The Journal of Open Source Software

STAR: Semantic Temporal Associative Retrieval - A Local-First Graph-Based Context Engine

R. S. Balch II [1]

1 Independent Researcher, New Mexico Tech Affiliated

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 24 February 2026
Published: unpublished

Summary

STAR (Semantic Temporal Associative Retrieval) is a local-first, graph-based information retrieval system designed to enable resource-constrained devices to navigate large-scale personal knowledge corpora. Unlike traditional dense vector retrieval systems that require loading complete indices into RAM, STAR implements a sparse bipartite graph approach that retrieves only relevant "atoms" of information required for a given query.

The system uses a physics-inspired scoring model combining three factors multiplicatively: semantic co-occurrence (shared tags), temporal decay (recent memories weighted higher), and structural similarity (SimHash fingerprint proximity). This multiplicative approach ensures any zero factor eliminates irrelevant results, providing precise, explainable retrieval.

STAR has been production-validated on a 28-million-token corpus of chat history and personal documents, achieving sub-200ms query latency on 4GB RAM consumer hardware without GPU acceleration. The browser paradigm architecture—treating AI memory like web browsers treat the internet—enables universal deployment from $200 laptops to supercomputers.

Statement of Need

The Problem

Current Retrieval-Augmented Generation (RAG) systems for AI memory require high-specification servers with GPUs and substantial RAM, locking personal AI memory behind cloud subscriptions and enterprise infrastructure.

The author encountered this when accumulating 40 chat sessions (~18M tokens). When forced to start new sessions due to context limits, summaries proved insufficient—models needed full conversational history. Existing solutions required either: - Cloud dependencies (privacy concerns, recurring costs) - Local vector databases requiring 4-8GB RAM just for the index - Enterprise hardware inaccessible to individual researchers

Research Purpose

STAR addresses this gap by implementing sparse graph retrieval that: 1. **Runs on consumer hardware** (4GB RAM, CPU-only) 2. **Operates locally** (no cloud dependencies, data sovereignty) 3. **Provides explainable results** (tag paths show why each result was retrieved) 4. **Scales linearly** ($O(k \cdot \bar{d})$ complexity vs. $O(n)$ for dense vectors)

Target users include researchers managing large literature corpora, developers maintaining AI-assisted projects, privacy-conscious users, and resource-constrained environments.

State of the Field

Dense Vector RAG (HNSW, FAISS)

Systems like HNSW (Malkov & Yashunin, 2018) and FAISS (Johnson et al., 2019) represent state-of-the-art approximate nearest neighbor search. However, they require loading complete vector indices into RAM (4-8GB for modest corpora), restricting deployment to high-specification servers. Vector similarity also provides limited explainability—results match because embeddings are "close," but specific reasoning remains opaque.

Method	Time Complexity	Space Complexity	Explainability	Hardware
Dense Vector ANN (HNSW)	$O(n \log n)$ or $O(n)$	$O(n \cdot d)$	Opaque (black box)	GPU preferred
STAR (Sparse Graph)	$O(k \cdot \bar{d})$	$O(\|E\|)$	**Native (tag paths)**	**CPU-only**

Where: - n = total atoms - k = query tags (typically 5–20) - $\bar{d}$ = average tag degree (typically 10–100) - d = vector dimension (typically 768–1536) - $|E|$ = sparse edges (typically $10 \cdot n$)

For personal knowledge graphs, $k \cdot \bar{d} \ll n$, making STAR asymptotically faster than dense retrieval.

Graph-Based Memory Systems

Recent work explores graph structures as alternatives to dense vectors. T-Retriever (C. Wei et al., 2026) introduces tree-based hierarchical retrieval using semantic-structural entropy but does not incorporate temporal decay. PersonalAI (Menschikov et al., 2025) proposes a knowledge graph framework with hyper-edges for personalized LLM agents but focuses on framework design rather than production implementation.

STAR contributes a complete, deployed system with validated performance on 28M tokens of real-world data. The bipartite graph approach (Atoms $\times$ Tags) enforces strict separation between content and metadata, enabling O(1) deduplication via SimHash (Charikar, 2002) and disposable index architectures.

Personal AI Memory

Second Me (J. Wei et al., 2025) proposes LLM-based memory parameterization requiring significant computational resources. STAR achieves similar associative retrieval goals through deterministic physics-based scoring, enabling deployment on minimal hardware.

Build vs. Contribute

Existing sparse retrieval libraries (Lucene, Terrier) focus on traditional keyword search without temporal decay modeling, graph-based associative traversal, SimHash deduplication, or byte-offset lazy loading. STAR's unified field equation combining semantic, temporal, and structural factors in a multiplicative scoring model represents a novel contribution not present in existing packages.

Software Design

Architecture: The Browser Paradigm

STAR implements the "Browser Paradigm" for AI memory: just as browsers render websites by loading only necessary shards (HTML, CSS, JS) rather than the entire internet, STAR retrieves only relevant atoms required for the current query. This enables universal deployment across hardware capabilities.

