English Context Neuron: Experiment Results

Experiments 2, 3/6, 4, 5.2, Oracle Mixing, “the”, and 100-Word Lexicon

Claude and MJC — February 24–25, 2026

1. Experiment 3/6: Context Neuron SN

1.1. The SN Model

Three event spaces: `byte_output` (256, OUTPUT), `byte_prev` (256), `context` (3: lowercase, uppercase, non-English). Absolute patterns from byte_prev to context (weight 255). Bigram and context LPPs target byte_output. Shift chain carries output to prev.

1.2. Bug fix: ES clearing

`um_step` only cleared byte_output before each step. Intermediate ESs (like context) accumulated support via max-min and were never cleared. Fixed by clearing all ESs except shift targets, then restoring envelope. Backward-compatible: existing models have no intermediate ESs.

1.3. Scoring results

Data size	Bigram only	Bigram + Context	Delta
4K	4.160	4.127	−0.033
64K	4.239	4.292	+0.053
1M	4.059	4.342	+0.283

The context neuron **hurts at scale**. Sharpest-LPP scoring selects the context distribution when its $s_1 - s_2$ gap exceeds the bigram’s, but the context prediction (byte class) is less specific. A model with absolute patterns but no context→output LPP scores exactly 4.059 at 1M (matches bigram-only), confirming the regression comes from scoring competition.

1.4. Learned distributions

Despite the scoring problem, the context LPP learns meaningful transitional distributions. After 10M: lowercase entropy = 4.28 bits (top: t 21%, e 10.5%), non-English entropy = 5.11 bits (top: space 16%, [16%). The model learns $p(\text{next} \mid \text{prev} \in \text{class})$, not unigram letter frequencies — hence ‘t’ is overrepresented via the ‘th’ bigram.

2. Experiment 2: Observation-Only LPP

2.1. Implementation

LPP mode field: `ACTIVE` (default: participates in both f and ω_0), `OBSERVE` (ω_0 only), `FROZEN` (f only). SN syntax: `LPP "from" "to" OBSERVE.`

An OBSERVE LPP participates in ω_0 (learning: creates entries, increments log-stochastic counts) but is excluded from f (forward pass) and scoring. This lets us collect statistics on a distribution without affecting predictions.

2.2. Verification

Post-load state for the context model:

```
LPP 0: byte_prev->context     ACTIVE
LPP 1: byte_prev->byte_output ACTIVE
LPP 2: context->byte_output   OBSERVE
```

After training on 4K: LPP 2 has 291 learned entries (learning happens). Conjunction entries preserved. OBSERVE annotation survives save/load round-trip.

2.3. RNG divergence

Scale	Bigram only	Observe-only	Delta
1K	5072.8 bits	5077.6 bits	+4.8
4K	17036.2	16898.4	−137.8
16K	74164.3	74789.6	+625.3
64K	277812.6	275566.5	−2246.1

The observe-only model does NOT match bigram-only. The sign alternates. Cause: the observe-only LPP's ω_0 learning consumes random numbers, shifting the bigram LPP's learning trajectory. A model with the same ESs and absolute-pattern LPPs but no context→output LPP (neither learning nor scoring) matches bigram-only **exactly** at all scales. The RNG divergence is inherent to shared-RNG online learning.

Implication: observation-only LPPs are not passive probes — they perturb the model via RNG coupling. For clean A/B comparison, use frozen scoring (train separate models, compare frozen scores).

3. Experiment 5.2: The 100-Word SN Model

3.1. Architecture

Four event spaces: `byte_output` (256, OUTPUT), `byte_prev` (256), `context` (2: English, non-English), `word` (100 events, one per top-100 word).

Four LPPs:

- `byte_prev` → `context`: absolute patterns (a-z, A-Z → English; else → non-English).
- `context` → `word`: absolute patterns. English → all 100 word events with support = $\lceil \log_2(\text{freq}) \rceil$. All words activated simultaneously.
- `byte_prev` → `byte_output`: bigram (learned online).

- `word` $\rightarrow$ `byte_output`: conjunction patterns spelling each word at weight 255 (absolute). For word $w = c_1 c_2 \ldots c_n$: (word_w, prev_space) $\rightarrow$ output_c_1, (word_w, prev_c_1) $\rightarrow$ output_c_2, ... Total: 685 entries for 100 words.

Word "the" (support 23, freq 7.8M):

```
(word_the, prev_space) -> output_t  weight 255
(word_the, prev_t)     -> output_h  weight 255
(word_the, prev_h)     -> output_e  weight 255
```

Design error: weight 255 makes these absolute (deterministic) patterns. This is wrong: orthography is distributional, not deterministic ("the" can be misspelled; "th" could continue as "them", "then", "there", etc.). The absolute patterns override the word→output LPP's learned distribution at every position where the conjunction fires, preventing the LPP from contributing its distributional knowledge. The word→output connection should be a learned LPP, not absolute patterns.

