LTMi-XT v0.1 — A Retrieval Format with Hash-Derived Topological Indexing

Thomas Garren[1]

May 9, 2026

Abstract.

Abstract

LTMi-XT (Layered Topological Memory indexing — Extended Technology) is an open Apache-2.0 file format for retrieval-grounded generation. Each record (a *locus*) consists of a self-contained atomic statement, a four-level breadcrumb hierarchy, a deterministic 3D lattice coordinate derived by hashing the breadcrumb prefix, byte-offset provenance back to source, a temporal horizon (short / long / archived), and a confidence score. The format is dual-purpose: it supports breadcrumb-anchored retrieval (LLM answers grounded to specific loci with full provenance) and direct ingestion as fine-tune training rows. This document specifies the format, its design rationale, and its relationship to related work in atomic-fact retrieval. Reference implementation and benchmarks are released under Apache 2.0 at `github.com/Chorozion/LTMi-XT`.

---

1. Motivation

Retrieval-augmented generation (RAG) systems must commit to a representation of retrieved knowledge. The dominant choice — embedding chunks into vector spaces and retrieving by approximate cosine similarity — has known weaknesses: chunk boundaries fragment claims, embedding models gate retrieval behind a learned similarity function, and the retrieval system has no explicit handle on factual atomicity, provenance, or temporal validity.

Atomic-fact decomposition [@min2023factscore; @afev2025] addresses the chunk-boundary and atomicity problem by splitting prose into self-contained factual claims. This improves verification but does not by itself address indexing, provenance, or temporal validity.

Memory-layer products (Mem0, Letta, Zep) layer memory hierarchies on top of LLM agents but typically do not expose a portable open-format specification for the underlying record structure, making interchange and direct fine-tune ingestion difficult.

LTMi-XT is the format we converged on after building several internal retrieval pipelines that needed to support both (a) breadcrumb-anchored retrieval at inference time and (b) anchor-token-masked fine-tune training [@garren2026anchortokenmasking] from the same data. The format prioritizes:

- **Atomicity** — each record encodes a single self-contained claim
- **Reproducible indexing** — coordinates derived deterministically from content, no embedding model required
- **Provenance** — byte-offsets back to source for every claim
- **Temporal awareness** — explicit short-term / long-term horizon with consolidation rules
- **Format portability** — JSONL-native, Apache 2.0 open spec, no service dependency
- **Dual consumption** — same artifact serves retrieval and training

2. Format definition

2.1 Bundle structure

An LTMi-XT bundle is a JSON document with the following top-level structure:

```
{
  "success": true,
  "manifest": { ... manifest metadata ... },
  "loci": [ ... array of locus records ... ]
}
```

The bundle MAY be serialized as a single JSON document or as JSONL (manifest on first line, loci on subsequent lines). Both are equivalent at the data level.

2.2 Manifest schema

```
{
  "v": "ltmi/0.1",
  "kind": "manifest",
  "corpus_id": "c-<blake2b-128-hex>",
  "loci": <integer count>,
  "lattice": { "dim": 64, "shape": "cube" },
  "created": "<ISO-8601 datetime>",
  "sources": ["s-<source-id>", ...],
  "producer": "<tool name + version>",
  "crystallizer_model": "<model identifier>",
  "topologizer_model": "<model identifier>"
}
```

The `corpus_id` is the BLAKE2b-128 of the canonical concatenated source content.

2.3 Locus schema

Each locus represents a single atomic factual claim:

```
{
  "id": "a-<blake2b-128-hex>",
  "breadcrumb": ["topic", "subtopic", "concept", "claim"],
  "lattice": [x, y, z],
  "statement": "<atomic factual claim as natural language>",
  "kind": "fact" | "rule" | "definition",
  "confidence": <float in [0, 1]>,
  "horizon": "short" | "long" | "archived",
  "decay": <float in [0, 1]>,
  "source": {
    "id": "s-<source-id>",
    "offset": [<byte-start>, <byte-end>]
  },
  "first_seen": "<ISO-8601 datetime>",
  "last_referenced": "<ISO-8601 datetime>",
  "references": <integer count>
}
```

Field semantics:

Field	Required	Type	Description
`id`	Yes	string	`a-` prefix + BLAKE2b-128 hex of canonical locus JSON
`breadcrumb`	Yes	array	Exactly 4 strings: topic > subtopic > concept > claim
`lattice`	Yes	array	3 integers in [0, 63] — derived from breadcrumb prefix hashes
`statement`	Yes	string	The atomic claim as a complete natural-language sentence
`kind`	Yes	enum	`fact`, `rule`, or `definition`
`confidence`	Yes	float	Crystallizer's confidence in the claim's faithfulness to source
`horizon`	Yes	enum	`short` (decaying), `long` (consolidated), or `archived`
`decay`	Yes	float	Current decay weight; reset to 1.0 on creation, multiplied per access pattern
`source.id`	Yes	string	Source document identifier (`s-` prefix)
`source.offset`	Yes	array	`[byte_start, byte_end]` in source document

