Context Events and the Induction of Priors

v2: Prediction Loss as Oversupport

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. Central to this version: we identify ordinary prediction loss (cross-entropy at the output) as *oversupport at the output ES*—the model supports event e but the observation forces event $x \neq e$. Backpropagation already measures this oversupport, but only at the output. The right event spaces are all you need: ramifying this mechanism through every ES in the model gives a general-purpose surprise and learning signal without requiring gradients.

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*.*

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 Prediction Loss as Oversupport

This is the central insight of v2.

3.1 The Time-Step Collapse

In the UM runner, at each time step t:

1. The forward pass runs, producing a support distribution over the output ES: $s(e_1), s(e_2), \ldots, s(e_{256})$ for byte prediction.
2. After normalization (softmax or simple division), this becomes a probability distribution $P(e_i)$.
3. The actual byte x_t is observed, which is equivalent to *forcing* $s(x_t) = 255$ while the model may have predicted $s(x_t) = 20$ and $s(e) = 200$ for some other event e.

The prediction and the input at the next time step *are the same event* in the output ES. The model's prediction and the world's observation collapse onto the same event space at the same moment.

Definition 4 (Oversupport)**.** *An event space* ES_a *exhibits* oversupport *at time t when the model assigns high support to an event e but the observed event is $x \neq e$:*

$$s_{\mathrm{model}}(e) \gg s_{\mathrm{model}}(x), \quad \textit{but } x \textit{ is the actual observation.}$$

Definition 5 (Undersupport)**.** *An event space* ES_a *exhibits* undersupport *at time t when no event has substantial support:*

$$\max_{e \in \mathrm{ES}_a} s_{\mathrm{model}}(e) < \tau$$

The model has no confident prediction.

Proposition 1 (Prediction Loss = Output Oversupport)**.** *The cross-entropy loss at the output ES,*

$$\mathcal{L}_t = -\log P(x_t),$$

measures exactly the oversupport at the output ES. When the model strongly supports the wrong event, $P(x_t)$ is small and $\mathcal{L}_t$ is large. When the model correctly supports x_t, $P(x_t)$ is large and $\mathcal{L}_t$ is small.

This is not a metaphor or an analogy. Cross-entropy loss *is* oversupport. They are the same mathematical object viewed from two perspectives: the loss function perspective (ML) and the event space perspective (UM).

3.2 Backpropagation Already Measures This

Gradient descent with backpropagation computes $\nabla_\theta \mathcal{L}_t$—the gradient of the oversupport at the output ES with respect to all model parameters. This is already the standard training procedure for neural networks.

But it only measures oversupport at the output ES. The output ES is unique in that:

- It has an explicit observation (the next byte).
- It is normalized via softmax to give well-behaved probabilities.
- The loss function is defined directly on it.

No other ES in a typical model receives this treatment. The hidden layers of a neural network have no explicit "observation" to compare against, no softmax, no loss function.

3.3 The Signal of Special Treatment

Just as with the bias conjecture (20260216 archive)—where the special treatment of the bias term in the output layer was a signal that a similar mechanism should be applied throughout—the special treatment of the output ES is a signal.

Remark 1 (The Ramification Principle)**.** *When the output ES has a mechanism (oversupport detection via loss, softmax normalization, gradient-based correction) that is absent from other ESs, this is a strong signal that the mechanism should be ramified throughout the model. The output ES is not special; it is merely the one ES we already handle correctly.*

The right event spaces are all you need. If we identify the correct factorization of E into event spaces, and equip each ES with oversupport detection and a correction mechanism, then the entire learning process reduces to:

1. Forward pass (max-min update).
2. For each ES with an observable: detect oversupport.
3. Update patterns and connections in response.

No backpropagation is needed, because each ES has its own local surprise signal.

3.4 The Binary Case

In a binary event space $\mathrm{ES} = \{e_0, e_1\}$:

- **Oversupport = contradiction.** Both events have support, but only one can be true. The model says "yes" and the world says "no" (or vice versa). This is a logical contradiction between model and world.
- **Undersupport = excluded middle.** Neither event has support. The model has nothing to say. This violates the law of the excluded middle: one of the two must be true, but the model doesn't know which.

This gives a precise logical semantics to surprise in the UM.

3.5 Two Explanations for Every Surprise

For any oversupport event at any ES, there are exactly two possible explanations (which may combine):

1. **Genuine luck.** The world is actually surprising. A rare event has occurred—we have won the lottery. The model is correct; reality is improbable. In compression terms, this is irreducible entropy.
2. **Model error.** The event space is wrong, or the patterns connecting it are wrong. The model has the wrong factorization of E, or the weights are incorrectly learned. This is reducible compression gap.

Distinguishing between these two cases is the central challenge of learning. If surprise is genuine luck, no model change helps. If surprise is model error, the correct response is to add or refine event spaces and connections.

Over many observations, systematic oversupport (the same kind of error repeating) is strong evidence for model error. Isolated oversupport is more likely genuine luck. The learning function ω should respond to systematic patterns, not individual events.