3.2. Scoring results

Scale	Bigram only	Word-100	Delta
4K	4.160	4.211	+0.051
64K	4.239	4.157	**−0.082**
256K	4.080	4.118	+0.038
1M	4.059	4.137	+0.078

The word model helps at 64K (−0.082 bpc) but hurts at 256K and 1M. An observe-only variant (word→output marked OBSERVE_ONLY) gives **identical** scores, confirming the word LPP never wins the sharpest-LPP competition — the difference is entirely from RNG divergence.

3.3. Before-and-after comparison

Before training: 685 conjunction entries at weight 255 (word spellings). Zero single-source entries. Word events activated with frequency-based support (17–23).

After training (64K): 685 conjunction entries at weight 254 (capped from 255 by loader). **10,147 single-source entries** from ω_0 learning. These represent $p(\text{output} \mid \text{word active})$ — but since all 100 words fire simultaneously during English text, **all words learn the same distribution**: the English letter marginal.

Top outputs for word "the" (expected: t, h, e, space):

```
  space(17)  n(16)  d(15)  a(15)  r(15)  o(15)  i(14)  m(14)
```

Top outputs for word "of" (expected: o, f, space):

```
  space(16)  n(16)  t(16)  r(16)  e(15)  d(15)  a(15)  s(15)
```

All 100 words converge to the same distribution: the English letter frequency table at log-stochastic scale. The conjunction entries (actual word spellings) are the only word-specific structure; the 10,147 learned entries are uninformative.

3.4. Diagnosis

The word model fails because **all 100 word events fire simultaneously**. The context→word LPP activates every word whenever the context is English. Since all words observe every byte, ω_0 learns the same marginal distribution for all words. The conjunction patterns (word spellings) are drowned out.

The fix requires **selective word activation**: only the current word should be active. This is exactly the settling mechanism — the model must disambiguate which word it's in based on observed bytes. Without settling, word events are broadcast ("I know 100 words exist") rather than selective ("I'm currently in word 'the' ").

4. Summary

Three experiments, one core finding: **the architecture works, the scoring doesn't**.

1. **Exp 3/6**: Context neuron learns distinct distributions per byte class, but sharpest-LPP scoring lets the coarse context beat the fine bigram. Fixed um_step ES-clearing bug.
2. **Exp 2**: Observation-only LPPs work as specified (learn without scoring). RNG coupling means they're not perfectly passive probes.
3. **Exp 5.2**: 100-word model with two compounding errors: (a) absolute conjunction patterns (weight 255) override the word LPP's learned distribution, treating orthography as deterministic when it is distributional; (b) broadcast activation (all words fire in English) means ω_0 learns the marginal, not word-specific distributions. The word LPP never wins sharpest-LPP — the only scoring effect is RNG divergence.

The path forward: selective word activation via settling, or LS subtraction (Exp. 4) to compose distributions algebraically rather than through max-min competition.

5. Experiment 4: Log-Stochastic Algebra

5.1. LSA primitives

Implemented $\oplus$ (LS addition) and $\ominus$ (LS subtraction) in `#umr_core`. Four CLI commands: `lsa-dist`, `lsa-subtract`, `lsa-add`, `lsa-factor`.

5.2. The surgery model

Four event spaces: `byte_output` (256, OUTPUT), `byte_prev` (256), `envelope` (1, "True." always-on), `context` (3). Three LPPs: envelope→output (unigram marginal), byte_prev→context (absolute), context→output (OBSERVE_ONLY).

Trained on 1M: unigram baseline = 5.365 bpc. Per-context distributions: lowercase $H = 4.153$ bits (sharp), non-English $H = 5.031$ bits (flat).

5.3. Promotion vs surgery

Promotion (observe-only → active, no surgery):

Scale	Unigram only	Promoted	Delta
64K	5.323	5.260	$-$**0.063**
1M	5.365	5.245	$-$**0.120**

The per-context distributions are sharper than the unigram marginal, so sharpest-LPP correctly selects the context at most positions. **The English context neuron helps compression by 0.12 bpc** (unigram baseline).

Surgery (`lsa-factor`: LS-subtract per-context from marginal):

Passes	64K bpc	Direction
1	10.308	+4.985
2	14.529	+9.206
5	15.573	+10.250
10	14.070	+8.747

Catastrophic regression: 10.3 bpc after one pass (vs 5.3 baseline). Iteration makes it worse (14.5 at 2 passes).

5.4. LATD: Root Cause

The catastrophic regression is **not** a fundamental problem with LS subtraction — it is an experimental bug.

The envelope→output LPP and context→output OBSERVE LPP learn independently (separate ω_0 calls with separate RNG seeds). Their support values do not maintain the algebraic constraint $s_{\text{context}} \leq s_{\text{marginal}}$:

- **46.7%** of per-context entries have $s_{\text{context}} > s_{\text{marginal}}$
- **18 bytes** appear in a context distribution ($s > 0$) but not in the marginal ($s = 0$)
- Example: byte 'e' has marginal $s = 15$ but lowercase context $s = 17$

When $s_{\text{context}} > s_{\text{marginal}}$, LS-subtract clamps to 0, and the residual LPP has no entry for that byte. After one pass of surgery, 65 of 195 marginal entries are zeroed out. In UM terms, support 0 means *silent* (no pattern has spoken), not *impossible* — the softmax scoring floor ($2^0 = 1$) still assigns nonzero probability. But the concentration of support mass onto the surviving 130 entries distorts the scored distribution severely.

