The Tock Step: Domain-Native Architecture from Evidence

Claude and MJC

February 12, 2026

Abstract

The tock step in the CMP tick-tock cycle is not merely extraction of interpretable features from trained weights—it is the construction of a domain-native Universal Model via evidence. We formalize this: the tock step discovers the *next event space* E_{k+1} and the maps connecting it to existing event spaces $E_0, \ldots, E_k$. Learning the architecture IS choosing the next E_i plus maps. We develop the *factorization tower* $E_0 \twoheadrightarrow E_1 \twoheadrightarrow \cdots \twoheadrightarrow E_n$ of progressively coarser event spaces, and show that the ability to *backtrack* through this tower—reverting from a coarser to a finer factorization—is essential for tasks that require resolution lost in the forward (coarsening) direction. The "strawberry problem" (counting characters within tokens) is the canonical example: the tokenizer's factorization is a quotient that discards character-level structure, and no amount of forward computation within the token-level event space can recover it. The fix is not a better tokenizer but the ability to backtrack through the factorization. We connect this to the experimental program: the backward trie, MI-per-offset analysis, and skip-k-gram discovery are all instances of the tock step discovering new E_i.

1 Introduction

The January 31 Tock paper [2] defined the tock step as "extracting interpretable structure from trained weights." Five ESes explained 59% of an RNN's compression. The February 4 "Tock 1" analysis [3] went deeper: 18 natural event spaces, 302 significant patterns, word boundaries and syllable momentum.

But these tock steps had a limitation: they took the *architecture* as given (128 hidden neurons, tanh, BPTT-50) and searched for interpretable structure *within* it. The architecture was fixed; only the interpretation changed.

This paper argues that the tock step should be understood more broadly: *learning the architecture itself.* The architecture of a UM is its collection of event spaces $\{E_i\}$ and the maps between them. The tock step discovers the next event space E_{k+1} and the maps connecting it to the existing $E_0, \ldots, E_k$. This is *architecture from evidence*—not architecture search by hyperparameter optimization, but architecture *discovery* by counting.

The key equation:

$$\text{tock step} = \text{next } E_{k+1} + \text{maps to/from existing } \{E_i\}. \tag{1}$$

The maps are the patterns P in the UM five-tuple. The event spaces are the E. The tock step expands both simultaneously.

2 The Architecture Is the Event Spaces

Definition 1 (Architecture of a UM)**.** *The* architecture *of a Universal Model $u = (e, t, p, f, \omega)$ is the collection of event spaces $\mathcal{E} = \{E_0, E_1, \ldots, E_n\}$ together with the maps (patterns) connecting them:*

$$\mathcal{A} = (\mathcal{E}, \{p_{ij} : E_i \to E_j\}_{i,j}).$$

The architecture determines what can be represented*—which events are distinguishable, which patterns are expressible, which questions can be asked.*

Remark 2 (Architecture vs. parameters)**.** *In a neural network, the architecture (number of layers, hidden size, activation function) is chosen before training. The parameters (weights, biases) are learned during training. In the UM framework, this distinction maps cleanly:*

Neural network	*UM*
Architecture (fixed)	*Event spaces $\{E_i\}$ (fixed)*
Parameters (learned)	*Patterns $\{p_{ij}\}$ (learned)*
Hyperparameter search	*Tock step (discover next E_i)*

The tock step is the UM's version of architecture search—but driven by evidence, not by grid search or random sampling.

Example 3 (The sat-rnn's architecture)**.** *The 128-hidden tanh RNN has:*

- *$E_0 = \{0, \ldots, 255\}$: the byte input space (256 events).*
- *$E_1, \ldots, E_{128}$: binary hidden event spaces (2 events each), one per neuron.*
- *$E_{out} = \{0, \ldots, 255\}$: the byte output space.*
- *Maps: W_x (byte $\to$ hidden), W_h (hidden $\to$ hidden), W_y (hidden $\to$ output), b_y (bias).*

The architecture was chosen before training. Training discovered the parameter values but not the event spaces themselves. The tock step of the February 9 factor map [4] revealed that these 128 binary ESes actually encode $\sim$20 functional features (word length, in-tag, offset conjunctions, etc.), but this was an after-the-fact *interpretation, not a construction.*

Definition 4 (Domain-native architecture)**.** *An architecture $\mathcal{A}$ is* domain-native *when the event spaces $\{E_i\}$ correspond to natural distinctions in the domain:*

- *For English text: word boundary (binary), character class (vowel/consonant/digit/punct/space), tag state (in/out of markup), word length (modular counter), etc.*
- *For images: edge/no-edge, object class, spatial position, color channel, etc.*

A domain-native architecture is automatically interpretable (CMP's equivalence thesis [1]).

