

OmniCortex Storage Architecture

Technical Guide: Why SQLite Powers Your Memories

The SQLite Decision

OmniCortex uses SQLite as its storage engine. This wasn't a default choice - it was a deliberate architectural decision based on the specific requirements of a memory system for AI assistants.

Zero configuration. Single file. Full-text search built-in.

Why SQLite Wins for Memory Storage

- **Zero Configuration** - No server to install, no ports to configure, no credentials to manage. Install OmniCortex, run setup, restart Claude Code. Done.
- **Single-File Portability** - Each project's memories live in one .db file. Copy the file to backup. Move it between machines. No export/import complexity.
- **FTS5 Full-Text Search** - SQLite's FTS5 extension provides enterprise-grade full-text search with BM25 ranking - the same algorithm used by search engines.
- **ACID Transactions** - Every memory write is atomic. Power loss mid-write? Database stays consistent. No corrupted JSON files.
- **Proven at Scale** - SQLite handles databases up to 281 TB. Your memories won't outgrow it. Used by Firefox, Chrome, iOS, Android, and millions of applications.

Database Alternatives Comparison

Database	Config Required	Full-Text Search	Best For
SQLite + FTS5	None	Built-in (BM25)	Single-user, local-first
PostgreSQL	Server + credentials	pg_trgm extension	Multi-user, cloud apps
MongoDB	Server + credentials	Atlas Search (\$)	Document stores
Pinecone/Chroma	API keys + cloud	Semantic only	Vector-only search
JSON Files	None	Manual implementation	Simple key-value

Why NOT Other Databases?

- **PostgreSQL** - Requires server installation, port configuration, user management. Overkill for single-user local memory storage. Great for web apps, wrong for CLI tools.
- **Vector Databases** - Pinecone, Chroma, Weaviate excel at semantic search but can't do keyword search natively. OmniCortex needs both - hence hybrid search with SQLite + embeddings.
- **JSON Files** - No built-in search. No transactions. Corruption risk on interrupted writes. Fine for config, wrong for growing data.

Hybrid Search Architecture

OmniCortex combines two search technologies for comprehensive memory retrieval:

FTS5 Keyword Search

- SQLite's Full-Text Search 5
- BM25 relevance ranking
- Exact phrase matching
- Fast: <10ms for 10K memories

Semantic Embeddings

- sentence-transformers model
- 384-dimension vectors
- Conceptual similarity
- Optional: pip install [semantic]

Hybrid Mode

When both are available, OmniCortex uses **hybrid search**: 40% keyword relevance + 60% semantic similarity. This catches both exact matches and conceptually related memories.

Schema Overview

Table	Purpose	Key Columns
memories	Knowledge store	id, content, context, type, importance, tags
activities	Audit trail	event_type, tool_name, success, duration_rms
sessions	Work sessions	session_id, started_at, summary
memory_relationships	Links between memories	source_id, target_id, relationship_type
memory_fts	Full-text index	FTS5 virtual table on content+context
memory_embeddings	Vector storage	memory_id, embedding (BLOB)

Storage Locations

Location	Path	Size Expectation
Project DB	.omni-cortex/cortex.db	~1KB per memory
Global Index	~/ .omni-cortex/global.db	Subset of all projects
Embeddings	Inside cortex.db	~1.5KB per memory (if enabled)

Backup & Migration

SQLite's single-file design makes backup trivial:

- **File Copy** - Simply copy the .db file. Works while OmniCortex is not running.
- **Export Tool** - Use `cortex_export(format='sqlite', output_path='/backup.db')` for a consistent snapshot even while running.
- **Cloud Sync** - The .omni-cortex folder can be synced via Dropbox, Google Drive, etc. Just avoid simultaneous writes from multiple machines.

Scaling Considerations

When SQLite is Enough (99% of Users)

SQLite comfortably handles **millions of memories** with sub-second search. A typical developer accumulates ~1,000-10,000 memories per year. You won't hit limits.

When to Consider Alternatives

- Team-wide shared memory (multiple users) → Consider PostgreSQL backend
- 100K+ memories with heavy semantic search → Consider dedicated vector DB
- Real-time sync across devices → Consider cloud-hosted solution

Performance Characteristics

Operation	1K Memories	10K Memories	100K Memories
Keyword search	<5ms	<10ms	<50ms
Semantic search	<100ms	<200ms	<500ms
Hybrid search	<150ms	<300ms	<700ms
Insert memory	<5ms	<5ms	<10ms
List memories	<10ms	<20ms	<100ms

**Benchmarks on typical hardware. Semantic search requires first-run model download (~90MB).*

Simple by design. Powerful when needed. No infrastructure required.