The Lexicon Embedding:
From Bytes to Words in the Universal Model

Claude and MJC

March 2026

Abstract

We synthesize six months of work on byte-level universal models into a single account of how the UM acquires a lexicon. The key result is that word events enter the memory trace explicitly, producing a two-level encoding: a word event (from a learned vocabulary) followed by residual bits that specify the exact spelling. This is lossless dimensionality reduction—unlike tokenization, no letter-level information is discarded. The orthographic embedding (word $\leftrightarrow$ letters) is a learned bijection that works bidirectionally through the same LPP counts. We formalize the encoding, derive the MDL criterion for lexicon discovery, present the experimental results that motivate the design (100-word mixture: -0.552 bpc at 10M, -0.111 vs KN-6), and lay out the concrete experimental plan for the full-lexicon implementation.

1. Background: The Universal Model

A universal model (UM) is a 5-tuple (E, P, T, f, ω): event spaces E, patterns P connecting events across spaces, a thought vector T giving the current support (log-count) for each event, a forward pass f, and a learning rule ω.

The forward pass is max-min:

$$f_P(T)_j = \max_i \min(T_i, P_{ij}).$$

Each pattern is a weighted connection from a source event to a target. Conjunction patterns combine multiple sources: $\min(T_a, T_b, w)$.

Learning (ω_0) is log-stochastic counting: when joint event (a, b) is observed, the LPP entry's support s increments with probability 2^{-s}, so $\mathbb{E}[s] \approx \log_2(\text{count})$.

The output distribution is softmax over log-support: $p(b) = 2^{s(b)} / \sum_{b'} 2^{s(b')}$.

These are the only primitives. Everything that follows is built from event spaces, patterns, and this forward pass.

2. The Agent Model and Working Memory

The model is an agent whose output O is both its prediction and its input. At each time step:

1. **Forward**: apply f_P to produce a distribution at O.

2. **Score**: measure $-\log_2 p(x_t)$ from that distribution.
3. **Observe**: set O to the actual byte x_t (support 255).
4. **Learn**: observe joint events in all LPPs.
5. **Shift**: clear memory ESs, copy $O \to O^{-1} \to O^{-2} \to \cdots$

The shift chain is implemented by absolute patterns ($w = 255$) in P. Memory ESs are cleared each step (transient support—the biological default). The shift chain depth determines the model's context: depth 1 gives bigrams, depth 2 gives trigrams, etc.

Training vs. inference differ only at step 3: the environment provides the byte (training) or the agent samples from its distribution (inference). The architecture is identical.

Conjunction patterns arise from threshold creation: when a joint event in an LPP reaches sufficient support ($\tau \approx 4$, meaning ~16 observations), the joint event is reified as a neuron in a conjunction ES. A trigram "the" is the neuron for the joint event ($O^{-2}[\texttt{t}], O^{-1}[\texttt{h}]$), which then connects via a second LPP to $O[\texttt{e}]$.

3. The Memory Trace

The memory trace is the residual: what the model cannot predict. At each position t, the model provides a probability distribution $p_t(\cdot)$ over the 256 possible bytes. The actual byte x_t is encoded against this distribution via range coding. The trace is the range-coded bitstream—the surprise at every position, compressed. The model and the trace are separate: the frozen model (.sn file) plus the trace file together recover the original input exactly.

This has been implemented and verified: the range coder round-trips all of enwik9 (1 billion bytes, zero mismatches).

As the model improves, the trace shrinks (less residual surprise). The Tick-Tock process exploits this: freeze the model, extend the architecture, replay the trace with the extended model, write a shorter trace. Each cycle produces a better model and a shorter trace. In the brain, memory traces are themselves patterns in P; we maintain an external trace to simulate memory, which can be replayed to improve the model.

4. The Orthographic Model

We have a byte-level model that predicts bytes within word spans. With a depth-k shift chain and word-boundary clearing (all memory ESs clear on space/punctuation), the model captures within-word letter statistics without cross-word leakage.