3 The Tock Step as Architecture Discovery

Definition 5 (Tock step (expanded))**.** *Given an existing UM with architecture $\mathcal{A}_k = (\{E_0, \ldots, E_k\}, \{p_{ij}\})$ and a dataset D:*

1. ***Identify the gap**: compute the prediction error (bpc, cross-entropy, or luck) under the current architecture. Identify* where *the error concentrates—which positions, which contexts, which output events are most surprising.*

2. ***Search for the next** E_{k+1}: find a new event space that, if added to $\mathcal{A}_k$, would maximally reduce the prediction error. This is a search over possible factorizations of the residual information.*

3. ***Learn the maps**: once E_{k+1} is identified, compute the patterns $p_{i,k+1}$ and $p_{k+1,j}$ connecting it to existing event spaces. These are the conditional count tables (the output of the standard learning function ω_0).*

4. ***Update the architecture**: set $\mathcal{A}_{k+1} = (\{E_0, \ldots, E_{k+1}\}, \{p_{ij}\} \cup \{p_{i,k+1}, p_{k+1,j}\})$.*

Proposition 6 (Tock is evidence-based)**.** *Every component of the tock step is derived from evidence (data counts):*

- *The gap is computed from prediction errors on D.*
- *The search for E_{k+1} uses mutual information with the output, computed from co-occurrence counts.*
- *The maps $p_{i,k+1}$ are log contingency tables—the output of ω_0.*

No beliefs (axioms, postulates) or abductions (pattern commitments) are involved. The tock step is purely evidential.

Remark 7 (The tock step in the experimental program)**.** *The February 7–12 archive contains multiple instances of the tock step:*

Method	*What it discovers*	*New E_i*
Backward trie [5]	*MI-ranked offsets*	*offset conjunctions*
Skip-k-gram analysis	*informative offset pairs*	*2-offset ESes*
Factor map [4]	*neuron → conjunction*	*domain features*
ES discovery [6]	*SVD of skip-bigram matrix*	*event clusters*
Weight construction [7]	*shift-register groups*	*hash-based ESes*

Each discovers a new event space from data statistics, then builds maps connecting it to the existing architecture. This is the tock step, performed manually by the researchers. The goal is to make it automatic.

4 The Factorization Tower

Definition 8 (Factorization tower)**.** *A* factorization tower *is a sequence of event spaces connected by surjections (quotient maps):*

$$E_0 \xrightarrow{\pi_1} E_1 \xrightarrow{\pi_2} E_2 \xrightarrow{\pi_3} \cdots \xrightarrow{\pi_n} E_n, \tag{2}$$

where each $\pi_k : E_{k-1} \twoheadrightarrow E_k$ is a surjection that maps finer events to coarser events.

Example 9 (The text tower)**.** *For natural language text:*

Level	*Event space*	*Size*
E_0	*Bits*	*2*
E_1	*Bytes (characters)*	*256*
E_2	*Character classes*	*∼10 (vowel, consonant, digit, ...)*
E_3	*Subword tokens (BPE)*	*∼50k*
E_4	*Words*	*∼100k*
E_5	*Phrases / syntactic units*	*unbounded*

Each level is a quotient of the one below: π_2 maps bytes to character classes, π_3 maps character sequences to tokens, π_4 maps token sequences to words.

Proposition 10 (Information loss in the tower)**.** *Each surjection π_k loses information:*

$$I(E_{k-1}) \geq I(E_k), \tag{3}$$

with equality iff π_k is a bijection. The lost information is the within-class *structure that the quotient discards.*

The total information loss across the tower:

$$I(E_0) - I(E_n) = \sum_{k=1}^{n} [I(E_{k-1}) - I(E_k)] = \sum_{k=1}^{n} H(\mathit{fiber}_k), \tag{4}$$

where $H(\mathit{fiber}_k)$ is the entropy of the fibers of π_k (the within-class structure at level k).