Verification: computing the marginal synthetically via LS-addition of all three context distributions gives **zero violations**. The constraint $s_{\text{marginal}} \geq \max_c s_{\text{context},c}$ holds exactly when the marginal is derived from the per-context distributions rather than learned independently.

Fix: compute the marginal FROM per-context values (LS-add all contexts). Don't learn it independently. This eliminates the constraint violations (0 violations with synthetic marginal).

5.5. Exp 4b: Corrected surgery (synthetic marginal)

Applied the fix: LS-add all three context distributions to produce a synthetic marginal that satisfies $s_{\text{marginal}} \geq \max_c s_{\text{context},c}$ by construction, then LS-subtract per context.

Model	64K bpc	1M bpc	Δ vs unigram (1M)
Unigram (observe context)	5.316	5.365	—
Promoted (no surgery)	5.256	5.245	**−0.120**
Old surgery (indep. marginal)	10.308	—	+4.985
New surgery (synth. marginal)	5.616	5.653	+0.288

The synthetic marginal eliminates the catastrophic regression ($+5.0 \to +0.3$), confirming the LATD diagnosis. The remaining +0.3 bpc is from LS-subtraction information loss: each integer decrement loses ~50% precision in linear terms, so the residual's support values are systematically deflated. The promoted per-context LPP beats the deflated residual everywhere, including at non-English positions where the context distribution ($H \approx 5.1$) is *flatter* than the original marginal ($H \approx 4.9$).

Conclusion: LSA surgery is algebraically correct but operationally dominated by promotion at integer precision. Promotion (−0.120 bpc) is the right approach; surgery adds nothing beyond what promotion already gives.

5.6. Key finding

Context neurons help when sharper than the baseline. The unigram ($H = 4.9$) is flat, so *any* context that separates data into classes will be sharper. The bigram ($H \approx 3$–4) is already sharp, so context ($H \approx 4.2$) rarely beats it. This explains why Exp. 3/6 (bigram + context) regressed while Exp. 4 (unigram + context) improved: the context neuron is useful *between* the unigram and bigram baselines.

6. Oracle Mixing: Context Neuron Ceiling

6.1. Setup

Oracle mixing answers: *how much can the context neuron help, given perfect knowledge of which context applies?* At each position, we know from the data whether the output byte is lowercase, uppercase, or non-English. We use the per-context distribution from the OBSERVE LPP to score that position, rather than the unigram marginal.

This is not a compressor (it uses future information). It measures the **ceiling** — the maximum gain available from the binary English-context ES.

6.2. Per-context distributions

The OBSERVE LPP (context→output, trained on 1M) captures three conditional distributions:

Context	Entropy (bits)	Top bytes
Lowercase	4.153	t (21%), e (10.5%), a (7.3%)
Uppercase	5.048	T (13.5%), S (7.1%), A (6.5%)
Non-English	5.110	space (16%), [(16%), . (6%)

Data coverage: 68.3% lowercase, 3.7% uppercase, 27.9% non-English.

6.3. Oracle ceiling

Scoring	bpc	Delta vs unigram
Unigram (marginal)	5.365	—
Promoted (lagging context)	5.245	**−0.120**
Oracle (true context)	4.961	**−0.404**

The oracle context neuron improves the unigram baseline by 0.404 bpc. Promotion (using the previous byte's class as context, a one-step-lagging indicator) captures 30% of this ceiling (−0.120 bpc).

The 70% gap between promoted and oracle represents positions where the lagging indicator gives the wrong context — primarily word boundaries (space followed by lowercase) and tag transitions. A model that correctly tracks word state would approach the oracle ceiling without requiring future information.

6.4. Interpretation

The oracle experiment validates the context neuron *architecture*: the per-context distributions are genuinely informative, and the 0.404 bpc ceiling is substantial (7.5% of the unigram baseline). The problem is not "does the context neuron help?" but "how do we activate the right context at the right time?" — which is the settling problem from a different angle.

7. The Word "the": First Lexicon Entry

7.1. LATD

Zero misspellings of "the" in enwik9 (Wikipedia is well-edited). The word "the" occurs 6.4M times in 1B bytes (∼69K per 10M). Coverage: 2.07% of bytes are inside a "the" span (3 bytes each).

"the"-prefix words per 10M: their(2156), they(1679), these(1066), there(994), them(982), then(752). Word boundary detection: space, newline, tab, and XML delimiters (`>` before, `<` after) plus standard punctuation.

7.2. Design

Binary event space `word_the` (2 events: "the", "not-the"). **Oracle activation**: at each position, we *know from the data* whether the current byte is inside a "the" span, and set the word event accordingly. This is the same principle as the absolute patterns mapping bytes to context classes — oracle knowledge embedded in the model.

