The Memory Trace

Definition, Replay, and the Path to the Lexicon

Claude and MJC — February 24, 2026

1. What Is a Memory Trace?

A memory trace is the organism's compressed record of experience.

At each moment t, the organism observes a **joint event**: one active event per event space. In a UM with event spaces $\mathrm{ES}_1, \ldots, \mathrm{ES}_k$, let $v_i(t)$ be the active event in ES_i at time t. The joint event at t is the tuple $(v_1(t), \ldots, v_k(t))$.

Under $\mathrm{E} \to \mathbb{N}$, each event space gets a prime p_i, and the event index becomes the exponent:

$$S_t = \prod_{i=1}^{k} p_i^{v_i(t)}$$

S_t is a single integer whose prime factorization encodes every observable fact at time t. Reading off the exponent of p_i tells you which event was active in ES_i.

The memory trace over n observations is the sequence $(S_1, S_2, \ldots, S_n)$. Equivalently, choosing a base $b_{\text{time}} > S_{\max}$ (where $S_{\max}$ is the largest possible state integer), the trace is the number:

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

This is a positional number system in base b_{time}. Each "digit" is S_t — the full state at time t. The base must exceed $S_{\max}$ so that the digits don't overlap; then reading digit t of $P(E)$ recovers S_t exactly. If $b_{\text{time}} \leq S_{\max}$, the summation destroys information.

1.1. The Null Model

The simplest memory trace uses no model at all. With a single event space (the byte ES, $|\mathrm{ES}| = 256$) and base $b_{\text{time}} = 256$:

$$P(E) = \sum_{t=1}^{n} 256^{t} \cdot x_t$$

This is the file itself, written as a base-256 number. The memory trace under the null model *is* the raw data — 10^9 bytes, uncompressed. Every model beyond the null model compresses this by assigning non-uniform probabilities to the events, so that each position costs fewer bits to encode. The digits are all still there; they just cost less each, because the model concentrates probability on the events that actually occur.

1.2. The Organism's Experience as a Number

$P(E)$ is a single (enormous) integer that encodes everything the organism has ever observed. The positional encoding guarantees that this is lossless: given $P(E)$ and the base, we can recover every

joint event at every position.

For enwik9 ($n = 10^9$), with $T_{\text{env}} = 30$: the organism has experienced $\sim 2^{30}$ moments of existence. The trace has 2^{30} digits.

2. Interior Events

Events live in three regions: input (I), output (O), and hidden (H). An **interior event** is one in H: a neuron's activation, a context state, a conjunction pattern firing.

The memory trace of an interior event is its **projection**: the subsequence of the trace restricted to that event space's prime.

If ES_j uses prime p_j, the trace of ES_j is the sequence of exponents:

$$(v_j(1),\ v_j(2),\ \ldots,\ v_j(n))$$

Or via $\text{E} \to \mathbb{N}$, the per-ES product:

$$M_j = p_j^{\sum_t v_j(t)}$$

This exponent $\sum_t v_j(t)$ is the total count of event v_j across all time — exactly what LSI approximates as 2^{s_j}.

Projection property: the full trace factors into per-ES traces, each independently readable. No information from other ESes leaks in; no information from this ES is lost.

3. Boolean Algebra

The forward pass at each position computes a boolean sentence over events:

$$\text{output}[b] = \bigvee_{\text{LPPs}} \big(\text{source\_event} \ \wedge\ w(\text{source}, b)\big)$$

where $\bigvee = \max$ (disjunction) and $\bigwedge = \min$ (conjunction). Each disjunct is one LPP's contribution.

This sentence IS the memory trace's content at position t: what the organism "knows" that produces its prediction. Every correct prediction and every error is recorded in the trace as a boolean sentence over events.

This is why memory traces matter: they make the model's reasoning visible. Not as a post-hoc explanation, but as the literal computation that produced each prediction.

4. Clearing and the Write Event

An event space may be **cleared**: its support goes to zero, losing the information it carried. This is the biological default — neural firing is transient, not sustained.

Clearing calls for a write. Before an ES is cleared, its current content — the support vector — must be committed to long-term storage. Otherwise the information is lost. The clearing operation is therefore the nearby trigger of a **memory trace write event**: the organism commits its current trace, then clears the working ES.

4.1. The Write Target

The write target is T — the organism's long-term storage. In the CMP formalism, T is a capability directly exposed by the environment, external to the LPPs.

