Context Events and the Induction of Priors

Claude and MJC

18 February 2026

Abstract

We formalize the concept of a *context event*: an atomic event in a causally prior event space that, through the UM forward pass, induces a prior distribution on downstream event spaces. We conjecture that every missing prior in P-programming corresponds to a missing context event. This provides a unified framework for understanding KN smoothing, neural reuse, and the gap between counting-based and optimal compression. We describe a methodology for discovering context events and translating them into P-programs.

1 Introduction

In the UM framework, compression proceeds by learning joint event distributions across event spaces via Learned Probabilistic Patterns (LPPs). The KN-6 model achieves 1.784 bpc on enwik9 using a tower of byte-context event spaces connected by LPPs with KN smoothing.

Yet significant compression remains on the table. The gap between KN-6 (1.784 bpc) and the current best compressed size arises because KN smoothing uses a crude sliding window as context, discarding most sequential information.

We argue that this gap is precisely characterized by *missing context events*: atomic events that, if present in the model, would condition the downstream LPPs to produce tighter output distributions.

2 Definitions

Definition 1 (Event Space)**.** *An event space* ES_a *is a named set of atomic events* $\{e_1, \ldots, e_n\}$ *with a support function* $s : \mathrm{ES}_a \to [0, 255]$ *assigning each event a current support value.*

Definition 2 (Context Event)**.** *Given a target LPP from* ES_a *to* ES_b*, a* context event $e_c \in \mathrm{ES}_c$ *is an atomic event in a causally prior event space such that the LPP, conditioned on* e_c *having nonzero support, produces a more accurate output distribution on* ES_b*.*

More precisely: let $P(b \mid a)$ denote the distribution induced by the LPP alone, and $P(b \mid a, e_c)$ the distribution when the context event e_c is active. Then e_c is a context event if

$$H[Y \mid X, e_c] < H[Y \mid X]$$

where X ranges over ES_a and Y over ES_b.

Definition 3 (Prior Induction)**.** *When a context event* e_c *has support* $s(e_c) = k > 0$*, and* e_c *is connected to events in* ES_a *via atomic patterns with weights* $w_{c,i}$*, the UM forward pass induces a prior on* ES_a*:*

$$s(e_i) \leftarrow \max\big(s(e_i),\ \min(k, w_{c,i})\big)$$

This is the standard max-min update. The pattern of weights $\{w_{c,i}\}$ *characterizes which events in* ES_a *are "expected" under context* e_c*, and with what strength.*

3 The Conjecture

Conjecture 1 (Missing Prior = Missing Context Event)**.** *In P-programming, whenever a model exhibits a systematic bias—a prior that should be present but isn't—there exists a context event e_c in some event space ES_c such that adding e_c and its connections to the model corrects the bias.*

This is a constructive claim: not merely that better models exist, but that the improvement always takes the specific form of adding a context event. The UM's universality guarantees that such an event exists; the conjecture asserts it is always atomic (a single event in a single ES), not a complex construction.

3.1 Why atomic?

The max-min forward pass means that a single well-placed context event can restrict a downstream distribution just as effectively as a complex gating mechanism. If e_c has support 200 and connects to events $\{e_1, e_3, e_7\}$ in ES_a with weight 200, while not connecting to $\{e_2, e_4, e_5, e_6, e_8\}$, then under the forward pass only $\{e_1, e_3, e_7\}$ receive induced support from e_c. This is Boolean selection—exactly what "context" means.

4 KN Smoothing as Crude Context

In KN-6, the context for predicting byte t is the preceding k bytes $(b_{t-k}, \ldots, b_{t-1})$. This context is represented as an event in $\mathrm{ES}_{\mathrm{context}_k}$ via hashing.

The KN interpolation chain (order $6 \to 5 \to \cdots \to 1 \to$ unigram) is a tower of progressively coarser contexts. Each drop in order discards one byte from the left of the window. In our framework:

- Each order k defines a context event space ES_k with events indexed by k-grams.
- The interpolation from order k to order $k-1$ is a quotient: the finer context is projected to the coarser one by dropping the leftmost byte.
- The KN discount D compensates for the fact that the finer context has less support (fewer observations).

What KN gets right: the sliding window captures local sequential dependencies.

What KN misses: everything outside the window. The context event for "we are inside an HTML tag" requires information from potentially hundreds of bytes earlier. KN-6 cannot represent this.

This is precisely a missing context event. An event $e_{\mathrm{in\_tag}} \in \mathrm{ES}_{\mathrm{structure}}$ that has support whenever we are inside $\langle \cdots \rangle$ would dramatically sharpen the byte distribution (only tag-interior characters are expected).

5 Neural Reuse via Context Events

A central problem in compression: the same local patterns (e.g., "th" $\to$ "e") occur in many different contexts (normal text, tags, attributes, URLs). A single LPP captures the average, losing the context-specific distribution.

Context events solve this: the LPP between byte context and byte output remains the same, but different context events in a prior ES activate different subsets of the LPP's patterns.

Example 1. *Consider the bigram LPP from 1-byte context to output. In KN-6, $P(\text{'e'} \mid \text{'h'})$ is a single number. But with context events:*

- $e_{\text{word\_start}}$ *active: "h" is likely "he", "his", "had"*
- $e_{\text{in\_tag}}$ *active: "h" is likely "href", "html"*
- $e_{\text{after\_digit}}$ *active: "h" is likely a hex digit*

Each context event induces a different prior over the output ES, restricting the LPP to its relevant joint events.

