
LTMi-XT: A Retrieval and Training Record Format with Hash-Derived Lattice Addressing

Format specification v0.1, with the v0.3.1 semantic amendment to the lattice field

Thomas Garren

SOPHIA XT LLC

19 September 2026 · Specification v0.1 (format), v0.3.1 (semantics)

sophiaxt.com

CC BY 4.0. Code referenced herein is Apache 2.0.

Abstract

LTMi-XT, Layered Topological Memory Indexing, is an open record format for retrieval-grounded generation. One record, called a locus, carries a single self-contained statement, a four-level breadcrumb path, a deterministic three-integer lattice address derived from that path, byte-offset provenance into its source, a memory horizon with a decay weight, and a confidence score. A bundle of loci serves two consumers without conversion: a retriever answering queries with provenance attached, and a fine-tuning pipeline reading the same file as training rows. This document specifies the container, both records, the identifier derivation, the lattice derivation, the horizon model, and the reference retrieval algorithm in enough detail to implement from. It also reports a published null result: a three-way ablation found the lattice coordinate carries no semantic signal a model can learn from, so the field is specified here as a deterministic address and nothing more. The format has not been benchmarked against alternative retrieval approaches and this document claims no accuracy advantage over any of them.

1. Status of this document

The wire format specified here is v0.1 and has not changed since 2026-05-08. What changed is the meaning assigned to one field. The v0.3.1 amendment of 2026-05-12 reclassified the *lattice* field from a model-conditioning signal to a deterministic per-locus address. Section 10 gives the evidence. Readers of the earlier v0.1 PDF should treat this document as replacing it.

Three further corrections are folded in. The lattice derivation is now stated once, normatively, matching the shipped reference implementation, because the three derivations previously in circulation disagree (section 6.3). The horizon half-life is corrected to the value the reference implementation uses (section 7). The retrieval-accuracy comparison that appeared in the earlier PDF is withdrawn (section 11).

Keywords MUST, MUST NOT, SHOULD and MAY carry their usual specification force. The reference implementation is the TypeScript tree under *reference/ts/* at github.com/Chorozion/LTMi-XT, released Apache 2.0. This document is CC BY 4.0.

2. What the format is for

A retrieval system has to commit to a representation of what it retrieved. Chunk-and-embed is the common choice and it has three costs. Chunk boundaries cut claims in half. Retrieval runs behind a learned similarity function, so the index cannot be reproduced without pinning a model version. And the stored unit has no place to record where the text came from, how confident the extractor was, or whether the claim has gone stale.

LTMi-XT fixes the unit instead of the search. A locus is one claim, written as one sentence that stands on its own, with the byte range it came from attached. Its address in the index is computed from its breadcrumb path by a hash, so two machines with no shared model produce the same index from the same input. Its horizon field records whether the claim is settled or provisional. The same file is readable as training data because a locus already has the shape of a question-and-answer pair: the breadcrumb is the question side and the statement is the answer side.

The format does not attempt to be a better ranking function. Section 8 describes the reference retriever, which is deliberately simple, and section 11 says what is not known about how it compares to anything else.

3. The bundle

3.1 On-disk layout

A bundle is a directory. The primary file is JSONL; the rest are companions that **MUST** be reproducible from the primary file alone, so a producer **MAY** ship only the primary file for transport.

my-corpus.ltm/	
corpus.ltm	primary JSONL: manifest line, then loci
manifest.json	the manifest line again, for 0(1) reads
breadcrumb-tree.json	nested navigable view of the paths
instruction.md	usage instruction written for a model
provenance.csv	locus to source to offsets, tabular
sources/	verbatim source texts, one per source id

The directory MAY be packaged for transport, in which case the recommended extension is *.ltmi.tar.gz*. A bare primary file uses *.ltmi*, and the recommended media type is *application/x-ltmi+json*. Sources are stored verbatim rather than by reference so a bundle is self-contained: a consumer can re-run crystallization against the stored text and check that the loci it gets match the loci it was given.

