CompanyMind

AI-Powered Enterprise Knowledge Base

Semantic Search & Retrieval-Augmented Generation

MongoDB Atlas Vector Search · React 18 · Groq Llama 3 70B · FastAPI · MiniLM-L6-v2

"Stop searching for keywords. Start finding answers."

Repository	github.com/nikhilkumarpanigrahi/Companymind
License	Apache 2.0
Date	February 2025

1. Problem & Solution

The Problem

Traditional enterprise search fails at three levels:

- **Keyword mismatch** — Searching "how to speed up database queries" misses a document titled "Database Indexing Strategies" because exact words don't match
- **No semantic understanding** — keyword search can't bridge vocabulary gaps between question and answer
- **No synthesis** — even when documents are found, users must read and piece together answers manually

Organizations lose **20% of productive time** searching for information they already have (McKinsey).

Our Solution

CompanyMind converts every document into a **384-dimensional vector embedding** that captures semantic meaning. When you search, your query is embedded into the same vector space and MongoDB Atlas finds the most similar documents using **cosine similarity** — understanding intent, not matching strings.

The **RAG pipeline** goes further: it retrieves top matching documents and feeds them to **Llama 3 70B**, which generates a comprehensive, cited answer — streamed token-by-token via **Server-Sent Events**.

Mode	How It Works	Latency
Semantic Search	Query → 384-dim embedding → MongoDB `$vectorSearch` → ranked results with cosine scores	< 200ms
Ask AI (RAG)	Query → retrieve top 5 docs → assemble context → Llama 3 70B generates cited answer → SSE stream	1–3s

---

2. Architecture

```
+----------------+        +-------------------------------------+        +------------
------+
|                |  HTTP  |          Express.js API             |  query |  MongoDB
Atlas    |
|  React 18 SPA  |------->|          (Port 8080)                |------->|
$vectorSearch    |
|  Vite + TS     |        |  +----------+  +-----------------+ |        |  $text
(BM25)     |
|  Tailwind CSS  |<-------|  | 3-Layer  |  | RRF Hybrid      | |        +------------
------+
|                | JSON/  |  | Cache    |  | Search Engine   | |
+----------------+  SSE   |  +----------+  +-----------------+ |
```

```
+--------+-----------------+--------+
         |                 |
+--------v------+    +------v-----------+
| FastAPI       |    | Groq API         |
| MiniLM-L6-v2  |    | Llama 3.3 70B    |
| (Port 8000)   |    | SSE Streaming    |
+---------------+    +------------------+
```

Tech Stack

Layer	Technology	Purpose
Frontend	React 18 + TypeScript + Vite 6 + Tailwind CSS 3	SPA with dark theme, SSE streaming, voice input
Backend	Express.js 4 + Zod + Helmet + compression	REST API with validation, security, rate limiting
Database	MongoDB Atlas	`$vectorSearch` (HNSW, cosine, 384-dim) + `$text` (BM25)
Embeddings	FastAPI + SentenceTransformers `all-MiniLM-L6-v2`	384-dim L2-normalized vectors, self-hosted, zero API costs
LLM	Groq API + Llama 3.3 70B Versatile	RAG answer generation with real-time SSE streaming
Deployment	Docker multi-stage, Render, AWS ECS Fargate	Non-root containers, health checks, zero-download images

3. Core Innovation: Hybrid Search with RRF

CompanyMind's key technical contribution is its **Reciprocal Rank Fusion (RRF)** hybrid search — combining vector similarity and BM25 keyword relevance into a single, parameter-free ranking.

Why Not Simple Score Interpolation?

Linear interpolation (`α · vector_score + (1-α) · text_score`) requires careful tuning of α, score normalization, and assumes comparable distributions. **RRF uses only rank positions**, making it robust regardless of score scales.

