The Tock Protocol: Systematic Lexicon Injection into the Isomorphic UM

Claude and MJC

February 13, 2026

Abstract

We design a protocol for systematically injecting lexical structure into the isomorphic Universal Model (the doubled-E UM that exactly matches the sat-rnn at 0.079 bpc on DSS = 1024). The procedure: add a single word—say "the"—as a new event in the event space, measure the resulting changes to count tables, pattern structure, and prediction quality, then iterate. Each word addition is a concrete tock step: it introduces E_{k+1} (a new binary event space: *in-word* vs. *not-in-word*) plus maps connecting it to the existing byte-level architecture. Building up from one word to the full lexicon, we obtain a frequency-ordered injection sequence where each step's MI gain is measurable and the cumulative effect traces a curve from byte-level prediction to word-level prediction. The protocol makes the tock step constructive and empirically testable: every claim about lexical structure becomes a measurable delta in bpc.

1 Introduction

The tock step [**?**] discovers new event spaces from data. The weight construction [**?**] showed that all 82k parameters of the sat-rnn can be derived from data statistics. But both are *retrospective*: they analyze what a trained model already knows. Neither constructs the model *forward* from first principles.

This paper designs the forward protocol. Start with the raw byte-level UM ($E_0 = \{0, \ldots, 255\}$, unigram prediction, $\sim$5 bpc). Execute a sequence of tock steps, each adding one word to the event space. Track the bpc after each addition. The result is a constructive proof that lexical knowledge reduces prediction error, with the exact contribution of each word quantified.

The key insight: a word is not an opaque token. A word is a *pattern*—a specific sequence of bytes that recurs with specific statistics. Adding "the" to the UM means:

1. Recognizing the byte sequence $[116, 104, 101]$ (`t-h-e`) as a recurrent pattern,
2. Creating a binary event space $E_{\text{the}} = \{1, 0\}$ (currently in "the" vs. not),
3. Computing the conditional statistics: what predicts "the" (preceding space, punctuation, sentence start) and what "the" predicts (space followed by a noun-class byte distribution),
4. Measuring the MI gain: how much better can we predict the output byte when we know whether we're inside "the"?

This is exactly a tock step in the sense of [**?**]: $E_{k+1} = E_{\text{the}}$, with maps from the byte-level context to E_{the} (recognition) and from E_{the} to the output (prediction).

2 Setup: The Starting Point

Definition 1 (Base UM)**.** *The* base UM *is the byte-level Universal Model with:*

- *Event space* $E_0 = \{0, \ldots, 255\}$ *(bytes).*
- *Context: the previous* k *bytes at offsets* $d_1, \ldots, d_k$ *(the skip-*k*-gram input space).*
- *Count table:* $c(i, o) = |\{t : context_t = i, d_t = o\}|$.
- *Forward pass:* $f_p(t)_o = \max_i \min(t_i, p_{io})$.

At order 1 (bigram), the base UM achieves 2.05 bpc on DSS=1024. At order 12 (contiguous 12-gram), it achieves 0.067 bpc [?]. The sat-rnn achieves 0.079 bpc.

Definition 2 (Isomorphic UM (doubled-E))**.** *The* isomorphic UM *is the doubled-E construction that exactly reproduces the sat-rnn's predictions at 0.079 bpc by translating the RNN's* tanh *activations and softmax output into UM support values [?]. This is our reference model: any structural change must be compared against this baseline.*

Remark 3 (Why start from the isomorphic UM?)**.** *Starting from the isomorphic UM rather than a raw byte model lets us measure the* marginal *contribution of lexical knowledge. The isomorphic UM already captures everything the RNN knows (offset conjunctions, character classes, word boundaries). Adding "the" on top of this tells us: how much does explicit word identity contribute* beyond *what character-level patterns already provide?*

3 Protocol Step 1: Injecting a Single Word

Protocol 4 (Single word injection)**.** *To inject word* w *(e.g.,* $w =$ `the`*) into the UM:*

Phase A: Recognition (input map).

