Exploring the Prime Encoding: Eight Experiments on 1024 Bytes

Claude

10 February 2026

Abstract

The prime power encoding maps event spaces to primes and event values to exponents, projecting each position's state into a single integer. We apply this encoding to 1024 bytes of enwik9 (52 distinct characters) and run eight experiments that build intuition for the encoding's properties. The dataset becomes a 497-digit integer $P(E)$. We show: GCD structure reveals shared state; transition ratios characterize dynamics; modular arithmetic performs pattern matching; quotient classes achieve 19.7× compression; cross-position products separate bigram contexts; the encoding carries 1651 bits; residues mod small primes project the data; and the first-occurrence alphabet ordering is 24% suboptimal vs. frequency-optimal assignment.

1 Setup

We use 1024 bytes of enwik9 beginning with `<mediawiki xmlns=...`, containing 52 distinct byte values. Two event spaces are assigned primes:

Event Space	Prime	Size	Range of exponents
Input character	$p_1 = 2$	52 events	1–52
Time (position)	$p_2 = 3$	1024 events	1–1024

The input character ES uses *first-occurrence ordering*: the first byte in the data (`<`) gets event ID 1, the second distinct byte (`m`) gets 2, etc. This ordering is itself an embedding of the alphabet—it encodes when each character first appears.

Each position t encodes as $S_t = 2^{\text{eid}(t)} \cdot 3^t$, and the full encoding is $P(E) = \sum_{t=1}^{1024} S_t$, a 497-digit (1651-bit) integer.

2 Experiment 1: GCD Across Time Steps

Question: What does $\gcd(S_t, S_{t-1})$ reveal?

Since both terms contain powers of 3, the GCD is always nontrivial: $\gcd(S_t, S_{t-1}) \geq 3^{t-1}$. This is uninteresting—the time prime dominates.

More revealing: the *timeless residual* $R_t = S_t/3^t = 2^{\text{eid}(t)}$. Since all R_t are powers of 2, $\gcd(R_t, R_{t-1}) = 2^{\min(\text{eid}(t),\text{eid}(t-1))} \geq 2$ always.

Lesson: With only one non-time prime, GCD is trivially informative. The experiment becomes meaningful with 2+ content primes (e.g., adding neuron ESes). Then $\gcd(R_t, R_{t-1}) > 1$ only when positions share a specific event value—GCD detects *which facts persist* between time steps.

Application: In the full encoding with 128 neuron primes, $\gcd(R_t, R_{t-1})$ would factor into exactly the primes for neurons that didn't change between steps. The factorization of the GCD IS the list of persistent state bits.

3 Experiment 2: Transition Ratios

Question: What is the "velocity" in event space?

The transition ratio $S_{t+1}/S_t = 2^{\Delta\text{eid}} \cdot 3$, where $\Delta\text{eid} = \text{eid}(t+1) - \text{eid}(t)$. The exponent difference Δ is the "step" in input-character space.

Δ	Count	Interpretation
$+1$	171	Adjacent characters in first-occurrence order
0	103	Same character repeated
-8	46	Jump back 8 positions in the alphabet
-6	39	Jump back 6
$+5$	34	Jump forward 5
-4	33	Jump back 4
$+10$	31	Jump forward 10

71 distinct deltas out of a possible range $[-47, +41]$. The most common transition is $\Delta = +1$ (171 times), meaning the next character has the next-higher event ID. This is an artifact of the first-occurrence ordering: characters that often co-occur tend to have adjacent event IDs.

Lesson: The transition ratio converts dynamics into multiplicative structure. Repeated ratios = repeated transitions = patterns. The ratio $2^0 \cdot 3 = 3$ (same character) appears 103 times, giving the character-repetition rate directly from the prime factorization.

Application: Collecting all 1023 transition ratios gives a compact summary of the data's character-level dynamics. Clustering the ratios would reveal character transition classes. In the full encoding with neuron primes, each transition ratio's factorization would show which neurons flipped.

4 Experiment 3: Modular Pattern Matching

Question: Can we find character occurrences using only integer divisibility?

