EngramDB

A purpose-built graph vector database for AI agent memory

The Memory Crisis in Agentic AI

- 🤖 **"What was I doing again?"** – Agents suffer from short-term memory loss when context exceeds limits
  - *"Users of Claude Code notice the AI forgets files previously discussed as sessions grow longer"*

The Memory Crisis in Agentic AI

- 🧠 **Current Memory Hacks = Inefficient Band-aids**
  - Long contexts slow down and cost a fortune (quadratic scaling: 10× context = 100× cost)
  - Memory summarization leads to critical information loss

The Memory Crisis in Agentic AI

- 💡 **Existing Memory Tools Fall Short**
  - Vector DBs: Semantic search without relationships between memories
  - Graph DBs: Relationships without vector semantics
  - No solution combines both for efficient agent memory

Sources: IBM Research on context scaling costs, Graphlit survey of memory frameworks

The Agent Memory Problem: Why It Matters

Pain Points

- **Context-Window Pressure** - Agents hit limits & degrade
- **Limited Persistence** - Restarting loses all history
- **Inefficient Token Usage** - Context stuffing wastes budget
- **No Structured Memory** - Just flat collections of text
- **Multi-Agent Isolation** - No memory sharing across agents

Real-World Examples

- Claude Code introduces `/compact` command - clear sign of context limitations
- Feature requests for "Advanced Memory Tool for Claude Code"
- LangChain & LlamaIndex offer vector stores but no graph integration
- Agents routinely forget their own reasoning and have to restart

Sources: Game-Changing Context Management in Claude Code (bgadoci.com), GitHub Issue #87 Claude Code

Why Long Contexts Aren't Enough

- **Quadratic Scaling Costs** - 2× tokens = 4× compute/memory
  - *"You're wasting computation to essentially do a Ctrl+F through irrelevant data"* - IBM Research
- **Diminishing Returns** - Beyond certain point, more context causes plateau or accuracy decline
  - *"Loss in the middle"* phenomenon where important information gets buried
- **Not Real-Time Friendly** - Ultra-long contexts introduce unacceptable latency
  - *"Not practical for real-time applications"* - Memory retrieval is faster
- **Unstructured Blob vs. Database** - No way to query specific information
  - Memory databases retrieve in milliseconds what an LLM takes seconds to process

Sources: IBM Research on larger context windows, MasteringLLM on long-context LLMs vs RAG