Experimental results with the "the" experiment confirm the mechanism works at three depths:

Depth	What it captures	Entries	Entropy
Unigram	bag of letters: {t, h, e}	3	1.40 bits
Bigram	transitions: th, he	2	0.07 bits/step
Trigram	the word itself	1	0.04 bits

The unigram says "{t, h, e} equally likely." The bigram says "after t comes h" (near-deterministic). The trigram says "the word IS 'the'" (one entry, 0.04 bits of entropy). Word boundary clearing ensures each depth sees only within-word context: the bigram has exactly 2 entries (th, he), not ∼65 entries leaked from adjacent words.

This is the downward-looking embedding: given a word event, the model predicts its spelling. The same LPP counts also work upward: given a partial spelling, the counts give a distribution over which word it might be. This bidirectionality is a structural property of the joint count table (the sufficient statistic for both directions—see *Bayes from Counting*, 20260212 archive).

5. The 100-Word Experiment

We built a 100-word oracle lexicon and measured the gain from word-level prediction against a bigram baseline. The key results:

5.1. Oracle word selection fixes broadcast

The original Exp 5.2 failed because all 100 word events fired simultaneously in English text, causing ω_0 to learn the English letter marginal for every word. The fix: oracle activation (one word event active at a time). Each word then learns its own orthographic distribution—"the" shows {t, h, e, T, H, E}, "of" shows {o, f, O, F}—not the shared marginal.

5.2. The word mixture from LPP counts

The LPP's joint count table is bidirectionally Bayesian: it is the sufficient statistic for both word → letters and letters → word (see *Bayes from Counting* [3]). The word distribution comes for free from the same counts that give the spelling. At each within-word position k, the mixture over candidate words is:

$$P(\text{byte}_k) = \sum_{w \in C_k} P(w \mid \text{prefix}) \cdot \mathbb{I}[w[k] = \text{byte}_k],$$

where C_k is the set of words matching the observed prefix, and $P(w \mid \text{prefix}) \propto \text{count}_w$. This is not a separate mechanism; it is what the LPP already gives us.

Scale	Bigram only	Online mixture	Δ
100K	4.443	3.533	−**0.910**
1M	4.178	3.539	−**0.639**
10M	4.106	3.554	−**0.552**

Online word frequency estimation matches the oracle exactly. The Laplace-1 prior is not a separate mechanism: it falls directly out of softmax over log-support starting at 0 (every word begins with $s = 0$, i.e., $2^0 = 1$ count). This is a real compressor component: causal, online, no lookahead.

5.3. Against KN-6

At 100K: KN-6 alone 2.719 bpc, mixed 2.608 bpc, **gain** −0.111 **bpc (4.1%)**.

Position 0 (word start) contributes 49% of the total gain. This is the *change of base*: at word boundaries, word-frequency-weighted prediction beats byte-frequency prediction. KN-6 scores 5.1 bpc at word starts; the mixture scores 4.6 bpc.

5.4. The L_∞ vs L_1 gap

Max-min (the UM forward pass) is L_∞: the most-supported prediction wins. The word mixture is L_1: a probability-weighted average over candidates. For multi-hypothesis prediction (multiple plausible words), L_1 dominates L_∞. This is the marginal dominance problem from a different angle: the fix must operate at the distribution level, not the support level.

6. From 100 Words to the Full Lexicon

We now have all the pieces. The step is: go from 100 oracle-selected words to the entire lexicon, discovered from data, with word events encoded explicitly in the memory trace.

6.1. What a word event is

A word event is a compound event: the conjunction of byte events observed during a word span, as seen through the shift chain at the moment of reset (space, punctuation, tag delimiter).

After processing "the␣", the shift chain holds $O^{-3}[\texttt{t}] \wedge O^{-2}[\texttt{h}] \wedge O^{-1}[\texttt{e}]$. This conjunction IS the word "the"—its identity as seen through the model's structure.

Words are actual words, defined by the community and discovered by the model via frequency. The organism does not have a pre-loaded dictionary. It processes bytes, maintains the shift chain, and discovers recurrent patterns at word boundaries. "the" emerges because the conjunction at reset recurs ∼2 million times in enwik9—far above any threshold.

