Compact Hierarchical Memory Engine (CHME): A Deterministic In-Memory Subsystem for Document-Grounded QA

Tahsin Özgür Koç

26.02.2026

Abstract

Compact Hierarchical Memory Engine (CHME) is a lightweight, deterministic, in-memory memory subsystem for document-grounded question answering. Rather than a monolithic RAG stack, CHME exposes stable APIs for ingestion, indexing, retrieval, prompt construction, LLM invocation, and snapshot persistence. The design emphasizes low operational complexity, reproducibility, and integration ergonomics for assistants, bots, and backend services. This paper specifies the system model, data structures, ingestion pipeline, retrieval and ranking logic, and persistence mechanics, and it summarizes computational characteristics and current limitations.

1 Problem Statement

Many applied assistants require a local, explainable memory substrate that can ingest heterogeneous Markdown documentation, preserve document hierarchy, and retrieve evidence deterministically without reliance on external databases. CHME addresses this by building an explicit hierarchical memory graph and a lexical inverted index in memory, then orchestrating retrieval and answer generation through a single `MemoryEngine` interface.

2 System Model

CHME operates over a corpus $\mathcal{D}$ of Markdown files and constructs:

- a set of collections $\mathcal{C} = \{C_1, \ldots, C_m\}$,
- per-collection rooted trees (document root $\rightarrow$ section $\rightarrow$ chunk),
- a global engine-level forest across collections.

Each node is represented as:

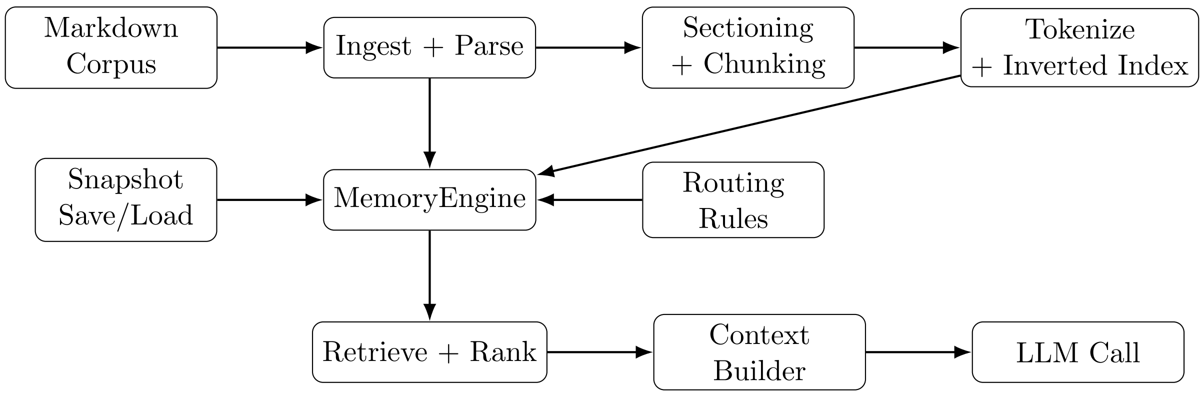
$$\text{Node} = (id, text, parent, children, depth, docId, tokens, embedding?)$$

where `embedding` is optional and unused in the baseline retrieval.

Depth semantics:

- `depth = 0`: document root
- `depth = 1`: section
- `depth = 2`: chunk (retrieval unit)

3 System Design



4 Core Data Structures

At collection scope, CHME maintains:

- `documents: Map<string, {id, path}>`
- `nodes: Map<string, Node>`
- `keywordIndex: Map<string, Set<string>>`

The hierarchy is an adjacency-list representation of rooted trees with a `parent` pointer for upward traversal and `children[]` for downward traversal. The inverted index maps token → set of chunk-node IDs, enabling efficient lexical candidate generation.

5 Ingestion Pipeline

Input is a filesystem path containing `.md` files (recursive traversal).

5.1 Parsing

Markdown headings (`#`, `##`, `###`) are parsed into hierarchical sections. Plain-text files fall back to a synthetic single section.

5.2 Chunking

Sections are split using a sliding window:

- max chunk size: 800 characters,
- overlap: 100 characters,
- preferred boundaries: sentence/newline, then whitespace fallback.

Let s_i and e_i be start/end offsets of chunk i. The next window starts at:

$$s_{i+1} = \max(e_i - 100,\ s_i + 1)$$

5.3 Tokenization

Chunk tokens are produced by lowercasing, punctuation removal, whitespace splitting, stopword filtering, and uniqueness via a set. Only chunk nodes are inserted into `keywordIndex`.