The memory trace and the model are **separate**. The model is the organism's learned structure: event spaces, LPPs, patterns, weights. The memory trace is the compressed residual: the data that the model alone cannot reconstruct. Together — model plus trace — they are sufficient to recover the original experience.

5. Decomposition: Model + Residual

The memory trace decomposes into two parts:

1. **Model**: the learned structure (ESes, LPPs, weights). This is the organism's understanding of the domain — the patterns it has discovered. The model is stored separately from the trace.
2. **Residual** (the trace itself): the information that the model doesn't predict — the actual sequence, compressed under the model's predictions. The total residual is $\sum_t -\log_2 p(x_t)$ bits. The encoding resets at word boundaries (§9.1), discarding inter-word predictive value — which is negligible until the lexicon exists in H to exploit it.

For the unigram case: model = 256 bytes (log-support values), residual = 681 MB, total $\approx$ 681 MB.

As the model improves, the residual shrinks. The model grows (more LPP entries, more ESes), but the residual shrinks faster. The sum — model size plus residual — decreases monotonically if the model is learning real structure.

6. Replay: The Tick-Tock Process

The memory trace is not just a record. It is a **training signal**.

The process:

1. **Observe**: the model reads the input (enwik9) once, learning online. It writes a memory trace — the compressed residual under its current predictions.
2. **Freeze**: the model is frozen. The trace is now decodable only with this frozen model.
3. **Extend**: the model architecture grows. New event spaces, new LPPs, new neurons ($+E$, $+P$). The frozen weights are preserved; new structure is added on top.
4. **Replay**: the extended model replays the memory trace. Because the frozen sub-model can still decode the trace, the extended model sees the original data — but now with more structure to learn from it.
5. **Rewrite**: the extended model writes a new, shorter memory trace. If it is not shorter, we stop and reconsider.

Each cycle is a **Tick** (replay = training on the full input) followed by a **Tock** (architecture growth). The organism reads the original input exactly once. Every subsequent pass is a replay of its own memory.

This is learning: the model and its memory co-evolve, each cycle producing a better model and a shorter trace.

7. Querying the Trace: Look at the Data

What good is a memory if you don't access it?

Between the Tick and the Tock — after the model has been trained but before the architecture is extended — we **query** the trace to understand where the model currently is. Querying is the most powerful tool we have, both to improve the model and to diagnose it.

The principle is **LATD**: Look At The Data. Not aggregate statistics, not distributions over distributions, not summary metrics — the actual cases where the model is right and the actual cases where the model is wrong. We have the tools to do this because the memory trace makes every prediction and every error individually visible.

7.1. Anatomy of a Query

A query starts from an error. The model predicted wrong at position t. From the boolean sentence (§3), we know exactly which pattern fired and why it predicted what it did. A query traces this backward:

1. **The error**: at position t, pattern P fired with source event e, predicted byte b', actual byte was b. The sentence says why: "e was active AND the weight $w(e, b')$ was highest."
2. **The pattern's evidence**: pattern P learned its weights from specific positions in the dataset — the positions where event e was active and the model observed the output. These are concrete, listable positions. We can display them with their surrounding context.
3. **The exceptions**: among those positions, some had output b' (the cases that taught the pattern its current prediction) and some had other outputs (the cases where the rule doesn't hold). The error at position t is one of these exceptions. The query returns *all* of them.

The result is not a number. It is a set of positions from the dataset: the cases that explain the pattern, combined with the cases where the pattern fails. The error is totally explainable in data-pattern terms, to whatever explanatory depth we want — we can always look at more cases.

7.2. Example

The model has learned: after "`th`", predict "`e`". At position 47,203, the actual byte is "`i`" (the word is "`this`"). The query returns:

> Pattern: `th` → output. Learned from 12,847 positions.
> `th`→`e`: 8,203 cases ("the", "these", "them", ...)

`th`→`i`: 1,891 cases ("this", "thin", "think", ...)
`th`→`a`: 1,402 cases ("that", "than", "thank", ...)
`th`→`o`: 876 cases ("those", "though", ...)
...

This is not a histogram. It is a list of actual positions, each viewable in context. The error at position 47,203 is explained: the pattern correctly learned that "`e`" is most common after "`th`", but position 47,203 is one of the 1,891 "`i`" cases. The model needs a longer context ("`thi`" vs "`the`") to distinguish them — which is exactly the structure the next Tock should add.