Extended shift chain: `byte_output` → `byte_prev` → `byte_prev2`. Three OBSERVE LPPs capture the spelling at increasing depth:

1. **Unigram**: `word_the` → `byte_output`. "What bytes appear when 'the' is active?"
2. **Bigram**: `byte_prev` → `byte_output`, conjunction with `word_the`. "Given the previous byte *within the word*, what's next?"
3. **Trigram**: `byte_prev2` → `byte_output`, conjunction with `word_the`. "Given the byte two steps back *within the word*, what's output?"

Word boundary clearing: The bigram and trigram OBSERVE LPPs only see within-word context. At the first position of the word (position 0, the 't'), the shift chain still holds the previous word's bytes — the conjunction is *not observed*. At position 1 ('h'), `byte_prev` = 't' is within-word, so the bigram is observed. At position 2 ('e'), both `byte_prev` and `byte_prev2` are within-word, so both bigram and trigram are observed. This prevents the previous word's ending from leaking into the spelling distribution.

The scoring bigram LPP (ACTIVE) does all prediction. All three word LPPs are OBSERVE-only. Support value for oracle activation: $\lfloor \log_2(\text{count}) \rfloor$ (e.g., 17 at 10M).

7.3. Neutrality

Data size	the-word bpc	Bigram-only bpc	Delta
1M	4.166	4.172	−0.006
10M	4.136	4.140	−0.004

Deltas are within LSI variance (±0.02 bpc). All three OBSERVE LPPs are confirmed **neutral**: adding the word event and its three observation-only LPPs does not change the model's scoring.

7.4. Depth 1: Unigram spelling

The word_the → byte_output OBSERVE LPP learned 3 entries:

Byte	Support	Prob.	Note
`t`	15	0.25	
`h`	16	0.50	$\approx \frac{1}{3}$ each
`e`	15	0.25	

Entropy: 1.40 bits (theoretical uniform $\frac{1}{3}$ = 1.585 bits). The marginal is uniform because "the" is always spelled t-h-e (zero misspellings) and each occurrence contributes one observation of each letter. The slight asymmetry (`h` one increment ahead) is log-stochastic noise.

7.5. Depth 2: Bigram spelling

The conjunction OBSERVE LPP (byte_prev $\wedge$ word_the $\rightarrow$ byte_output) has exactly 2 entries at 10M — purely within-word:

Prev byte	$\rightarrow$ Output	Prob.	Entropy
`t`	`h`	0.996	0.068 bits
`h`	`e`	0.996	0.068 bits

These are the letter bigrams `th` and `he` from the bag-of-letters paper — the complete within-word bigram spelling of "the". Near-deterministic because "the" has zero misspellings. The boundary transition (space $\rightarrow$ t) is correctly excluded by word boundary clearing: at position 0 the shift chain holds the previous word's context.

7.6. Depth 3: Trigram spelling

The conjunction OBSERVE LPP (byte_prev2 $\wedge$ word_the $\rightarrow$ byte_output) has exactly 1 entry at 10M:

Prev2 byte	$\rightarrow$ Output	Prob.	Entropy
`t`	`e`	0.998	0.036 bits

This is the letter trigram `the` from the bag-of-letters paper: "two bytes after t, the output is e." Only fires at position 2 of the word, where both `byte_prev2` = 't' and `byte_prev` = 'h' are within-word. Previous versions had ~65 entries because the shift chain leaked the previous word's ending across the boundary — word boundary clearing eliminates this entirely.

The trigram result is the most striking: a single entry at 0.036 bits of entropy. Conditioned on being inside "the" and knowing the byte two steps back was 't', the output is 'e' with 99.8% probability. For "the", the trigram is the word itself.

7.7. Interpretation

This is the first correct lexicon entry in the UM. Four key points:

1. **The word IS a distribution.** MJC (20260214): "the" is "a distribution over sets of events including the typos" — even though there are zero typos in enwik9, the machinery correctly learns the distributional projection. A word with misspellings would show them in the OBSERVE LPP.
2. **Oracle activation is justified.** We know which positions are "the" from the data, same as we know which bytes are lowercase. The imperfect pattern-based detector comes AFTER we know what the correct distribution looks like.
3. **The OBSERVE LPP is the right tool.** Previous Exp 5.2 used weight-255 conjunction patterns (deterministic, overrides other LPPs) — **wrong**. The OBSERVE LPP captures the *distributional* spelling without interfering with scoring.

4. **Depth separates marginal from conditional.** The unigram says "{t,h,e} equally likely" (H=1.5 bits). The bigram says "after t comes h" (H=0.07 bits). The trigram says "the word IS 'the'" (H=0.04 bits, 1 entry). Word boundary clearing ensures each depth sees only within-word context. The shift chain provides depth naturally — no special machinery required.

The conjunction OBSERVE LPP with word boundary clearing is the mechanism for per-word conditional distributions. Standard `um_lpp_observe` captures marginals (unigram). The `from2` field on LPP entries captures conjunctions (bigram, trigram): the source is a byte-level event, the secondary source is the word event, and the target is the output byte. This is exactly the bag-of-letters structure from #bag_of_letters: unigram events {t, h, e}, bigram events {th, he}, trigram event {the}. The OBSERVE LPPs learn these distributions without any additional ES creation.