5.4 Deterministic IDs

CHME assigns stable identifiers:

- root: `{docId}:root`
- section: `{docId}:section:{sectionIndex}`
- chunk: `{docId}:{sectionIndex}:{chunkIndex}`

6 Retrieval and Ranking

Given a query q:

1. tokenize q with the same lexical normalizer,
2. gather candidate chunk IDs via the inverted index,
3. fall back to all chunk nodes if no candidates exist,

4. score each chunk by keyword overlap:

$$\text{score}(x, q) = |T_x \cap T_q|$$

5. sort by descending score, tie-breaking by lexicographic `node.id`,
6. apply section-aware round-robin selection for diversity.

This produces deterministic outputs under a fixed corpus and configuration.

7 Formal Determinism Definition

Let $\mathcal{D}$ be a fixed corpus and θ a fixed configuration (chunking parameters, normalization rules, routing rules, and ranking policy). Define the retrieval function:

$$\text{Retrieve}(q, \mathcal{D}, \theta) \to R$$

where R is an ordered list of chunk-node IDs returned for query q.

7.1 Deterministic Retrieval

Retrieve is deterministic iff:

$$\forall \text{ runs } i, j : \; R_i = R_j$$

under the same $(q, \mathcal{D}, \theta)$. In CHME, determinism is ensured by fixed tokenization, fixed candidate generation, and total ordering via tie-breaking.

7.2 Component Determinism Scores

For each component $c \in \{\text{route}, \text{retrieve}, \text{prompt}\}$, define a determinism indicator per query and iteration. The empirical determinism score is:

$$D_c = \frac{1}{|Q|(N-1)} \sum_{q \in Q} \sum_{i=2}^{N} \mathbf{1}\left[s_{q,i}^{(c)} = s_{q,1}^{(c)}\right]$$

where $s_{q,i}^{(c)}$ is the signature of component c for query q at iteration i.

7.3 Lexicographic Tie-Breaking and Total Ordering

When scores are equal, CHME orders items by lexicographic `node.id`. This induces a total order because lexicographic comparison over fixed-length strings is total and antisymmetric. Therefore, for any pair of nodes a, b, exactly one of $a < b$, $a = b$, or $a > b$ holds.

7.4 Proof Sketch: Strict Weak Ordering of Ranking

Define the ranking key as a tuple $(\text{score}(x, q), \text{node.id})$ with primary descending score and secondary ascending lexicographic ID. The comparison relation is:

$$x \prec y \iff (\text{score}(x, q) > \text{score}(y, q)) \text{ or } (\text{score}(x, q) = \text{score}(y, q) \land \text{id}(x) < \text{id}(y))$$

This relation is irreflexive and transitive. It is a strict weak ordering because equivalence classes are formed only by identical keys, and the lexicographic tie-break refines all equal-score ties into a total order. Thus, the ranking is deterministic and stable across runs.

8 Context Construction

Top-ranked chunks are rendered into prompt context blocks with section headings:

- `## <section title>` or fallback `## Section`

Global ask mode additionally annotates the source collection:

- `### Collection: <collectionId>`

A context budget L (characters) is enforced with block-boundary preservation and sentence-aware truncation if the first block exceeds L.

9 LLM Integration Layer

The `callLLM` interface supports a local provider (default) and an OpenAI-compatible chat endpoint. Operational properties include:

- deterministic intent when `temperature = 0`,
- timeout control (default 30s),
- graceful failure contract (empty string fallback on call/parse failure).

The model call is strictly downstream of retrieval and prompt construction, so core memory logic does not depend on model availability.

10 Orchestration via `MemoryEngine`

`MemoryEngine` centralizes:

- collection lifecycle and ingestion,
- global/scoped ask dispatch,
- routing rule management,
- retrieval and prompt building APIs,
- runtime LLM configuration,
- snapshot save/load.

This creates a single integration surface for external applications and minimizes direct coupling to lower-level modules.

11 Auto-Collection Routing

For large corpora, `ingestAuto` assigns files to collections deterministically:

1. explicit regex routing rules (priority aware),
2. first-folder path slug fallback,
3. default collection fallback (`general`).

Question-time routing computes per-collection lexical match scores and selects top-N collections with deterministic tie-breaks.