Definition 11 (Product vs. sum factorization)**.** *A factorization can decompose events in two ways:*

- ***Product*** *(Cartesian): $E = E_a \times E_b$. Total event $= (e_a, e_b)$. Both components are accessible independently. $I(E) = I(E_a) + I(E_b)$.*
- ***Sum*** *(sequential/quotient): $E_{\mathit{fine}} \twoheadrightarrow E_{\mathit{coarse}}$. The coarse event is a function of the fine event. The fine-grained structure is* inside *the coarse event, accessible only by backtracking.*

Remark 12 (The critical distinction)**.** *Tokenization is a sum factorization: a word is a sequence of characters, and the token is a function of that sequence. The characters are inside the token.*

A binary ES decomposition is a product factorization: the hidden state $h = (h_1, \ldots, h_{128})$ decomposes into 128 independent binary events. Each bit is independently accessible.

Product factorizations preserve access; sum factorizations hide structure. The tock step should prefer product factorizations when possible, because they don't require backtracking.

5 Backtracking the Factorization

Definition 13 (Backtracking)**.** *Given a factorization tower* (2) *and a task that requires resolution at level E_{k-1} but the system is currently operating at level E_k,* backtracking *is the operation of reverting to the finer level:*

$$\mathit{backtrack}_k : E_k \to 2^{E_{k-1}} \tag{5}$$

which maps each coarse event to its fiber*—the set of fine events that map to it under π_k.*

The backtrack map π_k^{-1} is always well-defined (as a set-valued map), since π_k is surjective. But it produces a *set*, not a single event: the system knows which coarse class it's in, but not which fine event within the class. This is the information that was lost in the forward (coarsening) direction.

Theorem 14 (Backtracking recovers lost information)**.** *The information available after backtracking is:*

$$I(\textit{after backtrack}_k) = I(E_k) + H(\textit{fiber}_k \mid \textit{context}), \tag{6}$$

where the conditional fiber entropy $H(\textit{fiber}_k \mid \textit{context})$ depends on what the system knows from context about which fine event within the fiber is actual.

If context determines the fine event exactly: $H(\textit{fiber}_k \mid \textit{context}) = 0$ and backtracking is free.

If context gives no information: $H(\textit{fiber}_k \mid \textit{context}) = H(\textit{fiber}_k)$ and backtracking costs the full within-class entropy.

5.1 The Strawberry Problem

Example 15 (Counting characters within tokens)**.** *A language model tokenizes "strawberry" as, say, ["straw", "berry"]. The user asks: "How many r's in strawberry?"*

***The problem**: "r" is not an event in the token-level event space E_3. The model operates on tokens; counting characters requires the character-level event space E_1. The tokenizer's quotient $\pi_3 : E_1^* \twoheadrightarrow E_3^*$ discards character-level structure.*

***The fix**: backtrack through π_3. Map each token to its constituent characters:*

$$\textit{"straw"} \mapsto [s, t, r, a, w], \qquad \textit{"berry"} \mapsto [b, e, r, r, y].$$

Now count: three occurrences of "r."

***Why current LLMs fail**: the transformer's computation graph operates entirely at the token level. There is no mechanism to backtrack through π_3 and operate at E_1. The system* could *memorize character counts for common words, but this is a hack (storing the answer to every possible question) rather than a solution (having access to the character-level event space).*

***The UM solution**: maintain the full tower. Normally operate at whatever level is efficient (E_3 or E_4). When a task requires finer resolution, backtrack to the appropriate level. The backtrack is possible because the finer event spaces are part of the architecture, not discarded preprocessing.*

Proposition 16 (When backtracking is needed)**.** *A task requires backtracking through π_k when the task's answer depends on within-fiber structure at level k—information that π_k discards.*

Formally: a task τ requires level E_{k-1} when $I(\tau; E_{k-1}) > I(\tau; E_k)$, i.e., the task has mutual information with the finer level that is absent at the coarser level.

For the strawberry example: $I(\textit{"count r's"}; E_1) = \log_2 10$ (need to inspect each character), while $I(\textit{"count r's"}; E_3) \approx 0$ (tokens don't distinguish characters internally).

6 Domain-Native Construction

Definition 17 (Domain-native tock sequence)**.** *Starting from the minimal architecture $\mathcal{A}_0 = (\{E_0\}, \emptyset)$ where E_0 is the raw observation space (bytes), the* domain-native tock sequence *is:*

1. *$\mathcal{A}_0$: bytes only. Prediction: unigram model. BPC ≈ 5.0 (byte entropy of English).*
2. *$\mathcal{A}_1$: bytes + bigram ES. Tock discovers that adjacent bytes are highly correlated. $E_1 = E_0 \times E_0$ (input-output pairs). Adds the bigram count table. BPC ≈ 3.5.*
3. *$\mathcal{A}_2$: + word boundary ES. Tock discovers that the space character creates a binary partition with high MI. $E_2 = \{0, 1\}$ (word-internal vs. boundary). BPC ≈ 3.0.*

4. $\mathcal{A}_3$: *+ tag state ES. Tock discovers that* $\langle$ *and* $\rangle$ *create a binary partition (in-tag vs. out-of-tag) with high MI at large offsets.* $E_3 = \{0, 1\}$. $BPC \approx 2.5$.

