The Trigram Embedding: A P-Program for the Orthographic Bijection

Design Document for Exp L2

Claude and MJC — March 6, 2026

1. Goal

From the research agenda (`#lexicon-path-agenda`, MJC 20260301): the word model becomes an embedding. A word event, expressed in characters, translates into a word event which combines a word embedding with extra bits of detail. The orthographic embedding gives a bijection from words to characters, which is learned. This must be a real bijection—readable in both directions from the same LPP counts.

The design constraint is P-programming: the architecture is expressed as event spaces, LPPs, and shifts. No imperative code outside the UM runner. The architecture must appear in SN and be loadable by both the C runner and the JS viewer.

A secondary goal is model size. The total information content of the model—the volume of information inside the SN, not the surface syntax—should be minimized. Higher-level structures represent *differences* to lower-level structure already represented. The LPP entries are the incompressible core; everything else is compressible overhead.

2. Architecture

2.1. The layers

The model has three layers of event spaces, each built from the one below via threshold creation on an LPP:

```
Layer 0 (bytes):
  ES "The output is ..." 256 OUTPUT.
  ES "The previous byte is ..." 256.
  SHIFT "The output is ..." "The previous byte is ...".

Layer 1 (bigrams):
  ES "The bigram is ..." dynamic.
  LPP "The previous byte is ..." "The output is ..."
      THRESHOLD 4 "The bigram is ...".

Layer 2 (words via trigram chaining):
  ES "The previous bigram is ..." dynamic.
  ES "The word is ..." dynamic.
  SHIFT "The bigram is ..." "The previous bigram is ...".
  LPP "The previous bigram is ..." "The output is ..."
      THRESHOLD 4 "The word is ...".
```

Layer 0 is the standard working-memory agent model: the output ES and a shift chain of depth 1. At each step, the previous byte is available.

Layer 1 introduces bigram events. The LPP between "previous byte" and "output" records joint byte pairs. When a pair reaches threshold $\tau = 4$ ($\sim$16 observations), a new event is created in the bigram ES. This ES grows dynamically. A bigram event named "The bigram is 'th'." means: the joint event (previous byte = 't', output = 'h') has been observed enough to justify a neuron.

Layer 2 introduces word events via trigram chaining. After the bigram ES is populated, we shift bigram events forward one step (SHIFT bigram $\rightarrow$ previous bigram). The LPP between "previous bigram" and "output" records trigram-like structures: (previous bigram, current byte). When these reach threshold, they create events in the word ES.

2.2. Why this is a trie

Consider the word "the" followed by a space. The event sequence is:

Position	Output	Prev byte	Bigram fired
t	't'	(boundary)	—
$t+1$	'h'	't'	"th"
$t+2$	'e'	'h'	"he"
$t+3$	' '	'e'	"e␣" (boundary)

At position $t+2$, the LPP entry (prev bigram = "th", output = 'e') fires. This is effectively the trigram "the". At position $t+3$, the entry (prev bigram = "he", output = ' ') fires—this is the word-ending trigram "he␣".

For the word "that":

Position	Output	Prev byte	Bigram
t	't'	(boundary)	—
$t+1$	'h'	't'	"th"
$t+2$	'a'	'h'	"ha"
$t+3$	't'	'a'	"at"
$t+4$	' '	't'	"t␣" (boundary)

The Layer 2 entries that fire are: (th, a), (ha, t), (at, ' '). The last one, (at, ' '), is the boundary entry. "hat" would share the same boundary entry (at, ' '), but differ at the previous step: "hat" has (ha, t) preceded by (·a, h), while "that" has (ha, t) preceded by (th, a).

This is the trie structure: working backward from the word boundary, each Layer 2 entry extends the word by one byte. The path through the trie from the boundary back to the word's start uniquely identifies the word. Two words with the same ending are distinguished by their earlier entries.

2.3. Incremental word identification

The trie can be built up incrementally using a second shift at Layer 2. At each step within a word:

1. The current bigram fires in Layer 1.
2. After the bigram shift, the previous bigram is available.
3. The Layer 2 LPP checks (previous bigram, current output).
4. If this combination has been seen enough, a trie node event exists for it.

The word event at the boundary is the *last* trie node in the chain. But it need not be the conjunction of all trie nodes—it is simply the Layer 2 event that fires at the boundary position. Two distinct words that happen to share the same final (prev bigram, boundary byte) would collide, but this is rare: the previous bigram encodes the last two bytes before the boundary byte, so the boundary event encodes the last three characters of the word.