3.2 Container rules for corpus.ltm

- Encoding is UTF-8.
- The line separator is LF. CRLF MUST NOT be used.
- One JSON object per line. Multi-line objects MUST NOT be used.
- The first line MUST be a manifest record.
- Every subsequent line MUST be a locus record, to end of file.
- Trailing blank lines are not permitted.

Readers MUST preserve fields they do not recognise when they round-trip a bundle. This is what lets a v0.1 tool sit in a pipeline beside a newer producer without silently dropping data.

3.3 Shared field types

- **Identifier:** a type prefix plus 32 lowercase hex characters. Loci use *a-*, sources *s-*, corpora *c-*. Derivation is in section 5.
- **Timestamp:** ISO-8601 UTC with a Z suffix, for example *2026-05-08T12:34:56Z*.
- **Lattice coordinate:** three integers, each in [0, 63].
- **Confidence and decay:** floats in [0.0, 1.0].

4. Records

4.1 The manifest record

Field	Req	Type	Value
v	yes	string	Format version. MUST be "ltmi/0.1".
kind	yes	string	MUST be "manifest".
corpus_id	yes	string	Content-addressed corpus id, c- prefix.
loci	yes	integer	Number of locus records that follow.
lattice	yes	object	{"dim": 64, "shape": "cube"}.
created	yes	string	Bundle creation time.
sources	yes	array	Source ids referenced by the loci.
producer	no	string	Producer software name and version.
crystallizer_model	no	string	What produced the statements.
topologizer_model	no	string	What produced the breadcrumbs.
notes, tags	no	string, array	Free-form.

A consumer SHOULD reject a bundle whose v it does not recognise. It MAY ignore unknown optional fields inside a

version it does recognise.

4.2 The locus record

```
{
  "id": "a-<32 hex>",
  "breadcrumb": ["topic", "subtopic", "concept", "claim"],
  "lattice": [x, y, z],
  "statement": "<one self-contained sentence>",
  "kind": "fact",
  "confidence": 0.0 to 1.0,
  "horizon": "short" | "long",
  "decay": 0.0 to 1.0,
  "source": { "id": "s-<32 hex>", "offset": [start, end] },
  "first_seen": "<ISO-8601 UTC>",
  "last_referenced": "<ISO-8601 UTC>",
  "references": 0
}
```

Field	Req	Notes
id	yes	Content-addressed. Derived in section 5, not assigned.
breadcrumb	yes	Length exactly 4: topic, subtopic, concept, claim. Trailing nulls are allowed for shallower paths. A leading null is not.
lattice	yes	Three integers in [0, 63], derived from the breadcrumb by section 6. A consumer MAY recompute it and reject a locus that disagrees.
statement	yes	One sentence, at most 600 characters. MUST NOT open with an unresolved pronoun. MUST NOT bundle two claims.
kind	yes	One of: fact, definition, claim, example, instruction, opinion, uncertainty. Defaults to claim when absent.
confidence	yes	The producer's confidence that the statement is faithful to the source span, not its confidence that the claim is true.
horizon	yes	Either "short" or "long". There is no third value.
decay	yes	Current weight. A long-horizon locus MUST have decay exactly 1.0.
source	yes	Source id plus byte offsets [start, end) into that source. start must be less than or equal to end and both must be valid for the source.
first_seen	yes	When the locus entered the bundle.
last_referenced	yes	When retrieval last returned it.
references	no	Retrieval counter, default 0. Drives consolidation.
relations	no	Array of {type, target} where target is a locus id. Free-form graph.
extraction_pass	no	1 for first crystallization, 2 or more after edits.
notes, tags	no	Free-form.

Application-specific fields MUST use the *x-* prefix for experimental keys or the underscore prefix for implementation-internal keys, which a consumer SHOULD strip. Everything else in the key space is reserved.