Algorithm

```
function reciprocalRankFusion(vectorResults, textResults, k = 60) {
  const scoreMap = new Map();

  vectorResults.forEach((doc, rank) => {
    const id = doc._id.toString();
    scoreMap.set(id, (scoreMap.get(id) || 0) + 1 / (k + rank + 1));
  });

  textResults.forEach((doc, rank) => {
    const id = doc._id.toString();
    scoreMap.set(id, (scoreMap.get(id) || 0) + 1 / (k + rank + 1));
  });

  return sorted(scoreMap); // Documents in BOTH lists get boosted scores
}
```

Documents in **both** result sets receive higher combined scores. Threshold filtering (`1/(K + limit*3)`) removes noise. K=60 follows the original RRF paper.

4 Search Strategies Compared

Strategy	MongoDB Operator	What It Measures
Regex	`$regex`	Baseline — brute-force pattern match
Full-Text	`$text` + textScore	Traditional keyword search with stemming
Vector	`$vectorSearch` cosine	Pure semantic similarity (384-dim HNSW)
Hybrid RRF	Vector + Text + RRF merge	Best-of-both-worlds fusion

The built-in **benchmark suite** runs all 4 strategies on the same query, reporting per-method latency, result overlap matrix, and average relevance scores — quantitatively proving hybrid outperforms any single method.

4. RAG Pipeline — How Ask AI Works

```
  User Question: "How does vector search find relevant documents?"
         |
  1. EMBED QUERY -----------> MiniLM-L6-v2 -> 384-dim vector (~50ms, cached <1ms)
         |
  2. VECTOR SEARCH ----------> MongoDB $vectorSearch, HNSW, top 5 results (~40ms)
         |
  3. CONTEXT ASSEMBLY -------> Each source: title + relevance % + content (1500
chars)
         |                   + last 3 conversation turns (sliding window memory)
         |
  4. LLM STREAMING ----------> Groq Llama 3 70B, temp=0.3, max 1024 tokens
         |                   SSE: sources -> token x N -> done (with metadata)
         |
  5. CITED ANSWER ------------> Every claim backed by [Source: title] references
```

What Makes the RAG Unique

- **Context-grounded** — System prompt forces LLM to use *only* retrieved documents, preventing hallucination
- **Multi-turn memory** — Sliding window of last 3 Q&A pairs enables natural follow-up questions
- **Nginx-aware SSE** — `X-Accel-Buffering: no` header prevents proxy buffering; errors sent as SSE events, never dropping the connection
- **Source transparency** — Sources displayed *immediately* before the answer starts streaming

5. Triple-Layer Caching

CompanyMind implements **3 independent LRU caches** that progressively eliminate latency:

```
Request: "vector databases"
       |
[Tier 3] Search Result Cache -----> 200 entries, 5-min TTL
       | miss                        Key: "hybrid:<query>:<limit>"
       |
[Tier 2] Node Embedding Cache ----> 500 entries, 10-min TTL
       | miss                        Key: text.trim().toLowerCase()
       |
[Tier 1] Python Embedding Cache --> 2,000 entries, no TTL
       | miss                        Key: MD5(text)
       |
MiniLM-L6-v2 Inference (~50ms)
```

Tier	Location	Max Size	TTL	Cache-Hit Latency
Search results	Node.js	200	5 min	< 1ms
Embeddings	Node.js	500	10 min	< 1ms
Embeddings	Python	2,000	None	< 1ms (skips model)

On a cache-warm system, repeated queries return in **< 1ms**.

---

6. Embedding Model: Why MiniLM-L6-v2

Property	Value	Advantage
Architecture	6-layer Transformer	2x faster than 12-layer BERT
Dimensions	384	50% less storage than BERT (768), 75% less than OpenAI (1536)
Normalization	L2-normalized	Cosine similarity = dot product (MongoDB optimizes this)
Model size	~80 MB	Fits in Docker image, no GPU needed
Inference	~50ms/query (CPU)	Fast enough for real-time search
Hosting	Self-hosted (FastAPI)	Zero API costs, no rate limits, full data privacy

The embedding engine uses a **singleton pattern** — model loaded once at startup, reused for all requests, avoiding the ~2s cold-start per request.

