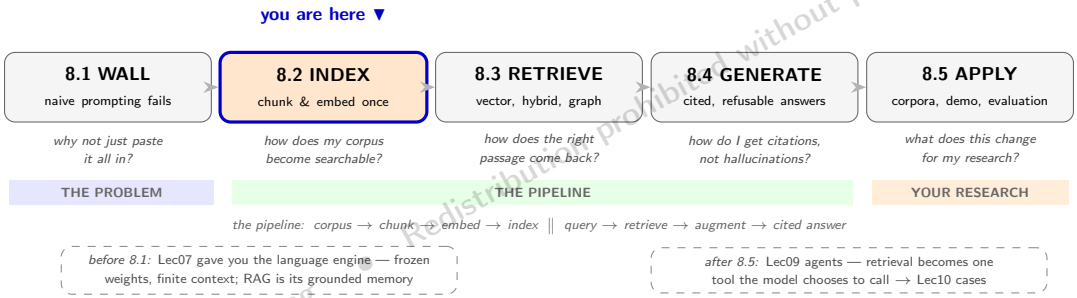
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 8.2: Index — Chunking and Embeddings

Zhigang Feng

2026-07-10 19:43:53

Map: Act 8.2 — Build the Index

Act 1 hit the wall. Act 2 builds the machine's offline half: the index.



Route this deck: one pipeline picture → chunking (what is a retrieval unit?) → embeddings (text becomes geometry). Thread A: the FOMC minutes get chunked and embedded here; Thread B: the same recipe indexes the OLG literature in Acts 8.3 and 8.5.

RAG: The Principled Solution

Index once, query many

The Key Insight: Index Once, Query Many

Don't paste the whole corpus into every prompt — pre-process it once, then fetch only what each question needs.

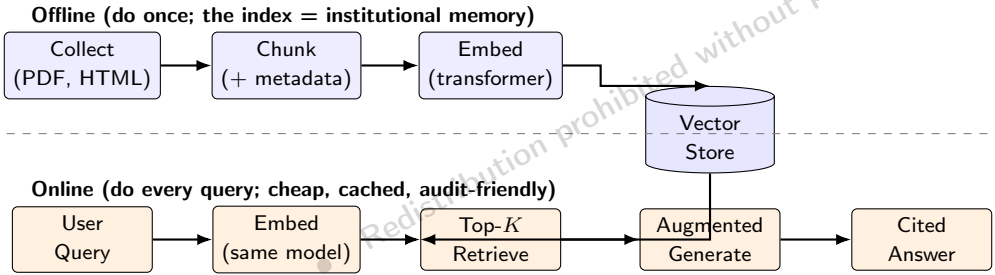
- **The recipe:**
  1. **Pre-process documents once.** Split them into chunks, convert each chunk to a numerical fingerprint (an *embedding*), store everything in a database.
  2. **At query time, fetch only what's relevant.** Convert the question to an embedding, look up the closest chunks, paste *those few chunks* into the prompt.
  3. **Let the LLM answer from the retrieved evidence**, citing which chunks it used.

- **A two-phase architecture:** *offline*, index everything once; *online*, retrieve a few relevant pieces per query and generate.

- Cost is now proportional to *what you actually need*, not to the size of the corpus. Storage is amortized; indexes are updatable; the model only ever sees relevant text, so quality and faithfulness improve.

The Whole Pipeline in One Picture

The machine for the rest of the lecture — offline index built once, online loop run thousands of times.



Five parts: **chunker** + **embedder** *(this act)* → **vector store** + **retriever** *(8.3)* → **prompt** + **generator** *(8.4)*.

One rule already: use the *same embedder* for indexing and queries — otherwise the geometries don't line up.

Definition and Origin

Retrieval Augmented Generation (RAG): before answering, retrieve relevant documents from an external store and supply them to the model as context for generation.