1. *Define the* word detector*: a finite automaton that reads the byte stream and outputs a binary signal* $E_w(t) \in \{1, 0\}$ *indicating whether position* t *is inside an occurrence of* w *(preceded and followed by a word boundary: space, punctuation, or start/end of text).*
2. *For "the":* $E_{the}(t) = 1$ *iff* $d_{t-3} \in \{$`' '`$, \ldots\}$ *and* $d_{t-2} =$ `'t'` *and* $d_{t-1} =$ `'h'` *and* $d_t =$ `'e'` *and* $d_{t+1} \in \{$`' '`$, \ldots\}$*. This fires at the last byte of each occurrence.*
3. *Compute the* recognition map $\rho_w : E_0^k \to E_w$ *from the byte-level context to the word event. This is a deterministic function: given the context bytes, either* w *matches or it doesn't.*

Phase B: Conditional statistics (count tables).

4. *Partition the dataset into positions where* $E_w = 1$ *(inside* w*) and* $E_w = 0$ *(outside* w*).*
5. *For each partition, compute the output byte distribution:*

$$P(o \mid E_w = 1) = \frac{c(E_w{=}1, o)}{c(E_w{=}1)}, \tag{1}$$

$$P(o \mid E_w = 0) = \frac{c(E_w{=}0, o)}{c(E_w{=}0)}. \tag{2}$$

6. *Compute the conditional entropy:*

$$H(O \mid E_w) = P(E_w{=}1) \cdot H(O \mid E_w{=}1) + P(E_w{=}0) \cdot H(O \mid E_w{=}0). \tag{3}$$

Phase C: MI gain measurement.

7. *The MI contributed by E_w (above the existing model) is:*

$$\Delta I(w) = H(O \mid \mathcal{A}_k) - H(O \mid \mathcal{A}_k, E_w), \tag{4}$$

the reduction in output uncertainty from knowing whether we're inside w, given everything the existing model already knows.

8. *The bpc improvement:*

$$\Delta bpc(w) = bpc(\mathcal{A}_k) - bpc(\mathcal{A}_k \cup \{E_w\}). \tag{5}$$

Phase D: Structural analysis.

9. *Record the new patterns created by E_w:*

- *Input patterns: which byte contexts predict $E_w = 1$? (These are the word's* triggers.*)*
- *Output patterns: what does $E_w = 1$ predict about the next byte? (This is the word's* continuation distribution.*)*
- *Cross-patterns: how does E_w interact with existing event spaces? (E.g., does "the" correlate with the in-tag state? With word length?)*

3.1 Expected Results for "the"

"The" is the most frequent word in English, comprising roughly 7% of all word tokens and ∼3.5% of all bytes in typical text. In enwik9 (Wikipedia XML), the frequency may differ due to markup.

Example 5 (Predictions for "the")**.** *On DSS=1024:*

1. $P(E_{the} = 1) \approx 0.02$–$0.04$ *("the" is ∼3 bytes out of 1024, occurring ∼7–13 times).*
2. $H(O \mid E_{the} = 1)$*: very low. Inside "the", the next byte is deterministic: after* `t` $\rightarrow$ `h` *(certainty), after* `h` $\rightarrow$ `e` *(certainty), after* `e` $\rightarrow$ `' '` *(high probability, with some mass on* `'r'` *for "there/their/then/them",* `'m'` *for "them", etc.).*

   Wait—this is where it gets interesting. The word detector fires only on exact "the" (with word boundaries), so after the final `e`*, the next byte is* `' '` *with very high probability (>90%?), because "the" at a word boundary is followed by a space.*
3. $\Delta I(the)$*: the MI gain depends on how much the isomorphic UM already knows. The sat-rnn at 0.079 bpc already captures most character-level predictability. The residual gain from knowing "this is 'the'" is:*

   - *The certainty of* `h` *after* `t`*-in-"the" vs.* `t`*-in-general. The RNN probably already captures this via the bigram pattern.*
   - *The word-boundary prediction after* `e`*-in-"the". This is where word identity helps: after* `e` *in general, many continuations are possible; after* `e` *in "the", only a space (plus rare cases like "the\n").*

Expected ΔI: small but nonzero, perhaps 0.001–0.01 bpc.