To find all positions where the input is a specific character with event ID k: test whether S_t is divisible by 2^k but not 2^{k+1}.

Pattern	Event ID	Modular count	Direct count
input = ' ' (space)	9	127	127
input = 'e'	3	107	107

Both methods agree perfectly.

Lesson: Pattern matching in the prime encoding reduces to divisibility testing. The pattern "input is space" becomes the purely arithmetic predicate $2^9 \mid S_t \wedge 2^{10} \nmid S_t$. No string operations, no character comparisons—just integer arithmetic on the encoded state.

Application: Multi-position patterns become simultaneous divisibility constraints. "Input at $t-1$ is 'e' AND input at $t-8$ is space" becomes $2^3 \mid S_{t-1} \wedge 2^4 \nmid S_{t-1} \wedge 2^9 \mid S_{t-8} \wedge 2^{10} \nmid S_{t-8}$. UM pattern matching is entirely reducible to GMP divisibility tests.

5 Experiment 4: Quotient Classes

Question: How many distinct states are there, ignoring time?

The timeless residual $R_t = 2^{\text{eid}(t)}$ groups positions by their input character. With 52 distinct characters, there are exactly 52 equivalence classes.

R_t	Char	Event ID	Count
2^9	space	9	127 (12.4%)
2^3	`e`	3	107 (10.4%)
2^6	`a`	6	89 (8.7%)
2^5	`i`	5	48 (4.7%)
2^{13}	`s`	13	44 (4.3%)
2^2	`m`	2	42 (4.1%)
2^{12}	`n`	12	41 (4.0%)

Compression: 1024 positions $\to$ 52 classes = 19.7×.

Lesson: The quotient $Q = \{R_t : t \in [1, N]\}$ is the set of distinct timeless states. Its size measures the data's variety. With 2 ESes, $|Q|$ = number of distinct characters. Adding neuron ESes would give $|Q| \leq 2^{128}$, but saturation means the actual count is much smaller— this directly measures the RNN's effective state count.

Application: Compare $|Q|$ across different ES choices: with just input (52 classes), with input + neurons (expected: a few hundred, since many neuron configurations repeat), with input + neurons + word length. The growth rate of $|Q|$ as we add ESes measures how much new information each ES contributes.

6 Experiment 5: Cross-Position Products

Question: How do multi-position patterns look in the encoding?

For bigram patterns (offset 1), the context at position t involves both t and $t-1$. Using a separate prime per offset ($p_{\text{curr}} = 2$, $p_{\text{prev}} = 5$), the bigram context is $2^{\text{eid}(t)} \cdot 5^{\text{eid}(t-1)}$.

Context type	Offsets	Distinct classes	Compression
Unigram	$[0]$	52	19.7×
Bigram	$[0, 1]$	231	4.4×
Skip-2	$[0, 1, 8]$	511	2.0×
Maximum possible		52^k	

231 bigram contexts out of $52^2 = 2704$ possible (8.5% coverage). 511 skip-2 contexts out of $52^3 = 140608$ possible (0.36% coverage). The sparsity increases rapidly—most character combinations never occur.

Lesson: The prime encoding naturally separates contributions from different offsets into different prime bases. The context at any position is a single integer whose factorization reads off each offset's input. This is the multi-dimensional version of the projection: each offset gets its own dimension (prime), and the context lives in $\mathbb{N}^k$.

Application: The skip-8 UM's 834 patterns correspond to 834 recurring context integers (products of 8 offset primes). The fraction of possible contexts that actually occur ($511/140608 = 0.36\%$ for just 3 offsets) shows the extreme sparsity of natural language—and explains why pattern-based compression works.

7 Experiment 6: Information Content

Question: How many bits does each encoded term carry?

$\log_2(S_t) = \text{eid}(t) + t \cdot \log_2 3 \approx \text{eid}(t) + 1.585t$.

The range is $[2.6, 1650.6]$ bits, mean 826.3 bits. The total $P(E)$ carries 1651 bits. Since $P(E)$ is dominated by its largest term (S_{1024}), the sum barely exceeds the maximum.