5. $\mathcal{A}_4$: *+ offset conjunctions. Tock discovers that pairs of offsets (e.g.,* $(d_1, d_2) = (1, 7)$*) carry more MI than individual offsets.* E_4 *= 2-offset conjunction ESes.* $BPC \approx 1.0$.

6. *And so on: each tock discovers the next most informative ES.*

Remark 18 (Comparison with neural architecture search)**.**

	Neural arch. search	*Domain-native tock*
Search space	*Hyperparameters*	*Event spaces*
Objective	*Validation loss*	*MI with output*
Method	*Grid/random/Bayesian*	*Counting + quotients*
Cost	*Train many models*	*Count once*
Result	*Architecture (opaque)*	*Architecture (interpretable)*
Backtracking	*Not applicable*	*Built in (tower)*

The domain-native tock is cheaper (no training), interpretable (every E_i *has a name), and supports backtracking (the tower is maintained).*

7 The MI Criterion for E_{k+1} Discovery

Definition 19 (Residual MI)**.** *Given architecture* $\mathcal{A}_k$ *with event spaces* $E_1, \ldots, E_k$*, the* residual mutual information *for a candidate event space* E' *is:*

$$\Delta I(E') = I(O; E' \mid E_1, \ldots, E_k) = I(O; E_1, \ldots, E_k, E') - I(O; E_1, \ldots, E_k), \tag{7}$$

the additional MI that E' *provides about the output* O *beyond what the existing event spaces already capture.*

Proposition 20 (Greedy tock optimality)**.** *At each step, choosing* $E_{k+1} = \arg\max_{E'} \Delta I(E')$ *greedily maximizes the prediction improvement. This is a greedy algorithm for the set-cover problem over the output entropy: each new* E_i *"covers" a portion of the remaining uncertainty about* O.

Remark 21 (Connection to offset MI)**.** *In the experimental program, the MI between offset* d *and the output was computed for all offsets* $d = 1, \ldots, 50$*. The backward trie [5] ranks offsets by MI, discovering that* $(d = 1) > (d = 8) > (d = 2) > (d = 7) > \cdots$*. Each offset corresponds to a candidate event space* $E'_d = \{0, \ldots, 255\}$ *(the byte at offset* d*). The greedy tock selects* E'_1 *first, then* E'_8*, then* E'_2*, matching the empirically discovered skip-pattern order.*

The 2-offset conjunctions add product ESes: $E'_{(d_1,d_2)} = E'_{d_1} \times E'_{d_2}$*. The factor map [4] shows that the trained RNN discovers exactly these product ESes, with the dominant pair* $(1, 7)$ *used by 52/128 neurons.*

8 Backtracking Through Tokenization

The factorization tower makes the tokenization problem precise.

Definition 22 (Tokenizer as quotient)**.** *A tokenizer* $\pi : \Sigma^* \to V^*$ *maps character sequences to token sequences. This is a quotient: each token* $v \in V$ *corresponds to a character sequence* $\pi^{-1}(v) \in \Sigma^*$*. The quotient discards the internal structure of each token.*

Proposition 23 (What tokenization loses)**.** *For a tokenizer with vocabulary V of mean token length $\bar{\ell}$:*

1. ***Character identity****: the specific characters within a token are not individually addressable. Only the token-level event space is available.*
2. ***Character position****: the position of a character within its token is not represented. "r" in position 3 of "straw" vs. position 3 of "berry" are indistinguishable at the token level.*
3. ***Character count****: the number of occurrences of a specific character within a token is not represented. "berry" has two r's; this fact is lost in the token "berry."*

Information lost per token: up to $\bar{\ell} \cdot \log_2 |\Sigma|$ bits of character-level structure.

