The Unigram Memory Trace

Product Encoding, Quotients, and the LSI Prior

Claude and MJC — February 22, 2026

1. The Simplest Memory Trace

Consider the simplest possible model of enwik9: count each byte's occurrences, predict from the marginal distribution. No context, no LPPs, no shift chain. One event space (byte, 256 events), one envelope.

After observing $n = 10^9 \approx 2^{30}$ bytes under log-stochastic increment (LSI), each byte b has log-support $s_b \approx \log_2(\text{count}_b)$. The envelope support is $T_{\text{env}} \approx 30$: the organism has experienced $\sim 2^{30}$ moments of existence.

The **memory trace** is 256 bytes: the log-support values $s_0, s_1, \ldots, s_{255}$. This is the organism's compressed record of 10^9 observations.

1.1. Empirical Results

On enwik9 (10^9 bytes), 206 of 256 byte values are observed ($s > 0$). The top entries:

Byte	s_b	$s_b - T_{\text{env}}$	$p(b)$
`e`	27	-3	0.121
`o`	27	-3	0.121
`(space)`	26	-4	0.061
`i, l, r, t`	26	-4	0.061
`a, c, h, m, ]`	25	-5	0.030

The softmax distribution $p(b) = 2^{s_b} / \sum 2^{s_b}$ gives the unigram model. Scoring enwik9 under this frozen distribution: **5.448 bpc** (681 MB). Shannon entropy with exact counts: 4.926 bpc (616 MB). The gap is 0.522 bpc.

2. Product Encoding Instead of Arithmetic Coding

Arithmetic coding encodes a sequence by subdividing $[0, 1)$ according to symbol probabilities. It produces a single binary fraction — opaque, sequential, not randomly accessible. You must decode from the beginning.

The product encoding is the alternative. Each byte value b gets a prime p_b. The observation of byte x_t at position t contributes p_{x_t} to a running product:

$$M = \prod_{t=1}^{n} p_{x_t} = \prod_{b=0}^{255} p_b^{\text{count}_b}$$

M is a single (enormous) integer. Its prime factorization *is* the frequency table: the exponent of p_b tells you how many times byte b appeared.

2.1. Readability

The product M is readable in the middle. Given M and a position t, we can determine x_t without decoding from the beginning — we need only the partial product context and modular arithmetic on M. More precisely: if we augment M with positional structure (see §**??**), the prime factorization at any position is locally extractable.

This is the fundamental advantage over arithmetic coding: the product is a *number*, not a bitstream. Numbers have algebraic structure. Bitstreams do not.

2.2. Huffman vs. Arithmetic: Vocabulary Size

Huffman coding assigns integer-length codes to symbols. Its overhead relative to entropy is bounded by 1 bit per symbol for binary alphabets, but decreases as $O(1/\log|\Sigma|)$ for vocabulary size $|\Sigma|$. With $|\Sigma| = 256$, the Huffman overhead is small (< 0.01 bpc for typical distributions).

The product encoding is analogous to Huffman in that each symbol maps to a fixed "code" (a prime), but the composition rule is multiplication rather than concatenation. The overhead depends on the prime assignment: assigning smaller primes to more frequent bytes minimizes the product's size.

3. The Quotient

The quotient for byte b measures deviation from uniform:

$$Q_b = \frac{p(b)}{1/256} = 256 \cdot p(b) = \frac{256 \cdot 2^{s_b}}{\sum 2^{s_b}}$$

In log-support terms, this is directly readable:

$$s_b - T_{\text{env}} \approx \log_2\left(\frac{\text{count}_b}{n/256}\right) = \log_2\left(\frac{\text{freq}_b}{\text{uniform}}\right)$$

The column $s_b - T_{\text{env}}$ in the memory trace tells you, for each byte, how much more or less frequent it is than uniform. This *is* the quotient, stored as one integer per byte.