6 The Boolean Interpretation

Every context event has a Boolean reading. The max-min forward pass is:

$$s(y) = \max_{\text{patterns}} \min(s(x_1), s(x_2), \ldots, s(x_n), w)$$

which, when all supports are binary ($\{0, 255\}$), reduces to:

$$y = \bigvee_{\text{patterns}} (x_1 \wedge x_2 \wedge \cdots \wedge x_n)$$

This is disjunctive normal form (DNF). Each pattern is a conjunction (AND of its inputs); the max aggregates disjunctively (OR over patterns).

A context event adds one more term to each conjunction:

$$y = \bigvee (c \wedge x_1 \wedge x_2 \wedge \cdots \wedge x_n)$$

where c is the context. This is the probabilistic syllogism:

> When c AND x_1 AND $\cdots$ AND x_n are all supported (each with at least support k), then y is supported (at support k).

7 The Methodology

We propose a systematic procedure for identifying and implementing context events:

1. **Identify the residual.** Given a target LPP, compute the gap between the model's prediction and the empirical distribution. Where is the model systematically wrong?
2. **Boolean interpretation.** Express the residual in Boolean terms: "the model predicts x but the data shows y whenever [some condition holds]."
3. **Natural language.** State the condition in natural language: "we are inside an HTML tag", "the current word is a number", "we just saw a capital letter after a period".
4. **Reify via E/N/Q.** Express the condition as an event in the integer representation (um-arithmetic-v4): identify the quotient space, the projection, the relevant equivalence class.
5. **Query the dataset.** Find concrete examples in the data where the condition holds and verify that it explains the residual.
6. **P-program.** Implement the context event as an ES with events detected by a P-program (a nested UM instance). Connect it to the target LPP's input ES.

8 Concrete Example: Word Length

Our factor map analysis (20260209 archive) showed that *word length* is the single most predictive feature of the hidden state across all 128 neurons of the trained RNN ($R^2 = 0.84$).

In context-event terms:

- $ES_{\text{word\_len}}$ contains events $\{w_0, w_1, \ldots, w_{20}\}$ for current word length.
- A P-program detects word length: reset counter on space/punctuation, increment on letters.
- This ES connects to the byte-output LPP as a context event.
- The prior induced: at w_0 (word boundary), space and short-word starters are likely; at w_5, mid-word continuations dominate.

The factor map showed that adding word length explains 91% of the sat-rnn's compression gain (0.50 bpc vs. 0.079 bpc floor). This is the most powerful single context event we have identified.

9 Open Issues for P-Programming

Several practical issues arise when implementing context events:

9.1 Discovery

How do we systematically discover which context events to add? The residual analysis (Step 1) is easy in principle but requires tooling in the UM runner. We need:

- Per-position surprise measurements (which bytes are poorly predicted)
- Grouping of high-surprise positions by shared features
- Automated hypothesis generation: "positions where surprise $> k$ share feature f"

9.2 Representation

Context events detected by P-programs require nested UM instances. The current umr infrastructure supports LPPs and simple ESs. We need:

- A way to define a P-program that reads the input stream and emits context events
- Interface events connecting the P-program's output to the main model's context ES
- Incremental update: the P-program must run online, one byte at a time

9.3 Surprise Mechanism

When a context event is *absent* (no event in ES_c has support), the downstream LPP operates without any prior—exactly the current KN situation. When a context event is *wrong* (the active context event doesn't match the actual condition), the induced prior hurts rather than helps. This is *oversupport*—a form of surprise.

We need machinery to detect and respond to both:

- **Undersupport**: no context event fires → the model is uncertain about the context
- **Oversupport**: the wrong context event fires → the model's prior is wrong

Both are signals for learning: undersupport means we need a new context event; oversupport means an existing one needs refinement.

9.4 The Carrier Signal

Memory traces can be stored "inline" via a carrier signal: an event representing not an instant but a span of nonzero duration λ. The joint event $(e., t_i, e., t_{i+1})$ encodes temporal adjacency and can be recorded by an LPP. This converts sequential memory into associative memory within the UM framework.

However, this is the most P-programming-intensive part of the context events story, and we defer detailed treatment to a future paper, noting it as a key open problem.

10 Connection to the KN-Quotient Tower

The kn-quotient papers (20260216 archive) established that KN smoothing has an algebraic structure: contexts are elements of $\mathbb{Z}/256^k\mathbb{Z}$, order reduction is a ring surjection, and the discount D corresponds approximately to removing common evidence via GCD.

Context events extend this tower upward. Above the byte-level tower (orders 1–6), we add semantic event spaces: word boundaries, syntactic structure, document-level features. Each new ES adds a "floor" to the tower, and the context events from that floor condition the LPPs below.

The key insight from kn-quotient-v2 is that the KN sliding window is *already* a context event mechanism—just a very crude one. The k-gram context IS the context event; the interpolation chain IS the fallback when the context event has insufficient support. What we propose is not a departure from KN but a generalization: richer context events, detected by P-programs rather than fixed sliding windows.

11 Conclusion

The context event concept unifies several threads:

- The gap between KN-6 (1.784 bpc) and optimal compression is characterized by missing context events.
- Neural reuse (same patterns in different contexts) is solved by context events inducing different priors on shared LPPs.
- The max-min forward pass naturally implements Boolean context conditioning.
- The methodology (Boolean $\to$ NL $\to$ E/N/Q $\to$ quotient $\to$ query $\to$ P-program) provides a systematic path from residual analysis to model improvement.

The next step is to implement context event discovery and insertion in the UM runner, and to demonstrate that adding a single well-chosen context event (word length) measurably improves KN-6 compression on enwik9.