5. Identifiers

Identifiers are content-addressed, so two producers given the same input emit the same ids and bundles from different machines can be compared directly.

```
locus_id = 'a-' + h128(canonical_json({breadcrumb, statement, source}))
source_id = 's-' + h128(source_text)
corpus_id = 'c-' + h128(canonical_json(sorted(list_of_locus_ids)))
```

5.1 The hash: h128

h128(s) is the first 32 hex characters of the BLAKE2b-512 digest of s encoded as UTF-8. This is a truncation, not a parameter change, and the distinction is load-bearing: BLAKE2b mixes its output length into the initial state, so BLAKE2b with a 128-bit digest length gives a different value than a truncated 512-bit digest of the same input. An implementation that reaches for the 128-bit variant of its hash library will produce identifiers that do not match any other implementation. Truncate.

5.2 Canonical JSON

Canonical JSON here is a subset of RFC 8785. Objects sort their keys lexicographically at every level, no whitespace is

emitted anywhere, undefined or absent values are dropped rather than serialised, strings are escaped per ECMA-404, and non-finite numbers are an error. The number-normalisation rules of full RFC 8785 are not required, because every value that reaches the hash in this format is a string, an integer, or an array of those. An implementation that uses full RFC 8785 is conforming; the subset exists so that a small implementation does not have to carry a number formatter.

5.3 Test vector

Computed from the reference implementation. An implementation that reproduces this line has its hash, its truncation, and its canonical form all correct.

```

breadcrumb = ["AI", "Diffusion Models", "Masked Diffusion",
              "Architecture"]
statement  = "Cassandra T1 uses 28 transformer layers with a
              2,048 hidden size."
source     = {"id": "s-9a8b7c6d5e4f3a2b", "offset": [1245, 1308]}

canonical_json ->
{"breadcrumb":["AI","Diffusion Models","Masked Diffusion",
"Architecture"],"source":{"id":"s-9a8b7c6d5e4f3a2b","offset":
[1245,1308]},"statement":"Cassandra T1 uses 28 transformer layers
with a 2,048 hidden size."}

locus_id -> a-c9734de9fefbb93969d0501e2952e037

```

6. The lattice address

6.1 Normative derivation

The address uses the first three breadcrumb levels. The fourth level, the claim, is deliberately excluded, so every claim under

```

def lattice_coord(breadcrumb):
    topic    = breadcrumb[0] or ''
    subtopic = breadcrumb[1] or ''
    concept  = breadcrumb[2] or ''

    hx = h128('ltmi/topic:' + topic)
    hy = h128('ltmi/sub:'   + topic + '|' + subtopic)
    hz = h128('ltmi/con:'   + topic + '|' + subtopic + '|' + concept)

    # first byte of each digest, i.e. int(hex[:2], 16)
    return (first_byte(hx) % 64,
            first_byte(hy) % 64,
            first_byte(hz) % 64)

```

h128 is the function from section 5.1. The lattice is a cube of 64 cells on each axis, 262,144 cells in total.

6.2 What the address guarantees

Loci sharing the first k breadcrumb levels share the first k lattice coordinates, for k in 1 to 3. That is mechanical, not statistical: the same input string hashes to the same byte. It gives a retriever a cheap hierarchical pre-filter, because a Chebyshev neighbourhood around a query cell contains every locus whose path is close to the query's path. It does not give

one concept shares a cell. Missing levels are treated as the empty string. Each axis hashes a domain-separated string and takes the first byte modulo 64.

the retriever any notion of similarity between paths that do not share a prefix. Two neighbouring cells are neighbours by accident of hashing, and a retriever MUST NOT read meaning into the distance between loci that share no prefix.

6.3 Three derivations, one normative

Three incompatible derivations have circulated. They produce different coordinates for the same breadcrumb, shown here for the test-vector path above. Only the first conforms.

A bundle written by one of the lower two rows will not validate against a reader implementing the top row. Producers of existing bundles should recompute coordinates; the loci themselves are unaffected, because the locus id does not include the lattice field.

