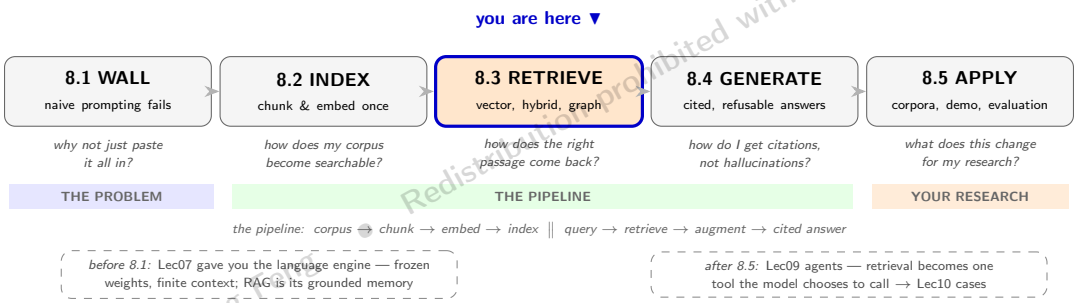
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 8.3: Retrieve — Vector Search and GraphRAG

Zhigang Feng

2026-07-10 19:44:22

Map: Act 8.3 — Bring the Right Passage Back

The index exists; this act is the online lookup — and what to do when similarity alone is not enough.



Route this deck: vector search in practice → where similarity fails → GraphRAG and the GRAM case study. Thread A: filter FOMC chunks by date and type, then query them; Thread B: the OLG literature becomes a typed graph.

Vector Search in Practice

From similarity math to a working index

Why a Vector Database?

Millions of chunks now carry 1024-dim vectors — where do they live?

- **The naive option:** a flat numpy array on disk; search by computing similarity to every vector and sorting.
  - Works fine for $10{,}000$ vectors; at $1{,}000{,}000$ a query takes seconds; at $10{,}000{,}000$ it is impractical.
  - No metadata filters. No incremental updates. No persistence.

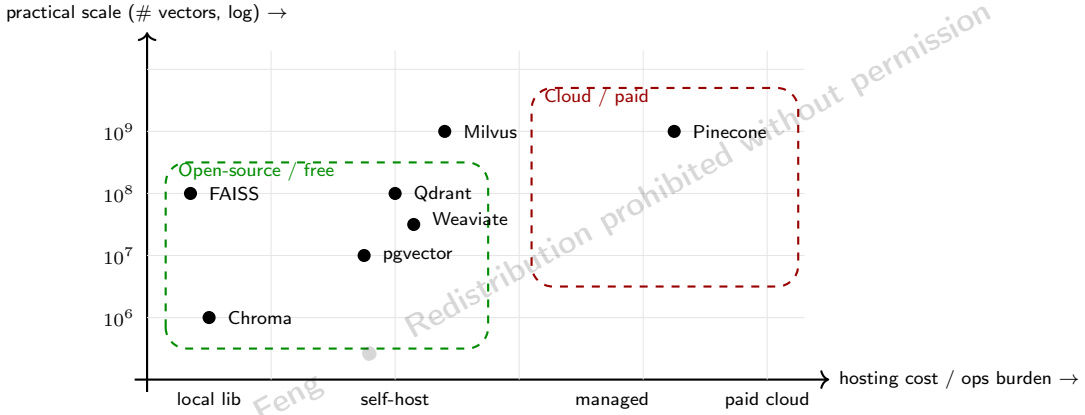
- **A vector database** solves all four:
  1. *Scale:* approximate nearest-neighbor search over millions of vectors in milliseconds.
  2. *Metadata filters:* restrict search to "date $\geq$ 2020 AND doc_type = minutes."
  3. *Incremental updates:* add new chunks as documents arrive, without rebuilding.
  4. *Persistence:* the index lives on disk and survives restarts.

Similarity Search 101 — and Why Approximate

Retrieval is nearest-neighbor search: given a query vector q, find the K closest chunk vectors d_i.