3.1. Decomposition

The total information in the raw sequence decomposes:

$$\underbrace{8.000}_{\text{uniform}} = \underbrace{(8-H)}_{\text{quotient saving}} + \underbrace{H}_{\text{residual (ordering)}}$$

With exact counts: $H_0 = 4.926$ bpc, quotient saves 3.074 bpc (384 MB).

With LSI: frozen = 5.448 bpc, quotient saves 2.552 bpc (319 MB).

The compressed memory trace has two parts:

1. **Model** (the quotient): 256 bytes. The log-support values.
2. **Residual** (the ordering): 5.448 bpc $\times$ 10^9 / 8 = 681 MB. Which specific byte at each position, given the frequencies.

Total: 256 bytes + 681 MB $\approx$ 681 MB.

4. The LSI Gap Is Not Overfitting

The LSI distribution gives 5.448 bpc; exact counts give 4.926 bpc. The gap is 0.522 bpc (65 MB). Where does it come from?

LSI stores $s_b \approx \log_2(\text{count}_b)$, giving $\sim$30 distinct support levels for 206 active bytes. Many bytes with different true frequencies share the same support value: `e` (12.8%) and `o` (7.0%) both get $s = 27$. The distribution is coarser than reality.

This is **not a bug**. For pure compression, you would use exact counts (memorize the dataset perfectly). But the UM is building a *model*, not a lookup table. The LSI prior says: "I have seen this byte roughly 2^s times." It does not distinguish between 95 million and 70 million occurrences — both round to $2^{27} \approx 134$ million.

This is underfitting, not overfitting. The model generalizes: it would give sensible predictions on new Wikipedia data, because it has learned the *scale* of each byte's frequency without memorizing the exact count. For the Hutter Prize (compress this specific dataset), exact counts would be better. For a model of English, LSI is correct.

The gap quantifies the cost of generalization: 0.522 bpc per byte, or 65 MB total. This is the price of having a model that transfers, expressed in bits.

5. Positional Structure

The product $M = \prod p_b^{\text{count}_b}$ encodes the *multiset* (bag of bytes). To encode the *sequence*, we need positional information.

Under E$\to$N, position t gets its own prime p_{time}, and the state at position t is:

$$S_t = p_{\text{time}}^t \cdot p_{x_t}^1$$

The full product is $\prod_t S_t = p_{\text{time}}^{n(n-1)/2} \cdot \prod_b p_b^{\text{count}_b}$, which conflates time information. More usefully, we keep the S_t as a *sum*:

$$P(E) = \sum_{t=1}^{n} p_{\text{time}}^t \cdot p_{x_t}$$

This is the E$\to$N encoding from `#event_arithmetic`. Each term is distinct (the p_{time}^t factor is unique per position). The prime factorization of any term reads off that position's byte value. Random access is immediate: extract the coefficient of p_{time}^t.

The compressed representation is then:

1. The frequency table (256 bytes, the quotient).

2. The sequence encoding: $P(E)$ modulo the frequency information.

The quotient factors out the "expected" product (uniform frequencies), leaving a residual that encodes only the deviation. This residual is smaller than the original by exactly the quotient's contribution: 2.552 bpc with LSI, 3.074 bpc with exact counts.

6. Summary

1. The unigram memory trace is 256 bytes: log-support values under LSI.
2. $T_{\text{env}} = 30$: the organism has observed $\sim 2^{30}$ time steps.
3. The quotient $s_b - T_{\text{env}}$ is directly readable as log-frequency-ratio.
4. Product encoding (E$\rightarrow$N) replaces arithmetic coding's opaque bitstream with an algebraically structured integer, readable in the middle via prime factorization.
5. The LSI gap (0.522 bpc) is the cost of generalization, not a deficiency. The model underfits the dataset, which is correct behavior for a model (though not for a pure compressor).
6. Total compressed size: model (256 B) + residual (681 MB) = 681 MB.
7. The path forward: add context (bigrams, trigrams, ...) to reduce the residual. Each level of the factor tower captures more structure in the joint events, shrinking the residual toward the Hutter Prize target.