- **Origin:** Lewis, Perez, Piktus, et al. (2020), "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS*.
- Originally a research architecture for open-domain QA. Now the *de facto* production pattern for any LLM application over a custom knowledge base: customer support, legal research, medical Q&A, code search — and academic research.
- **Key idea (worth repeating):** the LLM is the *reasoner*. The vector store is the *memory*. The retriever is the *librarian*. RAG is how we connect them.

Chunking

What is a retrieval unit?

Why Chunk At All?

Documents are too big to embed as a single vector — the corpus must be split into retrieval units first.

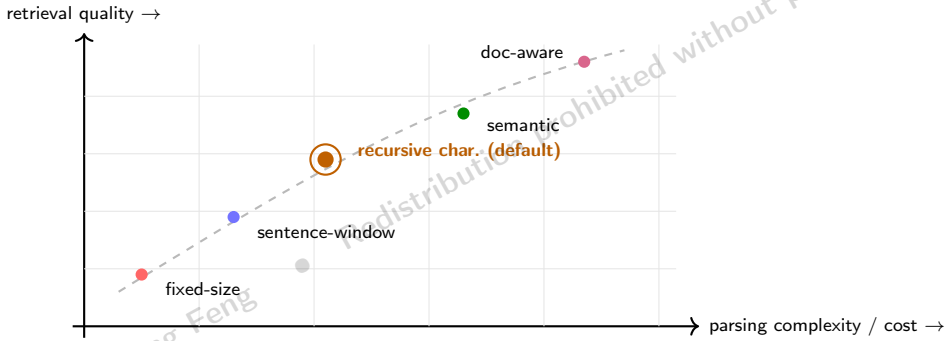
- **Three reasons to chunk:**
  1. **Embedder input limits.** Most embedding models accept 512–8192 tokens per input. A 60-page FOMC transcript blows past every limit.
  2. **Granularity of relevance.** Index a 30-page document as one vector and the embedding averages over many topics — the *one paragraph* on inflation expectations disappears into the mean.
  3. **Generator context budget.** Only a few chunks go into the prompt. Smaller chunks ⇒ more chunks fit ⇒ richer evidence base.

- **Trade-off:** chunks too large ⇒ noisy embeddings; too small ⇒ context lost.

Choosing the chunking strategy is the most consequential decision in a RAG pipeline.

Chunking Strategies: The Cost–Quality Trade-Off

One rule governs the menu: better retrieval costs more parsing effort — the five common strategies line up along that single trade-off.



Read it as a frontier: more effort → higher retrieval quality, with diminishing returns. **Rule of thumb:** start at *recursive character* (LangChain's default); climb to *document-aware* for richly structured corpora (FOMC, 10-K), and to *semantic* only when recursive retrieval is *measurably* noisy. Chunk *size* and *overlap* are separate knobs, tuned next.

Fixed-Size vs. Recursive: An FOMC Example

The same paragraph, two chunkers — one destroys meaning, one preserves it.

- **Source paragraph** (FOMC minutes, simplified):
  "Participants observed that inflation remained elevated. Several noted that core services prices, especially shelter, continued to rise. A few participants expressed concern about persistence in wage growth. Most agreed that policy should remain restrictive."

- **Fixed 60-character chunks:**
  `[■Participants observed that inflation remained elevat■]`
  `[■ed. Several noted that core services prices, especi■] ...`

  ⇒ **Cuts mid-word.** Embeddings of these fragments are nearly useless.

- **Recursive chunking** (split on paragraph → sentence → word, target ∼120 chars):
  `[■Participants observed that inflation remained elevated.■]`
  `[■Several noted that core services prices, especially shelter, continued to rise.■]`

  ⇒ Each chunk is a self-contained sentence. Embeddings carry meaning.