Introducing EngramDB

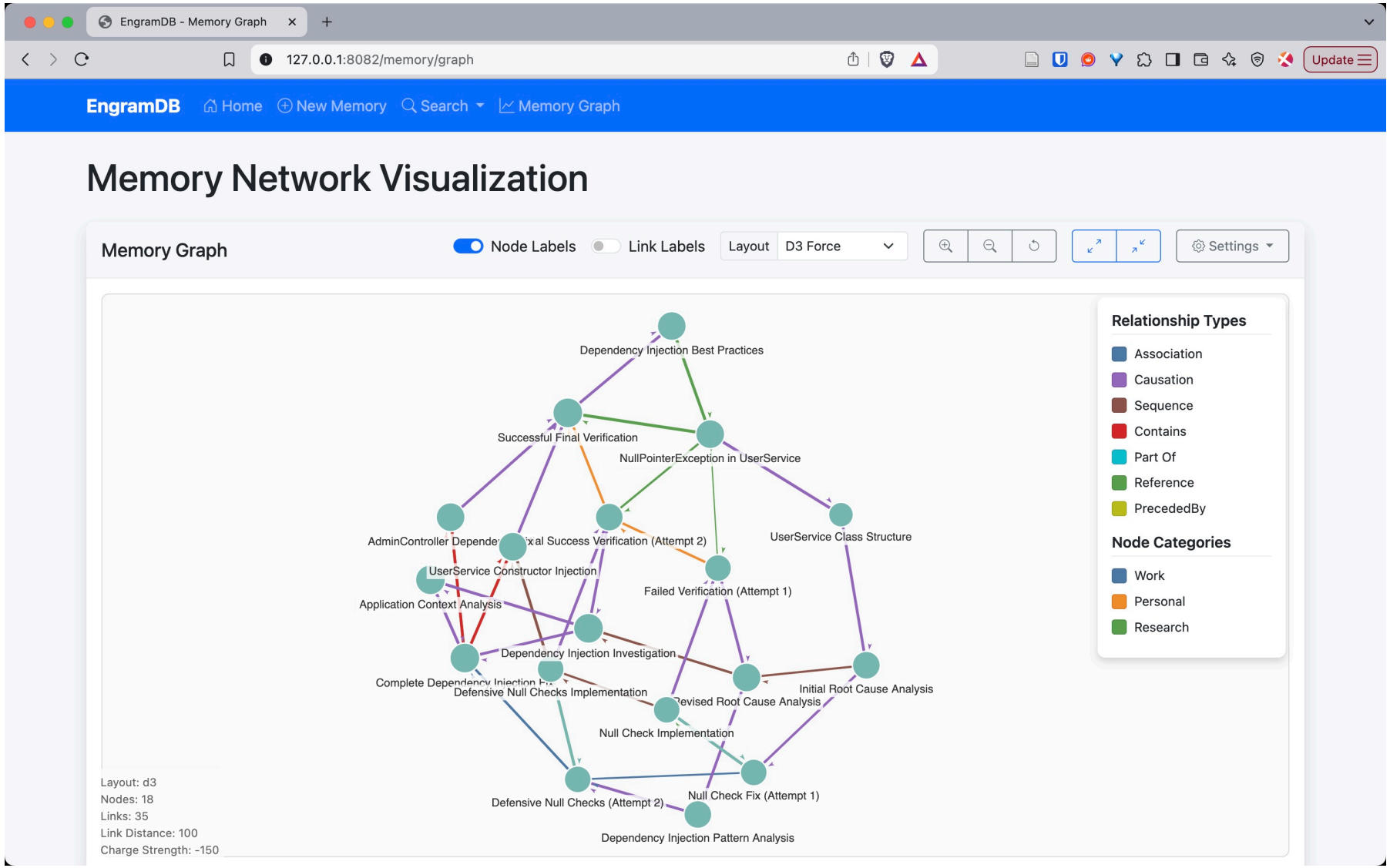
> The first purpose-built graph vector database designed specifically for AI agent memory

What Makes EngramDB Different

- **Native Graph + Vector Integration**
- **Agent-First Memory Design**
- **Lightweight & Embeddable**
- **Temporal Memory Layers**
- **Developer-Friendly API**

Why It Matters

- Rich, structured agent memories
- Efficient context window management
- Easy integration with agent frameworks
- Reduced token usage & costs
- Improved agent reliability