- **Three common similarity functions:** cosine $\cos(\theta) = \frac{q \cdot d_i}{\|q\| \, \|d_i\|}$; dot product $q \cdot d_i$; (negative) Euclidean distance $-\|q - d_i\|$.
- **Why cosine is the default:** it ignores vector length. Modern embedders L2-normalize their outputs, so cosine = dot product: *higher dot product* ⇒ *more similar*. Geometrically: each chunk is an arrow from the origin; cosine measures the *angle* to the query arrow.
- **Why approximate?** Exact search touches every vector: 10^7 vectors × 1024 dims ≈ 10^{10} operations *per query* — seconds of latency.
- **ANN (approximate nearest neighbor):** trade ∼1% of recall for orders-of-magnitude speedups. Names to recognize when you read the docs: **HNSW** (graph walks) and **IVF** (cluster, then search the nearest cells). In practice *your vector DB picks the default* — you won't tune this by hand.

Vector DB Landscape: Scale vs. Hosting



For an economist starting out: Chroma or **FAISS** for laptop/cluster work (free, local, simple); **pgvector** if you already run Postgres (reuses existing infra); **Pinecone / Qdrant cloud** if you need a hosted index and have budget.

Hybrid Retrieval, Plus a Light Re-Rank

Dense retrieval finds meanings, not strings — so production systems pair it with a keyword engine and a re-ranking pass.

- **The dense-only failure case:** *"find every speech mentioning Section 13(3)"* — a precise legal-citation query. Embeddings cluster similar *meanings*, not similar *strings*; the top-10 neighbors discuss the discount window in general, not the section.
- **The fix — hybrid:** also run a classical sparse retriever in parallel. **BM25** (TF-IDF-based) scores exact term matches, catching rare strings, acronyms, statute citations, tickers. Merging the two ranked lists is a solved problem (*rank fusion*) that your library implements.
- **A light re-rank on top:** the fast bi-encoder returns a top-50 shortlist; a *cross-encoder* then reads query and candidate *together* and reorders — top-5 in, much sharper relevance out. Open default: `bge-reranker-large`.

Hybrid dense+sparse (+ re-rank) is the **production default** (mid-2026). Pure dense is rarely competitive.

Metadata Filters

Research queries carry structural constraints that pure semantic search cannot express.

- **Real example:** *"Find all FOMC discussions of unemployment from 2020 onwards, excluding press conferences."*
  - "Discussions of unemployment" ⇒ semantic search.
  - "From 2020 onwards" ⇒ filter on `date` $\geq$ `2020-01-01`.
  - "Excluding press conferences" ⇒ filter on `doc_type` $\neq$ `press_conference`.

  Modern vector DBs combine similarity with SQL-style filters in a single query.
- **This is exactly why preserving rich metadata at chunking time matters** (the Act 8.2 gotcha). If chunks don't carry `date`, `doc_type`, `speaker`, you cannot filter on them later.

Rule of thumb: add *more* metadata than you think you need. It is cheap at index time and impossible to reconstruct after the fact.

Code: Build a Chroma Index and Query It

```python
import chromadb
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("BAAI/bge-large-en-v1.5")
client = chromadb.PersistentClient(path="./fomc_index")  # on disk
coll = client.get_or_create_collection("fomc")

# Add chunks with embeddings + metadata
coll.add(
    ids=[f"chunk_{i}" for i in range(len(chunks))],
    documents=chunks,
    embeddings=model.encode(chunks, normalize_embeddings=True).tolist(),
    metadatas=[{"meeting": "2024-06", "doc_type": "minutes"} for _ in chunks],
)

# Query with a metadata filter
q = model.encode(["How did the Committee discuss inflation?"],
                 normalize_embeddings=True).tolist()
results = coll.query(query_embeddings=q, n_results=5,
                     where={"doc_type": "minutes"})
```

When Similarity Is Not Enough

Relationship questions need edges, not echoes

Where Vanilla Vector RAG Starts to Break

Dense retrieval is a local similarity machine — strong on lookup, blind to structure.

