Event Arithmetic: E onto $\mathbb{N}$

Claude and MJC

10 February 2026

Abstract

We present a concrete encoding of Universal Machine event spaces as prime powers. Each event space (input character, position, neuron sign, etc.) is assigned a unique prime; the exponent encodes which event within that space is true. This is *not* a polynomial (in a polynomial, bases are variables and exponents are fixed); here the bases are fixed primes and the exponents are the variables. Each position's state is a point in $\mathbb{N}^k$ projected into a single integer. The dataset becomes a sum of these integers—a 1024-point cloud encoded as one number. We show that (1) decomposing by the time prime trivially recovers per-position states, (2) recurring exponent patterns across related points correspond to UM patterns, and (3) the factorization of event spaces is essential for tractability: an unfactored 2^{128}-entry space can only be learned as a lookup table, while factoring into 128 binary spaces makes it learnable. We connect this encoding to the sat-rnn (128 hidden, 0.079 bpc) and the superset skip-8 UM (834 patterns, 0.043 bpc) via a factor-permutation map. A GMP experiment on 1024 bytes of enwik9 with 2 event spaces (input character, time) yields a 497-digit $P(E)$.

1 Introduction

The CMP formalism [1] treats a predictive model as a collection of events—English declarative sentences that are true or false at each moment. An event might be "The input character is 'e' " or "Neuron h_{42} is active" or "The word length is 3." A *product* of events is a conjunction: "The input is 'e' AND the position is 42."

A Universal Machine (UM) predicts by identifying which events are true and computing a conditional probability for the next output. The *pattern inventory*—the set of event conjunctions the UM uses—determines its compression rate. The skip-8 UM achieves 0.043 bpc with 834 patterns; the sat-rnn achieves 0.079 bpc via 128 hidden neurons whose activations encode a different, entangled representation of the same events.

This paper asks: can we make the event structure *computable* as integer arithmetic? The answer is yes, via the prime power encoding.

2 Event Spaces

Definition 1 (Event Space)**.** *An* event space *(ES) is a finite set of mutually exclusive, collectively exhaustive atomic events. At any moment, exactly one event in each ES is true.*

For our 1024-byte dataset with a 128-neuron sat-rnn, the relevant event spaces are:

Event Space	Size	Example
$\mathcal{E}_{\text{input}}$	256	"The input is `0x65`"
$\mathcal{E}_{\text{time}}$	1024	"The position is 42"
$\mathcal{E}_{h_0}$	2	"h_0 is $+1$"
$\vdots$	$\vdots$	$\vdots$
$\mathcal{E}_{h_{127}}$	2	"h_{127} is -1"
$\mathcal{E}_{\text{wl}}$	$\sim$20	"Word length is 3"

The input character ES has 256 values. These can also be seen as 8 binary sub-event-spaces (the bits of the byte), which would be useful if we later want to factor the alphabet. For now, one ES with 256 values suffices.

2.1 Why Factoring Matters

The entire hidden state could be treated as a single unfactored ES with $2^{128} \approx 10^{38}$ values. This is a lookup table—unlearnable, unstorable, incompressible. Factoring into 128 binary ESes gives the same information in 128 bits: same content, tractable form.

Proposition 1. *Compression is factoring event spaces. The UM's patterns are a factorization of the joint event space into conditionally independent pieces. The RNN's neurons are another factorization. The factor map ϕ connects the two.*

This is the deep point: an unfactored E can only be learned as a massive LUT. Factoring is essential, even if only into 2^{dim}.

3 The Prime Power Encoding

Definition 2 (Prime Power Encoding)**.** *Assign each event space $\mathcal{E}_k$ a unique prime p_k. If the current value in $\mathcal{E}_k$ is $v_k \in \{0, 1, \ldots, |\mathcal{E}_k| - 1\}$, encode it as $p_k^{v_k}$.*