Theorem 24 (The strawberry theorem)**.** *Let τ be a task that requires character-level resolution (e.g., counting a specific character in a word). Let M be a model that operates solely on the token-level event space E_V. Then:*

$$P(M \text{ answers } \tau \text{ correctly}) \leq P(\text{answer is retrievable from } E_V), \tag{8}$$

where the right side is the probability that the answer happens to be determinable from the token sequence alone (e.g., the word is always tokenized in a way that isolates the target character, or the model has memorized the answer).

For novel words or unusual tokenizations, the right side approaches zero. The model cannot systematically solve character-level tasks at the token level.

Remark 25 (The general principle)**.** *The strawberry problem is not about characters and tokens specifically. It is about* any *task that requires resolution at a level finer than the model's operating event space.*

Other instances:

- ***Phoneme counting****: "How many syllables in 'antidisestablishmentarianism'?" requires phoneme-level ESes.*
- ***Syntactic parsing****: "Is 'the' the subject or object?" requires word-level role ESes.*
- ***Logical reasoning****: "Does $A \wedge B$ imply C?" requires proposition-level ESes (cf. the logic paper [8]).*

In each case, the model's current factorization doesn't resolve the relevant distinctions, and backtracking to a finer level is needed.

9 The Tock Step and the Three Sources of Support

The tock step produces new event spaces via evidence—direct data counts. This places it squarely in the first of the three sources of support identified in [9]:

1. **Evidence** (observation and inference): the tock step counts co-occurrences, computes MI, and discovers event spaces. All support is derived from data.
2. **Belief** (explicit choice): an architect who *declares* "use 128 hidden neurons" or "use BPE with 50k vocabulary" is making a belief-based architectural choice. This is not a tock step—it is a prior commitment to a factorization, made without evidence specific to the domain.

3. **Abduction** (short-circuiting induction): when a researcher sees the pattern in the backward trie and says "aha, the dominant offsets are $(1, 7)$ because offset 7 captures word-initial context while offset 1 captures the current character class"—that's abduction. The pattern has been observed a finite number of times; the researcher commits to it because they understand *why* it holds. Abduction can short-circuit the tock step: instead of waiting for full MI analysis, recognize the structure and commit.

Remark 26 (Neural architecture search is belief, not evidence)**.** *Standard neural architecture search (NAS) operates by trial and error: propose an architecture (belief), train it (tick), evaluate (partial evidence), repeat. The architecture choices are beliefs that are tested against evidence but not derived from it.*

The domain-native tock derives the architecture from evidence directly. No training is needed. The event spaces are the ones that the data supports, not the ones that an architect guesses.

10 The Closed Loop

Theorem 27 (Tock closes the CMP loop)**.** *The tick-tock cycle with architecture-level tock is:*

1. ***Initialize**: $\mathcal{A}_0 = (\{E_0\}, \emptyset)$ (raw observations only).*
2. ***Tock**: discover E_{k+1} from data statistics. Compute maps $\{p_{i,k+1}\}$ via counting.*
3. ***Tick**: use the expanded architecture $\mathcal{A}_{k+1}$ for prediction. Observe outcomes. Update counts (parameter-level learning).*
4. ***Evaluate**: compute residual error. If error is acceptable, stop. Otherwise, return to step 2.*

This cycle converges to a domain-native architecture: the event spaces are exactly those that the data supports, the maps are the count tables, and the predictions are the UM's forward pass (existential quantification over probabilistic syllogisms [8]).

Remark 28 (No gradient descent)**.** *The closed loop uses no gradient descent. Step 2 (tock) uses MI analysis. Step 3 (tick) uses counting (ω_0). Step 4 (evaluate) uses the forward pass (f). All operations are $O(N)$ in the data size, where N is the number of observations.*

The weight construction results of February 11 [7] demonstrate this concretely: all 82k parameters of the sat-rnn were constructed from data statistics (skip-bigram log-ratios, Hebbian covariance, hash-based shift-register design) with zero *gradient descent, achieving 1.89 bpc analytically and 0.59 bpc with one round of W_y optimization (vs. 4.97 bpc for the trained model).*

11 What Backtracking Means for Architecture

Definition 29 (Architectural backtracking)**.** *Given a factorization tower and a task τ:*

1. *Determine the finest level E_k required: the smallest k such that $I(\tau; E_k) = I(\tau; E_0)$ (no further information is lost at level k).*
2. *If $k < n$ (the current operating level), backtrack from E_n to E_k via the inverse maps $\pi_{k+1}^{-1}, \ldots, \pi_n^{-1}$.*
3. *Perform the computation at level E_k.*
4. *Return to level E_n for subsequent operations.*