6.2. The orthographic embedding as bijection

The word ↔ letter mapping is a bijection between two event spaces, mediated by the LPP's joint count table.

Downward (word → letters): given word event w, the OBSERVE LPP gives $p(\text{byte} \mid w)$ at each depth. For "the": unigram {t, h, e} at $\frac{1}{3}$ each, bigram th→h and he→e near-deterministic, trigram "the" at 0.04 bits.

Upward (letters → word): given a partial spelling, the same counts give $p(w \mid \text{prefix})$. After seeing "th", the candidates narrow to {the, that, this, there, they, them, then, . . . }, weighted by frequency. This is the Bayesian mixture from §5.2.

The embedding works by driving the model forward event by event. Each letter event updates the posterior over candidate words. At the point of causal capture (when the word is identified, typically at or before the word boundary), the posterior collapses to one word plus residual bits for variant spellings.

6.3. The two-level trace

The memory trace becomes a sequence of word events with residual bits:

1. At each word boundary, emit a **word event** from the vocabulary. The word event is encoded against the word-level distribution (word frequencies, or eventually word-bigram predictions).
2. For each word event, emit **residual bits** that specify the exact spelling. Given the word "the", the orthographic embedding predicts t-h-e-space with near certainty, so the residual is near zero. For "The" (capitalized), a few bits encode the deviation. For a misspelling "teh", more bits encode the transposition.
3. At non-word positions (punctuation, XML tags, numbers), fall back to byte-level encoding against the byte model.

This is lossless: word event + residual bits + byte-level fallback recovers every byte of the original. Unlike tokenization, no letter-level information is discarded. The word event provides a distribution over spellings; the residual encodes the actual spelling against that distribution.

This gives two *views* on the memory trace. We can look at just the word events—a quotient that discards the remainder—or we can look at the full picture, which captures distinctions like the word "the" followed by ",␣" versus "␣". The difference between these views is the energetic surprise response of the organism: "playing out" the low-level details from a high-level memory trace happens in response to surprise, which is an ES-level event. When the word-level prediction suffices (the common case), the organism coasts; when it doesn't (an unusual delimiter, a capitalization), surprise forces attention to the residual.

An aside on the relationship to classical embeddings: if the word events and the residual bits were encoded not continuously (as in arithmetic coding) but with a reset (as in Huffman coding), then under Huffman the word bits would be identical for "the" and "The"—only the spelling residual differs. The deeper point is that under a careful choice of event ordering—the $E \to N$ map—the spelling *residual* bits would also tend to look the same across words that share the same transformation. All capitalized words would have similar low-order residual bits; all words with a doubled letter would share a pattern in the residual. A higher abstraction level could then recognize "capitalized" or "misspelled" directly from the residual bits, without knowing the word. This depends entirely on the $E \to N$ map. We learn over events, not over compressed bits, so this is not our current direction, but it illuminates how the residual carries structured information, not noise.

6.4. The word-ending character

The word-ending character (space, comma, period, newline, etc.) is part of the word's "token." From the word event, we get a distribution over the delimiter that follows (mostly space, sometimes punctuation). The actual delimiter is encoded as part of the residual.

This handles the inventory of inter-word characters without a separate model. The word "the" followed by a comma is the same word event as "the" followed by a space; the delimiter is a residual detail.

6.5. Abduction: choosing the word event

When writing the trace (the initial pass over the data), the model must commit to a word event at each word boundary. This is abduction: given the observed spelling, pick the highest-probability word.

For well-edited text (Wikipedia), this is trivial: nearly every word span matches exactly one vocabulary entry. The interesting cases are:

- **Capitalization**: "The" and "the" map to the same word event; the case is residual.
- **Rare words**: words below the frequency threshold are encoded byte-by-byte (the escape mechanism).
- **Variant spellings**: "colour"/"color" could be two word events or one with a residual bit. The MDL criterion decides.