For words that share a 3-character suffix, disambiguation requires looking further back. This is where the trie depth matters.

2.4. Trie depth and word coverage

Depth 1 (single Layer 2 entry at boundary): identifies words by their last 3 characters. This covers all words of length ≤ 3 uniquely. For longer words, it identifies a suffix class.

Depth 2 (boundary entry + one predecessor): identifies words by their last 4 characters. A second SHIFT at Layer 2 (previous trie node) plus a conjunction LPP would extend the identification.

Depth k: identifies words by their last $k + 2$ characters.

The question for the experiment is: what depth is needed to disambiguate most of the vocabulary? If most English words are unique in their last 3 characters, depth 1 suffices for the majority.

3. The Bijection

3.1. Forward: letters → word

As bytes arrive during a word span, bigrams fire (Layer 1), previous bigrams shift, and Layer 2 entries fire. At the word boundary, the active Layer 2 event identifies the word (to the depth of the trie). The posterior over word events is given by the LPP counts read in the "upward" direction:

$$P(w \mid \text{bigram}, \text{output}) = \frac{c(w, \text{bigram}, \text{output})}{c(\text{bigram}, \text{output})}$$

For a well-spelled known word, this collapses to a single candidate (plus residual for case, delimiter, etc.).

3.2. Backward: word → letters

Given a word event, the same LPP counts read "downward" give the spelling distribution:

$$P(\text{output} \mid w, \text{bigram}) = \frac{c(w, \text{bigram}, \text{output})}{c(w, \text{bigram})}$$

At each position within the word, the previous bigram is known (from the preceding step of the replay), and the word event is given. So the distribution over the next byte is determined by the three-way count table.

This is the same bidirectionality that the spec (§6) describes for two-input LPPs: the joint count table is a sufficient statistic for both conditional directions. No separate "recognition model" and "generation model" are needed—they are the same LPP read from different sides.

3.3. What makes this a real bijection

A bijection requires that both directions are near-deterministic for known words. The forward direction (letters → word) is near-deterministic by construction: a known word's spelling uniquely identifies it. The backward direction (word → letters) is near-deterministic because spelling is regular: given the word "the" and the position, the next byte is determined.

The residual measures the gap from exact bijectivity. For the common case (standard spelling, standard delimiter), the residual is near zero. For capitalization, one bit. For unusual delimiters, a few bits. The total encoding is: word event (from the word-level distribution) + residual (from the orthographic model).

In the UM's algebra, the word event is the quotient and the residual is the remainder. Together they recover the original byte sequence (the division algorithm). The quotient chain:

$$\text{bytes} \xrightarrow{\pi} \text{word event} \oplus \text{residual}$$

is lossless precisely because both π and π^{-1} are computable from the same LPP counts.

4. Model Size

4.1. Layer 1: bigram ES

The bigram ES has at most $256 \times 256 = 65{,}536$ possible events, but threshold creation limits it to frequently-occurring pairs. At 10M bytes of English, the number of bigrams above threshold 16 is empirically ∼2,000–4,000.

Each bigram event is one LPP entry (from, to, weight)—three integers. The entry already exists in the LPP before threshold creation (it was counted by ω_0). Creating the bigram event adds an event to the ES but no new LPP entry. The model cost per bigram neuron is: one event name in the bigram ES (compressible overhead) plus whatever downstream LPPs connect to it.

4.2. Layer 2: word ES

The word ES grows via threshold creation on the (prev bigram, output) LPP. Its size is bounded by the number of distinct (bigram, byte) pairs at word boundaries—which is bounded by the vocabulary size times the mean word length, minus the overlaps from shared suffixes.

For the 8,048 reified words from Exp L1, the Layer 2 entries are at most $\sum_w (\text{len}(w) - 1)$ for bigram-chained identification. With mean word length ∼5, this is ∼32,000 entries. But the trie structure shares nodes: "the" and "there" share the "th" bigram and possibly the "he" continuation.

The actual number of distinct Layer 2 entries is an empirical question for the scaling experiments.

4.3. Information content

The total model information decomposes as:

$$I(\text{model}) = I(\text{unigram}) + I(\text{bigram} \mid \text{unigram}) \tag{1}$$
$$+ I(\text{trigram chain} \mid \text{bigram}) + I(\text{word} \mid \text{trigram chain}) \tag{2}$$

Each level adds only the *new* information—the deviation from what the lower level predicts. The LPP entries at each level ARE this residual information. The generalized ω records deviations, not absolute counts.