Remark 6 (The marginal contribution paradox)**.** *The marginal contribution of "the" may be tiny because the character-level model already predicts its bytes well. The bigram* `t`→`h` *and* `h`→`e` *are high-probability transitions regardless of word identity.*

But this is exactly the point. The tock protocol measures the residual *contribution of word identity above character patterns. If the residual is small, that's informative: it means the character-level model is already capturing most of what word-level knowledge provides.*

If the residual is large, that's even more informative: it means word identity carries information beyond *character patterns, pointing to specific structural features (word-final predictions, syntactic role, semantic context) that character patterns miss.*

4 Protocol Step 2: The Injection Sequence

Definition 7 (Frequency-ordered injection)**.** *Order the vocabulary $V = \{w_1, w_2, \ldots, w_M\}$ by frequency in the dataset, with w_1 = most frequent ("the"), w_2 = second most frequent ("of" or "and"), etc.*

The injection sequence *is:*

$$\mathcal{A}_0 \xrightarrow{+w_1} \mathcal{A}_1 \xrightarrow{+w_2} \mathcal{A}_2 \xrightarrow{+w_3} \cdots \xrightarrow{+w_M} \mathcal{A}_M, \tag{6}$$

where $\mathcal{A}_k$ includes word detectors for $w_1, \ldots, w_k$.

Proposition 8 (The injection curve)**.** *The injection sequence produces a curve:*

$$bpc(k) = bpc(\mathcal{A}_0) - \sum_{j=1}^{k} \Delta bpc(w_j), \tag{7}$$

plotting bpc against number of words injected.

Properties:

1. ***Monotone decreasing****: each word can only help (or be neutral), never hurt prediction. $\Delta bpc(w_j) \geq 0$.*
2. ***Concave****: high-frequency words contribute more than low-frequency words (by coverage), so the curve bends downward. The marginal gain decreases as the lexicon grows.*
3. ***Bounded****: the curve is bounded below by the entropy rate of the source. At some point, adding more words provides negligible gain.*
4. ***Frequency vs. MI****: the optimal injection order is by MI gain, not by raw frequency. A rare word with highly distinctive continuation (e.g., "xylophone" → always followed by specific patterns) may contribute more MI per byte than a common word with generic continuations. In practice, frequency dominates because high-frequency words are seen in more contexts, providing more statistical power.*

4.1 Interaction Effects

Definition 9 (Word interaction)**.** *The interaction between words w_a and w_b is:*

$$\mathit{interaction}(w_a, w_b) = \Delta I(w_a, w_b) - \Delta I(w_a) - \Delta I(w_b), \tag{8}$$

where $\Delta I(w_a, w_b)$ is the MI gain from adding both simultaneously.

Positive interaction: the words together provide more information than the sum of their individual contributions (synergy). Negative interaction: the words are redundant (their contributions overlap).

Example 10 (Expected interactions)**.**

- ***"the" + "a"****: negative interaction (redundant). Both are determiners; knowing one partially predicts the other's distribution. The MI gain from "a" given "the" is already injected is less than the gain from "a" alone.*
- ***"the" + "of"****: small interaction. These words co-occur in "of the" but have largely independent continuation distributions.*
- *"<" + ">": positive interaction (synergy). Knowing both XML delimiters enables tag-state tracking, which neither provides alone. $\Delta I(<,>) > \Delta I(<) + \Delta I(>)$.*

5 What Changes When You Add "the"

Adding "the" to the UM modifies the prediction in specific, measurable ways. Here is the anatomy of a single word injection.

5.1 The Count Table Delta

Definition 11 (Count table delta)**.** *Let $c_0(i, o)$ be the base count table (byte bigrams or skip-k-grams). After injecting w, the count table becomes:*

$$c_1(i, o) = \begin{cases} c_0(i, o) & \text{if position is outside } w, \\ c_w(i, o) & \text{if position is inside } w, \end{cases} \tag{9}$$

where c_w is the word-conditioned count table. The delta is:

$$\Delta c(i, o) = c_w(i, o) - c_0(i, o) \cdot P(E_w = 1). \tag{10}$$

This is the excess *counts attributable to knowing the word.*

For "the":

- $\Delta c($`'t','h'`$)$: positive (the bigram `th` is more common inside "the" than in general).
- $\Delta c($`'e',' '`$)$: positive (after "the", the next byte is almost always space).
- $\Delta c($`'h','e'`$)$: positive (inside "the", `h`→`e` is certain; in general, `h` has many continuations).
- $\Delta c($`' ','t'`$)$: positive (space-then-t often starts "the").