6.6. The MDL criterion

The lexicon that minimizes total encoding is the one that minimizes:

$$\text{Total} = \underbrace{|\text{lexicon}|}_{\text{model cost}} + \underbrace{\sum_{\text{words}} -\log_2 p(w \mid \text{context})}_{\text{word-level trace}} + \underbrace{\sum_{\text{words}} \text{residual}(w)}_{\text{spelling residual}} + \underbrace{\sum_{\text{non-words}} -\log_2 p(b \mid \text{byte context})}_{\text{byte fallback}}$$

In UM terms, adding a word to the lexicon means adding to both E and P: a new event in the word ES (the word event itself) plus the patterns from that event onto the letter-level events (the orthographic projection). The model cost of a word is the cost of encoding these additions—concretely, the word event's SN entry (its name, ES membership) plus its LPP entries (one per observed letter at each depth of the orthographic model). The savings are: every occurrence of that word in the trace is now one word event (encoded against the word-level distribution) instead of n byte events.

A word is worth adding when its frequency times its per-occurrence savings exceeds its model cost. The closest practical mechanism we have for this decision is the threshold at log-support $\tau \approx 4$ ($\sim$16 observations) used for bigram and trigram neuron creation: joint event reification. The same threshold governs here—a word-boundary conjunction that has been observed enough times to justify reification as a word event. The MDL criterion is the theoretical justification for this threshold: the word pays for itself when the trace savings exceed the model cost of carrying it.

For enwik9, the top 100 words cover 17% of byte positions and provide -0.552 bpc gain. Extending to the full lexicon ($\sim$70K words in enwik9) covers the vast majority of English text. The long tail of rare words has diminishing returns, bounded by the MDL criterion.

7. The Factor Tower

The word-level trace is the first step up the factor tower: byte $\rightarrow$ word $\rightarrow$ phrase $\rightarrow$ sentence $\rightarrow$ discourse.

Each level is a change of base. The byte trace has 10^9 digits in base 256. The word trace has $\sim$1.5 $\times 10^8$ digits (one per word) in base $|W|$ ($\sim$70K), plus residual bits. Fewer digits, higher base, more information per digit.

Once the lexicon exists, the same Tick-Tock process continues. Word events become the input to phrase-level LPPs: which word follows which. The word-bigram distribution is sharper than the byte-bigram distribution at word boundaries (the change-of-base gain). Phrase patterns emerge from word-pair frequencies, just as word patterns emerged from byte-pair frequencies.

The mathematical framework is the same at every level. Events are integers; event spaces are rings $\mathbb{Z}/N\mathbb{Z}$; the quotient map is modular reduction; CRT factorizes independent sub-spaces (see *Integer Factorization of Events*, 20260216 archive). When $|W|$ is coprime to 256 (e.g., $|W| = 65537$, a Fermat prime), the word and byte event spaces are algebraically independent, and CRT gives the exact decomposition.

8. The Algebraic Structure

The lexicon embedding has a precise algebraic interpretation in the integer framework of [2].

8.1. Event spaces as quotient rings

Every event space of size N is the ring $\mathbb{Z}/N\mathbb{Z}$. The byte output space is $\mathbb{Z}/256\mathbb{Z}$. A word vocabulary of size $|W|$ defines a word event space $\mathbb{Z}/|W|\mathbb{Z}$.

The projection from bytes to words is a quotient map: the byte sequence $(b_1, \ldots, b_k)$ during a word span maps to a word event w, with the remainder (the exact spelling modulo word identity) as the residual. In the language of [2]: division is the quotient map, and the remainder is what was forgotten.

8.2. CRT and independence

The Chinese Remainder Theorem gives the decomposition: when $\gcd(|W|, 256) = 1$, the word and byte event spaces are algebraically independent:

$$\mathbb{Z}/(|W| \cdot 256)\mathbb{Z} \;\cong\; \mathbb{Z}/|W|\mathbb{Z} \;\times\; \mathbb{Z}/256\mathbb{Z}.$$

Since $256 = 2^8$, any odd $|W|$ satisfies $\gcd(|W|, 256) = 1$. For example, $|W| = 65537$ (a Fermat prime), or more practically, any odd vocabulary size. This algebraic independence means the word event and the byte event carry no redundant information—the word event is “orthogonal” to the byte-level residual.