Derivation	Coordinate	Status
Domain-separated prefix, first byte mod 64 (section 6.1)	[63, 11, 53]	Normative. What the reference implementation computes.
Slash-joined lowercased prefix, BLAKE2b-128, first four bytes big-endian mod 64	[24, 24, 28]	Non-conforming. Published in the v0.1 PDF of 2026-05-09.
Concatenated prefix, BLAKE2b-128, first byte AND 63	[29, 19, 44]	Non-conforming. Sketched in paper.md section 5.2.

The example locus in the repository's file-format document carries an illustrative coordinate that matches none of the three. It is prose, not a test vector. Use the table above.

7. Horizon, decay and consolidation

Operational corpora have a long tail of settled reference material and a busy head of recent material that will be wrong next month. A flat index weights both the same. LTMi-XT separates them with two horizons and a decay weight.

7.1 Assignment at ingest

- A source flagged canonical initialises all its loci long.
- A source flagged transient initialises all its loci short.
- Otherwise, a source whose timestamp is inside the recency window (default 30 days) initialises short, and an older

source initialises long.

7.2 Decay

A long-horizon locus has decay fixed at 1.0 and does not fade. A short-horizon locus decays by elapsed days since its last reference:

```
decay = exp(-ln(2) * delta_days / half_life)    # half_life = 7 days
```

The half-life is a deployment parameter and the reference implementation uses seven days. An earlier draft of this document gave one hour, which was the value a browser demonstration used so that decay would be visible inside a single page visit. Seven days is the value to implement.

7.3 Consolidation and eviction

Retrieval increments *references* and refreshes *last_referenced* on every locus it returns. A short-horizon locus whose reference count reaches the consolidation threshold, default 3, is promoted to long and its decay is reset to 1.0. Promotion is one-way in v0.1; there is no demotion path and no third horizon.

A short-horizon locus whose decay has fallen below 0.05 with no reference in 90 days is eligible for eviction during a compaction pass. Eviction removes the locus from the index and leaves its source text in place, so the claim can be recovered by re-crystallizing.

8. The reference retrieval algorithm

This is what the reference implementation does. It is one reasonable way to read a bundle, not a required one, and section 11 is honest about how little is known about how well it works.

```
def retrieve(query, loci, radius=4, k=8):
    bc_q = derive_query_breadcrumb(query) # may return None
    cell_q = lattice_coord(bc_q) if bc_q else None

    scored = []
    for locus in loci:
        if cell_q:
            d_lat = chebyshev(locus.lattice, cell_q)
            if d_lat > radius:
                continue
            d_pref = prefix_match(locus.breadcrumb, bc_q) # 0..4
            score = (0.4 * max(0, 1 - d_lat / radius)
                    + 0.2 * locus.confidence
                    + 0.1 * locus.decay
                    + 0.3 * (d_pref / 4))
        else:
            overlap = keyword_overlap(query, locus) # 0..1
            score = (0.7 * overlap
                    + 0.2 * locus.confidence
                    + 0.1 * locus.decay)
            if score <= 0.05:
                continue
        scored.append((score, locus))

    top = nlargest(k, scored)
    touch([l for _, l in top]) # references += 1, may consolidate
    return top
```

The query-side breadcrumb is the one part of the pipeline that is not deterministic. The reference implementation asks a language model to assign a path to the query, at temperature 0, and falls back to keyword overlap when that call fails or returns something unparseable. Callers that already know the path, for instance a navigation UI where the user clicked into a subtree, SHOULD pass the breadcrumb directly and skip the call. Chebyshev distance is the max of the three per-axis absolute differences. `prefix_match` counts leading breadcrumb levels that are equal and non-null, stopping at the first mismatch.

Results SHOULD be presented to a downstream model with the full breadcrumb path attached, not as bare sentences. The path is the context that a chunk does not have, and dropping it discards most of what the format is for.