- It misses three important structures:
  1. **Relationships between objects.** Two chunks may matter because their entities are *linked*, even if the wording differs.
  2. **Global structure.** Nearest-neighbor search has no notion of themes, camps, or argument clusters across an entire corpus.
  3. **Coverage.** If the same fact appears in many chunks, top-K retrieval returns duplicates instead of a representative overview.

- **Economist examples:**
  - "Which Senators most often connect inflation to housing supply constraints?"
  - "What are the main camps in recent papers on LLMs and education?"

- Once the question is **multi-hop** or **corpus-level**, plain top-K chunk retrieval is often the wrong abstraction.

Concrete Break Point: The Phillips-Curve Question

Question: *"Which Phillips-curve papers use survey expectations and are cited by post-2020 FOMC speeches?"*

Vanilla vector RAG

- Embeds the query as one vector.
- Returns 5 chunks that *sound like* "Phillips curve + survey expectations."
- Has **no notion of citation edges** between NBER papers and FOMC speeches.
- Best case: a chunk that *discusses* both topics, even if no citation link exists.
- Failure mode: confident, fluent, but the cited link is fictional.

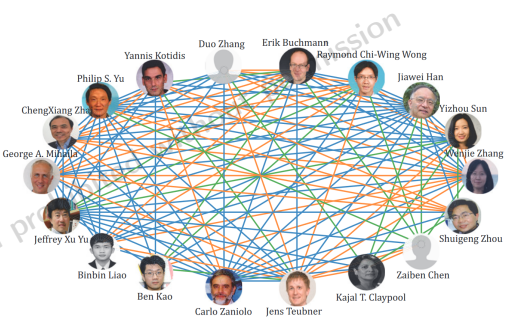
GraphRAG

- Walks the citation graph: NBER paper → `cited_by` edge → FOMC speech node.
- Filters by node attribute: `date` $\geq$ 2020.
- Filters by paper attribute: `uses_data: SurveyOfProfessionalForecasters`.
- Returns the actual subgraph; only then generates from the supporting chunks.
- Failure mode: degrades to "no path found," not invention.

Rule: reach for GraphRAG when the question is *about a relationship*, not just *about a topic*. Top-K similarity cannot manufacture an edge that the index does not represent.

Knowledge Graphs: Adding Structure to Retrieval

- A **knowledge graph (KG)** stores facts as **nodes + edges** rather than isolated chunks.
- Canonical form: `<head, relation, tail>` — e.g. `<Powell, discusses, inflation expectations>`, or across papers: `<Paper B, cites, Paper A>`.
- **Build/use:** offline, extract entities and relations (optionally summarize communities); online, retrieve the *relevant subgraph* — entity nodes, chunk nodes, community summaries.
- The gain: retrieval is no longer "find text that sounds similar" — it can *follow links*.



Once facts are stored as a graph, retrieval can follow links rather than isolated paragraphs.

Vanilla RAG vs. GraphRAG: A Direct Comparison

Match the retrieval architecture to the question type — neither dominates everywhere.

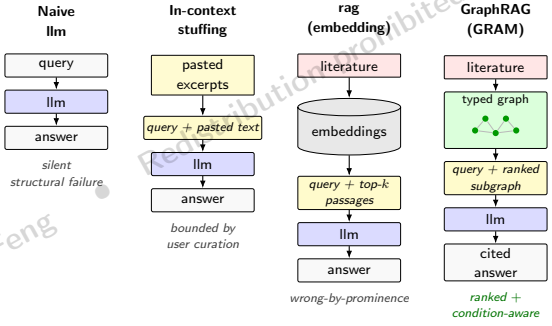
Question type	Vector / hybrid RAG	GraphRAG
Specific fact lookup	Strong when the exact chunk is nearby	Strong if fact is attached to an entity or path
Relationship query	Often requires several retrieved chunks and luck	Natural: traverse edges and retrieve neighborhoods
Global summarization	Top-K chunks under-cover the corpus	Community reports summarize modules of the corpus
High redundancy corpus	May retrieve near-duplicate boilerplate	Can collapse repeated facts into entities/communities
Cost and complexity	Simpler, cheaper, easier to debug	More expensive offline; KG quality matters

Research rule: start with vector + BM25 + re-rank. Move to graph retrieval *only* when the question requires relationships or corpus-level themes you can't get from local similarity.