In practice, we do not need to enforce coprimality; the CRT tells us that when it holds, the factorization is *exact*, and when it does not (shared factor of 2), there is a small coupling that the residual absorbs.

8.3. Bayes from counting and bidirectionality

The LPP between the word ES and the byte ES is a joint count table $c(w, b)$. By the framework of [3], this single table gives both directions:

- **Downward**: $P(b \mid w) = c(w, b)/c(w)$. The orthographic model.
- **Upward**: $P(w \mid b) = c(w, b)/c(b)$. The recognition model.
- **Bayes**: $P(w \mid b)/P(b \mid w) = P(w)/P(b)$. Consistency of the two partial quotients.

The GCD decomposition separates common evidence (how often the word occurs) from differential evidence (which byte follows). The conditional probability depends only on the *reduced counts* $r(w, b) = c(w, b)/\gcd_b c(w, b)$; the GCD cancels. This means the orthographic embedding is insensitive to absolute frequency—a rare word and a common word with the same spelling pattern have the same conditional distribution over bytes.

The tropical approximation connects this to the UM forward pass: min in the max-min rule upper-bounds the integer GCD. For integer counts where the smallest count divides all others (common in natural data), the bound is tight and the UM's forward pass is exact Bayesian inference.

8.4. The two-level encoding as quotient chain

The two-level trace is a chain of quotient maps:

$$\text{byte sequence} \xrightarrow{\pi_W} \text{word event} \xrightarrow{\text{AC}} \text{compressed word}$$

with the remainder at each step:

$$\text{byte sequence} = \underbrace{\text{word event}}_{\text{quotient}} + \underbrace{\text{spelling residual}}_{\text{remainder}}.$$

The word event is encoded against the word-level distribution (the quotient), and the residual is encoded against the orthographic model (the remainder). Both use the range coder. The total is lossless because the quotient and remainder together recover the original (the division algorithm).

This is the concrete instance of the general principle from [2]: compression is factoring the event integer into quotient-remainder pairs.

9. P-Programming the Lexicon

The lexicon implementation is a P-program: a composition of event spaces and patterns in the UM's concrete representation. Following the P-program framework (see *P-Programs*, 20260215 archive), each component is a small UM instance with interface events (visible to the rest of the model) and internal events (hidden).

9.1. Prior P-programming work

The following P-programming primitives, developed in earlier papers, are the building blocks from which the lexicon implementation will be composed:

- **Position counter**: deterministic event chain $\text{pos}_0 \to \text{pos}_1 \to \cdots$, reset at word boundaries. Tracks within-word position without counting. (*P-Programs* [9], §3.1)

- **Letter accumulator**: latch-based bag-of-letters—the set of bytes seen since the last word boundary. Cleared on reset. (*P-Programs* [9], §3.2)
- **Bag-of-letters recognition**: graded support over candidate words from the accumulated letter set. (*P-Programs* [9], §3.3)
- **Shift chain**: $O \to O^{-1} \to O^{-2} \to \cdots$ with absolute patterns, holding the conjunction of recent bytes. The depth determines context length. (*Working Memory* [5], §3)
- **Threshold creation**: when a joint event reaches $\tau \approx 4$ ($\sim$16 observations), the joint event is reified as a neuron. This is the standard ω architecture extension (adding to E and P). Applied at word boundaries, it creates word events. (*CMP* [1], §4)
- **Abductive learning**: safe combination of P-programs—the conjecture that composing well-formed P-programs preserves correctness. (*P-Programs* [9], §5)

The lexicon implementation will compose these primitives. The details of that composition—which events to create, which patterns to connect, how to handle words longer than the shift chain—are the subject of the experimental plan (§10), not predetermined by design.

9.2. What the SN model looks like

In the SN interchange format, the lexicon model contains:

1. ES declarations: `byte_output` (256), `byte_prev` (256), ..., `byte_prev_k` (256), `word` (dynamic).
2. LPPs: shift-chain LPPs (`byte_output` $\to$ `byte_prev`, etc., weight 255), orthographic OBSERVE LPPs (`word` $\to$ `byte_output`, learned weights).
3. DYNAMICS block: specifies clearing of memory ESs and shift timing.
4. Word events: one event per discovered word in the word ES, with support reflecting log-frequency.