Proposition 30 (Backtracking is compositional)**.** *Backtracking through the tower is composable:* $\pi_n^{-1} \circ \cdots \circ \pi_{k+1}^{-1} = (\pi_{k+1} \circ \cdots \circ \pi_n)^{-1}$*. A single backtrack to level k gives the same result as sequential backtracks through each intermediate level.*

Remark 31 (The cost of backtracking)**.** *Backtracking from* E_n *to* E_k *increases the event space from* $|E_n|$ *to* $|E_k|$*, expanding the computation by a factor of* $|E_k|/|E_n|$*. For the text tower (bytes to tokens with mean length 4), backtracking from tokens to bytes increases the sequence length by* $\sim 4\times$*.*

This cost is the price of resolution. The domain-native UM pays this cost only when needed—most operations stay at the coarse level, and backtracking occurs only for tasks that require it. A system that always operates at the finest level (character-by-character) avoids the need for backtracking but pays the cost for every operation. A system that always operates at the coarsest level (token-by-token) avoids the cost but cannot solve fine-grained tasks. The tower provides the best of both: coarse-level efficiency for typical tasks, with the ability to backtrack for atypical ones.

12 Discussion

12.1 The tock step is the missing piece

CMP defines the UM five-tuple $u = (e, t, p, f, \omega)$ and the standard learning function ω_0. But ω_0 only learns patterns *within* a fixed architecture (fixed E). It does not learn the architecture itself.

The tock step fills this gap. It is the meta-learning function $\Omega_{\text{meta}} : \mathcal{A}_k \times D \to \mathcal{A}_{k+1}$ that expands the architecture by discovering new event spaces. Together, ω_0 (parameter learning) and Ω_{meta} (architecture learning) constitute the complete learning system.

12.2 Why current LLMs can't backtrack

Modern LLMs (transformers with BPE tokenization) make an irrevocable architectural choice at the tokenization step: the factorization π : chars $\to$ tokens is fixed before training and cannot be undone at inference time. The model operates at the token level, period.

This means every task that requires character-level resolution (spelling, counting, rhyming, anagram-solving) must be handled by memorization or heuristics within the token-level event space. The strawberry problem is not a failure of scale or training—it is a failure of architecture. No amount of data or compute can give the model access to an event space it doesn't have.

The fix is not a "better tokenizer" but a *tower*: maintain multiple levels of the factorization simultaneously, with the ability to move between them as the task demands.

12.3 Architecture from evidence vs. architecture from belief

The standard ML pipeline: an architect chooses the architecture (belief), then training fills in the parameters (evidence). The architecture choice is a prior commitment—often informed by experience, but not derived from the specific data at hand.

The domain-native tock reverses this: the architecture is derived from the data (evidence), and the parameters are the count tables (also evidence). The entire model is evidence-based. The only prior commitment is the choice of E_0 (the raw observation space), which is typically uncontroversial (bytes for text, pixels for images).

This is the strongest reading of CMP's equivalence thesis: *interpretability and efficiency are the same problem because both are solved by the evidence-driven factorization.* The domain-native tock makes this constructive.

References

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

[2] Claude and MJC. *Tock: Extracting Interpretable Structure from Learned Models*. Hutter archive, 31 Jan 2026.

[3] Claude and MJC. *UM Interpretation (Tock 1): Event Spaces, Patterns, and a Semantic Gap*. Hutter archive, 4 Feb 2026.

[4] Claude and MJC. *The Real Factor Map: Interpretable Patterns onto Hidden Dynamics*. Hutter archive, 9 Feb 2026.

[5] Claude and MJC. *Pattern Chains: Explicit $i \to \cdots \to o$ from Data*. Hutter archive, 7 Feb 2026.

[6] Claude and MJC. *The Carrier Signal Problem: Why Product Patterns Need Orthogonal Offsets*. Hutter archive, 12 Feb 2026.

[7] Claude and MJC. *Weight Construction: All 82k Parameters from Data Statistics*. Hutter archive, 11 Feb 2026.

[8] Claude and MJC. *Logic from Counting: Existential Quantification, Probabilistic Syllogisms, and the Derivation of Formal Inference from the Universal Model*. Hutter archive, 12 Feb 2026.

[9] Claude and MJC. *No Support Is Not Disbelief: The Epistemology of Zero in the Universal Model*. Hutter archive, 12 Feb 2026.

[10] Claude and MJC. *A Mathematical Review of CMP*. Hutter archive, 12 Feb 2026.