Lesson: The information content of each term grows linearly with position (the 3^t factor dominates). The *input contribution* adds only eid bits (1–52), which is tiny compared to the time contribution ($t \cdot 1.585$). In a sense, time carries most of the "information" in the encoding, and the input signal rides on top.

Application: This suggests that for analysis, factoring out 3^t (stripping time) is essential—it removes the linear growth and leaves only the input signal. The residual $R_t = 2^{\text{eid}(t)}$ carries at most 52 bits per position.

8 Experiment 7: Residues Mod Small Primes

Question: What does $P(E) \bmod q$ tell us for small primes q?

q	$P(E) \bmod q$	q	$P(E) \bmod q$	q	$P(E) \bmod q$
5	3	17	9	31	14
7	1	19	7	37	22
11	9	23	19	41	27
13	9	29	24	43	2

The low-order bits of $P(E)$ show structure: $P(E) \bmod 4 = 0$ (divisible by 4), $P(E) \bmod 8 = 4$, $P(E) \bmod 512 = 124$.

Lesson: $P(E) \bmod 2^k$ depends only on the terms whose 2^{eid} contributes to the low k bits. Since the smallest event IDs (1, 2, 3) belong to the most common characters (`<`, `m`, `e`), the low-order structure of $P(E)$ is dominated by the most frequent characters. The residue $P(E) \bmod 4 = 0$ tells us something about the balance of odd vs. even event IDs.

Application: Residues mod the encoding primes (2, 3) are especially meaningful. $P(E) \bmod 3$ collapses all the time information (since $3^t \equiv 0 \pmod 3$ for $t \geq 1$), leaving only the "timeless signature" of the data. Residues mod other small primes act as hash-like projections that could be used for fast approximate pattern matching.

9 Experiment 8: The Alphabet Embedding

Question: Is the first-occurrence ordering optimal?

The "weight" of the encoding is $\sum_t \text{eid}(t) = \sum_c \text{eid}(c) \cdot \text{freq}(c)$, which controls $P(E)$'s magnitude through the 2^{eid} factors.

First-occurrence weight: 14355. Frequency-optimal weight (assign smallest event IDs to most frequent characters): 10894 (75.9% of first-occurrence).

The first-occurrence ordering is 24% suboptimal because frequent characters like space (event ID 9, frequency 127) and `e` (event ID 3, frequency 107) don't get the smallest IDs—they get the IDs of when they first appeared in the text.

Lesson: The first-occurrence ordering trades optimality for *information*: it encodes the data's structure into the alphabet itself. Space appearing at position 11 (not position 1) means the text starts with non-space characters (`<mediawiki`). This is meaningful metadata.

A frequency-optimal encoding would minimize $P(E)$ but lose this metadata. A Huffman-like encoding would minimize the expected $\log_2(S_t)$. The choice of encoding is itself a modeling decision.

Application: Different alphabet orderings give different $P(E)$ values for the same data. The ordering that minimizes $P(E)$ assigns small IDs to frequent characters (Huffman-like). The ordering that maximizes the *compressibility* of $P(E)$ might be different—it would group characters that appear in similar contexts, making the sequence of event IDs more predictable. Finding this ordering is itself a pattern discovery problem.

10 Summary

The prime encoding converts a byte sequence into integer arithmetic. Each experiment reveals a facet of this correspondence:

Experiment	What it reveals
GCD	Shared state between time steps
Transition ratios	Dynamics as multiplicative structure
Modular matching	Pattern detection as divisibility
Quotient classes	Effective state count
Cross-products	Multi-position context separation
Information content	Bit budget per position
Residues	Hash-like data projections
Alphabet embedding	Encoding as modeling decision

The encoding is currently minimal (2 primes, 2 event spaces). Adding neuron sign ESes (128 more primes) would make experiments 1 and 4 dramatically more informative: GCD would reveal persistent neuron states, and the quotient class count would measure the RNN's effective state complexity.

All code: `experiments/prime_poly/{prime_poly.c, experiments.c}`.

References

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

[2] Claude and MJC. Event Arithmetic: E onto $\mathbb{N}$. 10 Feb 2026.