Template for the 100-word experiment. The previous Exp 5.2 failed because the word ES was not an ES — broadcast activation meant all words learned the same marginal. Oracle word selection fixes this: each word gets its own activation, its own OBSERVE LPPs at all three depths, and word boundary clearing prevents cross-word contamination. The only change for scaling from 1 word to 100: the word_the ES grows from 2 events to |vocab| + 1 events, with oracle activation selecting the correct word at each position.

SN model: `/tmp/the-word-10000K.sn` (5 ESs, 4 LPPs, 771 events).

8. The 100-Word Lexicon: Corrected Exp 5.2

8.1. LATD

The original Exp 5.2 (§4) failed because all 100 word events fired simultaneously, causing ω_0 to learn the English letter marginal for every word. MJC's diagnosis: "the problem was that Word ES was not an ES" — in UM terms, an ES requires that exactly one event be active. Broadcast activation violated this by firing all 100 words in English context.

The fix is **oracle word selection**: at each position, we know from the data which word (if any) the current byte belongs to. Only that word's event is active. This is identical in principle to the "the" experiment (§7), scaled from 1 word to 100.

8.2. Design

Word selection. Pre-scan the data for the top 100 words by frequency. A word is a maximal run of alphabetic bytes (a-z, A-Z), lowercased for matching. At each byte position, we know: (a) whether it's inside a word, (b) which word it belongs to. Non-top-100 words get "other" (event 100).

Architecture. Five event spaces: `byte_output` (256, OUTPUT), `byte_prev` (256), `byte_prev2` (256), `word` (101: 100 words + "other"), `envelope` (1, always-on). Extended shift chain: byte_output → byte_prev → byte_prev2.

Four LPPs:

1. `byte_prev` → `byte_output`: bigram (ACTIVE, learned online). Does all prediction.
2. `word` → `byte_output`: unigram OBSERVE. "What bytes appear when word w is active?"
3. `byte_prev` → `byte_output`: bigram OBSERVE, conjunction with word. "Given the previous within-word byte, what's next?"

4. `byte_prev2` $\rightarrow$ `byte_output`: trigram OBSERVE, conjunction with word. “Given the byte two steps back within the word, what’s output?”

Word boundary clearing as in §7: bigram conjunction observed only at word position ≥ 1, trigram only at position ≥ 2. Oracle activation sets the word event at each position.

Triple hash. With 100 words sharing the same byte-level LPPs, conjunction entries need a 3-way key: (byte_prev, word, byte_output). A standard 2-key hash would collide when different words share the same (prev, output) pair (e.g., “th” in “the” and “that”). The triple hash `um_triple_hash(from1, from2, to)` includes the word event in the FNV-1a computation.

8.3. Results at 10M

	Value	Note
Top 100 coverage	17.0%	of all bytes inside a top-100 word
Unique words found	73,055	alpha-only, lowercased
Scoring	4.106 bpc	vs 4.14 bigram-only (neutral)
LPP entries	17,745	across all 4 LPPs

8.4. Per-word unigram distributions

Each word now learns **its own** orthographic distribution, not the English marginal. Selected words:

Word	Count	Entries	Top letters	H (bits)
the	81,423	6	h(36%) t(36%) e(18%) T(9%)	1.84
of	51,458	4	o(66%) f(33%)	1.04
and	32,803	6	a(33%) d(33%) n(33%)	1.69
in	29,475	4	n(61%) i(30%) I(8%)	1.38
a	24,636	2	a(96%) A(3%)	0.32
that	8,221	6	t(57%) a(28%) h(14%)	1.53
with	7,267	5	h(49%) i(24%) t(12%) w(12%)	2.01

Contrast with old Exp 5.2. The old model showed all words converging to `space(17) n(16) d(15) a(15) r(15)` — the English letter marginal. Now “the” shows {t, h, e, T} and “of” shows {o, f}. Each word’s unigram is its bag of letters.

Case variants. Words with uppercase variants (sentence-initial “The”, “In”, “A”) show the capital letter as an additional entry. “the” has 6 entries (t, h, e, T, H, E) because “The” contributes T, H, E while “the” contributes t, h, e. The capital entries have lower support (9% for T vs 36% for t), reflecting that “The” is $\sim 4\times$ less frequent than “the”.

Entropy tracks word length. Single-letter words have low entropy: “a” at H=0.32 (the a/A split), “s” at H=0.73. Two-letter words: “of” at H=1.04, “to” at H=1.04. Three-letter words: “the” at H=1.84, “and” at H=1.69. Four-letter words: “that” at H=1.53, “with” at H=2.01. The theoretical $\log_2(n)$ for uniform distribution over n distinct letters matches approximately: $\log_2(3) = 1.58$ for 3-letter words, $\log_2(4) = 2.0$ for 4-letter words with all unique letters. “that” (H=1.53) is below the 4-letter uniform because ‘t’ appears twice, concentrating 57% of mass.