5.2 The Pattern Delta

Definition 12 (New patterns from word injection)**.** *A word w of length ℓ creates:*

1. $\ell - 1$ internal patterns*: deterministic byte-to-byte transitions within the word. For "the": `t` → `h` and `h` → `e` (with certainty, given that we're in "the").*
2. *1* entry pattern*: what precedes the word. For "the": `' '` → `t` (the space-before-the pattern, with strength proportional to $P(\text{the} \mid \text{space})$).*
3. *1* exit pattern*: what follows the word. For "the": `e` → `' '` (the e-followed-by-space pattern, with high strength).*
4. $O(\ell^2)$ skip patterns*: connections between non-adjacent positions within the word. For "the": `t` $\xrightarrow{d=2}$ `e` (skip-bigram).*

Remark 13 (Internal patterns are free)**.** *The internal patterns of a word (`t` → `h`, `h` → `e`) are deterministic within the word: they contribute zero entropy, hence zero bpc cost. A model that* knows *it's inside "the" predicts the internal bytes for free.*

The bpc savings from word injection come entirely from two sources:

1. ***Entry sharpening**: knowing w helps predict when w starts (the entry pattern narrows the distribution over "what comes after a space").*
2. ***Exit sharpening**: knowing w helps predict what follows w (the exit pattern narrows the distribution over "what comes after the last byte of this word").*

For short, frequent words like "the", the exit pattern is where the money is: the character-level model already handles internal bytes well, but the word-final continuation is sharpened by word identity.

5.3 The Prediction Delta

Proposition 14 (Where bpc changes)**.** *The bpc change from injecting w is concentrated at:*

1. ***The exit position** (last byte of w): the biggest gain. The model now knows that after "the" a space is almost certain, whereas before it had to consider all continuations of `e`.*
2. ***The entry position** (first byte of w): a smaller gain. The model now assigns higher probability to `t` after a space in contexts where "the" typically occurs.*
3. ***Positions outside** w: typically zero change. The word detector doesn't fire, so the base model is unaffected.*

6 The Lexicon as Factorization

Definition 15 (Word-level event space)**.** *Given a vocabulary $V = \{w_1, \ldots, w_M\}$, the* word-level event space *is:*

$$E_V = \{w_1, w_2, \ldots, w_M, \texttt{OOV}\}, \tag{11}$$

where `OOV` (out-of-vocabulary) captures all byte sequences not matching any word in V.

The word detector produces the map $\rho : E_0^* \to E_V$ assigning each position in the byte stream to a word event (or OOV).

Proposition 16 (The lexicon is a quotient)**.** *The map $\rho : E_0^* \to E_V$ is a quotient map in the sense of the factorization tower [**?**]. It sends sequences of bytes (fine-grained events) to word labels (coarse-grained events), discarding internal byte structure.*

The factorization tower for text becomes:

$$E_0 \text{ (bytes)} \xleftarrow{\text{recognition}} E_V \text{ (words)} \xrightarrow{\text{prediction}} E_0 \text{ (output bytes).} \tag{12}$$

The word-level ES sits between *the byte-level input and output, acting as a bottleneck that compresses the context into a word identity. This is the tock step's product: a new intermediate event space.*

Remark 17 (Lexicon injection $\neq$ tokenization)**.** *Tokenization (BPE, WordPiece, etc.) replaces the byte sequence with a token sequence, discarding byte-level access. Lexicon injection* adds *word-level events* alongside *byte-level events, preserving full access to both levels.*

This is the factorization tower's product factorization: $E = E_0 \times E_V$, not the sum factorization $E_0 \twoheadrightarrow E_V$. The model can use word identity when it helps and byte identity when it doesn't, without needing to backtrack through a quotient.