7.3. Queries Inform the Tock

The Tick produces a trained model and a trace. Queries reveal what the model doesn't know — not as aggregate statistics, but as specific data cases that the model gets wrong and the specific patterns responsible. The Tock adds structure to address those cases: new event spaces, new LPPs, new patterns that distinguish the exceptions from the rule.

Without LATD, the Tock is blind. With it, we see the actual data behind every error, and the architecture growth is targeted at real cases rather than guessed from summary metrics.

8. Embeddings as Change of Base

An embedding is a change of base of an event stream.

Under the null model, events arrive in base 256 (one byte per position). The memory trace is the raw file. Under the unigram model, we use the byte frequency distribution to compress, but the base is still 256 — one event per position.

To move beyond bytes, we change the base. A word-level event space has $|\mathrm{ES}_{\mathrm{word}}| \gg 256$ events, but many fewer positions (one per word, not one per byte). The memory trace in this higher base has fewer digits, each carrying more information.

8.1. Chunking and the Reset Signal

Arithmetic coding is continuous: each symbol's probability subdivides an interval. The encoded bitstream is opaque and not interpretable in the middle. Huffman coding chunks: each symbol maps to a fixed code, interpretable at chunk boundaries.

The difference is the **reset signal**. At chunk boundaries, we evaluate against a static distribution rather than a distribution computed in the moment. For byte-level English, the natural reset signal is the space character: at word boundaries, we have relatively high uncertainty about what comes next.

This suggests the architecture: accumulate byte-level events into a working memory ES, and on the reset signal (space), emit a word-level event to the memory trace. The word event summarizes everything observed since the last reset.

8.2. Bag of Letters

The word-level event is not semantic — it is **orthographic**. It is constructed by looking *down* toward letters, not up toward word sequences. The word "the" produces:

> "There is a `t`." ∧ "There is an `h`." ∧ "There is an `e`." ∧ "There is a `th` bigram." ∧ "There is an `he` bigram."

The "the" event is the nexus of these orthographic observations in the dataset. It is also the nexus of upward-looking observations — that "the" appears before nouns, etc. — but we do not yet have those; they come later, as the factor tower grows.

The bag-of-letters embedding can then be specialized with a few additional bits to give the exact spelling. The difference between "the" and "teh" is orthographically distinct (different bigram events) but semantically negligible. The embedding captures the common structure; the residual captures the rare variations.

9. The Path Forward

The goal is not to replicate n-gram models in the UM. KN-6 demonstrates what a small model on bytes can learn, but anything that does not reach the lexicon is a dead end. The path forward:

1. **Bag of letters**: accumulate byte events into working memory. On space (reset signal), write a word event. This produces an orthographic embedding — a change of base from 256 to $|\mathrm{ES}_{\mathrm{word}}|$.
2. **Lexicon**: the vocabulary of word events, discovered from the data. The Tick-Tock process discovers which letter combinations recur and creates events for them.
3. **Upward**: once the lexicon is in the model, the same process continues. Word events become the input to phrase-level patterns, then sentence structure, then discourse, then world knowledge. Each level is a Tick (replay with new structure) and a Tock (architecture growth).

Compression falls out of understanding. Each level of the factor tower captures real structure in the data — not by engineering n-gram tricks, but by building the organism's model of the domain. The memory trace at each level is the proof: it is shorter than the trace at the level below, because the model understands more.

10. Summary

1. A memory trace is a sequence of joint events, encoded as digits in a positional number system. Base b_{time} must exceed $S_{\max}$; otherwise the encoding destroys information. Under the null model (base 256), the trace is the raw file.
2. Interior events have their own traces: projections onto their primes.
3. The forward pass at each position is a boolean sentence over events — the trace's content at that moment, making every prediction and every error visible.

4. Clearing an ES triggers a write event: the organism commits its trace to T before the information is lost. The model and the trace are separate.

5. Replay is learning: freeze the model, extend the architecture, replay the trace, write a shorter trace. Each cycle is a Tick (training) + Tock (growth).

6. LATD: Look At The Data. Queries trace each error back to the pattern that caused it, the data positions that taught the pattern, and the specific exceptions where the pattern fails. Not aggregate statistics — actual cases, to whatever explanatory depth we want.

7. An embedding is a change of base. The bag-of-letters embedding changes base from bytes to word events, chunking at the space reset signal. This is orthographic (looking down), not semantic (looking up).

8. The path leads through the lexicon and upward. Compression falls out of understanding.