8.5. Per-word bigram distributions

Bigram entry counts per word (selected):

Word	Length	Bigrams	Expected
the	3	5	2 × (case)
of	2	3	1 × (case)
and	3	5	2 × (case)
in	2	3	1 × (case)
a	1	0	0 (no within-word bigram)
s	1	0	0 (no within-word bigram)
to	2	4	1 × (case)
that	4	—	3 × (case)

Single-letter words correctly have zero bigram entries (word boundary clearing excludes position 0). Two-letter words have ∼1 within-word bigram plus case variants ("of" → `o→f, O→f`). Three-letter words have ∼2 bigrams plus variants ("the" → `t→h, h→e, T→h, H→e`). The extra entries beyond the lowercase-only count come from sentence-initial capitalization.

8.6. Comparison: old vs new Exp 5.2

	Old Exp 5.2 (§4)	New (100-word lexicon)
Activation	Broadcast (all 100)	Oracle (1 at a time)
LPP type	Weight 255 absolute	OBSERVE (distributional)
Word boundary	None	Cleared at position 0
ω_0 learns	English marginal	Per-word spelling
Entries (1M)	10,147 (all identical)	17,745 (word-specific)
Scoring	Neutral (RNG noise)	Neutral (OBSERVE)

The key difference: oracle activation makes the word ES a *true* event space (one event active at a time), so ω_0 learns $p(\text{output} \mid \text{word} = w)$ rather than $p(\text{output} \mid \text{English})$. The result is a 100-word lexicon where each entry captures its own orthographic projection at three depths.

8.7. Oracle word bigram ceiling

How much would it help to use the per-word bigram distribution instead of the general bigram? At within-word positions (depth ≥ 1) in a top-100 word, we score using the per-word conjunction bigram OBSERVE LPP instead of the general bigram, taking the *better* of the two. This is an oracle ceiling: it uses future information (which word we're in) that a real compressor wouldn't have.

Scale	Standard	Oracle	Δ	Positions used
100K	4.443	4.302	−**0.140**	9,569 (9.6%)
1M	4.178	3.920	−**0.258**	114,584 (11.5%)
10M	4.106	3.785	−**0.321**	1,083,907 (10.8%)

A 0.321 bpc ceiling from 100 words at 10M. The oracle gain scales with data: -0.14 at 100K $\rightarrow$ -0.26 at 1M $\rightarrow$ -0.32 at 10M, roughly doubling per decade. This is because the per-word bigram OBSERVE LPP approaches deterministic with more observations: at 10M, "the" has ∼80K occurrences, making $p(\texttt{h} \mid \texttt{t}, \text{word=the}) \approx 1$.

Coverage and gain structure. The oracle fires at ∼11% of positions (within-word depth ≥ 1 for top-100 words). At these positions, the per-word bigram is near-deterministic ($H \approx 0.07$ bits) while the general bigram has $H \approx 3$ bits. The average saving per oracle position: ∼3 bits. This is the spelling information that the lexicon captures.

Comparison with context neuron ceiling. The context neuron oracle ceiling was -0.404 bpc off a 5.4 bpc unigram baseline (7.5% gain). The word bigram oracle ceiling is -0.321 bpc off a 4.1 bpc bigram baseline (**7.8% gain**). Proportionally, the word lexicon is equally powerful despite operating on a much stronger baseline. And this is only 100 words at bigram depth — trigram depth and more words would increase the ceiling further.

Max-min can't deliver this. The forward pass (max-min) merges all LPP contributions non-subtractively. Making the conjunction LPPs ACTIVE would boost the correct byte's support but cannot suppress the incorrect bytes contributed by the general bigram. The gap between general-bigram scoring and oracle-word-bigram scoring is exactly the marginal dominance problem. The fix operates at the distribution level (selecting which LPP's distribution to use), not the support level.

8.8. How much can prefix matching capture?

Without oracle word knowledge, the simplest approach is **prefix matching**: as we process bytes within a word, we track which top-100 words match the prefix seen so far. Once the prefix uniquely identifies a word, we use its per-word bigram distribution for subsequent positions.

For each word, we compute the *unique prefix length* (UPL): the shortest prefix that uniquely identifies it among the top 100. Capturable positions are those at depth $\geq$ UPL (where we know the word from the prefix alone).

Word	Len	UPL	Capturable/Oracle	Note
the	3	>3	0/2	"the" prefixes 7 words
of	2	2	0/1	unique at end, nothing left
and	3	3	0/2	"and" unique but at end
quot	4	1	3/3	'q' unique → all captured
with	4	2	2/3	"wi" unique → 2 of 3
from	4	2	2/3	"fr" unique
american	8	3	5/7	"ame" unique
category	8	3	5/7	"cat" unique

Measured prefix gain. The prefix matcher was implemented and tested. At each within-word position, it tracks which top-100 words match the observed prefix. When uniquely identified, it uses the per-word bigram OBSERVE distribution for scoring.