Field	Required	Type	Description
`first_seen`	Yes	datetime	When the locus first entered the bundle
`last_referenced`	Yes	datetime	When the locus was last accessed by retrieval
`references`	Yes	integer	Count of retrievals; threshold-triggers horizon promotion

2.4 Lattice coordinate derivation

The `lattice` field is computed deterministically from the breadcrumb. Pseudocode:

```
def lattice_for(breadcrumb: list[str]) -> tuple[int, int, int]:
    assert len(breadcrumb) == 4
    coords = []
    for axis_idx, prefix_levels in [(0, 1), (1, 2), (2, 3)]:
        prefix = "/".join(breadcrumb[:prefix_levels]).lower()
        h = blake2b(prefix.encode("utf-8"), digest_size=16).digest()
        coord = int.from_bytes(h[:4], "big") % 64
        coords.append(coord)
    return tuple(coords)
```

This produces: - **x-axis**: derived from `breadcrumb[0]` alone (topic-level grouping) - **y-axis**: derived from `breadcrumb[0:2]` (topic + subtopic) - **z-axis**: derived from `breadcrumb[0:3]` (topic + subtopic + concept)

Topology property: loci sharing a `k`-prefix in their breadcrumb hierarchy share `k` lattice coordinates. This means semantically-related loci cluster in lattice space without an embedding model. Retrieval can use Chebyshev-distance neighborhoods in lattice space as a hierarchical pre-filter before fine-grained matching.

2.5 Horizon and decay

The `horizon` field encodes temporal validity: - `short`: short-term memory; subject to decay - `long`: consolidated; promoted from short after sufficient references - `archived`: explicitly retired; not retrieved by default but retained for provenance

Decay rule (continuous-time exponential):

```
decay(t) = decay_0 · exp(-ln(2) · (t - last_ref) / half_life)
```

where `half_life` is configurable per implementation (reference: 1 hour for short, 30 days for long).

Reinforcement (on retrieval): `decay *= 1.05`, capped at 1.0 **Negative reinforcement** (on miss): `decay *= 0.98` **Promotion** (short → long): when `references ≥ 3` (configurable threshold) **Demotion** (long → archived): when `now - last_referenced > 90 days` (configurable)

2.6 File extension and MIME

- File extension: `.ltmi`
- Recommended MIME type: `application/x-ltmi+json`
- Equivalent JSONL form MAY use `.ltmi.jsonl`

3. Design rationale

3.1 Why deterministic hash-derived coordinates

Most retrieval systems use either: 1. **Vector embeddings** — semantic similarity but requires an embedding model in the inference path 2. **Graph edges** — explicit relations but require manual or LLM-extracted construction

Hash-derived lattice coordinates offer a third option: **embedding-free deterministic positioning** that respects hierarchical structure. The position of a locus in lattice space is fully determined by its breadcrumb, with zero learned components. This has several practical consequences:

- **Reproducibility**: same input → same lattice position, always
- **No GPU dependency** at retrieval time (no embedding model needs to run)
- **Hierarchical locality**: shared breadcrumb prefixes → shared coordinates
- **Auditability**: the position can be re-derived from the data and verified
- **Composability**: lattice neighborhoods compose via Chebyshev distance without re-computing similarities

The trade-off is loss of semantic flexibility. The lattice respects the *hierarchical decomposition decided at crystallization time* but does not capture cross-hierarchy semantic similarity. For retrieval cases where this matters, lattice retrieval composes with secondary semantic search (embeddings as a finer-grained step after lattice neighborhood pre-filter).

3.2 Why four-level breadcrumb

The hierarchy depth was chosen to balance specificity vs combinatorial explosion: - **Level 1 (topic)**: domain — `Astronomy`, `Medicine`, `Cassandra T1 Model` - **Level 2 (subtopic)**: subdomain — `Solar System`, `Cardiology`, `Architecture` - **Level 3 (concept)**: specific concept — `Jupiter`, `Heart Chambers`, `Layer Configuration` - **Level 4 (claim)**: most-specific identifier — `mass and orbit`, `four chambers`, `28 layers, hidden size 2048`

Three levels was insufficient for distinguishing closely-related claims; five became unwieldy and produced very thin lattice-cell populations. Four was the empirical sweet spot in our internal benchmarks.

3.3 Why explicit horizon and decay

LLM agent systems frequently need to model the temporal validity of retrieved knowledge — facts go stale, configurations change, conversation context expires. Explicit horizon + decay gives the retrieval layer a principled way to:

- Down-weight stale claims without deleting them (important for provenance audit trails)
- Promote frequently-referenced claims to a long-term tier with longer half-life
- Archive without losing — a `archived` locus is still retrievable by explicit ID, just excluded from default retrieval

This is drawn from cognitive-science models of episodic / semantic memory consolidation. We do not claim biological plausibility; we claim the model is mechanically useful for agentic retrieval.