11.1 Routing Scoring Formalization

Let T_q be the query token set and let T_x denote the token set for chunk node x. For a collection C, define the routing score:

$$\mathrm{score}(C, q) = \sum_{x \in \mathrm{Chunks}(C)} |T_x \cap T_q|$$

Collections are ranked by descending $\mathrm{score}(C, q)$, and ties are resolved by lexicographic ordering of collection IDs. The top-N routing set is then:

$$\hat{C}_N(q) = \mathrm{TopN}_{\mathrm{lex}}(\{\mathrm{score}(C, q)\}_{C \in \mathcal{C}}, N)$$

Because the scoring function and tie-break rule are deterministic under fixed inputs, routing decisions are stable across runs.

12 Snapshot Persistence

CHME supports compressed JSON snapshots (`.chme.json.gz`) with:

- per-collection state files,
- engine manifest (`_engine.chme.json.gz`),
- schema versioning,
- source metadata (path, size, mtime),
- optional freshness-based partial re-ingest.

Load strategy is replace-existing, enabling fast startup for large corpora while preserving deterministic state restoration.

12.1 Snapshot State Equivalence

Define engine state:

$$S = (\text{Collections}, \text{Nodes}, \text{Index}, \text{RoutingRules}, \text{Config})$$

Snapshot roundtrip correctness requires:

$$\text{Load}(\text{Save}(S)) \equiv S$$

Equivalence is defined by structural equality of:

- node IDs and their associated text payloads,
- parent-child adjacency lists,
- inverted index postings for every token,
- collection manifests (document IDs and paths),
- routing rules and configuration parameters.

Under deterministic ID construction and stable serialization, this equivalence is preserved.

13 Benchmark Results

We summarize two benchmark tracks executed on a controlled FinTech SaaS corpus: a deterministic-core stability benchmark and a factual-retrieval benchmark with two model backends (local Qwen 7B and Mistral Small). All runs use 15 Markdown files across three collections (`compliance`, `payments`, `risk`).

13.1 Deterministic-Core Benchmark (2026-02-25)

Configuration: Run ID `2026-02-25T22-29-27-176Z`, 12 queries, 10 iterations (120 total executions), local model `qwen2.5:7b`, live LLM enabled.

Data integrity.

Collection	Documents	Sections	Chunks	Nodes	Keyword Terms
compliance	5	30	50	85	206
payments	5	30	50	85	215
risk	5	30	50	85	201

Determinism and routing. Route, retrieve, and prompt determinism were all perfect ($D_{\text{route}} = D_{\text{retrieve}} = D_{\text{prompt}} = 1.00$). Snapshot roundtrip consistency passed with zero mismatches ($R = 1$). Primary route strict accuracy was 11/12 = 91.67%, with the single mismatch occurring on a cross-domain query (`payments_chargeback_limit_signals` routed to `compliance`).

Latency.

Stage	Mean (ms)	P50	P95	Min	Max
route	0.05	0.04	0.08	0.03	0.21
retrieve	0.09	0.09	0.15	0.03	0.40
prompt	0.07	0.06	0.15	0.02	0.28
ask (LLM)	6024.56	5674.27	10511.43	1467.93	14315.63

Memory operations are sub-millisecond; end-to-end latency is dominated by model inference.

13.2 Factual Retrieval Benchmarks (2026-02-25)

Configuration: Run ID `2026-02-25T23-06-19-372Z` (local Qwen 7B) and `2026-02-25T23-50-48-221Z` (Mistral Small). Both use 15 files, 3 collections, 2 iterations, `topK=5`, `topCollections=3`, and `topKPerCollection=3`. Snapshot save/load enabled and passed in both runs.

Relevance metrics (both models). Route Top-1 Accuracy = 1.0000, Route Recall@N = 1.0000, Hit@K = 0.9167, MRR@K = 0.7986, Evidence Precision@K = 0.4000, Evidence Recall = 0.8194.

Factuality (answer-level grading).

Metric	Qwen 7B	Mistral Small	Δ
Mean Factual Score	0.5456	0.5943	+0.0487
Mean Keyword Coverage	0.7500	0.7500	0.0000
Mean Grounding Ratio	0.3411	0.4385	+0.0974
Empty Answers	0	0	0

Both runs fall below the 0.60 factuality threshold, indicating retrieval is strong but answer grounding remains the limiting factor. This suggests model quality materially affects answer-level grading outcomes even when routing and retrieval are stable.

Latency summary. Local Qwen 7B: ingest 14.88 ms, save 10.70 ms, load 7.26 ms, ask mean 6177.73 ms (p95 10121.29 ms). Mistral Small: ingest 19.79 ms, save 13.08 ms, load 7.95 ms, ask mean 1982.39 ms (p95 3168.77 ms).