Scale	Standard	Oracle	Prefix	Δ Oracle	Δ Prefix	Capture
100K	4.443	4.302	4.404	−0.140	−0.039	27.9%
1M	4.178	3.920	4.067	−0.258	−0.111	43.1%
10M	4.106	3.785	3.980	−0.321	**−0.126**	39.3%

The prefix matcher captures **39% of the oracle ceiling at 10M**: -0.126 bpc via 394K positions (vs 1.08M oracle). This is a causal predictor — it only uses bytes seen so far.

The 61% gap comes from: (a) short common words sharing prefixes ("the" prefixes 7 top-100 words, "in" shares with "is"/"it"/"id"); (b) 2-letter words unique only at the end, leaving no positions to exploit. "the" alone accounts for 163K uncapturable oracle positions (14.5% of total oracle).

Capturing more of the ceiling requires weighting over multiple candidate words rather than waiting for unique identification.

8.9. Bayesian word mixture

A Bayesian mixture over candidates: at each within-word position k, compute $P(\text{byte}_k) = \sum_{w \in C_k} P(w \mid \text{prefix}) \cdot \mathbb{I}[w[k] = \text{byte}_k]$, where C_k is the set of top-100 words matching the observed prefix, and $P(w \mid \text{prefix}) \propto \text{count}_w$. This is causal (uses only bytes seen so far) and applies at *all* word positions including position 0.

Scale	Standard	Oracle	Prefix	Mixture	Online	Δ Online
100K	4.443	4.302	4.404	3.527	3.533	$-$**0.910**
1M	4.178	3.920	4.067	3.541	3.539	$-$**0.639**
10M	4.106	3.785	3.980	3.555	3.554	$-$**0.552**

"Mixture" uses pre-scanned word frequencies from the full data. "Online" learns word frequencies causally, incrementing counts as words complete. Both use the same Bayesian candidate weighting.

The mixture beats the oracle ceiling. At 10M, the mixture gives -0.552 bpc (vs -0.321 oracle). This is not a contradiction: the mixture and oracle measure different things.

The oracle replaces the general bigram at within-word positions (depth ≥ 1) for top-100 words. It fires at 1.1M positions.

The mixture predicts at *all* word positions (2.5M positions), including position 0 of every word. At position 0, the word frequency distribution is much sharper than the general bigram: $P(\texttt{t} \mid \text{word start}) \approx 0.12$ (concentrated on common first letters) vs $P(\texttt{t} \mid \text{space}) \approx 0.04$ (spread across all bytes). This is the **first change of base**: at word boundaries, word-frequency-weighted prediction beats byte-frequency prediction.

Scaling. The mixture gain decreases with data (-0.92 at 100K $\rightarrow$ -0.55 at 10M) because the general bigram improves faster than the word predictor. At 100K, the bigram barely captures letter transition statistics, while the word predictor already benefits from stable word frequencies. At 10M, the bigram is strong (4.1 bpc) but the word predictor still provides complementary information.

Online = oracle. Replacing pre-scanned word frequencies with online counts (Laplace-1 prior, incremented on word completion) loses nothing measurable. At 10M: online -0.552 = mixture -0.552. At 100K the gap is 0.005 bpc; at 1M the online version *slightly wins* (-0.002). Word frequencies converge so fast (top-100 words appear thousands of times in the first 100K bytes) that online estimation is free. The word mixture is now a **real compressor component**: causal, online, no lookahead. It captures **13% of the bigram baseline**.

8.10. Against KN-6

The previous measurements used a simple bigram (4.1 bpc at 10M) as the baseline. The real question: does the word mixture help against KN-6 (2.7 bpc at 100K, ∼1.8 bpc at 1B)?

At 100K with learned sigmoid mixture weight:

Model	bpc	Δ
KN-6 alone	2.719	—
KN-6 + word mixture	2.608	**−0.111**

The word mixture gives −0.111 bpc (4.1%) against KN-6 at 100K. The learned mixing weight converges to $w = 0.267$ (27% word model, 73% KN-6). The word model alone is terrible (17.2 bpc) because it only predicts one byte per position per candidate word; the value comes entirely from the mixture.

Why does the word mixture help against KN-6? KN-6 uses 6-byte contexts. Inside common words, 6 bytes of context often suffice to predict the next byte (e.g., "`anarch`" → "`i`" with high probability). But at word boundaries (position 0 of a new word), the context is the end of the *previous* word plus whitespace—which tells you less about what word is starting. The word mixture's "change of base" concentrates probability on the first letters of common words, which KN-6 must learn from the context of each preceding word separately.

Where the gain comes from:

Position	KN-6 bpc	Mixed bpc	Gain	N
0 (word start)	5.135	4.592	+**0.543**	14144
1	2.604	2.507	+0.097	13689
2	2.518	2.455	+0.063	11372
3	2.190	2.101	+0.089	9175
4–9	1.9–2.1	—	+0.01–0.03	—
10+	—	—	0.000	—

Position 0 contributes 49% of the total gain. This is the "change of base": KN-6 scores 5.1 bpc at word boundaries (6 bytes of preceding context only partially constrain the next word), while the word mixture concentrates probability on common first letters (−0.54 bpc).