4 The Conjecture

Conjecture 1 (Missing Prior = Missing Context Event)**.** *In P-programming, whenever a model exhibits systematic oversupport at any ES—a pattern of incorrect predictions that repeats—there exists a context event e_c in some ES such that adding e_c and its connections to the model eliminates the systematic component.*

This refines the v1 conjecture: it's not just "missing priors" in the abstract, but specifically *systematic oversupport* that indicates a missing context event. Isolated oversupport (genuine luck) does not require a new context event.

5 KN Smoothing as Crude Context

In KN-6, the context for predicting byte t is the preceding k bytes $(b_{t-k}, \dots, b_{t-1})$. The KN interpolation chain (order $6 \to 5 \to \cdots \to 1 \to$ unigram) is a tower of progressively coarser contexts.

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.

Our surprise analysis confirms this empirically: KN-6 achieves 0.38 bpc inside XML tags but 2.46 bpc outside them (86.8% of mean surprise). The tag structure is highly predictable, but only if you know you're inside a tag—a context event that the 6-byte window almost never captures.

6 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. The oversupport that KN-6 experiences when predicting tag-interior "h" (expecting text patterns) is eliminated.

7 The Boolean Interpretation

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

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

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).

In the Boolean interpretation, oversupport at a binary ES is a logical contradiction: the model's conjunction says y should be true, but the observation says y is false. The correction is either (a) the conjunction was wrong (model error—refine the pattern) or (b) the conjunction was right but a rare exception occurred (genuine luck—no correction needed).

8 The Methodology

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

1. **Identify systematic oversupport.** Score the data with the current model. Find positions where oversupport repeats in a pattern—not isolated high-surprise events, but clusters sharing a common feature.
2. **Boolean interpretation.** Express the pattern: "the model systematically oversupports e when the actual observation is x, whenever [some condition holds]."
3. **Natural language.** State the condition: "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. Verify that the condition separates high-oversupport from low-oversupport positions.
6. **P-program.** Implement the context event as an ES with events detected by a P-program. Connect it to the target LPP's input ES. The model can now condition on the context, eliminating the systematic oversupport.

9 Empirical Validation

Our surprise analysis (`umr surprise enwik9 1000000`) provides the first empirical test of the context events framework:

- **in_tag**: 0.38 bpc inside tags vs. 2.46 bpc outside (86.8% of mean). This is the single most powerful context event we have identified. It would eliminate the systematic oversupport that KN-6 experiences at tag boundaries.

- **word_len**: Boundaries (word_len=0) have 3.98 bpc; mid-word (len 3–4) has 1.73 bpc. Word position is the second most powerful context event.
- **Surprise distribution**: 40.7% of positions have < 0.25 bpc (nearly free); 8% have > 7.75 bpc (nearly random). The heavy tail is where context events are most needed.
- **Top surprises**: 44–54 bits per byte at positions with rare markup, non-ASCII characters, and structural transitions. These are extreme oversupport events.

10 Open Issues for P-Programming

10.1 Ramifying Oversupport Detection

The key open problem: how to give every ES in the model the same oversupport detection that the output ES already has. This requires:

- An "observation" for each ES—either from the data directly (for ESs with ground truth, like tag structure) or from a consistency check (does the ES's state match what the downstream ESs expect?).
- A normalization step—analogous to softmax at the output—that converts support values to a probability distribution over each ES.
- A local correction signal—analogous to the cross-entropy gradient—that adjusts patterns connected to the ES.

If this can be done for every ES, the UM learns without backpropagation: each ES is a local learning unit with its own surprise signal.

10.2 Discovery

How do we systematically discover which context events to add? The surprise analysis (Step 1) is easy in principle but requires grouping systematic oversupport into patterns. We need:

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

10.3 The Carrier Signal

Memory traces can be stored "inline" via a carrier signal: an event representing 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. We defer detailed treatment to a future paper.

11 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. The KN sliding window is *already* a context event mechanism—just a crude one. What we propose is a generalization: richer context events, detected by P-programs rather than fixed sliding windows, with oversupport detection at every level of the tower rather than only at the output.

12 Conclusion

The context event concept, combined with the identification of prediction loss as oversupport, unifies several threads:

- Cross-entropy loss IS oversupport at the output ES. Backpropagation IS oversupport-driven correction. These are not analogies; they are identities.
- The special treatment of the output ES (softmax, loss, gradients) is a signal to ramify the same mechanism throughout the model. The right event spaces are all you need.
- In a binary ES, oversupport is contradiction and undersupport is the excluded middle. Every surprise has two explanations: genuine luck or model error.
- The gap between KN-6 and optimal compression is systematic oversupport at missing context events, confirmed empirically (in_tag: 86.8% of mean surprise).
- The methodology (identify systematic oversupport $\rightarrow$ Boolean $\rightarrow$ NL $\rightarrow$ E/N/Q $\rightarrow$ query $\rightarrow$ P-program) provides a path from loss to model improvement that does not require gradients.

The next step is to implement ramified oversupport detection in the UM runner: equip each ES with its own local surprise signal and correction mechanism, and demonstrate that this allows the model to learn context events automatically.