Practical Defaults + Code

Three defaults cover most projects: chunk size 500–1000 tokens (∼2–4 paragraphs); overlap 10–20%; **always preserve metadata** — doc ID, page, section, date, author.

```python
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=800,            # ~600 tokens
    chunk_overlap=100,         # ~80 tokens of overlap
    separators=["\n\n", "\n", ". ", " ", ""],
)

chunks = splitter.create_documents(
    texts=[fomc_minutes_text],
    metadatas=[{"meeting": "2024-06", "doc_type": "minutes"}],
)
```

The Economist's Chunking Gotcha

Real institutional documents have structure that generic chunkers destroy — and the structure is exactly what you will want to filter on later.

- **FOMC minutes:** organized as *speaker turns* ("Participants generally agreed. . . ", "A few members noted. . . "). Cutting across speaker boundaries fuses unrelated viewpoints.
- **Congressional Record:** hierarchical metadata — Congress / session / chamber / date / speaker / bill. Lose the hierarchy and you can't filter or cite properly.
- **SEC 10-K filings:** standardized item numbers (Item 1, Item 1A Risk Factors, Item 7 MD&A). A query about "risk factors" should retrieve only *Item 1A* content — which only works if you parse the structure first.
- **Practical fix:** write a corpus-specific parser that emits chunks with rich metadata. Extra effort up front; a large payoff in retrieval quality and filtering (Act 8.3 uses exactly this metadata).

Don't reach for the generic chunker. **Let your corpus tell you what a chunk should be.**

Embeddings

Text becomes geometry

From Text to Vectors: What and Why

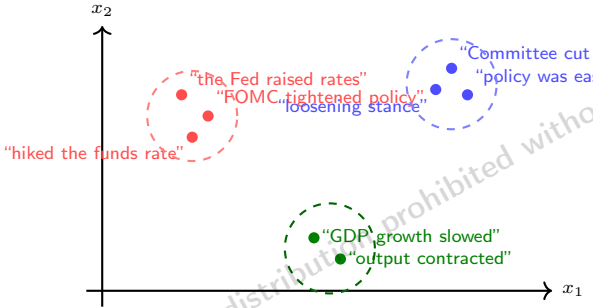
An embedding turns a piece of text into a point in space — and turns "find relevant passages" into geometry.

- An **embedding** is a function $E : \text{text} \to \mathbb{R}^d$ mapping text to a fixed-length vector; typical $d \in \{384, 768, 1024, 1536, 3072\}$.
- **The crucial property:**

$$\text{semantically similar texts} \implies \text{nearby vectors in } \mathbb{R}^d$$

- **Why this is useful:** once the corpus lives in a vector space, "find documents about inflation expectations" becomes *nearest-neighbor search* around the query's embedding.
- **Callback:** Lec. 7 T3b ("Token Embeddings vs. Document Embeddings — Bridge to Lec08") showed token embeddings living *inside* the transformer. Here a separate *retrieval-tuned* model produces one vector per *chunk*.

Geometric Intuition: Trained So Similar Texts Land Nearby



- Hawkish clusters left; dovish right; growth talk elsewhere. **Distance in vector space ≈ distance in meaning** — "find me speeches like this one" becomes nearest-neighbor search under cosine (the hook Lec. 7 T2b planted).
- **Why it works:** a retrieval embedder is a transformer encoder *pooled* to one vector per text and *trained contrastively* — similar pairs pulled together, unrelated pushed apart — so the geometry is learned for similarity. *Sentence-BERT lineage: Reimers & Gurevych (2019); mechanics in Lec. 7 T3b.*

Choosing an Embedder (mid-2026)

Four models cover a researcher's realistic day-one choices — pick by hosting constraint, not leaderboard decimals.

Model	Dim	Access	Why pick it
OpenAI `text-embedding-3-small`	1536	API (metered)	cheap default, no infra
OpenAI `text-embedding-3-large`	3072	API (metered)	top accuracy, paid
`voyage-3`	1024	API (metered)	top retrieval benchmarks
`bge-large-en-v1.5`	1024	local (free)	best open-source default

Model facts current as of mid-2026.