Core Technical Features

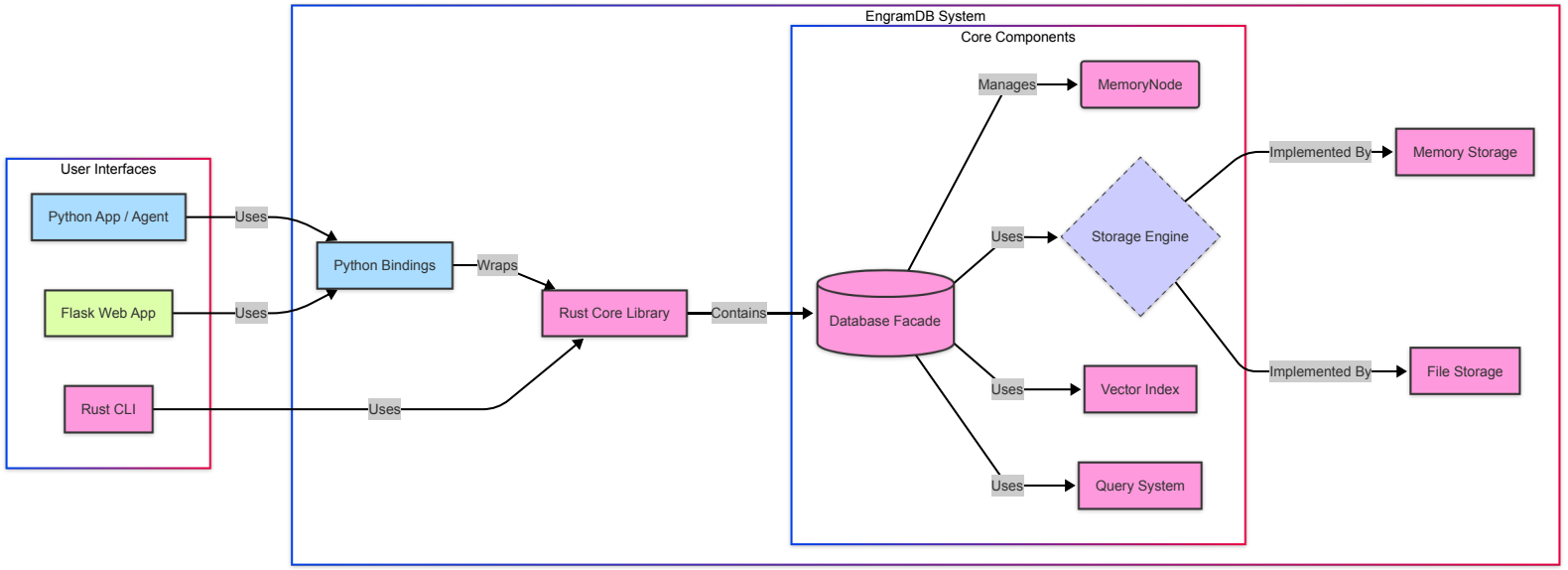
Current Capabilities

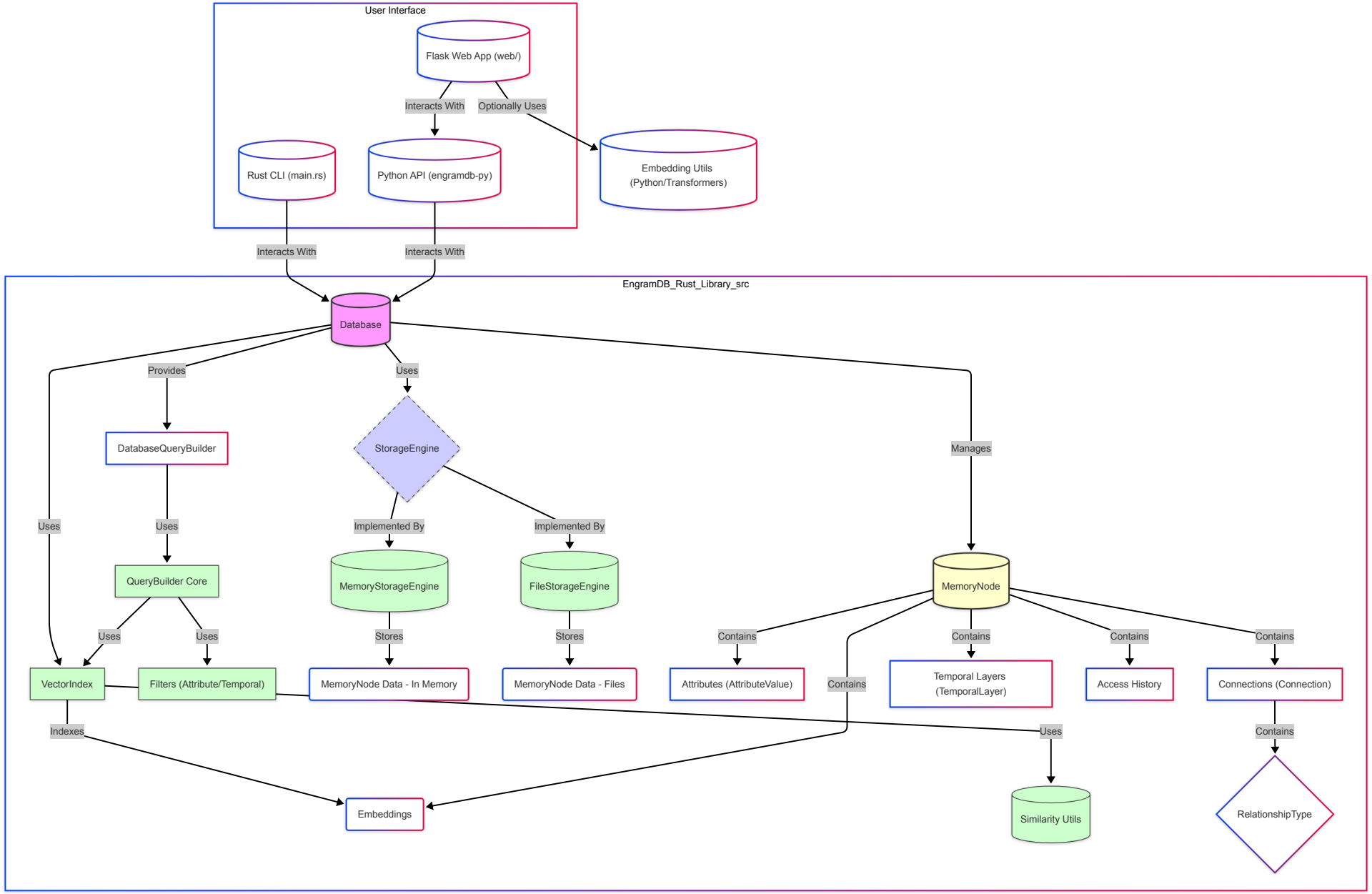
- **Dual-nature storage**: Graph relationships + Vector embeddings
- **Memory nodes** with attribute-based metadata
- **Connection-based graph** support for linked memories
- **Native vector search** for semantic retrieval
- **Python SDK** with clean, idiomatic API

Coming Soon

- Advanced vector indexing (HNSW algorithm)
- Transaction support (ACID compliance)
- Query optimization
- Connection pooling
- Multi-client concurrency
- Multi-agent memory sharing

Technical Architecture





Vector Indexing Capabilities

- **HNSW Algorithm** for fast approximate nearest neighbor search
- **Hybrid indexing** combining exact and approximate methods
- **Customizable similarity functions** for domain-specific use cases
- **Memory-efficient storage** through product quantization
- **Scalable** to handle millions of embeddings

```
// Configurable indexing
db.configure_index()
    .vector_algorithm(VectorAlgorithm::HNSW)
    .with_parameters(HNSWParams {
        m: 16,                  // Maximum connections per node
        ef_construction: 100,   // Size of dynamic candidate list
        ef: 10,                 // Size of dynamic list during search
    })
    .build()?;
```

Engineered for AI Memory

- **Engrams**: Rich data units with vectors, metadata, and connections
- **Temporal layers**: Track memory evolution over time
- **Custom query language** for complex memory retrieval
- **Connection-based reasoning** using graph relationships

```
# Example query language:
FIND MEMORIES
SIMILAR TO [0.1, 0.3, 0.5, 0.2] WITH SIMILARITY > 0.7
WHERE attribute.category = "meeting" AND attribute.importance > 0.8
CREATED WITHIN LAST 7 DAYS
CONNECTED TO "d8f7a2e5-1c3b-4a6d-9e8f-5b7a2c3d1e0f" BY "Association"
LIMIT 10
```