Case Study: GRAM — Four Mechanisms on One OLG Query

The instructor's own GRAM: a domain-typed GraphRAG over a curated 157-paper OLG corpus — the research-grade sibling of the Act 8.5 demo.

"OLG life-cycle, idiosyncratic + aggregate shocks, incomplete intergenerational asset markets — which solver class is admissible for the joint stationary distribution in recursive equilibrium?"



Why economics is special: every paper is math and prose — math is analysis, prose is mechanism. An econ assistant needs both; GRAM sits between math-IR and legal/clinical RAG.

Wrong-by-Prominence vs. Condition-Aware Retrieval

- **Wrong-by-prominence:** pure embedding retrieval routes the query to the *most prominent* paper, whether or not it fits the asked-for conditions:
  - "OLG with capital" ⇒ Diamond (1965), *not* Auerbach–Kotlikoff (1987).
  - "heterogeneous-agent macro" ⇒ Aiyagari, *not* Krusell–Smith.

  The vocabulary is present; the applicability filter is absent.
- **Condition-aware:** admissibility is *conditional* — add aggregate risk and a different solver class becomes admissible. Embedding similarity cannot encode that dependency; a walk over a *typed* graph of model objects can.
- **Two corpus lessons worth copying:** a non-trivial *post-cutoff share* (RAG demonstrably adds value); *citation connectivity* (unlinked documents are unreachable).

Honest framing. These labels name each mechanism's *failure mode* on this query type. GRAM's condition-aware advantage is the **gap a planned three-condition evaluation (naive / RAG / GraphRAG) is designed to measure** — a design goal, not yet a reported win; one in-progress, unpublished pipeline.

Choosing the Retrieval Architecture

Question type	Best first choice	Why
Precise factual lookup or citation hunt	Hybrid dense + sparse RAG	Exact strings, dates, and citations matter more than global structure.
Specific multi-hop entity question	Graph-based RAG	The answer depends on linked entities and paths rather than one chunk.
Broad summarization over a large corpus	Graph / hierarchical retrieval	You need coverage of themes, not just the closest paragraphs.
Fast prototype on a changing corpus	Vanilla RAG first	A flat vector index is easy to build and update; add structure only if it buys accuracy.

Rule of thumb: start with hybrid dense+sparse. Escalate to graph retrieval when the task requires *relationships*, *coverage*, or *multi-hop reasoning*.

Names to recognize in this space: Microsoft GraphRAG, LightRAG, HippoRAG, RAPTOR — Edge et al. (2024); Guo et al. (2024); Gutiérrez et al. (2024); Sarthi et al. (2024).

Bridge: The Right Passage Is Back — Now Answer From It

Act 3 delivered the online lookup: hybrid vector+keyword search with metadata filters as the default, graph retrieval when the question is about relationships.

- The GRAM case shows where this is heading for research corpora: typed structure over the literature, so retrieval respects *conditions*, not just vocabulary.
- Eventually an *agent* will choose vector vs. graph vs. web search itself — Act 8.5 closes on that bridge to Lecture 9.
- **Next — Act 8.4 (GENERATE):** retrieved chunks are not an answer yet. The generation contract: a prompt template that grounds, cites, and *refuses* — plus the decision matrix against fine-tuning.

Arc: Act 1 WALL → Act 2 INDEX → **Act 3 RETRIEVE** *→ Act 4 GENERATE → Act 5 APPLY.*