Component	Browser Equivalent	Anchor Engine Implementation
HTML/CSS/JS shards	Web page components	Atoms (tags + byte offsets)
DOM tree	Document structure	Tag graph $G = (A, T, E)$
Lazy loading	On-demand resource fetch	Radial inflation from disk
Cache	Browser cache	Ephemeral PGlite index

The hybrid architecture uses: - **Node.js** as the "Browser Shell" (UI, networking, OS integration) - **C++ N-API modules** as the "Rendering Engine" (text processing, SimHash fingerprinting) - **PGlite** (PostgreSQL-compatible) for sparse graph storage - **Filesystem pointers** for content (disposable, rebuildable indices)

Data Model: Compound → Molecule → Atom

Level	Role	Content Stored	Example
Compound	Document reference	Full text (temporary)	`ChatSessions.yaml` (91.88MB)
Molecule	Semantic chunk	Chunk text + byte offsets	Bytes 1024–2048
Atom	Tag/concept	**Metadata only**	`#authentication`, `#session`

Content lives in the filesystem; the database stores only pointers (byte offsets + tags). This separation enables: - O(1) deduplication via 64-bit SimHash fingerprints - Ephemeral indices (database wiped on shutdown, rebuilt from source) - Lazy loading (content read from disk only when needed)

The Unified Field Equation

The gravity score for query q and candidate atom a is:

$$W(q,a) = |T(q) \cap T(a)| \cdot \gamma^{d(q,a)} \times e^{-\lambda \Delta t} \times \left(1 - \frac{H(h_q, h_a)}{64}\right)$$

Where: - $|T(q) \cap T(a)|$: Shared tag count (semantic co-occurrence) - $\gamma^{d(q,a)}$: Damping factor raised to hop distance (default $\gamma = 0.85$) - $e^{-\lambda \Delta t}$: Temporal decay ($\lambda = 0.0001$ s^{1}, ~115 min half-life) - $1 - H(h_q, h_a)/64$: SimHash similarity (0-63 Hamming distance normalized)

Design rationale: Multiplicative scoring ensures any zero factor eliminates noise. Additive approaches accumulate weak signals; multiplicative approaches require all factors to contribute.

Retrieval Protocol: Planets and Moons

STAR implements a three-phase retrieval protocol:

Phase 1 — Anchor Discovery (Planets)

High-precision seed set via direct matching using: - Full-text search (BM25-style) via PostgreSQL FTS - Radial inflation from atom positions - Engram cache for O(1) frequent entity lookup

Output: 20–200 anchor atoms with $d(q, a) = 0$

Phase 2 — Radial Inflation (Moons)

High-recall expansion via recursive tag-walker graph traversal:

```python
def radial_inflation(anchors, radius=1, max_per_hop=50):
    current_hop = anchors
    all_results = set(anchors)

    for hop in range(radius):
        candidates = get_connected_nodes(current_hop)
        weighted = apply_unified_field_equation(candidates, anchors)
        top_k = select_by_gravity(weighted, max_per_hop)
        all_results.update(top_k)
        current_hop = top_k

    return all_results
```

Output: 40–500 associated atoms ranked by gravity score

Phase 3 — Elastic Context Assembly

Token-budget compliance with maximal coherence: - Merge atoms within 500-byte proximity from same source - Snap to sentence boundaries for narrative flow - Progressive inflation (top 10% get 2× radius, etc.)

Result: 40–100 atoms → 8–12 coherent paragraphs

SQL-Native Implementation

The equation executes as a single recursive SQL CTE in PGlite:

```sql
WITH RECURSIVE hop_traversal AS (
  -- Anchors at hop 0
  SELECT anchor_id, 0 as hop_distance FROM anchors

  UNION ALL

  -- Recursive expansion
  SELECT t2.atom_id, ht.hop_distance + 1
  FROM hop_traversal ht
  JOIN tags t1 ON ht.atom_id = t1.atom_id
  JOIN tags t2 ON t1.tag = t2.tag
  WHERE ht.hop_distance < max_radius
)
SELECT atom_id,
  ((shared_tags / 10.0) * POWER(0.85, hop_distance)) *
  EXP(-0.0001 * time_delta) * simhash_similarity as gravity_score
FROM candidates
ORDER BY gravity_score DESC;
```

The Journal of Open Source Software

Trade-off: Recursive CTEs add query complexity but enable precise hop-distance tracking for proper damping application. The $O(k \cdot \bar{d} \cdot r)$ complexity remains tractable for personal-scale corpora.

Quality Assurance

STAR includes a comprehensive test suite to ensure correctness and reproducibility. The `tests/` directory contains unit tests for core components (atomizer, fingerprinting, graph traversal) and integration tests that verify end-to-end search behavior. A benchmarking framework (`benchmarks/`) provides reproducible performance measurements for ingestion throughput, search latency, and memory usage under varying corpus sizes. Tests can be run manually using `pnpm test`, and all benchmarks reported in this paper can be reproduced using the provided scripts, ensuring transparent validation of the performance claims.