9.3. Recovering longer spellings

For words longer than the shift chain depth, there are two P-programming strategies:

Extended shift chain. Increase the shift chain depth from 3 to 10–12. Each additional depth adds one ES of 256 events and 256 absolute patterns. The cost is linear in depth: $k \times 256$ events and $k \times 256$ patterns. For $k = 12$, this covers words up to 12 letters (98% of English). The trade-off: more events in the model, potentially more LPP entries. But the conjunctions that fire at word boundaries are exactly the words in the vocabulary—the rest never accumulate support.

Trigram chaining. Keep the shift chain at depth 2–3 but exploit the fact that trigram events chain across positions. The trigram "the" at position 0–2 connects to the trigram "he␣" at position 1–3. This chaining may require shifting *trigram events* (not bytes) through a secondary shift chain. The advantage: the conjunction space is smaller (only frequent trigrams, not all 256^k).

The disadvantage: more complex P-programming, and the chain's reliability depends on trigram coverage.

The choice between these strategies is not purely empirical at the current level. The question is which architecture best supports the next steps up the factor tower: phrase-level patterns, sentence structure, and beyond. Trigram chaining preserves the compositional hierarchy that higher levels can exploit; extended shift chains give a flat representation that is simpler but may not compose as naturally.

10. Experimental Plan

10.1. Phase 1: Full-lexicon word discovery

Build the word vocabulary from the data during the online training pass:

1. Process enwik9 with the working-memory agent model (shift chain depth ≥ 3, word-boundary clearing).
2. At each word boundary (space, punctuation), the shift chain holds a conjunction of byte events. Record this conjunction.
3. When a conjunction recurs above threshold τ, create a word event (a neuron in the word ES).
4. The word ES grows dynamically. Expected size: $\sim$70K words for enwik9.

This is threshold creation applied to word-boundary conjunctions—the same mechanism that creates bigram and trigram neurons, operating at the word level. No external word-counting subroutine; the UM discovers words through its own event spaces.

Prediction: the discovered vocabulary should closely match the $\sim$73K unique words found by the oracle word scanner in the 100-word experiment.

10.2. Phase 2: Orthographic OBSERVE LPPs

For each discovered word event, learn the spelling distribution. The "the" experiment showed this works at three depths; the question is how to scale it to the full vocabulary.

Two approaches to recovering the full spelling from the word event:

1. **Extend the shift chain** past depth 3. A shift chain of depth k lets the model see k previous bytes. For words up to length k, the spelling is fully determined by the conjunction at reset. Most English words are ≤ 10 letters, so depth 10–12 would cover the vast majority.
2. **Trigram chaining**: keep the shift chain at depth 2–3 but rely on chaining through trigram events to recover longer spellings. The trigram "the" connects to the trigram "he␣" at the next position. This may require a shift chain for the trigrams themselves (shifting trigram events rather than bytes).

Both approaches are P-programming strategies: they specify which event spaces to create and which patterns to connect, not imperative code. The right choice is not determined by which gives shorter traces at the current level. What matters is which architecture will be most useful as we continue up the abstraction chain—from words to phrases to sentence structure. An extended shift chain creates a uniform representation that phrase-level LPPs can operate on directly; trigram chaining preserves the hierarchical structure (bytes → trigrams → words) that the factor tower expects. The forward-looking criterion dominates the immediate empirical outcome.

Prediction: each word should learn near-deterministic bigram spelling (entropy < 0.1 bits/step for common words, as seen with "the"), with case variants appearing as low-support alternatives.

10.3. Phase 3: Word-level trace encoding

Implement the two-level trace:

1. At word boundaries: encode the word event against the word frequency distribution using the range coder.
2. Within the word: encode the residual (actual spelling against the orthographic model's prediction) using the range coder.
3. At non-word positions: encode bytes against the byte model.