Per category: top-100 words gain +0.242 bpc, other words +0.102 bpc (the mixture helps even for out-of-vocabulary words via prefix filtering), non-words 0.000 (as expected).

Open question. Does the gain persist at scale? At 1B, KN-6 is 1.784 bpc. The 100K gain of −0.111 represents 4.1% of KN-6; if this ratio holds, the 1B gain would be ∼0.073 bpc. The hutter_score16 baseline (KN-6 + match + sparse = 1.588 bpc) already captures some word-boundary effects via the match model, which may reduce the word mixture's added value.

8.11. UM architecture for word mixture

How would the word mixture be expressed as a UM P-program? The mixture requires:

1. A word event space (E_{word}, 100 events) with *multiple simultaneous activations* at word boundaries, weighted by log-frequency.

2. A suppression mechanism: as bytes arrive, incompatible word events lose support.

3. An output that weights byte predictions by surviving word supports.

Step 1 is standard UM: word events activate at word boundaries.

Step 2 requires a new operation. In the current UM, support only increases via f (max-min patterns set support). Suppression (zeroing incompatible events) is not expressible in max-min because max-min is monotone non-decreasing. This is the same problem as settling: the forward pass cannot *narrow* a distribution, only broaden it.

Step 3 IS the settling problem. Max-min across 100 word patterns produces $\text{sup}[\text{byte}] = \max_w \min(\text{sup}[w], \text{spelling}[w]$ This is *winner-take-all*: the most frequent surviving word dominates all byte predictions. The mixture instead computes $P(\text{byte}) = \sum_w P(w) \cdot \mathbb{I}[w[\text{pos}] = \text{byte}]$, which is a *weighted sum*, not a max.

The fundamental gap: max-min $= L_\infty$ norm; mixture $= L_1$ norm. The word mixture demonstrates that L_1 (probability-weighted average) outperforms L_∞ (max) for multi-hypothesis prediction. The settling conjecture from the timing resolution paper is confirmed: the fix must operate at the distribution level.

Proposed solution: replace max-min output aggregation with *sharpness-weighted mixture* for competing P-programs. Each P-program (word model, KN-6, match) produces a distribution; the final output is a weighted combination where weights are learned online (sigmoid mixing) or entropy-derived (sharper model gets more weight). This is exactly what the word-kn6 experiment implements externally; the architectural question is how to express it within f.

9. Updated Summary

Seven experiments, the 100-word lexicon, oracle ceiling, Bayesian word mixture, and KN-6 integration. Eight core findings:

1. **Context neurons work**, but only when sharper than the baseline LPP. Gain: -0.12 bpc (unigram promotion), -0.404 bpc (oracle ceiling). The context neuron sits between unigram and bigram in sharpness.

2. **LSA subtraction**: the catastrophic regression was an experimental bug (independent ω_0, 54 constraint violations). Fixing it (synthetic marginal) reduces regression from $+5.0$ to $+0.3$ bpc. But LSA surgery is operationally dominated by simple promotion at integer precision.

3. **The first lexicon entry works at three depths.** The word "the" with oracle activation and word boundary clearing correctly learns unigram ({t,h,e}, H=1.5), bigram ({th, he}, H=0.07), trigram ({the}, H=0.04). Neutral scoring confirmed.

4. **Oracle word selection fixes broadcast activation.** The 100-word lexicon demonstrates that oracle activation (one word at a time) makes ω_0 learn per-word distributions instead of the English marginal. Each word's unigram matches its bag of letters, with case variants. This is the corrected Exp 5.2.

5. **The oracle word bigram ceiling is -0.321 bpc at 10M.** Using per-word bigram distributions at within-word positions (11% of bytes) instead of the general bigram yields 3.785 bpc (vs 4.106 standard). The gain scales with data (-0.14 at 100K, -0.26 at 1M, -0.32 at 10M) and is proportionally as large as the context neuron ceiling (7.8% vs 7.5% of respective baselines).

6. **Marginal dominance blocks max-min delivery.** The ceiling cannot be reached via the forward pass (max-min) because conjunction LPPs boost correct bytes but cannot suppress incorrect bytes from the general bigram. The fix must operate at the distribution level, selecting which LPP to use rather than merging them.

7. **Bayesian word mixture gives -0.552 bpc at 10M.** A frequency-weighted mixture over candidate words exceeds the oracle ceiling (172% of oracle) by predicting at *all* word positions including position 0. This is the first **change of base**: at word boundaries, word-frequency-weighted prediction beats byte-frequency prediction. The word mixture captures 13% of the bigram baseline. Online word frequency estimation (Laplace-1, causal) loses nothing: this is a real compressor component.

8. **Word mixture helps KN-6 by -0.111 bpc at 100K (4.1%).** The word mixture provides complementary information to KN-6, primarily at word boundaries where KN-6's byte context is less informative than the word frequency distribution. The learned mixing weight is $w = 0.27$ (27% word, 73% KN-6).

Remaining experiments: 1 (discover from data), 5b (selective word activation without oracle), 7 (tags), 8 (protocol).