Agent-Friendly API

Rust Example

```rust
// Create a memory node with embedding
let memory = MemoryNode::new(vec![0.1, 0.2, 0.3]);
memory.set_attribute("type", "conversation");
memory.set_attribute("importance", 0.8);

// Save to database
db.save(&memory)?;

// Retrieve similar memories
let results = db.query()
    .similar_to(&memory.vector, 0.7)
    .with_attribute("type", "conversation")
    .limit(5)
    .execute()?;
```

Python Example

```python
# Create a memory node with embedding
memory = MemoryNode([0.1, 0.2, 0.3])
memory.set_attribute("type", "conversation")
memory.set_attribute("importance", 0.8)

# Save to database
db.save(memory)

# Retrieve similar memories
results = db.query()\
    .similar_to(memory.vector, 0.7)\
    .with_attribute("type", "conversation")\
    .limit(5)\
    .execute()
```

Why This Approach Wins

Market Validation

- Industry convergence toward **GraphRAG** (Graph + Vector)
- Pinecone's blog on "Vectors and Graphs: Better Together"
- Neo4j & Microsoft exploring graph-enhanced retrieval
- Memgraph 3.0 adding vector search to graph DB

EngramDB's Edge

- **Native integration** of both paradigms
- **Purpose-built** for agent use cases
- **Lightweight design** vs. heavyweight servers
- **Single dependency** vs. multiple integrations
- **Developer-first** experience

Sources: Pinecone blog on GraphRAG, Memgraph announcement on GraphRAG support

Implementation Roadmap

Phase 1 (Current Work)

- HNSW algorithm implementation
- Read/write concurrency
- Agent framework integrations
- Transaction support
- Hybrid vector indexing
- Query language for memory retrieval

Phase 2 & 3 (Future)

- Advanced query optimization
- Connection pooling
- Schema migration tools
- Self-contained packaging
- Cross-language clients
- Benchmarking suite
- EngramDB Cloud (managed option)

Supporting Multi-Agent Systems

The Future Is Multi-Agent

- Industry moving toward multi-agent systems
- Agents need shared memory to collaborate
- Communication across contexts requires persistence
- Graph structure enables agent-to-agent knowledge sharing

EngramDB Multi-Agent Features

- Shared memory pools with access controls
- Agent-specific memory partitions
- Cross-agent memory connections
- Memory-based communication channels
- Role and permission management

Market Size & Opportunity

Vector Database Market

- Growing from ~$1.6B (2023) to **$7+ billion by 2030**
- 24% CAGR fueled by AI applications
- EngramDB targets growing slice of this market

AI Agents Market

- Exploding from ~$5.4B (2024) to tens of billions by 2030
- All agents require memory to function effectively

Developer Adoption Signals

- Chroma: 2M+ monthly downloads, 15K+ GitHub stars
- Growing demand for agent memory solutions
- Enterprise need for memory in AI deployments
- New use cases: personal AI assistants, research agents, robotics

Business Model & Market Strategy

- **Open Core Model**
  - Core product: Apache 2.0 license
  - Enterprise features: Commercial licensing
- **Key Target Segments**
  - AI Agent Developers (early adopters)
  - AI Startups & Research Labs
  - Enterprise AI Teams
- **Revenue Streams**
  - EngramDB Cloud (future managed service)
  - Enterprise Support & Services
  - White-label & OEM partnerships

Team

- **Core Development**: Rust & database engineering experts
- **AI Integration**: Machine learning & LLM specialists
- **Operations**: Scaling & infrastructure veterans
- **Business**: Experience in developer tools & B2B SaaS

Actively growing our team with passionate engineers and AI researchers

Investment Opportunity

- Seeking pre-seed funding to accelerate development
- Capital allocation:
  - 60% Engineering team expansion
  - 30% Marketing/developer relations
  - 10% Operations/infrastructure
- Clear market opportunity with growing adoption metrics
- Strategic partnerships in development
- Positioned at intersection of two major trends: AI agents & knowledge management

