP readout) on 1024 bytes, generalizing better than the trained sat-rnn on unseen data (5.6 vs 8.2 bpc). The gap between the construction (0.137 bpc) and the UM information-theoretic floor (0.043 bpc) quantifies the *readout loss*; an MLP readout recovers one-third of it.

1 Priors Over Patterns

1.1 The SN Prior

We built the SN intentionally: pattern strengths are integers in $[0, 255]$, the event space is a designed binary ES with softmax between layers, and the pattern inventory is a sparse subset of $I^k \times O$. Because we designed this macrostate, we have a prior over what patterns the SN can express—and equally important, what it cannot. The SN is a designed dynamics, where the pattern strengths play the role of energy levels in the physics analogy: higher strength = more support = lower luck = more probable.

1.2 The RNN Prior

The sat-rnn has architectural constraints that bound its learnable patterns:

1. **BPTT-50**: Gradients are truncated at 50 timesteps. No pattern longer than 50 bytes can be learned through gradient flow. This is a hard cutoff, not a soft decay.
2. **128-neuron bottleneck**: All temporal patterns must be compressed into W_h's $128 \times 128 = 16{,}384$ float weights. The pattern-chain UM uses 6,180 explicit data-term patterns to achieve 0.067 bpc; the RNN must encode equivalent information in fewer parameters, but with the advantage of factoring through hidden neurons.

3. **Exponential falloff**: Within the 50-step horizon, longer patterns are harder to learn (vanishing gradients, competition for hidden-state capacity). We expect an approximately exponential distribution over pattern lengths: many short patterns, exponentially fewer long ones.

The RNN prior is therefore: an exponential distribution over pattern lengths, with a hard cutoff at 50, compressed through 128 neurons.

1.3 Design Implications

Because we know both priors, we can design the UM's pattern inventory to match. The pattern-chain UM (pattern-chain.pdf) already shows that orders 1–12 suffice to surpass the sat-rnn. The question is: what is the *shape* of the optimal pattern distribution? The backward trie provides the answer.

2 The Backward Trie

The forward trie asks: "given context, what follows?" The backward trie inverts this: "given an output byte y, what input patterns predict it?"

For each output byte y, we collect all positions t where $y = x_{t+1}$ and examine the preceding bytes $x_t, x_{t-1}, \ldots$ These form the *support set* of y—the input events that provide evidence for predicting y.

The backward trie yields not only contiguous n-grams (the pattern-chain UM's inventory) but also *skip-patterns*: pairs of input bytes at non-adjacent offsets that jointly predict the output. These are precisely the patterns that require W_h to carry information across time steps.

2.1 Atomic Patterns (Depth 1)

At depth 1, the backward trie gives the bigram structure from the output's perspective. Each (input, output) pair is an atomic pattern—it cannot be decomposed further.

Output	Count	#Inputs	H(in|out)	Top inputs
`' '`	127	8	1.64	`' ':83, '.':19, 'e':14`
`'e'`	107	15	3.06	`'m':31, 'c':26, 'k':13`
`'a'`	89	14	2.78	`'n':28, 'p':25, 'i':13`
`'i'`	48	11	2.82	`'d':12, 'k':12, 'W':6`
`'s'`	44	9	2.15	`'e':26, 'a':4, 'n':3`
`0x0A`	19	1	0.00	`'>':19`

Table 1: Atomic patterns for the most common outputs. Newline (0x0A) is perfectly predicted by a single input (`'>'`): $H = 0$ bits. Space is predicted by only 8 distinct inputs, dominated by space itself (65%). Total: 151 atomic patterns.

The conditional entropy H(input|output) measures how "focused" each output's support set is. Low entropy means a few dominant atomic patterns; high entropy means many inputs contribute.

2.2 Context Growth

As we increase the backward trie depth, the number of distinct contexts grows:

Output	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$
`' '`	8	14	20	29	36	43
`'e'`	15	25	28	36	38	43
`'a'`	14	23	26	33	37	39
`'p'`	7	7	7	10	10	14

Table 2: Distinct contexts per output at each depth. Space grows from 8 atomic inputs to 43 depth-6 contexts. The letter `'p'` has a plateau (7 contexts at depths 1–3), meaning its prediction is dominated by a few fixed patterns.

3 Skip-2-Grams from the Backward Trie

The backward trie at depth 1 gives atomic patterns at each offset. A *skip-2-gram* combines two atomic patterns at different offsets: $(x_a@\text{offset}_a,\ x_b@\text{offset}_b) \to y$. This is the simplest skip-pattern: two input observations, separated by a gap, jointly predicting the output.