9. Reading a bundle as training data

The second consumer reads the same file as supervised pairs. The default conversion templates a question from the breadcrumb and takes the statement as the target. The conversion is deterministic, so two runs over one bundle produce byte-identical training data, which is what makes a training set reproducible from a corpus id.

The structure also lines up with asymmetric masking objectives. In anchor-token masking, the prefix is never masked and the target is always subject to the noise schedule; a locus already separates the two, so no reshaping step sits between the retrieval corpus and the training corpus. That work is reported separately and its scope is stated there, not here.

10. The lattice ablation: a published null

The earlier version of this document described the lattice as a semantic geometry that a model could learn to use. An ablation published on 2026-05-12 in the reference repository shows it is not. The result is reproduced here because a reader of the specification should not have to find it in a separate file.

Every interval spans zero. All three runs landed at the same training loss. Unforced generation was byte-identical between the random and the hashed arm, with a zero-width bootstrap interval, which means the trained weights were functionally the same model.

Comparison against the breadcrumb hash	Point	95% interval
PCA projection, forced corpus overlap		-0.002 [-0.034, +0.029]
PCA projection, forced English ratio		+0.011 [-0.012, +0.033]
Random coordinate, forced corpus overlap		-0.016 [-0.054, +0.022]
Random coordinate, forced English ratio		-0.013 [-0.044, +0.016]
Random coordinate, unforced generation		0 [0.0000, 0.0000]

The projection scheme does produce a coordinate space with topic structure: it reaches 96.7% nearest-neighbour topic accuracy in coordinate space against 21.7% for the hash. None of that geometry reached the model. A coordinate space can be beautifully organised and still carry nothing a network can use, and that is what happened here.

10.3 What this does and does not mean

It means the lattice field is a deterministic address, not a learned semantic geometry. That is how section 6 now specifies it, and it is why the prefix-locality property in section 6.2 is described as mechanical rather than as evidence of anything.

It does not mean the format failed. What was tested was a speculative extra use of one field: feeding coordinates into an experimental attention path as a conditioning signal. The file format, the identifier scheme, the provenance model, the horizon model and the retrieval pipeline were not on trial and are not affected by the result. The format is in production use behind live endpoints with a public specification, and it continues to do the job it was written for. One extra claim was made for one field, the claim was tested, and the claim did not survive. The field stays, with its job correctly described.

An adversarial review of the ablation raised three points worth carrying forward: 36 held-out pairs is a small sample against a noise floor of roughly 0.02 from non-deterministic floating-point kernels; the conditioning path as implemented gave the coordinate too weak a gradient to matter, so the null bounds that implementation rather than the idea in general; and prior published work had already documented this class of partial-attribution failure in retrieval-conditioned architectures. The findings document lists that prior work.

10.1 What was tested

Three training runs, identical in every respect except how each locus got its coordinate: the breadcrumb hash specified in section 6, a three-dimensional PCA projection of frozen-encoder embeddings quantised onto the same cube, and a uniform-random coordinate per locus from a fixed seed. Same architecture, same 144 training triples, same warm start, same optimiser, same 500-step schedule. Evaluation was a paired-by-query bootstrap with 2,000 resamples over 36 held-out query and corpus pairs across three domains, under both forced-anchor and unforced decoding.

10.2 Result

11. What this specification does not establish

It does not establish that this retrieval approach is better than any other. LTMi-XT has not been benchmarked against alternative retrieval approaches in any way that would support a comparative claim, and this document makes none. There is one small comparison in the repository, run on three corpora each under 1.1 kilobytes with 15 queries, in which one arm makes a model call and the baselines do not. It is too small and too confounded to carry a conclusion and the earlier version of this document should not have summarised it as a result. Anyone choosing a retrieval strategy should measure on their own corpus.

It does not establish that the lattice improves retrieval. The hierarchical pre-filter is cheap and reproducible. Whether filtering by breadcrumb neighbourhood beats not filtering at all has not been measured. The repository's own current guidance is to rank with a classical information-retrieval primitive and treat the lattice as an address.