Call to Action

- **Investors**: Join us in solving the memory crisis for AI agents
- **Developers**: Try our alpha release and provide feedback
- **Partners**: Explore integration opportunities
- **Team**: We're hiring founding engineers

Contact: info@engramdb.com | www.engramdb.com

Competitive Landscape

Solution	Vector Search	Graph Relations	Agent-Memory Focus	Developer Experience
EngramDB	**Yes** (built-in ANN)	**Yes** (native graph)	**Yes – designed specifically for agent memory**	Lightweight, embeddable, simple APIs
Chroma	Yes – robust vector search	No – no concept of edges/links	No – general-purpose vector DB	Very easy: pip install, runs locally
Pinecone	Yes – scalable cloud vectors	No – metadata only, no graph	No – general vector DB, requires custom memory logic	Easy API but cloud-only, no local option
Weaviate	Yes – vectors + hybrid filters	*Partial* – has cross-references but "not optimized for graph queries"	No – not built for agent memory	Moderate: requires server or cloud service
Neo4j	No native vector support	Yes – full-fledged graph DB	No – general graph use cases	Heavyweight: server, Cypher query language
SQLite + sqlite-vec	Yes – via `sqlite-vec` extension	Limited – relational only	No – DIY solution for embedding memory	Easy for small scale, SQL knowledge needed

| **Mem0** | Yes – via other DBs | No – no graph linking | Yes – memory orchestration layer | Good APIs but requires external vector store |

Sources: Weaviate docs on "not a pure graph DB", Mem0 documentation on vector store preferences

Appendix: Industry Trends

- **Convergence of Vectors and Graphs**
  - Growing demand for combined semantic + structural knowledge
  - GraphRAG emerging as powerful paradigm
- **Lightweight & Local Preference**
  - Developers prefer tools that can run locally/embedded
  - Privacy and cost advantages over cloud-only services
- **Increasing Context ≠ Decreasing Need**
  - Longer LLM contexts create need for smarter retrieval
  - Hybrid approaches (context + retrieval) becoming standard

Sources: Industry analysis of GraphRAG trends, developer surveys on local vs. cloud preferences

Appendix: Gaps in Current AI Memory Infrastructure

Current Limitations

- **No Native Memory Modules** in most agent frameworks
- **Limited Persistent Memory** across sessions
- **Context Window Pressure** as conversations grow
- **Lack of Structured Memory/Knowledge**
- **Multi-Agent Memory Sharing** challenges

Real-World Examples

- Claude Code's `/compact` command (reactive vs. proactive)
- Feature request: "Advanced Memory Tool for Claude Code"
- LangChain/LlamaIndex offer RAG but no graph memory
- Agents have to restart to clear context often
- Multiple agents can't easily share knowledge

Sources: GitHub issue #87 on Claude Code, Game-Changing Context Management blog post, Graphlit survey

Appendix: Community & Ecosystem Building

- **Growing Developer Ecosystem**
  - Sample memory recipes for popular agent frameworks (LangChain, LlamaIndex)
  - Open-source contributions and extensions
  - Integrations with AI agent platforms
- **Educational Resources**
  - Patterns for AI memory design
  - Best practices for graph-vector memory
  - Community forums and knowledge sharing
- **Metrics Dashboard**
  - Performance benchmarks
  - Memory efficiency visualizations

Appendix: New Use Cases on the Horizon

Personal AI Assistants

- Remembering user preferences and history
- Building a knowledge graph of user's life
- Maintaining consistent personality
- Recalling past interactions with context

Enterprise Applications

- AI research agents that build knowledge bases
- Customer service bots with persistent memory
- Enterprise knowledge graph construction
- Multi-agent systems for complex workflows

Modern agent architecture requires a memory layer that scales with complexity. EngramDB provides the structured, efficient memory these next-gen applications need.

Thank You!

Contact: nikola@engramdb.com