The range coder infrastructure already exists and is verified lossless. The extension is switching between word-level and byte-level distributions at word boundaries.

Prediction: the two-level trace should be shorter than the byte-only trace. The gain comes from two sources:

- *Change of base* at word boundaries (position 0): the word frequency distribution is sharper than the byte model's prediction. Measured: -0.543 bpc per word-start position vs KN-6 (§5.3).
- *Spelling compression*: within-word positions are near-deterministic once the word is known. Measured: bigram entropy 0.07 bits/step for "the."

Expected gain: -0.5 to -1.0 bpc over the byte-only bigram model, with the lower end achievable at 10M and the upper end requiring full-enwik9 training.

10.4. Phase 4: Replay and the second Tick

1. Freeze the word-level model (Tick 0 complete).
2. Decode the word-level trace. The frozen model reconstructs the original bytes.
3. Add word-bigram LPPs: $p(w_{t+1} \mid w_t)$. This is the phrase-level structure.
4. Replay the trace with the extended model. The model now sees word events and learns word-bigram statistics.

5. Write a new trace. If word-bigram predictions are sharper than word-unigram, the trace is shorter.

This is the first full Tick-Tock cycle: byte model $\to$ word model $\to$ word-bigram model.

Prediction: word-bigram entropy is $\sim$9 bits/word (measured from the Tock empirical paper), vs word-unigram entropy $\sim$12 bits/word ($\log_2 70000 \approx 16$ bits without frequency weighting, $\sim$12 with). The $\sim$3 bits/word gain distributed over $\sim$5 bytes/word gives $\sim$0.6 bpc additional improvement.

11. Why Not Tokenization

Tokenization is an "attractive nuisance" (see *Tokenization as Information Loss*, 20260215 archive): it appears to simplify by reducing the number of events per sequence, but it fundamentally discards information.

11.1. Tokenization as coarsening

A tokenization $\tau : \Sigma^* \to T^*$ is a surjective map from byte sequences to token sequences. It induces a coarsening of the event space: $E_{\text{token}} \le E_{\text{byte}}$. Three types of information are discarded:

1. **Spelling variants**: if "the" and "teh" map to the same token, the model cannot distinguish standard from variant spelling. The lost information is $H(\text{spelling} \mid \text{token})$.
2. **Internal position**: within a token, the byte-level position is invisible to the token-level model.
3. **Letter composition**: the letter accumulator (which bytes have appeared so far) is collapsed to a single token identity.

11.2. The strawberry theorem

A token-level model cannot solve the letter-counting problem ("how many r's in strawberry?") unless the vocabulary encodes every possible letter count as a separate token (exponentially many) or the model has access to byte-level decomposition (which undoes the tokenization). In the UM's extended event space, the letter accumulator provides this information as a first-class event at every position.

11.3. The formal bound

For any fixed tokenization τ with mean token length $\bar{L}$:

$$h_b \le \frac{h_\tau}{\bar{L}} + \frac{H(\text{bytes} \mid \text{tokens})}{\bar{L} \cdot N},$$

where h_b is the byte-level entropy rate and h_τ is the token-level rate. The second term is the information loss per byte. For BPE on enwik8 ($\bar{L} \approx 4.5$), the lower bound on loss is ≥ 0.05 bpc, and the worst case (texts with misspellings, formatting, code) reaches ≥ 0.12 bpc.

Token-level modeling is strictly suboptimal unless the tokenization is a sufficient statistic—which no fixed tokenization is for natural language.

11.4. Our approach: extension, not coarsening

The architectural distinction:

- **Tokenization**: $E_{\text{byte}} \to E_{\text{token}}$ (coarsening, information loss).
- **Lexicon embedding**: $E_{\text{byte}} \to E_{\text{byte}} \times E_{\text{word}}$ (product, no information loss).

The byte-level model is preserved intact. The word event is an *additional* structure inside H, not a replacement for the byte level. Every letter event is still observable. The residual bits encode any spelling detail that the word event does not predict. The total (word event + residual) is lossless.

For well-spelled text, the residual is tiny (near-deterministic orthography). For noisy text, the residual grows—but so would the information loss from tokenization, which we avoid.