Research Impact Statement

Production Validation

STAR has been production-validated since February 2026 on a corpus of: - **28 million tokens** (~100MB) - **151,876 atoms** (tag/concept units) - **280,000 molecules** (semantic chunks) - **436 files** (compounds)

Ingestion Performance

Dataset	Size	Molecules	Atoms	Time	Throughput
Chat Sessions (monolith)	91.88MB	214,000	776	177.8s	1,203 mol/s
GitHub Archive	2.66MB	36,793	497	22.4s	1,642 mol/s
Code Repository	0.94MB	20,916	199	25.0s	836 mol/s
Total System	~100MB	**280,000**	**1,500**	**~4 min**	**1,200 mol/s**

Search Performance

Search Type	Budget	Results	Latency (p95)	Use Case
Standard (70/30)	16k tokens	40–100 atoms	**150ms**	Daily queries
Max Recall (3-hop)	65k+ tokens	200–500 atoms	**690ms**	Research
Keyword (direct FTS)	4k tokens	20–50 atoms	**100ms**	High precision

Memory Management

Phase	RSS Memory	Notes
Peak (ingestion)	1,657MB	During 91MB file processing
Idle (post-cleanup)	510MB	After 5min idle
Reduction	**-69%**	1,147MB saved via GC

Key Achievement: Sub-200ms query latency on 4GB RAM consumer hardware without GPU acceleration.

External Use and Integrations

The system is designed as agent-harness agnostic, providing stateless context retrieval via HTTP API for integration with: - OpenCLAW framework (primary target) - Custom agent frameworks - Direct API integrations - CLI automation

Reproducibility

All benchmarks are reproducible using the included `benchmarks/` directory: - `ingestion-benchmark.ts`: Measures molecule processing rates - `search-benchmark.ts`: Measures query latency distributions - `comparison-framework.ts`: Framework for cross-system evaluation

Community Readiness

- **License:** AGPL-3.0 (open source, copyleft)
- **Version:** 4.2.0 (stable production release)
- **Documentation:** Comprehensive specs, standards (77 architecture standards), and API documentation
- **Containerization:** Docker and docker-compose support for easy deployment
- **Repository:** https://github.com/RSBalchII/anchor-engine-node

Reproducibility and Deployment

STAR includes comprehensive containerization support:

```
docker-compose up -d
curl http://localhost:3160/health
```

The single-stage Docker build based on Node.js 20 LTS includes persistent volumes, health checks, and resource limits (2 CPU, 2GB RAM) matching tested constraints, enabling researchers to reproduce benchmarks with identical environments.

AI Usage Disclosure

Generative AI tools were used in the development of this software and paper. GitHub Copilot, Gemini, Qwen Coder, Kimi AI, and Deepseek Coder assisted with code scaffolding, SQL query patterns, documentation drafts, and grammar checking.

The human author (R.S. Balch II) reviewed all AI-generated code, made all architectural decisions (browser paradigm, Unified Field Equation, three-tier data hierarchy), verified mathematical correctness, conducted all benchmarks on production hardware, and edited all documentation for technical accuracy.

AI tools did not provide: core algorithm design, mathematical derivations, research direction, benchmark methodology, or production validation. The Browser Paradigm, Unified Field Equation, Planets and Moons protocol, and ephemeral index design are original human contributions.

The author bears complete responsibility for accuracy, originality, licensing compliance, and reproducibility. All benchmarks were measured on production hardware (Omen 17 with RTX 4090, 64GB RAM, Intel i9-13980HX). No AI tools were used for peer review simulation, editor communication, or generating fake data.

Competing interests

The author declares no competing interests.

Acknowledgments

This research was conducted as independent work without external funding.

The STAR algorithm builds upon foundational work in similarity estimation (Charikar's SimHash), graph-based search (PageRank), and information retrieval (sparse vector models). The implementation uses PGlite by ElectricSQL and open-source tools from the Node.js ecosystem.

References

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, 380–388. https://doi.org/10.1145/509907.509965

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535–547. https://doi.org/10.1109/tbdata.2019.2921572

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(4), 824–836.

Menschikov, M., Evseev, D., Dochkina, V., Kostoev, R., Perepechkin, I., Anokhin, P., Burnaev, E., & Semenov, N. (2025). PersonalAI: A systematic comparison of knowledge graph storage and retrieval approaches for personalized LLM agents. *arXiv Preprint arXiv:2506.17001*.

Wei, C., Qin, H., He, S., Wang, Y., & Chen, Y. (2026). T-retriever: Tree-based hierarchical retrieval augmented generation for textual graphs. *arXiv Preprint arXiv:2601.04945*.

Wei, J., Ying, X., Gao, T., Bao, F., Tao, F., & Shang, J. (2025). AI-native memory 2.0: Second me. *arXiv Preprint arXiv:2503.08102*.