The backward trie produces these directly: for a given output y, we examine which input bytes appear at offset a and which appear at offset b, and count how often each (x_a, x_b, y) triple co-occurs. The contiguous n-grams from the pattern-chain UM are the special case where offsets are consecutive ($a = 1, b = 2$); skip-2-grams with non-adjacent offsets require the hidden recurrence W_h to bridge the gap.

3.1 Offset Pair Analysis

For each offset pair (a, b) with $a < b$, we compute the conditional entropy $H(Y \mid X_a, X_b)$ and the improvement over the better single-offset predictor:

a	b	$H(Y\|X_a, X_b)$	Improvement	#Patterns
1	2	0.556	1.487	340
1	4	0.511	1.532	421
1	8	0.495	1.547	510
2	6	0.623	1.667	477
2	11	0.570	1.719	557
3	12	0.658	1.887	581
6	15	0.837	2.082	624

Table 3: Skip-2-gram analysis at DSS = 1024. Offset (1,2) is the trigram case; wider pairs show more improvement over single offsets (because the inputs are more independent) but produce more patterns.

A skip-2-gram backoff predictor using offset $(1, 2)$ achieves 0.82 bpc, down from the bigram's 2.04 bpc—a factor of 2.5 improvement from a single additional input position.

3.2 Survival Under DSS Doubling

With DSS = 1024, the RNN trains on too little data for most patterns to generalize. We test this directly by comparing patterns in the first 1024 bytes against the second 1024 bytes:

Offset pair	Survive	Artifact	Survival %	Count r
(1, 2)	80	260	23.5%	0.880
(1, 4)	82	339	19.5%	0.814
(1, 8)	90	420	17.6%	0.833
(2, 6)	81	396	17.0%	0.819
(3, 12)	96	485	16.5%	0.859

Table 4: Pattern survival when DSS doubles from 1024 to 2048 bytes. "Survive" = pattern appears in both halves. "Artifact" = first half only. Survival drops with wider skips (more combinatorial, more data-specific). But surviving patterns' counts correlate at $r > 0.8$—strong stability.

The surviving 20% of patterns carry 60–71% of occurrences: high-count patterns survive because they reflect structural regularities (XML tags, common words) rather than accidents of the specific 1024 bytes. Examples:

- **Surviving** (structural): `' '' '`→`' '` (spaces), `'m''a'`→`'e'` ("name"-like), `'.''>'`→`' '` (XML tag close).
- **Artifact** (first half only): `'k''i'`→`'i'`, `'t''h'`→`'t'`—words that happen not to recur.
- **New** (second half only): `'d''i'`→`'>'`, `'<'' '`→`'/'`—XML structures absent from first 1024 bytes.

This gives a concrete prediction: when the RNN retrains on 2048 bytes, W_h entries encoding surviving patterns should strengthen, while entries encoding artifacts should weaken or be repurposed.

4 Skip-k-Grams: Greedy Offset Selection

Skip-2-grams use two offsets. We generalize to skip-k-grams by greedily adding the offset that most reduces $H(Y \mid X_{\text{offsets}})$. Starting from offset 1 (the strongest single predictor), we add offsets one at a time:

k	Offsets	bpc	#Patterns	Contiguous-k bpc
1	[1]	2.042	151	2.042
2	[1, 8]	0.495	510	0.556
3	[1, 8, 20]	0.148	700	0.187
4	[1, 8, 20, 3]	0.069	712	0.114
8	[1, 8, 20, 3, 27, 2, 12, 7]	0.043	834	—

Table 5: Greedy skip-k-gram results at DSS = 1024. Skip-4 with offsets $[1, 8, 20, 3]$ achieves 0.069 bpc with 712 patterns—nearly matching contiguous order-12 (0.067 bpc, 6180 patterns), a 9× compression of the pattern inventory. Offset 8 is chosen before offset 2 because it provides more *complementary* MI: offset 2 is correlated with offset 1 (adjacent bytes), while offset 8 captures independent structure.

At DSS = 2048, the greedy offsets change to $[1, 4, 14, 29]$—the dominant patterns shift, and the optimal skip structure adapts. This confirms that offset selection is data-dependent, not architectural.

5 Skip-Patterns in the RNN Hidden State

We trace the skip-2-gram patterns through the sat-rnn by running the forward pass on the full dataset and recording the hidden state $h_t \in [-1, 1]^{128}$ at every position.