13.3 Retrieval–Generation Gap Analysis

The benchmarks show high retrieval quality (Hit@K = 0.9167, Evidence Recall = 0.8194) alongside lower answer grounding (Grounding Ratio $\approx$ 0.34–0.43) and modest factual scores ($\approx$ 0.55–0.59). This gap reflects two distinct properties:

- **Retrieval Sufficiency:** Relevant evidence is present within the retrieved set.
- **Generative Faithfulness:** The model's answer is tightly grounded in retrieved evidence.

High recall does not imply high grounding because generation can introduce unsupported statements, paraphrase beyond retrieved content, or underutilize context. Grounding ratio is bounded by how much of the answer is directly supported by the provided context, whereas recall only indicates that evidence exists within the retrieval set. Thus, a system can be retrieval-sufficient yet generatively unfaithful.

13.4 Failure Mode Taxonomy

Observed benchmark behavior aligns with the following failure categories:

- **Cross-domain routing ambiguity:** A cross-domain query was routed to `compliance` instead of `payments`, reducing strict primary-route accuracy.
- **Overlapping lexical domains:** Shared terms across collections increase lexical collisions, leading to suboptimal chunk ranking.
- **Model hallucination under partial grounding:** Even with relevant context present, answer generation can extrapolate beyond retrieved evidence, reducing grounding ratio.
- **Context budget truncation effects:** When the context window truncates, high-value evidence may be excluded, which can degrade faithfulness without lowering recall metrics.

14 Computational Characteristics

Let N be the number of chunk nodes in a collection, V the unique vocabulary size, and k the query token count. Approximate costs:

- Index insert during ingest: $O(\text{total chunk tokens})$
- Candidate generation: $O\left(\sum_{t \in T_q} |\text{postings}(t)|\right)$
- Scoring: $O(|\text{candidates}| \cdot k)$
- Ranking: $O(|\text{candidates}| \log |\text{candidates}|)$

Space costs: nodes $O(N)$ and inverted index postings $O(P)$, where P is total token-node associations.

14.1 Computational Scaling Discussion

Posting list growth. The inverted index scales with the number of token-to-chunk associations. As documents grow, P can increase superlinearly with vocabulary diversity and chunk overlap.

Worst-case lexical queries. In the worst case, a query with high-frequency tokens (e.g., domain-wide boilerplate) can induce large postings lists, making candidate generation approach $O(N)$.

Sliding-window chunking growth. With overlap, the number of chunks scales as:

$$\text{chunks} \approx \frac{\text{doc length}}{(\text{chunk size} - \text{overlap})}$$

This increases memory linearly in document length and inversely in effective stride.

Asymptotic comparison to ANN vector search. Lexical indexing yields exact matching with deterministic ordering, while ANN-based vector search typically offers sublinear approximate retrieval at the cost of probabilistic recall and non-determinism from approximate nearest neighbor structures. Asymptotically, ANN can reduce query time from $O(N)$ to $O(\log N)$ or better, but requires additional infrastructure and tolerates approximate recall.

15 Deterministic RAG Positioning

CHME occupies a deterministic lexical memory position relative to probabilistic vector retrieval systems. Lexical memory emphasizes reproducibility and auditability: identical inputs yield identical retrieval outputs and prompt construction, enabling stable debugging and compliance traceability. In contrast, vector retrieval often trades determinism for semantic coverage and approximate search efficiency. Operationally, lexical memory minimizes infrastructure requirements (no vector databases, no embedding pipelines) but may require careful query normalization and chunking policy to handle semantic variation.

16 Limitations and Scope

The current core intentionally omits:

- embedding-based semantic retrieval,
- learned rerankers,
- external vector or database backends.

These components are additive future layers; the present lexical-hierarchical core is designed to be compact and stable.

17 Threats to Validity

- **Single-domain corpus limitation:** Benchmarks are based on a FinTech SaaS corpus, which may not reflect broader domains with different lexical distributions.
- **Limited query set:** The 12-query set constrains statistical power and may underrepresent difficult retrieval cases.
- **Model-dependent factual grading:** Answer-level factuality depends on the grading model and its alignment with evidence-grounded criteria.
- **Lexical bias risk:** Lexical overlap can overemphasize surface term matches, which may not align with true semantic relevance.

18 Conclusion

I present CHME as a disciplined memory architecture built on hierarchical node graphs, lexical indexing, deterministic routing and retrieval, and snapshot persistence. I show that it provides a reliable document memory substrate for assistants and applications requiring auditable evidence paths and low operational overhead, while remaining extensible for future semantic upgrades.