3.4 Why byte-offset provenance

Many retrieval-augmented systems lose track of where a retrieved claim originated, making fact-checking opaque. Byte-offset provenance lets any consumer of the bundle:

- Verify the claim is faithful to source by re-reading the source range
- Highlight source spans in UI presentations
- Detect when source content has been edited (offset no longer matches)
- Support legal / compliance use cases that require auditable source attribution

3.5 Why dual-purpose (retrieval and training)

Many retrieval formats are designed for retrieval only and then re-shaped (often lossily) for fine-tune ingestion. LTMi-XT is designed from the start so that the same `.ltmi` bundle can be:

- Loaded by a retrieval system to serve LLM queries

- Loaded by a fine-tune training pipeline as (Q, A) pairs derived from `(breadcrumb-templated question, statement)`
- Loaded by anchor-token-masked fine-tuning [@garren2026anchortokenmasking] where the breadcrumb tokens function as anchor positions

This dual-purpose design eliminates a class of ETL plumbing and keeps training data and retrieval data in lockstep.

4. Reference implementation

Reference implementation (TypeScript + Node.js) is released under Apache 2.0 at:

`github.com/Chorozion/LTMi-XT`

The reference implementation provides: - **CLI tool** for crystallization, indexing, retrieval, training-export - **Python and TypeScript SDKs** for embedding the format in larger systems - **Reference HTTP server** with retrieval endpoints - **Web UI** for inspecting `.ltmi` bundles - **Benchmarks** comparing LTMi-XT lattice retrieval against BM25 and vector RAG baselines

Quick-start:

```
npm install @sophiaxt/ltmi-xt
sophiaxt-cli crystallize --input ./docs/ --output ./bundle.ltmi
sophiaxt-cli retrieve --bundle ./bundle.ltmi --query "What is X?"
```

5. Related work

Work	Relationship
FActScore [@min2023factscore]	Atomic-fact decomposition, evaluation focus. LTMi-XT extends with indexing + temporal model.
AFEV [@afev2025]	Atomic-fact extraction with verification pipeline. LTMi-XT format-compatible.
Memo / Letta / Zep	Agent memory products. Solve overlapping problems with proprietary internal formats.
GraphRAG [@edge2024graphrag]	Entity graph extraction with hierarchical summarization. Complementary; could compose with LTMi-XT loci as graph nodes.
Standard vector RAG	Embedding-based retrieval. LTMi-XT trades semantic flexibility for reproducibility + provenance.

6. Versioning and stability

- This document specifies LTMi-XT **v0.1**.
- Backward-incompatible changes will increment the major or minor version.
- Forward-compatible additions (new optional fields) MAY occur within a minor version.
- The `manifest.v` field carries the format version; consumers SHOULD reject bundles with unknown major versions.

7. Empirical baselines

The reference implementation includes benchmark scripts comparing LTMi-XT lattice retrieval against keyword-overlap and BM25 baselines on a small set of internally-curated corpora. Honest summary:

- LTMi-XT lattice retrieval matches or modestly exceeds keyword-overlap baselines on *hierarchical-prefix queries* (queries that mention topic/subtopic terms).
- BM25 outperforms LTMi-XT on *open-ended free-text queries* without hierarchical structure.
- LTMi-XT's reproducibility and provenance properties are advantages that are not measured by retrieval-accuracy benchmarks alone.

The format's largest empirical contribution to date is as a *training data substrate* for asymmetric-masking objectives [@garren2026anchortokenmasking], where the breadcrumb-statement structure enables anchor-token masking to produce a measurable out-of-distribution generalization advantage in retrieval-grounded fine-tuning.

8. License and citation

This specification document is released under **Creative Commons Attribution 4.0 (CC BY 4.0)**. The reference implementation is released under **Apache 2.0**.

To cite the format:

```
@techreport{garren2026ltmixt,
  author      = {Garren, Thomas},
  title       = {LTMi-XT v0.1: A Retrieval Format with Hash-Derived
                 Topological Indexing},
  institution = {SOPHIA XT LLC},
  year        = {2026},
  month       = {May},
  url         = {https://sophiaxt.com/research/ltmi-xt}
}
```

References

[min2023factscore] Min, S., Krishna, K., Lyu, X., et al. (2023). FActScore: Fine-grained Atomic Evaluation of Factual Precision. *EMNLP 2023*.

[afev2025] Liu, X., et al. (2025). Atomic Fact Extraction and Verification. *arXiv:2506.07446*.

[edge2024graphrag] Edge, D., et al. (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *Microsoft Research*.

[garren2026anchortokenmasking] Garren, T. (2026). Anchor-Token Masking Produces an Out-of-Distribution Generalization Advantage in Retrieval-Grounded Masked-Diffusion Language Models. *Technical report, SOPHIA XT LLC.*

Specification v0.1 · 2026-05-09 · SOPHIA XT LLC · CC BY 4.0 (this document) / Apache 2.0 (reference implementation).