5.1 Neuron Universality

The most striking finding: the *same neurons* carry skip information regardless of skip distance. We score each neuron by $\Delta_j \times |C_j|$, where Δ_j is the average absolute change in h_j when the skip-pattern's earlier input is processed, and $C_j = W_y[y, j] \cdot h_j[t]$ is the neuron's contribution to the correct output via W_y.

Neuron	Avg $\|\Delta h\|$	Avg W_y contrib	Score	Role
h52	0.930	+0.580	0.540	highway
h8	1.006	+0.482	0.485	input gate
h68	0.684	+0.582	0.398	output amp
h61	0.681	+0.438	0.298	output amp
h15	0.706	+0.412	0.291	output amp
h73	0.881	+0.279	0.246	relay
h59	0.778	+0.313	0.244	relay

Table 6: Top neurons by information flow score, averaged over all offset pairs (1,2), (1,4), (1,8). Scores are nearly identical across offsets—the RNN uses a universal information highway.

The neurons separate into roles:

- **Input-responsive**: h3, h97, h20, h117 change most when any input byte is processed ($|\Delta h| > 1.0$).
- **Output-contributive**: h68, h52, h15, h61 contribute most to correct predictions via W_y.
- **Highway**: h52 and h8 score highest on the product—they both respond to inputs and contribute to outputs.

These are different sets. Information flows: input $\to$ Wx $\to$ input-responsive neurons $\to W_h \to$ output-contributive neurons $\to W_y \to$ output. The skip-pattern is compressed into this pipeline.

5.2 The W_h Highway

The top W_h connections reveal how information persists:

The spectral norm of W_h is $\sigma_1 = 5.5$, well above 1. Combined with tanh saturation, this means the hidden state constantly amplifies and clips—information does not decay, it is reshaped at every step. This explains both the chaotic sensitivity observed in the export-gap analysis and the fact that skip-patterns of different lengths use the same neurons: the information highway operates at saturation regardless of how many steps it has been propagating.

Connection	Weight	Interpretation
h8 $\rightarrow$ h52	$+1.28$	main highway
h8 $\rightarrow$ h8	-1.25	self-inhibit (fire once)
h8 $\rightarrow$ h90	$+1.13$	fanout
h20 $\rightarrow$ h97	$+1.01$	secondary highway
h68 $\rightarrow$ h99	$+0.99$	output relay
h50 $\rightarrow$ h76	$+1.39$	strongest overall

Table 7: Top W_h connections. The h8$\rightarrow$h52 pathway dominates: h8 responds to inputs (high Δ), self-inhibits (fires for one step), and feeds h52 which has the highest combined score. This single pathway carries skip-pattern information across arbitrary gaps within its persistence window.

6 Connection to Attention

The backward trie computes, for each output, which inputs at which offsets provide how much information. This is *exactly* what attention does:

1. **Query**: the output position (which byte are we predicting?).
2. **Keys**: input bytes at each offset (what context is available?).
3. **Values**: the information each key provides (MI or pattern strength).
4. **Attention weights**: how much to attend to each offset, determined by the backward trie's conditional probabilities.

The difference: the backward trie is static and exhaustive (computed from the full dataset), while attention learns to compute these weights at runtime from learned embeddings. Attention includes:

- An **embedding onto a prior**: the position encoding and initial query distribution give the model a prior over which offsets matter before seeing any data. This is analogous to our order-0 marginal (the global prior).
- **Runtime selection**: instead of storing all skip-patterns explicitly, attention computes which pattern to apply based on the current input. This is exponentially more efficient for long-range patterns.

The RNN's recurrent weights W_h are a compressed version of this: they implement a fixed attention pattern (determined by training) that applies the same "which offsets matter" weights at every timestep. The RNN cannot dynamically adjust its attention—it is a static backward trie, compressed into 128 neurons. Our neuron analysis shows this compression is extreme: essentially 2–3 W_h pathways (h8$\rightarrow$h52, h20$\rightarrow$h97) carry all temporal information, regardless of skip distance.

7 Constructing RNN Weights from Data Patterns

The preceding sections analyzed the sat-rnn's learned representation. We now ask the converse: can we *construct* RNN weights directly from data patterns, bypassing gradient-based training entirely? This is the "write-back" direction of reverse isomorphism—writing UM patterns into RNN weights.

7.1 Bigram Construction

As a baseline, we construct an RNN that implements bigram conditional distributions $P(y \mid x)$.