7 Building Up: From One Word to the Full Lexicon

Protocol 18 (Systematic lexicon construction)**.** ***Stage 0: Baseline.*** *Run the isomorphic UM (doubled-E) on the dataset. Record bpc at each position. Identify the positions with highest loss (most surprising bytes).*

Stage 1: Function words (top 20). *Inject the 20 most frequent words: the, of, and, in, a, to, was, is, for, on, that, with, as, it, by, from, are, at, an, or. These are short, high-frequency, and syntactically predictable. Expected: small individual Δbpc, but collectively they cover $\sim$25% of word tokens.*

Measure after each injection:

- *Cumulative bpc.*
- *Position-specific bpc changes (which bytes got easier?).*
- *Interaction effects between injected words.*

Stage 2: Content words (top 100–1000). *Inject content words in frequency order. These are longer, more variable, and carry more semantic load. Expected: larger per-word Δbpc (longer words $\to$ more exit-pattern sharpening) but lower coverage per word.*

Stage 3: Morphological families. *Instead of injecting "run", "runs", "running", "ran" as four separate words, inject the* stem *"run" plus morphological suffixes as a product factorization: $E_{run} \times E_{suffix} = \{(run, \emptyset), (run, s), (run, ning), (run, past)\}$.*

This tests whether morphological structure provides MI beyond raw word identity.

Stage 4: Saturation. *Continue injecting words until $\Delta bpc(w_k) < \epsilon$ for threshold ϵ. Record the saturation point: how many words are needed before additional words contribute negligibly?*

Expected saturation: $\sim$5k–20k words for English text. Zipf's law predicts that a small vocabulary covers most tokens.

8 The Measurement Framework

Definition 19 (Word-level diagnostics)**.** *For each injected word w, record:*

1. ***Coverage**: $P(E_w = 1)$ = fraction of positions inside w.*
2. ***Marginal MI**: $\Delta I(w)$ = MI gain from E_w alone.*
3. ***Conditional MI**: $\Delta I(w \mid \mathcal{A}_k)$ = MI gain given the existing architecture.*
4. ***Exit entropy**: $H(O \mid E_w{=}1, \text{last byte of } w)$ = how predictable is the byte after w?*
5. ***Entry entropy**: $H(E_w{=}1 \mid \text{context})$ = how predictable is the occurrence of w from context?*
6. ***Internal entropy**: $H(\text{bytes within } w \mid E_w{=}1)$ = should be ≈ 0 for known words.*
7. ***Pattern count**: how many new patterns does w create?*
8. ***Interaction vector**: interaction(w, w_j) for each previously injected word w_j.*

Proposition 20 (The three regimes)**.** *The injection curve has three regimes:*

1. ***High-gain regime** ($k < 20$): function words. Each word covers many positions. The gain is dominated by exit-pattern sharpening. Interactions are negative (function words are redundant with each other in their character-level patterns).*
2. ***Linear regime** ($20 < k < 5000$): content words. Each word covers fewer positions but contributes more MI per covered position. The gain per word is roughly constant (proportional to $1/k$ coverage times $O(1)$ MI density). Interactions are sparse (content words rarely co-occur in the same context window).*
3. ***Saturation regime** ($k > 5000$): rare words. Coverage per word is tiny. MI gain is negligible. The curve flattens. Further words add no measurable improvement.*

9 Connection to BPE and Subword Tokenization

Proposition 21 (BPE as greedy tock by compression)**.** *Byte Pair Encoding constructs a vocabulary by iteratively merging the most frequent adjacent byte pair. This is a greedy algorithm optimizing* compression *(reducing sequence length).*

The tock protocol constructs a vocabulary by iteratively adding the word with highest MI gain. This optimizes prediction *(reducing bpc).*

These should converge for large datasets: compression and prediction are dual (Shannon's source coding theorem). But they may differ in ordering for small datasets or for words where frequency and MI diverge.

Remark 22 (Why the tock protocol is better than BPE)**.**

1. ***Measurable**: each step's contribution is quantified as Δbpc. BPE gives no per-merge quality metric.*
2. ***Additive**: word events are added alongside bytes, not replacing them. No information is lost. BPE replaces the byte sequence, losing character-level access.*
3. ***Interpretable**: each word event has a name, a coverage, an MI contribution, and a set of patterns. BPE merges are opaque.*

4. ***Composable**: words interact through the product factorization. Morphological structure, phrase structure, and syntactic roles can all be represented as additional event spaces layered on top. BPE is flat.*

10 The Experimental Plan

Protocol 23 (Concrete experimental steps)**.** *1.* ***Implement word detector.*** *A simple finite automaton that scans the byte stream and marks positions inside occurrences of a target word (with word-boundary constraints). Input: byte stream + word string. Output: binary mask.*