11.5. The value of abstraction

Tokenization's appeal is not only computational efficiency; it is *abstraction.* Discarding information is what abstraction does. A word-level model that ignores spelling details operates in a lower-dimensional space where the relevant structure (meaning, syntax, discourse) is more accessible. This is a genuine advantage.

Our approach preserves this advantage. The word event is a lower-dimensional representation: $\sim$70K word events vs. 256^k byte sequences. Higher-level operations (phrase prediction, discourse structure) operate on word events, not bytes. The spelling residual is *available* but not *required*—higher levels can ignore it entirely, achieving the same dimensionality reduction as tokenization.

The difference is that the residual is not *destroyed.* When a higher level needs byte-level information, it can expend energy to replay the residual through the orthographic model and recover the exact spelling. This is not free: replaying the residual requires running the orthographic model forward, which costs computation. But the information is there, stored losslessly in the trace.

This gives the organism an *attention mechanism* governed by surprise. At the ES level, surprise is the energetic signal that the current abstraction level is insufficient. When the word-level prediction suffices (the common case), the organism operates entirely at the word level—the residual exists in the trace but is not consulted. When surprise fires—a word seems wrong, a misspelling is suspected, someone asks "how many r's in strawberry?"—the organism attends to the residual, replaying byte-level events from the memory trace through the orthographic model.

The full picture: the organism operates at higher levels of abstraction, maintaining the ability to retrieve low-level events from the memory trace by expending energy. The attention mechanism—when to descend to a lower level—is governed by surprise, which is itself an ES-level event. This is the bridge between lossless encoding (everything is preserved) and efficient operation (most of the time, the higher level suffices). Tokenization achieves efficiency by destroying information; we achieve it by making the low-level information *available but not mandatory.*

12. Summary

1. The UM discovers words from data via threshold creation on word-boundary conjunctions. No external lexicon.
2. The orthographic embedding is a bijection: word $\to$ letters via OBSERVE LPPs at three depths, letters $\to$ word via Bayesian mixture over the same counts.
3. The memory trace encodes word events explicitly (change of base), with residual bits for spelling variants. Lossless.
4. The MDL criterion selects the vocabulary: add a word when its frequency $\times$ savings exceeds its model cost.
5. Measured gains: -0.552 bpc (bigram baseline, 10M), -0.111 bpc (vs KN-6, 100K). Position $0 = 49\%$ of gain.
6. The factor tower continues: word events feed phrase-level LPPs, each Tick-Tock cycle producing a shorter trace.
7. Unlike tokenization, this is lossless. The word event is a summary, not a replacement. The strawberry problem does not arise.
8. The approach achieves tokenization's abstraction benefit (operating in word-dimensional space) while preserving the ability to retrieve byte-level events by expending energy. Surprise governs this attention mechanism: descend to the residual when the word level is insufficient.

References

[1] Michaeljohn Clement. *CMP*. `https://cmpr.ai/cmp.pdf`, 2026.

[2] Claude and MJC. *Integer Factorization of Events*. Hutter archive, Feb 2026.

[3] Claude and MJC. *Bayes from Counting*. Hutter archive, Feb 2026.

[4] Claude and MJC. *Kneser–Ney on the Integers, v2: The Ring Structure*. Hutter archive, Feb 2026.

[5] Claude and MJC. *Working Memory in the Universal Model*. Hutter archive, Feb 2026.

[6] Claude and MJC. *The Memory Trace*. Hutter archive, Feb 2026.

[7] Claude and MJC. *The Path to the Lexicon*. Hutter archive, Feb 2026.

[8] Claude and MJC. *English Context Neuron: Experiment Results*. Hutter archive, Feb 2026.

[9] Claude and MJC. *P-Programs: Explicit Pattern Programs for the Extended Event Space*. Hutter archive, Feb 2026.

[10] Claude and MJC. *Tokenization as Information Loss*. Hutter archive, Feb 2026.

[11] Claude and MJC. *The Embedding Conjecture*. Hutter archive, Feb 2026.