Architecture. The input-to-hidden weights W_x are set to large random values (scale = 10, seed 42), ensuring tanh saturation: each input byte maps to a fixed binary hash $h \in \{-1, +1\}^{128}$. The recurrent weights $W_h = 0$ (no memory). For the readout layer W_y, we solve via least squares: given the 256×128 hash matrix H and interpolated target distributions $P_\lambda(y|x) = 0.95 \cdot P_{\text{bigram}}(y|x) + 0.05 \cdot P_{\text{unigram}}(y)$, we solve $(H^T H + 0.01I)W_y^T = H^T L$ where L contains the log-probability targets.

Result. The constructed RNN achieves **2.10 bpc** on 1024 bytes, matching the counting bigram baseline (2.05 bpc). Cross-evaluation on unseen data: 4.45 bpc (1024-trained) vs 2.22 bpc (2048-trained)—the constructed model generalizes in proportion to its training data, as expected for a bigram model.

7.2 Multi-Offset Construction

We extend this to multi-byte context. For each of k offsets, we allocate $128/k$ neurons to hash the input byte at that offset. The hidden vector h_t is constructed by concatenating the binary hashes:

$$h_t = [\text{hash}(x_{t-o_1+1}),\ \text{hash}(x_{t-o_2+1}),\ \ldots,\ \text{hash}(x_{t-o_k+1})]$$

where $o_1, \ldots, o_k$ are the offsets and each hash uses $128/k$ neurons. The readout W_y is optimized via gradient descent on cross-entropy (1000 epochs, W_x and W_h frozen). For contiguous offsets, W_h implements a shift register; for non-contiguous offsets, h is constructed directly from data.

k	Offsets	Neurons/off	Train bpc	Test bpc	UM floor
1	[1]	128	2.054	5.02	2.042
2	[1, 2]	64	0.702	5.52	0.556
2	[1, 8]	64	0.895	5.89	0.495
4	[1, 2, 3, 4]	32	0.352	5.13	0.114
4	[1, 8, 20, 3]	32	0.267	5.70	0.069
8	[1..8]	16	0.228	5.47	—
8	[1,8,20,3,27,2,12,7]	16	**0.190**	5.59	0.043
Trained sat-rnn (BPTT-50)			0.079	8.22	—

Table 8: Multi-offset construction at DSS = 1024. Train bpc after 1000 epochs of W_y-only optimization; test bpc on unseen second 1024 bytes; UM floor is the information-theoretic conditional entropy $H(Y|X_{\text{offsets}})$ from the skip-k-gram analysis. At $k \geq 4$, greedy offsets beat contiguous by 17–24%. All constructed models generalize better than the trained sat-rnn (5.0–5.9 vs 8.2 bpc on unseen data).

7.3 Analysis

Four findings emerge from the construction experiments:

1. **Greedy offsets improve construction.** At $k = 4$, greedy offsets [1,8,20,3] achieve 0.267 bpc vs contiguous [1..4] at 0.352—a 24% improvement. At $k = 8$, greedy achieves 0.190 vs contiguous 0.228 (17% improvement). The greedy advantage grows with k because distant offsets contribute more independent information than adjacent ones.

2. **Readout loss.** The gap between the construction's bpc and the UM information-theoretic floor quantifies the *readout loss*: how much the linear softmax readout fails to recover from the hash features. For 1 offset: gap = 0.012 (nearly optimal). For greedy-8: gap = 0.147. The readout loss comes from two sources: (a) the random hash introduces collisions ($128/k$ neurons per offset), and (b) the softmax is a linear classifier on features that encode nonlinear interactions between offsets.

3. **Construction beats training for generalization.** Every constructed model generalizes better than the trained sat-rnn on unseen data (5.0–5.9 vs 8.2 bpc). The sat-rnn's BPTT-50 training drives it deep into memorization (0.079 bpc train) at the cost of generalization. The constructed model, with frozen hash functions, cannot overfit the temporal structure—only the readout is trained.

4. **Remaining gap to sat-rnn.** The best linear construction (0.190 bpc) is still 2.4× the sat-rnn's 0.079 bpc. An MLP readout reduces this to 0.137 bpc (1.7×). See §7.4.