At Layer 1, a bigram entry's weight encodes $\log_2 c(a, b)$. But the *information content* is $\log_2 c(a, b) - \log_2 c(a) - \log_2 c(b) + \log_2 N$, which is the pointwise mutual information. The raw count is decomposable into the marginal (already known from the unigram) plus the deviation.

This suggests that a compressed model representation would store LPP entries as PMI values rather than raw counts. The UM's current ω_0 stores raw log-counts. A "differential ω" would store the difference from the lower-level prediction. This is not yet formalized but is the direction for minimizing model size as the factor tower grows.

5. Scaling Experiments

Before running the full L2 experiment, small-scale measurements:

5.1. Bigram ES statistics

At 10K, 100K, 1M, 10M:

- How many bigrams reach threshold?
- What fraction of byte positions have a bigram event active?
- Bigram ES size as a function of data size (scaling law).

This tells us the Layer 1 cost.

5.2. Trigram chain statistics

With bigram events available:

- How many (prev bigram, output) pairs reach threshold?
- At word boundaries, how many words are uniquely identified by the boundary (prev bigram, byte) event alone (depth 1)?
- How many require depth 2 (two consecutive Layer 2 events)?

- How many require depth 3+?

This tells us the Layer 2 cost and the trie depth needed.

5.3. Suffix collision rate

Among the 8,048 reified words from Exp L1:

- How many share a 3-character suffix? (depth 1 collisions)
- How many share a 4-character suffix? (depth 2 collisions)
- At what suffix length do collisions vanish?

This is computable from the L1 vocabulary alone, without running a model.

5.4. Spelling entropy

For each reified word, measure the conditional entropy $H(\text{byte}_i \mid \text{word}, \text{byte}_{<i})$ at each position within the word. The "the" experiment showed $H \approx 0.07$ for bigram conditioning. How does this generalize across the full vocabulary?

This tells us the residual cost of the backward direction.

6. What We Need to Extend

6.1. Threshold creation on two-input LPPs

The current runner supports two-input LPP entries (from, from2, to, w) as a runtime convenience. Threshold creation currently operates on single-input LPPs. Extending it to two-input LPPs—creating a new event when (from, from2) $\rightarrow$ to reaches threshold—is needed for Layer 2. The SN grammar already has a slot for this:

```
LPP "prev bigram" "output" THRESHOLD 4 "word".
```

with conjunction entries:

```
from1 from2 to weight.
```

6.2. Differential ω

The observation that higher-level LPP entries encode deviations from lower-level predictions suggests a "differential ω": instead of recording $\log_2 c(a, b)$, record $\log_2 c(a, b) -$ prediction from below. The UM's algebra (log-stochastic arithmetic) may need extension to support this cleanly. This is a theoretical question for the paper, not a prerequisite for the experiments.

6.3. Bidirectional replay

The SN round-trip invariant requires that the model can be loaded and run in both directions. For the forward pass, this is the standard f_0. For the backward pass (word → letters), we need to read the LPP "in reverse"—computing $P(a \mid b)$ from the table that was learned as $P(b \mid a)$. This is already supported by the bidirectionality of the joint count table (§6 of the spec), but the runner does not yet implement a reverse-read mode.

7. Predictions

1. The bigram ES will have ∼3,000 events at 10M, growing as $O(\sqrt{N})$ or slower (most bigrams are discovered early).
2. Depth 1 (3-character suffix) will uniquely identify >80% of word tokens. Depth 2 will cover >95%.
3. The total Layer 2 entries will be ∼20,000–40,000, dominated by the long tail of rare words.
4. The backward spelling entropy will be < 0.1 bits/byte for common words, rising to ~ 0.5 for rare words with irregular patterns.
5. Total model size (LPP entries at all layers) will grow sub-linearly with vocabulary size, because the trie shares prefixes and suffixes.

8. Reproduction

The scaling experiments require only the existing `umr` binary and the Exp L1 vocabulary. The suffix collision analysis requires no runner at all—it is a property of the word list.

```
./umr word-discover enwik9 10000000 4 /dev/null   # reuse L1 vocabulary
# Suffix collision analysis: from L1 word list
# Bigram ES scaling: extend umr with bigram threshold creation
# Trigram chain: extend umr with Layer 2 LPP
```

Source: `#lexicon-path-agenda` (research agenda), `#lexicon_embedding_paper` §7–8 (bijection, P-programming), `#umr_spec` §3, §6, §7 (LPP, bidirectionality, threshold creation).