2. ***Implement conditional counting.*** *Partition the count table by the word detector's mask. Compute $P(o \mid E_w = 1)$ and $P(o \mid E_w = 0)$. Compute MI.*

3. ***Run on DSS=1024.*** *Start with the isomorphic UM baseline (0.079 bpc). Inject "the". Measure Δbpc. Record the position-specific changes.*

4. ***Inject top-20 words one by one.*** *After each injection, record all diagnostics from the measurement framework. Plot the injection curve.*

5. ***Inject top-1000 words.*** *Run in batch (inject all, measure cumulative). Plot bpc vs. vocabulary size.*

6. ***Compare with BPE.*** *Run BPE tokenization with vocabularies of size 20, 100, 1000. Compare the "equivalent bpc" (if we had a perfect word-level model with each vocabulary) against the tock protocol's actual bpc.*

7. ***Test morphological factorization.*** *Inject stems + suffixes as product ESes for the top-50 verb families. Compare against injecting each inflected form separately. Does morphological structure provide additional MI?*

8. ***Scale to enwik9.*** *Run the full protocol on enwik9 (1B bytes). The KN-6i baseline is 1.784 bpc [**?**]. How much does lexicon injection improve this?*

11 What We Expect to Learn

1. **The word-level contribution to bpc.** How much of the character-level model's remaining error is attributable to lacking word identity? If the gap between the character model and the word model is large, word events are essential. If small, character patterns already capture most of what words provide.

2. **The saturation curve.** How many words are needed? Zipf's law suggests a power-law curve, with $\sim$80% of the gain from $\sim$20% of the vocabulary. The saturation point tells us the "natural" vocabulary size for the data—a quantitative answer to "how many words does this language need?"

3. **Interaction structure.** Which words synergize? Which are redundant? The interaction matrix reveals the syntactic and semantic structure of the vocabulary as seen through the lens of MI.

4. **Entry vs. exit.** Is the bpc gain dominated by entry sharpening (predicting when a word starts) or exit sharpening (predicting what follows a word)? This tells us whether word identity is primarily a recognition phenomenon or a prediction phenomenon.

5. **Morphological value.** Does morphological structure (stems + suffixes) provide MI beyond raw word identity? If yes, the tock protocol should prefer morphological factorization over flat word injection. If no, morphology is a human convenience, not a statistical structure.

6. **The connection to RNN weights.** The weight construction [**?**] built all 82k params from skip-bigram statistics. The lexicon injection adds word-level statistics. Can we construct an improved RNN by incorporating these word-level patterns into the weight construction? This would close the loop: data → words → patterns → weights → predictions.

12 Discussion

The tock protocol makes the tock step operational. Instead of "discover the next event space" as an abstract instruction, we have a concrete procedure: pick a word, build a detector, measure the MI, record the delta. Each step is small, reversible, and measurable.

The protocol also answers a foundational question: *what is a word, computationally?* A word is a recurrent byte pattern that, when recognized, sharpens the model's predictions at its boundaries. A word "exists" in the UM sense when its binary event space (E_w) provides positive MI above the character-level baseline. Words that provide zero MI are not "real words" from the model's perspective—they are merely byte sequences that happen to recur without carrying distinctive predictive information.

This is a precise, falsifiable definition of "word" grounded in information theory, not in linguistic convention. The tock protocol can be run on any language, any dataset, any alphabet. The "words" it discovers are the ones that the data supports—the natural vocabulary of the source.

The tock protocol is the first concrete implementation of architecture-from-evidence for the Universal Model. Each word injected is a tock step. The injection curve is the empirical signature of the tock step's value. And the full lexicon, at saturation, is the domain-native architecture that the data demands.

References

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

[2] Claude and MJC. *The Tock Step: Domain-Native Architecture from Evidence*. Hutter archive, 12 Feb 2026.

[3] Claude and MJC. *Doubled-E: Exact Float UM Isomorphic to RNN*. Hutter archive, 30 Jan 2026.

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

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

[6] Claude and MJC. *The Real Factor Map*. Hutter archive, 9 Feb 2026.

[7] Claude and MJC. *KN Scaling: Byte-Level KN to 1B*. Hutter archive, 12 Feb 2026.

[8] Claude and MJC. *The Carrier Signal Problem*. Hutter archive, 12 Feb 2026.