7.4 MLP Readout

The readout loss (0.147 bpc for greedy-8) is partly due to the softmax's linearity. We replace the linear readout with a 2-layer MLP ($128 \to d \to 256$ with ReLU), keeping the hash features frozen:

Offsets	Readout	d	Train bpc	Test bpc	Readout loss
$[1, 8, 20, 3, 27, 2, 12, 7]$	Linear	—	0.190	5.59	0.147
$[1, 8, 20, 3, 27, 2, 12, 7]$	MLP	128	0.158	5.07	0.115
$[1, 8, 20, 3, 27, 2, 12, 7]$	MLP	256	**0.137**	**4.92**	0.094
$[1, 8, 20, 3]$	Linear	—	0.267	5.70	0.198
$[1, 8, 20, 3]$	MLP	256	0.168	5.01	0.099
Trained sat-rnn (BPTT-50)			0.079	8.22	—

Table 9: MLP readout reduces the readout loss by 36% for greedy-8 (0.147→0.094) and 50% for greedy-4 (0.198→0.099). The best constructed model (greedy-8, MLP-256) reaches 0.137 bpc—1.7× the sat-rnn but still generalizing better on unseen data (4.92 vs 8.22 bpc).

The MLP recovers about one-third of the readout loss. The remaining gap (0.094 bpc above the UM floor) has two sources: (a) hash collisions at 16 neurons per offset (2^{16} states for 256 values provides margin, but the random projection is not information-theoretically optimal), and (b) the MLP sees each offset's hash independently—it cannot compute *joint* features across offsets the way the UM's counting can. The sat-rnn avoids both limitations by using all 128 neurons as a single entangled representation.

8 Implications

1. The backward trie provides the **ground truth attention map** for this dataset: which inputs at which offsets predict which outputs, with exact information-theoretic weights.

2. The backward trie directly produces skip-2-grams (and higher-order skip-k-grams by combining more offsets). These are the patterns that W_h must carry across time.

3. The RNN's W_h compresses all skip-patterns into $\sim$3 dominant pathways, used universally for all skip distances. The spectral norm $\sigma_1 = 5.5$ combined with tanh saturation explains both chaotic sensitivity and offset-invariant information flow.

4. At DSS = 1024, $\sim$80% of skip-2-gram patterns are artifacts. The survival rate under DSS doubling (17–24%, with count $r > 0.8$) gives a concrete bound on useful capacity: the RNN wastes most of its representational budget on patterns that will not generalize.

5. **DSS doubling test**: We trained matched RNNs on 1024 and 2048 bytes (same architecture, same sat_train procedure, 4000 epochs).

   - The 1024-trained model achieves 0.087 bpc on its training data but **8.80 bpc** on the second 1024 bytes—total failure.
   - The 2048-trained model achieves 0.17 bpc on the first half and 0.23 bpc on the second half—it generalizes.
   - W_h correlation between models: $r = 0.06$ (effectively zero). Different random seeds converge to different neuron assignments. The highways we identified (h8→h52, h20→h97) are artifacts of the specific initialization, not structural features.
   - Spectral norm *increased* from 5.5 to 6.0 with more data.

   The lesson: individual W_h weights are not comparable across random seeds due to neuron permutation symmetry. The right abstraction is *patterns*, not weights. The survival analysis from the skip-2-gram data operates at this level and correctly identifies which patterns generalize. The UM resolves this: patterns are first-class, and the arbitrary neuron assignment disappears.

6. **Construction decomposes the RNN's job.** The construction experiment separates the RNN into three independent problems: (a) *offset selection* (which past positions to attend to—solved by the greedy skip-k-gram analysis); (b) *encoding* (how to carry offset information in W_h—the shift register is one solution); (c) *readout* (how to map the encoded state to output probabilities—solved by W_y optimization). The trained sat-rnn solves all three jointly via BPTT-50, entangling the solutions. The construction solves them independently, achieving better generalization at the cost of worse training bpc.

7. **Transformers succeed** because they implement the backward trie dynamically: for each output, they learn to attend to the informative offsets, with capacity that scales with context length rather than being bottlenecked through a fixed hidden state.

Reproducibility

Repository: `https://github.com/inimino/hutter` (commit: `f808449`)

Model: Saturated RNN, 256→128→256, tanh, BPTT-50. Checkpoint: `sat_model.bin` (329 KB). Performance: 0.079 bpc on 1024 bytes.

Data: First 1024 bytes of enwik9 (`http://mattmahoney.net/dc/enwik9.zip`).

To reproduce:

```
git clone https://github.com/inimino/hutter.git
cd hutter && git checkout f808449
bash reproduce.sh
```

Source files: `backward_trie.c`, `skip2gram.c`, `skip_kgram.c`, `skip2_survival.c`, `skip2_rnn.c`, `compare_wh.c`, `construct_rnn.c`, `construct_skip.c`, `construct_skip_greedy.c`, `construct_skip_mlp.c`.