Concretely:

$$\begin{aligned}
\mathcal{E}_{\text{input}} \to p_1, &\quad \text{input 'e' } (101) \to p_1^{101} \\
\mathcal{E}_{\text{time}} \to p_2, &\quad \text{position } 42 \to p_2^{42} \\
\mathcal{E}_{h_0} \to p_3, &\quad h_0 = +1 \ \ (\mapsto 1) \to p_3^1 \\
\mathcal{E}_{h_1} \to p_4, &\quad h_1 = -1 \ \ (\mapsto 0) \to p_4^0 = 1 \\
&\ \ \vdots \\
\mathcal{E}_{h_{127}} \to p_{130}, &\quad h_{127} = +1 \ \ (\mapsto 1) \to p_{130}^1 \\
\mathcal{E}_{\text{wl}} \to p_{131}, &\quad \text{wl} = 3 \to p_{131}^3
\end{aligned}$$

where sign_{01} maps $\{-1, +1\} \to \{0, 1\}$.
The state at position t is the product of all prime powers:

$$S_t = p_1^{x_t} \cdot p_2^t \cdot \prod_{j=0}^{127} p_{j+3}^{\text{sign}_{01}(h_{j,t})} \cdot p_{131}^{\text{wl}_t}$$

Each S_t is a single integer whose prime factorization completely encodes every observable fact at position t. The exponent of p_1 gives the input character. The exponent of p_2 gives the time. The exponent of p_{j+3} gives neuron j's sign. The encoding is lossless and unique (by the fundamental theorem of arithmetic).

4 The Prime Encoding as Projection

Note: we initially called this a "prime polynomial," but in a polynomial the base is the variable ($a_0 + a_1 x + a_2 x^2$); here the bases are fixed primes and the *exponents* are the variables. The structure is a projection of points in $\mathbb{N}^k$ into $\mathbb{N}$.

Definition 3 (Prime Encoding)**.** *The* prime encoding *of a dataset is:*

$$P(E) = \sum_{t=0}^{N-1} S_t = \sum_{t=0}^{N-1} p_1^{x_t} \cdot p_2^t \cdot \prod_{j=0}^{127} p_{j+3}^{sign_{01}(h_{j,t})} \cdot p_{131}^{wl_t}$$

Each term S_t is a point $(v_1(t), v_2(t), \ldots, v_k(t))$ in $\mathbb{N}^k$ projected into $\mathbb{N}$ via the prime encoding. The encoding is a bijection (by the fundamental theorem of arithmetic). The terms are distinct (the p_2^t factor is unique to each t). The sum preserves which events co-occur at which positions—it encodes the full 1024-point cloud in event space as a single integer.

$P(E)$ is a very large integer. For $N = 1024$ with 2 event spaces (input character + time), $P(E)$ has 497 digits. With 131 event spaces (adding neuron signs + word length), terms reach thousands of digits. This is why GMP (GNU Multiple Precision Arithmetic) is required: floating-point arithmetic would destroy the factorization structure.

4.1 Factoring Time First

Since p_2^t gives each term a unique power of the time prime, we can decompose:

$$P(E) = \sum_{t=0}^{N-1} p_2^t \cdot R_t$$

where $R_t = S_t / p_2^t$ is the "timeless residual" at position t.

Everyone already has the intuition for this: it's "what happened at each position." The time prime separates the terms cleanly. Factoring R_t recovers the per-position event values.

Example 1. *At position $t = 42$, suppose the input is 'e' (101), $h_0 = +1$, $h_1 = -1$, and word length is 3. Then:*

$$R_{42} = p_1^{101} \cdot p_3^1 \cdot p_4^0 \cdot \ldots \cdot p_{131}^3 = p_1^{101} \cdot p_3 \cdot p_{131}^3 \cdot \prod_{j:h_j=+1} p_{j+3}$$

The prime factorization of R_{42} directly reads off the event values.

5 UM Patterns as Sub-Products

A UM pattern is a conjunction of events across related positions. For example, the skip-pattern "when input at offset -1 is 'e' and offset -8 is ' ', predict 'n'" involves three terms of $P(E)$:

- Term at t: has p_1^{110} (output 'n' = 110)

- Term at $t-1$: has p_1^{101} (input 'e' = 101)
- Term at $t-8$: has p_1^{32} (input ' ' = 32)

Pattern discovery amounts to finding *cross-term constraints*: positions where specific prime exponents recur in related terms of $P(E)$.

The UM's pattern count equals the number of terms satisfying the constraint. The UM's prediction is the conditional distribution of p_1 exponents (output character) given the constraint on neighboring terms.

5.1 From Superset UM to Patterns

For the GMP experiment, we start from the superset UM (skip-8, offsets $[1, 8, 20, 3, 27, 2, 12, 7]$, 834 patterns, 0.043 bpc). The event spaces for this UM are simpler: input character at each of the 8 offsets, plus output character, plus time. That is 10 event spaces $\rightarrow$ 10 primes.