For an academic researcher:

- **Local + free:** `bge-large-en-v1.5` — runs on a laptop GPU; competitive with paid APIs.
- **Cloud + simple:** `text-embedding-3-small` — metered but inexpensive, zero infrastructure.
- **Top accuracy with budget:** `voyage-3` or `text-embedding-3-large`.

Economists' Embedding Example: Fed Chair Speeches

Embed every Fed Chair speech 2010–2025, project to 2D, and the policy eras separate — with no labels supplied.

- **The setup.** Embed each speech (Bernanke, Yellen, Powell) with `bge-large-en-v1.5`; project to 2D with UMAP; plot.
- **What you see:**
  - Crisis-era Bernanke (2010–2014) clusters together — QE, financial stability.
  - Normalization-era Yellen (2015–2017) forms a distinct cluster — gradual exit, forward guidance.
  - Powell splits in two: 2020–2021 dovish "patient" speeches vs. 2022–2024 hawkish "restrictive" speeches.
- **Why this matters for research:** the clustering comes *purely from the text*. The economist gains a reproducible measure of stance shifts, linkable to rate decisions, market reactions, or macro outcomes.
- *Sources in this literature: Hansen, McMahon & Prat (2018, QJE) — FOMC transcripts; Shapiro & Wilson (2021, REStat) — Fed sentiment; Gentzkow, Kelly & Taddy (2019, JEL) — text-as-data survey.*

The Domain-Adaptation Problem

Generic embedders learned English from the open web — they did not learn Fed English.

- **Where they struggle:**
  - "*patient*" in a Fed statement = *dovish* — a domain code word.
  - CUSIP codes, BEA NIPA tables, FRED series IDs — alphanumeric strings with no web-English meaning.
  - "Section 13(3)", "LSAP" — technical citations the embedder has rarely seen.

- **Two fixes:**
  1. **Fine-tune the embedder** on a domain corpus — a few GPU-hours (`sentence-transformers`, `MultipleNegativesRankingLoss`), real quality gains.
  2. **Hybrid retrieval** (Act 8.3): combine dense embeddings with classical sparse retrieval (BM25), which catches the exact-match queries the embedder misses.

- **In practice:** hybrid is cheaper and gets you 80% of the gain. Fine-tune only when retrieval quality is demonstrably your bottleneck.

Code: Embed FOMC Chunks

```python
from sentence_transformers import SentenceTransformer

# 1. Load a strong open-source embedder
model = SentenceTransformer("BAAI/bge-large-en-v1.5")

# 2. Suppose `chunks` is a list of FOMC chunk strings
chunks = [
    "Participants observed that inflation remained elevated.",
    "Several noted that core services prices continued to rise.",
    # ... thousands more
]

embeddings = model.encode(    # one numpy vector per chunk
    chunks,
    batch_size=32,
    normalize_embeddings=True,    # so cosine sim = dot product
)

print(embeddings.shape)  # (N_chunks, 1024) -- ready for the store
```

Bridge: The Corpus Is Now Geometry

Act 2 built the offline half: chunks that respect the corpus's structure, carrying metadata, each with a learned vector.

- **Three load-bearing decisions made here:** chunk strategy (recursive vs. document-aware), embedder choice (open-source vs. API), domain adaptation (hybrid first; fine-tune only if retrieval bottlenecks).
- **Hands-on, two scales:** browse `wiki_demo/` (Act 8.1's manual wiki — markdown articles + an index, no retriever); then preview `demos/M4_rag_olg_demo` — a runnable *TF-IDF* retriever over an OLG survey paper, the sparse cousin of today's embeddings. Full classroom run in Act 8.5.
- **Next — Act 8.3 (RETRIEVE):** where the vectors live and how the right passage comes back — vector databases, similarity search, hybrid dense+sparse, metadata filters, and what to do when similarity alone is not enough (GraphRAG).

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