It does not establish crystallization fidelity. No human-graded precision and recall against a gold-standard set of atomic claims has been run. Loci look plausible and breadcrumb assignment looks consistent, which is an observation, not a measurement.

It does not establish behaviour at scale or over time. Published bundles are small. The horizon and decay model is implemented and specified but has never been measured over a period long enough for decay to matter, which would take weeks of real reinforcement events.

12. Limitations of the format itself

- **The breadcrumb is a commitment.** The address is fixed at crystallization time. Re-deciding that a claim belongs

under a different concept moves its cell, and a consumer holding the old coordinate will not find it.

- **Four levels is a choice, not a result.** Three levels ran together claims that needed separating and five thinned cell populations, but that judgement came from looking at internal corpora, not from an experiment, and it is reported here as a design decision.
- **Cross-hierarchy similarity is lost.** Two claims that mean nearly the same thing under different topics are far apart and always will be. Deployments that need that similarity should compose the lattice pre-filter with a semantic search step rather than expecting the lattice to supply it.
- **Query-side breadcrumb derivation is the weak joint.** The lattice path depends on a model assigning a path to the query, and when that fails the system silently drops to keyword overlap, which misses whenever the query and the target share no vocabulary.
- **Crystallization is not reproducible across model runs.** Coordinates are deterministic given a breadcrumb, and ids are deterministic given a locus, but the statements and breadcrumbs themselves come from a model call that is not byte-stable. Determinism in this format starts after extraction, not before it.
- **Offsets go stale.** Byte-offset provenance is exact against the source as stored. Edit the source without re-crystallizing and every offset into it is wrong. This is

detectable, since the stored source is content-addressed, but detection is the consumer's job.

13. Conformance

A producer conforms to v0.1 when every bundle it emits satisfies:

- Every required field of every record is present and well-typed.
- Every locus id is derivable per section 5.
- Every lattice coordinate is derivable per section 6.1.
- Every source offset is valid against the source it names.
- Every long-horizon locus has decay exactly 1.0.

A consumer conforms to v0.1 when:

- It accepts any bundle whose manifest version is "ltmi/0.1".
- It preserves unknown optional fields across a round trip.
- It rejects bundles with missing or malformed required fields.

Backwards-incompatible changes increment the minor version within the 0.x series. Version 1.0 will mark a stabilised format. A producer SHOULD emit only the highest version it supports.

14. License and citation

This specification is released under Creative Commons Attribution 4.0. The reference implementation is released under Apache 2.0.

```
@techreport{garren2026ltmixt,
  author      = {Garren, Thomas},
  title       = {LTMi-XT: A Retrieval and Training Record Format
                with Hash-Derived Lattice Addressing},
  institution = {SOPHIA XT LLC},
  year        = {2026},
  url         = {https://sophiaxt.com/research/ltmi-xt}
}
```

References

- [1] LTMi-XT reference implementation and specification sources. Apache 2.0. github.com/Chorozion/LTMi-XT
- [2] Empirical findings, 2026-05-12. The three-way coordinate ablation summarised in section 10, with full tables, the adversarial review, and the list of corroborating prior work. github.com/Chorozion/LTMi-XT/blob/main/docs/empirical-findings-2026-05-12.md
- [3] Rundgren, A., Jordan, B., and Erdtman, S. (2020). JSON Canonicalization Scheme (JCS). RFC 8785.
- [4] Min, S., Krishna, K., Lyu, X., et al. (2023). FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. EMNLP 2023.
- [5] Liu, X., et al. (2025). Atomic Fact Extraction and Verification. arXiv:2506.07446.
- [6] Garren, T. (2026). Anchor-Token Masking Produces an Out-of-Distribution Generalization Advantage in Retrieval-Grounded Masked-Diffusion Language Models. SOPHIA XT LLC. doi:10.5281/zenodo.20496783