---

7. API Overview

Core Endpoints (Express.js — Port 8080)

Method	Endpoint	Purpose
`GET`	`/health`	Server health + DB status (always 200)
`POST`	`/api/documents`	Create document (auto-embedded)
`GET`	`/api/documents`	List documents
`GET`	`/api/documents/stats`	Category/tag analytics
`GET`	`/api/search?q=...`	**Hybrid RRF search** (primary)
`POST`	`/api/search`	Pure vector search (API-to-API)
`POST`	`/api/ask`	RAG answer (non-streaming)
`POST`	`/api/ask/stream`	**RAG answer (SSE streaming)**
`GET`	`/api/ask/analytics`	Query metrics dashboard
`POST`	`/api/benchmark`	Run 4-strategy benchmark

Embedding Service (FastAPI — Port 8000)

Method	Endpoint	Purpose
`GET`	`/health`	Model status + cache hit rate
`POST`	`/embed-query`	Single text → 384-dim vector
`POST`	`/embed-batch`	Batch texts → vectors (max 512)

Validation & Security

- **Zod schemas** on every endpoint — title (1–300), content (1–10,000), query (1–1,000), question (1–2,000)
- **Rate limiting** — 100 req/min general, 30 req/min search/ask
- **Helmet** security headers, **1 MB body limit**, **CORS**, **non-root Docker**

8. Performance Metrics

Metric	Value	Details
Embedding generation	~50ms	MiniLM-L6-v2, 384 dimensions, CPU-only
Vector search	20–80ms	MongoDB Atlas `$vectorSearch` HNSW index
Hybrid search (end-to-end)	< 200ms	Including embedding + parallel search + RRF merge
RAG answer	1–3s	Groq Llama 3 70B, 1024 max tokens, SSE stream
Cache hit (any tier)	< 1ms	3-tier LRU eliminates redundant computation

Example Semantic Queries

Query	What It Finds (No Keyword Overlap!)
"speed up my app"	Caching, indexing, optimization docs
"protecting against cyber threats"	XSS/CSRF, OWASP, DevSecOps docs
"how machines learn from data"	ML, deep learning, neural network docs
"how transformers work in AI"	Attention mechanisms, BERT, transformer architecture

---

9. Design Decisions & Trade-offs

Decision	Rationale
Separate embedding microservice	ML model in Python (best ecosystem), API in Node.js (best for Express). Language-agnostic scaling.
In-memory LRU over Redis	Zero infrastructure. For finite knowledge base queries, LRU provides sub-ms lookups without ops burden.
CPU-only PyTorch	MiniLM runs in ~50ms on CPU. GPU would add ~3 GB to Docker image + require NVIDIA runtime.
RRF over learned ranking	Parameter-free, robust. Learning-to-rank needs training data — overkill for this scope.
Groq over OpenAI	Groq's Llama 3 70B in 1–3s enables real-time SSE streaming. Free tier is generous.
CommonJS (.cjs) backend	Mongoose + some deps most reliable with `require()` . ESM still inconsistent server-side.
$vectorSearch over custom HNSW	Atlas handles index maintenance, replication, scaling. No reinventing the wheel.

Resilient startup	Server listens *before* DB connects. Health checks always respond. Critical for Render/ECS cold starts.

---

10. What Makes CompanyMind Different

1. **Hybrid RRF search** — mathematically proven fusion of semantic + keyword relevance, not just one or the other
2. **Triple-layer caching** — three independent LRU caches reduce repeat query latency to < 1ms
3. **Production-ready SSE streaming** — Nginx-aware, graceful error events, never drops connection
4. **Multi-turn RAG conversations** — sliding window context for natural follow-ups
5. **Built-in benchmark suite** — quantitatively proves vector search superiority with overlap analysis
6. **Zero-download Docker** — ML model baked into image via multi-stage build
7. **Voice search** — Web Speech API for hands-free knowledge retrieval
8. **Self-hosted embeddings** — zero API costs, no rate limits, full data privacy

---

CompanyMind — Built for the MongoDB AI Hackathon

Turning documents into knowledge, and knowledge into answers.