TriBridRAG

Production-grade Retrieval-Augmented Generation combining Vector, Sparse, and Graph search

What it is

A corpus-first RAG app that runs vector (pgvector), sparse/BM25 (PostgreSQL FTS), and graph (Neo4j) retrieval in parallel. It fuses and optionally reranks results, with a FastAPI backend and React TypeScript UI.

Who it is for

Primary persona: developers/operators indexing and querying corpora via UI, API, or MCP. (Not found explicitly in repo.)

What it does

- Tri-brid retrieval: vector (pgvector), sparse (PostgreSQL FTS/BM25), graph (Neo4j traversal).
- Fusion via Reciprocal Rank Fusion (RRF) or weighted scoring.
- Optional reranking: local cross-encoder or cloud APIs (Cohere/Voyage/Jina).
- Corpus-first indexing: gitignore-aware loader, code-aware chunking, embeddings, graph builder.
- Full-stack UI + API: FastAPI backend; React+TypeScript+Zustand frontend; types generated from Pydantic.
- MCP server at /mcp with tools: search, answer, list_corpora.
- Observability, tracing, eval, and cost estimation (pricing from data/models.json).

How it works (repo-backed)

- Web UI (web/, React TS) calls FastAPI endpoints (/api/*).
- Config+API types: server/models/tribrid_config_model.py -> scripts/generate_types.py -> web/src/types/generated.ts.
- Indexing: corpus folder -> loader/chunker/embedder/graph_builder -> Postgres (pgvector + FTS) and Neo4j (entities+edges).
- Query: vector.py + sparse.py + graph.py run in parallel -> fusion.py -> (optional) rerank.py -> results/answers.
- Tracing: per-request debug info and ring-buffer trace store (server/services/traces.py).

How to run (minimal)

```
cp .env.example .env # add API keys
./start.sh # starts Postgres+Neo4j+API+UI
```

UI: http://localhost:5173/web API docs: http://localhost:8012/docs

Source: README.md + repo tree (generated locally).