The state at position t in the UM encoding:

$$S_t^{\text{UM}} = p_{\text{out}}^{y_t} \cdot p_{\text{time}}^{t} \cdot \prod_{k=1}^{8} p_{\text{off}_k}^{x_{t-o_k}}$$

This is a much smaller encoding (10 primes vs. 131), but it encodes exactly the information the UM uses for prediction. The GMP integers are still large (exponents up to 255), but manageable.

6 The Factor-Permutation Map

The sat-rnn's hidden state $h_t \in \{-1, +1\}^{128}$ is a different encoding of overlapping information. The isomorphic UM (the RNN viewed as a counting machine) uses 128 binary ESes for the neuron signs plus the input ES.

The factor-permutation map connects the UM encoding to the RNN encoding:

$$\text{UM primes} \sigma \text{RNN primes}$$

Since we know the neuron ordering, σ maps the UM's 10 offset-input primes onto combinations of the RNN's 128 neuron-sign primes. This map σ *is* the factor map ϕ from the previous papers, now expressed as a permutation/combination of prime bases.

The isomorphic UM doesn't have a convenient prime factorization (the tanh dynamics mix primes in nonlinear ways), but the UM's patterns—which are conjunctions of input events—do factorize cleanly. The factor-permutation map is between these two factorizations.

7 Discussion

$P(E)$ is the dataset viewed as a number-theoretic object—a point cloud in $\mathbb{N}^k$ projected into a single integer. Its structure—which primes appear in which terms, with what exponents—encodes every pattern the UM can discover.

Three observations:

Exact arithmetic preserves structure. Floating-point log-space computation is efficient but destroys the factorization. GMP preserves it. The cost is computational (large integers), but for $N = 1024$ with small prime counts, this is tractable.

Pattern discovery = factorization. Finding recurring sub-products across terms of $P(E)$ is exactly the problem of finding common factors in a set of integers. This connects UM pattern discovery to classical number theory.

Factoring ESes is compression. The choice of how to factor the joint event space determines what is learnable. 2^{128} as one ES = LUT = unlearnable. 128 binary ESes = 128 bits = tractable. The UM's 10 offset-input ESes are an even coarser factoring that still captures 0.043 bpc. Compression = choosing the right factorization.

8 Experimental Results

We implemented the prime encoding for 1024 bytes of enwik9 with $k = 2$ event spaces: input character ($p_1 = 2$) and time ($p_2 = 3$). The input character ES uses first-occurrence ordering from the data: the first byte '`<`' gets event ID 1, '`m`' gets 2, '`e`' gets 3, etc. This ordering is itself an embedding—it encodes the data's character frequencies.

52 distinct byte values appear in the dataset, giving 52 events in the input ES and 1024 events in the time ES.

Position	Char	$S_t = 2^a \cdot 3^t$	Digits
$t = 1$	`<`	$2^1 \cdot 3^1 = 6$	1
$t = 2$	`m`	$2^2 \cdot 3^2 = 36$	2
$t = 3$	`e`	$2^3 \cdot 3^3 = 216$	3
$t = 8$	`i`	$2^5 \cdot 3^8$	6
$t = 1024$	`a`	$2^6 \cdot 3^{1024}$	491
$P(E) = \sum S_t$			497

$P(E)$ is a 497-digit integer. Verification: subtracting all 1024 reconstructed terms from $P(E)$ yields exactly zero. The prime factorization of each term recovers the input character and position.

The digit count grows linearly with position (dominated by 3^t), from 1 digit at $t = 1$ to 491 digits at $t = 1024$. The input exponent adds at most $\lfloor 52 \cdot \log_{10} 2 \rfloor = 16$ digits. Average term: 249 digits.

9 Next Steps

Extending the experiment:

1. Define event spaces for the superset UM (10 primes).
2. For each of 1024 positions, compute S_t^{UM} as `mpz_t`.
3. Sum to get $P(E)$.
4. Factor by time, verify per-position residuals match data.
5. Search for common sub-products → recover pattern inventory.
6. Compare recovered patterns with the known 834-pattern inventory.

References

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

[2] Claude and MJC. The Factor Map. 9 Feb 2026.

[3] Claude and MJC. Pattern Chains. 8 Feb 2026.

[4] Claude and MJC. Sparse Differentiation. 9 Feb 2026.